논문 2010-47TC-6-1

시스톨릭 어레이 구조와 CORDIC을 사용한 고속/저전력 Extended QRD-RLS 등화기 설계 및 구현

(Design and Implementation of Hi-speed/Low-power Extended QRD-RLS Equalizer using Systolic Array and CORDIC)

문 대 원*, 장 영 범**, 조 용 훈***

(Dae Won Moon, Young Beom Jang, and Yong Hoon Cho)

요 약

이 논문에서는 시스톨릭 어레이 구조를 갖는 고속/저전력 Extended QRD-RLS 등화기 구조를 제안한다. 기존의 시스톨릭 어레이 구조를 갖는 Extended QRD-RLS 등화기는 입력행렬의 QR분해를 위해서 벡터모드 CORDIC을 사용하여 벡터의 각도를 계산하고, 회전모드 CORDIC에서는 이 각도를 전달받아 벡터를 회전시킨다. 제안된 등화기 구조에서는 벡터모드 CORDIC과 회전모드 CORDIC이 정 반대방향으로 회전하는 것을 이용하여 구현 하드웨어의 크기를 현저히 감소시켰다. 이와 더불어 제안구조에서는 벡터모드 CORDIC과 회전모드 CORDIC을 동시에 동작함으로써 계산시간을 1/2로 감소시킬 수 있었다. 제안구조의 HDL 코딩과 칩 설계를 통하여 기존의 시스톨릭 어레이 구조와 비교하여 23.8%의 구현면적 감소를 확인하였다.

Abstract

In this paper, we propose a hi-speed/low-power Extended QRD-RLS(QR-Decomposition Recursive Least Squares) equalizer with systolic array structure. In the conventional systolic array structure, vector mode CORDIC on the boundary cell calculates angle of input vector, and the rotation mode CORDIC on the internal cell rotates vector. But, in the proposed structure, it is shown that implementation complexity can be reduced using the rotation direction of vector mode CORDIC and rotation mode CORDIC. Furthermore, calculation time can be reduced by 1/2 since vector mode and rotation mode CORDIC operate at the same time. Through HDL coding and chip implementation, it is shown that implementation area is reduced by 23.8% compared with one of conventional structure.

Keywords: equalizer, CORDIC, Extended QRD-RLS, systolic array structure

I. 서 론

적응 알고리즘을 사용하는 등화기는 전송된 데이터의 복원을 위한 효과적인 방법이다. 대표적인 적응 알고리즘으로는 LMS(Least Mean Square) 알고리즘과 RLS(Recursive Least Square) 알고리즘이 있다.

RLS 알고리즘은 뛰어난 수렴 성능 때문에 LMS 알고리즘에 비하여 많이 선호된다. 그러나 RLS 알고리즘은 무선 단말기의 비용 감소에 방해가 되는 엄청난 연산을 요구한다. 따라서 RLS 필터링의 연산을 줄이기 위하여 QRD(QR-Decomposition)를 사용한 행렬의 triangularization 방식이 소개되었다.[1~2] 또한 이와 같은 QRD-RLS 알고리즘의 효과적인 병렬처리 구현을 위하여 Givens 회전을 사용한 시스톨릭 어레이 구조가 제안되었다.[3~4] 시스톨릭 어레이는 입력 신호 행렬의 특정한 엘리먼트를 없애기 위한 벡터 회전을 수행한다. 회전 각도의 계산은 삼각함수 연산을 필요로 하는데 이를 위하여 CORDIC(COordinate Rotation DIgital

* 학생회원, 상명대학교 컴퓨터정보통신공학과
(Graduate School, Sangmyung University)
** 정회원, 상명대학교 공과대학 정보통신공학과
(College of Engineering, Sangmyung University)
*** 정회원, (주) 코메스타
(COMESTA.Inc)
※ 본 연구는 교육과학기술부와 한국연구재단의 지역혁신인력양성사업으로 수행된 연구결과임.
접수일자: 2009년8월31일, 수정완료일: 2010년6월3일

Computer) 알고리즘이 사용될 수 있다.[5~6]

CORDIC 알고리즘을 사용하는 시스톨릭 어레이 구조의 연산은 *shift*와 *adder/subtracter*를 사용하는 반도체로 효율적으로 구현될 수 있다. 이와 같은 CORDIC 시스톨릭 어레이 구조에서는 입력행렬의 QR 분해를 수행하는 RLS 알고리즘의 연산을 위하여 벡터모드 CORDIC 블록과 회전모드 CORDIC 블록을 사용한다.[7]

이 논문에서는 시스톨릭 어레이 구조의 PE (Processing Elements)인 벡터모드 CORDIC 블록과 회전모드 CORDIC 블록이 반대 방향으로 회전하는 것을 이용하여 구현면적을 감소시킨 구조를 제안한다. 이와 더불어 벡터모드 CORDIC 블록과 회전모드 CORDIC 블록이 동시에 동작하게 함으로써 연산 시간을 1/2로 감소시킬 수 있음을 보였다.

이 논문의 Ⅱ장에서는 등화기에 사용되는 QRD-RLS 알고리즘과 Extended QRD-RLS 알고리즘에 대하여 알아보고 Ⅲ장에서는 고속/저전력의 CORDIC 시스톨릭 어레이 구조를 제안한다. Ⅳ장에서는 HDL과 칩 구현을 통하여 제안 구조의 구현면적 감소 효과를 살펴본다.

Ⅱ. 등화기용 Extended QRD-RLS 알고리즘

1. QRD-RLS 알고리즘

QRD-RLS(QR Decomposition based Recursive Least Squares) 알고리즘이 고속 수신기에 많이 사용되기 위해서는 계산의 복잡도가 감소되어야 한다. 이 알고리즘에서 사용되는 표기는 다음과 같다.

$d(i)$: 송신되는 파일롯 신호

$\mathbf{w}(n)$: 구하려는 필터 계수 벡터

$\mathbf{u}(i)$: 수신된 파일롯 신호 벡터

λ^{n-i} : 과거 파일롯 신호의 영향이 감소되는 것을 나타내는 forgetting factor

여기에서 목표는 필터계수 벡터 $\mathbf{w}(n)$를 구하는 것이며 이 벡터는 다음의 비용함수를 최소화하도록 결정되어야 한다.

$$\epsilon = \| \mathbf{Q}(n)\Lambda(n)\mathbf{d}(n) - \mathbf{Q}(n)\Lambda(n)\mathbf{A}(n)\mathbf{w}(n) \|^2 \tag{1}$$

이 식에서 $\mathbf{Q}(n)$은 unitary 행렬이며, Λ는 망각계수 λ로 이루어진 행렬이다. 또한 $d(n)$은 송신된 파일롯 벡터이며 $\mathbf{A}(n)$은 수신된 파일롯 신호로 만들어진 Toeplitz 행렬이다. 위의 비용함수가 최소가 되기 위해서는 다음의 식을 만족하여야 한다.

$$\mathbf{R}(n)\mathbf{w}(n) = \mathbf{P}(n) \tag{2}$$

이 식에서 $\mathbf{R}(n)$은 상삼각행렬(upper triangular matrix)이고 벡터 $\mathbf{P}(n)$은 $\mathbf{Q}(n)\Lambda(n)\mathbf{d}(n)$의 윗부분이다. 상삼각행렬 $\mathbf{R}(n)$을 구하기 위해 다음의 Givens 회전을 사용하여 구할 수 있다.

$$\mathbf{J}_M(n)\cdots\mathbf{J}_2(n)\mathbf{J}_1(n)\mathbf{Q}'(n-1)\Lambda(n)\mathbf{A}(n) = \begin{bmatrix}\mathbf{R}(n)\\ \mathbf{0}\\ \mathbf{O}^T\end{bmatrix} \tag{3}$$

벡터 $\mathbf{P}(n)$도 역시 Givens 회전을 사용하여 다음과 같이 구할 수 있다.

$$\begin{bmatrix}\mathbf{P}(n)\\ \mathbf{V}(n)\end{bmatrix} = \mathbf{J}_M(n)\cdots\mathbf{J}_2(n)\mathbf{J}_1(n)\begin{bmatrix}\lambda\mathbf{P}(n-1)\\ \lambda\mathbf{V}(n-1)\\ d^*(n)\end{bmatrix} \tag{4}$$

이와 같이 $\mathbf{R}(n)$과 $\mathbf{P}(n)$을 구하였으므로 목표가 되는 벡터는 다음과 같이 구할 수 있다.

$$\mathbf{w}(n) = \mathbf{R}(n)^{-1}\mathbf{P}(n) \tag{5}$$

이 식에서 상삼각행렬 $\mathbf{R}(n)$의 역행렬을 구하는 계산의 복잡도를 줄이기 위해 다음 절에서 다루는 Extended QRD-RLS 알고리즘이 제안되었다.

2. Extended QRD-RLS 알고리즘

QRD-RLS 알고리즘에서 구한 $\mathbf{R}(n)$과 $\mathbf{P}(n)$을 update하는 식을 합하여 표현하면 다음과 같다.

$$\begin{bmatrix}\mathbf{R}(n) & \mathbf{P}(n)\\ \mathbf{0} & \alpha(n)\end{bmatrix} = \mathbf{Q}(n)\begin{bmatrix}\lambda\mathbf{R}(n-1) & \lambda\mathbf{P}(n-1)\\ \mathbf{u}^T(n) & d(n)\end{bmatrix} \tag{6}$$

망각계수를 곱하여 다음의 상삼각행렬을 새로 정의한다.

$$\tilde{\mathbf{R}}(n) = \begin{bmatrix}\lambda\mathbf{R}(n) & \lambda\mathbf{P}(n)\\ \mathbf{0} & \alpha(n)\end{bmatrix} \tag{7}$$

위의 행렬의 역행렬은 다음과 같이 표현될 수 있다.

$$\tilde{\mathbf{R}}^{-1}(n) = \begin{bmatrix}\lambda^{-1}\mathbf{R}^{-1}(n) - \mathbf{w}(n)/\alpha(n)\\ 0 \qquad\qquad 1/\alpha(n)\end{bmatrix} \tag{8}$$

직교변환 행렬인 $\mathbf{Q}(n)$이 상삼각행렬 $\mathbf{R}(n)$을 update하는 것과 같은 방법으로 하삼각행렬도 다음과 같이 update한다.

$$\begin{bmatrix}\tilde{\mathbf{R}}^{-T}(n)\\ \mathbf{v}(n)\end{bmatrix}=\mathbf{Q}(n)\begin{bmatrix}\tilde{\mathbf{R}}^{-T}(n-1)\\ 0\end{bmatrix} \tag{9}$$

Extended QRD-RLS 알고리즘은 상삼각행렬과 하삼각행렬의 두 개의 행렬이 다음과 같이 update된다.

$$\begin{bmatrix}\tilde{\mathbf{R}}(n) & \widetilde{\mathbf{R}^{-T}}(n)\\ 0 & \mathbf{v}(n)\end{bmatrix}=\mathbf{Q}(n)\begin{bmatrix}\tilde{\mathbf{R}}(n-1) & \widetilde{\mathbf{R}^{-T}}(n-1)\\ \widetilde{\mathbf{u}^{T}}(n) & 0\end{bmatrix} \tag{10}$$

위의 update 식에서 목표 벡터인 $\mathbf{w}(n)$은 상삼각행렬 $\mathbf{R}(n)$의 우측 아래 코너에 놓여있는 $\alpha(n)$과 하삼각행렬 $\mathbf{R}^{-1}(n)$의 맨 아랫줄을 곱하여 얻어진다. 위의 update 식이 시작되기 전에 초기 값은 다음과 같이 주어진다.

$$\mathbf{R}=\begin{bmatrix}\lambda & 1/\lambda & & \\ & \lambda & 1/\lambda & \\ & & \ddots & \ddots \\ & & \lambda & 1/\lambda\end{bmatrix} \tag{11}$$

Extended QRD-RLS 등화기에서 이 구조의 속도를 증가시키고 구현의 복잡도를 감소시키기 위한 구조를 다음 절에서 제안한다.

Ⅲ. 제안된 고속/저전력 Extended QRD-RLS 등화기

1. 제안된 고속/저전력 시스톨릭 구조

지난 절에서 기술된 Extended QRD-RLS 등화기는 5-by-5 크기의 경우에, update 식은 다음 그림과 같은 시스톨릭 어레이 구조로 구현될 수 있다.

그림 1의 시스톨릭 어레이 구조의 각각의 PE는 CORDIC 알고리즘을 사용하여 쉽게 구현될 수 있다.[7] 그림 1에서 원모양은 벡터모드로서 벡터의 직각 좌표가 입력되면 그 벡터의 각도와 크기가 계산된다. 네모 모양은 회전모드로서 벡터와 각도가 입력되면 각도만큼 회전된 후의 벡터의 좌표를 출력한다. 그림 1에서 첫

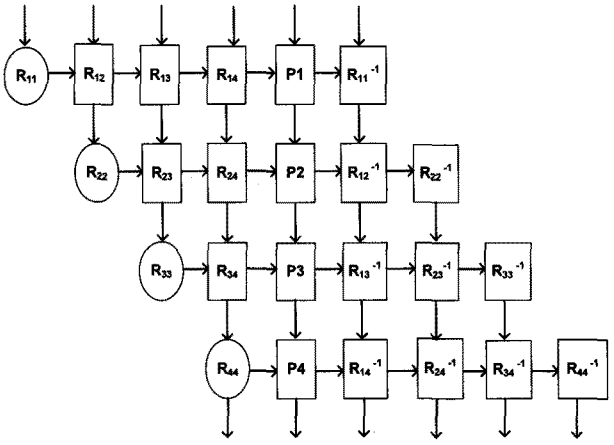
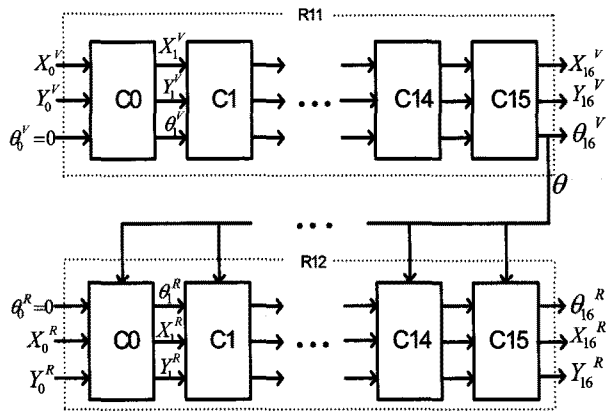
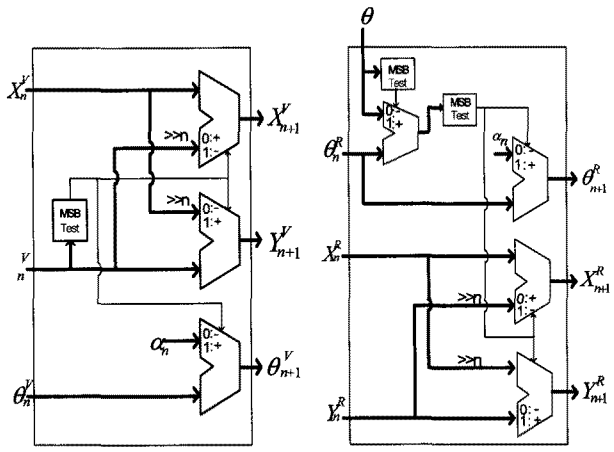


그림 1. 5-by-5 입력 행렬을 위한 확장 시스토릭 어레이 구조의 블록도

Fig. 1. Block diagram of Extended Systolic Array Architecture for 5-by-5 Input matrix.



(a)



(b) (c)

그림 2. 기본적인 R_{11}과 R_{12}의 구조, (a) 블록도, (b) R_{11}의 세부구조, (c) R_{12}의 세부구조

Fig. 2. Basic R_{11} and R_{12} structure, (a) Block diagram, (b) Cn of R_{11}, (c) Cn of R_{12}.

번째 clock에 R_{11}이 벡터의 각도를 계산하여 첫 번째 행의 나머지 5개의 회전모드 CORDIC에 그 각도를 제공한다. 두 번째 clock에서 5개의 회전모드 CORDIC은 입력벡터를 그 각도만큼 회전시켜 그 좌표를 아래의 PE에 제공한다. 벡터모드와 회전모드의 CORDIC PE는 모두 16스텝의 회전을 통해 결과 값을 출력한다고 가정하면 R_{11}과 R_{12}의 세부 구조는 그림 2와 같다. 그림 2에서 보듯이 R_{11}의 벡터모드 CORDIC은 예를 들어 (0.866, 0.5)의 벡터가 입력되면 16스텝의 회전을 통해 30도의 각도를 출력하게 된다. R_{12}의 회전모드 CORDIC은 이 30도를 입력으로 받아서 입력 벡터를 30도 만큼 회전시키게 된다.

그림 2에서 보듯이 벡터모드 R_{11}과 회전모드 R_{12}는 각각 벡터 update 블록, 각도 update 블록, 그리고 회전방향결정 블록으로 구성된다. 다만 R_{11}의 회전방형결정 블록은 벡터의 y 좌표를 사용하며, R_{12}의 회전방향 결정 블록은 각도를 사용하여 회전방향을 결정한다.

이때에 벡터모드 R_{11}의 16스텝 회전방향과 회전모드 R_{12}의 16스텝 회전방향은 정확히 반대가 된다. 이와 같은 아이디어는 주파수 옵셋 동기화기에서도 사용되고 있다.[8] 예를 들어 R_{11}에 (0.8437, 0.5369)의 벡터가 입력되면 이 벡터의 각도는 다음과 같이 3 스텝 회전으로 각도가 계산된다.

$$
\begin{aligned}
&C_0\text{블록: } Y_0 = 0.5369,\ MSB\ Test = 0(CW, -)\\
&X_1 = X_0 + Y_0\tan 45 = 0.8437 + 0.5369\times 1 = 1.3806\\
&Y_1 = Y_0 - X_0\tan 45 = 0.5369 - 0.8437\times 1 = -0.3068\\
&\theta_1 = \theta_0 - 45 = -45\\
&C_1\text{블록: } Y_1 = -0.3068,\ MSB\ Test = 1(CCW, +)\\
&X_2 = X_1 - Y_1\tan 26.565 = 1.3806 - (-0.3068)\times 0.5 = 1.534\\
&Y_2 = Y_1 + X_1\tan 26.565 = -0.3068 + 1.3806\times 0.5 = 0.3835\\
&\theta_2 = \theta_1 + 26.565 = -45 + 26.565 = -18.435\\
&C_2\text{블록: } Y_2 = 0.3835,\ MSB\ Test = 0(CW, -)\\
&X_3 = X_2 + Y_2\tan 14.036 = 1.534 + 0.3835\times 0.25 = 1.629875\\
&Y_3 = Y_2 - X_2\tan 14.036 = 0.3835 - 1.534\times 0.25 = 0\\
&\theta_3 = \theta_2 - 14.036 = -18.435 - 14.036 = -32.471
\end{aligned}
\tag{12}
$$

위의 예에서는 3 스텝에 계산이 끝나는 벡터를 사용하였다. 3 스텝에서 Y_3=0이 되었으므로 정확하게 벡터가 벡터 평면의 x축에 일치하였으며 이때의 각도가 32.471도 임을 알 수 있다. 즉 주어진 벡터의 각도는 32.471도이다. 이 각이 R_{12}에 제공되면 다음과 같이 벡터를 32.471도 만큼 회전시킨다. R_{12}의 입력벡터가 (0.9, 0.4)인 경우에 다음과 같이 회전한다. 여기에서 θ=32.471(양수)이므로 θ가 입력되는 ALU는 항상 subtracter로 동작한다.

$$
\begin{aligned}
&C_0\text{블록: } 0 - 32.471 = -32.471,\ MSB\ Test = 1(CCW, +)\\
&X_1 = X_0 - Y_0\tan 45 = 0.9 - 0.4\times 1 = 0.5\\
&Y_1 = Y_0 + X_0\tan 45 = 0.4 + 0.9\times 1 = 1.3\\
&\theta_1 = \theta_0 + 45 = 45\\
&C_1\text{블록: } 45 - 32.471 = 12.529,\ MSB\ Test = 0(CW, -)\\
&X_2 = X_1 + Y_1\tan 26.565 = 0.5 + 1.3\times 0.5 = 1.15\\
&Y_2 = Y_1 - X_1\tan 26.565 = 1.3 - 0.5\times 0.5 = 1.05\\
&\theta_2 = \theta_1 - 26.565 = -45 - 26.565 = -8.435\\
&C_2\text{블록: } 18.435 - 32.471 = -14.036\ MSB\ Test = 1(CCW, +)\\
&X_3 = X_2 - Y_2\tan 14.036 = 1.15 - 1.05\times 0.25 = 0.8875\\
&Y_3 = Y_2 + X_2\tan 14.036 = 1.05 + 1.15\times 0.25 = 1.3375\\
&\theta_3 = \theta_2 + 14.036 = -18.435 + 14.036 = 32.471
\end{aligned}
\tag{13}
$$

위와 같이 3 스텝을 거쳐서 벡터가 회전된다. 식 (12)는 -+-로 회전하였으며 식 (13)은 +-+로 회전함을 알 수 있다. -는 시계방향(clock wise, CW), +는 시계반대방향(counter clock wise, CCW)의 회전을 나타낸다. 이와 같이 벡터모드 R_{11}과 회전모드 R_{12}는 정반대 방향으로 회전함을 알 수 있다. 또 다른 예로서 R_{11}에 (0.866, 0.5)의 벡터가 입력되면 (-+-+ --+- ++-- +-++)의 방향으로 16번 회전하여 θ=30도의 출력을 R_{12}로 보낸다. 이때 R_{12}의 회전방향은 +30도를 회전시키기 위해 내부에서 (+-+- ++-+ --++ -+--)의 회전방향을 갖는다. 이 두 PE의 회전 방향을 비교해 보면 정 반대의 방향을 갖고 있음을 볼 수 있다. 즉 첫 번째 행의 R_{11} PE와 나머지 5개 PE는 정 반대 방향으로 회전하고 있음을 볼 수 있다. 따라서 R_{11} CORDIC은 각도를 5개의 회전모드 PE에 공급할 필요가 없으며 R_{12} 등의 첫 번째 행의 5개 PE는 R_{11}의 16스텝 회전을 모니터링하면서 반대 방향으로 회전하면 회전된 벡터 값을 얻을 수 있게 된다. 즉 벡터모드 R_{11}의 Cn은 각도 update 블록이 필요 없으며, 회전모드 R_{12}는 각도 update 블록과 회전방향결정 블록이 필요 없게 된 간단한 구조를 얻을 수 있다. 이와 같이 제안된 구조는 그림 3과 같다.

그림 3의 R_{11} 벡터모드의 블록 Cn의 내부구조는 각도 update 블록이 없다. 왜냐하면 R_{12}등의 회전모드 PE로 각도를 보낼 필요가 없기 때문이다. 따라서 제안된 Cn 블록은 그림 2의 Cn 블록과 비교하여 간단해진 회로가 된다. 예를 들어 벡터모드 R_{11}에 입력된 벡터가 (0.8437, 0.5369)의 경우를 살펴보자. 이때에 벡터모드

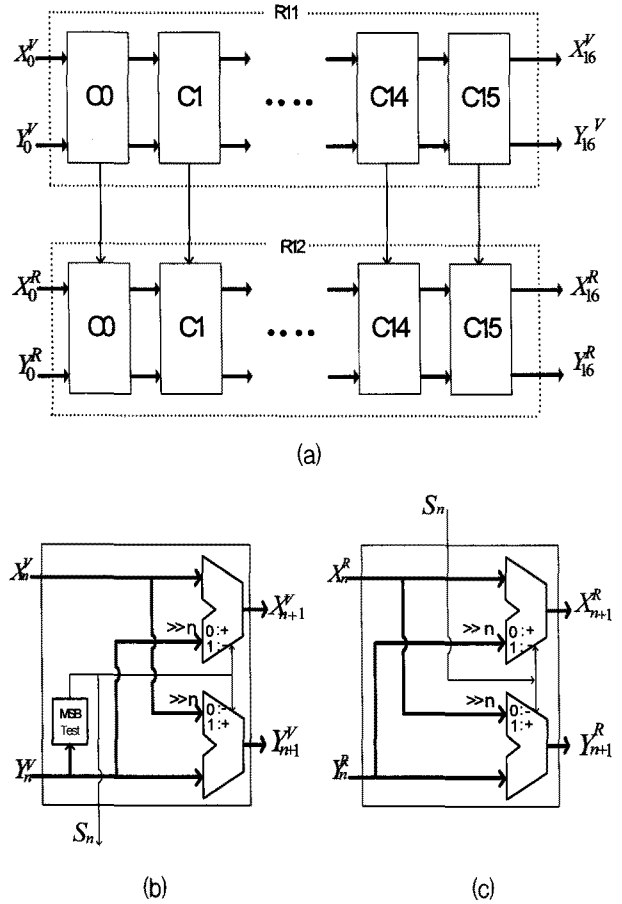


그림 3. 제안된 R_{11}과 R_{12}의 구조, (a) 블록도, (b) R_{11}의 세부구조, (c) R_{12}의 세부구조

Fig. 3. Proposed R_{11} and R_{12} structure, (a) Block diagram, (b) Cn of R_{11}, (c) Cn of R_{12}.

R_{11} PE의 3 스텝의 벡터 값은 다음과 같이 회전한다.

$$
\begin{aligned}
&C_0\text{블록}: Y_0 = 0.5369,\ MSB\ Test = 0(CW, -)\\
&X_1 = X_0 + Y_0\tan 45 = 0.8437 + 0.5369\times 1 = 1.3806\\
&Y_1 = Y_0 - X_0\tan 45 = 0.5369 - 0.8437\times 1 = -0.3068\\
&C_1\text{블록}: Y_1 = -0.3068,\ MSB\ Test = 1(CCW, +)\\
&X_2 = X_1 - Y_1\tan 26.565 = 1.3806 - (-0.3068)\times 0.5 = 1.534\\
&Y_2 = Y_1 + X_1\tan 26.565 = -0.3068 + 1.3806\times 0.5 = 0.3835\\
&C_2\text{블록}: Y_2 = 0.3835,\ MSB\ Test = 0(CW, -)\\
&X_3 = X_2 + Y_2\tan 14.036 = 1.534 + 0.3835\times 0.25 = 1.629875\\
&Y_3 = Y_2 - X_2\tan 14.036 = 0.3835 - 1.534\times 0.25 = 0
\end{aligned}
\tag{14}
$$

위의 식에서 보듯이 MSB Test 결과가 양수이면 0이므로 X는 adder로, Y는 subtracter로 동작한다. 위의 식에서 보듯이 식 (12)의 벡터모드 CORDIC에서 사용되는 각도 update 식은 사용되지 않는다.

R_{12}의 Cn 구조도 단순해진 것을 볼 수 있다. 그림 3에서 보듯이 R_{11} Cn의 회전방향 블록에서 받은 신호 비트를 사용하여 벡터의 회전 방향을 결정하면 된다. R_{12}의 입력벡터가 (0.9, 0.4)인 경우 다음과 같이 회전한다.

$$
\begin{aligned}
&C_0\text{블록}: (CCW, +)\\
&X_1 = X_0 - Y_0\tan 45 = 0.9 - 0.4\times 1 = 0.5\\
&Y_1 = Y_0 + X_0\tan 45 = 0.4 + 0.9\times 1 = 1.3\\
&C_1\text{블록}: (CW, -)\\
&X_2 = X_1 + Y_1\tan 26.565 = 0.5 + 1.3\times 0.5 = 1.15\\
&Y_2 = Y_1 - X_1\tan 26.565 = 1.3 - 0.5\times 0.5 = 1.05\\
&C_2\text{블록}: (CCW, +)\\
&X_3 = X_2 - Y_2\tan 14.036 = 1.15 - 1.05\times 0.25 = 0.8875\\
&Y_3 = Y_2 + X_2\tan 14.036 = 1.05 + 1.15\times 0.25 = 1.3375
\end{aligned}
\tag{15}
$$

이 예제에서 보듯이 제안된 벡터모드 PE와 회전모드 PE는 하드웨어 구현의 복잡도를 크게 줄일 수 있다. 기존의 구조와 비교하여 5-by-5 입력에 대하여 사용되는 블록의 감소는 다음과 같다.

제안 구조는 구현면적의 감소 뿐 아니라 계산속도도 빨라지게 된다. 5-by-5 경우에 총 소요 clock을 계산해 보기로 한다. 기존의 구조에서는 첫 번째 clock에 R_{11}의 PE가 입력 벡터의 각도를 계산하여 첫 번째 행의 나머지 5개의 회전모드 PE에 그 결과를 보낸다. 두 번

표 1. 제안구조의 사용된 블록의 수

Table 1. Number of block in proposed structure.

모드	벡터 update 블록	제안 구조	기존 구조
벡터모드 PE(4개)	벡터 update 블록	4	4
	각도 update 블록	0	4
	회전방향결정블록	4	4
회전모드 PE(20개)	벡터 update 블록	20	20
	각도 update 블록	0	20
	회전방향결정블록	0	20

표 2. 기존 시스톨릭 어레이 구조의 전체 동작 clock 수

Table 2. Number of clock for systolic array in conventional CORDIC structure.

7 5 3 1	8 6 4 2	8 6 4 2	8 6 4 2	8 6 4 2	8 6 4 2		
	9 7 5 3	10 8 6 4	10 8 6 4	10 8 6 4	10 8 6 4	10 8 6 4	
		11 9 7 5	12 10 8 6	12 10 8 6	12 10 8 6	12 10 8 6	12 10 8 6
			13 11 9 7	14 12 10 8	14 12 10 8	14 12 10 8	14 12 10 8

표 3. 제안된 시스톨릭 어레이 구조의 전체 동작 clock 수

Table 3. Number of clock for systolic array in proposed CORDIC structure.

4 3 2 1	4 3 2 1	4 3 2 1	4 3 2 1	4 3 2 1	4 3 2 1	
	5 4 3 2	5 4 3 2	5 4 3 2	5 4 3 2	5 4 3 2	5 4 3 2
		6 5 4 3	6 5 4 3	6 5 4 3	6 5 4 3	6 5 4 3
			7 6 5 4	7 6 5 4	7 6 5 4	7 6 5 4

째 clock에서는 첫 번째 행의 회전모드 5개 PE가 입력 벡터를 입력 된 각도만큼 회전시켜 아래의 PE로 벡터의 좌표 값을 보낸다. 따라서 기존 구조에서는 표 2와 같이 총 14 clock이 필요하게 된다.

표 2는 그림 1의 24개의 각각의 PE가 동작하는 clock을 보여주고 있다. 예를 들면 R_{11}의 PE는 1,3,5,7 번째 clock에서만 동작하고 나머지 clock에서는 동작하지 않는다. 제안 구조에서는 첫 번째 행의 1개의 벡터모드 PE와 5개의 회전모드 PE가 동시에 회전하므로 같은 clock에 계산이 진행된다. 즉 제안 구조는 그림 1의 같은 행에 위치하고 있는 1개의 벡터모드 CORDIC과 5개의 회전모드 CORDIC은 같은 clock에 동작한다. 따라서 5-by-5 입력에 대한 제안 구조의 필요 clock 수는 다음 표와 같다.

표 3에서 보듯이 총 7 clock 만에 전체 계산이 끝나게 됨을 볼 수 있다. 이는 표 2의 14 clock과 비교하여 2분의 1로 clock 수가 감소되었다. 다음 절에서는 실제 예제를 통하여 2개의 상삼각행렬과 하삼각행렬이 update 됨을 확인해보기로 한다.

2. 설계 예제

이 절의 설계 예제에서 사용한 행렬은 다음과 같다.

$$A=\begin{bmatrix} 10 & -6 & 8 & 9 \\ -13 & -15 & 11 & -8 \\ 11 & -14 & -8 & -2 \\ 9 & 14 & -9 & 9 \end{bmatrix}, d=\begin{bmatrix} 151 \\ 64 \\ -36 \\ -29 \end{bmatrix} \quad (16)$$

A는 입력행렬인 4-by-4행렬이고, d는 4-by-1 열행렬이다. 그림 1의 시스톨릭 어레이 구조는 P1, P2 P3, P4의 PE를 경계로 하여 양쪽으로 삼각 행렬을 취하고

표 4. Clock 1의 계산 결과

Table 4. Computation result in clock 1.

(1, 10) (10.05, 0)	(0, -6) (-5.97, -0.6)	(0, 8) (7.96, 0.8)	(0, 9) (8.96, 0.9)	(0, 151) (150.02, 15.04)	(1, 0) (0.1, -1.0)	
	1	0	0	0	0	1
		1	0	0	0	0	1
			1	0	0	0	0	1

표 5. Clock 2의 계산 결과

Table 5. Computation result in clock 2.

(10.05, -13) (16.43, 0)	(-5.97, -15) (8.22, -13.90	(7.96, 11) (-3.83, 13.02	(8.96, -8) (11.81, 2.19)	(150.02, 64) (41.26, 158.0)	(0.1, 0) (0.06, 0.08)	
	(1, -0.6) (1.16, 0)	(0, 0.8) (-0.41, 0.68)	(0, 0.9) (-0.46, 0.77)	(0, 15.04) (-7.71, 12.91)	(0, -1) (0.51, -0.85)	(1, 0) (0.86, 0.51)
		1	0	0	0	0	1
			1	0	0	0	0	1

있다. 이 구조의 각각의 PE에 대한 초기화 조건은 다음과 같다. 즉 R_{11}, R_{22}, R_{33}, R_{44}와 R_{11}^{-1}, R_{22}^{-1}, R_{33}^{-1}, R_{44}^{-1} 등 8개의 PE는 1로 초기화되며 그 밖의 모든 PE는 0으로 초기화된다.

R_{11}의 벡터모드 CORDIC 블록은 초기화 값 1과 입력된 10을 사용하여 입력벡터 (1, 10)을 구성한다. 이 벡터를 사용하여 16 스텝의 벡터모드 회전을 수행하면 (10.05, 0)을 출력하게 된다. 이와 동시에 R_{12}를 비롯한 5개의 회전모드 CORDIC은 R_{11}과 반대방향으로 16 스텝의 회전을 수행한다. R_{12} 등 5개의 회전모드 PE의 입력 벡터는 각각 다음과 같다. 즉, (0, -6), (0, 8), (0, 9), (0, 151), (1, 0)의 벡터가 입력되며 16 스텝 후의 회전된 출력 벡터는 (-5.97, -0.6), (7.96, 0.8), (8.96, 0.9), (150.02, 15.04), (0.1, -1.0)이다. 이를 표로 나타내면 다음과 같다.

표 6. Clock 3의 계산 결과

Table 6. Computation result in clock 3.

(16.43, 11) (19.77, 0)	(8.22, -14) (-0.96, 16.20)	(-3.83, -8) (-7.63, -4.52)	(11.81, -2) (8.7, -8.23)	(41.26, -36) (14.27, -52.86)	(0.06, 0) (0.05, -0.03)	
	(1.16, -13.9) (13.94, 0)	(-0.41, 13.02) (-13.01, 0.68)	(-0.46, 2.19) (-2.22, -0.27)	(-7.71, 158) (-158.07 7, 5.52)	(0.51,0.08) (-0.04, 0.51)	(0.86 ,0) (0.07, 0.86)
		(1, 0.68) (1.21, 0)	(0, 0.77) (0.43, 0.64)	(0, 12.91) (7.29, 10.66)	(0, -0.85) (-0.48, -0.7)	(0, 0.51) (0.29, 0.42)	(1, 0) (0.83, -0.56)
			1	0	0	0	0	1

표 7. Clock 4의 계산 결과
Table 7. Computation result in clock 4.

(19.77, 9) (21.72, 0)	(-0.96, 14) (4.93, 13.14)	(-7.63, -9) (-10.68, -5.03)	(8.7, 9) (11.64, 4.59)	(14.27, -29) (0.97, -32.3)	(0.05, 0) (0.05, -0.02)			
	(3.94, -16.2) (21.38, 0)	(-13.01, -4.52) (-5.07, -12.81)	(-2.22, -8.23) (4.79, -7.05)	(-158.08, -52.86) (-63.05. -154.29)	(-0.04, -0.03) (0, -0.05)	(0.07, 0) (0.05, 0.05)		
		(1.21, 0.68) (1.4, 0)	(0.43, -0.27) (0.24, -0.45)	(7.29, 5.52) (9.06, 1.24)	(-0.48, 0.52) (-0.17, -0.69)	(0.29, 0.86) (0.67, 0.6)	(0.83, 0) (0.72, -0.4)	
			(1, 0.64) (1.18, 0)	(0, 10.66) (5.71, 9.0)	(0, -0.7) (-0.38, -6.0)	(0, 0.42) (0.23, 0.36)	(0, -0.56) (-0.3, -0.48)	(1, 0) (0.84, -0.54)

표 8. Clock 5의 계산 결과
Table 8. Computation result in clock 5.

21.72	4.93	-10.68	11.64	0.97	0.05			
	(21.38, 13.14) (25.09, 0)	(-5.07, -5.03) (-6.95, -1.63)	(4.79, 4.59) (6.48, 1.4)	(-63.05, -32.3) (-70.63, 5.49)	(0, -0.02) (-0,01, -0.02)	(0.05, 0) (0.04, -0.02)		
		(1,39, -12.81) (12.82, 0)	(0.24, -7.05) (7.04, -5.02)	(9.06, -154.29) (154.35, -7.64)	(-0.17, -0.05) (0.03, -0.17)	(0.67, 0.05) (0.02, 0.67)	(0.72 ,0) (0.08, 0.72)	
			(1.18, -0.45) (1.27, 0)	(5.71, 1.24) (4.89, 3.2)	(-0.38, 0.69) (-0.6, 0.51)	(0.23, 0.6) (0, 0.64)	(-0.3, -0.4) (-0.14, -0.49)	(0.84, 0) (0.79, 0.3)

표 9. Clock 6의 계산 결과
Table 9. Computation result in clock 6.

21.72	4.93	-10.68	11.64	0.97	0.05			
	25.09	-6.95	6.48	-70.63	-0,01	0.04		
		(12.82, -1.63) (12.98, 0)	(7.04, 1.4) (6.8, 2.28)	(154.35, 5.49) (152.43, 24.83)	(0.03, -0.02) (0.03, -0.01)	(0.02, -0.02) (0.02, -0.02)	(0.08, 0) (0.08, 0.01)	
			(1.27, -0.52) (1.37, 0)	(4.89, -7.64) (7.42, -5.22)	(-0.6, -0.17) (-0.49, -0.39)	(0, 0.67) (-0,26 0.62)	(-0.4, 0.01) (-0.2, 0.35)	(0.73, 0) (0.38, -0.63)

표 10. Clock 7의 계산 결과
Table 10. Computation result in clock 7.

21.72	4.93	-10.68	11.64	0.97	0.05			
	25.09	-6.95	6.48	-70.63	-0,01	0.04		
		12.98	6.8	152.43	0.03	0.02	0.08	
			(1.37, 2.28) (2.66, 0)	(7.42 ,24.83) (25.09, 6.44)	(-049, -0.01) (-0.26, 0.41)	(-0.26 ,-0.02) (-0.15 0.21)	(-014, 0.72) (-0.20, 0.61)	(079, 0) (038, 0.3)

표 4에서는 첫 번째 행의 각각의 PE에 대한 입력 벡터와 출력 벡터를 나타내고 있다. 2번째 clock부터 7번째 clock까지의 각각의 PE에서의 입력 및 출력 벡터를 나타내면 표 6~10과 같다.

표 10에서 보듯이 7 clock이 지나면 다음과 같이 $\mathbf{R}^{-1}(n)$와 $\mathbf{P}(n)$의 계산이 완료된다.

$$\mathbf{R}^{-1}(n) = \begin{bmatrix} 0.05 & -0.01 & 0.03 & -0.26 \\ 0 & 0.04 & 0.02 & -0.15 \\ 0 & 0 & 0.08 & -0.20 \\ 0 & 0 & 0 & 0.38 \end{bmatrix} \tag{17a}$$

$$\mathbf{P}(n) = \begin{bmatrix} 0.97 \\ -70.63 \\ 152.43 \\ 25.09 \end{bmatrix} \tag{17b}$$

위에서 구한 $\mathbf{R}^{-1}(n)$와 $\mathbf{P}(n)$을 곱하면 다음과 같이 등화기 필터 계수의 계산이 완성된다.

$$\mathbf{w}(n) = \begin{bmatrix} -0.9182 \\ -3.3752 \\ 6.7892 \\ 9.4491 \end{bmatrix} \tag{18}$$

지금까지 예제를 통하여 제안 구조는 7 clock에 계산이 완료됨을 보였다.

Ⅳ. 구현 및 고찰

1. Verilog-HDL 시뮬레이션

이전 절에서 제안한 등화기 구조에서 구현면적의 감소와 계산 속도의 증가를 달성하였다. 이 절에서는 5-by-5의 시스톨릭 어레이 구조에 대하여 제안구조와 기존구조를 HDL로 시뮬레이션하여 동작(function)을 검증하고 구현면적을 비교해보기로 한다. 시뮬레이션의 각 구조의 입력 값들의 정세도는 부호 1비트, 정수 8비트, 소수 7비트인 총 16비트로 구성하였다.

기존구조는 벡터모드 CORDIC의 회전방향결정 블록을 거쳐 얻어진 θ값을 회전모드 CORDIC에 입력시켜서 벡터를 회전시키는 구조이며, 벡터모드와 회전모드를 개별모듈로 설계하였고 시스톨릭 어레이 구조의 동작검증을 위해 TOP 모듈에서는 각 모드를 인스턴스화해서 설계하였다. 5-by-5 입력행렬의 시스톨릭 어레이 구조는 총 4행으로 각 행의 PE는 벡터모드 1개, 회전모드 5개로 구성된다. 각 단의 벡터모드 CORDIC은 각도계산을 위해 1 clock에 동작하도록 구현하였고, 같은 행의 5개의 회전모드 CORDIC은 벡터모드에서의 연산 결과 값인 θ값을 입력으로 받아 1 clock에 동작하도록 조합논리회로로 구현하였다. 이에 따라 병렬 파이프라인 구조의 동작을 적용하여 동작에 필요한 총 clock수는

14 clock이 소요되도록 설계하였다. 회전각 계산에 사용되는 총 16개의 θ값들은 LUT를 이용하였다.

제안구조는 하나의 제어신호로 벡터모드와 로테이션 모드가 동시에 연산되는 구조로 구현하였다. 따라서 벡터모드의 연산 결과 값인 θ값을 회전 모드로 전달할 필요가 없기 때문에 구조가 간단해진다. 제안구조의 시뮬레이션에 소요되는 총 clock 수는 7 clock으로, 기존구조와 비교하여 2배로 계산이 빨라진다. 즉 제안구조는 하나의 제어신호를 이용하기 때문에 각 행의 벡터모드 1개와 회전모드 5개가 동시에 동작하므로 1 clock이 소요된다. 지금까지 기존구조 및 제안구조에 대하여 RTL 코딩의 function simulation을 통하여 출력 값이 정확하게 계산됨을 확인하였다. 다음 절에서는 완성된 RTL 코드로 Synthesis와 Pre-Simulation을 통하여 Front-end 칩 구현을 진행한다.

2. 칩 구현

이 절에서는 지난 절에서 시뮬레이션 한 각각의 구조들에 대한 Front-end 칩 구현을 진행하였다. 먼저 Synopsys Design Compiler Synthesis Tool을 사용하여 각각의 구조를 합성하여 구조에 대한 면적을 비교해 본다. 합성을 위한 Front-end 과정에서는 매그나칩 0.25-Micron 2.5V 공정을 사용하였고, 셀 라이브러리는 hsm222a_bcc를 사용하였다. 각 구조는 동일한 제한 조건을 적용하여 진행하였다. 제안구조와 기존구조의 합성결과에 대한 Area_Report는 다음과 같다.

Front-end 구현 결과를 정리하면 표 12와 같다. 각 구조의 Pre-Simulation을 위하여 netlist와 SDC, SDF 파일을 write하여 PrimeTime을 이용하여 각 구조에 대한 타이밍을 검증하였고 Formality을 사용하여 function을 검증하였다. 표 12에서 보듯이 제안구조는 기존구조와 비교하면 23.8%의 구현면적이 감소됨을 보였다.

표 11. 제안구조의 합성결과에 대한 Area_Report
Table 11. Area_Report for synthesis result of proposed structure.

구분	기존구조의 Area_Report	제안구조의 Area_Report
Hierarchial Cell Count	4127	2880
Hierarchial Port Count	233024	151616
Leaf Cell Count	214979	150360
Combinational Area	11519268	8638802
Noncombinational Area	283754.	354815
Net Area	0	0
Design Area	11803023	8993618

표 12. 제안구조와 기존구조의 면적비교
Table 12. Gate count comparison for proposed structure.

구분	기존구조	제안구조
셀	214979	150360
면적(㎟)	11803023	8993618
면적비율	100%	76.2%

V. 결 론

이 논문에서는 Extended QRD-RLS 알고리즘으로 불리는 시스톨릭 어레이 구조의 등화기에 대한 고속/저전력 설계 및 구현 방법을 제안하였다. 제안된 시스톨릭 어레이 구조에서는 회전모드 CORDIC 블록의 회전방향이 벡터모드 CORDIC 블록과 정반대임을 이용하여 구현면적을 감소시킬 수 있음을 보였다. 즉 벡터모드 CORDIC 블록에서는 각도 update 블록을 없앨 수 있고, 회전모드 CORDIC 블록에서는 각도 update 블록과 회전방향결정 블록을 없앨 수 있음을 보였다. 이와 더불어 제안구조는 벡터모드 CORDIC 블록과 회전모드 CORDIC 블록을 같은 clock에 동작시킴으로써 계산 시간을 1/2로 감소시킬 수 있다. 따라서 제안된 등화기 구조는 Extended QRD-RLS 적응 등화기의 고속/저전력 구현에 널리 사용될 수 있음을 보였다.

참 고 문 헌

[1] S. Haykin, "*Adaptive filtering theory*," 3rd Ed. Prentice Hall, New Jersey.

[2] S. Haykin, A. H. Sayed, J. R. Zeidler, P. Yee, and P. C. Wei, "*Adaptive tracking of linear time-variant systems by extended RLS algorithms*," *IEEE Trans, Signal Proc.* Vol. 45, No. 5, pp. 1118-1127, May 1997.

[3] L. Gao and K. K. Parhi, "*Hierarchical pipelining and folding of QRD-RLS adaptive filters and its application to digital beamforming*," *IEEE Trans, Circuits and Systems II*, Vol. 47, No. 12, pp. 1503-1519, Dec. 2000.

[4] Z. Chi, J. Ma and K. K. Parhi, "*Hybrid annihilation transformation for pipelining QRD-*

based least square adaptive filters," IEEE Trans, Circuits and Systems II, Vol. 48, No. 7, pp. 661-674, Jul. 2001.

[5] J. E. Volder, "*The CORDIC trigonometric computing technique," IRE Trans, on Electronic Computer 8*, pp. 330-334, 1959.

[6] Y. H. Hu, "*CORDIC-based VLSI architecture for digital signal processing," IEEE Signal Processing Magazine*, Vol. 9, pp. 16-35, 1992.

[7] T. Z. Mingqian, A. S. Madhukumar and F. Chin, "*QRD-RLS adaptive equalizer and its CORDIC- based implementation for CDMA systems", International Journal on Wireless & Optical Communications*, Vol. 1, No. 1, pp. 25-39, 2003.

[8] K. I. Lee, J. H. Lee, J. K. Lee and Y. S. Cho, "*A compact CORDIC algorithm for synchronization of carrier frequency offset in OFDM modems," IEICE Transactions on Communications*, Vol. E89-B, No. 3, pp. 952-954, 2006.

저 자 소 개

문 대 원(학생회원)
2006년 상명대학교 정보통신 공학과 졸업.(공학사)
2010년 상명대학교 대학원 컴퓨터 정보통신공학과 대학원 졸업(공학석사).

<주관심분야 : 통신신호처리, 비디오신호처리, SoC 설계>

조 용 훈(정회원)
1986년 연세대학교 전기공학 졸업.(공학사)
1988년 연세대학교 대학원 전기공학과 졸업. (공학석사)
2001년 연세대학교 대학원 전기전자공학과 졸업. (공학박사)
2006년~2008년 한남대 겸임교수
2002년~현재 ㈜코메스타 부사장/연구소장, 전주대학교 겸임교수.
<주관심분야 : 디지털 통신, 디지털 모뎀, 이동통신 시스템, 광대역 전송 기술>

장 영 범(정회원)
1981년 연세대학교 전기공학과 졸업.(공학사)
1990년 Polytechnic University 대학원 졸업.(공학석사)
1994년 Polytechnic University 대학원 졸업.(공학박사)
1981년~1999년 삼성전자 System LSI 사업부 수석연구원.
2002년~현재 상명대학교 정보통신공학과 교수.
<주관심분야 : 통신신호처리, 비디오신호처리, SoC 설계>