論文 95-10-10-17

고속 Hadamard 변환 알고리즘을 위한 효율적인 VLSI 구조

正會員 玄 鎭 一*, 李 熏 馥**, 車 鎭 鍾*, 金 在 錫*, 金 景 洙*

Low Latency VLSI Architectures for Fast Hadamard Transform

Jin-Il Hyun*, Hoon-Bok Lee**, Jin-Jong Cha*, Jaeseok Kim*, Kyungsoo Kim* Regular Members

要 約

고속 Hadamard 변환 (FHT) 알고리즘을 VLSI 로 구현하기 위한 프로세서 구조를 설계하고 하드웨어 복잡도와 처리 시간을 분석하였다. FHT용 프로세서 구조에는 파이프라인 구조, 2진 Tree 구조 등이 있는데 일반적인 FHT 알고리즘 또는 수정된 FHT 알고리즘으로부터 설계된다. 구조의 비교 결과 수정된 알고리즘을 위한 구조가 일반적인 알고리즘을 위한 구조보다 하드웨어 복잡도 또는 처리 시간에서 우수한 것으로 나타났다. 또한 2진 Tree 구조는 처리 시간이 가장 짧은 반면 파이프라인 구조가 하드웨어 복잡도 및 처리 시간을 동시에 고려할 때 가장 우수하다.

ABSTRACT

A couple of VLSI architectures for implementing the Fast Hadamard Transform (FHT) algorithm are presented and analyzed in terms of the hardware complexity and the latency time. Those architectures are derived from an ordinary or a modified FHT algorithm. The analysis shows that the architectures derived from the modified FHT algorithm yields lower latency time and/or lower hardware complexity. The analysis also shows that the pipeline architecture is suitable for the low latency VLSI implementation with a moderate hardware complexity while the binary tree architecture has the lowest latency time.

* 한국전자통신연구소 VLSI 구조연구실
** 한림대학교 전자공학과
論文番號 : 95199-0531
接受日字 : 1995年 5月 31日

Ⅰ. 서 론

Hadamard 변환은 디지털 통신 시스템의 오류 정정 부호화, 영상 분석 시스템 등에 널리 사용되어 왔던 일종의 직교 변환이다. [1] [2] [3] 최근에는 CDMA 디지털 이동통신 시스템의 기지국이 단말기로 부터 송신된 직교 변조 신호를 복구하는데 사용되고 있는데 직교 변조 신호의 실시간 복구 및 Hadamard 변환기의 VLSI 구현을 위하여 효율적인 하드웨어 구조의 설계가 요구되고 있다 [4].

Hadamard 변환을 직접 수행할 경우 N 개의 입력 데이타에 대하여 N^2의 연산이 필요하게 되어 실시간 처리가 어려우나 고속 Fourier 변환과 같이 Hadamard 행렬의 특성을 이용하여 연산을 줄이는 고속 Hadamard 변환 (FHT, Fast Hadamard Transform) 알고리즘의 경우에는 N 개의 입력 데이타에 대하여 $N/2 \log_2 N$의 버터플라이 (Butterfly) 연산이 요구된다. 따라서 FHT는 비교적 구현이 용이하며 이를 구현하기 위한 순차적[4] 또는 파이프라인 방식의 하드웨어 구조가 발표되어 있다 [5]. 순차적 구조의 경우 프로세서를 1개 사용하기 때문에 프로세서당 최소한 $N/2 \log_2 N$의 버터플라이 연산 수행이 필요하다. 반면에 파이프라인 구조의 경우 $\log_2 N$ 개의 프로세서를 사용하기 때문에 프로세서 1개당 처리하는 연산 수가 줄어들어 처리 시간을 줄일 수 있다. 하지만 FHT를 위한 기존의 파이프라인 구조의 경우 프로세서의 효율이 50%에 불과하며 데이타 입력 후 FHT 결과 출력까지의 지연 시간 (Latency Time)이 길다. 따라서 FHT를 위한 효율적인 VLSI 구현을 위해서는 하드웨어 복잡도는 줄이면서 처리 속도를 개선하는 방안에 대한 연구가 필요한 상황이다.

본 논문에서는 FHT 알고리즘을 구현하기 위한 기존 하드웨어 구조의 지연 시간 특성을 분석하고 이를 개선하기 위한 방안을 제시하고자 한다. 제 Ⅱ 장에서는 FHT 알고리즘 및 CDMA 디지털 이동통신 시스템의 FHT 응용 예를 살펴보고 제 Ⅲ 장에서는 FHT 알고리즘을 위한 기존 하드웨어의 구조를 분석한다. 제 Ⅳ 장에서는 프로세서의 효율을 높이거나 병렬 처리를 함으로써 지연 시간을 줄일 수 있는 방안 및 하드웨어 구조를 제시하고 제 Ⅴ 장에서 결론을 맺는다.

Ⅱ. 디지털 통신 시스템의 Hadamard 변환 응용 및 FHT 알고리즘

1. 디지털 이동 통신 시스템의 오류 정정을 위한 Hadamard 변환

디지털 이동 통신 시스템에서 채널 오류 정정을 위한 기법으로 Convolution coding, Interleaving 이 널

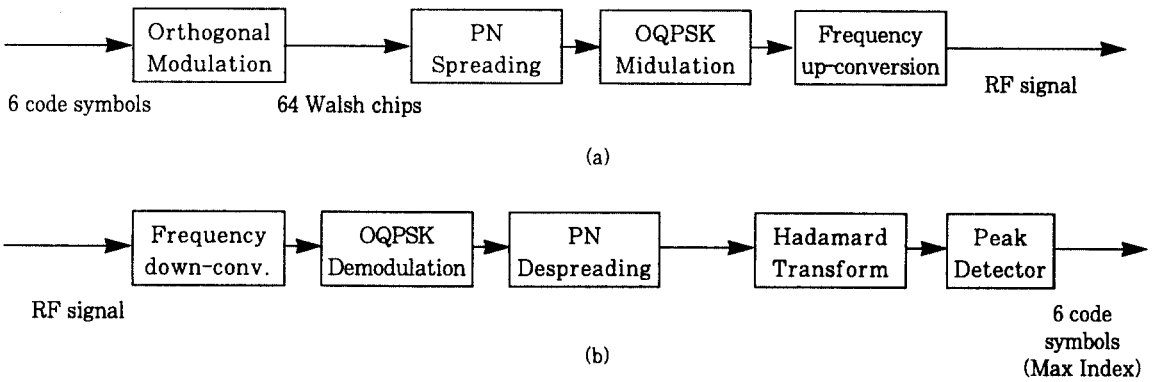


그림 1. Hadamard 변환 오류 정정 기법을 이용한 디지털 통신 시스템 구성도
(a) CDMA 디지털 이동통신 역방향 링크 송신부
(b) CDMA 디지털 이동통신 역방향 링크 수신부

Fig. 1. Block diagram of Digital communication system using Hadamard transform method
(a) Reverse link transmitter of CDMA mobile system
(b) Reverse link receiver of CDMA mobile system

리 사용되고 있으나 CDMA 디지털 이동통신 시스템의 역방향 링크에서는 이와 더불어 Walsh-Hadamard 직교 변조가 사용되고 있다[6]. 송신부에서는 [그림 1(a)]에 나타나 있듯이 Convolution coding, Interleaving 을 거친 코드 심볼을 일정한 크기로 나누어 Walsh-Hadamard 직교 변조를 수행한 후 PN 확산, OQPSK 변조를 거쳐 RF 신호를 송신한다[7].

Walsh-Hadamard 직교 변조는 $\log_2 N$ 비트의 입력 데이타에 따라 N×N Walsh-Hadamard 행렬 중 한 열을 선택하여 N 개의 데이타를 생성하는 것으로서 이 때 사용되는 Walsh-Hadamard 행렬은 다음과 같이 순환적으로 구해진다.

$$H_1 = 1 \tag{1}$$

$$H_2 = \begin{bmatrix} H_1 & H_1 \\ H_1 & -H_1 \end{bmatrix} \tag{2}$$

$$H_{2n} = \begin{vmatrix} H_n & H_n \\ H_n & -H_n \end{vmatrix} \tag{3}$$

따라서 N이 8인 경우의 Walsh-Hadamard 행렬은 다음과 같은 값을 갖게 되는데 행렬의 계수가 1 또는 -1 임을 알수 있다.

$$H_8 = \begin{vmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{vmatrix} \tag{4}$$

CDMA 디지털 이동통신 시스템의 역 방향 링크에서는 N이 64인 Walsh-Hadamard 행렬을 사용하고 있는데 매 6 비트의 코드 심볼 단위로 이에 해당되는 Walsh-Hadamard 행렬의 한 열 즉 64 Walsh Chip을 송신하게 된다. 수신부에서는 [그림 1(b)]에 나타나 있듯이 RF 신호를 기저대역 신호로 변환하고 OQPSK 복조, PN 역확산을 거쳐 64개 심볼 단위로 나눈다. 다음에는 수신된 64개의 심볼이 Walsh-Hadamard 행렬 중 어느 열과 가장 일치하는가를 결정하여 원래 송신된 6개의 심볼을 복구하는데 이 이 과정은 Hadamard 변환과 Hadamard 변환 결과 중 최대 값을 찾아내는 과정으로 구성된다. 입력 벡타 X 에 대하여 Hadamard 변환은 다음과 같이 정의된다.

$$Y = 1/N\ H_N\ X \tag{5}$$

여기서 X는 길이가 N인 입력 벡터이고 HN는 크기가 N×N인 Walsh-Hadamard 행렬이며 Y는 길

이가 N인 출력 벡터이다. Hadamard 변환 결과인 벡타 Y의 각 요소는 길이 N인 입력 벡터 중 Walsh-Hadamard 행렬의 각 열과 유사한 정도를 나타낸다고 볼 수 있기 때문에 벡타 Y 요소 중 최대 값을 찾아내면 Walsh-Hadamard 행렬 중 송신된 열을 알게 되고 송신된 열의 위치는 Walsh-Hadamard 변조 이전의 코드 심볼 값을 가르키게 된다 . [그림 1]과 같은 시스템의 경우 6 비트의 코드 심볼을 위하여 실제로는 64 Walsh Chip을 송신하기 때문에 채널 통과시 64 Chip 중 일부에서 오류가 발생하더라도 6 비트의 코드 심볼을 성공적으로 복구할 수 있게 된다. Hadamard 함수의 자유거리 (dmin , Free Distance)는 N/2로서 (dmin - 1)/2 개의 오류까지 정정 할 수 있다.

[그림 1]의 Walsh-Hadamard 행렬을 이용한 오류 정정 관련 블럭 중 Walsh-Hadamard 직교 변조기는 식 (1) (2) (3)의 행렬 생성 방법을 이용하여 간단하게 구현할 수 있으나 Hadamard 변환기의 구현은 간단하지 않으며, 수신부를 VLSI로 구현할 경우 Hadamard 변환기가 차지하는 칩의 면적이 매우 높다 [5]. 그러므로 Hadamard 변환기를 위한 효율적인 알고리즘 및 하드웨어 구조 설계는 집적도가 높은 수신부 구현의 필수적인 조건이다.

2. 고속 Hadamard 변환 알고리즘

Walsh-Hadamard 행렬의 계수는 1 또는 -1의 값을 갖기 때문에 Hadamard 변환은 다른 변환에 비하여 비교적 연산이 간단한 편이다. 하지만 Hadamard 변환을 (5)의 정의대로 수행할 경우 N^2에 비례하는 수만큼의 덧셈 또는 뺄셈 연산이 필요하게 된다.

고속 Hadamard 변환 알고리즘은 Walsh-Hadamard 행렬의 특성을 이용하여 Hadamard 변환에 필요한 연산수를 줄이는 방식으로서 (3)의 특성을 이용하면 N 데이타에 대한 Hadamard 변환 결과인 Y 벡터의 상위 반과 하위 반은 각각 다음과 같이 길이 N/2인 새로운 벡터에 대한 Hadamard 변환으로 부터 구해진다.

$$\begin{bmatrix} y_0 \\ \cdots \\ y_i \\ \cdots \\ y_{N/2-1} \end{bmatrix} = H_{N/2} \begin{bmatrix} xl_0 \\ \cdots \\ xl_i \\ \cdots \\ xl_{N/2-1} \end{bmatrix} \quad (6)$$

$$\begin{bmatrix} y_{N/2} \\ \cdots \\ y_{j+N/2} \\ \cdots \\ y_{N-1} \end{bmatrix} = H_{N/2} \begin{bmatrix} xl_{N/2} \\ \cdots \\ xl_{j+N/2} \\ \cdots \\ xl_{N-1} \end{bmatrix} \quad (7)$$

여기서 x1i 와 x1j+N/2 는 다음과 같이 데이타 xi 간의 연산으로 얻어진 중간 결과이다.

$$x1i = (xi+xi+N/2)/2 \quad (8)$$

$$x1j+N/2 = (xj - xj+N/2)/2 \quad (9)$$

이 과정을 반복하면 길이 N/2 벡터에 대한 Hadamard 변환은 길이 N/4 벡터에 대한 Hadamard 변환으로부터 구해지며 최종에는 길이가 2인 벡터에 대한 Hadamard 변환 즉 2 데이타 간의 연산으로 Hadamard 변환이 완료되게 된다.

N이 8인 경우의 FHT 알고리즘 신호 흐름도가 〔그림 2〕에 그려져 있는데 첫 단계에서는 N=8에 대한 (8)과 (9)의 연산을 수행하고 다음 단계에서는 N=4, 최종 단계에서는 N=2에 대한 연산을 수행한다. 일반적으로 FHT 알고리즘은 $\log_2 N$ 단계로 구성되며 각 단계에서는 N/2 개의 버터플라이 연산이 수행된다. 버터플라이 연산은 〔그림 3〕과 같이 덧셈과 뺄셈, SHIFT RIGHT 동작에 의한 나누기 2 연산에 의하여 간단하게 구성된다. 〔그림 2〕의 신호 흐름도를 모든 단계에서 규칙적이 되도록 변형한 것이 〔그림 4〕인데 이 것은 〔그림 2〕에서 중간 연산 결과의 위치를 조정함으로써 구해진다. 〔그림 2〕에서 볼수 있듯이 N개의 입력에 대한

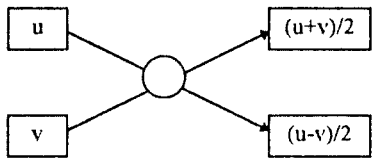


그림 3. FHT 알고리즘의 버터플라이 연산
Fig. 3. Butterfly processor of FHT

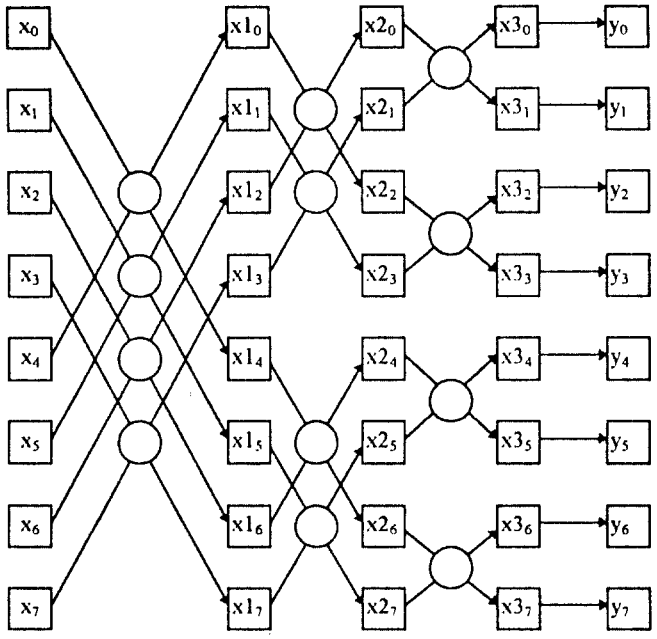


그림 2. FHT 알고리즘의 신호 흐름도
Fig. 2. Signal flow of FHT

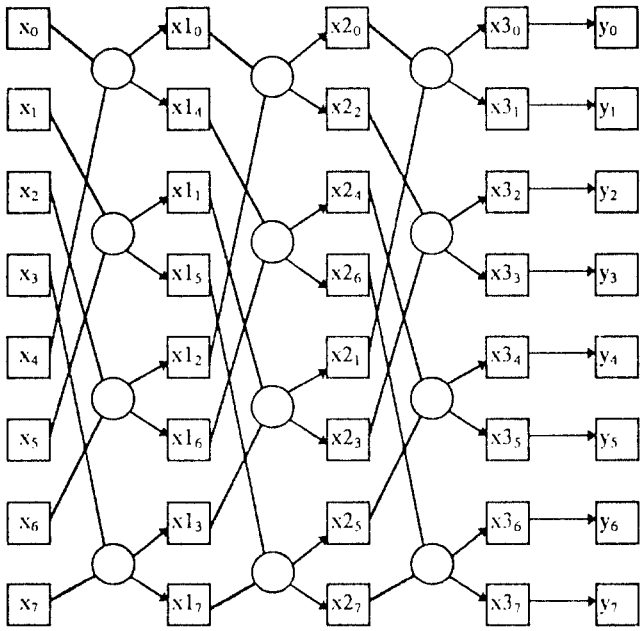


그림 4. FHT 알고리즘의 규칙화된 신호 흐름도
Fig. 4. Regular signal flow of FHT

고속 Hadamard 변환은 $N/2\ \log_2 N$ 개의 버터플라이 연산을 필요로 하여 행렬과 벡터의 직접 곱셈 방식보다 연산 요구량이 $1/N\ \log_2 N$ 배로 줄어든다.

Ⅲ. FHT를 위한 하드웨어 구조 분석

FHT 알고리즘을 구현하는 방법에는 하나의 프로세서를 이용하여 모든 연산을 순차적으로 수행하는 방법과 여러개의 프로세서를 사용하여 연산을 분담시킴으로써 FHT 처리 시간을 줄이는 방법이 있을 수 있다. 프로세서를 여러개 사용할 경우 프로세서 간의 연결 방식 및 연산 분담 방법에 따라 프로세서의 효율이 달라지고 이에 따라 FHT 처리 시간이 달라지게 된다. 이 장에서는 기존의 단일 프로세서 구조와 파이프라인 구조의 프로세서 효율, FHT 처리 시간 및 개선 할 사항을 분석한다.

1. 단일 프로세서 구조

단일 프로세서 구조는 〔그림 5〕와 같이 하나의 프로세서 (PE) 를 이용하여 연산을 수행한다. 입력 데이타는 우선 메모리에 저장되는데 N개의 입력 데이타 중 처음 N/2 개는 메모리 뱅크-1 에, 나머지 N/2 개는 메모리 뱅크-2 에 저장된다. 연산은 중간 연산 결과를 메모리에 저장하거나 읽을 때 2개의 메모리 뱅크를 효율적으로 사용할 수 있도록 〔그림 4〕에 있는 신호 흐름도를 따라 수행한다. 각 단계에서는 위쪽으로 부터 아래 쪽 순서로 수행하며 한 단계가 완료되면 다음 단계로 이동한다. 한편 각 단계의 연산 결과 중 처음 N/2 개는 메모리 뱅크-1에 저장하고 나머지는 메모리 뱅크-2에 저장한다. 각 단계의 수행에는 모두 N/2의 연산이 필요하며 $\log_2 N$ 단계의 연산 결과가 FHT 출력이 된다. 버터플라이 연산 시간을 단위 시간으로 했을 때 입력이 완료된 후 최종 결과가 출력되기 까지에는 $N/2\ \log_2 N$ 단위 시간이 소요된다.

2. 파이프라인 구조

파이프라인 구조는 $\log_2 N$ 개의 프로세서를 사용하여 프로세서 한 개가 한 단계를 담당하게 하고 각 프로세서가 동시에 연산을 수행하도록 함으로써 FHT 처리 시간을 줄일 수 있는 방안이다. FHT를 위한 파이프라인 구조는 〔그림 6(a)〕과 같이 1차원 프로세서 어레이로 구

성되어 있는데 각 프로세서는 입력된 데이타에 대하여 [그림 2]에 나타나 있는 FHT 신호 흐름도의 한 단계 연산을 수행하고 그 결과를 다음 단계 프로세서로 넘겨 준다. 각 프로세서는 [그림 6(b)]와 같은 내부 구조를 갖고 있는데 1 단계 프로세서는 N/2 , 2 단계 프로세서는 N/4, 최종 단계 프로세서는 1 개 데이타를 위한 버퍼를 갖고 있다. 전 단계 프로세서 또는 외부로 부터 입력되는 데이타는 FIFO 버퍼의 입력 다중화기를 통하여 버퍼가 채워질 때까지 버퍼에 일단 저장된다. 다음 데이타가 들어오면 버퍼의 출력 데이타와 버터플라이 연산기에서 연산이 이루어 진다. 연산 결과 중 하나는 출력 다중화기를 통하여 다음 단계 프로세서로 전달되고 하나는 버퍼 입력 다중화기를 통하여 버퍼에 저장된다. 다음 입력 데이타에 대해서도버퍼에 저장된 입력 데이타가 소진될 때 까지 이 과정이 반복된다. 이 경우 버퍼에는 다음 단계 프로세서로 보낼 버터플라이 연산 결과로 채워져 있게 된다. 새로운 입력 데이타가 들어오면 다시 버퍼에 저장되는데 버퍼에 들어 있던 연산 결과는 출력 다중화기를 통하여 다음 단계 프로세서로 차례대로 전달된다.

입력 데이타가 8개 일 경우 [그림 6(a)]의 구조로서 [그림 2] 있는 FHT 알고리즘을 수행할 경우 시간에 따른 각 프로세서의 동작을 보면 [그림 6(c)]와 같다. [그림 6(c)]에서 알 수 있듯이 파이프라인 구조는 최종 데이타가 입력된 후 최종 FHT 결과가 출력되기 까지 N-1 단위 시간이 걸리고 있으며 지연시간 과 소요되는 프로세서 수의 곱이 $N\log_2 N$ 이다. 이는 FHT 알고리즘에서 필요로 하는 전체 버터플라이 연산 $N/2\ \log_2 N$ 개를 $\log_2 N$ 개의 프로세서를 사용할 경우 얻을 수 있는 최적값의 2배가 되는데 첫 번째 프로세서를 제외한 각

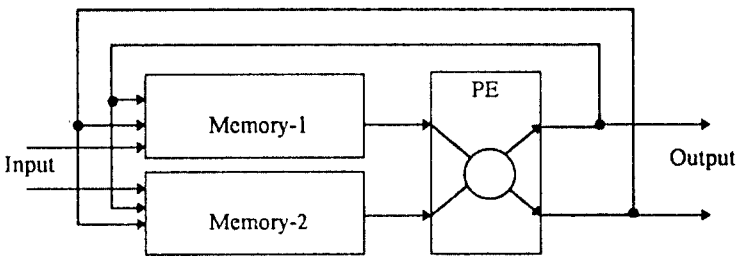
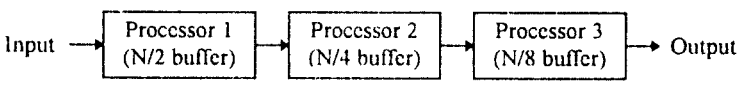
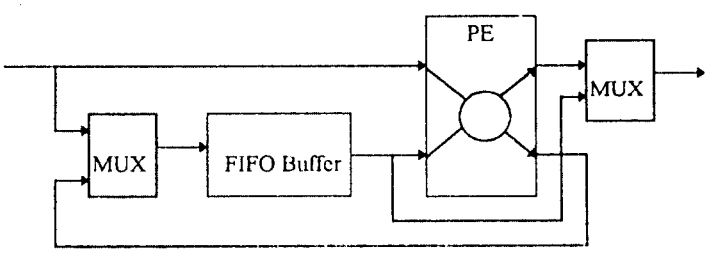


그림 5. FHT를 위한 단일 프로세서 구조
Fig. 5. Architecture of FHT using unit processor



(a)



(b)

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Input	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7							
Processor-1					x_0 op x_4	x_1 op x_5	x_2 op x_6	x_3 op x_7							
	x_0 -> buf	x_1 -> buf	x_2 -> buf	x_3 -> buf	$x1_4$ -> buf	$x1_5$ -> buf	$x1_6$ -> buf	$x1_7$ -> buf							
Processor-2							$x1_0$ op $x1_2$	$x1_1$ op $x1_3$			$x1_4$ op $x1_6$	$x1_5$ op $x1_7$			
					$x1_0$ -> buf	$x1_1$ -> buf	$x2_2$ -> buf	$x2_3$ -> buf	$x1_4$ -> buf	$x1_5$ -> buf	$x2_6$ -> buf	$x2_7$ -> buf			
Processor-3								$x2_0$ op $x2_1$		$x2_2$ op $x2_3$		$x2_4$ op $x2_5$		$x2_6$ op $x2_7$	
							$x2_0$ -> buf	$x3_1$ -> buf	$x2_2$ -> buf	$x3_3$ -> buf	$x2_4$ -> buf	$x3_5$ -> buf	$x2_6$ -> buf	$x3_7$ -> buf	
Output								$x3_0$	$x3_1$	$x3_2$	$x3_3$	$x3_4$	$x3_5$	$x3_6$	$x3_7$

(c)

그림 6. N=8인 경우, FHT를 위한 파이프라인 구조 (N=8인 경우)
(a) FHT를 위한 프로세서 어레이구조 (b) 프로세서 내부 구조
(c) 시간에 따른 프로세서의 동작

Fig. 6. Pipe-line architecture of FHT(Case N=8)
(a) Array architecture of FHT (b) Detail structure of processor
(c) Operation diagram of processor

프로세서가 최초 연산을 시작하여 최종 연산을 마칠 때까지의 전체 시간 중 50% 만 연산에 사용되고 있기 때문이다.

Ⅳ. FHT 처리 시간 개선을 위한 구조

제 Ⅲ 장에 기술된 기존의 FHT용 구조 들은 병렬 연산 구조가 아니거나 병렬 연산 구조일 경우에도 프로세서의 효율이 50%에 불과하다. 여기서는 파이프라인 구조의 프로세서 효율을 향상 시키는 구조를 제시하고 FHT 처리 시간을 더욱 단축시키고자 할 때 이에 대한 방안으로 병렬성을 확장 할 수 있는 구조를 설계하고 성능을 분석한다. 또한 FHT 알고리즘의 연산 순서를 바꿈으로써 보다 효율적인 병렬 구조를 설계할 수 있음을 설명하고 새로운 구조를 제시한다.

1. 향상된 프로세서 효율의 파이프라인 구조

〔그림 6〕의 FHT를 위한 파이프라인 구조에서 프로세서 효율이 낮은 것은 각 프로세서의 버터플라이 연산기에서는 2개의 연산 결과를 배출하나 다음 단계 프로세서로는 이중에 하나만 전달되고 하나는 현 단계 프로세서의 버퍼에 저장되어 있다가 다음 단계 프로세서로 전달되는데 이 과정에서 프로세서가 연산을 수행하지 않기 때문이다. 따라서 버터플라이 연산 결과를 다음 단계 프로세서로 전달하는 과정을 개선하면 프로세서의 효율을 높일 수 있다.

〔그림 7(b)〕의 프로세서 기본 구조는 〔그림 6(b)〕를 개선한 것으로서 2개의 FIFO 버퍼, 2개의 다중화기와 1개의 버터플라이 연산기로 구성되어 있다. 버퍼 중 하나는 같은 크기의 버퍼 2개로 나누어져 있어 2개의 출력이 가능하도록 되어 있다. 또한 버터플라이 연산기의 양 쪽 입력에 다중화기가 달려 있어 버퍼 출력이나 외부

입력 중 하나를 선택하도록 되어 있다. 프로세서의 동작은

동작-1: 버퍼-1에 입력 데이타를 저장하는 동작,

동작-2: 프로세서 입력-1과 버퍼-1 출력 데이타에 대하여 버터플라이 연산을 수행하여 연산 결과 중 하나는 다음단계 프로세서로 전달하고 다른 하나는 버퍼-2에 저장하는 것,

동작-3: 프로세서 입력-2 와 입력-3에 대하여 버터플라이 연산을 수행하여 결과를 다음 단계 프로세서와 버퍼-2에 저장하는 것

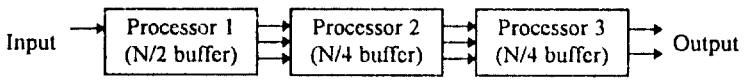
등 모두 3 종류가 있다.

입력 데이타가 8개인 경우의 FHT 연산 과정이 〔그림 7(d)〕에 나타나 있는데 프로세서-1에서는 처음 4 단위 시간 동안에는 동작-1이 이루어지고 다음에는 동작-2가 이루어지면서 FHT 알고리즘의 첫 단계 연산이 수행된다. 프로세서-2 에서는 처음 2 단위 시간에는 동작-1이, 다음 2 단위 시간에는 동작-2, 다음에는 동작-3 순으로 2 번째 단계가 수행된다. 프로세서-3 에서는 처음 1 단위 시간에 동작-1, 다음 1 단위 시간에 동작-2, 다음 1 단위 시간에 동작-1과 동작-3이 동시에 이루어지면서 3 번째 단계가 수행된다. 최종 결과는 2개씩 동시에 출력되며 최종 입력이 완료된 싯점에서 최종 출력이 완료되기 까지 소요되는 시간은 N/2 단위 시간이다.

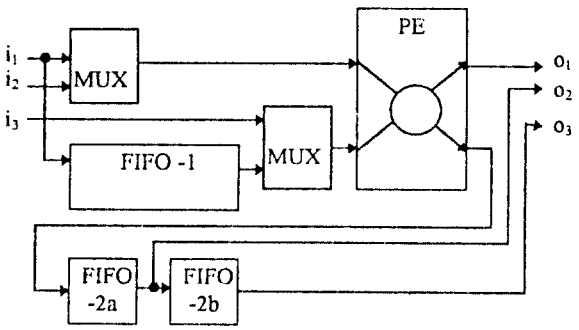
따라서 제 Ⅲ 장의 파이프라인 구조에 비하여 지연 시간이 반으로 감소하였는데 이는 모든 프로세서 들이 최초 연산을 시작하여 최종 연산을 마칠 때까지 100%의 효율로 동작하기 때문이다. 프로세서의 기본 구조는 〔그림 6(b)〕에 있는 프로세서에 비해 총 버퍼의 크기가 2배가 되는데 이는 프로세서의 효율이 높아지기 때문에 생기는 문제이다. 그러나 실제 상황에서는 프로세서-1과 프로세서-2 에서는 버퍼-1과 버퍼-2가 동시에 사용되는 경우가 없기 때문에 〔그림 7(c)〕와 같이 간략화된 프로세서 구조를 사용할 수 있고 이에 따라 전체적으로 필요한 버퍼의 양은 〔그림 7(a)〕에 나타난 바와 같이 된다. 〔그림 4 〕의 구조에서는 총 버퍼의 크기가 N-1이나 〔그림 7〕의 구조에서필요로 하는 버퍼의 양은 2(N-1)-3/4N으로서 약 25%의 버퍼를 더 필요로 한다.

2. 병렬 프로세서 구조

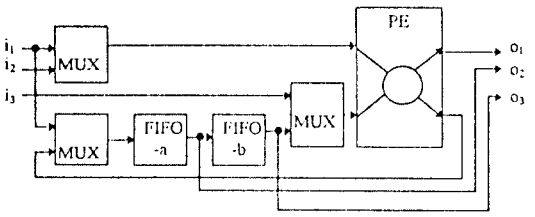
고속 Hadamard 변환 알고리즘은 한 단계의 연산은 그 전단계의 연산 결과에 종속적이지만 각 단계 내의 연산 들은 서로 독립적으로 수행될 수 있는 특성을 갖고 있다. 따라서 각 단계의 버터플라이 연산을 병렬로 수행할 경우 FHT 처리 시간을 더욱 단축시킬 수 있다. 〔그



(a)



(b)



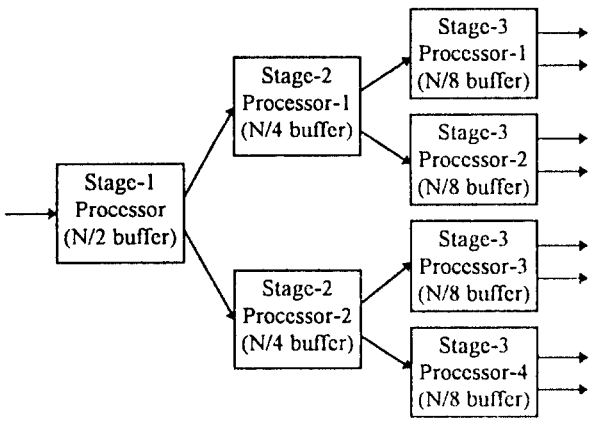
(c)

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Input	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7							
Processor-1					x_0 op x_4	x_1 op x_5	x_2 op x_6	x_3 op x_7							
	x_0 -> buf1	x_1 -> buf1	x_2 -> buf1	x_3 -> buf1											
					$x1_4$ -> buf2	$x1_5$ -> buf2	$x1_6$ -> buf2	$x1_7$ -> buf2							
Processor-2							$x1_0$ op $x1_2$	$x1_1$ op $x1_6$	$x1_4$ op $x1_6$	$x1_5$ op $x1_7$					
					$x1_0$ -> buf1	$x1_1$ -> buf1									
							$x2_2$ -> buf2	$x2_3$ -> buf2	$x2_6$ -> buf2	$x2_7$ -> buf2					
Processor-3								$x2_0$ op $x2_1$	$x2_2$ op $x2_3$	$x2_4$ op $x2_5$	$x2_6$ op $x2_7$				
							$x2_0$ -> buf1		$x2_4$ -> buf1						
								$x3_1$ -> buf2	$x3_3$ -> buf2	$x3_5$ -> buf2	$x3_7$ -> buf2				
Output								$x3_0$	$x3_1$ $x3_2$	$x3_3$ $x3_4$	$x3_5$ $x3_6$	$x3_7$			

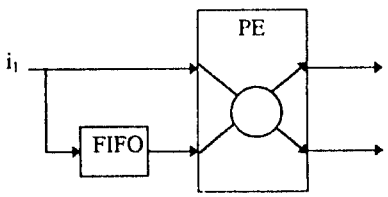
(d)

그림 7. FHT를 위한 고효율 파이프라인 구조 (N=8 인 경우)
(a) FHT를 위한 고효율 프로세서 어레이구조 (b) 프로세서 기본 구조
(c) 버퍼 절약형 프로세서 구조 (d) 시간에 따른 프로세서의 동작

Fig. 7. Efficient pipe-line architecture of FHT (case N=8)
(a) Efficient processor array of FHT (b) Basic structure of processor
(c) processor structure for minimum buffer size (d) Operation diagram of processor



(a)



(b)

그림 8. FHT를 위한 2진 Tree 병렬 구조
(a) 2진 Tree 병렬 구조 (b) 프로세서 구조

Fig. 8. Binary tree structure for FHT
(a) Binary tree structure (b) Structure of processor

림 8]에 FHT를 위한 2진 Tree 병렬 구조와 프로세서 구조가 나타나 있는데 1단계에는 1개, 2 단계는 2개, 3 단계는 4개의 프로세서가 있으며 각 단계별로 버터플라이 연산을 병렬로 처리한다. 따라서 입력데이타 수가 N인 FHT를 위하여 전체적으로 필요한 프로세서의 수는 N-1개 이며 버퍼의 크기는 $N/2 \log_2 N$가 된다. 한편 최종 입력 완료 후 최종 출력 완료까지는 log2N의 단위 시간이 필요하게 된다.

3. 수정된 FHT 알고리즘을 이용한 FHT 프로세서 구조

지금까지 제시된 FHT 프로세서 구조들은 [그림 2]나 [그림 4]의 신호흐름도를 파이프라인 방식이나 병렬 연산 방식으로 구현하고 있는데 모두 입력 벡타 길이가 N인 Hadamard 변환을 벡타 길이 N/2 인 Hadamard 변환 2개로 바꾸는 방식을 사용하고 있다. 변환 순서를 역으로 하여 우선 벡타 길이 2에 대한 Hadamard 변환을 수행하고 다음에 벡타 길이 4에 대한 변환 순서로 진행하면 [그림 9]와 같은 신호 흐름도를 얻을 수 있다.

수정된 FHT 알고리즘을 수행하기 위한 파이프라인 구조를 설계하면 [그림 10(a)]와 같이 된다. 전체적인 구조는 [그림 6]의 구조와 같은 형태이며 프로세서 구조도 동일하다. 다만 최종단 프로세서에서 동시에 2개의 연산 결과가 출력될 수 있도록 되어 있다. 또한 FHT 알고리즘 수행 순서가 변경됨으로 인하여 프로세서의 버퍼 크기가 바뀌어 있다. 시간상의 동작을 보면 [그림 10(b)]와 같은데 동작 방식은 제 Ⅲ 장에 기술된 [그림 6]의 파이프라인 구조와 같다. 연산 결과를 하나씩 출력할 경우 [그림 6]의 구조와 같은 시간이 소요되지만 2개의 결과를 동시에 출력할 경우 [그림 6]의 구조에 비해 FHT 처리 시간이 단축되고 있다. 전체적으로 필요한 프로세서의 수는 log2N 이며 전체 버퍼 메모리의 크기는 N-1 이고 최종 데이타 입력 후 최종 출력까지는 N/2-1 단위 시간이 소요된다.

수정된 FHT 알고리즘의 각 단계 별 연산 중 병렬화가 가능한 것들을 병렬 처리하기 위한 구조가 [그림 11]에 그려져 있는데 기본적인 구조는 [그림 8]의 구조

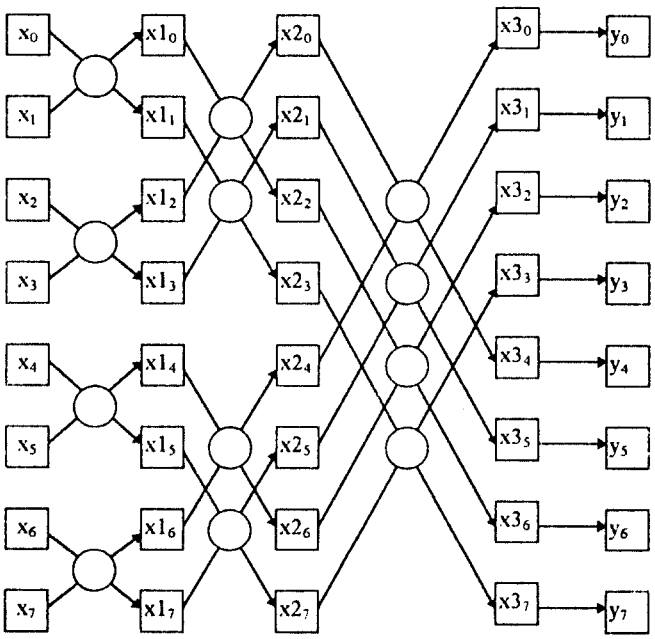
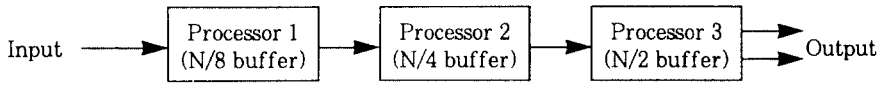


그림 9. 수정된 FHT 알고리즘 의 신호 흐름도
Fig. 9. Signal flow of modified FHT algorithm



(a)

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Input	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7							
Processor-1		x_0 op x_1		x_2 op x_5		x_4 op x_6		x_6 op x_7							
	x_0 -> buf	$x1_1$ -> buf	x_2 -> buf	$x1_3$ -> buf	x_4 -> buf	$x1_5$ -> buf	x_6 -> buf	$x1_7$ -> buf							
Processor-2				$x1_0$ op $x1_2$	$x1_1$ op $x1_3$			$x1_4$ op $x1_6$	$x1_5$ op $x1_7$						
		$x1_0$ -> buf	$x1_1$ -> buf	$x2_2$ -> buf	$x2_3$ -> buf	$x1_4$ -> buf	$x1_5$ -> buf	$x2_6$ -> buf	$x2_7$ -> buf						
Processor-3								$x2_0$ op $x2_4$	$x2_1$ op $x2_5$	$x2_2$ op $x2_6$	$x2_3$ op $x2_7$				
				$x2_0$ -> buf	$x2_1$ -> buf	$x2_2$ -> buf	$x2_3$ -> buf								
Output								$x3_0$ $x3_4$	$x3_1$ $x3_3$	$x3_2$ $x3_6$	$x3_3$ $x3_7$	($x3_4$)	($x3_5$)	($x3_6$)	($x3_7$)

(b)

그림 10. 수정된 FHT 알고리즘을 위한 파이프라인 구조
(a) 수정된 FHT를 위한 프로세서 어레이구조
b) 시간에 따른 프로세서의 동작
Fig. 10. Pipe-line architecture for modified FHT algorithm
a) Array structure of Processor for modified FHT algorithm
(b) Operation of processor

와 동일하다. 다만 각 프로세서의 버퍼크기가 1 이 되어 전체 버퍼 크기가 N-1 이 되는 점이 다르다. 지연 시간은 log2N 이다.

V. 검토 및 결 론

고속 Hadamard 알고리즘을 구현하기 위한 기존의 단일 프로세서 구조, 파이프라인 구조의 소요 메모리 크기 및 성능을 분석하고 파이프라인 구조에서 프로세서의 효율을 개선하기 위한 구조를 제시하였으며 연산의 병렬성을 향상 시키기 위한 2진 Tree 구조에 대하여 성능 및 프로세서의 수, 버퍼 메모리의 크기를 분석하였다. 또한 일반적으로 알려진 FHT 알고리즘의 연산 순서와 다른 수정된 FHT 알고리즘을 위한 파이프라인 구조 및 2진 Tree 구조를 설계하고 분석하였다. 여러가지 구조의 하드웨어 복잡도 및 성능을 종합하면 [표-1]과 같다.

표-1에서 볼 수 있듯이 FHT 알고리즘을 수행하는데 필요한 총 버터플라이 연산 수는 $N/2\ \log_2 N$ 으로서 단일 프로세서 구조는 $N/2\ \log_2 N$ 시간에 걸쳐 이를 수행한다. 다수의 프로세서를 사용하는 병렬 구조는 프로세서 수가 증가함에 따라 FHT 처리 시간이 반비례하여 감소하는 것이 바람직하다. 일반적인 FHT 알고리즘을 구현하는 파이프라인 구조-1 은 $\log_2 N$의 프로세서를 사용하여 연산 시간을 단축시키고 있으나 프로세서의 효율

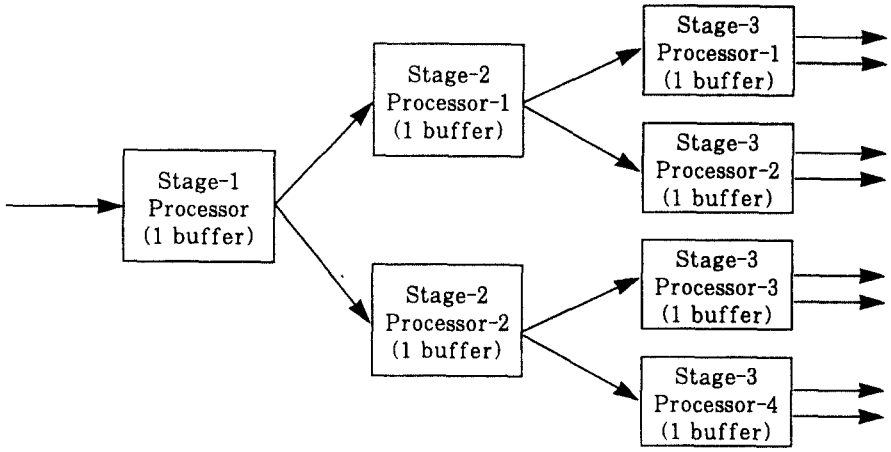


그림 11. 수정된 FHT 알고리즘을 위한 2진 Tree 구조
Fig. 11. Binary tree structure for modified FHT algroithm

표 1. FHT 구조의 하드웨어 복잡도 및 성능 비교
Table 1. Comparison of FHT structure with hardware size & performance

FHT 구조	프로세서 수	버퍼 크기	지연시간
단일 프로세서	1	N	$N/2\log_2 N$
파이프라인 구조-1	$\log_2 N$	N-1	N-1
고효율 파이프라인구조	$\log_2 N$	5/4N-2	N/2-1
2진 Tree 구조-1	N-1	$N/2\log_2 N$	$\log_2 N$
파이프라인 구조-2	$\log_2 N$	N-1	N/2-1
2진 Tree 구조-2	N-1	N-1	$\log_2 N$

이 50%에 불과하다. 프로세서의 효율을 높인 고효율 파이프라인 구조는 프로세서의 효율이 100%로서 FHT 처리 시간이 기존의 파이프라인 구조에 비해 50%로 감소되었으며 이를 이루기 위한 버퍼 메모리의 증가는 25%이다. 한편 2진 Tree 구조-1 은 가장 높은 성능을 실현할 수 있는 구조이지만 소요되는 버퍼 메모리의 크기가 너무 크다. 이에 비해 수정된 FHT 알고리즘에 기초한 파이프라인 구조-2 와 2진 Tree 구조-2 는 버퍼 메모리 크기와 처리 시간 면에서 일반적인 FHT 알고리즘을 위한 구조 보다 바람직한 것으로 나타나 있다.

이상에서 FHT 알고리즘을 위한 기존의 구조를 분석하고 하드웨어 복잡도 또는 처리 시간을 개선하기 위한 몇 가지 구조를 제시하고 비교 분석하였다. 비교 결과 짧은 처리시간이 우선적일 경우에는 수정된 FHT 알고리즘에 기초한 2진 Tree 구조가 적합하며 버퍼 메모리 크기, 프로세서 수를 고려할 경우에는 기존 파이프라인 구조의 효율을 높이는 구조보다 수정된 FHT 알고리즘으로 부터 출발한 파이프라인 구조가 바람직한 것으로 나타났다.

참고문헌

1. M. Harwit and N. J. A. Sloane, *Hadamard Transform Optics*, Academic Press, 1979.
2. N. Armed and K. R. Rao, *Orthogonal Transform for Digital Signal Processing*,

Springer-Verlag, 1975.

3. A. H. Desoky and S. A. Hall, "Entropy Measures for Texture Analysis Based on Hadamard Transform," *Proceedings of 1990 IEEE Southeastcon*, vol.2, pp.467-470, New Orleans, April. 1990.
4. C. Anshi, L. Di and Z. Renzhong, "A Research on Fast Hadamard Transform (FHT) Digital Systems," Proceedings of 1993 *IEEE Region 10 Conference on Computer, Communication, Control and Power Engineering*, vol.3, pp.541-545, Beijing, Oct. 1993.
5. 현진일, 차진종, 강인, "Efficient design method of Hadamard transformer," 대한전자공학회 추계학술대회 논문집 제17권 제2호, pp.1411-1414, Nov. 1994.
6. R. Kerr, K. Gilhousen, B. Weaver, "The CDMA Digital Cellular system an ASIC overview," *IEEE 1992 Custom Integrated Circuit Conference*, pp.10.1.1-10.1.7, May. 1992.
7. J. Hinderling, T. Rueth, K. Easton, "CDMA Mobile Station MODEM ASIC," *IEEE 1992 Custom Integrated Circuit Conference*, pp.10.2.1-10.2.5, May. 1992.

玄 鎭 一(Jin-Il Hyun) 정회원

1961年 3月 15日生
1989年 2月 : 숭실 대학교 전자공학과 학사
1989年 2月~현재 : 한국전자통신연구소 연구원
※주관심 분야 : VLSI design, DSP 및 통신용 모뎀

李 熏 馥(Hoon-Bok Lee) 정회원

1954年 4月 19日生
1977年 2月 : 서울대학교 전자공학과 학사
1979年 2月 : 한국과학원 전기및 전자공학과 석사
1988年 8月 : Arizona state uni. 전기.전산공학과 박사
1979年 2月~1995年 8月 : 한국전자통신연구소 책임연구원
1995年 9月~현재 : 한림대학교 전자공학과 부교수
※주관심 분야 : VLSI 구조 및 통신 회로, DSP 설계

車 鎭 鍾(Jin-Jong Cha) 정회원

1956年 9月 17日生
1980年 2月 : 한양대학교 전자공학과 학사
1982年 2月 : 한국과학원 전기및 전자공학과 석사
1982年 2月~1985年 4月 : 전자기술연구소 선임연구원
1985年 5月~현재 : 한국전자통신연구소 책임연구원
※주관심 분야 : 디지털 통신용 ASIC설계, 영상 통신용 ASIC설계, DSP 설계, 고속 비이폴라 디지털 회로설계

金 在 錫(Jaeseok Kim) 정회원

1955年 10月 1日生
1977年 2月 : 연세대학교 전자공학과 학사
1979年 2月 : 한국과학원 전기및 전자공학과 석사
1988年 8月 : Rensselaer Polytechnic Institute 전자공학과 박사
1979年 2月~1984年 4月 : 전자기술연구소 선임연구원
1988年 8月~1993年 5月 : AT&T Bell Lab. MTS(Member of Technical Staff)
1993年 5月~현재 : 한국전자통신연구소 VLSI구조연구실장
※주관심 분야 : VLSI design, CAD, DSP 및 통신용 모뎀

金 景 洙(Kyungsoo Kim) 정회원

1951年 12月 21日生
1977年 2月 : 서강대학교 전자공학과 학사
1977年 2月~1985年 4月 : 전자기술연구소 선임연구원
1985年 5月~현재 : 한국전자통신연구소 집적회로연구부장
※주관심 분야 : VLSI design and test, DSP 및 통신용 모뎀 설계