PROJETO DE GRADUAÇÃO

ANÁLISE DE RISCOS EM PROJETOS DESCRITOS EM HARDWARE ENVOLVENDO FPGA POR MEIO DE FMEA E TESTES FUNCIONAIS

Por,
EVANDRO ALMEIDA DE OLIVEIRA
15/0034300

Brasília, 21 de julho de 2023.

UNIVERSIDADE DE BRASILIA

FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA DE PRODUÇÃO

UNIVERSIDADE DEBRASILIA
Faculdade de Tecnologia
Departamento de Engenharia de Produção

PROJETO DE GRADUAÇÃO

ANÁLISE DE RISCOS EM PROJETOS DESCRITOS EM HARDWARE ENVOLVENDO FPGA POR MEIO DE FMEA E TESTES FUNCIONAIS

Por,

EVANDRO ALMEIDA DE OLIVEIRA
15/0034300

Relatório submetido como requisito parcial para
obtenção do grau de Engenheiro de Produção

Banca Examinadora

Prof. Dra. Andrea Cristina dos Santos _____
UnB / EPR(Orientador)

Prof. Dr. Jones Yudi Mori Alves da Silva _____
UnB / EPR

Brasília, 21 de julho de 2023.

À minha esposa Gleycinaira e meus filhos Gabriel e Elisa, com amor.

Aos meus pais Expedito e Margarida

Agradecimentos

Primeiramente, gostaria de agradecer a Deus pelo amor incondicional e proteção constante ao longo dessa caminhada cuidando de tudo de forma especial.

À minha querida esposa Gleycinaira Alves e aos meus filhos Gabriel e Elisa, pela compreensão e companheirismo nos momentos de estudo e por serem minha maior alegria e inspiração nos momentos de desânimo.

A Profa Andrea por aceitar esse desafio e pela paciência nos apontamentos para a conclusão deste trabalho mesmo com uma carga de trabalho acima do normal. Ao agradecê-la eu queria alcançar todos os brilhantes Professores da UnB com quem tive o prazer e a sorte de receber porções de conhecimento importantíssimos para minha carreira.

Agradecimento especial ao meu filho Gabriel que no ano em que nasceu eu iniciava esse curso. Mesmo já estabilizado e com uma formação superior, como que uma promessa não cumprida ao meu pai que era pedreiro e queria que seu filho mais novo fosse engenheiro, acabei sacrificando o pouco do nosso tempo juntos em busca desse ideal. Me perdôe meu filho, não se comprometa a nada que roube nosso ou o seu precioso tempo em família, pois o tempo não volta jamis. Saiba que procurarei recompensá-los da melhor forma possível.

Aos meus pais, Expedito e Margarida, minha bússola moral. Agora que sou pai entendo o valor de nunca desperdiçar uma boa conversa e de ser grato pelas pequenas coisas da vida. Amo vocês.

"Máquinas podem fazer qualquer coisa
que possamos descrever de maneira
suficientemente precisa.
— Alan Turing

Resumo

Este trabalho desenvolve uma análise de riscos gerenciais e técnicos em projetos envolvendo FPGA para aplicação em Criptografia de dados por meio da Análise de Modos de Falhas e seus Efeitos - FMEA associado aos testes de softwares como elementos essenciais para garantia da qualidade e confiabilidade de soluções computacionais. A análise de riscos é um processo sistemático de identificação, avaliação e mitigação dos riscos associados a um projeto ou processo. Envolve a identificação de potenciais falhas, a análise de suas causas e consequências, e o desenvolvimento de estratégias para reduzir ou eliminar os riscos identificados. A análise de riscos é aplicada em diversas áreas, incluindo engenharia, medicina, indústria, entre outras. A FMEA é uma ferramenta específica de análise de riscos utilizada em engenharia. Ela consiste em identificar as possíveis falhas em um sistema, componente ou processo, determinar as causas dessas falhas e avaliar seus efeitos. No contexto de projetos de desenvolvimento de software é crucial que a validação e verificação ocorra de forma paralela desde a codificação num modelo estruturado denominado Modelo V onde, para cada fase do projeto, que vai da codificação até as espeficifações dos requisitos de segurança, existe a sua contraparte no lado da validação, que vai da codificação até os testes de validação. Considerando o amplo espectro que esse modelo aborda, é preciso reduzir seu alcance no sentido de permitir que as demais fases sejam complementadas em trabalhos futuros. Nesse sentido este trabalho verificará, por meio de testes modulares no MATLAB, a adequabilidade aos requisitos estabelecidos e sua consequente funcionalidade em diferentes cenários. Por meio de testes em algoritmos de alto nível. Os testes de software são realizados em diferentes fases do processo de desenvolvimento, desde testes unitários que verificam componentes individuais até testes de integração que avaliam a interação entre os diversos módulos do sistema. Também são conduzidos testes de validação para garantir que o software atenda aos requisitos do usuário final. Neste trabalho serão feitos testes unitários ou modulares e testes de integração. Em suma, a análise de riscos, a FMEA, os projetos de desenvolvimento de software e os testes de software são componentes cruciais para garantir a qualidade e confiabilidade dos produtos e sistemas. A aplicação de técnicas de análise de riscos e a realização de testes abrangentes são práticas essenciais para identificar e mitigar falhas, melhorar a eficiência do desenvolvimento de software e atender aos requisitos dos usuários finais.

Abstract

This work develops an analysis of managerial and technical risks in projects involving FPGA for application in Data Encryption through Failure Mode and Effecta Analysis - FMEA associated with software testing as essential elements to guarantee the quality and reliability of computational solutions. Risk analysis is a systematic process of identifying, assessing and mitigating the risks associated with a project or process. It involves identifying potential failures, analyzing their causes and consequences, and developing strategies to reduce or eliminate identified risks. Risk analysis is applied in several areas, including engineering, medicine, industry, among others. FMEA is a specific risk analysis tool used in engineering. It consists of identifying possible failures in a system, component or process, determining the causes of these failures and evaluating their effects. In the context of software development projects, it is crucial that validation and verification occur in parallel from coding in a structured model called Model V where, for each phase of the project, from coding to specification of security requirements, there is the its counterpart on the validation side, which runs from coding to validation tests. Considering the broad spectrum that this model addresses, it is necessary to reduce its scope in order to allow the other phases to be complemented in future works. In this sense, this work will verify, through modular tests in MATLAB, the adequacy to the established requirements and its consequent functionality in different scenarios. Through testing on high-level algorithms. Software testing is performed at different stages of the development process, from unit tests that verify individual components to integration tests that evaluate the interaction between the various system modules. Validation tests are also conducted to ensure that the software meets end-user requirements. In this work unit or modular tests and integration tests will be done. In short, risk analysis, FMEA, software development projects and software testing are crucial components to ensure the quality and reliability of products and systems. Applying risk analysis techniques and performing comprehensive testing are essential practices for identifying and mitigating failures, improving software development efficiency, and meeting end-user requirements.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

FPGA	Field Programmable Gate Array
AES	Advanced Encryption Standard
ASIC	Aplication Specific Integrated Circuit
CMOS	Complementary Metal Oxid Semiconductor
CPLD // CAD DUT	Device Under Test
EDA	Eletronic Design Automation
FAB	Força Aérea Brasileira
FIPS	Federal Information Processing standards
HPS	Hard Processor System
HSIC	High Speed Integrated Circuits
IDE	Integrated Development Environment
MATLAB	Matrix Laboratory
PLD	Programmable Logic Devices
RDS	Rádio Definido por Software
RTL	Register Transfer Level
SDF	Standards Delay Format
SoC	System on Chip
UUT	Unit Under Test
VHDL	Very HSIC Description Language
VLSI	Very Large Scale Integration

Sumário

1 INTRODUÇÃO

No processo de desenvolvimento de produtos é de salutar importância a estruturação de uma análise de riscos robusta que considere as fases do ciclo de vida de desenvolvimento do produto de forma integrada. Nas implementações de software ou soluções computacionais que incorporam alguns produtos, essa análise de riscos é vital dado que os testes de conformidade, importantes para o atendimento do propósito do produto ou aplicação, são realizados ao passo em que os códigos são desenvolvidos e as correções a serem feitas precisam ser exequíveis.

As soluções computacionais que visam resolver problemas ou melhorar de alguma forma a maneira como as operações são feitas para o bem estar dos usuários devem atender principalmente alguns requisitos básicos. Eles precisam antes de tudo cumprir o propósito para o qual ele foi criado.

A aplicação do Processo de Gerenciamento de Riscos está presente na área financeira para estudos de viabilidade de determinados investimentos e para citar apenas alguns, tem-se o risco de mercado, risco de crédito e risco da taxa de juros, dentre outros. Na área da logística existem os riscos associados ao transporte, oscilações de ressuprimento por parte dos fornecedores, crise internacionais dentre outros. É possível identificar várias áreas em que o Processo de Gerenciamento de Riscos é importante e deve ser aplicado para diminuir os impactos negativos nos objetivos estratégicos de determinado ramo.

No desenvolvimento de software não é diferente. Existe uma sequência de fases no processo de desenvolvimento de um software, ou solução computacional, em que o gerenciamento de riscos caminha lado a lado no sentido de permitir que as implementações modulares não demandem demasiadamente homem/hora na correção de erros ou *debugs* na solução final.

Como modelo escolhido tem-se o trabalho, (FLESCH *et al.*, 2020), onde se observa uma aplicação de uma norma específica, para uma aplicação própria, no mapeamento das ameaças críticas que poderiam comprometer a confiabilidade do produto e que usa FPGA em sua implementação. Esse trabalho guarda forte relação com o estudo de caso, (OLIVEIRA, 2023), onde uma implementação em software de um algoritmo de criptografia por meio de FPGA, que também possui inúmeras questões críticas, é checada, fase a fase,

quanto à confiabilidade à norma e atendimento aos seus requisitos críticos .

Por confiabilidade entende-se como a qualidade que determinado produto tem no tempo, (JORGENSEN, 2018) e por requisitos críticos a capacidade de atender imperiosamente às necessidades dos clientes, aqui é colocado como a capacidade de atender o propósito para o qual foi criado. Isso em soluções de software faz mais sentido apenas é dito de outra forma.

1.1 Definição do Problema

Numa implementação de software que incorpora determinado produto ou aplicação a testabilidade acompanha todo o ciclo de desenvolvimento e por vezes algumas correções precisam ser feitas para atender ao objetivo final. Quando se trata de software embarcado tal como o desenvolvido ao longo dessa pesquisa, é preciso dividir todo o código em módulos, os mais simples possível, e que possa ser testado para se prosseguir ao módulo seguinte.

No entanto convém destacar um problema sério no desenvolvimento de soluções utilizando essa tecnologia, que será melhor detalhada no tópico de Referencial Teórico, que é a capacidade de ser sintetizável. Existem restrições que precisam ser obedecidas para que determinada idéia, a grosso modo, possa ser sintetizada num chip e que execute as funções programadas. Sendo assim essa passa a ser a principal preocupação ou restrição com regra maior que será observada ao longo do teste e desenvolvimento, qual seja, o risco de determinada abordagem não ser sintetizável na placa objeto do estudo.

1.2 Objetivos

1.2.1 Objetivo Geral

Identificar e qualificar os riscos inerentes ao desenvolvimento da solução computacional por meio de testes de funcionalidade e integração de módulos.

1.2.2 Objetivos específicos

Descrever com base na norma que publicou o algoritmo criptográfico escolhido as funções que o código precisa executar e elencar possíveis riscos; e

Elaborar e executar testes de conformidade para os módulos do algoritmo. Esses testes visam sempre a capacidade principal de realizar a função para o qual ele foi programado,

funcionalidade, e de ser integrado mantendo o fluxo dos dados com base em uma sequência definida.

1.3 Limitações da Pesquisa

1.4 Motivação

FPGA (Field Programmable Gate Array) é um tipo de circuito integrado que pode ser programado e reconfigurado pelo usuário para realizar diferentes funções lógicas. Os FPGAs são amplamente utilizados em diversas aplicações, como processamento de vídeo e imagem, computação de alto desempenho, prototipagem de circuitos, sistemas embarcados, entre outras.

A implementação em FPGA oferece algumas vantagens em relação aos seus principais concorrentes, os circuitos ASIC (Application Specific Integrated Circuits), que são projetados para uma aplicação específica e não podem ser alterados após a fabricação. Algumas dessas vantagens são:

- Maior flexibilidade e adaptabilidade: os FPGAs permitem modificar o design do circuito de acordo com as necessidades do usuário ou do ambiente, sem a necessidade de produzir um novo chip.
- Menor tempo de desenvolvimento e custo: os FPGAs podem ser programados e testados rapidamente, sem a necessidade de passar por etapas complexas de fabricação e validação que envolvem os ASICs.

No entanto, a implementação em FPGA também envolve alguns riscos que devem ser considerados e mitigados pelos desenvolvedores. Alguns desses riscos são:

- Maior complexidade e dificuldade de projeto: os FPGAs exigem um alto nível de conhecimento técnico e ferramentas especializadas para realizar o projeto, a síntese, a simulação e a depuração do circuito. Além disso, os FPGAs possuem limitações de recursos, como número de portas lógicas, memória, pinos de entrada e saída, que devem ser respeitadas e otimizadas.
- Maior complexidade e dificuldade de projeto: os FPGAs exigem um alto nível de conhecimento técnico e ferramentas especializadas para realizar o projeto, a síntese, a simulação e a depuração do circuito. Além disso, os FPGAs possuem limitações de recursos, como número de portas lógicas, memória, pinos de entrada e saída, que devem ser respeitadas e otimizadas.

- Portanto, a análise de riscos envolvendo implementação em FPGA para soluções computacionais é uma etapa fundamental para garantir a qualidade, a segurança e a eficácia do projeto. Os desenvolvedores devem identificar, avaliar e controlar os riscos potenciais, bem como planejar e executar medidas preventivas e corretivas para evitar ou minimizar os impactos negativos dos riscos.

Considerando isto dito acima, a principal motivação deste trabalho é, entender a complexidade envolvida no uso dessa tecnologia com tantas vantagens de desempenho, tempo de desenvolvimento e custo, equilibrar por meio de uma análise de riscos as desvantagens que se apresentam de uma forma integrada e otimizada. Permitindo, ao passo do desenvolvimento, correções que produzam o mesmo efeito mitigando ou excluindo os riscos críticos.

Esse trabalho dá usa como estudo de caso um trabalho de mestrado em que uma proposta de implementação em software de uma criptografia de chaves simétrica, conhecida mundialmente por sua robustez e segurança, e apresentada por meio da tecnologia FPGA onde a análise de riscos não fora evidenciada.

2 REVISÃO BIBLIOGRÁFICA E REFERENCIAL TEÓRICO

2.1 Revisão Bibliográfica

Este tópico relaciona de forma resumida as principais obras e trabalhos utilizados como referência neste trabalho e que tiveram contribuições tanto teórica quanto em termos práticos. Muitos outros meios foram utilizados que vão de vídeos aulas até pesquisas de fóruns de discussão em assuntos específicos, no entanto aqui estão as principais obras publicadas utilizadas e necessários para a construção de conhecimento.

Tese de Doutorado da Dra Viviane - Aqui são lançadas as bases do gerenciamento de riscos gerenciais e técnicos no desenvolvimento de projetos de produtos de forma orientadora e esclarecedora. Na medida em que a metodologia é exposta, outras referências no assunto de gerenciamento de riscos são lançadas, as quais também foram consultadas conforme item abaixo.

Gerenciamento de Riscos - Essa obra além de lançar os principais conceitos do tema gerenciamento de riscos aponta que os riscos precisam ser priorizados tendo em vista que recursos, pessoas e tempo não são suficientes para lidar com todos os riscos e assim é preciso mensurar e priorizar adequadamente o impacto x probabilidade como uma boa estratégia empresarial. Apesar de ter uma abordagem voltada para governança empresarial e estratégias, serviu para entender a importância do assunto e vislumbrar o que poderia ser aproveitado no gerenciamento de riscos no desenvolvimento de um software.

Introdução à Engenharia de Produção - Traz o conceito da (Failure Mode and Effects Analysis - FMEA) Análise dos modos de falhas e seus efeitos, a forma como se constrói o formulário e sua fórmula. Essa referência é importante para se adaptar ao que se pretende mensurar com o desenvolvimento do software proposto.

Proposal of a Functional Safety Methodology Applied to Fault Tolerance in FPGA Applications - Este artigo traz uma uma proposta de metodologia para aplicações tolerantes à falhas em projetos envolvendo FPGA. Coloca a norma, IEC 61508, que estabelece

níveis de segurança de integridade para aplicações críticas como equipamentos espaciais, comunicações militares dentre outras.

Component Failure Analysis (CFA): A New Method for Implemented FPGA Design Failure Analysis - Este artigo traz um método probabilístico de analisar falhas de componentes que usa técnicas específicas de FPGA e algoritmos para identificar sinalizações de eventos únicos. Esses eventos são críticos e precisam ser identificados pois suas operações também são críticas. Traz uma contribuição no sentido em que coloca uma norma como balizadora dos parâmetros a serem observados, o que facilita sobremaneira o trabalho do desenvolvimento numa aplicação de algum método da qualidade por exemplo.

Engenharia de Software: Qualidade e Produtividade com Tecnologia - Este livro do Kechi Hirama incorpora ao conhecimento as nuances relacionadas ao desenvolvimento de software e como a qualidade deve suportar a construção do software de maneira modular e que ela, TQM - Total Quality Management, deve aplicar suas técnicas, métodos e princípios para o sucesso organizacional a longo prazo. Destaca que sucesso como projeto entregue dentro das expectativas do cliente em relação a custo, prazo e atendimento. Essa abordagem que é confirmada ao longo da estrutura do livro guarda grande relação com o que está sendo proposto nesse trabalho considerando que a escolha pela Tecnologia FPGA, mesmo que não implementada numa placa real, já traz em sua natureza, a flexibilidade e redução de custos quando comparadas aos softwares construídos para processadores, tem a facilidade de conhecimento bem disseminado e ao alcance de todos além de se ter ferramentas de confirmação ou testes de funcionalidade. Restaria portanto a aplicação da base da Qualidade Total na eleição de que parâmetros deveriam ser avaliados como críticos ou obrigatórios para terem uma pontuação maior.

Software Testing: A craftsmans Approach - Em sua quarta edição essa obra lança as bases do que é preciso saber para que se implemente testes de softwares de acordo com o que existe na literatura atual. Conceitua os tipos de testes sobre a perspectiva do testador e do objeto de teste, apresenta as principais definições sobre o tema e estabelece o ciclo de teste de softwares desde a especificação até a solução de falhas num fluxograma sequencial.

A Practitioner's Guide to Software Test Design - De Lee Copeland e juntamente com a obra acima, foram as bases para entendimento e utilização de como um teste de software pode ser feito e sua importância para o encadeamento dos módulos e depois integração. Coloca os conceitos de forma mais técnica de Black Box e White Box com relação à necessidade ou não de se conhecer o que está sendo feito pelo código ou somente as entradas e saídas.

Matlab Essential for Engineers and Scientists - De Brian Hahn e Daniel T. Valentine, essa foi a referência mais completa de MATLAB encontrado e que melhor explica os de-

talhes dessa linguagem. Como o trabalho foi feito nas fases anteriores à descrição apenas para representar a extenuante e complexa matemática da criptografia em linguagem de programação, esse conhecimento e uso das facilidades que o MATLAB possui no tratamento de matrizes foi importantíssimo. Isso passa a entendimento de como os números são tratados, que é diferente de como são mostrados, como eles são referenciados e como são visualizados e principalmente as funções nativas do MATLAB como deslocamento para esquerda e direita onde no VHDL seria feita por algumas dezenas de linhas de código, um único comando implementar essas funções muito embora se saiba que internamente são centenas de linhas da mesma forma sendo que isso não é preocupação do usuário.

2.2 Fundamentação Teórica

2.2.1 Field Programmable Gate Array

Os FPGAs são componentes programáveis da classe de Elementos Lógicos Programáveis - PLD os quais, por meio de conexão de milhares de portas lógicas dispostas em uma matriz, implementam as funções mais básicas da eletrônica digital. A partir dessas conexões é possível realizar somas, subtrações até circuitos mais complexos como multiplexadores, codificadores, dentre outros. Os FPGA possuem várias vantagens em relação a seu principal concorrente, os Standard Logic, e aqui cabe destacar que as tecnologias que executam circuitos digitais é bem vasta, vide 2.1, onde se percebe as principais famílias das tecnologias empregadas para implementação em hardware, (TOCCI *et al.*, 2018).

Dentre as principais vantagens estão o custo e a flexibilidade de programação. No entanto, outras vantagens a depender das restrições do projeto como consumo de energia e tempo de desenvolvimento também podem ser citadas.

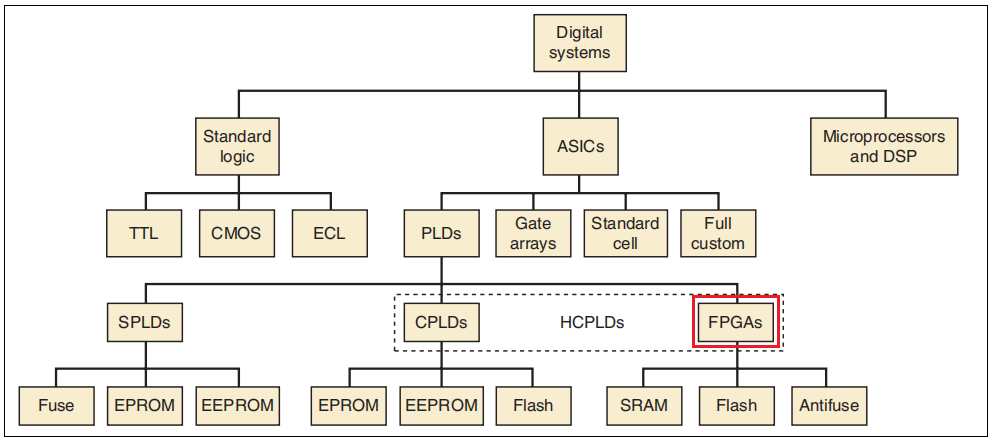

FIGURA 2.1 – Sistemas Digitais por famílias - adaptado (TOCCI *et al.*, 2018)

Para efeito deste trabalho, como não tem o escopo de demonstração real tal como no

estudo de caso escolhido, será considerada apenas a forma como a tecnologia é descrita ou programada. Existe uma diferença entre programar e descrever no campo da eletrônica digital. Enquanto a programação diz respeito apenas ao uso de uma linguagem de programação, interpretada ou não, para a execução de um código por meio de um processador, a descrição diz respeito ao arranjo de portas lógicas que executam determinadas função booleanas onde o seu encadeamento executa uma função. As linguagens de descrição são de mais difícil aprendizado devido o seu baixo nível de abstração pois descreve sistemas complexos nos mínimos detalhes.

Ainda sobre a tecnologia FPGA cabe o destaque que se trata de uma inovação que permite que seja reconfigurável pelo usuário, e daí a sigla "Field Programmable Gate Array", mais flexíveis em termos de finalidade, maior performance, maior custo benefício se considerar uma placa dos tipos didáticas puramente composta por FPGA e o tempo de desenvolvimento menor comparados aos processadores Standard Logic (TTL ou CMOS);

Como toda tecnologia utlizada para processamento de dados, seja ela ASICs ou FPGA, alguns testes quanto a segurança precisam ser feitos e esses testes visam, dentre vários outros, avaliar imunidade a interferências ocasionadas por radiações, flutuações de Energia, etc, (Flesch, 2020).

A figura 2.2 abaixo ilustra a matriz que configura um FPGA onde se percebe que o que está de posse do desenvolvedor que vai descrever esse hardware por meio de uma linguagem de descrição de hardware, seja ela VHDL ou Verilog, são as entradas I/O que ativam os módulos elementares, CLB, para o resultado que se quer. Isto, é claro é feito pela ferramenta que compila o código escrito, (INTEL, 2023).

2.2.2 Linguagem de Descrição VHDL

O entendimento do conceito por trás da Linguagem de Descrição VHDL, do inglês - Very High Speed Integrated Circuits Description Language - passa obrigatoriamente por uma comparação com a linguagem de programação tal como a maioria das pessoas estão acostumadas a ouvir e lidar.

A principal diferença entre uma linguagem de programação e a descrição de circuitos lógicos por meio de VHDL é que uma linguagem de programação é usada para criar programas que são executados por um processador, enquanto que VHDL é usada para criar circuitos eletrônicos que são gravados em um circuito integrado. VHDL é uma linguagem específica para o design de circuitos digitais em CPLDs, FPGAs e ASICs. Uma linguagem de programação tem uma sintaxe e uma semântica definidas, enquanto que VHDL tem uma sintaxe baseada na linguagem ADA e uma semântica baseada em modelos de simulação.

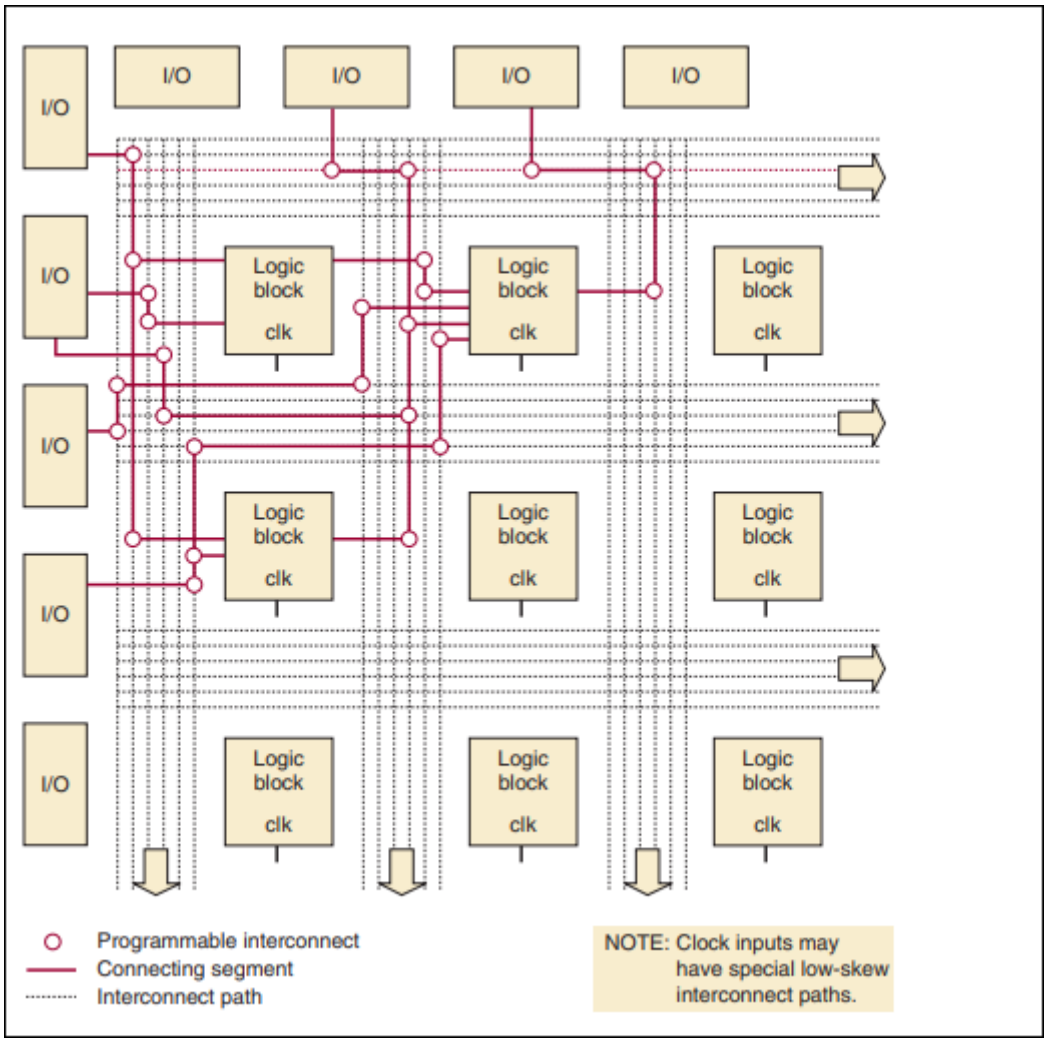

FIGURA 2.2 – Representação de um circuito FPGA - (TOCCI *et al.*, 2018)

A linguagem ADA é uma linguagem de programação de alto nível, desenvolvida originalmente para aplicações de sistemas embarcados e de tempo real. Ela foi projetada com foco em segurança, confiabilidade e ”manutenabilidade”. Ela é especialmente adequada para aplicações críticas, como sistemas aeroespaciais, militares, médicos e industriais, onde erros de software podem ter consequências graves. Esta linguagem, que é a base da linguagem HDL, VHDL ou Verilog, foi nomeada em homenagem a Ada Lovelace, considerada a primeira programadora da história, (D´AMORE, 2012).

A capacidade de simulação foi um achado incrível em termos de uso para as verificações que os designs precisavam fazer em projetos muitos extensos quanto à funcionalidade, (LAMERES, 2019). Por meio dela também é possível fazer a síntese que é a fase em que é possível implementar uma descrição funcional, textos padronizados de acordo com a sintaxe, num circuito pronto para ser criado num hardware real no nível de portas lógicas.

A Figura 5.1 abaixo ilustra as fases que precisam ser obedecidas para que um projeto utilizando HDLs tome forma. As ferramentas que performam cada fase podem variar a depender do fabricante da placa FPGA, mas nada foge dessa sequência de projeto.

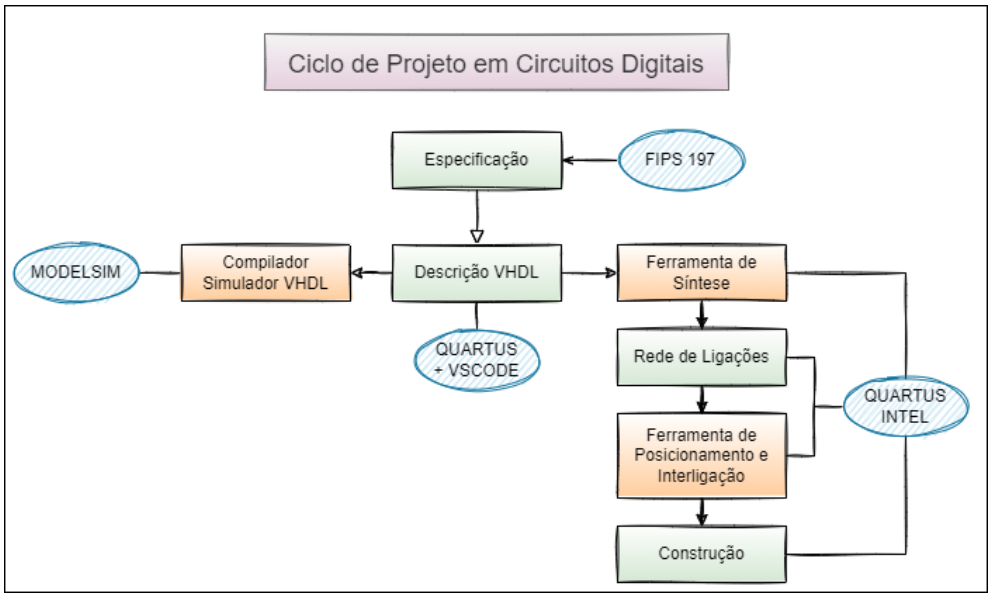

FIGURA 2.3 – Ciclo de Projeto - adaptado (D´AMORE, 2012)

Nessa figura são nominadas algumas ferramentas, em azul, que foram utilizadas no estudo de caso para atuar em cada uma dessas fases. Dessas ferramentas, também chamadas de CAD, do inglês Computer Aided Design, saem os arquivos de extensões próprias para uso na fase seguinte do fluxograma até a fase final da construção no hardware ou gravação na placa real. O teste ou simulação real depende de periféricos que sejam adicionados de acordo com a aplicação. Nesse caso, que se tratava de uma implementação de algoritmo criptográfico, uma entrada de stream de textos para texto em claro e chaves seria um boa opção desde que a placa tenha a capacidade de armazenamento ou quantidade de portas I/O adequadas. Esses conceitos serão melhor detalhados quando do tópico metodologia.

2.2.3 Advanced Encryption Standard - AES

O Advanced Encryption Standard - AES é um algoritmo de criptografia classificado como de chave simétrica criado pelos criptógrafos Vincent Rijmen e Joan Daemen, belgas, em atendimento a uma chamada pública da Secretaria de Comércio do governo americano para o novo padrão de criptografia em segurança de dados. Esse trabalho foi selecionado entre os concorrentes e publicado no NIST - National Institute of Standards and Technology - em 2001 por meio da norma FIPS PUB 197 - Federal Information Processing Standards Publications naquele mesmo ano. Este documento descreve sua finalidade, prncipio de funcionamento, agência mantenedora e aprovadora, dentre outras questões relacionadas à normas americanas, (DAEMEN; RIJMEN, 2001).

O AES, comumente chamado de Rijndael em homenagem aos seus criadores, é utilizado largamente nas mais diferentes aplicações para segurança de dados e tem se mostrado forte o suficiente para resistir a ataques de força bruta até os dias atuais. Ele possui três versões de chaves possíveis, conforme publicação, para encriptação e decriptação dos textos. O tamanho da chave tem relação direta com a dificuldade de quebra considerando uma capacidade computacional fixa. No tocante a esse assunto critérios merecem ser elecados para classificação de robustez de uma solução criptográfica, (STALLINGS, 2015). Seriam eles:

- A incapacidade de um oponente, conhecendo o algoritmo e uma parte do texto cifrado, conseguir extrair o texto em claro ou descobrir sua chave; e
- A incapacidade de um oponente, conhecendo uma ou várias partes dos textos cifrados e seus respectivos textos claros, extrair outros textos claros ou descobrir a chave.

A história por trás da criptografia, e de forma mais genérica a criptologia, é bem vasta e acompanha a história das guerras e a própria essência das transações comerciais quando grupo de pessoas precisavam garantir a segurança de informações ao passarem por trechos hostis ou por território inimigo.

Nos dias atuais as técnicas foram aperfeiçoadas com a entrada da computação, as implementações de segurança e ganham um novo cenário. Tem-se uma constante corrida entre os criptógrafos e criptoanalistas que constantemente buscam criar e aperfeiçoar meios e técnicas de garantir a segurança da informação e por outro lado, tem-se os criptoanalistas que buscam descobrir vulnerabilidades que possam ser devidamente exploradas para exporem essas informações seguras.

A Figura 2.4 abaixo ilustra, de forma muito simplificada, como se dá uma encriptação e decriptação com algoritmos de chave simétrica. Nessa figura é possível perceber a

importância que a chave tem no processo e o quão seguro deve ser o processo de geração e distribuição das mesmas entre os participantes de uma rede de informações sensíveis.

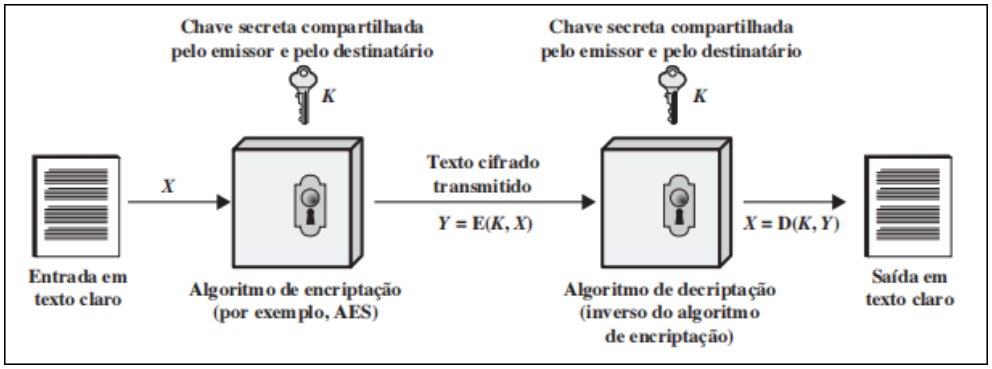

FIGURA 2.4 – Sistema de Criptografia de Chave Simétrica (STALLINGS, 2015)

2.2.4 Gerenciamento de Riscos em Projetos

Como dito no tópico de definição do problema, os riscos associados aos projetos envolvendo FPGA, programados por meio de uma linguagem de descrição de hardware, residem em boa parte na incapacidade de determinada abordagem ou parte do código para determinada tarefa não ser sintetizável considerando as características da placa alvo da implementação. Por sintetizável entende-se como a capacidade de algum código poder ser descrito em hardware que execute a função para a qual foi projetado, (INTEL, 2023).

Existem, no entanto, outros tipos de riscos que precisam ser cobertos na tarefa de identificação, análise e tratamento, seguindo a proposta contida em (GRUBISIC *et al.*, 2009), na estruturação de um software. Eles dizem respeito a finalidade de cada parte que constitui o todo. Para isso é que os termos que ora são tratados como riscos, passam a ser tratados como falhas ou bugs no sentido de que a não satisfação dos testes implica no insucesso total e que portanto o termo mitigação não se aplica. A solução das falhas tem que ser completa e isso só é conseguido com um banco de testes que execute todas as possibilidades e isso para alguns softwares se torna inviável.

Risco é a possibilidade de ocorrência de um evento que venha a ter impacto no cumprimento dos objetivos, (FRAPORTI; BARRETO, 2018). Por mais que esse conceito tenha uma forte ligação com objetivos de empresas no sentido organizacional, pode-se extrapolar para projetos de software e para essa área o termo risco por vezes é chamado de falha, (JORGENSEN, 2018). Numa sequência de termos segundo essa obra, tem-se:

- *Error* - pessoas cometem erros e quando isso ocorre em codificação tem-se o que se chama de *bugs*;
- *Fault* - defeito é o resultado de um erro;

- *Failure* - uma falha ocorre quando um código contendo um defeito é executado.

A figura 2.5 ilustra como é o ciclo de um teste de software.

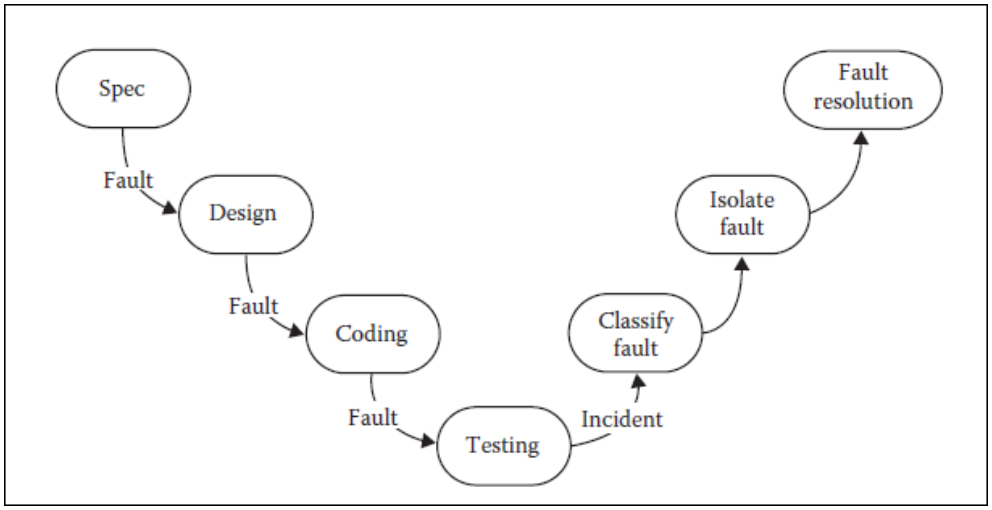

FIGURA 2.5 – Ciclo de um teste de Software - (JORGENSEN, 2018)

Dentro do Processo de Gerenciamento de Projetos - PDP os riscos técnicos estão associados às incertezas das entradas, métodos e saídas das atividades, enquanto que os riscos associados à escopo, tempo, qualidade dentre outros são tidos como riscos gerenciais, (GRUBISIC *et al.*, 2009). Como relacionar os riscos técnicos e gerenciais, para questões de PDP e GP em projetos de desenvolvimento de software é uma questão que pode ser respondida com uma delimitação do escopo do que de fato impacta nas entregas. Isso por si só permite identificar as principais preocupações para que o software proposto funcione, funcionalidade, dentre todas as demais preocupações gerenciais de mudança de necessidades de clientes, comunicação com a equipe, etc.

Como se trata de uma aplicação especificada por meio de uma publicação aberta e disponível para diferentes abordagens de desenvolvimento, a afirmação acima de não fazer sentido se preocupar com necessidades de clientes, comunicação com equipe e mudanças de escopo citados por (GRUBISIC *et al.*, 2009) parece fazer sentido. No entanto, a referência precisa ter um forte apelo de testabilidade de ferramentas que performam a aplicação que se pretende desenvolver. Para isso, os próximos tópicos sobre módulos do algoritmo de criptografia e a ferramenta de teste, MATLAB, elucidam essas questões pontualmente.

2.2.5 Failure Mode and Effects Analysis - FMEA

Como uma importante ferramenta da qualidade voltada para a produção, o FMEA além de permitir analisar de forma qualitativa os modos de falha, permite estudar os efeitos gerados por eles, (BALLESTERO-ALVAREZ, 2012). Pode ser representado por uma tabela onde são colocados todos os riscos nos processos em linhas e colunas referentes

Modelo de Formulário para a Análise do Modo e do Efeito de Falha (FMEA)

1	2	3	4	5	6	7	8	9	10	11	12	13			
												Resultados das ações			
Função do processo	Modo de falha potencial	Efeito potencial da falha	Índice de severidade	Causa e mecanismo potencial	Índice de ocorrência	Controles atuais do processo	Índice de detecção	NPR	Ações recomendadas	Responsável e prazo	Ações tomadas	Severidade	Ocorrência	Detecção	NPR

FIGURA 2.6 – Modelo de FMEA - (BALLESTERO-ALVAREZ, 2012)

a: seus efeitos, gravidade dos efeitos sobre o projeto ou consecução dos objetivos, causas dessas falhas, alguma forma de se detectar essas falhas ou algum controle que fora criado, responsáveis e prazos, ações recomendadas, etc. A figura 2.6 abaixo ilustra o modelo fornecido por (BALLESTERO-ALVAREZ, 2012).

Este modelo serve como base para uma construção simplista do projeto de desenvolvimento de software. Essa ideia sendo extrapolada para o projeto deste trabalho teria uma sequência dos módulos que compõem o algoritmo e uma descrição do caso de falha qual a severidade, probabilidade de ocorrer e sua probabilidade de detecção de alguma forma. Estes graus podem ser numerais e pode-se citar as origens ou causas dele ocorrer de forma descritiva. O que se quer com essa ferramenta de qualidade na produção adaptada para o desenvolvimento de software é poder saber onde se deve alocar homem hora ou atenção especial na codificação considerando que a integração dos módulos deve ser funcional e poder ser sintetizável como prometido na metodologia.

Para tanto o modelo adaptado para a conter apenas as colunas contidas na tabela 2.1 abaixo onde os comentários são listados aqui.

- Os valores dos parâmetros de probabilidade, severidade e probabilidade de ocorrência vão de 1 a 7, onde 1 representa pequena probabilidade ou pouco grave enquanto que 7, alta probabilidade e risco alto;
- Como para todos os processos se tem um processo de testes baseado num gabarito foi atribuído à essa coluna o valor de 6 para todos. O valor 7 seria se estes testes fossem de fato infalíveis;
- Os sete processos compõem os módulos da encriptação com um acréscimo da função de criação das SBox e Inv-SBox como estão dispostas no Apêndice A;
- As células não numeradas são qualitativas e conforme visto em (BALLESTERO-ALVAREZ, 2012), devem ser descritas.

- O parâmetro que se procura como referência para priorização é o NPR pois engloba tanto a probabilidade de ocorrência como de detecção e a severidade ou gravidade da falha. Quem possui mais influência sobre determinada saída, como ajustar os processos ou, como colocado por (BALLESTERO-ALVAREZ, 2012), fatores, de modo que as saídas estejam num nível desejado ou tenha variação mínima.

TABELA 2.1 – Modelo de FMEA adaptado para Desenvolvimento da Aplicação - Autor

Processo	1	2	3	4	5	6	7	8	9	10	11			
											Resultados das ações			
	Função do submódulo	Modo de Falha Potencial	Efeito Potencial da Falha	Índice de Severidade (Aplica-se somente ao efeito)	Causa provável	Índice de ocorrência	Controle que pode ser implementado	Índice de Detecção	NPR = (Severidade) x (Ocorrência) x (Detecção)	Ações Tomadas	Severidade	Ocorrência	Detecção	NPR
1	Add round	Essa função é mais simples de ser implementada e é a mais requisitada ao longo do algoritmo	Como ela é a mais requisitada, se falhar tem um impacto grande na integração, mas por ser simples de se implentar, tem severidade baixa	5	escrita errada da fórmula que executa o comando bitwise - XOR	5	Sim. Por meio de um gabarito como foi feito.	6	150	De mais simples implementação, porém a que mais se repete, teve segundo maior NPR, logo executar uma vez, testar e copiar em todas as instâncias onde essa função é chamada	NIL	NIL	NIL	NIL
2	Expansion Key	Função mais complexa, porém executada uma única vez implica alta severidade e alta probabilidade de ocorrer	Esse processo é composto de subprocessos que podem falhar	5	Integração dos submódulos, rotação, multiplicação de vetor, mudança de linha	6	Gabarito	6	180	Maior anteção e alocação de mais recursos humanos neste processo. Escolha de codificador e revisor	NIL	NIL	NIL	NIL
3	SBox / Inv_Sbox	Criação das tabelas que serão usadas nas substituições	Erro de digitação. São 256 números hexadecimais	3	Erro de digitação	4	Gabarito	6	72	Cuidado na transcrição dos números	NIL	NIL	NIL	NIL
4	SubBytes	Trata-se apenas de uma forma de substituição. Se a tabela do processo 3 estiver correta, esse processo também estará.	pouco provável	2	indexação errada	3	Gabarito	6	36	Cuidado na transcrição dos números	NIL	NIL	NIL	NIL
5	Shift rows	Deslocamento de linhas. Muito fácil de ser feito tanto em MATLAB quanto em VHDL	pouco provável	2	aplicação da fórmula	3	Gabarito	6	36	Cuidado nas multiplicações dos números	NIL	NIL	NIL	NIL
6	Mix columns	Mistura de colunas. Um pouco mais difícil do que o processo anterior apenas no VHDL por envolver multiplicação de vetores.. Em MATLAB bem fácil	pouco provável	3	multiplicação não bem feita entre vetores	3	Gabarito	6	54	Cuidado nas multiplicações dos números	NIL	NIL	NIL	NIL
7	Integração	Se qualquer um dos processos acima falhar, a integração falhará e gerará se não um resultado diferente, uma disposição diferente	A junção de tudo pode ter várias fontes de falhas	6	Falha de disposição ou qualquer falha anterior	6	Gabarito	6	216	Maior alocação de RH. Sugestão de 2 codificadores e 2 revisores. Depois cruza-se os revisores e desenvolvedores.	NIL	NIL	NIL	NIL

2.2.6 Módulos do AES

Os módulos do AES são as partes mínimas que executam a mesma função reiteradas vezes de acordo com a versão do algoritmo. Existem 3 versões disponíveis, 128 bits, 192 bits e 256 bits e esses valores dizem respeito ao tamanho da chave e texto claro que são as entradas para que as operações contidas nos módulos sejam executadas, (DAEMEN; RIJMEN, 2001).

A figura 2.7 ilustra o encadeamento dos módulos para que o resultado final, texto cifrado, seja alcançado. Nesse caso está sendo exemplificada a versão de 128 bits em que tanto a chave fornecida pelo usuário quanto o texto claro tem tamanhos de 128 bits. Esses detalhes fazem toda a diferença na execução dos testes, pois é preciso saber o que se deve obter na saída em termos tanto de valores quanto na quantidade desses valores. Por valores cabe nesse momento destacar que todo o texto, seja chave ou texto em claro que vai ser criptografado, inicialmente é transformado em caracteres hexadecimal e nesse tipo de numeração todas as operações são executadas conforme ficará mais claro no detalhamento dos módulos mais a frente.

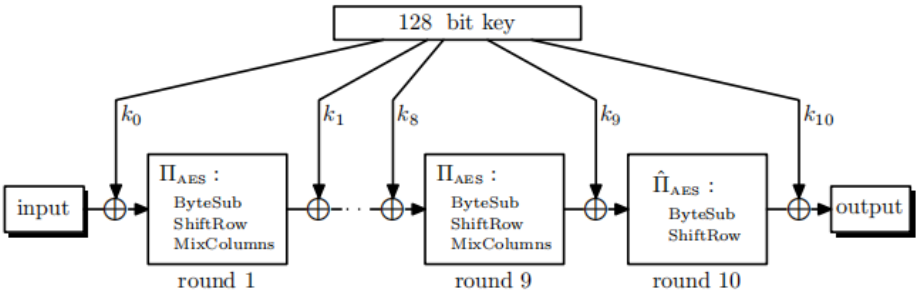

FIGURA 2.7 – Esquemático dos módulos do AES-128 - (BONEH; SHOUP, 2020)

O algoritmo AES possui uma relação entre a versão, 128, 192 e 256 bits, e o número de rodadas (*rounds*) previstas conforme tabela abaixo, no entanto esses detalhes serão comentados somente no tópico de desenvolvimento quando dos testes uma vez que sua explicação é exemplificada facilitando o entendimento pela visualização no tópico corrente.

A tabela 2.2 mostra como se dá essa relação em termos de tamanho da chave, número de rodadas e tamanho de chave expandida. Esse conhecimento aqui é imprescindível para entendimento dos módulos a seguir.

- *Expansion Key* - primeira operação executada no processo de criptografia e também usada na decriptografia, executa uma função de transformar uma chave de tamanho de 128 bits, 192 bits ou 256 bits em 11, 13 ou 15 blocos de 16 bytes a serem utilizados em cada rodada do algoritmo. A tabela 2.2 e a figura 2.8 ajuda a entender como se dá essa utilização. o símbolo $\bigoplus$ significa uma soma XOR (OR EXCLUSIVO) da

TABELA 2.2 – Versões do AES e tamanhos de Chave Expandida - (STALLINGS, 2015)

Tamanho da chave (words/bytes/bits)	4/16/128	6/24/192	8/32/256
Tamanho do bloco de texto claro (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Número de rodadas	10	12	14
Tamanho da chave de rodada (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Tamanho da chave expandida (words/bytes)	44/176	52/208	60/240

eletrônica digital, onde duas entradas são somadas resultando em uma única saída de acordo com seus valores. bits diferentes tem saída 1 e bits iguais saída 0.

Essa operação é invertível querendo dizer que uma saída, texto cifrado, sendo somado a mesma chave tem como resultado o texto claro que fora inserido no início. Essa propriedade é melhor descrita de acordo com a equações abaixo:

$$A \oplus B = C \tag{2.1}$$

$$A \oplus C = B \tag{2.2}$$

$$\text{chave} \oplus \text{texto claro} = \text{texto cifrado} \tag{2.3}$$

$$\text{chave} \oplus \text{texto cifrado} = \text{texto claro} \tag{2.4}$$

Trata-se de um sequência de operações em que o estado, aquele array $4x4$ bytes passa por uma rotação, substituição e multiplicação por um vetor RCj que é função da rodada se o indexador for múltiplo de 4 e, para os demais casos, apenas uma soma, OR EXCLUSIVO, entre o elemento imediatamente anterior e 4 posições antes.

- *add round* - essa função é aquela que executa a soma citada acima onde duas entradas, texto claro e chave ou texto cifrada e chave, dão origem a uma única saída, texto cifrado ou texto claro, para ser usada pelo módulo seguinte e é simbolizada na figura 2.7 acima $\bigoplus$ (XOR). Essa saída é chamada de **estado** e sempre tem o tamanho de 128 bits ou 16 bytes que são estruturados numa matriz de $4x4$.

- *sub bytes*

  Esse módulo ou função executa uma substituição de um valor da matriz de estado por outro valor. Trata-se portanto de uma aplicação em que cada valor da matriz do estado, 16 valores possíveis de 1 byte cada, é substituído por outro valor formando uma nova matriz. O detalhe interessante que precisa ser ressaltado para que os devidos créditos aos criptográfos sejam dados é que esses valores substitutos não são

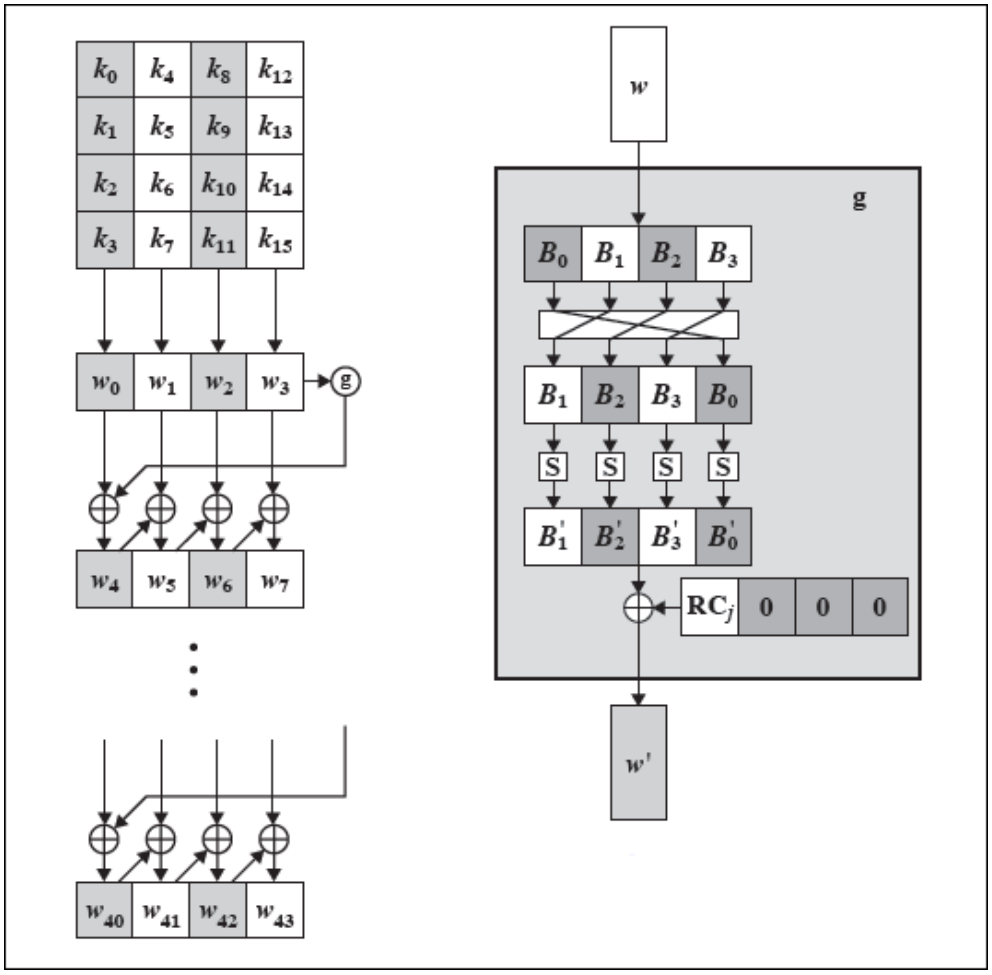

FIGURA 2.8 – Expansão de Chave - (STALLINGS, 2015)

valores dados ou aleatórios e sim calculados. Esse cálculo utiliza uma matemática da álgebra abstrata conhecida como Corpo finito de Galois notadamente a operação de inverso multiplicativo. Os detalhes dessa matemática não precisam ser enunciados aqui, mas apenas entender que assim como se tem um cálculo que para cada valor da matriz do estado se tem um novo valor substituído na criptografia, na decriptografia ocorre o mesmo, sendo uma função invertível como era de se supor.

A figura 2.9 abaixo ilustra como se dá esse processo de forma macro, porém para o fim da análise de risco que esse trabalho enfatiza, uma forma de testar se o código executa bem essa função seria testar todos os 256 valores possíveis dado que se trata de uma quantidade razoável para ferramentas rápidas como o matlab. Essa demonstração consta do desenvolvimento de testes por módulo.

Cada byte possui 8 bits e assim cada valor da matriz do estado possui 2 números hexadecimais já que 2^4x2^4. Um exemplo seria o número:

$$(E4)_{hex} = (1404)_{dec} = (11100100)_{bin}$$

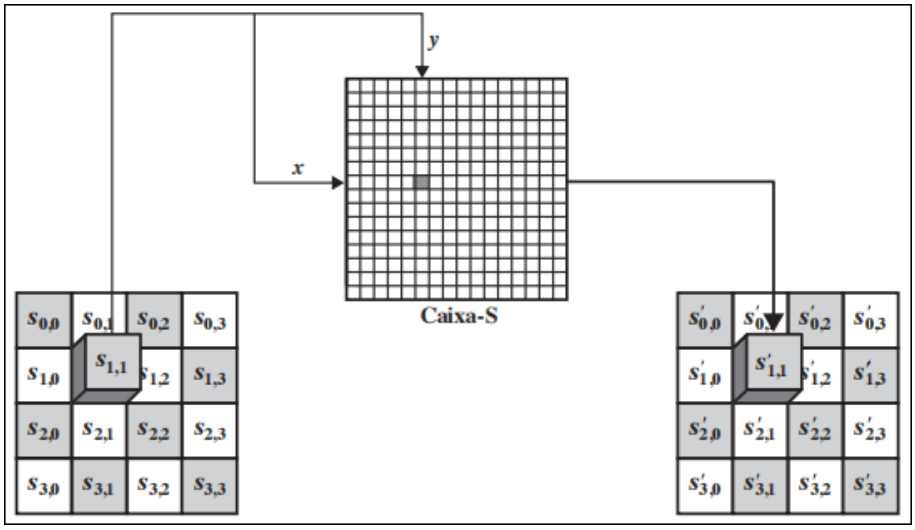

FIGURA 2.9 – Função Sub bytes - (STALLINGS, 2015)

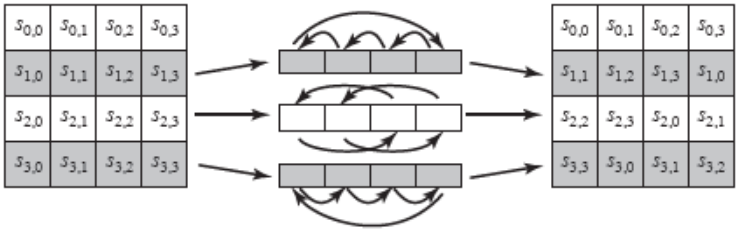

FIGURA 2.10 – Deslocamento de linhas - (STALLINGS, 2015)

Nesse caso uma tabela contendo todos os valores possíveis teria um índice de linha, para os 4 bits mais significativos, MSB, e um índice de coluna para os 4 bits menos significativos, LSB. Assim essa tabela teria um tamanho de 256 valores, 16 x 16.

Essa tabela sendo construída uma única vez, seria suficiente ocupar posições de memória para contê-los e formas rápidas de acesso como uma opção bem mais rápida do que processar o cálculo a cada valor de entrada da matriz de estado.

Seriam portanto 2 tabelas, A.1 e A.2, as quais constam do Apêndice A.

- *shif rows*

  A função ou módulo Shiftrows, deslocamento de linhas, como o nome sugere, desloca as linhas da matriz do estado em 0, 1, 2 e 3 posições para a esquerda na encriptação e para a direita na decriptação. É uma simples modificação de posições da matriz do estado trazendo confusão aos dados. No matlab essas funções são bem mais fáceis de se executar, devido às funções nativas da ferramenta, mas em VHDL é preciso ter um pouco mais de trabalho para se trabalhar com posições.

  A figura 4.4 abaixo ilustra como se dá essa modificação de posições. Por similaridade a recombinação do dado original é feita com o deslocamento no sentido contrário.

- *mix collumns*

Essa função realiza um embaralhamento dos valores das colunas da matriz que representa o estado tanto de forma direta, encriptação, quanto inversa, decriptação. Cada byte de uma coluna é mapeado para um novo valor que é função dos demais valores daquela coluna.

As fórmulas matemáticas que operacionalizam esse mapeamento são dadas pela equação 2.5 e 2.6 abaixo representando as figuras 2.11 e 2.12, respectivamente, em seguida.

$$\begin{aligned} 02 \cdot s_{0,j} \oplus 03 \cdot s_{1,j} \oplus 02 \cdot s_{2,j} \oplus s_{3,j} &= s'_{0,j} \\ s_{0,j} \oplus 02 \cdot s_{1,j} \oplus 03 \cdot s_{2,j} \oplus s_{3,j} &= s'_{1,j} \\ s_{0,j} \oplus s_{1,j} \oplus 02 \cdot s_{2,j} \oplus 03 \cdot s_{3,j} &= s'_{2,j} \\ 03 \cdot s_{0,j} \oplus s_{1,j} \oplus s_{2,j} \oplus 02 \cdot s_{3,j} &= s'_{3,j} \end{aligned} \tag{2.5}$$

$$\begin{aligned} 0E \cdot s_{0,j} \oplus 0B \cdot s_{1,j} \oplus 0D \cdot s_{2,j} \oplus 09 \cdot s_{3,j} &= s'_{0,j} \\ 09 \cdot s_{0,j} \oplus 0E \cdot s_{1,j} \oplus 0B \cdot s_{2,j} \oplus 0D \cdot s_{3,j} &= s'_{1,j} \\ 0D \cdot s_{0,j} \oplus 09 \cdot s_{1,j} \oplus 0E \cdot s_{2,j} \oplus 0B \cdot s_{3,j} &= s'_{2,j} \\ 0B \cdot s_{0,j} \oplus 0D \cdot s_{1,j} \oplus 09 \cdot s_{2,j} \oplus 0E \cdot s_{3,j} &= s'_{3,j} \end{aligned} \tag{2.6}$$

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

FIGURA 2.11 – Multiplicação de matrizes direta - (STALLINGS, 2015)

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

FIGURA 2.12 – Multiplicação de matrizes inversa - (STALLINGS, 2015)

Para a operação inversa resta multiplicar o estado pela matriz inversa calculada abaixo:

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \bullet \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \bullet \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Assim,

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix}$$

2.2.7 Ferramenta de Teste - MATLAB

O MATLAB (MATrix LABoratory) foi a ferramenta de teste escolhida para ser utilizada ao longo deste trabalho por se tratar de uma plataforma de programação livre para estudantes e bastante utilizada para análise numérica e visualização de dados além de ser de fácil acesso e uso (VALENTINE; HAHN, 2022).

Era preciso que fosse escolhida uma ferramenta computacional rápida e de alto nível que permitisse uma verificação do que estava sendo proposto para codificação em baixo nível, tal como a linguagem VHDL. Essa abordagem foi utilizada no trabalho estudo de caso e nesse aspecto, por mais que fosse exigido o domínio de conhecimento em duas linguagens, o indicador de homem-hora foi recompensado na economia de tempo que seria gasto nos *debugs*.

No entanto, o que se busca aqui, tal como descrito na metodologia é a utilização dessa mesma linguagem de alto nível para teste e análise de riscos no tocante a capacidade de ser sintetizável ou que pelo menos todas as possibilidades sejam testadas.

3 METODOLOGIA

A metodologia adotada para esse trabalho usa a aplicação integrada de riscos gerenciais e técnicos no processo de desenvolvimento de uma aplicação computacional por meio de um estudo de caso. Esse estudo de caso está contido no trabalho de mestrado (OLIVEIRA, 2023) onde se tem uma proposta de implementação do Algoritmo de Criptografia Rijndael de Criptografia Simétrica por meio de FPGA.

A partir desse trabalho, procura-se identificar, analisar quantitativamente e qualitativamente os riscos inerentes a esse tipo de projeto utilizando FPGA e que tenham potencial de gerar impactos negativos que possam afetar demasiadamente os objetivos do produto final. Os riscos, no entanto, foram delimitados à capacidade de ser sintetizável de onde se exclui os riscos inerentes às vulnerabilidades que os chips possuem tais como radiação de alta energia de partículas radioativas, dentre outras.

Também é levado em consideração, como modelos de análises de falhas o trabalho de (FLESCH IGOR TODESCHI, 2020) que propõe, com base na norma (FLESCH *et al.*, 2020), uma metodologia de segurança funcional para Confiabilidade Crítica também em projetos envolvendo FPGA.

Esse trabalho aponta outras fontes de consulta sobre que métodos de análise de falhas podem ser empregados nesse nicho de tecnologia de forma a tornar funcional determinada aplicação. A abordagem de reduzir soluções de códigos complexos em pedaços tão pequenos e simples quanto possível foi perseguido aqui e, via de regra, tem sido uma prática quase que unânime para os desenvolvedores de soluções computacionais que usam essa tecnologia, (KARIMI *et al.*, 2018).

O Modelo V, descrito na Norma Internacional IEC 61508-2, (FLESCH *et al.*, 2020), sugere verificação e validação em cada fase do desenvolvimento do projeto.

Para tanto tem-se a junção das áreas de gerenciamento de riscos e engenharia de software, bem com testes de software uma vez que no desenvolvimento de aplicações computacionais a testabilidade percorre basicamente todo o ciclo de desenvolvimento.

Para a engenharia de software foi escolhida a linguagem MATLAB para execução dos testes por se tratar de uma linguagem de fácil acesso e de rápido processamento para que

as possibilidades de valores de saídas dos módulos sejam testados e dispostos de forma indexada tal como o conceito de Estado sugere. O Estado é uma matriz contendo o resultado de uma operação pronto para ser utilizada por outra quantas vezes a versão do algoritmo prevê,

Além disso, como segunda parte da metodologia, tem-se a capacidade de síntese. Isso será feito por meio de um CAD (Computer Aided Design) chamado Quartus que executa a compilação e verificação de erros para a síntese, (INTEL, 2023). Esses testes não fazem parte do escopo deste trabalho uma vez que o CAD já executa apontando os pontos de conflito da tradução da linguagem em configuração dos elementos lógicos.

4 DESENVOLVIMENTO DO ESTUDO DE CASO

Os riscos em projetos de desenvolvimento de softwares objeto deste trabalho como mencionado na metodologia, leva em consideração o Modelo V em que cada fase passa por verificação e validação desde a codificação na base. A figura 4.1 ilustra o que está sendo dito onde se pode perceber o quão interessante e entrelaçado são as etapas pelas quais um insight se transforma em um entregável documentado e pronto para o lançamento.

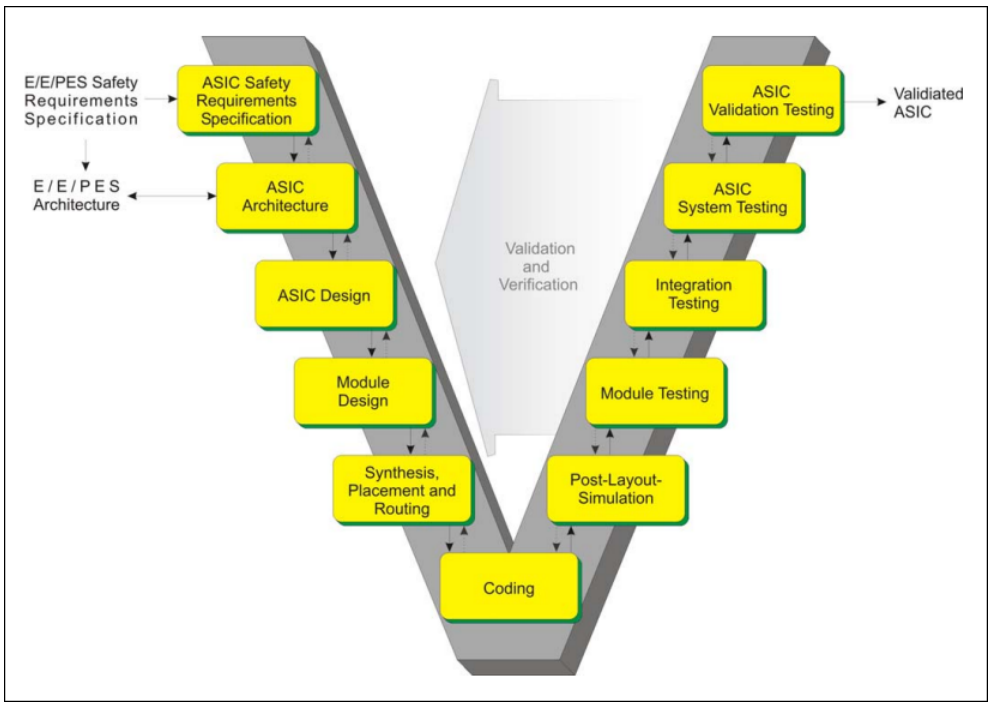

FIGURA 4.1 – Modelo em V para desenvolvimento de projetos (HAYEK; BöRCSöK, 2012)

Existem algumas variedades desse modelo no tocante ao produto que se queira desenvolver, no entanto, alguns aspectos são comuns.

Considerando um escopo exequível para esse tipo de trabalho será considerado apenas a codificação que permite a descrição de hardware e a capacidade de síntese e testes unitários.

4.1 Nivelamento quanto a testabilidade

Os riscos quanto a testabilidade tem o propósito de se verificar se a lógica que traduz a matemática de uma solução, nesse caso a matemática envolvida nesta criptografia publicada em NIST 197, é suficientemente forte para um conjunto de entradas de testes conforme ciclo explicado na figura 2.5.

Feita essa ressalva resta aplicar uma forma de verificar se todas as entradas previstas tem as respostas esperadas por cada módulo. É sabido que na criptografia tem-se um texto claro e uma chave que são misturados de uma forma específica de acordo com o algoritmo. Essa mistura se dá não pelo nosso código linguístico ou português e sim por meio de caracteres hexadecimais que são facilmente transformados em bits, menor unidade da informação e assim passar pelas portas lógicas da eletrônica. Os resultados são então recombinados no sentido inverso tendo o texto claro novamente desde que se use a mesma chave.

As publicações sobre testes de software, vide (JORGENSEN, 2018) e (COPELAND, 2004), trazem em comum duas opções para se testar softwares chamadas de black box, quando não se sabe o que é feito internamente e apenas se tem a visualização das variáveis de entradas e saídas, e a white box, onde se conhece o que está sendo processado internamente. Trazem entre outros conceitos níveis de testes onde citam:

- **Testes Unitários (*Units Test*)** como a menor parte de um software que o desenvolvedor cria;
- **Testes de Integração (*Integration Tests*)** onde se monta as unidades juntas em subsistemas e depois sistemas;
- **Testes de Sistemas (*System Testing*)** que incluem todos os testes possíveis de software e hardware daquilo que é entregue ao cliente como produto e envolve funcionalidade, usabilidade, segurança, disponibilidade, capacidade, performance, portabilidade e muito mais; e
- **Testes de Aceitação (*Acceptance Tests*)** como aquele que se passado sem problemas ao final se tem início a contraparte contratual do pagamento a grosso modo.

Quem define o que será testado em cada uma dessas fases é uma questão que pode ser colocada como fonte de riscos de projetos, que não é o foco deste trabalho, mas aponta o quão amplo e interligado é esse tema.

Existe ainda uma máxima dentro da temática de testes de software que não se pode testar tudo, (COPELAND, 2004), no entanto aqui cabe uma ressalva para esse trabalho em específico por que uma vez que o texto em claro ou a voz depois de digitalizada venha

a ser uma entrada do algoritmo ou do código escrito, assim como a chave, tudo vai ser tratado como números hexadecimais que possuem um tamanho fixo de 16 números, além disso, a aplicação que fora desenvolvida no estudo de caso não importa quão grande seja o texto a ser criptografado ou decriptografado, tudo será particionado em blocos de 128 bits, chamados de estado, que passará por todos os módulos até a sua recombinação e transformação em linguagem inteligível ao final.

4.2 Testes Unitários e Identificação de riscos

Como dito no subtópico acima, esse tipo de solução permite testar tudo, mas seria demasiadamente exaustivo e demorado fazê-lo. Por isso considerando que a publicação que tornou público o AES128/192/256 possui um exemplo de funcionamento de seus módulos para verificação e validação dos futuros desenvolvedores, esses resultados será usado neste trabalho como oráculo e somente ele será testado, mesmo a ferramenta MATLAB permitindo que seja feita com outras variáveis.

A primeira função é mais complexa é a função ou módulo de expansão de chave já explicada no tópico de fundamentação teórica. Todo o algoritmo foi publicado em pseudocódigo por que uma das premissas do concurso internacional era de que não se desse preferência por nenhuma linguagem e que pudesse ser implementado tanto em software quanto em hardware. Assim os pseudocódigos constam no apêndice A e as idéias de desenvolvimento pelas variadas linguagens constantes dos fóruns e repositórios da área trazem inúmeras idéias das mais variadas possíveis como um mural de pintura onde o artista, nesse caso, codificador, tem seu espaço.

Não é escopo deste trabalho construir nada que já tenha sido feito e sim apenas ter uma forma de testar e identificar riscos de implementação numa solução computacional para esse projeto.

4.2.1 Testes de Expansão de Chave (*Expansion Key*)

Esta função é a mais complexa entre todas, no entanto é feita uma única vez e as chaves da rodada conforme criadas, são chamadas de acordo com um indexador. Um bom teste além de verificar se o código coloca o resultado correto com base no gabarito da publicação, seria a verificação se cada valor do estado está devidamente posicionado para o uso futuro.

Será considerada a chave abaixo da referência (DAEMEN; RIJMEN, 2001) para se ter uma forma de verificar o correto funcionamento como teste de funcionalidade. Outros valores poderiam ser colocados tal como constante do Apêndice C para cada função dos

módulos.

2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C

Antes do teste de expansão é preciso ter uma subfunção que realiza a substituição. Essa função é chamada de SubWord no pseudocódigo constante do Apêndice A. Ela está disposta e demonstrada na figura 4.2 abaixo. Assim:

```
function [out]=sbox(in)
  in = [0x2b 0x7e 0x15 0x16 0x28 0xae 0xd2 0xa6 0xab 0xf7 0x15 0x88 0x09 0xcf 0x4f 0x3c];
  table = [
  '63'; '7C'; '77'; '7B'; 'F2'; '6B'; '6F'; 'C5'; '30'; '01'; '67'; '2B'; 'FE'; 'D7'; 'AB'; '76';
  'CA'; '82'; 'C9'; '7D'; 'FA'; '59'; '47'; 'F0'; 'AD'; 'D4'; 'A2'; 'AF'; '9C'; 'A4'; '72'; 'C0';
  'B7'; 'FD'; '93'; '26'; '36'; '3F'; 'F7'; 'CC'; '34'; 'A5'; 'E5'; 'F1'; '71'; 'D8'; '31'; '15';
  '04'; 'C7'; '23'; 'C3'; '18'; '96'; '05'; '9A'; '07'; '12'; '80'; 'E2'; 'EB'; '27'; 'B2'; '75';
  '09'; '83'; '2C'; '1A'; '1B'; '6E'; '5A'; 'A0'; '52'; '3B'; 'D6'; 'B3'; '29'; 'E3'; '2F'; '84';
  '53'; 'D1'; '00'; 'ED'; '20'; 'FC'; 'B1'; '5B'; '6A'; 'CB'; 'BE'; '39'; '4A'; '4C'; '58'; 'CF';
  'D0'; 'EF'; 'AA'; 'FB'; '43'; '4D'; '33'; '85'; '45'; 'F9'; '02'; '7F'; '50'; '3C'; '9F'; 'A8';
  '51'; 'A3'; '40'; '8F'; '92'; '9D'; '38'; 'F5'; 'BC'; 'B6'; 'DA'; '21'; '10'; 'FF'; 'F3'; 'D2';
  'CD'; '0C'; '13'; 'EC'; '5F'; '97'; '44'; '17'; 'C4'; 'A7'; '7E'; '3D'; '64'; '5D'; '19'; '73';
  '60'; '81'; '4F'; 'DC'; '22'; '2A'; '90'; '88'; '46'; 'EE'; 'B8'; '14'; 'DE'; '5E'; '0B'; 'DB';
  'E0'; '32'; '3A'; '0A'; '49'; '06'; '24'; '5C'; 'C2'; 'D3'; 'AC'; '62'; '91'; '95'; 'E4'; '79';
  'E7'; 'C8'; '37'; '6D'; '8D'; 'D5'; '4E'; 'A9'; '6C'; '56'; 'F4'; 'EA'; '65'; '7A'; 'AE'; '08';
  'BA'; '78'; '25'; '2E'; '1C'; 'A6'; 'B4'; 'C6'; 'E8'; 'DD'; '74'; '1F'; '4B'; 'BD'; '8B'; '8A';
  '70'; '3E'; 'B5'; '66'; '48'; '03'; 'F6'; '0E'; '61'; '35'; '57'; 'B9'; '86'; 'C1'; '1D'; '9E';
  'E1'; 'F8'; '98'; '11'; '69'; 'D9'; '8E'; '94'; '9B'; '1E'; '87'; 'E9'; 'CE'; '55'; '28'; 'DF';
  '8C'; 'A1'; '89'; '0D'; 'BF'; 'E6'; '42'; '68'; '41'; '99'; '2D'; '0F'; 'B0'; '54'; 'BB'; '16'];

      decimal=hex2dec(table);
      out=decimal(in + 1);
      out=reshape(string(dec2hex(out)),1,16);

  end
```

```
ans =

  1×16 string array

  Columns 1 through 9

    "F1"    "F3"    "59"    "47"    "34"    "E4"    "B5"    "24"    "62"

  Columns 10 through 16

    "68"    "59"    "C4"    "01"    "8A"    "84"    "EB"
```

FIGURA 4.2 – Subfunçao da Expansão de Substituição (Autor)

Há ainda uma subfunção dentro da função de expansão de chave chamada de RotWord, conforme pode ser visto no Pseudocódigo do Apêndice B.3, linha 13, mas aqui isso é feito de forma mais otimizada por meio de mudança de posição o que o MATLAB executa muito bem dada a sua capacidade operar com indexadores de matrizes. Assim o código abaixo executa a expansão de chave para a chave fornecida gerando os valores mostrados na

figura 4.3 logo após o código. Essa função não será testada todos os valores por gerar uma quantidade de tabelas alta, 16^{16} tabelas. Mas a expansão de todas as chaves previstas no gabarito da publicação (DAEMEN; RIJMEN, 2001) consta do Apêndice C.2 para conferência. O símbolo de ok representa que os valores estão de acordo com o gabarito e na disposição correta.

```
function [result] = key_schedule(~)

    keyHex=[0x2b 0x7e 0x15 0x16 0x28 0xae 0xd2 0xa6 0xab 0xf7 0x15 0x88 0x09 0xcf 0x4f 0x3c];

    round_key_256 = zeros(11,16);
    round_key_256(1,:) = keyHex;
    rcon = [1, 2, 4, 8, 16, 32, 64, 128, 27, 54];
        w0 = round_key_256(1,1:4);
        w1 = round_key_256(1,5:8);
        w2 = round_key_256(1,9:12);
        w3 = round_key_256(1,13:16);

    result(1) = bitxor(bitxor(sbox(w3(2)), w0(1)), rcon(1));
    result(2) = bitxor(sbox(w3(3)), w0(2));
    result(3) = bitxor(sbox(w3(4)), w0(3));
    result(4) = bitxor(sbox(w3(1)), w0(4));

    result(5:8)   = bitxor(result(1:4), w1);
    result(9:12)  = bitxor(result(5:8), w2);
    result(13:16) = bitxor(result(9:12), w3);

    result=reshape(string(dec2hex(result)),4,4);

end
```

```
ans =

  4x4 string array

    "A0"    "88"    "23"    "2A"
    "FA"    "54"    "A3"    "6C"
    "FE"    "2C"    "39"    "76"
    "17"    "B1"    "39"    "05"
```

FIGURA 4.3 – Resultado Expansão 128 bits - Validado

4.2.2 Testes de Somada da Rodada (Add round)

A função mais simples de testar é a *Add round*. Essa função executa uma operação bit a bit (*bitwise*) de duas entradas dadas. Assim o script abaixo:

4.2.3 Testes de substituição de bytes (*Subbytes*)

Essa função já fora testada e validada na expansão de chave como subfunção.

```
function [state_out] = shiftrows(state_in)

state_in=["D4" "27" "11" "AE" "E0" "BF" "98" "F1" "B8" "B4" "5D" "E5" "1E" "41" "52" "30"];

state_out(1) = state_in(1);
state_out(2) = state_in(6);
state_out(3) = state_in(11);
state_out(4) = state_in(16);

state_out(5) = state_in(5);
state_out(6) = state_in(10);
state_out(7) = state_in(15);
state_out(8) = state_in(4);

state_out(9) = state_in(9);
state_out(10) = state_in(14);
state_out(11) = state_in(3);
state_out(12) = state_in(8);

state_out(13) = state_in(13);
state_out(14) = state_in(2);
state_out(15) = state_in(7);
state_out(16) = state_in(12);

state_out=reshape(state_out,4,4);
end
```

d4	e0	b8	1e
27	bf	b4	41
11	98	5d	52
ae	f1	e5	30

```
ans =

  4×4 string array

    "D4"    "E0"    "B8"    "1E"
    "BF"    "B4"    "41"    "27"
    "5D"    "52"    "11"    "98"
    "30"    "AE"    "F1"    "E5"
```

d4	e0	b8	1e
bf	b4	41	27
5d	52	11	98
30	ae	f1	e5

FIGURA 4.4 – Validação Deslocamento de Linhas - Adaptado (AES..., 2019)

4.2.4 Testes de Deslocamento de Linhas (*Shift Rows*)

Para esse teste será criado um array 4 x 4 e espera-se que as modificações ocorram como esperado, a saber, $i - 1$ posições à esquerda na encriptação e $i - 1$ posições para a direita em que i é o número da linha do array. Assim:

No desenvolvimento desse módulo havia uma incoerência entre a disposição dos valores que vinha do módulo anterior e a disposição dos valores dentro do array. Isso foi consertado com um comando simples de transposição, 4.5, mas essa é uma das mais simples interações que se faz para que o valor correto, validado via gabarito, e a disposição correta para ser usada no próximo estágio, também esteja correto.

4.2.5 Testes de Mistura de colunas (*Mix Columnns*)

O resultado desse módulo e seu código, constam do Apêndice C, C.2.

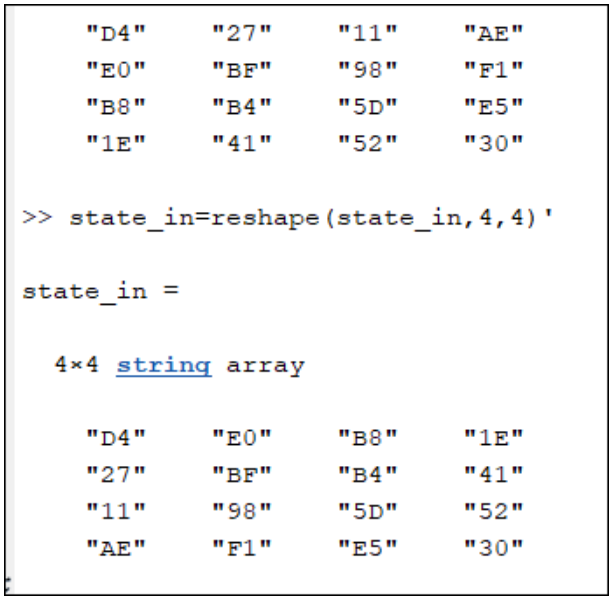

FIGURA 4.5 – Exemplo de debug de disposição- Autor

4.3 Testes de Integração

Nesse tópico finalmente se tem a junção de todos os módulos vistos até aqui numa função única. Na figura 4.6 retirada do resultado do script ou código contido no Apêncide C (Integração), C.1, e corrobora o que vinha sendo defendido.

A integração de encripitação é mesma da decriptação mudando-se apenas os valores que multiplicam as variáveis de entrada e as tabelas. Como se está comparando o resultado com um gabarito sabidamente correto e sem preferência por nenhuma linguagem, os testes tem sua validade.

Nesta figura tem-se duas marcações em vermelho para situar onde foram inseridas a entradas de texto claro (input) e chaves (key) com base no livro da referência (DAEMEN; RIJMEN, 2001). A segunda marcação demonstra que o resultado foi correto como se pretendia demonstrar. Houve uma mescla portanto de 7 funções em uma apenas além das construções da tabela com maior esforço sem dúvidas no último módulo que é o de integração como a tabela 2.1 apontava.

4.4 Aplicação da FMEA

A aplicação da FMEA nesse trabalho com base no estudo de caso gerou o resultado conforme Tabela 4.1 abaixo onde se percebe, pela modificação da ordem das atividades e pelos pesos no NPR, o que deve ser priorizado ou dada uma atenção especial.

Esta ferramenta, conforme dito no referencial teórico, serve para diagnosticar que

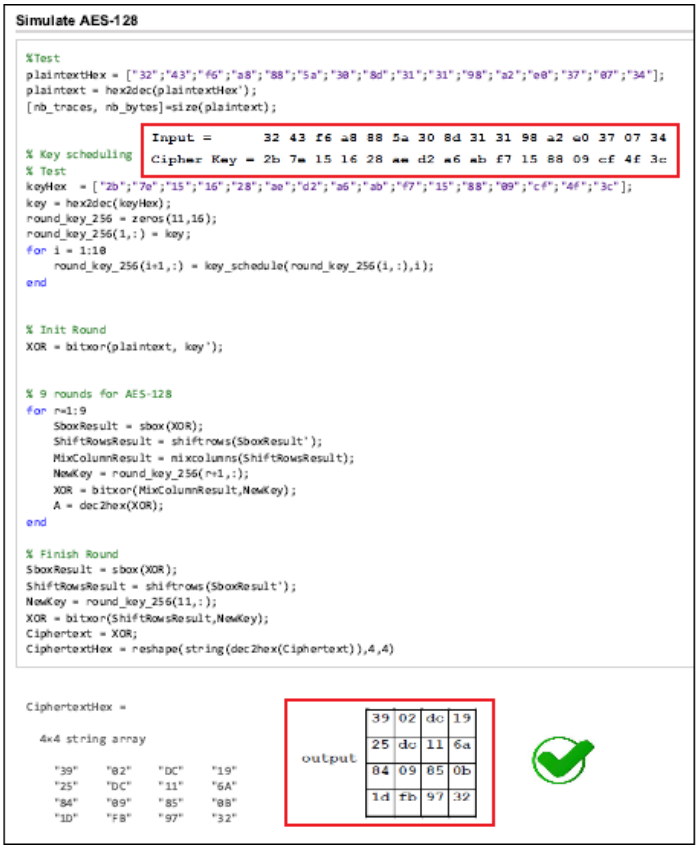

FIGURA 4.6 – Integração dos módulos - Adaptado (AES..., 2019)

atividades possuem riscos maiores, maior severidadade e grau de detecção com o fito de propor ações por parte da equipe de gerenciamento de riscos.

Segundo esse resultado a maior atenção e alocação de recursos devem ser dada no módulo de integração onde, por exemplo, poderia ser aplicado uma equipe de desenvolvimento e supervisão, aumento da equipe, maior prazo para testes, dentre outras opções. As sugestões de ações são qualitativas segundo (BALLESTERO-ALVAREZ, 2012).

TABELA 4.1 – Aplicação do FMEA para a Aplicação e Resultados

Processo	1	2	3	4	5	6	7	8	9	10	11			
	Função do submódulo	Modo de Falha Potencial	Efeito Potencial da Falha	Índice de Severidade (Aplica-se somente ao efeito)	Causa provável	Índice de ocorrência	Controle que pode ser implementado	Índice de Detecção	NPR = (Severidade) x (Ocorrência) x (Detecção)	Ações Tomadas	Resultados das ações			
											Severidade	Ocorrência	Detecção	NPR
7	Integração	Se qualquer um dos processos acima falhar, a integração falhará e gerará se não um resultado diferente, uma disposição diferente	A junção de tudo pode ter várias fontes de falhas	6	Falha de disposição ou qualquer falha anterior	6	Gabarito	6	216	Maior alocação de RH. Sugestão de 2 codificadores e 2 revisores. Depois cruza-se os revisores e desenvolvedores.	NIL	NIL	NIL	NIL
2	Expansion Key	Função mais complexa, porém executada uma única vez implica alta severidade e alta probabilidade de ocorrer	Esse processo é composto de subprocessos que podem falhar	5	Integração dos submódulos, rotação, multiplicação de vetor, mudança de linha	6	Gabarito	6	180	Maior anteção e alocação de mais recursos humanos neste processo. Escolha de codificador e revisor	NIL	NIL	NIL	NIL
1	Add round	Essa função é mais simples de ser implementada e é a mais requisitada ao longo do algoritmo	Como ela é a mais requisitada, se falhar tem um impacto grande na integração, mas por ser simples de se implentar, tem severidade baixa	5	escrita errada da fórmula que executa o comando bitwise - XOR	5	Sim. Por meio de um gabarito.	6	150	De mais simples implementação, porém a que mais se repete, teve segundo maior NPR, logo executar uma vez, testar e copiar em todas as instâncias onde essa função é chamada	NIL	NIL	NIL	NIL
3	SBox / Inv_Sbox	Criação das tabelas que serão usadas nas substituições	Erro de digitação. São 256 números hexadecimais	3	Erro de digitação	4	Gabarito	6	72	Cuidado na transcrição dos números	NIL	NIL	NIL	NIL
6	Mix columns	Mistura de colunas. Um pouco mais difícil do que o processo anterior apenas no VHDL por envolver multiplicação de vetores.. Em MATLAB bem fácil	pouco provável	3	multiplicação não bem feita entre vetores	3	Gabarito	6	54	Cuidado nas multiplicações dos números	NIL	NIL	NIL	NIL
4	SubBytes	Trata-se apenas de uma forma de substituição. Se a tabela do processo 3 estiver correta, esse processo também estará.	pouco provável	2	Indexação errada	3	Gabarito	6	36	Cuidado na transcrição dos números	NIL	NIL	NIL	NIL
5	Shift rows	Deslocamento de linhas. Muito fácil de ser feito tanto em MATLAB quanto em VHDL	pouco provável	2	aplicação da fórmula	3	Gabarito	6	36	Cuidado nas multiplicações dos números	NIL	NIL	NIL	NIL

5 CONCLUSÕES

5.1 Conclusão

5.1.1 Quanto ao gerenciamento de Riscos

Ainda não se tem como uma prática corrente publicar aquilo que deu errado. Esse trabalho de testes é demasiado ingrato pois aqui só se aponta os resultados dos testes e não as tentativas e debugs para que os resultados fossem satisfatório e assim a funcionalidade pudesse avançar. As modificações são extenuantes e impossível catalogar e vão desde arranjo correto das saídas que precisam estar indexadas, tipo de representação numérica que diverge bastante de linguagem pra linguagem e, internamente se trabalha bits e as saídas são em decimal, dentre outros problemas.

Isso é feito no momento da codificação pois se espera obter uma saída que permita ser utilizada pelo módulo seguinte naquilo que foi definido como testes de integração.

Se verificarmos os códigos contidos nas figuras 4.2 e C.1 se percebe o quanto a sub-função de subWord precisou ser adequada para que o *loop* requerido pela função maior pudesse funcionar. Isso é teste de software. Ainda é preciso codificar em VHDL, pois aqui estar a se demonstrar a matemática contida na criptografia e como ela se comporta numa linguagem de alto nível.

Num desenvolvimento empresarial, fora do contexto acadêmico, esse projeto contaria com especialistas de várias áreas e realmente o tempo de desenvolvimento seria bem menor.

Mesmo assim, reduzindo o escopo de um desenvolvimento de projeto, se consegue realizar a análise de riscos em projetos envolvendo FPGA, apenas a abstração matemática anterior a codificação na linguagem de descrição, por meio de um método eficaz e funcionais que diminua retrabalho e, principalmente a validação e verificação caminhem juntos.

Ainda existem muitas outras fases depois da codificação em alto nível. A rigor o demonstrado na figura 5.1 abaixo, nem comtempla a codificação em alto nível por que

isso não permite um transformação direta para a linguagem de descrição de hardware VHDL que de fato programa as portas lógicas contidas em uma placa.

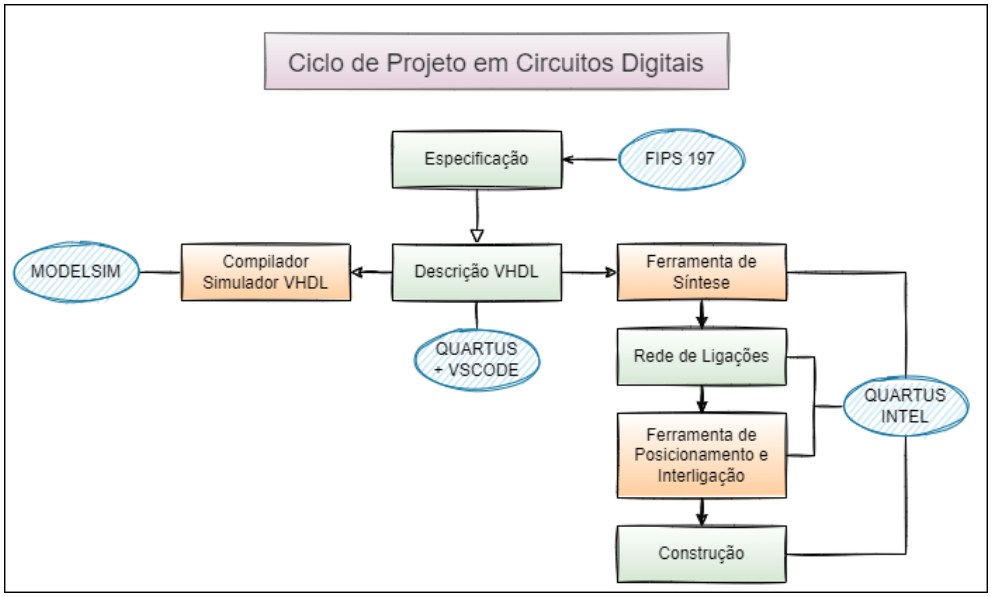

FIGURA 5.1 – Framework de desenvolvimento de Circuitos Digitais - Adaptado (D´AMORE, 2012)

Nesta figura se ver que após uma norma ou publicação que apresenta as especificações, já se dá a descrição na Linguagem VHDL, no entanto aqui se optou por ter uma linguagem de alto nível intermediária para se entender se a integração dos módulos representaria de forma correta a abrangente e complexa matemática ou lógica desenvolvida pelos criptógrafos. A escrita em linguagem VHDL de uma solução complexa como são os algoritmos de criptografia sem uma fase inicial não é uma abordagem inteligente dado o baixo nível, portas lógicas que essa linguagem utiliza. Seria como se tivêssemos que construir um circuito que realiza rotação de posições caracteres tendo somente manipulacão de ponteiros de memórias RAM por exemplo.

Como um risco gerencial potencial dos citados no trabalho (GRUBISIC *et al.*, 2009) convém destacar a correta avaliação homem-hora (HH) no desenvolvimente de aplicativos quando não se tem especialistas na área ou linguagem que se queira empregar. Esse ponto é basilar e de crucial importância e demonstrado na prática desse trabalho, por isso nenhuma referência para afirmar isso. Pois, por ser de cunho acadêmico que presume individualidade, mesmo participando de comunidaddes de desenvolvedores, por vezes os scripts lá contidos raramente funcionam ou estão prontos para uso. Vide repositório contido em (AES..., 2019) e os códigos aqui demonstrados.

Enquanto riscos gerenciais dizem respeito a capacidade de se gerar impacto negativos nos objetivos estratégicos da empresa e metas. Uma das fontes é o gerenciamento correto dos recursos e dentre eles os recursos humanos, (FRAPORTI; BARRETO, 2018).

No tocante aos riscos técnicos, pelo menos os contidos na norma (FLESCH *et al.*, 2020),

se identifica questões como efeitos de funcionalidade e qualidade, mas abrange também confiabilidade (*reliability*), disponibilidade (*availability*), confidencialidade (*confidentiality*), integridade (integrity) e manutenção (maintenance).

5.1.2 Quanto à Análise de Modos de Falhas e seus Efeitos

Quanto a esse quesito vale destacar, conforme pode ser visto na Tabela 2.1 o quanto o esforço deve ser priorizado por meio de alguma métrica. A qualidade como uma área de destaque na Produção possui várias ferramentas de análise qualitativa e quantitativa tal como o FMEA. Ali se evidenciou o que já se esperava com uma maior pontuação de fato à integração onde mais HH são gastos e maiores chances de problemas. Por conterem uma maior quantidade de linhas de códigos e como nem tudo pode ser testado, (JORGENSEN, 2018), é possível que algum teste não tenha coberto uma falha. E esta falha passando, somente na última fase do desenvolvimento é que ela ficará clara e aparecerá.

Os testes de integração foram realizados e demonstrados satisfatórios como visto na figura 4.6 no tópico anterior e os códigos no Apêndice C.

Num desenvolvimento real e sem redução de escopo como foi a proposta de estudo de caso, o gasto despendido de fato foi na integração e por vezes foi necessário retornar aos códigos das funções mais básicas onde já se achava ser um obstáculo vencido.

Esse trabalho mostrou não somente onde a atenção especial deve ser dada como propõe onde se deve realizar alguma ação gerencial para contornar esses possíveis problemas. De fato a Análise de Riscos para desenvolvimento de Softwares produtos ou projetos envolvendo dentre outras tecnologias, FPGA, mostrou-se ser muito adequada e oportuna por meio do FMEA.

Referências

AES 128-256 MATLAB: Repositório MATLAB. 2019. Available at: https://github.com/Anizetho/AES-128_AES-256_MATLAB. Accessed on: 22 OUT. 2022.

BALLESTERO-ALVAREZ, M. E. **Gestão de Qualidade, Produção e Operações**. 3a. ed. Atlas, 2012. ISBN 9788597021523. Available at: https://integrada.minhabiblioteca.com.br//books/9788597021523/. Acesso em: 21 juN. 2023.

BONEH, D.; SHOUP, V. **A graduate course in applied cryptography**. [*S.l.: s.n.*], 2020.

COPELAND, L. **A practitioner's guide to software test design**. [*S.l.*]: Artech House, 2004.

DAEMEN, J.; RIJMEN, V. RIJNDAEL: The advanced encryption standard. **Dr. Dobb's Journal: Software Tools for the Professional Programmer**, Miller Freeman Inc., v. 26, n. 3, p. 137–139, 2001.

D´AMORE, R. **VHDL: Descrição e Síntese de Circuitos Digitais**. 2rd. ed. Rio de Janeiro: LTC, 2012. ISBN 978-85-216-2054-9.

FLESCH, B. F.; TEDESCHI, I.; FIGUEIREDO, R. M. D.; PRADE, L. R.; SILVA, M. R. D. A functional safety methodology based on iec 61508 for critical reliability fpga-based designs. **International Journal of Emerging Technology and Advanced Engineering**, v. 10, n. 7, p. 12 – 19, 2020. Available at: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85099576911doi=10-.46338%2fijetae0720_03partnerID=40md5=4c9b6558e159c46c3fb0ccddff8adbe1.

FLESCH IGOR TODESCHI, R. M. d. F. L. R. P. M. R. d. S. B. F. A functional safety methodology based on iec 61508 for critical reliability fpga-based designs. **Polytechnic School 950 Unisinos ave - São Leopoldo / RS - Brasil**, 2020.

FRAPORTI, S.; BARRETO, J. Gerenciamento de riscos. **Porto Alegre: SAGAH**, 2018.

GRUBISIC, V. V. F. *et al.* Metodologia de gerenciamento integrado de riscos técnicos e gerenciais para o projeto de produtos. Florianópolis, SC, 2009.

HAYEK, A.; BöRCSöK, J. Sram-based fpga design techniques for safety related systems conforming to iec 61508 a survey and analysis. *In*: **2012 2nd International Conference on Advances in Computational Tools for Engineering Applications (ACTEA)**. **Proceedings** [...]. [*S.l.: s.n.*], 2012. p. 319–324.

INTEL, C. **Intel® Quartus® Prime Pro Edition User Guides - Combined**. Santa Clara, CA, EUA, 2023.

JORGENSEN, P. C. **Software testing: a craftsman's approach**. [*S.l.*]: CRC press, 2018.

KARIMI, M. M.; ANZAGIRA, A.; TAYLOR, W.; NELSON, J.; OSAREH, A. Component failure analysis (cfa): A new method for implemented fpga design failure analysis. *In*: IEEE. **SoutheastCon 2018**. **Proceedings** [...]. [*S.l.: s.n.*], 2018. p. 1–8.

LAMERES, B. J. **Quick start guide to VHDL**. [*S.l.*]: Springer, 2019.

OLIVEIRA, E. A. de. **Uma Proposta de Implementação do Algoritmo Rijndael de Criptografia Simétrica para Comunicação Segura por meio de FPGA**. 2023. 96 86f. Dissertation (Mestrado) — Instituto Tecnológico de Aeronautica, São José dos Campos, 2023. Available at: http://www.bdita.bibl.ita.br/http://www.bdita.bibl.ita.br/.

STALLINGS, W. **Criptografia e Segurança de Redes. Princípios e Práticas**. 6. ed. [*S.l.*]: Pearson Prentice Hall, 2015.

TOCCI, R.; WIDMER, N.; MOSS, G. **Digital Systems Principles and Applications**. 12. ed. [*S.l.*]: Pearson Education UK, 2018.

VALENTINE, D. T.; HAHN, B. H. **Essential MATLAB for engineers and scientists**. [*S.l.*]: Academic Press, 2022.

Apêndice A - Tabelas

A.1 Tabela SBox

TABELA A.1 – Tabela de Substituição da Função Sub bytes - (STALLINGS, 2015)

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

A.2 Tabela Inv SBox

TABELA A.2 – Tabela de Substituição da Função Inv Sub bytes - (STALLINGS, 2015)

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
	1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
	2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
	3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
	4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
	5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
	6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
	7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
	8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
	9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
	A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
	B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
	C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
	D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
	E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
	F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

Apêndice B - Pseudo-Códigos

B.1 Encriptação

Código B.1 – Pseudo código AES para Cifragem (DAEMEN; RIJMEN, 2001)

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
    byte state[4,Nb]
    state = in
    AddRoundKey(state, w[0, Nb-1])      // See Sec. 5.1.4
        for round = 1 step 1 to Nr1
            SubBytes(state)             // See Sec. 5.1.1
            ShiftRows(state)            // See Sec. 5.1.2
            MixColumns(state)           // See Sec. 5.1.3
            AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
        end for
    SubBytes(state)
    ShiftRows(state)
    AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
    out = state
end
```

B.2 Decriptação

Código B.2 – Pseudo código AES para decifragem (DAEMEN; RIJMEN, 2001)

```
InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
    byte state[4,Nb]
    state = in
    AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])   // See Sec. 5.1.4
    for round = Nr-1 step -1 downto 1
        InvShiftRows(state)                     // See Sec. 5.3.1
        InvSubBytes(state)                      // See Sec. 5.3.2
        AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
        InvMixColumns(state)                    // See Sec. 5.3.3
    end for
    InvShiftRows(state)
    InvSubBytes(state)
    AddRoundKey(state, w[0, Nb-1])
    out = state
end
```

B.3 Expansão de Chave

Código B.3 – Pseudo código AES para Expansão de Chave (DAEMEN; RIJMEN, 2001)

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
    word temp
    i = 0
    while (i < Nk)
        w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
        i = i+1
    end while
    i = Nk
    while (i < Nb * (Nr+1)]
        temp = w[i-1]
        if (i mod Nk = 0)
            temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
        else if (Nk > 6 and i mod Nk = 4)
            temp = SubWord(temp)
        end if
        w[i] = w[i-Nk] xor temp
        i = i + 1
    end while
end
```

Apêndice C - Testes Completos

C.1 Expansão Completa

Os resultados no MATLAB saem em decimal. Portanto foi adicionada uma linha para transformar em string hexadecimal e o *reshape* para facilitar a correção com a publicação

C.2 Mixtura de Colunas

Para essa função foram criadas duas tabelas multiplicando todos os valores possíveis por 02 e 03 uma vez que, conforme visto em (STALLINGS, 2015), na forma direta a mistura de colunas equivale a realizar uma multiplicação de matrizes e assim o valor que entrar do array ou estado anterior já tem seu equivalente multiplicado por 2 e por 3 cabendo apenas verificar onde e receber o valor por meio das linhas subsequentes. Isso é bem explorado por meio de memórias em FPGA e começa a apontar que estratégia será seguida.

C.3 AES Completo Integração

Código C.1 – AES Completo (DAEMEN; RIJMEN, 2001)

```
1
2  clc
3  clear all
4  %% Simulate AES-128
5  %Test
6  plaintextHex = ["32";"43";"f6";"a8";"88";"5a";"30";"8d";"31";"31";"98";"a2";"e0
       ";"37";"07";"34"];
7  plaintext = hex2dec(plaintextHex');
8  [nb_traces, nb_bytes]=size(plaintext);
9
10
11 % Key scheduling
12 % Test
```

```
keyHex  = ["2b";"7e";"15";"16";"28";"ae";"d2";"a6";"ab";"f7";"15";"88";"09";"cf
    ";"4f";"3c"];
key = hex2dec(keyHex);
round_key_256 = zeros(11,16);
round_key_256(1,:) = key;
for i = 1:10
    round_key_256(i+1,:) = key_schedule(round_key_256(i,:),i);
end


% Init Round
XOR = bitxor(plaintext, key');


% 9 rounds for AES-128
for r=1:9
    SboxResult = sbox(XOR);
    ShiftRowsResult = shiftrows(SboxResult');
    MixColumnResult = mixcolumns(ShiftRowsResult);
    NewKey = round_key_256(r+1,:);
    XOR = bitxor(MixColumnResult,NewKey);
    A = dec2hex(XOR);
end

% Finish Round
SboxResult = sbox(XOR);
ShiftRowsResult = shiftrows(SboxResult');
NewKey = round_key_256(11,:);
XOR = bitxor(ShiftRowsResult,NewKey);
Ciphertext = XOR;
CiphertextHex = reshape(string(dec2hex(Ciphertext)),4,4)
```

```
function [round_key_256] = expansao(keyHex)

    keyHex=["2b"; "7e"; "15"; "16" ;"28";"ae" ;"d2" ;"a6" ;"ab" ;"f7" ;"15"; "88" ;"09" ;"cf" ;"4f" ;"3c"];
    key = hex2dec(keyHex);
    round_key_256 = zeros(11,16);
    round_key_256(1,:) = key;
    rcon = [1, 2, 4, 8, 16, 32, 64, 128, 27, 54];

        for i = 1:10
        round=i;

        w0 = round_key_256(round,1:4);
            w1 = round_key_256(round,5:8);
            w2 = round_key_256(round,9:12);
            w3 = round_key_256(round,13:16);

        round_key(1) = bitxor(bitxor(sbox(w3(2)), w0(1)), rcon(round));
        round_key(2) = bitxor(sbox(w3(3)), w0(2));
        round_key(3) = bitxor(sbox(w3(4)), w0(3));
        round_key(4) = bitxor(sbox(w3(1)), w0(4));

        round_key(5:8)   = bitxor(round_key(1:4), w1);
        round_key(9:12)  = bitxor(round_key(5:8), w2);
        round_key(13:16) = bitxor(round_key(9:12), w3);

        round_key_256(i+1,:) = round_key;
    end
    result=reshape(string(dec2hex(round_key_256)),11,16)
end
```

```
result =

  11×16 string array

  Columns 1 through 9

    "2B"    "7E"    "15"    "16"    "28"    "AE"    "D2"    "A6"    "AB"
    "A0"    "FA"    "FE"    "17"    "88"    "54"    "2C"    "B1"    "23"
    "F2"    "C2"    "95"    "F2"    "7A"    "96"    "B9"    "43"    "59"
    "3D"    "80"    "47"    "7D"    "47"    "16"    "FE"    "3E"    "1E"
    "EF"    "44"    "A5"    "41"    "A8"    "52"    "5B"    "7F"    "B6"
    "D4"    "D1"    "C6"    "F8"    "7C"    "83"    "9D"    "87"    "CA"
    "6D"    "88"    "A3"    "7A"    "11"    "0B"    "3E"    "FD"    "DB"
    "4E"    "54"    "F7"    "0E"    "5F"    "5F"    "C9"    "F3"    "84"
    "EA"    "D2"    "73"    "21"    "B5"    "8D"    "BA"    "D2"    "31"
    "AC"    "77"    "66"    "F3"    "19"    "FA"    "DC"    "21"    "28"
    "D0"    "14"    "F9"    "A8"    "C9"    "EE"    "25"    "89"    "E1"

  Columns 10 through 16

    "F7"    "15"    "88"    "09"    "CF"    "4F"    "3C"
    "A3"    "39"    "39"    "2A"    "6C"    "76"    "05"
    "35"    "80"    "7A"    "73"    "59"    "F6"    "7F"
    "23"    "7E"    "44"    "6D"    "7A"    "88"    "3B"
    "71"    "25"    "3B"    "DB"    "0B"    "AD"    "00"
    "F2"    "8B"    "8C"    "11"    "F9"    "15"    "BC"
    "F9"    "86"    "41"    "CA"    "00"    "93"    "FD"
    "A6"    "4F"    "82"    "4E"    "A6"    "DC"    "4F"
    "28"    "F5"    "60"    "7F"    "8D"    "29"    "2F"
    "D1"    "29"    "41"    "57"    "5C"    "00"    "6E"
    "3F"    "0C"    "C8"    "B6"    "63"    "0C"    "A6"
```

FIGURA C.1 – Código e Resultado Expansão Completa - Autor

```
function [state_out] = mixcolumns(state_in)

    state_in= ["D4" "BF" "5D" "30" "E0" "B4" "52" "AE" "B8" "41" "11" "F1" "1E" "27" "98" "E5"];
    state_in=hex2dec(state_in);

        gmul2_hex = [
        '00'; '02'; '04'; '06'; '08'; '0a'; '0c'; '0e'; '10'; '12'; '14'; '16'; '18'; '1a'; '1c'; '1e';
        '20'; '22'; '24'; '26'; '28'; '2a'; '2c'; '2e'; '30'; '32'; '34'; '36'; '38'; '3a'; '3c'; '3e';
        '40'; '42'; '44'; '46'; '48'; '4a'; '4c'; '4e'; '50'; '52'; '54'; '56'; '58'; '5a'; '5c'; '5e';
        '60'; '62'; '64'; '66'; '68'; '6a'; '6c'; '6e'; '70'; '72'; '74'; '76'; '78'; '7a'; '7c'; '7e';
        '80'; '82'; '84'; '86'; '88'; '8a'; '8c'; '8e'; '90'; '92'; '94'; '96'; '98'; '9a'; '9c'; '9e';
        'a0'; 'a2'; 'a4'; 'a6'; 'a8'; 'aa'; 'ac'; 'ae'; 'b0'; 'b2'; 'b4'; 'b6'; 'b8'; 'ba'; 'bc'; 'be';
        'c0'; 'c2'; 'c4'; 'c6'; 'c8'; 'ca'; 'cc'; 'ce'; 'd0'; 'd2'; 'd4'; 'd6'; 'd8'; 'da'; 'dc'; 'de';
        'e0'; 'e2'; 'e4'; 'e6'; 'e8'; 'ea'; 'ec'; 'ee'; 'f0'; 'f2'; 'f4'; 'f6'; 'f8'; 'fa'; 'fc'; 'fe';
        '1b'; '19'; '1f'; '1d'; '13'; '11'; '17'; '15'; '0b'; '09'; '0f'; '0d'; '03'; '01'; '07'; '05';
        '3b'; '39'; '3f'; '3d'; '33'; '31'; '37'; '35'; '2b'; '29'; '2f'; '2d'; '23'; '21'; '27'; '25';
        '5b'; '59'; '5f'; '5d'; '53'; '51'; '57'; '55'; '4b'; '49'; '4f'; '4d'; '43'; '41'; '47'; '45';
        '7b'; '79'; '7f'; '7d'; '73'; '71'; '77'; '75'; '6b'; '69'; '6f'; '6d'; '63'; '61'; '67'; '65';
        '9b'; '99'; '9f'; '9d'; '93'; '91'; '97'; '95'; '8b'; '89'; '8f'; '8d'; '83'; '81'; '87'; '85';
        'bb'; 'b9'; 'bf'; 'bd'; 'b3'; 'b1'; 'b7'; 'b5'; 'ab'; 'a9'; 'af'; 'ad'; 'a3'; 'a1'; 'a7'; 'a5';
        'db'; 'd9'; 'df'; 'dd'; 'd3'; 'd1'; 'd7'; 'd5'; 'cb'; 'c9'; 'cf'; 'cd'; 'c3'; 'c1'; 'c7'; 'c5';
        'fb'; 'f9'; 'ff'; 'fd'; 'f3'; 'f1'; 'f7'; 'f5'; 'eb'; 'e9'; 'ef'; 'ed'; 'e3'; 'e1'; 'e7'; 'e5'
    ];
```

```
        gmul3_hex = [
        '00'; '03'; '06'; '05'; '0c'; '0f'; '0a'; '09'; '18'; '1b'; '1e'; '1d'; '14'; '17'; '12'; '11';
        '30'; '33'; '36'; '35'; '3c'; '3f'; '3a'; '39'; '28'; '2b'; '2e'; '2d'; '24'; '27'; '22'; '21';
        '60'; '63'; '66'; '65'; '6c'; '6f'; '6a'; '69'; '78'; '7b'; '7e'; '7d'; '74'; '77'; '72'; '71';
        '50'; '53'; '56'; '55'; '5c'; '5f'; '5a'; '59'; '48'; '4b'; '4e'; '4d'; '44'; '47'; '42'; '41';
        'c0'; 'c3'; 'c6'; 'c5'; 'cc'; 'cf'; 'ca'; 'c9'; 'd8'; 'db'; 'de'; 'dd'; 'd4'; 'd7'; 'd2'; 'd1';
        'f0'; 'f3'; 'f6'; 'f5'; 'fc'; 'ff'; 'fa'; 'f9'; 'e8'; 'eb'; 'ee'; 'ed'; 'e4'; 'e7'; 'e2'; 'e1';
        'a0'; 'a3'; 'a6'; 'a5'; 'ac'; 'af'; 'aa'; 'a9'; 'b8'; 'bb'; 'be'; 'bd'; 'b4'; 'b7'; 'b2'; 'b1';
        '90'; '93'; '96'; '95'; '9c'; '9f'; '9a'; '99'; '88'; '8b'; '8e'; '8d'; '84'; '87'; '82'; '81';
        '9b'; '98'; '9d'; '9e'; '97'; '94'; '91'; '92'; '83'; '80'; '85'; '86'; '8f'; '8c'; '89'; '8a';
        'ab'; 'a8'; 'ad'; 'ae'; 'a7'; 'a4'; 'a1'; 'a2'; 'b3'; 'b0'; 'b5'; 'b6'; 'bf'; 'bc'; 'b9'; 'ba';
        'fb'; 'f8'; 'fd'; 'fe'; 'f7'; 'f4'; 'f1'; 'f2'; 'e3'; 'e0'; 'e5'; 'e6'; 'ef'; 'ec'; 'e9'; 'ea';
        'cb'; 'c8'; 'cd'; 'ce'; 'c7'; 'c4'; 'c1'; 'c2'; 'd3'; 'd0'; 'd5'; 'd6'; 'df'; 'dc'; 'd9'; 'da';
        '5b'; '58'; '5d'; '5e'; '57'; '54'; '51'; '52'; '43'; '40'; '45'; '46'; '4f'; '4c'; '49'; '4a';
        '6b'; '68'; '6d'; '6e'; '67'; '64'; '61'; '62'; '73'; '70'; '75'; '76'; '7f'; '7c'; '79'; '7a';
        '3b'; '38'; '3d'; '3e'; '37'; '34'; '31'; '32'; '23'; '20'; '25'; '26'; '2f'; '2c'; '29'; '2a';
        '0b'; '08'; '0d'; '0e'; '07'; '04'; '01'; '02'; '13'; '10'; '15'; '16'; '1f'; '1c'; '19'; '1a'
    ];
    gmul2 = hex2dec(gmul2_hex);
    gmul3 = hex2dec(gmul3_hex);

    size_state = size(state_in);
    nb_states = size_state(1);
        state_out = zeros(nb_states,16);
```

```
    for i = 0 : 3
                c1 = state_in(:,i*4+1);
                c2 = state_in(:,i*4+2);
                c3 = state_in(:,i*4+3);
                c4 = state_in(:,i*4+4);
                state_out(:,i*4+1) = bitxor(bitxor(gmul2(c1+1), gmul3(c2+1)),bitxor(c3, c4));
                state_out(:,i*4+2) = bitxor(bitxor(c1, gmul2(c2+1)), bitxor(gmul3(c3+1), c4));
                state_out(:,i*4+3) = bitxor(bitxor(c1, c2), bitxor(gmul2(c3+1), gmul3(c4+1)));
                state_out(:,i*4+4) = bitxor(bitxor(gmul3(c1+1), c2), bitxor(c3, gmul2(c4+1)));
    end
    state_out=reshape(string(dec2hex(state_out)),4,4);
end
```

```
ans =

  4×4 string array

    "04"    "E0"    "48"    "28"
    "66"    "CB"    "F8"    "06"
    "81"    "19"    "D3"    "26"
    "E5"    "9A"    "7A"    "4C"
```

d4	e0	b8	1e
bf	b4	41	27
5d	52	11	98
30	ae	f1	e5

04	e0	48	28
66	cb	f8	06
81	19	d3	26
e5	9a	7a	4c

FIGURA C.2 – Operação de Mixtura de coluna