

Aceleração em Hardware da Transformada Numérica de Fourier para Criptografia Pós-Quântica em System-on-Chip

Pedro L. de Souza Leão, José R. de Oliveira Neto e Juliano B. Lima

Resumo—O CRYSTALS-Kyber é um mecanismo de encapsulamento de chaves baseado em reticulados que utiliza a Transformada Numérica de Fourier (NTT, do inglês *number-theoretic transform*) para acelerar as multiplicações polinomiais. Este trabalho apresenta uma arquitetura de hardware para o cálculo da NTT e de sua inversa, bem como sua implementação em uma plataforma FPGA-SoC. A arquitetura proposta utiliza uma unidade *butterfly* unificada para as operações de Cooley–Tukey e Gentleman–Sande, além de memórias DPRAM organizadas em um esquema *ping-pong*. Para validar o sistema, a arquitetura foi integrada como um periférico mapeado em memória de um processador *dual-core* Cortex-A9 presente na FPGA-SoC. Os resultados demonstraram o funcionamento correto do circuito em um ambiente *hardware-in-the-loop*, alcançando uma frequência máxima de operação de 104,56 MHz e uma utilização de 10.925 ALUTs, 6.404 registradores e 2 blocos DSP.

Palavras-Chave—NTT, PQC, CRYSTALS-Kyber, hardware, FPGA.

Abstract—CRYSTALS-Kyber is a lattice-based key encapsulation mechanism that uses the Number-Theoretic Transform (NTT) to accelerate polynomial multiplications. This work presents a hardware architecture for computing the NTT and its inverse, as well as its implementation on an FPGA-SoC platform. The proposed architecture utilizes a unified butterfly unit for Cooley–Tukey and Gentleman–Sande operations, in addition to DPRAM memories organized in a ping-pong scheme. To validate the system, the architecture was integrated as a memory-mapped peripheral of a dual-core Cortex-A9 processor present on the FPGA-SoC. The results demonstrated the correct operation of the circuit in a hardware-in-the-loop environment, achieving a maximum operating frequency of 104.56 MHz and a utilization of 10,925 ALUTs, 6,404 registers, and 2 DSP blocks.

Keywords—NTT, PQC, CRYSTALS-Kyber, hardware, FPGA.

I. INTRODUÇÃO

À medida que as redes eletrônicas se expandem e se integram, a segurança do tráfego de dados passa a depender crucialmente da solidez dos algoritmos criptográficos vigentes

P. L. de Souza Leão, J. R. de Oliveira Neto e J. B. Lima, Departamento de Eletrônica e Sistemas, Universidade Federal de Pernambuco, Recife; Centro Integrado de Segurança em Sistemas Avançados (CISSA), CESAR, Recife, e-mail: {pedro.souzaleao, joserodrigues.oliveiraneto, juliano.lima}@ufpe.br; Este trabalho foi parcialmente financiado pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) sob os processos nº 422306/2023-1, 442238/2023-1, 312935/2023-4, e 405903/2023-5, pela Financiadora de Estudos e Projetos (MCTI/FINEP/FNDCT/CENTROS TEMÁTICOS 2023, Auxílio Nº 1020/24), e pelo projeto Otimização de PQC para Infraestruturas Críticas, apoiado pelo Centro Integrado de Segurança em Sistemas Avançados (CISSA), com recursos financeiros do PPI IoT/Manufatura 4.0 do MCTI, processo número 08/2-24, firmado com a EMBRAPPII.

[1]. No entanto, esse cenário de confiabilidade encontra-se ameaçado pelo rápido desenvolvimento de tecnologias emergentes. Computadores quânticos criptograficamente relevantes terão o potencial de quebrar os esquemas de chave pública amplamente estabelecidos na atualidade [1]. Diante dessa vulnerabilidade, torna-se imperativa a adoção dos recentes padrões de criptografia pós-quântica e a pesquisa por soluções otimizadas para implementação desses algoritmos que garantam alto desempenho em cenários reais [1], [2].

Como resposta a esse cenário, o NIST (do inglês, *National Institute of Standards and Technology*) iniciou, em 2016, o processo de padronização da criptografia pós-quântica (PQC, do inglês *post-quantum cryptography*), com o objetivo de selecionar algoritmos resistentes a ataques realizados por computadores quânticos. Após múltiplas rodadas de avaliação, o algoritmo CRYSTALS-Kyber foi selecionado para padronização como mecanismo de encapsulamento de chaves baseado em reticulados, sendo posteriormente publicado como padrão ML-KEM [3].

O CRYSTALS-Kyber utiliza operações envolvendo multiplicações polinomiais em corpos finitos, nas quais a transformada numérica de Fourier (NTT, do inglês *number-theoretic transform*) desempenha papel fundamental para redução da complexidade computacional [3], [2]. Nesse contexto, arquiteturas em hardware implementadas em FPGA têm sido amplamente utilizadas para acelerar operações criptográficas devido à flexibilidade de reconfiguração, capacidade de paralelismo e redução da latência de processamento [2]. Diversos trabalhos da literatura propõem arquiteturas dedicadas para implementação da NTT e de sua inversa (INTT, do inglês *inverse NTT*) visando otimização de área, desempenho e consumo de energia [4], [5], [6].

Alinhado a essa vertente, este trabalho apresenta uma arquitetura em hardware para implementação conjunta NTT e da INTT empregadas no CRYSTALS-Kyber utilizando uma plataforma FPGA-SoC. A arquitetura proposta utiliza uma unidade *butterfly* unificada para execução das operações dos algoritmos Cooley–Tukey [7] e Gentleman–Sande [8], além de memórias DPRAM organizadas em esquema *ping-pong* para otimização do fluxo de dados durante a execução da transformada [9]. O acelerador de hardware proposto ocupa até 10.925 ALUTs (do inglês, *adaptive look-up table*), 6.404 registradores de um bit e 2 blocos DSP e possui frequência máxima de operação estimada de 104.56 MHz para o FPGA-SoC utilizado.

O restante deste artigo está organizado da seguinte forma: na

Seção II, é apresentada a fundamentação teórica relacionada à NTT e ao algoritmo CRYSTALS-Kyber. Na Seção III, é descrita a arquitetura proposta e os módulos desenvolvidos. Na Seção IV, são apresentados os resultados experimentais e a análise de desempenho da implementação. Por fim, na Seção V, são apresentadas as conclusões e trabalhos futuros.

II. FUNDAMENTAÇÃO TEÓRICA

Nesta seção são apresentados os principais conceitos teóricos relacionados à criptografia pós-quântica e à NTT utilizados neste trabalho.

A. CRYSTALS-Kyber

O CRYSTALS-Kyber é um dos principais algoritmos de PQC baseados em reticulados e foi selecionado pelo NIST para padronização de mecanismos de encapsulamento de chaves (KEM, do inglês *key encapsulation mechanism*) resistentes a ataques quânticos [10]. O algoritmo é fundamentado no problema M-LWE (do inglês, *module learning with errors*), cuja dificuldade computacional garante segurança mesmo diante de computadores quânticos.

As operações matemáticas utilizadas pelo CRYSTALS-Kyber são definidas em anéis de inteiros, nos quais as principais operações incluem soma e multiplicação polinomial. Dentre essas operações, a multiplicação de polinômios representa o principal custo computacional do algoritmo [3], [2].

Os polinômios e as operações algébricas pertencem ao anel polinomial $R_q = \mathbb{Z}_q[x]/(x^n + 1)$, em que q representa o módulo primo utilizado nas operações modulares e n define o grau dos polinômios empregados pelo algoritmo. Os parâmetros do CRYSTALS-Kyber, e empregados neste trabalho, são apresentados na Tabela I.

TABELA I: Parâmetros utilizados pelo CRYSTALS-Kyber

Parâmetro	Valor
Módulo primo (q)	3329
Grau do polinômio (n)	256
Tamanho dos coeficientes	12 bits
Anel polinomial	$R_{3329} = \mathbb{Z}_{3329}[x]/(x^{256} + 1)$

B. Transformada Numérica de Fourier

A NTT é uma adaptação da transformada discreta de Fourier (DFT, do inglês *discrete Fourier transform*) para corpos finitos, permitindo a realização de operações de convolução e multiplicação polinomial utilizando aritmética modular [2].

A principal aplicação da NTT em algoritmos de criptografia pós-quântica consiste na aceleração da multiplicação de polinômios. Em uma implementação direta, a multiplicação polinomial apresenta complexidade computacional quadrática $O(n^2)$; com a utilização da NTT, essa complexidade é reduzida para $O(n \log n)$, tornando a operação significativamente mais eficiente.

A NTT e sua transformada inversa (INTT) são dadas respectivamente por

$$\hat{a}_k = \sum_{j=0}^{n-1} a_j \omega^{jk} \mod q, \quad (1)$$

$$a_j = n^{-1} \sum_{k=0}^{n-1} \hat{a}_k \omega^{-jk} \mod q, \quad (2)$$

em que a_j representa os coeficientes do polinômio de entrada, $\hat{a}_k$ corresponde aos coeficientes no domínio da transformada, ω representa a n -ésima raiz primitiva da unidade no corpo finito $\mathbb{F}_q$, em que q é um primo ímpar. As potências de ω , utilizadas nas multiplicações ao longo dos estágios da transformada, são conhecidas como *twiddle-factors* [2].

A NTT transforma os coeficientes dos polinômios para o domínio da transformada, permitindo que a multiplicação seja realizada ponto a ponto. Após essa operação, aplica-se a transformada inversa para recuperar o resultado no domínio original. Dessa forma, sendo a e b polinômios sobre R_q , a multiplicação polinomial pode ser efetuada por

$$c = INTT(NTT(a) \circ NTT(b)), \quad (3)$$

em que $\circ$ é a multiplicação ponto a ponto dos coeficientes dos polinômios.

Na prática, a NTT e a INTT são implementadas utilizando algoritmos rápidos baseados na decomposição da transformada em múltiplos estágios. Os algoritmos de Cooley-Tukey (CT) [7] e Gentleman-Sande (GS) [8] estão entre os mais utilizados para o cálculo da NTT direta e inversa, respectivamente, devido à redução da complexidade computacional e à adequação para implementações em hardware.

Esses algoritmos rápidos realizam a decomposição da transformada em múltiplos estágios sucessivos. Em cada estágio, pares de coeficientes do polinômio são combinados por meio de operações aritméticas modulares. Essa abordagem permite reduzir significativamente a quantidade de operações necessárias para o cálculo da transformada, além de favorecer implementações em hardware devido à organização regular do fluxo de dados. No caso do CRYSTALS-Kyber, a NTT de 256 pontos é composta por 7 estágios de processamento. Os procedimentos exatos para o cálculo da transformada direta e inversa, em conformidade com o padrão FIPS 203, estão detalhados no Algoritmo 1 e no Algoritmo 2, respectivamente. Nos algoritmos apresentados, a função $\text{BitRev}_7(i)$ representa a operação de reversão da ordem dos bits de um inteiro i considerando representação de 7 bits, sendo utilizada para determinar a sequência de acesso aos *twiddle-factors* durante a execução da transformada [3].

III. AS ARQUITETURAS PROPOSTAS

Nesta seção é apresentada a arquitetura desenvolvida para a implementação da NTT em hardware utilizando FPGA.

A. Unidade Butterfly

A unidade *butterfly* (BU, do inglês *butterfly unity*) é o principal bloco computacional da NTT, sendo responsável pela execução das operações aritméticas realizadas em cada estágio da transformada. Essas operações incluem somas, subtrações e multiplicações modulares entre os coeficientes dos polinômios. Considerando A e B como os coeficientes de entrada processados em cada operação da transformada e ω

Algoritmo 1: $\text{NTT}(f)$ – Computa a NTT $\hat{f}$ do polinômio $f \in R_q$.

Entrada: Array $f \in \mathbb{Z}_q^{256}$ {Coeficientes do polinômio de entrada}
Saída: Array $\hat{f} \in \mathbb{Z}_q^{256}$ {Coeficientes da NTT do polinômio de entrada}

```

1:  $\hat{f} \leftarrow f$  {Computação numa cópia do array}
2:  $i \leftarrow 1$ 
3: for  $\text{len} \leftarrow 128$ ;  $\text{len} \geq 2$ ;  $\text{len} \leftarrow \text{len}/2$  do
4:   for  $\text{start} \leftarrow 0$ ;  $\text{start} < 256$ ;  $\text{start} \leftarrow \text{start} + 2 \cdot \text{len}$  do
5:      $\text{zeta} \leftarrow \zeta^{\text{BitRev}_7(i)} \pmod{q}$ 
6:      $i \leftarrow i + 1$ 
7:     for  $j \leftarrow \text{start}$ ;  $j < \text{start} + \text{len}$ ;  $j \leftarrow j + 1$  do
8:        $t \leftarrow \text{zeta} \cdot \hat{f}[j + \text{len}] \pmod{q}$ 
9:        $\hat{f}[j + \text{len}] \leftarrow (\hat{f}[j] - t) \pmod{q}$ 
10:       $\hat{f}[j] \leftarrow (\hat{f}[j] + t) \pmod{q}$ 
11:    end for
12:  end for
13: end for
14: return  $\hat{f}$ 

```

como o *twiddle-factor* associado ao estágio atual, a operação da NTT utilizando o algoritmo CT realiza as operações

$$\begin{aligned} O_1 &= A + B \cdot \omega \pmod{q}, \\ O_2 &= A - B \cdot \omega \pmod{q}. \end{aligned}$$

Por outro lado, a INTT utilizando o algoritmo GS realiza as operações

$$\begin{aligned} O_1 &= A + B \pmod{q}, \\ O_2 &= (A - B) \cdot \omega \pmod{q}. \end{aligned}$$

Devido às diferenças entre as operações realizadas pelos algoritmos CT e GS, implementações tradicionais da NTT e da INTT normalmente requerem duas arquiteturas *butterfly* distintas. Essa abordagem aumenta a complexidade do hardware e a utilização de recursos. Para evitar a utilização de duas unidades diferentes, neste trabalho foi adotada a *butterfly* proposta em [6], capaz de realizar tanto a NTT quanto a INTT utilizando a mesma estrutura de hardware, reduzindo a área ocupada, e reutilizar os mesmos componentes aritméticos durante ambas as operações.

B. Unidade de Redução Modular

As operações aritméticas realizadas na NTT e na INTT envolvem multiplicações modulares sucessivas, tornando necessária a utilização de unidades de redução modular eficientes para limitar os resultados ao intervalo definido pelo módulo primo q . Neste trabalho, a unidade de redução modular foi integrada à arquitetura da BU, sendo responsável pela realização das operações modulares associadas às multiplicações executadas em cada estágio da transformada.

Foram utilizadas duas abordagens distintas de redução modular com o objetivo de realizar uma comparação entre desempenho e utilização de recursos. A primeira abordagem

Algoritmo 2: $\text{INTT}(\hat{f})$ – Computa o polinômio $f \in R_q$ correspondente à representação NTT $\hat{f} \in T_q$.

Entrada: Array $\hat{f} \in \mathbb{Z}_q^{256}$ {Coeficientes da representação NTT de entrada}
Saída: Array $f \in \mathbb{Z}_q^{256}$ {Coeficientes da NTT inversa da entrada}

```

1:  $f \leftarrow \hat{f}$  {Computação numa cópia do array}
2:  $i \leftarrow 127$ 
3: for  $\text{len} \leftarrow 2$ ;  $\text{len} \leq 128$ ;  $\text{len} \leftarrow 2 \cdot \text{len}$  do
4:   for  $\text{start} \leftarrow 0$ ;  $\text{start} < 256$ ;  $\text{start} \leftarrow \text{start} + 2 \cdot \text{len}$  do
5:      $\text{zeta} \leftarrow \zeta^{\text{BitRev}_7(i)} \pmod{q}$ 
6:      $i \leftarrow i - 1$ 
7:     for  $j \leftarrow \text{start}$ ;  $j < \text{start} + \text{len}$ ;  $j \leftarrow j + 1$  do
8:        $t \leftarrow f[j]$ 
9:        $f[j] \leftarrow (t + f[j + \text{len}]) \pmod{q}$ 
10:       $f[j + \text{len}] \leftarrow \text{zeta} \cdot (f[j + \text{len}] - t) \pmod{q}$ 
11:    end for
12:  end for
13: end for
14:  $f \leftarrow f \cdot 3303 \pmod{q}$  {Multiplica cada entrada por  $3303 \equiv 128^{-1} \pmod{q}$ }
15: return  $f$ 

```

utiliza a unidade de redução modular proposta em [6], cuja principal vantagem é a utilização apenas de operações de soma e subtração para realizar a redução modular. A segunda abordagem utilizada para comparação é a redução de Barrett [11], realizando a aproximação da divisão modular por meio de operações de multiplicação e deslocamento. Ambas possuem a característica de não precisar de um estágio extra ao final do cálculo da transformada, como ocorrem com reduções baseadas no algoritmo de Montgomery [2]. A arquitetura da redução de Barrett desenvolvida neste trabalho foi implementada de forma totalmente combinacional, possibilitando a realização da redução modular em um único ciclo de Em contrapartida, essa abordagem aumenta o caminho crítico, o que tende a reduzir a frequência máxima do circuito.

C. Memória

Para o armazenamento dos coeficientes dos polinômios foram utilizadas duas memórias DPRAM (do inglês, *dual-port random-access memory*) organizadas em um sistema *ping-pong* [9]. Essa abordagem permite alternar entre as memórias durante a execução da NTT, de forma que uma memória seja utilizada para leitura enquanto a outra é utilizada para escrita.

A utilização do sistema *ping-pong* evita conflitos de acesso simultâneo aos dados e simplifica o fluxo de comunicação entre os estágios da transformada. Além disso, o uso de memórias *dual-port* permite a realização de operações de leitura e escrita em paralelo, contribuindo para aumentar a eficiência da arquitetura em hardware. Outra característica é a possibilidade de compartilhamento externo de uma das portas da DPRAM, possibilitando o uso do sistema proposto como um periférico acessado por memória.

Durante a execução da NTT, a BU realiza leituras dos coeficientes em uma memória e armazena os resultados na outra memória. Ao final de cada estágio da transformada, as funções das memórias são invertidas, permitindo que os dados gerados sejam utilizados como entrada no estágio seguinte.

Além das memórias DPRAM utilizadas para armazenamento dos coeficientes, a arquitetura também possui uma memória ROM (do inglês, *read-only memory*) responsável pelo armazenamento dos *twiddle-factors*. Esses valores são utilizados durante as multiplicações modulares realizadas em cada estágio da NTT e da INTT.

D. Unidade de Controle

A unidade de controle é responsável pela coordenação da execução da NTT, realizando o controle do fluxo de dados, sincronização da BU e geração dos endereços de acesso às memórias. A execução da NTT é controlada por uma máquina de estados finitos (FSM, do inglês *finite state machine*), responsável pelo acionamento da BU e pelo controle das operações de leitura e escrita na memória.

Conforme apresentado na Figura 1, a FSM é composta pelos estados *IDLE*, *COMPUTE*, *STORE* e *DONE*. Inicialmente, o sistema permanece em espera no estado *IDLE*. Após o recebimento do sinal de início, a BU é acionada no estado *COMPUTE*. Quando a operação é concluída, os resultados são armazenados na memória no estado *STORE*. Por fim, o sistema sinaliza o término da operação no estado *DONE*, retornando posteriormente ao estado inicial.

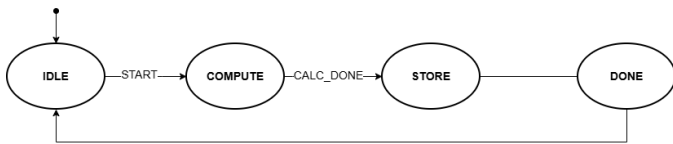


Fig. 1: Diagrama de estados da FSM utilizada no controle da NTT.

Em conjunto com a FSM, o gerador de endereços é responsável pela organização do fluxo de dados durante a execução da transformada. Esse módulo realiza a geração dos índices de leitura e escrita utilizados em cada estágio da NTT e da INTT, permitindo selecionar os coeficientes processados pela BU.

Para cada estágio da transformada, o gerador calcula os pares de coeficientes que devem ser processados pela BU. Esses pares são definidos a partir do estágio atual, que determina a distância entre os coeficientes acessados simultaneamente. Os endereços gerados correspondem às posições *A* e *B* utilizadas nas operações da *butterfly*, sendo calculados a partir de deslocamentos associados aos grupos e elementos pertencentes a cada estágio. Dessa forma, é possível percorrer corretamente os coeficientes armazenados na memória e garantir o sincronismo entre leitura, processamento e escrita dos resultados.

Durante a execução da NTT, o comprimento dos estágios é reduzido progressivamente a cada etapa da transformada, enquanto na INTT ocorre o processo inverso. Dessa forma, o gerador de endereços adapta dinamicamente a organização

dos acessos à memória de acordo com o modo de operação selecionado.

Os índices de acesso aos *twiddle-factors* também são calculados de acordo com o estágio atual da transformada, sendo utilizados nas multiplicações realizadas pela BU. Para a NTT, os fatores são acessados em ordem crescente, enquanto para a INTT os índices são percorridos em ordem inversa. Essa abordagem permite reutilizar a mesma arquitetura para ambas as transformadas, alterando apenas a lógica de geração dos endereços.

IV. RESULTADOS

A arquitetura proposta foi desenvolvida utilizando a linguagem de descrição de *hardware* SystemVerilog. A síntese foi realizada na FPGA Intel Altera Cyclone V SE 5CSEBA6U23I7 utilizando o Quartus Prime 20.1 Standard Edition. O dispositivo 5CSEBA6U23I7 possui uma FPGA com 110 mil elementos lógicos e um processador Dual-Core ARM Cortex-A9 integrado no mesmo chip. A Figura 2 apresenta uma visão geral da arquitetura implementada na FPGA neste trabalho. O acesso do microprocessador à memória do sistema proposto é realizada por meio da interface de comunicação Avalon. Dessa forma, o microprocessador atua como interface de teste entre o computador e a FPGA. Para isso, foi utilizada uma distribuição Linux executando no microprocessador.

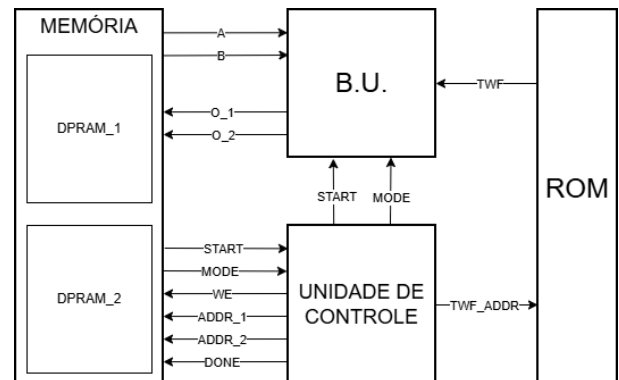


Fig. 2: Diagrama da arquitetura implementada na FPGA.

A DPRAM 1 foi compartilhada entre o microprocessador e a FPGA, permitindo a troca de dados com a arquitetura desenvolvida. Inicialmente, o microprocessador realiza a escrita dos coeficientes de entrada na memória e, após o término da execução da NTT ou INTT pela FPGA, os resultados armazenados são lidos novamente pelo microprocessador.

Além disso, foram desenvolvidos programas em linguagem C para estabelecer a comunicação entre o microprocessador e um computador por meio da interface Ethernet. Para os testes de validação, foram desenvolvidos programas em Python responsáveis por enviar os vetores de entrada para a FPGA e comparar os vetores de saída calculados em um ambiente *hardware-in-the-loop*. Os testes realizados utilizaram 1000 entradas geradas aleatoriamente e os resultados não apresentaram erros.

Para o chip utilizado, a partir das ferramentas disponibilizadas pelo software Quartus, a frequência máxima obtida

para a arquitetura com a unidade de redução proposta em [6] foi de 104,56 MHz. Já a implementação que adota a redução de Barrett apresentou frequência máxima de 64,96 MHz. A menor frequência obtida na arquitetura com a redução de Barrett está associada ao fato de a unidade de redução ter sido implementada de forma totalmente combinacional, além do uso de operações de multiplicação no processo de redução modular, o que aumenta o caminho crítico da arquitetura.

Considerando que a NTT de 256 pontos utilizada no CRYSTALS-Kyber é composta por 7 estágios e realiza 128 operações *butterfly* por estágio, a transformada completa executa 896 operações *butterfly*. Como cada operação necessita de 6 ciclos de *clock* para processamento e armazenamento dos resultados, a arquitetura realiza a NTT ou INTT completa em 5.376 ciclos de *clock*. Levando em consideração o número de ciclos necessários e a frequência máxima alcançada por cada arquitetura, o cálculo da NTT/INTT de 256 pontos leva 51,4 μ s ao utilizar a redução proposta em [6] e 82,8 μ s com a redução de Barrett. Em termos de *throughput*, as arquiteturas entregam 59,8 Mbps (megabits por segundo) e 37,1 Mbps, respectivamente.

Entretanto, a quantidade de ciclos necessária para execução da transformada pode ser reduzida por meio da utilização de técnicas de paralelismo e *pipeline*. Implementações utilizando múltiplas BU operando em paralelo ou arquiteturas com processamento em fluxo contínuo possibilitam aumentar o desempenho da NTT e da INTT, reduzindo a latência total da transformada ao custo de maior utilização de recursos de hardware. Estudos sobre a incorporação dessas estratégias encontram-se em curso.

A Tabela II apresenta a utilização de recursos de *hardware*, destacando as diferenças de área associadas à unidade de redução modular integrada à arquitetura da BU. Como um todo, o acelerador proposto ocupa 10.925 ALUTs, 6.404 registradores de um bit e 2 blocos DSP ao utilizar a redução proposta em [6], e 10.801 ALUTs, 6.404 registradores de um bit e 4 blocos DSP ao adotar a redução de Barrett [11]. Isso equivale a menos de 24% do consumo total do dispositivo utilizado para os testes, o que indica que ainda existe margem para a exploração de arquiteturas com maior grau de paralelismo.

TABELA II: Comparativo de utilização de recursos entre as arquiteturas propostas.

	ALUTs	Registradores	Blocos DSP
Módulos Comuns			
Gerador de endereços	92	41	1
Memória (DPRAM)	10418	6194	0
Memória (ROM)	37	0	0
Módulos Variáveis			
<i>Butterfly</i> (Ref. [6])	378	169	1
<i>Butterfly</i> (Barrett)	254	169	3

Observa-se que a arquitetura utilizando a redução proposta em [6] apresentou maior utilização de ALUTs combinacionais em comparação com a implementação baseada na redução de Barrett. Entretanto, a arquitetura com redução de Barrett utilizou uma quantidade maior de blocos DSP (do inglês,

digital signal processor) devido à presença de operações de multiplicação na unidade de redução modular. Já a arquitetura proposta em [6] realiza a redução utilizando apenas operações de soma e subtração, reduzindo a dependência de recursos dedicados da FPGA.

V. CONCLUSÕES

Neste trabalho, foi apresentada uma arquitetura em *hardware* para implementação da transformada numérica de Fourier (NTT) e de sua inversa (INTT) empregadas no algoritmo CRYSTALS-Kyber utilizando uma FPGA SoC contendo um processador Dual-Core ARM Cortex-A9 integrado. Além disso, foi realizada a comparação entre duas abordagens de redução modular: a unidade de redução proposta em [6] e a redução de Barrett, permitindo avaliar o impacto de cada arquitetura em termos de desempenho e utilização de recursos.

Como trabalhos futuros, prevê-se a integração do acelerador proposto em uma implementação híbrida (*hardware-software*) do algoritmo ML-KEM, permitindo a comparação de desempenho com soluções puramente em *software*. Adicionalmente, serão realizados estudos sobre diferentes técnicas de redução modular para a unidade *butterfly*, bem como a investigação de arquiteturas escaláveis com número configurável de unidades de processamento. O objetivo é otimizar o compromisso entre desempenho e consumo de recursos de *hardware* (*trade-off* área-velocidade), adequando o sistema às restrições de diferentes dispositivos-alvo.

REFERÊNCIAS

- [1] D. J. Bernstein and T. Lange, “Post-quantum cryptography,” *Nature*, vol. 549, no. 7671, pp. 188–194, 2017.
- [2] A. Satriawan, I. Syafalni, R. Mareta, I. Anshori, W. Shalannanda, and A. Barra, “Conceptual review on number theoretic transform and comprehensive review on its implementations,” *IEEE Access*, vol. 11, pp. 70 288–70 316, 2023.
- [3] NIST, “Module-lattice-based key-encapsulation mechanism standard,” 2024, FIPS 203.
- [4] Y. Xing, G. Li, Z. Ye, R. W. L. Luk, D. Chen, H. Yan, and R. C. C. Cheung, “High-Radix/mixed-Radix NTT multiplication algorithm/architecture co-design over Fermat modulus,” *IEEE Transactions on Computers*, vol. 74, no. 10, pp. 3519–3533, 2025.
- [5] T.-H. Nguyen, B. Kieu-Do-Nguyen, C.-K. Pham, and T.-T. Hoang, “High-speed NTT accelerator for CRYSTAL-Kyber and CRYSTAL-Dilithium,” *IEEE Access*, vol. 12, pp. 34 918–34 930, 2024.
- [6] F. Yaman, A. C. Mert, E. Öztürk, and E. Savaş, “A hardware accelerator for polynomial multiplication operation of CRYSTALS-Kyber PQ scheme,” in *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2021, pp. 1020–1025.
- [7] J. W. Cooley and J. W. Tukey, “An algorithm for the machine calculation of complex Fourier series,” *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965.
- [8] W. M. Gentleman and G. Sande, “Fast Fourier transforms: for fun and profit,” in *Proceedings of the November 7-10, 1966, Fall Joint Computer Conference*, ser. AFIPS ’66 (Fall). New York, NY, USA: Association for Computing Machinery, 1966, p. 563–578.
- [9] E. White, *Making embedded systems*, 2nd ed. Sebastopol, CA: O’Reilly Media, mar 2024.
- [10] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehle, “CRYSTALS-Kyber: A CCA-secure module-lattice-based KEM,” in *2018 IEEE European Symposium on Security and Privacy (EuroSP)*, 2018, pp. 353–367.
- [11] P. Barrett, “Implementing the Rivest Shamir Adleman public key encryption algorithm on a standard digital signal processor,” *Advances in Cryptology – CRYPTO’86*, 1986.