Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.34, NO.1, Feb. 2024 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2024.34.1.11

FPGA를 사용한 radix-2 16 points FFT 알고리즘 가속기 구현*

이 규 섭,[1†] 조 성 민,[1] 서 승 현[2‡]
[1,2]한양대학교 (대학원생, 교수)

Radix-2 16 Points FFT Algorithm Accelerator Implementation Using FPGA*

Gyu Sup Lee,[1†] Seong-Min Cho,[1] Seung-Hyun Seo[2‡]
[1,2]Hanyang University (Graduate student, Professor)

요 약

최근 신호처리, 암호학 등 다양한 분야에서 FFT(Fast Fourier Transform)의 활용이 증가함에 따라 최적화 연구의 중요성이 대두되고 있다. 본 논문에서는 FPGA(Field Programmable Gate Array) 하드웨어를 사용하여 radix-2 16 points FFT 알고리즘을 기존 연구들보다 빠르고 효율적으로 처리하는 가속기 구현 연구에 대해 기술한다. FPGA가 갖는 병렬처리 및 파이프라이닝 등의 하드웨어 이점을 활용하여 PL(Programmable Logic) 파트에서 Verilog 언어를 통해 FFT Logic을 설계 및 구현한다. 이후 PL 파트에서의 처리 시간 비교를 위해 PS(Processing System) 파트에서 Zynq 프로세서만을 사용하여 구현 후, 연산 시간을 비교한다. 또한 관련 연구와의 비교를 통해 본 구현 방법의 연산 시간 및 리소스 사용의 효율성을 보인다.

ABSTRACT

The increased utilization of the FFT in signal processing, cryptography, and various other fields has highlighted the importance of optimization. In this paper, we propose the implementation of an accelerator that processes the radix-2 16 points FFT algorithm more rapidly and efficiently than FFT implementation of existing studies, using FPGA (Field Programmable Gate Array) hardware. Leveraging the hardware advantages of FPGA, such as parallel processing and pipelining, we design and implement the FFT logic in the PL (Programmable Logic) part using the Verilog language. We implement the FFT using only the Zynq processor in the PS (Processing System) part, and compare the computation times of the implementation in the PL and PS part. Additionally, we demonstrate the efficiency of our implementation in terms of computation time and resource usage, in comparison with related works.

Keywords: FPGA, FFT, Accelerator

I. 서 론

시간의 연속적인 신호를 주파수 영역으로 변환하는 FFT(Fast Fourier Transform) 연산은 과학, 공학, 의학 등 다양한 분야에서 널리 사용되는 중요한 알고리즘이다. 신호처리, 주파수 특성 분석, 암호학 등의 분야에서 FFT 연산은 필수적인 역할을 수행한다. 특히, 양자컴퓨팅 환경에서도 안전하게 설계된 양자내성암호(Post-Quantum Cryptography) 중 격자 기반(Lattice-based) 암호에서 FFT 연산과 그의 변형인 NTT(Number Theoretic Transform) 연산이 주로 사용되고 있다. 이러한

Received(11. 27. 2023), Modified(12. 26. 2023), Accepted(12. 27. 2023)
* 본 연구는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. 2021R1A2C1095591)
† 주저자, r22jiw0n@hanyang.ac.kr
‡ 교신저자, seosh77@hanyang.ac.kr(Corresponding author)

다항식 곱셈에 사용되는 FFT 및 NTT 연산은 키 생성, 서명 생성(암호화) 및 검증(복호화) 과정에서 많은 연산량을 차지하고 있다. NIST 표준 양자내성 암호인 FALCON-512의 경우, 키 생성 과정에서 FFT 연산이 약 600회 이상 사용되기 때문에 하드웨어를 활용한 최적화 연구가 활발하게 이루어지고 있다. 이처럼 FFT 연산의 최적화는 향후 양자내성 암호가 기존 클래식 암호 알고리즘을 대체하는 과정에서 필수적이다.

FFT 알고리즘을 빠르고 효율적으로 구현하는 방법의 하나로 FPGA(Field Programmable Gate Array) 하드웨어를 사용한 가속화 연구들이 활발하게 이루어져 왔다.

FPGA는 고성능 연산 처리, 높은 병렬처리가 가능하며, 전력 소모량이 적다는 장점을 갖고 있어 알고리즘의 최적화에 효율적일 뿐만 아니라 임베디드 및 IoT 환경에 적합한 하드웨어이다. 또한 FPGA를 활용한 가속화 기법은 FFT 연산뿐만 아니라 다양한 알고리즘에 적용이 가능하여 연구의 중요성이 높다. 따라서 2005년부터 현재까지 관련 연구가 꾸준하게 진행되고 있다[1]. 특히, 독립적인 연산에 대한 병렬처리, 저장장치를 활용한 파이프라인 기법 적용 등 다양한 방식으로 FPGA의 장점들을 활용하여 알고리즘의 최적화 설계가 이뤄지고 있다[2].

본 논문에서는 Zynq 프로세서 기반 SoC (System on Chip) 하드웨어인 Zybo Z7-20 FPGA를 사용하여 효율적인 16-point radix-2 FFT의 구현 방안을 제시한다. 본 논문의 구조는 다음과 같다; 2장에서는 연구의 이해를 돕기 위해 FFT의 구조 및 FPGA에 대한 배경지식을 기술한다. 3장에서는 본 논문과 유사한 연구들을 분석하여 어떤 방식으로 각 연구에서 최적화하고 있는지 기술한다. 4장에서는 Zynq 프로세서만을 활용한 PS(Processing System) 파트의 구현과 FPGA의 병렬 처리, 파이프라이닝 등의 특성을 최대한 활용한 PL(Programmable Logic) 파트의 구현 방법에 대해 자세하게 기술한다. 5장에서는 구현 결과를 분석하고, 기존의 유사 연구 결과와 비교하여 본 연구에서 제안된 구현 방법이 리소스 사용 측면에서 얼마나 효율적인지를 검증한다.

결과적으로 본 연구에서 진행한 FFT 연산의 하드웨어 가속기 구현은 CPU를 이용한 방식보다 속도가 더 빠르고 유사 연구와 비교하여 리소스를 효과적으로 활용한다.

II. 배경 지식

본 장에서는 연구의 이해를 돕기 위해 구현한 Radix-2 FFT 연산과 사용되는 FPGA의 구조에 관해 설명한다.

2.1 Radix-2 Fast Fourier Transform

이산 푸리에 변환(DFT: Discrete Fourier Transform)은 시간 영역의 연속적인 신호를 주파수 영역으로 변환하여 신호의 주파수 성분을 분석할 수 있는 알고리즘이다. 이 변환은 특히 주기적인 패턴이나 주파수 영역에서 표현되는 특성을 갖는 신호를 분석하는 데 유용하다.

DFT는 주파수 변환의 기본적인 형태로써, 주어진 시퀀스 $x[n]$에 대한 DFT는 수식 1과 같이 정의된다.

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j(2\pi/N)kn} \quad (1)$$

이때 N은 데이터 포인트의 수이며, k는 주파수 인덱스이다.

DFT는 $O(N^2)$의 계산 복잡도를 차지한다. 분할정복 알고리즘(Divide and Conquer algorithm)을 사용하는 Cooley-Tukey 알고리즘을 기반으로 FFT는 이 계산을 최적화하여 복잡도를 $O(N\log N)$으로 줄인다. Radix-2 FFT는 잘 알려진 알고리즘 중 하나로 입력 데이터의 크기가 2의 거듭제곱일 때 가장 효율적이다.

Radix-2 FFT의 핵심은 butterfly 연산이다. 이 연산은 두 입력 데이터 포인트에 대한 덧셈 및 뺄셈과 회전자(twiddle factor)의 곱셈을 포함한다. 주어진 두 데이터 포인트 a,b에 대한 butterfly 연산은 수식 2와 같이 정의된다. 이때 $e^{-j(2\pi/N)kn}$은 회전자이다.

$$\begin{aligned} a' &= a + b \\ b' &= (a - b) \cdot e^{-j(2\pi/N)kn} \end{aligned} \quad (2)$$

본 논문에서 구현한 알고리즘은 radix-2 DIT(Decimation In Time) FFT로, 입력(time domain)을 나누는 방식이다. 또한, $N = 2^4$개의 데이터 포인트를 갖는 알고리즘으로 Fig.1.과 같이

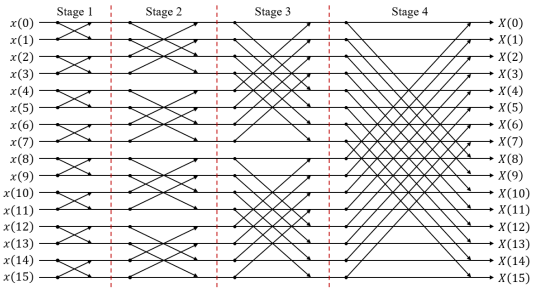

Fig. 1. The process of a radix-2 16 points FFT using butterfly units

butterfly 연산의 반복적인 구조로 이루어져 있다. 이는 하드웨어를 활용하여 구현할 때, 병렬 연산으로 빠르게 처리할 수 있으며, 4개의 stage가 끝나는 지점에 저장매체를 활용하면 파이프라이닝까지 활용할 수 있는 구조이다.

2.2 Field Programmable Gate Array

FPGA(Field-Programmable Gate Array)는 사용자가 직접 프로그래밍이 가능한 디지털 집적회로이다. FPGA는 Fig.2.와 같이 크게 CLBs, Interconnects, I/O Blocks의 3종류로 구성되며, 자세한 설명은 다음과 같다.

1. CLBs(Configurable Logic Blocks): CLBs는 기본적인 논리 연산 및 복잡한 함수를 수행할 수 있는 논리 게이트와 flip-flops로 구성된다.
2. Interconnects: 이들은 CLBs를 유연하게 연결할 수 있으며, 다양한 방식으로 구성이 가능하다.
3. I/O Blocks: 이들은 FPGA의 내부와 외부 간의 데이터 통신을 관리한다.

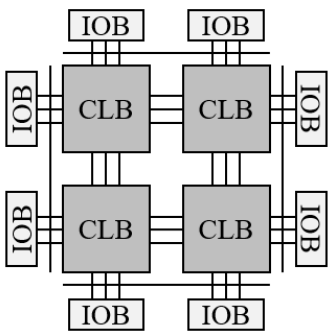

Fig. 2. Structure of FPGA Hardware

FPGA의 아키텍처는 병렬처리에 적합하게 설계되어 있어 고성능 연산 처리가 가능하다. 특히, 매트릭스 연산, 신호 처리, 복잡한 알고리즘 실행과 같은 연산 집약적인 작업에서 큰 장점을 지니고 있다. 또한, FPGA의 효율적인 전력 관리 기능으로 인해 낮은 전력 소모로 운영이 가능하다. 이러한 특성은 배터리로 구동되는 임베디드 시스템이나 IoT 환경에서 중요하기 때문에 FPGA는 해당 분야들에서도 널리 사용되고 있다.

본 논문에서 사용한 FPGA는 SoC(System on Chip) 하드웨어의 한 종류이다. FPGA SoC는 전통적인 FPGA 로직(PL: Programmable Logic)과 고정된 프로세서 시스템(PS: Processing System)을 하나의 칩 안에 통합한다.

PS 파트는 일반적으로 ARM과 같은 고정된 프로세서 아키텍처를 포함하며, OS 운영, 시스템 관리, 응용 프로그램 실행과 같은 소프트웨어 중심의 작업을 처리한다. 반면, PL 파트는 사용자가 특정 언어(VHDL 등)를 활용하여 회로를 설계할 수 있으며 병렬 처리나 신호 처리, 하드웨어 가속 등의 하드웨어 중심의 작업에 적합하다.

PS와 PL 사이의 통신은 일반적으로 AXI(Avanced eXtensible Interface)와 같은 고성능 인터페이스 버스를 통해 이루어진다. 이 인터페이스는 데이터 전송 및 명령을 효율적으로 처리하도록 설계되어 있으며, 두 파트 간의 빠르고 유연한 상호 작용을 가능하게 한다.

이러한 통합적 구조는 FPGA SoC가 높은 성능의 연산 처리와 유연한 프로그래밍 능력을 동시에 갖추게 한다. 따라서 FPGA SoC는 빠른 반응 시간, 고성능 데이터 처리, 그리고 소프트웨어와 하드웨어의 효율적인 상호 작용이 필요한 다양한 애플리케이션에서 사용될 수 있다.

III. 관련 연구

2005년 이후로 FPGA를 활용한 FFT 연산 최적화 기법들이 활발히 연구되고 있다. 그 중, 본 논문에서의 FPGA 최적화 대상 알고리즘인 16 points FFT와 같이 작은 input point에 대한 FFT의 FPGA 최적화 연구들은 다음과 같다.

2013년 Lakshmi Santhosh와 Anoop

Thomas는 radix-2^2 FFT 알고리즘을 FPGA 상에서 구현하였다 [3]. FFT 연산을 하드웨어 상에서 구현할 때 가장 바람직한 구조는 radix-4 알고리즘과 곱셈의 수가 동일하면서도 radix-2 알고리즘에 사용되는 butterfly 구조를 지녀야 한다. radix-4 알고리즘은 radix-2보다 효율적이기는 하지만 $4n$ point FFT만 처리할 수 있다. 따라서 radix-4 알고리즘과 동일한 곱셈 복잡성을 가지면서도 radix-2 알고리즘의 butterfly 구조를 유지하는 radix-2^2 알고리즘은 가장 하드웨어 지향적인 알고리즘이다. 그들은 이러한 radix-2^2 알고리즘을 FPGA 상에서 구현하고 resource 및 utilization 측면에서 기존의 radix-2 알고리즘과 비교 제시하였다.

Abhishek Mankar 등은 유닛의 입력에 계수를 분배하는 방식인 NEDA(NEw Distributed Arithmetic)를 활용하여 16 points radix-4 FFT를 FPGA 상에서 구현하였다 [4]. 그들은 이러한 NEDA 방식을 통해 곱하기 및 누적 (MAC: Multiply and ACcumulate) 유닛에서의 ROM 사용을 제거하였으며, 덧셈기 및 쉬프터만을 사용하여 MAC 유닛을 구현하였다.

2015년 Josue Saenz S. 등은 16 point 및 32 point radix-2 FFT 알고리즘을 VHDL (VHSIC Hardware Decription Language) 기술 언어를 통해 FPGA 상에서 구현하는 방안에 대해 연구하였다 [5]. VHDL이란 FPGA에서의 회로 설계를 위한 하드웨어 기술 언어로 기본적으로 복소수 상에서 작동하지 않는다. [5]에서는 복소수 연산이 존재하는 FFT 알고리즘을 VHDL을 통해 구현하기 위해 복소수 타입이 정의된 패키지를 생성하고 복소수의 덧셈, 뺄셈 및 곱셈 연산을 추가하였다. 그들은 또한 32 point 알고리즘 구현 시 PISO(Parallel In Serial Out) 및 SIPO(Serial In Parallel Out) 쉬프트 레지스터를 사용하여 데이터 입력 및 출력을 직렬화함으로써 I/O 블록의 사용을 줄여 사용되는 resource를 최소화하였다.

2018년 Ghattas Akkad 등은 HLS(High Level Synthesis) 기법을 활용하여 radix-2와 radix-4 FFT의 알고리즘을 구현하였다 [6]. HLS는 C나 C++과 같은 고급 언어로 알고리즘을 설계한 후, 이를 Verilog나 VHDL과 같은 하드웨어 기술 언어로 변환하는 기법을 의미한다. 그들은 VHDL 및 HLS 도구를 활용하여 FFT 구현의 다양한 구조를 제안하였다. 특히 병렬화, 파이프라이닝 및 하드웨어 재사용과 같은 최적화 기법을 적용하였다. 결론적으로 HLS를 사용하여 HDL 수준의 최적화와 유사한 성능을 달성할 수 있음을 보여주었으며, 이를 통해 개발 시간을 크게 단축하면서도 효율적인 결과를 얻을 수 있음을 보였다.

Vishwas Patil과 Manu T. M은 2021년 twiddle factor를 적절히 정규화하여 butterfly 구조에서의 복소수 곱셈을 단순한 곱셈기 및 덧셈기의 조합으로 대체하여 radix-2 FFT를 FPGA 상에서 가속화 설계하였다 [7]. 그들은 이를 통해 곱셈 연산의 산술 복잡도를 기존 $2N\log_2 N$에서 $1\frac{2}{3}\log_2 N$으로 16.68% 감소시켰으며, flip-flop의 수를 6.68% 감소시켰다.

위의 연구들은 FPGA의 하드웨어 특성을 최대한 활용하여 FFT 구현의 효율성을 향상시키기 위한 다양한 접근 방식을 제안하였다. 주요한 최적화 전략들로는 병렬 연산의 증대, 반복 구조의 단순화, 복잡한 산술 연산의 간소화 등이 포함되어 있다. 특히, 곱셈과 같은 연산의 복잡도를 줄이기 위한 연구와, FPGA 상에서의 리소스 및 연산 속도 최적화에 중점을 두었다.

본 논문에서도 이러한 연구의 흐름에 기반하여 FPGA를 활용한 FFT 연산 최적화에 대해 기술한다. 특히 Zynq SoC FPGA를 기반으로 한 16-point radix-2 FFT의 구현 방안에 초점을 두고 하드웨어 특성을 최대한 활용한 FFT 연산의 효율성 향상 방안을 제시하며, 기존 연구와의 결과를 비교하고자 한다.

IV. 최적화 구현 방법

본 장에서는 FPGA 구현 방법을 PS 파트만을 사용하여 제공되는 프로세서로 FFT를 구현하는 설계와 PS 파트와 PL 파트를 co-operation 하는 설계로 나누어 자세하게 기술한다. 구현에는 Digilent 社의 Zynq-7000시리즈 중 SoC 보드인 zybo z7-20(XC7Z020-1CLG400C) FPGA 하드웨어를 사용했다. 개발환경은 Xilinx 社의 Vivado 2020.2이며 PS 및 PL 파트에 필요한 C언어 애플리케이션 코딩은 Vitis 2020.2에서 구현했다.

4.1 Programmable Logic 구현

본 논문에서 제시한 PL 구현은 기존에 순차적으로 진행되던 FFT 연산에 대해 Flip-Flop 레지스터를 활용하여 파이프라이닝으로 구현함으로써 연산을 효율적으로 처리한다. 또한 독립적으로 반복되는 butterfly 연산에 대해 모듈화를 통한 병렬 연산이 가능하도록 설계한다.

PL을 활용한 전체적인 구현은 Fig.3.과 같다. PC에서 input으로 사용되는 16개의 point를 입력하면, UART 통신을 통해 FPGA의 PS 파트로 전달된다. 이후, PS 파트에서 AXI4_LITE 인터페이스를 통해 input point가 AXI_IP의 input port에 저장되고, FFT 연산이 이뤄지는 FFT_CORE_IP에 최종적으로 전송된다. 연산이 완료되면 결과 값이 output port에 저장되며, 다시 AXI4_LITE를 통해 PS에서 읽어오고 PC에서 결과 값을 확인한다. 이때, PS에서 전달받은 input point를 생성한 레지스터에 저장하거나 저장된 결과 값을 읽어오는 과정은 Xilinx에서 제공하는 함수를 활용하여 C언어로 구현했다.

실질적으로 FFT 연산이 진행되는 FFT_CORE_IP의 설계는 Fig. 4.와 같다. FPGA의 특징인 병렬처리와 파이프라인 기법을 최대한 활용했으며, FFT 연산에 활용되는 회전자를 상수로 저장하여 연산을 최소화하였다.

또한, Fig.1.과 유사한 구조로 16개의 input point에 대하여 4개의 stage에 따라 butterfly 연산이 8번 병렬로 진행되며, Fig.5.와 같이 구현된 butterfly 연산은 모듈화를 통해 연산이 진행될 때마다 호출된다. 연산 과정에서 반복적으로 사용되는 butterfly 연산에 대해 unrolled 구조로 설계했다. 이때 unrolled 구조란, 하드웨어 구현에 있어서 기본적으로 반복 실행되지만 서로 독립적이라면 병렬로

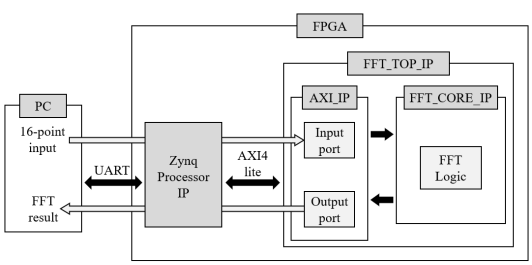

Fig. 3. Entire IP logic process for FFT accelerator

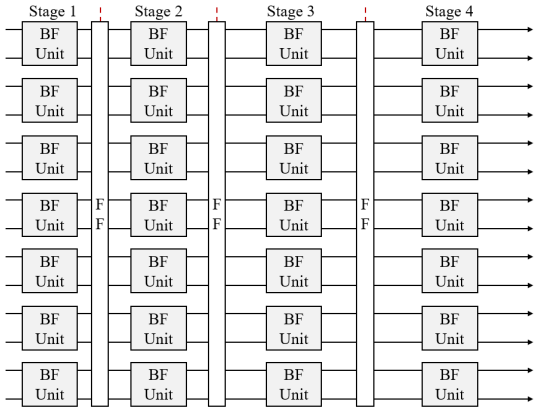

Fig. 4. Design structure of FFT_CORE_IP

한 번에 처리할 수 있게 설계가 가능한 구조를 의미한다. 따라서 구현과정에서 butterfly 연산을 해당 구조로 만들고 병렬처리가 가능하게 구현했다.

Fig.6.과 같은 파이프라인 기법을 활용하기 위해 각 stage의 연산이 완료되면 Flip-Flop(FF) 레지스터를 활용해서 연산의 중간 결과들을 저장해서 최적화를 진행했다. 각 stage마다 8번의 butterfly 연산을 수행하며, 이는 2개의 입력 값에 대해 연산을 진행한다. 중간에 레지스터를 활용함으로써 stage의 중간 결과값이 저장되며, 만약 연속적으로 값이 들어오게 된다면, 파이프라인의 효과를 극대화할 수 있다. 또한 각 연산에 필요한 회전자의 값을 미리 계산하여 9비트 파라미터로 저장함으로써, 연산 과정에서 필요한 복잡한 계산을 최소화할 수 있다.

이러한 방식은 전체 연산의 처리 속도와 효율적인 연산의 핵심적인 역할을 하며, 특히 독립적이고 반복적인 연산이 필요한 알고리즘에 효과적이다.

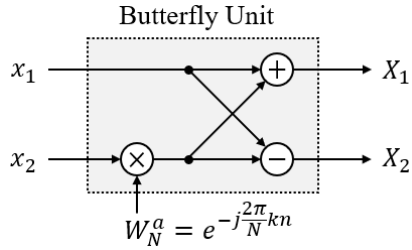

Fig. 5. Structure of butterfly unit

4.2 Processing System 구현

PL 파트에서 로직을 사용하지 않고 PS 파트에서

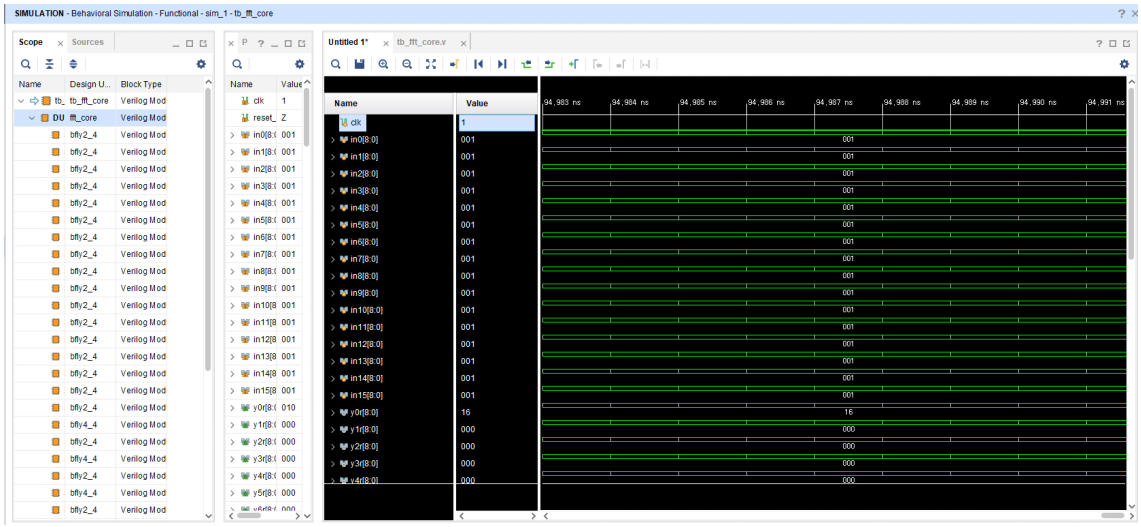

Fig. 6. Testbench result of the proposed FFT accelerator implementation on FPGA

FFT 연산을 수행하는 어플리케이션 코드의 구현은 Fig.8.과 같이 Zynq에서 제공하는 프로세서만을 사용한다. 해당 구현과 4.1의 PL 파트의 연산 구현과 동작 시간의 비교 분석을 통해 가속화 정도를 검증한다.

PS에서 구현된 FFT 연산은 PL에서 구현한 FFT와 동일한 기능을 하도록 radix-2 16 points로 설계되었으며, FPGA에 내장된 667MHz dual-core Cortex-A9을 사용하여 연산을 수행한다.

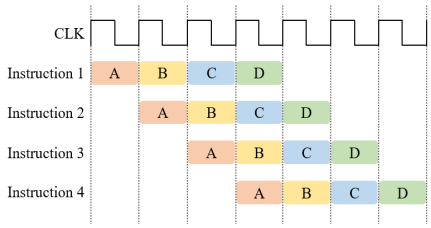

Fig. 7. Example of acceleration using pipeline structure

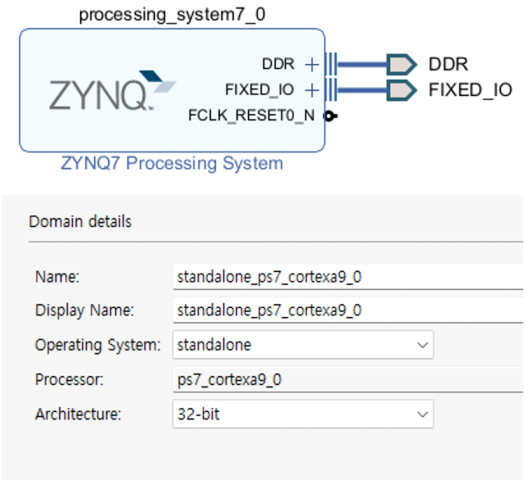

Fig. 8. Zynq processor IP in PS part

V. 구현 결과 및 비교 분석

본 장에서는 4장에서 언급한 내용을 바탕으로 FFT 가속기의 구현 결과와 관련 연구에서 제안한 디자인의 리소스 사용량을 비교 및 분석한다. 같은 수의 input point에 대한 FFT 구현 연구들을 비교 대상으로 하였다. 16개 이상이면서 2^n개의 input point(32 points, 64 points 등)를 대상으로 하는 구현의 경우, 입력 크기의 증가에 따라 자연스럽게 리소스가 증가하기 때문에 동일한 조건이 아니라고 판단하여 비교 대상에서 제외했다.

Fig.7.는 FFT 연산 모듈의 구현 정확성을 확인하는 테스트 벤치의 결과이다. Input point의 값으로 〔1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1〕라는 16개의 데이터를 보냈을 때, 연산의 결과 값으로 〔16,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0〕이 정상적으로 출력됨을 확인할 수 있다. 테스트 벤치 상에서는 정수 16개 input에 해당하는 output을 결과로 출력했지만, 실제 구현에서는 input과 output 모두 실수와 허수로 나눠져 연산이 진행된다.

Fig.9.은 구현한 FFT 연산의 전체 block design을 나타낸 것이다. 테스트 벤치에서 값이 정상적으로 출력이 되는 것을 확인 한 후에 Fig.3.와 같이 ZYNQ 프로세서 IP와 AXI4_lite가 포함된 FFT_TOP_IP가 결합되는 형태로 구성되어 있다.

Table 1.은 구현한 FFT_CORE 모듈을 ZYNQ 프로세서 IP 결합하여 합성 및 구현을 진행할 때 사용된 resource를 보여준다. 대표적으로 사용되는 LUT(Look Up Table)과 FF(Flip-Flop)의 사용량은 1% 미만으로 전체 주어진 리소스에 비해 충분한 사용량을 보인다.

연산과정에서 소요된 시간은 1,000회 반복 실험 후 평균값을 사용했으며 그 결과는 Table 2.와 같다. PS 에서의 연산시간이 PL에서의 연산시간보다 평균적으로 약 3 *us* 길었다. 또한 소요 clock cycle은 약 1000 cycle 정도의 차이가 발생했다.

Table 3.은 본 논문에서의 구현 방식과 유사한 연구들을 비교한 표이다. 비교 과정에서 각 연구마다 사용한 FPGA 보드, FFT 기법, 하드웨어 성능 등이 달라서 정확한 비교는 어렵지만, 제안한 구현과

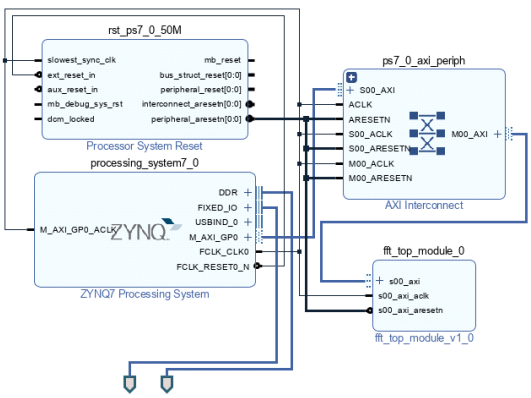

Fig. 9. Block design of the proposed FFT accelerator

Table 1. Detailed resource utilization of the proposed FFT accelerator design

resource	utilization	available	utilization (%)
LUT	497	53,200	0.93
LUT RAM	60	17,400	0.34
FF	802	106,400	0.75
BUFG	1	32	3.13

Table 2. Computation time of PS, PL part

Avg.	PS	PL
time (us)	21	18
Clock cycle	14,236	13,396

같은 수의 input point를 갖는 연구들에서 사용된 주요 리소스와 비교하면 LUT는 최소 50.5%에서 73.8% 감소된다. 또한 FF의 경우 [6]에 비해 4.1%, [4]에 비해 76.9% 감소됐다. [3]과 비교했을 때 증가했지만, IOB(Input Output Block) 수는 55.9% 감소했음을 확인할 수 있다. LUT 및 FF의 개수를 합한 전체 리소스(area)의 경우, 본 논문에서 제시한 구현의 area가 [3]에 비해 16.78%, [4]에 비해 75.78%, [6]에 비해 30.72% 감소하는 것을 확인할 수 있다. [5]의 경우 fully used LUT-FF의 정보만 언급했기에, LUT와 FF으로 구성된 slice의 개수를 보면 본 논문에서 제시한 방식이 결과적으로 적은 리소스를 사용했다고 할 수 있다.

VI. 결 론

본 논문에서는 격자 기반 양자내성암호에서 많은 연산량을 차지하는 FFT 연산에 대해 FPGA를 활용하여 빠르고 효율적으로 구현하였다. Table 2.에서 볼 수 있듯이, PL을 활용한 FFT 연산이 PS에

Table 3. Comparison of resource utilization in FFT implementation between related works and the proposed design

Method	FPGA	Radix	N	No. LUT	No. FF	No. IOB	No. slice
Proposed	Zybo z7-20	r-2	16	497	802	130	-
[3]	Spartan6	r-2	16	1,004	557	295	-
[4]	XC5VLX330-2FF1760	r-4	16	3,469	1,894	-	1,096
[5]	(VHDL language)	r-2	16	(fully used LUT-FF pairs) 295		51	2,402
[6]	Zynq 7z020clg484-1	r-4	16	1,039	836	-	-

서만 처리되는 것보다 빠르게 처리된다. 이는 FPGA의 고성능 연산 처리, 높은 병렬처리 등의 장점을 활용하여 성능 향상을 이룬 것으로 해석할 수 있다. 하지만 본 논문에서 실험한 FFT 연산의 동작은 한 번 연산하는 시간을 비교한 것이므로 큰 차이는 없다. 이는 반복적으로 FFT 연산을 진행할 때 연산 시간의 차이가 커질 것으로 사료된다. 또한 유사한 논문과의 결과 비교를 통해 본 연구의 구현 방식이 다른 연구들과 비교하여 리소스 사용에 있어서 효율성을 보인다. 이러한 결과는 본 논문에서 제시한 FPGA기반의 FFT 구현 방식이 다양한 분야에서 빠른 연산 처리와 효율적인 리소스 사용이 요구되는 알고리즘에 적용될 수 있음을 시사한다. 리소스 사용량의 비교 외에 FFT 연산의 동작 시간의 비교는 사용하는 하드웨어의 클럭 주파수에 따라 큰 차이를 보이므로 그에 대한 내용은 기술하지 않았다.

본 논문에서 제안한 FFT 연산 최적화 방식과 동일하게 양자내성암호 알고리즘 중 FALCON-512 키 생성 과정에 적용하면, input point의 개수를 증가시킨 형태로 512개의 input point를 갖는 FFT 연산에 대해 가속화할 수 있다. FALCON-512에서 가장 많은 연산 리소스를 차지하는 FFT 연산에 대해 최적화를 진행한다면, 알고리즘 가속에 큰 영향이 있을 것 이라고 판단한다. 향후 FALCON-512 알고리즘 키 생성 과정에서 사용되는 FFT 연산에 대해 최적화 구현하는 연구를 진행할 예정이다.

References

〔1〕 K.S. Hemmert and K. D. Underwood, “An analysis of the double-precision floating-point FFT on FPGAs.” 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM′05), IEEE, pp. 171-180, Apr. 2005.

〔2〕 N.H. Nguyen, S.A. Khan, C.H. Kim and J.M. Kim, “A high-performance, resource-efficient, reconfigurable parallel-pipelined FFT processor for FPGA platforms.” Microprocessors and Microsystems, vol. 60, pp. 96-106, Jul. 2018.

〔3〕 Lakshmi S, and Anoop Thomas, “Implementation of radix 2 and radix 2^2 FFT algorithms on Spartan6 FPGA.” 2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT), IEEE, pp. 1-4, Jul. 2013.

〔4〕 A. Mankar, A.D. Das and N. Prasad, “FPGA implementation of 16-point radix-4 complex FFT core using NEDA.” 2013 Students Conference on Engineering and Systems (SCES), IEEE, pp. 1-5, Apr. 2013.

〔5〕 Saenz. S. Josue, Susana Ortega Cisneros and Jorge Rivera Dominguez, “FPGA design and implementation of radix-2 fast Fourier transform algorithm with 16 and 32 points.” 2015 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC), IEEE, pp. 1-6, Nov. 2015.

〔6〕 G. Akkad, A. Mansiur, B.E. Hassan, F. Le Roy and M. Najem, “FFT radix-2 and radix-4 fpga acceleration techniques using hls and hdl for digital communication systems.” 2018 IEEE international multidisciplinary conference on engineering technology (IMCET), IEEE, pp. 1-5, Nov. 2018.

〔7〕 V. Patil and T.M. Manu, “Fpga implementation radix-2 dit fft using fixed point arithmetic and reduced arithmetic complexity.” 2021 International Conference on Intelligent Technologies (CONIT), IEEE, pp. 1-4, Jun. 2021.

〈저자소개〉

이 규 섭 (Gyu Sup Lee) 학생회원
2022년 2월: 한양대학교 ERICA 캠퍼스 전자공학부 졸업
2022년 3월~현재: 한양대학교 전자공학과 석사과정
〈관심분야〉 FPGA 구현, 양자내성암호, 임베디드 시스템 보안

조 성 민 (Seong-Min Cho) 학생회원
2019년 2월: 한양대학교 ERICA 캠퍼스 전자공학부 졸업
2019년 3월~현재: 한양대학교 전자공학과 석박사통합과정
〈관심분야〉 IoT 보안, 임베디드 시스템 보안, 양자 내성 암호

서 승 현 (Seung-Hyun Seo) 종신회원
2000년 2월: 이화여자대학교 수학과 졸업
2002년 2월: 이화여자대학교 컴퓨터학과 공학석사
2006년 2월: 이화여자대학교 컴퓨터학과 공학박사
2006년 5월~2006년 11월: 고려대학교 정보보호대학원 BK21 사업단 연구전임강사
2006년 12월~2010년 2월: 금융보안연구원 주임연구원
2010년 2월~2012년 2월: 한국인터넷진흥원 선임연구원
2012년 2월~2014년 5월: 미국 퍼듀대학교 컴퓨터학과 박사후연구원
2014년 6월~2015년 2월: 고려대학교 정보보호대학원 BK21+ 사업단 연구교수
2015년 3월~2017년 2월: 고려대학교 세종캠퍼스 수학과 조교수
2017년 3월~2020년 2월: 한양대학교 ERICA 캠퍼스 전자공학부 부교수
2020년 3월~현재: 한양대학교 ERICA 캠퍼스 전자공학부 교수
〈관심분야〉 암호프로토콜, 암호이론, IoT 보안, 블록체인 보안, 악성 코드 분석, 양자 내성 암호