반도체공학회 논문지 (TSE: Transactions on Semiconductor Engineering), Vol. 1, No. 1, Oct., 2023
https://doi.org/10.22895/tse.2023.0001 ISSN (On-line) 3022-2028

HLS를 이용한 FPGA 기반 양자내성암호 하드웨어 가속기 설계

FPGA-Based Post-Quantum Cryptography Hardware Accelerator Design using High Level Synthesis

정해성 [1], 이한영 [1], 이한호 [1,+]
(Haesung Jung[1], Hanyoung Lee[1], and Hanho Lee[1,+])

요 약

본 논문에서는 High-Level Synthesis(HLS)을 이용하여, 차세대 양자내성암호인 Crystals-Kyber를 하드웨어 가속기로 설계하여 FPGA에 구현하였으며, 성능 분석 결과 우수성을 제시한다. Crystals-Kyber 알고리즘을 Vitis HLS 에서 제공하는 여러 Directive 를 활용해서 최적화 설계를 진행하고, AXI Interface 를 구성하여 FPGA-기반 양자내성암호 하드웨어 가속기를 설계하였다. Vivado 툴을 이용해서 IP Block Design 를수행하고 ZYNQ ZCU106 FPGA 에 구현하였다. 최종적으로 PYNQ 프레임워크에서 Python 코드로 동영상 촬영 및 H.264 압축을 진행한 후, FPGA 에 구현한 Crystals-Kyber 하드웨어 가속기를 사용해서 동영상 암호화 및 복호화 처리를 가속화하였다.

ABSTRACT

This paper presents the design and implementation of Crystals-Kyber, a next-generation post-quantum cryptography, as a hardware accelerator on an FPGA using High-Level Synthesis (HLS). We optimized the Crystals-Kyber algorithm using various directives provided by Vitis HLS, configured the AXI interface, and designed a hardware accelerator that can be implemented on an FPGA. Then, we used Vivado tool to design the IP block and implement it on the ZYNQ ZCU106 FPGA. Finally, the video was recorded and H.264 compressed with Python code in the PYNQ framework, and the video encryption and decryption were accelerated using Crystals-Kyber hardware accelerator implemented on the FPGA.

KEY WORDS

Post-Quantum Cryptography; Crystals-Kyber; HLS; FPGA; Accelerator

I. 서 론

현재 여러 국가 및 기업에서 양자 컴퓨터의 상용화를 목적으로 활발한 연구를 진행하고 있으며 앞으로 몇 년 안에 양자 컴퓨터가 상용화될 것으로 예상되고 있다. 양자 컴퓨터가 상용화된다면 현재 많이 사용되는 RSA, ECC와 같은 암호 알고리즘은 더 이상 안전하지 않으며 AES, SEED, SHA3 등의 암호 알고리즘도 key 및 output의 길이를 증가시켜야 한다. 이는 양자컴퓨터에 적용할 수 있는 효율적인 알고리즘인 Shor, Grover 알고리즘이 있기 때문이다. Shor 알고리즘을 사용시 소인수 분해, 이산로그 문제를 수 분 이내에 해결 가능하며, Grover 알고리즘을 사용시 데이터 검색 속도를 향상시킬 수 있다. 그렇기에 양자컴퓨터로도 해독되지 않는 고성능 암호화 기술인 양자내성암호의 필요성이 대두되었다.

미국 국립표준 기술연구소(NIST)에서는 이러한 상황에 대비해 양자내성암호 (Post-Quantum Cryptography, PQC)에 대한 표준화 작업을 진행중이며, 2023년 8월 24일 3개의 표준 초안을 발표했다[2].

[1]Department of Electrical and Computer Engineering, Inha University
[+]Corresponding author: Hanho Lee, hhlee@inha.ac.kr
(Received Sept. 14, 2023, Revised Oct. 10, 2023, Accepted Oct. 16, 2023)

PKE/KEM 부분 에서는 격자 기반(Lattice-based) 암호화 알고리즘인 Crystals-Kyber를 표준 초안으로 선정했다.

Crystals-Kyber는 SVP(Shortest Vector Problem), CVP(Closest Vector Problem)와 같은 격자상의 수학적 난제를 기반으로 하며, 이 기법을 적용하면 양자 컴퓨터로도 암호화 및 복호화 수행시간이 오래 걸려서 보안성이 높다. 또한 이는 암호화 및 복호화에 사용하는 키와 암호문의 크기가 다른 암호 알고리즘 대비 상대적으로 짧아 주목을 받고 있는 Ring-LWE (Learning With Error) 기법을 사용하는 암호 알고리즘이다.

본 논문에서는 High-Level Synthesis (HLS)을 이용하여 Crystals-Kyber 양자내성암호를 FPGA 기반 하드웨어 가속기로 설계하고, 이를 이용하여 암호화 및 복호화 속도를 가속하는 방법을 제시한다.

II. 본 론

1. 배경지식

(1) Ring-LWE (Learning With Error)

다항식 환(Ring) Rq = Zq[x] / f(x) 상에서 모든 연산과 암호화를 진행한다. 일반적으로 f(x) = (XN + 1)의 형태, N은 2의 거듭제곱 형태, q = (1 mod 2N)의 형태를 만족하는 소수 형태로 정의한다.

$$a(x) = a_0 + a_1x + a_2x^2 + \ldots + a_{N-1}x^{N-1}$$
$$a_n \in Z/q, q \equiv (1 \bmod 2N) \quad (1)$$

Ring-LWE 시스템은 암호화를 진행할 때 사용하는 공개키(Public Key), 복호화를 진행할 때 사용하는 비밀키(Secret Key) 2개의 비대칭 키를 가지는 공개키 암호 방식을 취하고 있으며, 키를 생성하는 과정에서 공개키 (a, b)를 생성한다. a는 임의의 수(Random number), s는 비밀키를 의미하며, e는 이산 가우스 분포(Gaussian distribution)에 의해 생성된 값 즉, Small noise(Error)에 해당하는 값으로 공개키 (a, b)를 이용하여 메시지 m을 암호화해 (c1, c2)를 얻는 과정에 대해 식(2)를 통해 확인할 수 있다.

$$b = a \times s + e$$
$$c_1 = a \times e_1 + e_2 \quad (2)$$
$$c_2 = b \times e_1 + e_3 + m$$

복호화 과정은 암호화 과정보다 간단한 과정으로 비밀키 s를 이용하여 암호문 (c1, c2)를 다시 메시지 m으로 복원하는 과정이다. 키를 생성하는 단계에서 생성되어 암호화할 때 사용된 공개키 (a, b) 값에 상응하는 비밀키 s를 사용하여야 암호화 과정에서 삽입된 Small noise(Error)값을 소거하여 메시지 m 값을 복원할 수 있다.

Ring-LWE 기법에서 핵심적인 연산은 다항식의 곱셈 연산인데 일반적으로 다항식의 곱셈 연산은 합성곱(Convolution) 연산으로 처리하여 높은 시간 복잡도를 보여준다. 하지만 Ring-LWE에서는 다항식의 곱셈 연산에 NTT(Number Theoretic Transform)를 활용하여 처리한다. NTT를 사용하면 합성곱 연산 '*' 을 Point-wise 곱셈 연산 '•' 으로 변환해주며, 이를 통해서 시간 복잡도는 O(N2)에서 O(NlogN)으로 낮출 수 있다.

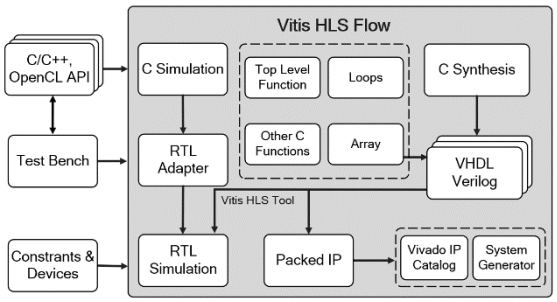

그림 1. Xilinx Vitis HLS Flow

(2) HLS (High-Level Synthesis)

HLS는 그림 1과 같이 여러 단계로 구성된다. 먼저 C/C++ 소스 코드에 대해서 유효성을 검사하기 위해 Testbench 코드를 작성하는 C Simulation 과정으로 시작된다. 다음 단계는 C Synthesis(합성)로 HLS Tool에서 제공하는 UNROOL, PIPELINE 등의 Directive를 적용하여 최적화하고 AXI4 Interface를 구성하여 C/C++ 코드에 대응하는 Verilog 코드를 생성한다. 합성 단계가 완료되면 최종적으로 C/RTL Co-Simulation을 진행하여 C/C++의 연산과 생성된 Verilog HDL의 연산이 동일한지 검증한다. 필요에 따라 Verilog HDL Source 또는 하드웨어 IP를 생성하여 Xilinx Vivado에서 활용할 수 있다. HLS를 이용하여 설계한 하드웨어가 기존 HDL 기반의 하드웨어 설계 프로세스에 비해 가지는 장점은 생산성이다. Low Level의 HDL을 사용하는 것이 아니라 C/C++ 같은 High Level의 언어를 사용하고 HLS Tool을 통해 HDL로 변환해주어서 설계하는 시간이 줄어들기 때문에 생산성이 뛰어나다고 할 수 있다. 또한 검증에 소요되는 시간 역시 크게 줄일 수 있다.

하지만 HLS를 이용한 설계 방식은 추상화 수준이 높기 때문에 둘 사이의 Trade-off가 중요하다.

Algorithm 1. Crystals-Kyber Key Generation [3]

Output: Secret Key sk ∈ $B^{12 \cdot k \cdot n/8}$
Output: Public Key pk ∈ $B^{12 \cdot k \cdot n/8+32}$
1: d ← B^{32}
2: (ρ, σ) := G(d)
3: N := 0
4: **for** i from 0 to k - 1 **do**
5: **for** j from 0 to k - 1 **do**
6: $\hat{A}[i][j]$:= Parse(XOF(ρ, j, i))
7: **end for**
8: **end for**
9: **for** i from 0 to k - 1 **do**
10: s[i] := $CBD_{\eta 1}$(PRF(σ, N))
11: N := N + 1
12: **end for**
13: **for** i from 0 to k - 1 **do**
14: e[i] := $CBD_{\eta 1}$(PRF(σ, N))
15: N := N + 1
16: **end for**
17: $\hat{s}$:= NTT(s)
18: $\hat{e}$:= NTT(e)
19: $\hat{t}$:= $\hat{A} \circ \hat{s} + \hat{e}$
20: pk := ($Encode_{12}(\hat{t} \bmod^{+} q) \,||\, \rho$)
21: sk := $Encode_{12}(\hat{s} \bmod^{+} q)$
22: **return** (pk, sk)

(3) PYNQ Framework

PYNQ는 Xilinx Zynq 및 Alveo 보드에서 적응형 컴퓨팅(Adaptive Computing) 플랫폼을 쉽게 설계할 수 있도록 하기 위해 AMD에서 제공하는 오픈 소스 프로젝트이다. 이를 사용하면 Python 언어와 Library를 사용하여 프로그래머블 로직과 마이크로프로세서의 장점을 모두 활용하여 임베디드 시스템을 구축할 수 있다. 이를 통해 단일 FPGA에서 다양한 임베디드 시스템을 구현할 수 있다. 또한 PYNQ는 Jupyter Notebook 환경을 제공하여 Python 언어를 활용하여 알고리즘이나 Logic을 프로그래밍할 수 있는 개발 환경을 제공한다. 이런 요소들에 의해서 애플리케이션의 빠른 시제품화를 하는데 효과적인 환경을 제공한다. 그림 2는 PYNQ Framework를 도식화한 것이다.

1. Crystals-Kyber 알고리즘

Crystals-Kyber는 미국 국립 표준 기술 연구소(NIST)에서 발표한 양자내성암호 표준 초안 중의 하나인 암호 알고리즘으로 양자 컴퓨터의 공격으로도 안전한 차세대 암호 알고리즘이다[1]. 또한 Ring-LWE 기법의 단점을 보완하기 위해 제안된 Module-LWE 기법을 사용하여 보안 매개변수 설정을 쉽게 하였고, 격자기반 수학적 난제를 기반으로 하여 보다 높은 보안성을 제공한다.

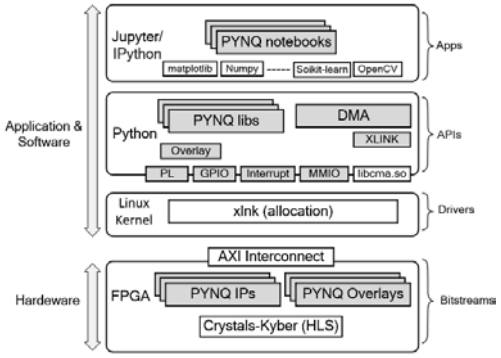

그림 2. PYNQ Framework

Algorithm 2. Crystals-Kyber Encryption [3]

Input: Public Key pk ∈ $B^{12 \cdot k \cdot n/8+32}$
Input: Message m ∈ B^{32}
Input: Random coins r ∈ B^{32}
Output: Ciphertext c ∈ $B^{du \cdot k \cdot n/8+dv \cdot n/8}$
1: N := 0
2: $\hat{t}$:= $Decode_{12}$(pk)
3: ρ := pk + 12 · k · n/8
4: **for** i from 0 to k - 1 **do**
5: **for** j from 0 to k - 1 **do**
6: $A^{T}[i][j]$:= Parse(XOF(ρ, j, i))
7: **end for**
8: **end for**
9: **for** i from 0 to k - 1 **do**
10: r[i] := $CBD_{\eta 1}$(PRF(r, N))
11: N := N + 1
12: **end for**
13: **for** i from 0 to k - 1 **do**
14: e_1[i] := $CBD_{\eta 2}$(PRF(r, N))
15: N := N + 1
16: **end for**
17: e_2 := $CBD_{\eta 2}$(PRF(r, N))
18: $\hat{r}$:= NTT(r)
19: u := $NTT^{-1}(\hat{A}^{T} \circ \hat{r}) + e_1$
20: v := $NTT^{-1}(\hat{t}^{T} \circ \hat{r}) + e_2$ + $Decompress_q(Decode_1(m), 1)$
21: c1 := $Encode_{du}(Compress_q(u, d_u))$
22: c2 := $Encode_{dv}(Compress_q(v, d_v))$
23: **return** c = (c_1| |c_2)

Algorithm 3. Crystals-Kyber Decryption [3]

Input: Secret Key sk ∈ $B^{12 \cdot k \cdot n/8}$
Input: Ciphertext c ∈ $B^{du \cdot k \cdot n/8+dv \cdot n/8}$
Output: Message m ∈ B^{32}
1: u := $Decompress_q(Decode_{du}(c), d_u)$
2: v := $Decompress_q(Decode_{dv}(c + d_u \cdot k \cdot n/8), d_v)$
3: $\hat{s}$:= $Decode_{12}$(sk)
4: m := $Encode_1(Compress_q(v - NTT^{-1}(\hat{s}^{T} \circ NTT(u)), 1))$
5: **return** m

(1) Key Generation(키 생성)

Crystals-Kyber는 기본적으로 공개키 암호 알고리즘으로 서로 다른 두 개의 키인 공개키(Public Key)와 비밀키(Secret Key)를 사용하여 입력 데이터의 암호화와 복호화를 진행한다. 알고리즘1의 Crystals-Kyber 키 생성 알고리즘을 보면 암호화 단계에서

사용하는 공개키와 복호화 단계에서 사용하는 비밀키를 생성하는 것을 볼 수 있다.

(2) Encryption(암호화)

알고리즘2의 Crystals-Kyber 암호화 알고리즘을 보면 키 생성에서 생성한 공개키(Public Key)와 암호화를 진행할 Message (m) 그리고 Random seed 값으로 사용되는 Coin (r)을 입력으로 받아 암호화 과정을 거쳐 Cipher text (c)를 출력한다.

(3) Decryption(복호화)

알고리즘3의 Crystals-Kyber 복호화 알고리즘은 키 생성에서 생성한 비밀키(Secret Key)와 암호화에서 출력된 Cipher text (c)를 입력으로 받아 복호화 과정을 거쳐 원본 Message (m)을 출력한다.

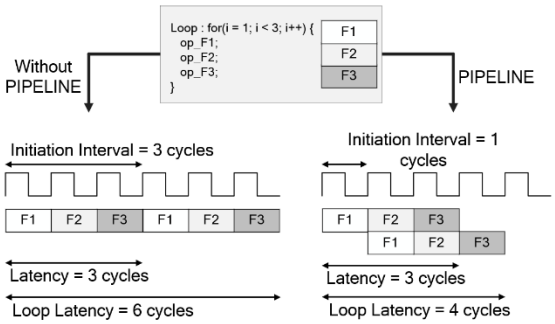

그림 3. PIPELINE 지시문(Directive)

```
ntt_loop_3:
for (j = start; j < start + len; ++j) {

    #pragma HLS LOOP_TRIPCOUNT max = 1
    // PIPELINE
    #pragma HLS PIPELINE II = 8

    buf = *(ptr_r + j + len);
    buf2 = *(ptr_r + j);
    t = fqmul(* (ptr_zetas + k), buf);
    buf = buf2 - t;
    buf2 = buf2 + t;
    *(ptr_r + j + len) = buf;
    *(ptr_r + j) = buf2;
}
```

그림 4. NTT PIPELINE 적용

3. HLS를 이용한 하드웨어 가속기 설계

FPGA Target Board를 Xilinx ZYNQ ZCU106 FPGA 보드로 설정하고 Vitis-HLS Tool을 사용하였다. Vitis-HLS Tool에서는 최적화에 필요한 반복문, BRAM, 연산 리소스 할당 등 여러 Pragma Directive를 제공한다. 이러한 Directive을 통해 Crystals-Kyber 양자내성암호 알고리즘을 최적화하고, Top function에는 입출력 포트에 대해 ARM에서 개발한 버스 통신규약(Bus Protocol)인 AXI (Advanced eXtensible Interface)를 적용했다.

(1) HLS 최적화 기법 (Directive)

본 논문에서 주로 사용된 지시어는 PIPELINE, UNROLL, BIND_OP으로 선언된 반복문을 최적화하고 C에서 RTL로 합성될 때 특정 연산에 특정 리소스를 할당한다. 이 지시어들을Crystals-Kyber의 주요 기능들에 적용했다.

#pragma HLS PIPELINE은 선언된 함수 또는 LOOP에 대한 최적화 Directive으로 현재 실행 중인 작업에 대해 동시성을 부여해 함수 또는 Loop의 시작 간격(Initial Interval)을 감소시킨다.

그림 3을 보면 PIPELINE이 적용된 함수 또는 Loop는 Clock 주기마다 새로운 입력을 받을 수 있어서 짧은 시간 내에 결과값을 출력할 수 있는 것을 볼 수 있다.

Crystals-Kyber에서 PIPELINE Directive가 적용된 부분은 다항식의 곱셈 연산을 수행 하는 NTT(Number Theoretic Transform) 함수에서 실질적인 연산 내용이 기술되어 있는 3번째 Loop로 타이밍 위반이 발생하지 않는 선에서 적용했다. 추가로 Keccak squeeze, Poly_tomsg, Top_func-tion에 PIPELINE을 적용했다.

#pragma HLS UNROLL은 C/C++ 함수의 Rolling되어 있는 Loop를 한 번의 반복에 대한 논리 로직을 합성한 후, 각 반복에 대해 같은 논리 로직을 순서대로 여러 번 수행하게 해준다. 그림 5와 같이 UNROLL Directive를 적용하게 되면 반복문을 펼쳐 단일 작업이 아닌 여러 개의 독립적인 작업으로 만들어 줄 수 있다. 따라서 반복문에 대해 여러 복사본을 생성할 수 있어 모든 반복에 대해 병렬 연산을 수행할 수 있고, Factor 값을 지정해 부분적인 적용도 가능하다. UNROLL Directive를 사용하게 되면 Loop의 반복 횟수에 기반하여 복사본을 생성하기 때문에 FPGA 내에서 Resource 사용량이 크게 급증한다. 따라서 필요한 부분에 적용하여 최적화를 진행한다.

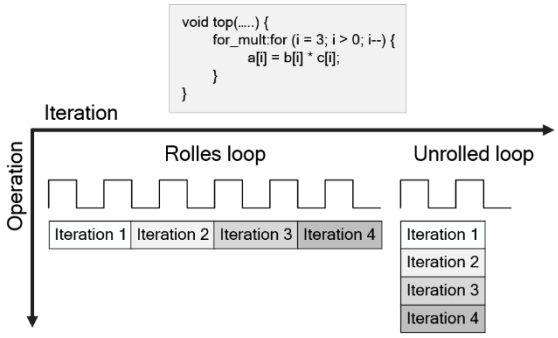

그림 5. UNROLL 지시문(Directive)

```
void polyvec_ntt (
    polyvec *r
) {
    unsigned int i;

    polyvec_ntt_loop_1:
    for (i = 0; I < KYBER_K; i++) {
        // UNROLL
        #pragma HLS UNROLL
        poly_ntt(&r -> vec[i]);
    }
}
```

그림 6. Polyvec_ntt UNROLL적용

```
int16_t barrett_reduce (
    int16_t a
) {
    int16_t t;

    #pragma HLS BIND_OP variable = a op = add impl = dsp
    #pragma HLS BIND_OP variable = a op = sub impl = dsp
    #pragma HLS BIND_OP variable = a op = mul impl = dsp

    #pragma HLS BIND_OP variable = t op = sub impl = dsp
    #pragma HLS BIND_OP variable = t op = add impl = dsp
    #pragma HLS BIND_OP variable = t op = mul impl = dsp

    int16_t tmp = ((int32_t)(((1U << 26) + (KYBER_Q >> 1)) / KYBER_Q) * a >> 26);
    #pragma HLS BIND_STROAGE variable = tmp type = fifo

    t = (tmp << 12) - (tmp << 8) - (tmp << 9) + tmp;

    return a - t;
}
```

그림 7. Barrett_reduction BIND_OP적용

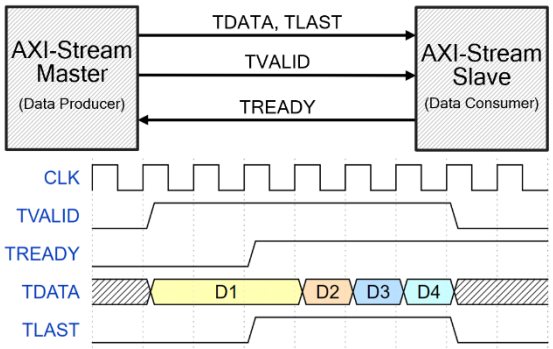

그림 8. AXI4 Stream 채널 데이터 전송

Crystals-Kyber에서 다항식 곱셈의 긴 Latency를 줄이기 위해 UNROLL Directive을 Polyvec_ntt에 적용하였고, Top_function에 UNROLL을 적용했다.
#pragma HLS BIND_OP는 함수에서 선언된 변수를 합성할 때 어떤 Resource로 Mapping할지 지정해 주는 Directive이다. 따라서 적용하려는 변수와 연산(mul, add, sub, div 등)에 대해 Resource를 특정하면 이를 반영해 합성을 진행하고, 또는 자동으로 결정지어 합성한다. 제공되는 옵션으로는 DSP와 Fabric(non-DSP)가 있다.

LUT(Look-Up Table)을 줄이고 연산 속도와 효율을 높이는 방향으로 최적화했다. 암호 알고리즘 내에서 연산 되는 모든 값들에 대해 Modulo연산을 담당하는 Barrett reduction, Montgomery reduction에 해당 Directive를 적용했다. 입력 받은 두 정수를 곱한 후 Montgomery 연산을 진행하는 fqmul 함수, Ring-LWE 기법에서 다항식의 곱셈에 있어 시간 복잡도를 줄여주는 NTT 등의 함수 내부에 있는 곱셈, 덧셈, 뺄셈 연산에 대해서 DSP로 합성되도록 BIND_OP Directive를 적용했다.

```
#Include "hls_stream.h"
#Include "ap_axi_sdata.h"
#Include "hls_vector.h"

typedef ap_axis<8, 0, 0, 0> charS;

void top_kyber (
    stream<charS> &m_stream,
    stream<charS> &c_stream,
    stream<charS> &m_d_stream
);
```

그림 9. 헤더 및 Stream 구조체 정의

(2) AXI4 Interface

HLS을 사용하여 Crystals-Kyber를 설계할 때 Top_function에 구성한 AXI Interface는 ARM에서 개발한 AMBA(Advanced Microcontroller Bus Architecture) Protocol 중 하나이며 데이터 전송을 위해 설계되었다. 그림 8은 AXI4 Stream 채널에서 데이터 전송을 나타낸 것으로 Slave의 TREADY 신호와 MASTER의 TVALID 신호가 둘 다 'HIGH' 일 때 일어난다.

AXI4 Stream은 AXI4 Lite와 달리 Burst 모드를 지원하여 PS-PL간의 고속 데이터 전송에 유용하다는 장점이 있어서 Vitis-HLS툴에서 AXI Interface를 Top_function에서 선언된 입출력 포트에 적용해 주었다. 또한 AXI4 Stream Interface를 사용하기 위해서 'ap_axi_sdata.h', 'hls_stream.h', 'hls_vector.h' 와 같은 헤더를 추가하여 Stream을 위한 전용 데이터 구조체를 그림 9처럼 정의해서 사용해야 하며, 정의된 Stream 구조체 데이터 Type의 변수에서 일반 데이터 Type의 변수로 변환해 주는 과정(Interface → 내부 변수)과 일반 데이터 Type의 변수에서 정의된 Stream 구조체 데이터 Type의 변수로 변환해 주는 과정(내부 변수 → Interface)이 필요하다.

(3) IP Block Design

FPGA 구현을 위해서 Vivado툴을 이용하여 IP Block Design을 하였다. HLS을 이용하여 설계했던 Crystals-Kyber IP, ZYNQ ZCU106 FPGA의 PS(Processing System) IP, DMA(Direct Memory Access)

IP, AXI Interconnect IP 등 주요 IP Block을 적절히 배치하여 연결했다. PS 및 PL 영역 간의 통신은 Stream to Memory Mapped (S2MM), Memory Mapped to Stream (MM2S) 두 가지 채널을 통해 이루어진다. 그림 10에서 IP Block Design 블록도를 제시했다.

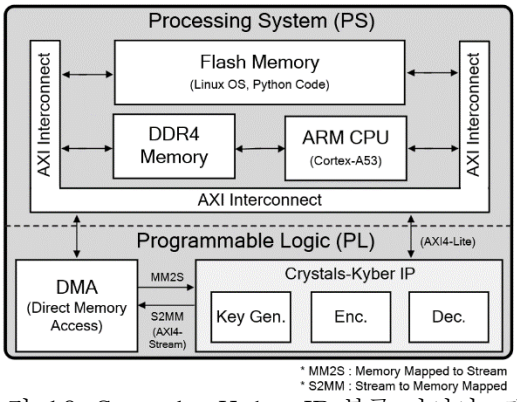

그림 10. Crystals-Kyber IP 블록 다이어그램

PS영역의 설계는 앞서 설명했던 PYNQ 플랫폼을 활용한다. 따라서 PS영역의 ARM 코어는 PYNQ Framework 내에서 동작한다. Python 코드는 다음 단계를 따른다.

Overlay 라이브러리를 사용하여 최종 설계한 하드웨어의 Bitstream파일을 ZYNQ ZCU106 FPGA에 구현하고, DMA 라이브러리를 활용하여 DMA IP에 대한 정보를 추출한다.

이후 DMA를 인스턴스화하며, 암호화할 입력 데이터를 DMA의 MM2S채널에 할당 하면 입력 데이터가 PS영역에서 PL영역으로 전송되어, Crystals-Kyber IP가 암호화 및 복호화를 수행한다. 이러한 과정을 거쳐서 암호화 및 복호화된 출력 데이터는 DMA의 S2MM채널에 할당하여 다시 PL영역에서 PS영역으로 데이터가 전송되도록 한다.

4) H.264 영상 압축 알고리즘

H.264는 MPEG-4 Part10, AVC으로도 불리는 동영상 압축을 위한 알고리즘으로 이전에 사용하던 MPEG-4 Part2 보다 더 향상된 성능을 가진다. 후속 규격으로 H.265 (HEVC), H.266(VVC)가 존재하나 알고리즘 복잡도로 인한 성능문제로 현재까지도 H.264 포맷이 가장 보편적으로 사용되고 있다.

본 논문에서는 PYNQ 프레임워크의 Jupyter Notebook 환경에서 Python코드를 이용해서 실시간으로 동영상을 촬영하고, 이를 H.264 포맷으로 압축하여 저장하였다. 이렇게 압축한 동영상을 Crystals-Kyber가 한번에 처리할 수 있는 32byte씩 나눠서 입력으로 넣어주었고, 이런 과정을 통해서 실시간으로 촬영한 영상의 암호화 및 복호화 수행 시간을 단축시킬 수 있었다.

Ⅲ. 성능 분석

본 논문에서 제안한 HLS를 이용한 양자내성암호 하드웨어 가속기 합성 결과를 기존 HLS로 구현한 Crystals-Kyber [12]와 성능 비교 분석하였다. 설계한 양자내성암호 하드웨어 구조를 FPGA에 구현한 후, 10초 분량의 영상을 촬영하여 암복호화한 후 수행 시간을 측정하고, MPEG-4 Part2 및 H.264로 압축한 후 암복호화하여 수행 시간을 측정하여 성능 비교 분석하였다. 사용한 툴은 Xilinx의 Vitis HLS와 Vivado이며, ZYNQ ZCU106 FPGA를 Target Board로 설정하여 설계하고, 클럭 주파수는(Clock Frequency)는 343.3MHz, 동영상은 30 fps로 촬영하였다.

표 1은 HLS의 Directive를 이용하여 최적화를 진행한 Crystals-Kyber의 합성 결과와 다른 논문에서 HLS만을 이용하여 설계한 Crystals-Kyber[12]의 결과를 비교 분석한 것이다. 본 논문에서 제안한 HLS를 이용한 양자내성암호 하드웨어 가속기 구조가 비교군 대비 LUT는 5.37배, FF은 5.11배, DSP는 5.95배로 FPGA Resource를 더 많이 사용하는 것을 볼 수 있다. 반면에, Latency는 24.08배 훨씬 더 향상되었으며, ATP(Area x Time)는 4.47배 더 향상된 것을 볼 수 있다.

표1. 구현 결과 및 성능 비교

	[12] HLS design	**Proposed HLS design**
LUTs	31,807 (13.8%)	170,945 (75.2%)
FFs	19,820 (4.3%)	101,342 (21.99%)
BRAMs	32 (10.3%)	40.5 (12.98)
DSPs	290 (16.8%)	1,728 (100%)
Clock Freq. (MHz)	207.7	343.3
Latency (μs)	931.7	38.684
Area x Time (LUTs x Latency)	29.6	6.61
FPGA	ZYNQ104	ZYNQ 106

표 2는 10초 분량의 영상을 각각 압축이 적용되지 않은 원본과, MPEG-4 Part2 압축, H.264 압축 알고리즘을 적용한 데이터를 Crystals-Kyber 암복호화 구조를 구현한

ZYNQ FPGA에서 암호화 및 복호화 처리 시간을 비교한 것이다. 즉, 실시간으로 촬영한 같은 영상에 대해서 각각의 압축 알고리즘을 적용한 후 암호화 및 복호화 처리 시간을 비교한 것이며, 그 결과 H.264 압축 알고리즘을 사용한 것이, MPEG-4 Part2 대비 4.45 ~ 4.68배 빠르며, 압축이 적용되지 않은 원본 대비 547 ~ 754배 빠른 성능을 보여주었다.

표 2. 10초 분량 영상 데이터 암복호화 수행시간(sec)

	640*480	1280 * 720	1920 *1080
무압축 원본	5,554	16,569	37,297
MPEG-4 Part2 압축	33.74	105.49	303.41
H.264 압축	7.36	22.53	68.07

V. 결 론

본 논문에서 제안한 양자내성암호 하드웨어 가속기 설계는 High-level synthesis를 이용했으며 여러 Directive를 적용하여 알고리즘을 최적화하고 성능을 향상시켰다. 설계한 하드웨어 구조를 ZYNQ ZCU106 FPGA 보드에 구현했으며, 보드의 PS(Processing System) 영역에 PYNQ 프레임워크를 사용해 Python 코드로 영상 촬영 및 H.264 압축을 수행하였다. 이후 압축한 영상을 설계한 양자내성암호 하드웨어 가속기를 사용해서 암호화 및 복호화 처리 속도를 가속화하였다. 전체 양자내성암호 하드웨어 가속기를 ZYNQ ZCU106 FPGA 보드를 통해서 구현하고 검증하였다. 양자내성암호 하드웨어 가속기를 더 최적화 하여 설계한다면 실시간 영상 암호화 및 복호화 시스템의 상용화가 가능할 것이다.

감사의 글

이 논문은 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원 (No. 2021R1A2C1011232)과 한국연구재단-시스템반도체융합전문인력 육성사업 지원을 받아 수행된 연구임 (No.2020M3H2A1076786).

참고 문헌

[1] “PQC (Post-Quantum Cryptography) Selected Algorithms 2022,” NIST, last modified Aug 24, 2023, accessed Sep 4, 2023. https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[2] “Comments Requested on Three Draft FIPS for Post-Quantum Cryptography” NIST, last modified Aug 24, 2023, accessed Sep 4, 2023. https://csrc.nist.gov/news/2023/three-draft-fips-for-post-quantum-cryptography.

[3] R. Avanzi, et. al. “CRYSTALS-Kyber Algorithm Specifications And Supporting Documentation (version 3.02),” NIST PQC Round 3 submission, Aug. 2021.

[4] Vitis High-Level Synthesis, “User Guide,” UG 1399, v2021.2, July 17, 2023.

[5] T. N. Tan, S. Kim, Y. Eom and H. Lee, “Area-Time Efficient Hardware Architecture for CRYSTALS-Kyber,” Applied Sciences, 12(11), 5305, May 24, 2022.

[6] D. T. Nguyen, V. B. Dang and K. Gaj, “A High-Level Synthesis Approach to the Software/Hardware Codesign of NTT-based Post-Quantum Cryptography Algorithms,” 2019 International Conference on Field-Programmable Technology (ICFPT), pp. 371-374, Dec., 2019.

[7] P. W. Shor, “Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer,” SIAM J. Comput., vol. 26, pp. 1484-1509, Oct. 1997.

[8] O. Regev, “On lattices, learning with errors, random linear codes, and cryptography,” in Proceedings of the ACM Symposium on Theory of Computing, Baltimore, USA, pp. 84-93, May. 2005.

[9] V. Lyubashevsky, “On ideal lattices and learning with errors over rings,” Annual international conference on the theory and applications of cryptographic techniques, pp. 1-23, 2010.

[10] T. Zijlstra, K. Bigou, A. Tisserand, “Lattice-Based Cryptosystems on FPGA: Parallelization and Comparison Using HLS”, IEEE Transactions on Computers, Vol. 71, Aug. 2022.

[11] J. P. Smith et al., "A High-Throughput Oversampled Polyphase Filter Bank Using Vivado HLS and PYNQ on a RFSoC," IEEE Open Journal of Circuits and Systems, Vol. 2, pp. 241-252, 2021, doi: 10.1109/OJCAS.2020.3041208.

[12] 이창현, “ 양자내성암호시스템의 HLS-기반 HW/SW 공동 설계와 HLS-RTL Hybrid 설계”, 공학석사논문, Feb. 2022.

[13] E. Ozcan and A. Aysu, "High-Level Synthesis of Number-Theoretic Transform: A Case Study for Future Cryptosystems," IEEE Embedded Syste

ms Letters, vol. 12, no. 4, pp. 133-136, Dec. 2020, doi: 10.1109/LES.2019.2960457.

[14] V. Kostalabros, J. Ribes-González, O. Farràs, M. Moretó and C. Hernandez, "HLS-Based HW/SW Co-Design of the Post-Quantum Classic McEliece Cryptosystem," 2021 31st International Conference on Field-Programmable Logic and Applications (FPL), Dresden, Germany, pp. 52-59, 2021.

[15] A. Guerrieri et al., “Design Exploration and Code Optimizations for FPGA-Based Post-Quantum Cryptography using High-Level Synthesis”, TechRxiv, Mar. 2022, doi: 10.36227/techrxiv.19404413.v1

정 해 성 (Haesung Jung)

2023년 2월 : 인하대학교 정보통신공학과 졸업
2023년 3월~현재 : 인하대학교 전기컴퓨터공학과 석사과정

<관심분야> 양자내성암호 하드웨어 가속기 설계, 집적회로 설계, 시스템반도체 설계

이 한 영 (Hanyoung Lee), 학생회원

2023년 2월 : 인하대학교 정보통신공학과 졸업
2023년 3월~현재 : 인하대학교 전기컴퓨터공학과 석사과정

<관심분야> 동형암호 하드웨어 가속기 설계, 집적회로 설계, 시스템반도체 설계

이 한 호 (Hanho Lee), 정회원

2000년 4월 : 미네소타대학교 전기컴퓨터공학과 박사 졸업
2000년 5월~2002년 8월 : Lucent Technologies 전문연구원
2002년 8월~2004년 8월 : 코네티컷대학교 전기컴퓨터공학과 조교수
2004년 9월~현재 : 인하대학교 정보통신공학과 교수

<관심분야> 디지털 집적회로 및 시스템 설계, 양자내성암호 및 동형암호 아키텍처