FPGA board를 통한 시스템 검증용 1D-CZP 패턴의 구현

박정환*, 장원우*, 이성목*, 김주현**, 강봉순*

*동아대학교 전자공학과

**삼성전기 주식회사

Implemention of 1D-CZP pattern for system verification through FPGA board

Jung-Hwan Park*, Won-woo Jang*, Sung-mok Lee*, Joo-Hyun Kim**, Bong-soon Kang*

*Department of Electronic Engineering, Dong-A University

** SAMSUNG Electro-Mechanics Co. Ltd

E-mail : drum0103@didec.donga.ac.kr

요 약

본 논문에서는 알고리즘의 테스트 패턴중 하나인 1D-CZP패턴의 하드웨어 구현을 제안한다. FPGA를 통한 알고리즘 검증 시 센서로부터 받아들이는 정보로만은 알고리즘의 완벽한 정상작동 유무를 판단하기 어렵기 때문에, 내부 패턴 Generator를 사용하여 센서의 정보와 함께 알고리즘의 정상작동 유무를 판단하게 된다. 본 논문은 필터의 주파수 특성 판단에 용이하며, 입력이 랜덤한 특징을 가지는 1D-CZP패턴을 ROM Table형태로 구현하며, 구현 시 사용되는 Modulus연산을 효율적으로 수정함으로, 하드웨어 사이즈가 작아진 1D-CZP패턴을 제안한다.

ABSTRACT

In this paper, we propose the 1D-CZP pattern for FPGA verification. The algorithm that was implemented by Verilog-HDL on FPGA board is verified before the chip is producted. Input through the external sensor might not be enough to verify the algorithm on FPGA board. Hence, both external input and internal input can lead the verification of the algorithm. This paper suggests the hardware implementation of compact 1D-CZP pattern that has the random input. It is useful to analyze the characteristics of the filter frequencies and organized as ROM Table which is efficient to Modulus operation.

키워드

FPGA, CZP, Circular Zone Plate, Modulus, Hardware

Ⅰ. 서 론

멀티미디어 장치와 mobile장치의 Slim화 소형화에 대한 관심이 점차 커짐으로 인해, 이를 가능하게 하는 SoC(System on Chip)에 대한 연구가 지속적으로 이루어지고 있다. SoC의 최종 목표인, 하나의 알고리즘을 Chip에 적용하기 위해선 많은 단계를 거치게 된다. 먼저, 필요한 알고리즘을 개발하고 Verilog-HDL과 같은 하드웨어 언어로 알고리즘을 구현한다. 다음으로 Chip으로 제작하기 전에 FPGA 검증을 통하여 필요 없는 회로와 게이트를 제거한 후, 최종적으로 Chip으로 제작하게 된다. FPGA 검증 과정은 현재 구현되어 있는 알고리즘이 하드웨어로 구현하였을 때, 정상작동하는지를 확인하는 과정으로, 센서로부터 실제 입력 데이터를 받아서, 출력 데이터가 원하는 대로 나오는지 확인한다. 하지만 예상되는 수많은 패턴을 일일이 다 검증하기엔 시간적으로나 인력적으로 비효율적이다. 그러므로, 알고리즘을 검증하기 위해 일반적으로 CZP(Circular Zone Plate)패턴을 사용한다. CZP패턴은 데이터의 입력이 랜덤하다는 특징이 있기 때문에 예상치 못한 입력에 대한

고려가 가능하다는 장점을 가지고 있다. 또한, 전 주파수 대역의 정보를 포함하고 있으므로 알고리즘에 사용된 필터의 특성을 확인 하는데 유용하다. 이에 본 논문에서는 1D-CZP패턴을 하드웨어로 구현하여, FPGA 내부 패턴 Generator에 포함시킴으로서, 센서로부터 받아들이는 외부 입력 외에 내부 패턴으로 알고리즘의 정상작동 유무를 검증할 수 있도록 설계되었다. 본 논문은 Ⅱ장에서 1D-CZP패턴의 생성방법에 대해 언급하며, Ⅲ장에서 하드웨어 구조 및 성능에 대해 언급할 것이며, Ⅳ장에서 결론을 맺는다.

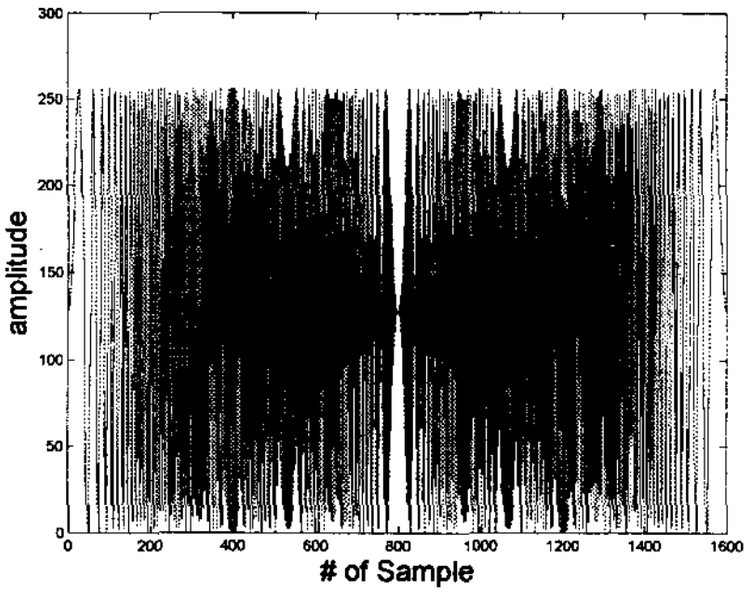
Ⅱ. 본 론

1. 1D-CZP(1 Dimension Circular Zone Plate)

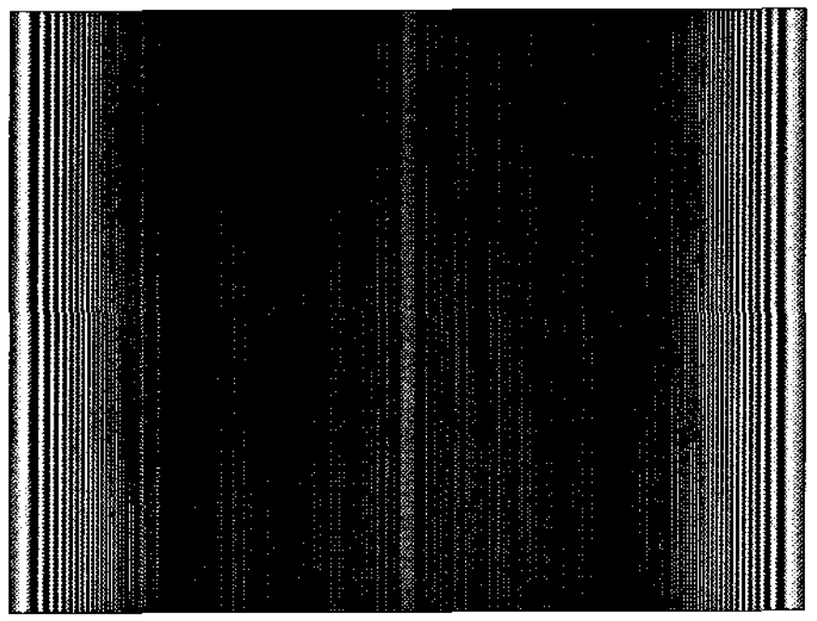
1D-CZP패턴은 일반적으로 데이터 입·출력의 정확성을 검증하거나, 필터의 주파수 특성을 파악하기에 유용한 테스트 패턴으로 사용되며, 다음의 수식을 사용하여 생성된다.

$$CZP = amplitude \times \sin(cxX^2) + amplitude \quad (1)$$

수식 1의 amplitude는 CZP패턴의 진폭을 나타내며 cx는 CZP패턴의 주기, X는 샘플의 개수를 나타낸다. 그림 1의 (a)는 수식 1을 사용하여 1D-CZP패턴을 생성한 것이며, 그림 1의 (b)는 (a)에서 생성된 1D-CZP패턴을 FPGA 내부 입력 패턴의 height에 맞게 확장한 결과를 나타낸 것이다. 그림 1의 (a)와 같은 1D-CZP 패턴은 0~255의 랜덤한 데이터가 입력되므로 데이터의 입·출력을 통한 알고리즘의 정상작동 유무를 판단할 수 있다. 또한, CZP패턴은 전주파수 대역을 포함하고 있기 때문에, CZP패턴을 필터에 통과 시키면 그 필터의 특징을 알아 볼 수 있다.



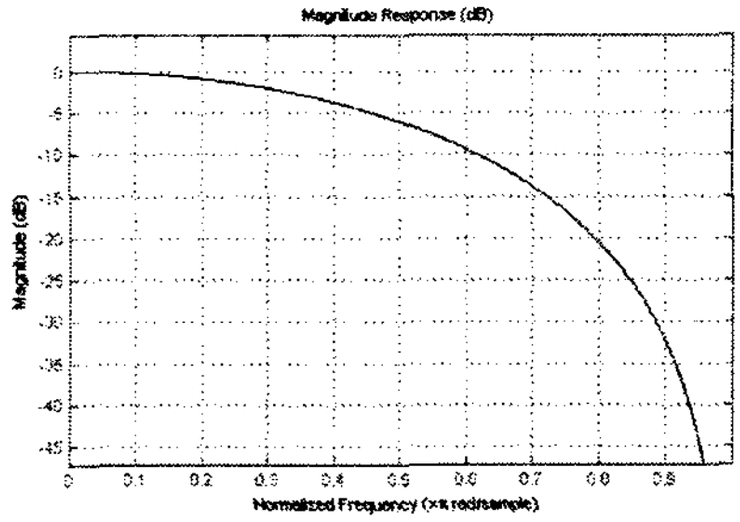
(a) amplitude : 128, cx : $\frac{\pi}{1600}$, 1<X<1600



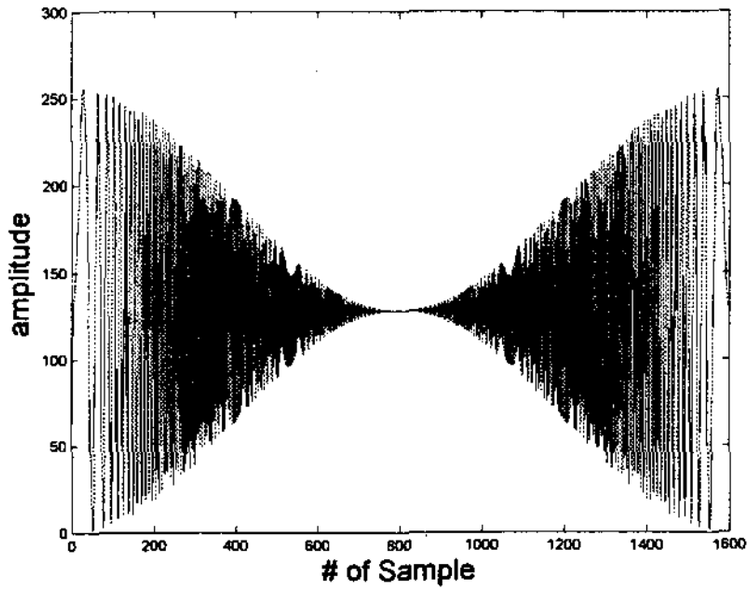
(b) FPGA 내부 패턴 이미지 2M(1600x1200)

그림 1. 1D-CZP패턴의 생성

그림 2의 (a)는 Low-Pass Filter의 효과를 가지는 [1 2 1]/4의 계수를 가지는 필터로, 저주파는 통과시키고, 고주파는 억제하는 특징을 가지고 있다. 그림 1의 (a)의 입력 CZP패턴이 저주파는 통과되고 고주파는 억제된 결과를 그림 2의 (b)에서 확인할 수 있다. 그림 3의 (a)는 High-Pass Filter의 효과를 가지는 [-1 2 -1]의 계수를 가지는 필터로, 저주파 억제의 특징을 가진다. 저주파가 억제되고 고주파가 통과된 결과를 그림 3의 (b)에서 확인할 수 있다.

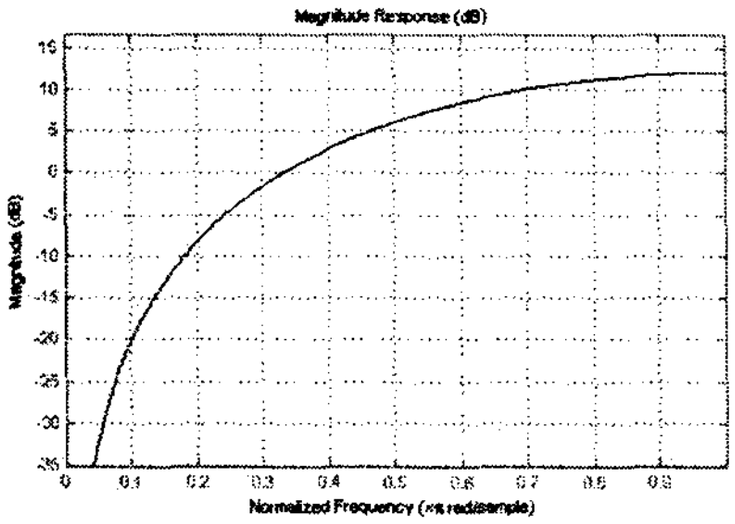


(a) [1 2 1]/4의 계수를 가지는 LPF

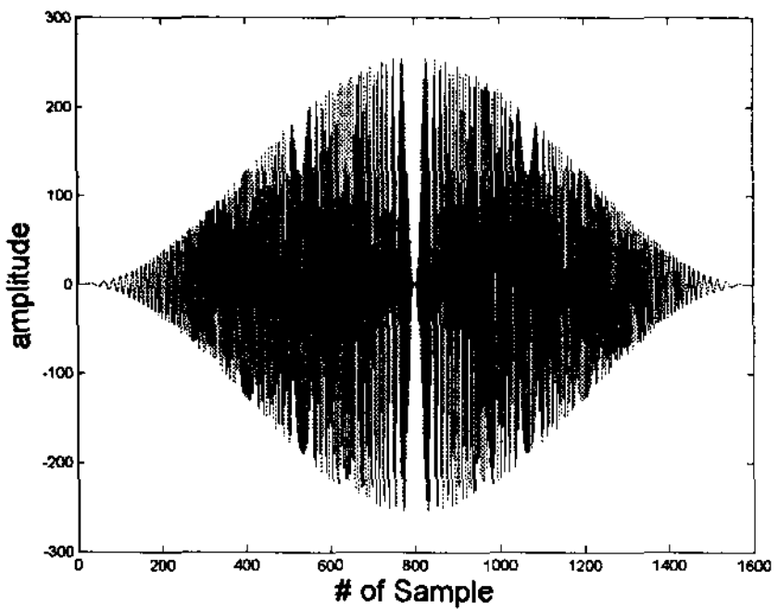


(b) [1 2 1]/4 필터에 통과된 입력 CZP

그림 2. CZP패턴을 사용한 필터의 주파수 특성 검증(LPF)



(a) [-1 2 -1]의 계수를 가지는 HPF



(b) [-1 2 -1] 필터에 통과된 입력 CZP

그림 3. CZP패턴을 사용한 필터의 주파수 특성 검증(HPF)

2. ROM Table의 생성과 Modulus

1D-CZP패턴을 생성하는 수식 1을 하드웨어로 구현하기 위해선 sine을 구현하여야 한다. sine을 구현하는 방법은 일반적으로 두 가지 방법이 있는데, 하나는 그림 4의 sine 곡선을 Piecewise Linear방법으로 여러 구간의 직선으로 제작하여 해당하는 Index에 맞게 sine값을 구하는 것이며, 다른 방법으로 미리 Index에 해당하는 sine값을 ROM Table에 저장해 놓고, 해당하는 Index에 맞게 sine값을 구하는 방법이다. 두 방법은 장단점이 있는데, Piecewise Linear방법은 sine 곡선을 여러개의 직선으로 대체하는 부분이 포함되므로 이 부분에서 오차가 발생한다는 단점이 있지만, 최종 완성된 하드웨어의 크기는 ROM Table방법에 비해 작다는 장점이 있다. 하지만 ROM Table을 통한 방법을 사용하면 하드웨어 크기는 많아지지만 만들기에 용이하며 오차율이 낮다는 장점이 있다. 본 논문에서는 데이터의 정밀함을 요구하는 테스트 패턴인 CZP패턴을 구현하기 때문에 ROM Table 방법을 사용하였다. 예를 들어 그림 4의 sine 곡선을 ROM Table로 구현하게 되면, Index 1이 입력으로 들어오면 출력되는 sine값은 128이 되고, Index가 400이 되면 sine 값은 255가 된다.

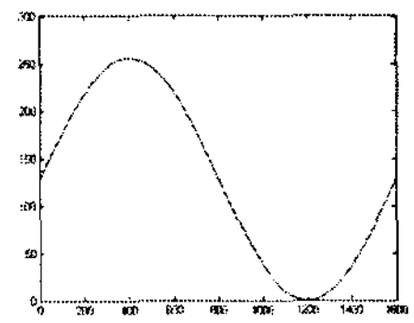


그림 4. 일반적인 sine 곡선

ROM_Table을 완성하였다면, 이에 맞는 Index를 구하는 방법이 필요하다. 일반적으로 Index는 수식 2와 같이 구할 수 있다.

$$Index = Modulus(counter, n\_sample) \quad (2)$$

수식 2의 counter는 n_sample만큼 증가하는 수이며, n_sample은 사용될 이미지의 width가 된다. 이 두 입력으로 Modulus, 즉 나머지 연산을 통해 Index를 구하게 된다. 본 논문에서 구현한 CZP패턴은 1.3M, 2M의 테스트 이미지를 목적으로 구현하였으므로, width가 각각 1280, 1600이 된다. 즉, 수식 3, 4와 같은 연산을 통해 Index를 구하게 된다.

$$Index(1.3M) = Modulus(counter, 1280) \quad (3)$$
$$Index(2M) = Modulus(counter, 1600) \quad (4)$$

이렇게 구해진 Index는 ROM_Table의 입력으로 Index에 해당하는 sine값을 출력하게 된다.

III. 하드웨어 구현 및 결과

그림 5는 본 논문에서 제안한 CZP패턴의 하드웨어 구조를 나타낸 개념도이다.

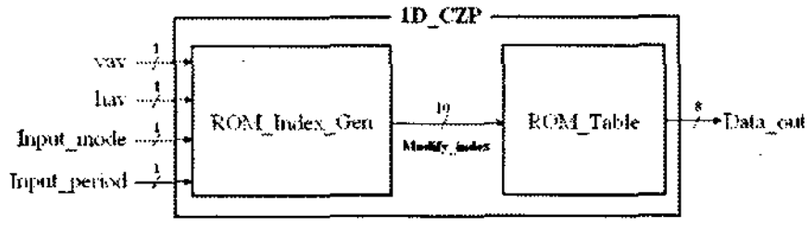


그림 5. 1D_CZP 패턴의 하드웨어 구조 개념도

하드웨어로 구현된 FPGA 내부 패턴인 CZP패턴은 외부의 데이터를 사용하지 않고 내부의 counter를 사용하여 내부 패턴을 직접 생성하는 것이므로, 입력은 신호의 동기를 맞춰주기 위한 hav, vav가 입력되고, CZP패턴의 주기를 정해주는 Input_period, 패턴의 크기를 지정해주는 Input_mode를 입력으로 받는다. 표 1은 Input_mode와 Input_period의 옵션에 관한 내용이다. 제안한 1D-CZP 패턴은 2주기 까지 선택이 가능하도록 설계되었다.

표 1의 입력신호는 그림 6의 ROM_Index_Gen 블록의 입력으로 들어간다.

표 1. 1D-CZP Pattern Input Signal

Input Signal		Description
Input_mode	0	1.3M(1280x1024)
	1	2M(1600x1200)
Input_period	0	1주기
	1	2주기

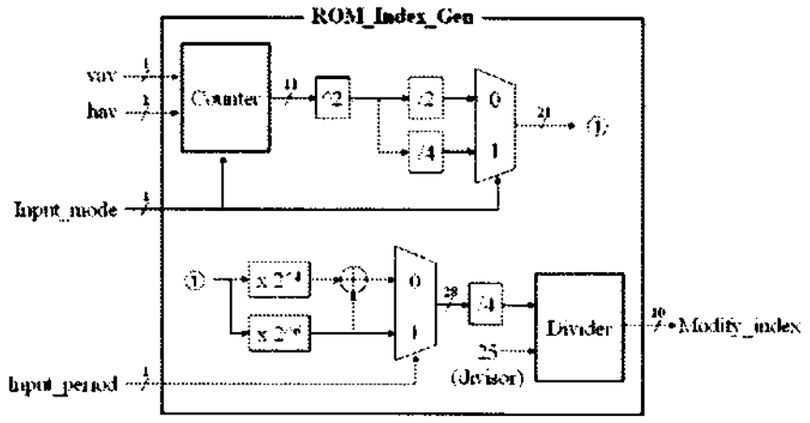


그림 6. ROM_Index_Gen Block Diagram

그림 6은 그림 5의 ROM_Index_Gen 블록을 세부적으로 그린 블록도로, ROM Table에 사용될 Index를 구하는 부분이다. 내부 Counter의 값을 제곱한 뒤, 이를 입력받은 주기에 맞게 재배열 하고, mudulus 연산을 통해 ROM Table에 사용될 Index를 뽑아내게 된다. Index를 얻기 위해 수식 3, 4를 사용하게 되는데, counter를 수식 3과 4와 같이 1280, 1600으로 나누게 되면 나누는 bit의 크기가 커지게 되어 하드웨어 크기가 커지는 원인이 된다. 그렇기 때문에 나누는수를 2^n의 형태로 유지하면 bit shift로 나눗셈을 구현할 수 있기 때문에 하드웨어 크기를 줄일 수 있다. 즉, 1280에 0.8을, 1600에 0.64를 각각 곱해주면, 나누는수가 1024로 통일되어 10bit right shift를 통해 구현이 가능하다. 수식 5와 6은 변경된 Modulus연산을 나타낸다.

$$\text{Index(1.3M)} = \text{Modulus(counter*0.8, 1024)} \quad (5)$$
$$\text{Index(2M)} = \text{Modulus(counter*0.64, 1024)} \quad (6)$$

소수점을 counter에 바로 곱하기는 하드웨어적으로 불가능하므로, 먼저 0.8과 0.64에 각각 100을 곱하여 이를 shifter and adder형태로 바꾸는 형태를 가지게 된다.

즉, counter × 80은 counter × 2^6+counter × 2^4으로 나타낼 수 있으며, counter × 64는 counter × 2^6으로 나타낼 수 있다. 100을 곱해서 연산된 결과이므로, 결과에 100을 다시 나누어 주어야한다. 100은 4 × 25로 구현이 가능하기 때문에, 4는 2^2이므로 shifter연산을 사용하면 되고, 25는 divider를 사용하여 최종 Index를 구하게 된다 [1]. 즉 수식 3, 4를 수식 5, 6으로 바꿈으로, 1280, 1600을 나누지 않고 bit shift후, 25로 나눔으로 하드웨어 크기를 줄일 수 있다. 그림 5의 ROM Table은 수식 7을 사용하여 구현한다.

$$\text{ROM Table} = 128 \times \sin((2 \times \pi \times x2)/1024)+128 \quad (7)$$

수식 7의 x2는 1에서 1024의 값을 가지는 값이다. 수식 7에 의해 생성된 ROM Table은 1024개의 Index를 가지며, ROM_Index_Gen 블록의 출력 결과인 Modify_index에 의해 선택되어 sine값을 출력하게 된다.

표 2은 본 논문에서 제안한 CZP패턴의 하드웨어 합성 결과이다.

표 2. 구현된 CZP패턴의 하드웨어 합성 결과

Operation Frequency(Mhz)	Gate Count	Max timing (ns)
40	9,154	18.16
50	9,154	18.16
60	9,224	15.77

제안한 하드웨어 구조는 하드웨어 언어인 Verilog_HDL로 설계 및 검증한 뒤, TSMC 0.25um라이브러리를 사용하여 합성하였다. 합성 결과는 표 2에서 볼 수 있듯이, 60Mhz동작을 만족하며, 60Mhz합성 시 9,224개의 Gate Count를 가진다.

Ⅳ. 결 론

FPGA를 통한 알고리즘 검증용 테스트 패턴 중 1D-CZP패턴을 하드웨어로 구현하기 위해 가장 중요한 점은 sine을 하드웨어로 구현하는 방법이다. 본 논문에서 제안한 방법은 ROM_Table을 사용하며, Index를 구하는 방법으로 Modulus를 사용하였다. 또한 Modulus연산시, 나누는 수를 2^n으로 바꾸어 shift연산을 수행함으로써, 사용되는 나눗셈기의 크기를 줄임으로, 전체 CZP패턴의 하드웨어 크기를 줄일 수 있었다. 이렇게 구현된 CZP패턴은 FPGA의 내부 패턴 Generator에 포함시킴으로써, 외부 입력을 제외한 내부 패턴으로 입·출력의 정상작동 유무, 사용된 필터의 특성 유무를 판단하는데 유용하게 사용될 수 있다.

감사의 글

본 연구에서 사용된 설계용 소프트웨어는 IDEC을 통하여 지원 받았습니다.

참고문헌

[1] Kyung-rin Kim, Jung-hwan Park, Kang-joo Kim, Bong-soon Kang, "Design of High Speed Divider for Multimedia System based on Mobile Application" The 16th Conference on KICS, IEEK, ICROS, pp. 63-66, Jun. 2007.