Relatório de Desenvolvimento de Dispositivo de Encriptação ChaCha20 utilizando NIOS II

1st Antônio Martins Moreira Neto
Departamento de Engenharia Elétrica e de Computação
UFBA - Universidade Federal da Bahia
Salvador, Brasil
antoniomoreira@ufba.br

2nd Bruno Santos Junqueira
Departamento de Engenharia Elétrica e de Computação
UFBA - Universidade Federal da Bahia
Salvador, Brasil
bruno.junqueira@ufba.br

3rd Fredson Menezes Sumi Barreto
Departamento de Engenharia Elétrica e de Computação
UFBA - Universidade Federal da Bahia
Salvador, Brasil
fredson.sumi@ufba.br

4th José Pedreira de Macedo Neto
Departamento de Engenharia Elétrica e de Computação
UFBA - Universidade Federal da Bahia
Salvador, Brasil
jose.pedreira@ufba.br

5th Pedro Hugo Passos da Silva Carlos
Departamento de Engenharia Elétrica e de Computação
UFBA - Universidade Federal da Bahia
Salvador, Brasil
pedro.hugo@ufba.br

***Resumo*—Neste relatório, apresentamos a implementação do algoritmo de criptografia ChaCha20 utilizando a plataforma FPGA, com ênfase na integração com o processador NIOS II. O trabalho foi desenvolvido em três etapas principais. A primeira etapa consistiu na implementação do algoritmo em linguagem C, proporcionando uma base inicial em software para validar o funcionamento do ChaCha20. A segunda etapa envolveu a execução do código no processador NIOS II, um CPU soft-core integrado à FPGA, permitindo a simulação e análise do desempenho do algoritmo em um ambiente de hardware. Na terceira e última etapa, focamos na otimização do código C original, identificando gargalos de desempenho e implementando melhorias em Verilog para maximizar a eficiência através do uso do FPGA. Este relatório documenta cada uma dessas etapas, discutindo os desafios encontrados, as soluções adotadas e os resultados obtidos, com o objetivo de demonstrar a viabilidade e as vantagens da utilização de FPGA para a execução otimizada de algoritmos de criptografia.**

***Index Terms*—Nios II, ChaCha20, Segurança de Dados, FPGA**

I. Introdução

O co-projeto de hardware/software refere-se ao desenvolvimento conjunto de componentes de hardware e software, com o objetivo de atender aos requisitos funcionais de um sistema dentro de um prazo reduzido. Esse processo envolve uma ampla variedade de componentes que podem ser aplicados em diferentes formas para proporcionar soluções alternativas e integradas, garantindo o melhor desempenho e eficiência.

Entre as tecnologias utilizadas nesse contexto, destaca-se o uso de FPGAs (Field-Programmable Gate Arrays), que são dispositivos reprogramáveis, amplamente usados em projetos que demandam flexibilidade e alto desempenho. Esses dispositivos permitem a implementação de funções específicas, desde o desenvolvimento de processadores customizados até a execução de algoritmos complexos em áreas como telecomunicações, controle industrial e processamento de imagem.

O avanço dessa tecnologia tem possibilitado sua aplicação em sistemas críticos e de alta demanda, como na integração de componentes para HPC (High-Performance Computing) e na colaboração com GPUs, como observado em grandes plataformas de computação em nuvem.

A. Objetivos

Este trabalho tem como foco o desenvolvimento e a análise de uma solução de criptografia baseada na cifra ChaCha20, com implementações em software e hardware. Para alcançar esse propósito, foram definidos os seguintes objetivos:

- **Implementação em Software:** Desenvolver uma aplicação de criptografia utilizando a cifra ChaCha20, implementada exclusivamente em software, utilizando a linguagem C e o processador Nios II. Essa fase do projeto buscará identificar e avaliar as funções críticas do algoritmo e seu impacto no desempenho geral do sistema.
- **Implementação em Hardware/Software:** Desenvolver uma solução híbrida que combine hardware e software, utilizando módulos IP em Verilog e o processador Nios II, visando otimizar o desempenho do sistema. A solução incluirá a identificação das partes críticas do algoritmo e sua migração para hardware, avaliando o impacto em termos de tempo de execução e consumo de recursos.

- **Análise e Comparação de Desempenho:** Realizar testes e comparações entre as duas implementações (software puro e hardware/software) para determinar qual solução oferece o melhor equilíbrio entre desempenho, consumo de recursos e eficiência para a cifra ChaCha20.
- **Validação e Integração no Ambiente Platform Designer (Qsys):** Validar a implementação do sistema híbrido no ambiente de desenvolvimento Platform Designer, verificando a integração e a funcionalidade dos módulos IP implementados em hardware.
- **Exploração e Otimização do Design:** Estudar a viabilidade de otimização do design, com o objetivo de reduzir a latência e a sobrecarga no sistema, por meio do uso eficiente de aceleradores de hardware.

II. Revisão Bibliográfica

A. ChaCha20

ChaCha20 é um algoritmo de cifra de fluxo desenvolvido por Daniel J. Bernstein, baseado na cifra Salsa20, mas com melhorias em termos de segurança e eficiência. É amplamente utilizado em protocolos modernos de segurança, como o TLS (Transport Layer Security).

1) Estrutura da Matriz: A cifra ChaCha20 utiliza uma matriz de 4x4 que combina uma chave secreta de 256 bits, um nonce de 64 bits e um contador de 64 bits. A matriz inicial, chamada de *state*, é formada da seguinte maneira:

$$\text{state} = \begin{bmatrix} \text{constant} & \text{constant} & \text{constant} & \text{constant} \\ \text{key} & \text{key} & \text{key} & \text{key} \\ \text{key} & \text{key} & \text{key} & \text{key} \\ \text{counter} & \text{nonce} & \text{nonce} & \text{nonce} \end{bmatrix}$$

a) Descrição da Matriz:

- **Constant**: São quatro constantes de 32 bits usadas para identificar a cifra ChaCha20. Elas são derivadas da string ASCII ”expand 32-byte k”.
- **Key**: A chave de 256 bits é dividida em oito blocos de 32 bits que preenchem as segunda e terceira linhas da matriz.
- **Counter**: Um contador de 32 bits, geralmente iniciado em zero, que permite a geração de diferentes blocos de fluxo de chave para mensagens longas.
- **Nonce**: Um valor de 96 bits que deve ser único para cada execução do algoritmo com a mesma chave, garantindo que a saída da cifra seja única.

2) Tabela de Constantes e Valores: A tabela a seguir resume os valores que compõem a matriz inicial:

Elemento	Valor (hexadecimal)	Descrição
`constant`	0x61707865	Derivado de ”expand 32-byte k”
`constant`	0x3320646e	Derivado de ”expand 32-byte k”
`constant`	0x79622d32	Derivado de ”expand 32-byte k”
`constant`	0x6b206574	Derivado de ”expand 32-byte k”
`key`	Chave de 256 bits	Oito palavras de 32 bits da chave
`counter`	Valor variável	Contador de 32 bits
`nonce`	Valor variável	Nonce de 96 bits

Tabela I
Elementos da Matriz Inicial de ChaCha20

Nota: Para a implementação do algoritmo ChaCha20, utilizamos como base o RFC 8439, que define parâmetros específicos para o algoritmo, incluindo um nonce de 96 bits e um contador de 32 bits.

B. Platform Designer (Qsys)

O Platform Designer, anteriormente conhecido como Qsys, é uma ferramenta de design de sistemas da Intel (Altera) usada para construir sistemas baseados em FPGA. Ele facilita a integração de componentes de hardware e software por meio de uma interface gráfica, permitindo a configuração rápida e eficiente de sistemas customizados. O Platform Designer é amplamente utilizado no desenvolvimento de sistemas embarcados, possibilitando a conexão de processadores, memórias, interfaces de comunicação e outros periféricos de forma modular e escalável.

C. Nios II

O Nios II é um processador soft-core da Intel (Altera) implementado em FPGAs. Ele é altamente configurável, permitindo que desenvolvedores ajustem suas características conforme as necessidades do projeto. Devido à sua flexibilidade, o Nios II é amplamente utilizado em aplicações embarcadas onde é necessário um equilíbrio entre desempenho e custo. A integração do Nios II com o Platform Designer facilita a criação de sistemas customizados, permitindo que diferentes blocos de hardware e software sejam combinados de maneira eficiente.

III. Metodologia

O desenvolvimento foi realizado em duas partes: a implementação em software puro e uma abordagem mista em hardware/software utilizando uma placa FPGA Altera DE2-115, da família Cyclone IV E. O desenvolvimento em software na linguagem C para a implementação da cifra ChaCha20 foi inicialmente realizado para testar o algoritmo com o objetivo de garantir sua funcionalidade. Após o teste em C, utilizando como referência a RFC 8439, permitiu uma análise precisa dos resultados e a validação da funcionalidade do algoritmo. A implementação mista em hardware/software utilizou Verilog para a modelagem da cifra ao nível de circuito e o processador Nios II, integrado através do Platform Designer do Quartus. Ao final do processo de implementação o ModelSim foi utilizado com o objetivo de validar o sistema de criptografia proposto.

A. *Ferramentas Utilizadas*

- *Quartus Prime* 18.1;
- Placa FPGA Altera DE2-115;
- Visual Studio Code;
- Repositório GitHub.

IV. Desenvolvimento

A. *Implementação em C*

O desenvolvimento do código em C foi baseado no RFC 8439, intitulado ”ChaCha20 and Poly1305 for IETF Protocols”. Esta implementação foca na geração de uma chave, configuração do estado inicial, execução de operações de mistura e aplicação de XOR entre a mensagem e um fluxo de chave para realizar a criptografia e descriptografia. Foram criadas funções no código para cada etapa desses processos, incluindo **chacha20_block**, **inner_block**, **quarter_round**, **left_rotate**, **generate_nonce** e **xor_message**.

A função **chacha20_block** configura o estado inicial do ChaCha20, incorporando a constante, a chave, o contador e o nonce. Em seguida, executa 20 rodadas do algoritmo (10 rodadas de **inner_block**). O estado resultante é combinado com o estado inicial para gerar um fluxo de chave, que será utilizado na criptografia ou descriptografia dos dados.

A função **inner_block** aplica oito operações de ”quarter round”em diferentes combinações de elementos do vetor de estado, repetindo esse processo 10 vezes para proporcionar maior difusão.

A função **quarter_round** envolve uma série de adições, XORs e rotações de bits (left rotate) para misturar quatro elementos específicos do estado. A operação ”quarter round”é repetida várias vezes para intensificar a difusão dos bits e garantir a segurança da criptografia.

A função **left_rotate** move os bits de um inteiro para a esquerda, reintroduzindo os bits que saem do lado mais significativo (MSB) na posição menos significativa (LSB). Essa operação é crucial para o processo de mistura de bits, aumentando a segurança da criptografia.

A função **generate_nonce** cria um nonce baseado em uma semente derivada da chave. Esse nonce assegura que cada operação de criptografia utilize um fluxo de chave único, mesmo se a mesma chave e contador forem reutilizados.

A função **xor_message** é responsável por criptografar uma mensagem usando uma operação de XOR entre os dados da mensagem original e um fluxo de chaves (key stream) gerado pelo algoritmo ChaCha20.

Assim, o software funciona criptografando inicialmente uma mensagem, que é, em seguida, descriptografada utilizando a função xor_message. A mensagem original, a criptografada e a descriptografada são impressas para comparação. A criptografia é realizada gerando um fluxo de chave ChaCha20 e aplicando a operação XOR entre o fluxo e a mensagem.

B. *Profile*

A partir do código em C implementado foi realizado a análise de desempenho do programa através de uma ferramenta de uma análise dinâmica processamento por cada chamada de função, onde obtivemos os seguintes resultados a seguir:

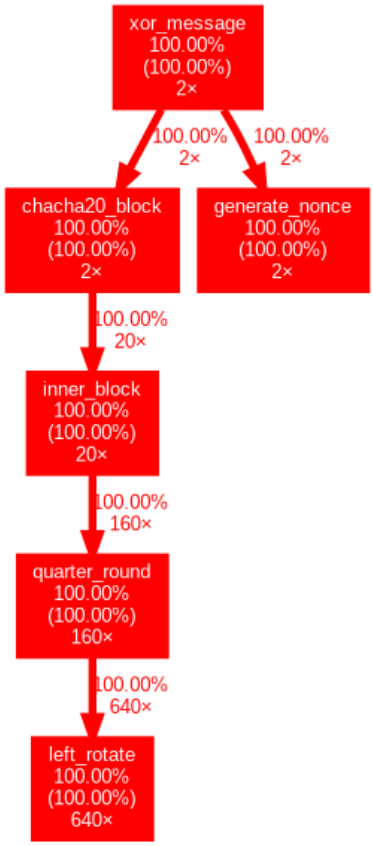

Figura 1. Profile Code

Utilizando Terminal via *Gprof*, obtivemos o seguinte desempenho por cada chamada de função:

Porcentagem	(Desempenho)	Chamada	Call
50.28	0.91	640	left_rotate
49.72	0.90	160	quarter_ round
0	0	20	inner_block
0	0	2	chacha20_block
0	0	2	generate_nonce
0	0	2	xor_message

Tabela II
Elementos das chamadas ChaCha20

Observação:: Através desses dados obtemos que onde ocorre um maior tempo de processamento se dá entre a função **left rotate** com 640 chamadas de sistemas e a função **quarter round** com 160 chamadas de sistemas.

C. *Implementação Verilog*

O módulo **ChaCha** é um componente de hardware escrito em Verilog que gerencia a cifra de fluxo ChaCha. Ele permite a leitura e escrita de dados por meio de uma interface com endereços e dados de 32 bits. O módulo mantém registradores internos para a chave, número de rounds e dados de entrada.

Ele também se comunica com um núcleo de criptografia **ChaCha_core** que realiza a operação de cifra. Além disso, o módulo contém lógica para decodificação de endereços e atualização dos registradores conforme as entradas de controle e dados são fornecidas.

O núcleo ChaCha_core é responsável pela geração do fluxo de chaves criptográficas usando uma versão simplificada do algoritmo ChaCha.

Fluxo da Logica Interna do modulo ChaCha_core:

1) O estado inicial é configurado a partir dos valores da chave, contador e vetor de inicialização.
2) O núcleo processa um bloco de dados em múltiplas rodadas, aplicando a função de "quarterrounds"do ChaCha.
3) Após as rodadas, o estado final é combinado com o bloco de dados de entrada para produzir a saída criptografada.

Função do módulo chacha_qr:

O código implementa a função "quarterround" (QR) do cifrador de fluxo ChaCha em Verilog 2001. Este módulo é projetado para ser usado em versões do cifrador com 1, 2, 4 ou até 8 funções QR paralelas.

D. Finite State Machine (FSM) do ChaCha20

A FSM (Finite State Machine) do algoritmo ChaCha20 controla o fluxo do algoritmo, passando por diferentes estados para realizar a criptografia. Aqui estão os estados e os sinais associados:

1) Estados e Sinais:

- **CTRL_IDLE**:
  - **Transição para**: `CTRL_INIT` quando `init` é ativado.
  - **Sinais**:
    - `block_ctr_set = 1`: Configura o contador de blocos.
    - `ready_new = 0`: Define o sinal de pronto como 0.
    - `ready_we = 1`: Habilita a escrita do sinal de pronto.
- **CTRL_INIT**:
  - **Transição para**: `CTRL_ROUNDS`.
  - **Sinais**:
    - `init_state = 1`: Inicializa o estado.
    - `qr_ctr_rst = 1`: Reseta o contador de quarter rounds.
    - `dr_ctr_rst = 1`: Reseta o contador de double rounds.
- **CTRL_ROUNDS**:
  - **Transição para**: `CTRL_FINALIZE` quando `qr_ctr_reg == 1` e `dr_ctr_reg == 19`.
  - **Sinais**:
    - `update_state = 1`: Atualiza o estado.
    - `qr_ctr_inc = 1`: Incrementa o contador de quarter rounds.
- **CTRL_FINALIZE**:
  - **Transição para**: `CTRL_DONE`.
  - **Sinais**:
    - `ready_new = 1`: Define o sinal de pronto como 1.
    - `ready_we = 1`: Habilita a escrita do sinal de pronto.
    - `update_output = 1`: Atualiza a saída.
    - `data_out_valid_new = 1`: Define o sinal de validade dos dados de saída como 1.
    - `data_out_valid_we = 1`: Habilita a escrita do sinal de validade dos dados de saída.
- **CTRL_DONE**:
  - **Transição para**: `CTRL_INIT` quando `init` ou `next_block` são ativados.
  - **Sinais**:
    - `ready_new = 0`: Define o sinal de pronto como 0.
    - `ready_we = 1`: Habilita a escrita do sinal de pronto.
    - `data_out_valid_new = 0`: Define o sinal de validade dos dados de saída como 0.
    - `data_out_valid_we = 1`: Habilita a escrita do sinal de validade dos dados de saída.
    - `block_ctr_set = 1`: Configura o contador de blocos (quando `init`).
    - `block_ctr_inc = 1`: Incrementa o contador de blocos (quando `next_block`).

2) Explicação da FSM:

- **CTRL_IDLE**: Este é o estado inicial onde a FSM aguarda a inicialização (`init`). Quando `init` é ativado, a FSM configura o contador de blocos e se prepara para iniciar o processo de criptografia, movendo-se para o estado `CTRL_INIT`.
- **CTRL_INIT**: Neste estado, a FSM inicializa os contadores de estado e de rounds, preparando-se para executar os rounds de criptografia. Em seguida, transita para o estado `CTRL_ROUNDS`.
- **CTRL_ROUNDS**: Aqui, a FSM executa os rounds de criptografia. Ela incrementa o contador de quarter rounds (`qr_ctr_inc`). Quando o contador de quarter rounds atinge `1` e o contador de double rounds atinge o valor especificado `19`, a FSM transita para o estado `CTRL_FINALIZE`.
- **CTRL_FINALIZE**: Neste estado, a FSM finaliza o processo de criptografia, atualizando a saída e sinalizando que os dados de saída são válidos. Em seguida, transita para o estado `CTRL_DONE`.
- **CTRL_DONE**: Este é o estado final onde a FSM aguarda uma nova inicialização (`init`) ou a solicitação de um próximo bloco (`next_block`). Dependendo do sinal recebido, a FSM reinicia o processo de criptografia, voltando ao estado `CTRL_INIT`.

Essa FSM é essencial para controlar o fluxo do algoritmo ChaCha20, garantindo que cada etapa do processo de crip-

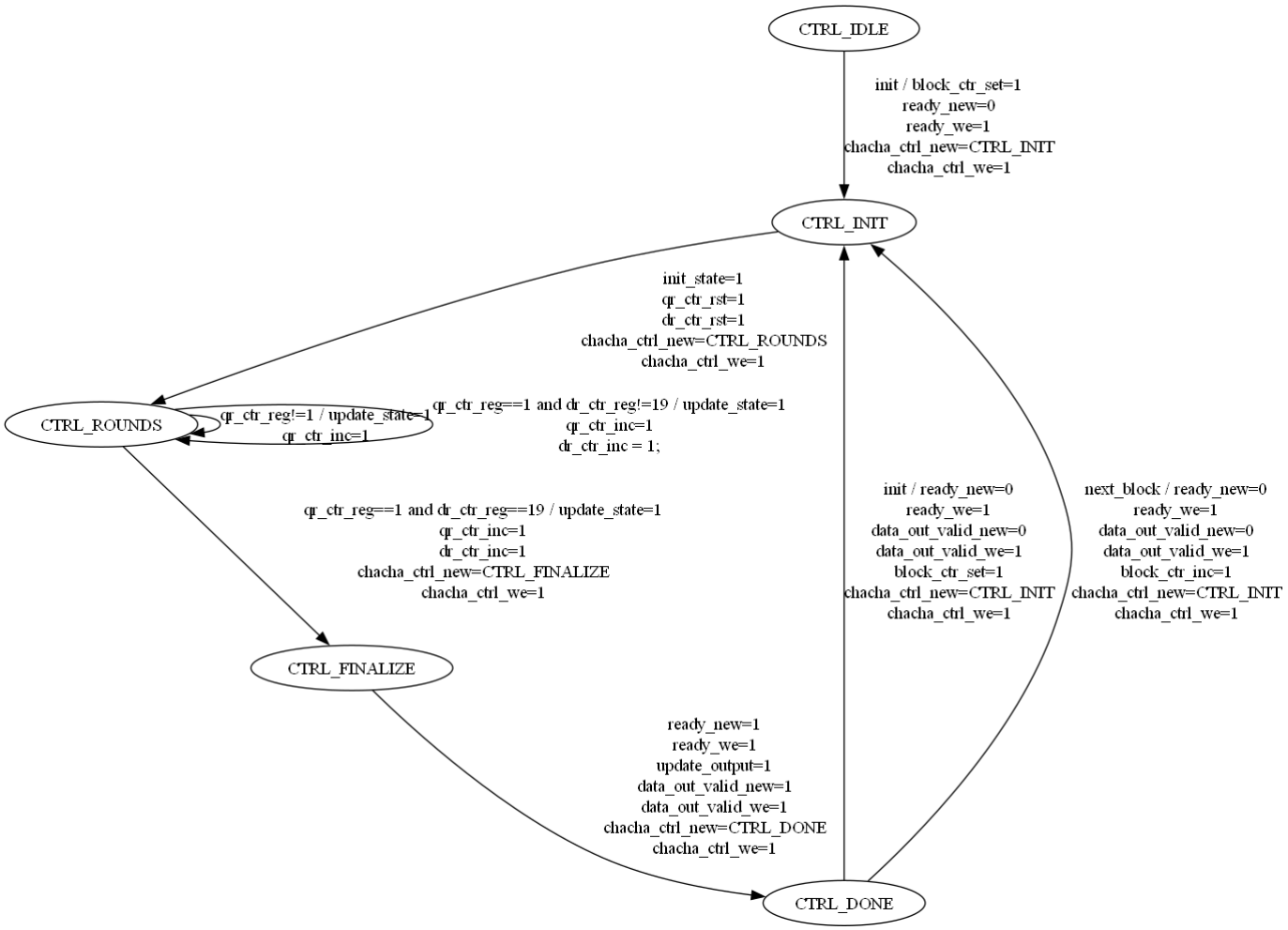

Figura 2. Diagrama da FSM do ChaCha20

tografia seja executada na ordem correta e que os sinais apropriados sejam gerados em cada estado.

E. Implementação Mista

A implementação mista deste projeto envolve a comunicação entre o software e o componente de hardware implementado em Verilog, especificamente no módulo `chacha20.v`. Essa comunicação é realizada através de endereços pré-determinados para a escrita e leitura de dados essenciais ao funcionamento do algoritmo ChaCha20.

Os endereços utilizados são definidos conforme mostrado na Tabela III:

Nome	Endereço	Descrição
ADDR_ROUNDS	0xb	Endereço de rounds
ADDR_DATA_IN	0x40 - 0x4F	Endereços de data_in
ADDR_KEY	0x17 - 0x1E	Endereços da chave de 256 bits
ADDR_DATA_OUT	0x80 - 0x8F	Endereços de data_out
ADDR_STATUS	0x9	Endereço que retorna o estado atual
ADDR_IV	0x20 - 0x22	Endereços do nonce
ADDR_CTRL	0x08	Endereço de controle (next_block, init)

Tabela III
TABELA DE ENDEREÇOS UTILIZADOS NA IMPLEMENTAÇÃO MISTA

V. RESULTADOS

Nesta seção, são apresentados os resultados dos testes simulados realizados para validar a correta integração entre o hardware implementado no FPGA e o software executado utilizando o processador Nios II. A simulação foi fundamental para verificar o comportamento do sistema como um todo e identificar possíveis discrepâncias entre a implementação teórica e a prática. Desse modo, foi possível verificar o funcionamento correto dos componentes individuais e a interação

entre eles. Além disso, a simulação permitiu a identificação de erros de projeto antes da implementação física no FPGA.

A. Testes Software

Foram realizados testes de software utilizando o Visual Studio Code e o Nios II, sendo o primeiro para execução fora do FPGA e o segundo para execução dentro do FPGA. No primeiro teste, foi usada uma mensagem curta de 33 caracteres, “ENGG57 - Laboratório Integrado IV”, na qual tanto a encriptação quanto a decriptação funcionaram corretamente, tanto no Visual Studio Code (Figura 3) quanto no Nios II (Figura 4). No entanto, no Nios II, não foi possível visualizar a mensagem encriptada por completo, pois um dos caracteres gerados foi “
0”, que é um caractere de término de string.

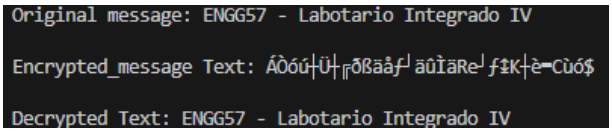

Figura 3. Mensagem curta no Visual Studio Code

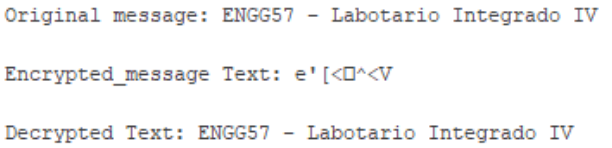

Figura 4. Mensagem curta no Nios II

No segundo teste, foi utilizada uma mensagem mais longa, com 237 caracteres: “Concepcao integrada de Hardware e Software. Integracao dos conceitos das disciplinas: Redes de Computadores e Projetos de Circuitos Integrados Digitais. Utilizacao de tecnicas de expressao grafica para projeto de Sistemas Computacionais.” Neste caso, apenas no Visual Studio Code, (Figura 5), o processo funcionou corretamente. Ao utilizar o Nios II (Figura 6), tanto a encriptação quanto a decriptação falharam, resultando em uma mensagem encriptada que não retornou à sua forma original. Em resposta a isso, foram realizadas tentativas de corrigir o problema, como o uso de variáveis estáticas para armazenar as mensagens em vez de variáveis dinâmicas e aumentar a memória.

Após notarmos um erro na função `xor_message`, em que estávamos tratando os dados como uma string ao invés de um array usando as funções da biblioteca `string.h`, corrigimos o problema. Inicialmente, estávamos utilizando a função `strlen()` para determinar o tamanho da string. No entanto, ao fazermos um loop `for` considerando a string inteira, de 64 em 64 bytes, conseguimos consertar o problema e gerar a mensagem encriptada corretamente com o ChaCha20.

B. Testes de Hardware/Software

A ferramenta ModelSim foi utilizada para simular o comportamento do circuito digital projetado, conforme ilustrado

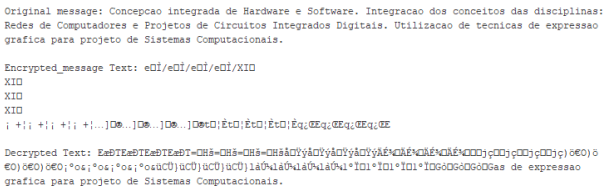

Figura 5. Mensagem longa no Visual Studio Code

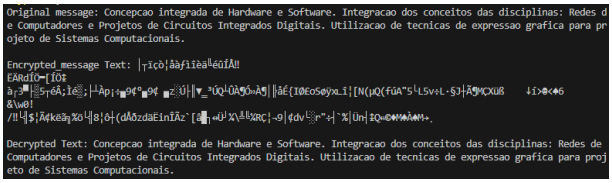

Figura 6. Mensagem longa no Nios II

na Figura 7. Embora a simulação baseada em formas de onda ofereça grande versatilidade na análise de sinais, a verificação de erros tornou-se um desafio para garantir o funcionamento esperado. Em contraste, na simulação por meio de testbench, uma das principais dificuldades encontradas é a criação de estímulos que representem com precisão as condições reais de operação. Se os estímulos aplicados no testbench não simularem adequadamente essas condições, problemas de sincronização podem ser mascarados.

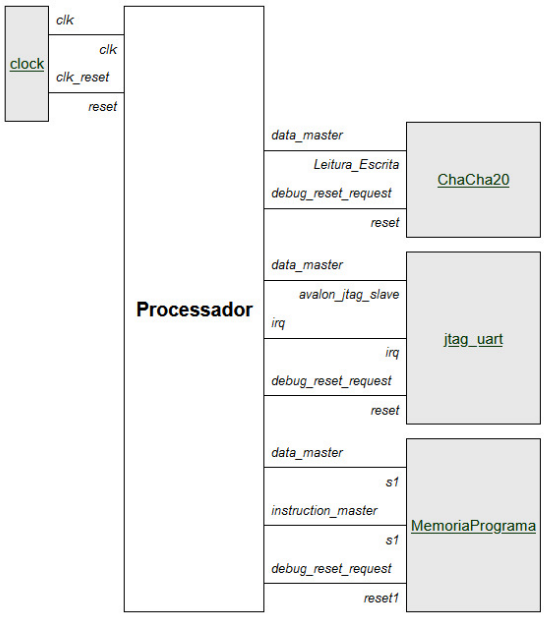

Figura 7. Componentes no esquemático do circuito digital

A simulação do componente ChaCha20 durante a utilização do ModelSim, cuja instância é ilustrada na Figura 8, apresentou desafios durante a simulação baseada na entrada de forma de onda. A complexidade do algoritmo de criptografia e a necessidade de sincronizar diversos sinais internos no software dificultaram a depuração e a identificação de erros, resultando

em testes demorados e inconclusivos. Após a implementação em Verilog por meio de testbenches, os testes passaram a ser mais ágeis, levando à conclusão de que a implementação proposta na instância estava correta.

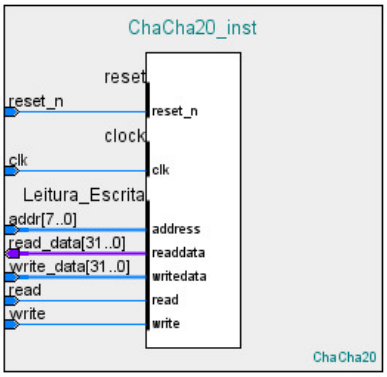

Figura 8. Instância do componente ChaCha20

Ao final da implementação em Verilog, por meio da escrita um arquivo testbench para simulação, foi possível constatar com sucesso o correto funcionamento dos componentes e a interação entre eles. O principal desafio encontrado nessa implementação foi um problema de sincronização enfrentado especificamente com a definição de estímulos. Esse problema foi superado ao utilizar os estímulos do master do Platform Designer. Os testes de validação realizados sem o uso (com a linguagem C) e com o uso do ModelSim (com a linguagem Verilog), apresentados respectivamente por meio das figuras 9 e 10, confirmaram que o circuito atende aos requisitos especificados.

```
Encrypted Message:
0x2234363
0xbf0ca967
0x4e15f409
0x8aab3710
0x5e0ad9d5
0x36ba5d9b
0xd792c4d9
0x7b9f31eb
0xa32fd07e
0xd8714247
0x29967bd0
0xd7b56535
0x42d1a0d
0x36479bfa
0xaf60c763
0x920fe6b3
```

Figura 9. Resultado dos testes no Visual Studio Code

```
data_out: 0x2234363,
data_out: 0xbf0ca967,
data_out: 0x4e15f409,
data_out: 0x8aab3710,
data_out: 0x5e0ad9d5,
data_out: 0x36ba5d9b,
data_out: 0xd792c4d9,
data_out: 0x7b9f31eb,
data_out: 0xa32fd07e,
data_out: 0xd8714247,
data_out: 0x29967bd0,
data_out: 0xd7b56535,
data_out: 0x42d1a0d,
data_out: 0x36479bfa,
data_out: 0xaf60c763,
data_out: 0x920fe6b3,
```

Figura 10. Resultado dos testes no ModelSim

VI. Dificuldades encontradas

Implementar um algoritmo criptográfico como o ChaCha20 de forma eficiente em hardware requer projetar uma arquitetura paralela que permita operações simultâneas. Isso envolve o uso de várias instâncias dos módulos de `quarterround`, que precisam ser corretamente sincronizadas.

VII. Conclusão

A presente atividade foi importante para revisar conceitos relacionados ao uso e sintaxe da linguagem C e em*Verilog*.

As questões (etapas) explicitadas na introdução deste relatório foram resolvidas, como se pode verificar nos resultados obtidos. Além disso, a prática serviu para observar os dados nas distintas plataformas tanto em software como em hardware.

Observando que o resultado obtido implementado em hardware, se obteve um desempenho mais satisfatório se comparado ao de software.

Referências

[1] Terasic Technologies Inc., *DE2-115 User Manual*, 2010. [Online]. Available: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=165&No=502.

[2] Intel Corporation, *ModelSim* - Intel® FPGA Edition Simulation Quick-Start*, 2019. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug_gs_msa_qii.pdf.

[3] Intel Corporation, *Nios II Processor Reference Guide*, 2020. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/nios2/n2cpu-nii5v1gen2.pdf.

[4] Intel Corporation, *Introduction to the Platform Designer Tool*, 2019. [Online]. Available: https://ftp.intel.com/Public/Pub/fpgaup/pub/Teaching_Materials/current/Tutorials/Introduction_to_the_Qsys_Tool.pdf.

[5] Intel Corporation, *Making Platform Designer Components*, 2020. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/making-platform-designer-components.pdf.

[6] Intel Corporation, *Platform Designer System Design Tutorial*, 2020. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/platform-designer-system-design-tutorial.pdf.

[7] Intel Corporation, *Avalon Interface Specifications*, 2020. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/avalon-interface-spec.pdf.