論文93-30A-3-12

계산 속도와 하드웨어 양이 조절 용이한 FFT Array Processor 시스템

(FFT Array Processor System with Easily Adjustable Computation Speed and Hardware Complexity)

柳 在 熙*

(Jaehee You)

要 約

본 논문에서는 간단한 multiplier - adder processing element를 여러가지 계산 속도에 따라 각각 필요한 최소한의 수만큼 중복 사용이 가능한 FFT array processor 알고리듬과 아키텍쳐가 논의된다. 이 아키텍쳐는 교체된 입력과 정상 배열된 출력을 위한 radix p constant geometry FFT 버터플라이 스테이지에 나타나는 p배의 대칭성에 바탕을 두고 있다. 또한, 높은 성능을 갖는 큰 radix FFT를 VLSI로 용이하게 구현하기 위해 간단한 적은 radix processing element를 중복 사용하여 큰 radix processing element를 구성하는 방법론이 논의된다. 제안된 아키텍쳐의 일반적인 성능 계산과, 계산속도와 하드웨어 양의 trade-off가 비교, 평가된다. 제안된 아키텍쳐를 바탕으로, radix 2, 8 point FFT processing element 칩이 설계되었으며, 그 구조와 결과가 논의된다.

Abstract

A FFT array processor algorithm and architecture which can use a minimum required number of simple, duplicate multiplier - adder processing elements according to various computation speed, will be presented. It is based on the p fold symmetry in the radix p constant geometry FFT butterfly stage with shuffled inputs and normally ordered outputs. Also, a methodology to implement a high performance high radix FFT with VLSI by constructing a high radix processing element with the duplications of a simple lower radix processing element will be discussed. Various performances and the trade-off between computation speed and hardware complexity will be evaluated and compared. Based on the presented architecture, a radix 2, 8 point FFT processing element chip has been designed and its structure and the results will be discussed.

Ⅰ. 서 론

Fast Fourier transform (FFT)이 통신, 레이다, 필터링, 영상 신호 처리, 음성 처리 등의 디지탈 신호 처리 분야에서 응용 분야가 매우 넓은 중요한 transform이라는 것은 잘 알려져 있다. 이러한 중요성 때문에 FFT 시스템을 VLSI로 효과적으로 구현하기 위한 다양한 방법이 개발되어 왔다. FFT

正會員, 弘益大學校 電子工學科
(Dept. of Elec. Eng., Hongik Univ.)
(※이 논문은 부분적으로 1992년도 교육부 지원 한국 학술 진흥재단의 자유공모과제 학술 연구 조성비에 의하여 작성되었음.)
接受日字: 1992年 1月 7日

VLSI 아키텍처는 크게 uniprocessor와 multiprocessor 아키텍처로 나눌 수 있다. 이 중에서 uniprocessor [1] FFT는 일반적으로 모든 시스템을 적은 수의 큰 칩에 집적하기 때문에 요구되는 칩의 수는 적으나, 한개의 칩에 대해 많은 양의 복잡한 설계가 요구되어 비용과 설계 시간을 증가시키고 yield를 감소시킨다. 이러한 이유와 함께 실리콘 처리 기술의 발전으로 VLSI에 의한 칩의 제조 비용이 감소함에 따라, yield가 높은 작은 칩을 반복 사용하는 multiprocessor array 아키텍처가 많은 관심을 끌고 있다. 지금까지의 FFT array 아키텍처를 VLSI 측면에서 살펴보면, 크게 직렬과 병렬 아키텍처로 나눌 수 있다. 일반적인 직렬 아키텍처 [2] 에서는, 한 버터플라이 스테이지 당 하나의 processing element (PE)가 버터플라이 계산을 순차적으로 수행한다. 그러나, 이러한 아키텍처에서는 필요한 전체 PE의 갯수는 적으나, 스테이지 간의 교체 (shuffling)를 PE 안에서 처리하여 칩 면적을 증가시킨다. 또한, 고속 시스템의 구현시 병렬 처리를 위해 간단한 하드웨어를 중복 사용하기 용이한 VLSI의 장점을 이용할 수 없어 계산 속도를 향상시키기 어렵다. 일반적인 병렬 아키텍처 [3 4] 에서는, radix p, n point FFT를 위해 버터플라이 스테이지 당 n/p 개의 PE가 요구된다. 교체가 스테이지 간의 interconnection에 의해 이루어지므로 PE 칩이 간단해지며, 병렬 처리에 의해 실행 시간은 매우 빠르지만, 길고 복잡한 interconnection을 필요로 하며, VLSI 구현시 많은 양의 하드웨어가 요구된다. 또한 큰 radix로 처리시 전체적인 PE와 버터플라이 스테이지의 갯수는 감소시킬 수 있으나 PE 자체가 복잡해진다. Prime factor FFT [5] 에서는 prime factor 조절에 의해 계산 시간과 하드웨어의 양을 조절할 수 있으나, 각 factor에 대해 서로 다른 PE가 요구되어 시스템 설계를 복잡하게 한다. SIMD 아키텍처 [6] 에서는 control unit이 필요하여 설계가 복잡해지며 전체적인 시스템 비용을 증가시킨다.

본 논문에서는 위의 모든 문제점을 해결하기 위해 radix의 선택에 따라 병렬처리 정도를 조정하여, 각 구현 환경에 맞추어 조정 용이한 계산 속도와 하드웨어 양을 갖음과 동시에, 간단하고 동일한 PE 칩을 반복 사용하므로써 VLSI 구현을 용이하게 하는 FFT 알고리즘과 아키텍처가 제안된다. 이는 또한 multiprocessor 구현시 시도되고 있는 wafer scale integration (WSI) 또는 wafer scale packaging (WSP)에서 요구되는 시스템의 조건과 동일하다. 먼저, Ⅱ절에서 FFT 알고리듬의 수학적 유도를 위한 정의가 간단히 소개되고, constant geometry FFT 알고리듬 (CGF)을 VLSI 구현에 알맞는 행렬 형태로 유도한다. Ⅲ절에서는 큰 radix FFT를 용이하게 구현하기 위해 적은 radix PE로 큰 radix 시스템을 구성하는 방법론이 소개된다. Ⅳ절에서는 개발된 알고리듬을 바탕으로 병렬처리 정도가 조절 가능하고 적은 radix PE로 고속의 큰 radix 시스템을 구성하는 아키텍처에 대해 논의된다. Ⅴ절에서는 제안된 아키텍처에 대해 latency, throughput 그리고 필요한 전체 PE의 갯수가 계산되고, 서로 비교된다. Ⅵ절에서 제안된 아키텍처의 장점이 요약되고, 마지막으로 Ⅶ절에서 제안된 아키텍처의 VLSI 구현을 실증하기 위해 실험용 radix 2, 8 piont FFT PE 칩의 설계 결과에 대해 논의된다.

Ⅱ. constant geometry FFT 알고리듬

CGF는 모든 버터플라이 스테이지가 동일한 구조로 이루어져 있으므로 한개의 버터플라이 계산을 위한 하드웨어를 중복 사용할 수 있어 FFT array 아키텍처에 특별한 장점을 지니고 있다. 먼저, CGF를 수학적으로 표시하기 위해 다음과 같이 정의한다.

정의 1. A가 p x q 행렬이고, B가 m x n 행렬이면, Kronecker product, A ⊗ B는 pm x qn 행렬이 되며, 다음과 같이 정의 된다.

$$A \otimes B = \begin{bmatrix} a_{(0)}B & a_{(0)}B & \cdots & a_{(0,q-1)}B \\ \vdots & \vdots & \vdots & \vdots \\ a_{p-1,0}B & a_{p-1,1}B & \cdots & a_{p-1,q-1}B \end{bmatrix}_{pm \times qn}$$

정의 2. d가 1 x n 행렬이고, A = diag(d) = diag(d0, d1, ⋯, dn-1)은 d0, d1, ⋯, dn-1을 diagonal entry로 갖는 대각 행렬이다.

정의 3. π는 교체 연산자이다. $\pi(p,n)$이 행렬의 앞에서 곱해지면, 행렬의 행이 교체되고, 뒤에서 곱해지면, 행렬의 열이 교체된다. 즉, FFT 입력 또는 출력의 교체를 위해 n = pm이라 하면,

$\pi(p,n)(x(0), x(1), \cdots, x(n-1))^T$
= (x(0), x(m), x(2m), ⋯, x((p-1)m), x(1), x(m+1), ⋯)^T

(x(0), x(1), ⋯, x(n-1)) $\pi(p,n)$
= (x(0), x(p), x(2p), ⋯, x((m-1)p), x(1), x(p+1), ⋯)

만일, $n = p^t$이고, F_p가 p point FFT 행렬이라면, 위 정의로부터 다음의 보조 정리들이 얻어질 수 있다.

보조 정리 1. $:\pi(p,n)(I_m \otimes F_p \otimes I_k)\pi(p,n)^T = I_{m/p} \otimes F_p \otimes I_{pk}$

보조 정리 2. $:(\pi(p,n)^q)^{-1} = (\pi(p,n)^q)^T$

보조 정리 3. $:\pi(p,n)^T = \pi(n/p,n)$

보조 정리 4. $:\pi(p,n)^t = I_n$

VLSI 구현에 적합한 CGF 알고리듬의 유도를 위해 n point Discrete Fourier Transform (DFT) F_n은 행렬 형태로 (1)과 같이 나타낼 수 있다.

$$F_n = (f_{pq}) \quad \text{단, } p : \text{row index},\ q : \text{column index},\ f_{pq} = W_n^{pq} = \exp(-i2\pi\, pq/n),\ i = \sqrt{-1} \qquad (1)$$

(1)로부터 교체된 입력과 정상 배열된 출력을 위한 radix p, n point Cooley - Tukey 분해 알고리듬 [7, 8] 은 (2)와 같이 나타낼 수 있다. 이에 관한 유도 과정은 생략하기로 한다.

$n = p^t$이라 하면,

$$F_nP(p) = A_tA_{t-1}\cdots A_1$$

$$A_q = (I_n/L_q \otimes (F_q \otimes I_m)) \cdot \mathrm{diag}(I_m, \triangle_m, \cdots, \triangle_m^{p-1}) = (I_n/L_q \otimes (F_p \otimes I_m))(I_n/L_q \otimes \mathrm{diag}(I_m, \triangle_m, \cdots, \triangle_m^{p-1}))$$

단, $L_q = p^q$, $m = L_q/p$, $\triangle_m = \mathrm{diag}(1, W_{Lq}, \cdots, W_L^{m-1}{}_q)$

$W_{Lq} = \exp(-i2\pi/L_q), q = 1, 2, \cdots, t \qquad (2)$

(2)에서 P(p)는 입력 digit reversal 교체 연산자이고, (3)과 같이 나타낼 수 있다.

$$P(p) = (I_n/L_t \otimes \pi(p,L_t))\cdots(I_n/L_1 \otimes \pi(p,L_1)) \qquad (3)$$

단, $L_q = p^q$, I_n/L_q 는 n/L_q x n/L_q의 단위 행렬이다.

이제, 위의 정의 1-3, 보조 정리 1-4와 (1), (2)를 바탕으로 CGF를 다음과 같이 유도할 수 있다. (2)의 A_q에서 첫번째 항, $I_{n/L} \otimes (F_p \otimes I_m)$을 보조 정리 2, 1, 2, 4를 차례로 적용하여 변형한 후, (2)의 A_q에 대입하면

$$A_q = \pi(p,n)^q (F_p \otimes I_{n/p}) \pi(p,n)^{q^T} (I_n/L_q \otimes \mathrm{diag}(I_m, \triangle_m, \cdots, \triangle_m^{p-1})$$

위의 A_q에 대해 B_q와 C_{q-1}을 다음과 같이 두고, C_{q-1}에 보조 정리 2, 2, 4, 1을 차례로 적용하면,

$$B_q = (F_p \otimes I_{n/p})\pi(p,n)T$$

$$C_{q-1} = \pi(p,n)^{(q-1)T} (I_n/L_q \otimes \mathrm{diag}(I_m, \triangle_m, \cdots, \triangle_m^{p-1}))\pi(p,n)^{(q-1)}$$

$$= \pi(p,n)\pi(p,n)^{qT} (I_n/L_q \otimes \mathrm{diag}(I_m, \triangle_m, \cdots, \triangle_m^{p-1}))\pi(p,n)^q\pi(p,n)^T$$

$$= \pi(p,n)(\mathrm{diag}(I_m, \triangle_m, \cdots, \triangle_m^{p-1}) \otimes I_n/L_q)\pi(p,n)^T$$

$$= \pi(p,n)\,\mathrm{diag}(I_{n/p}, \triangle_m \otimes I_n/L_q \cdots, \triangle_m^{p-1} \otimes I_{n/L})\pi(p,n)^T$$

위의 C_{q-1}에서 대각 행렬의 좌, 우측에서 $\pi(p,n)$, $\pi(p,n)^T$ 교체를 해주었으므로 C_{q-1}은 역시 대각 행렬이 됨을 알 수 있다. 이제, VP_q를 다음과 같이 둔다.

$$VP_q = B_q \cdot C_{q-1} = (F_p \otimes I_{n/p})\,\mathrm{diag}(I_{n/p}, \triangle_m \otimes I_{n/L}, \cdots, \triangle_m^{p-1} \otimes I_{n/L})\,\pi(p,n)^T$$

위의 VP_q는 구하고자 하는 radix p constant geometry 분해 행렬이다. 여기에 보조 정리 4를 적용하면, (4)와 같이 constant geometry FFT를 얻을 수 있다.

$$F_nP(p) = A_tA_{t-1}\cdots A_1 = \pi(p,n)^t\, VP_tVP_{t-1}\cdots VP_1 = VP_tVP_{t-1}\cdots VP_1 \qquad (4)$$

Ⅲ. 큰 radix FFT의 구현 알고리듬

FFT 시스템의 하드웨어 복잡도 및 양과 계산속도의 조절을 위해 본 논문에서는 radix 조절에 따라 병렬처리 정도를 조절시켰다. 즉 FFT 계산시 radix가 증가할수록 하드웨어 양과 계산속도가 증가된다. 이에 대해서는 Ⅳ절에서 후술한다. 그러나 고속 시스템을 위한 throughput이 높은 큰 radix 시스템은 VLSI 구현이 힘든 복잡한 PE를 필요로 하는 단점이 있다. 따라서 이절에서는 Ⅱ절에 설명된 CGF의 장점을 유지하면서, 큰 radix 시스템을 구현상 보다 편

리하고 간단한 적은 radix PE의 중복사용으로 만들 수 있는 알고리듬이 소개된다. 편의상 (4)에 나타난, radix p, n point constant geometry에 의해 분해된 행렬 VP_q를 반복하여 다음에 나타내었다.

$$VP_q = (F_p \otimes I_{n/p})\ \mathrm{diag}(I_{n/p},\ \triangle_m \otimes I_{n/L_q},\ \cdots,\ \triangle_m^{p-1} \otimes I_{n/L_q})\ \pi\ (p, n)^T$$

단 $L_q = p^q$, $m = L_q/p$, $\triangle_m = \mathrm{diag}(1,\ W_{L_q},\ \cdots,\ W_{L_q}^{m-1})$

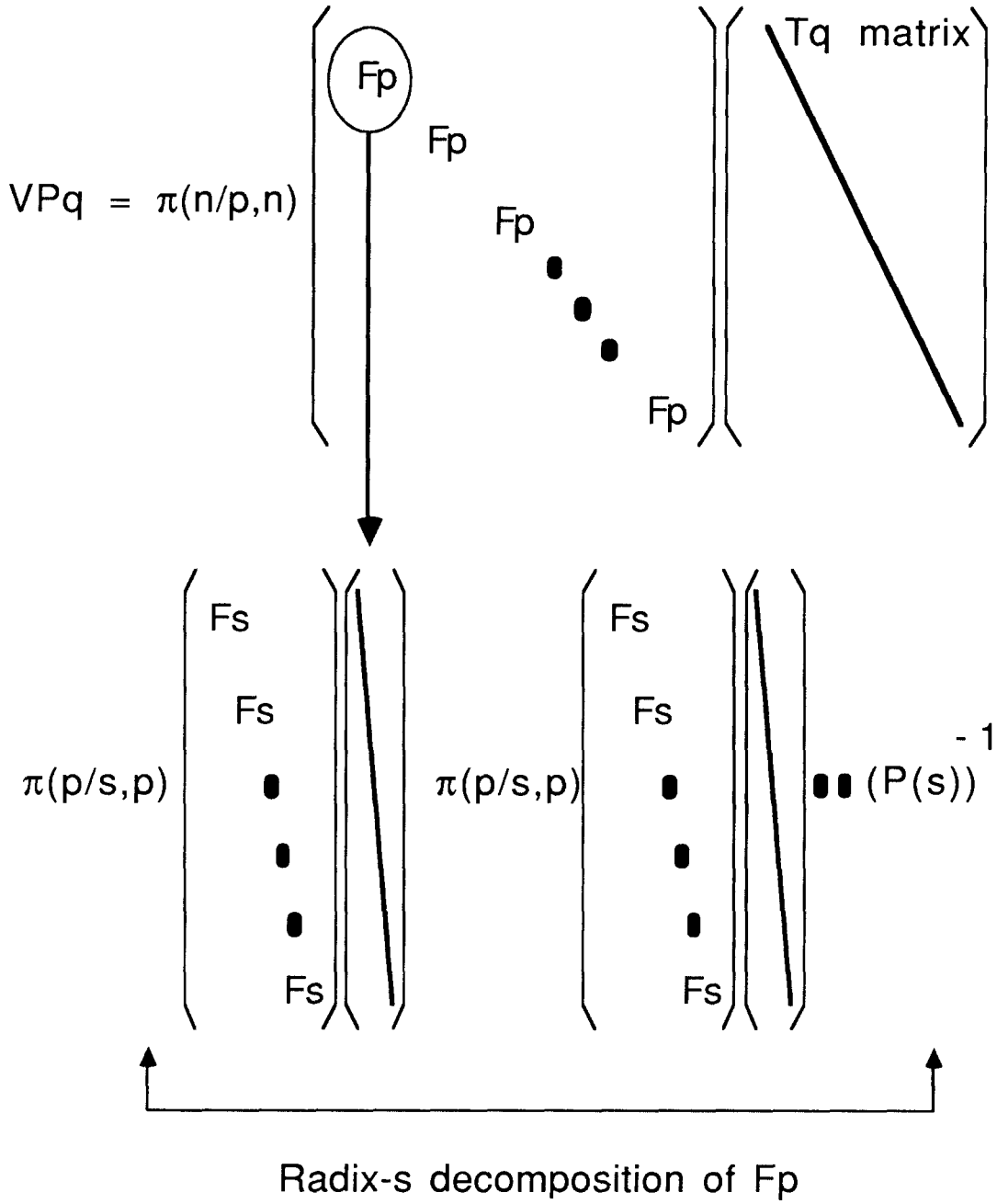

그림 1. 큰 radix FFT 버터플라이를 적은 radix FFT 버터플라이에 의해 분해하는 과정
Fig. 1. Decomposition of a high radix FFT butterfly into lower radx FFT butterflies.

$$W_{L_q} = \exp(-i2\pi / L_q)$$

VP_q에서, $G_q = \mathrm{diag}(I_{n/p}, \triangle_m \otimes I_{n/L_q}, \cdots, \triangle_m^{p-1} \otimes I_{n/L_q})\ \pi(p,n)^T$ 라 두면,

$$VP_q = (F_p \otimes I_{n/p})\ G_q$$

이제, 위 식에 보조 정리 1, 2, 2, 4, 3을 차례로 적용하면

$$\begin{aligned} VP_q &= \pi(p,n)^{(t-1)}\ (I_{n/p} \otimes Fp)\ \pi(p,n)^{(t-1)T} \cdot G_q \\ &= \pi(p,n)^T \cdot (I_{n/p} \otimes F_p)\ \pi(p,n) \cdot G_q \\ &= \pi(n/p,n) \cdot (I_{n/p} \otimes F_p)\ \pi(p,n) \cdot G_q \end{aligned}$$

다시, $R_q = \pi(n/p,n) \cdot (I_{n/p} \otimes F_p)\ \pi(p,n)$, G_q 라 한다. T_q는 대각 행렬의 좌우측에 각각 $\pi(p,n)$과 $\pi(p,n)^T$를 곱해 주었으므로 역시 대각 행렬이다.

$$VP_q = R_q T_q = \pi(n/p,n)\ (I_{n/p} \otimes F_p)\ T_q \qquad (5)$$

(5)는 변형된 분해 행렬을 나타내며 이를 바탕으로 한 행렬 VP_q의 구조가 그림 1에 보여진다. 그림 1에서 나타난 바와 같이 VPq는 좌측에 $\pi(n/p,n)$, 우측에 대각 행렬 T_q, 그리고 가운데에 대각 방향으로 F_p를 갖는 블록 대각 행렬로 분해될 수 있다. 이것은 F_n이 모두 F_p로 분해되었듯이, 각각의 F_p 역시 더 적은 radix s(단, $p = s^l$)를 갖는 F_s로 다시 분해될 수 있으며, 이러한 과정은 명백히 radix s보다 적은 radix로의 Fs의 분해로 반복적으로 확장될 수 있다. VLSI 측면에서 보면, $(n/p,n)$은 스테이지간의 interconnection으로 처리할 수 있고, 분해된 나머지 행렬들은 모두 대각 행렬들로 이루어져 있으므로 매우 용이하게 구현 가능하다.

Ⅳ. FFT 아키텍쳐

Ⅱ절의 (4)에 표시된 radix p, n point CGF에 대한 하나의 버터플라이 스테이지가 그림 2에 보여진다. CGF이므로 다른 버터플라이 스테이지도 같은 구조로 이루어지며 단지 twiddle factor만 다르다. 병렬처리 정도가 조절 가능한 CGF 아키텍쳐를 개발하기 위해 다음과 같은 사실에 유의할 필요가 있다. 즉, 각 radix p 스테이지에서 CGF의 성질에 의한 동일한 데이타 교체에 의해 그림 2에 나타난 바와 같이 출력이 p개의 동일한 그룹으로 나뉘어질 수 있음을 알 수 있다. 그 각각의 출력 그룹은 n/p개의 연속된 데이타를 갖는다. 그러므로, radix p CGF 버터플라이 스테이지는 p개의 동일한 PE들에 의해 병렬로 계산될 수 있음을 알 수 있다. 이때 radix p가 증가될수록 병렬 처리 정도가 증가되어 계산 속도가 증가된다. p개의 PE의 출력은 다음 버터플라이 스테이지에 있는 p개 PE의 입력이 되어 파이프라인식으로 계산될 수 있다. FFT length n에 대해 $n = p^l$을 만족하는 한 radix p는 어떠한 값을 가져도 상관없으므로 각 radix p에 따라 한 스테이지 당 PE의 갯수를 조정하여 계산 속도와 하드웨어의 양을 구현 환경에 알맞게 조정할 수 있다.

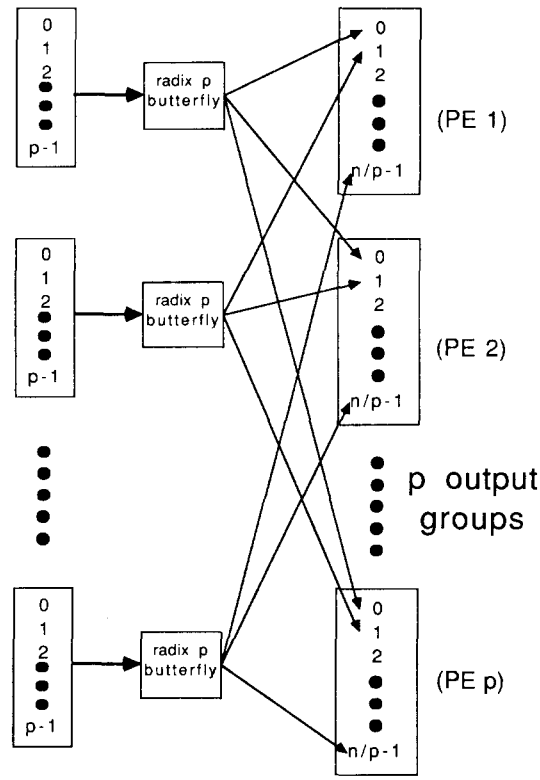

그림 2. Radix p constant geometry FFT 버터플라이 구조

Fig. 2. Radix p constant geometry FFT butterfly structure.

그림 3은 p배의 대칭성을 이용하여 제안된 아키텍쳐이다. $\log_p n$ 버터플라이 스테이지 중에서 단지 2개의 스테이지만이 보여진다. 다른 버터플라이 스테이지도 CGF의 성질에 의해 동일한 구조로 이루어진다. 그림 3에서 보여지는 각 PE는 n/p개의 연속적인 입력 데이타를 p개씩 순차적으로 처리한다. 한개의 버터플라이 스테이지 내에서는 p개의 버터플라이가 p

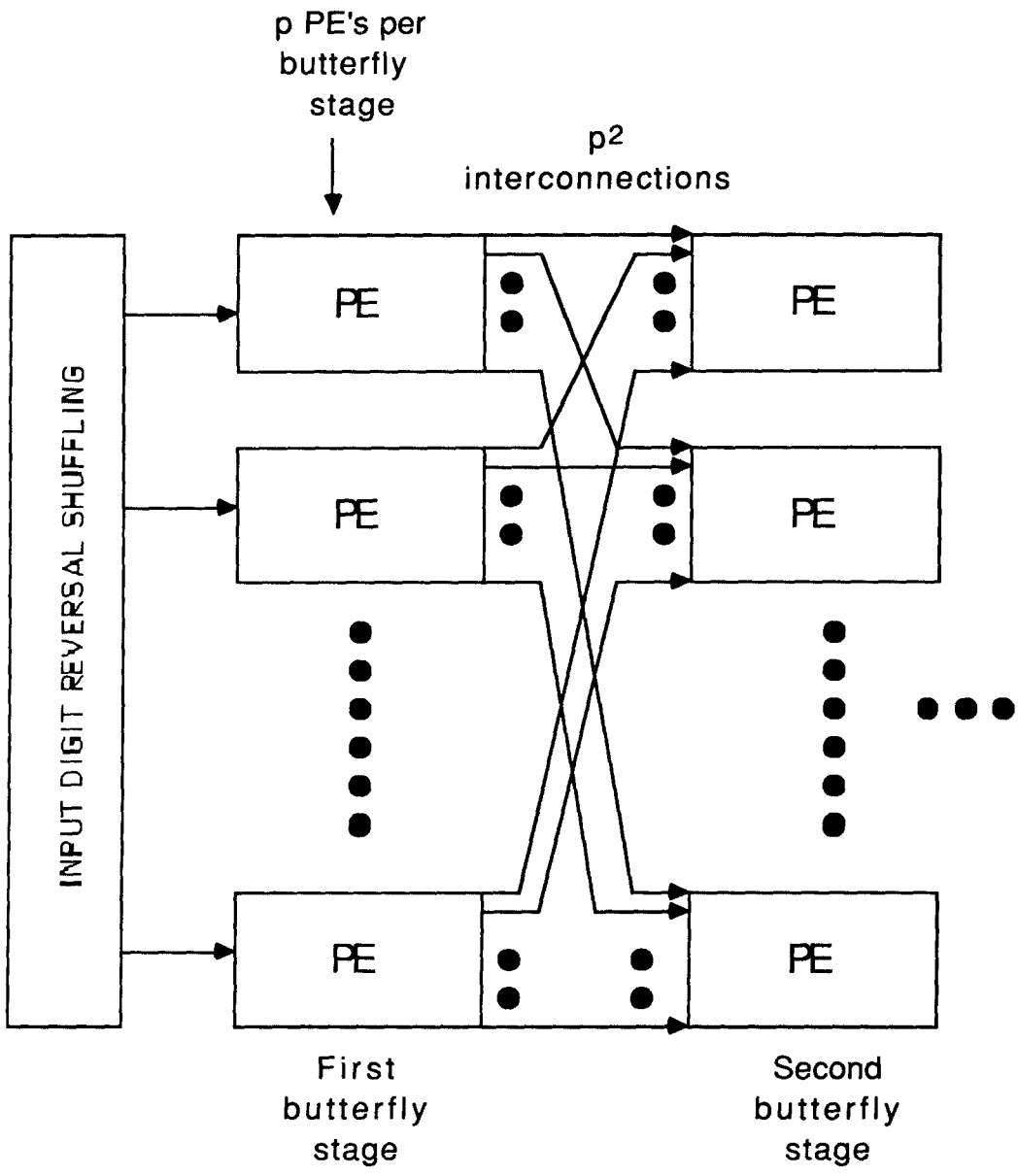

그림 3. Radix p 버터플라이 스테이지를 구현하기 위한 아키텍처
Fig. 3. An architecture to implement a radix p butterfly stage.

개 PE에 의해 병렬적으로 계산된다.

모든 다른 FFT 아키텍처와 마찬가지로 본 논문에서 제안된 아키텍처에서도 버터플라이 스테이지 간에 데이타 shuffler가 필요하다. 그림 4는 수정된 데이타 shuffler를 나타낸다. 한 버터플라이 스테이지 내에서 인접된 PE가 처리하는 데이타 사이에는 n/p데이타의 거리가 있으며, 스테이지간의 교체에 의해 n/p^2의 거리를 가진 출력을 발생 시킨다. 만일 이 출력이 다음 스테이지의 PE로 바로 전달될 경우 다음 스테이지의 출력은 n/p^3의 거리를 두고 모든 출력이 계산될 때까지 저장한다. 그러므로 데이타 shuffler가 각 스테이지마다 출력을 n/p^2의 거리를 두고 모든 출력이 계산될 때까지 저장한다. 그리고 나서 순차적으로 다음 스테이지의 PE들로 데이타를 입력한다. 그림 4에 나타난 바와 같이 데이타 교체 형태도 한 스테이지 당 p배의 대칭성에 의한 p개의 동일한 부분

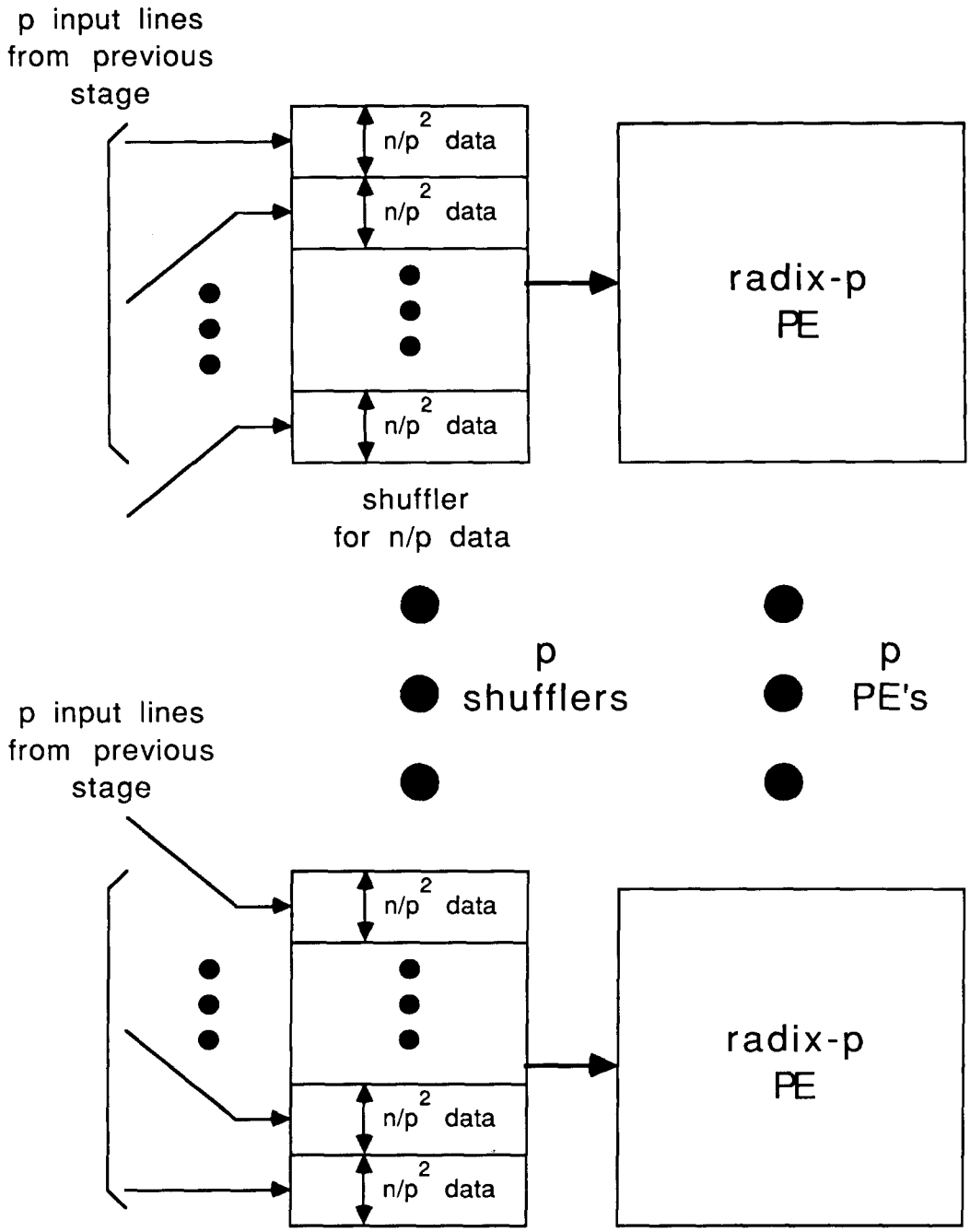

그림 4. Radix p, n point FFT를 위한 데이타 shuffler 구조
Fig. 4. Data shuffler structure for a radix p, n point FFT.

으로 구성되어 각 PE 안에 포함될 수 있다.

그림 5에 제안된 아키텍처를 위한 PE의 구조를 나타내었다. p^2 입력 레지스터는 버터플라이 계산에 필요한 p개의 연속된 입력을 p개의 interconnection에

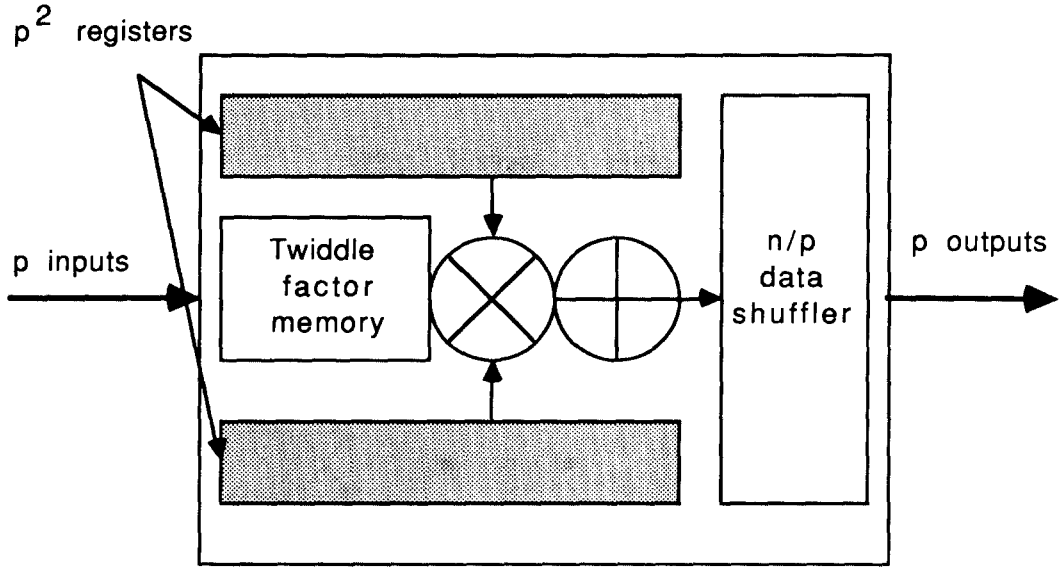

그림 5. Radix p, n point FFT를 위한 PE 구조
Fig. 5. PE structure for a radix p, n point FFT.

의해 받아 버터플라이 계산회로로 보낸다. Twiddle factor memory에 저장되어 있는 twiddle factor와 입력에 의해 multiplier와 adder에서 버터플라이 계산이 이루어진다. 계산된 출력은 다음 버터플라이 스테이지로 출력을 순차적으로 공급하기 위해 위에서 설명된 n/p 데이타 shuffler에서 교체된 후 보내진다.

위에서 설명된 아키텍처의 계산 속도를 향상시키기 위해서 병렬처리 정도를 증가시키기 위해서는 큰 radix 시스템이 필요하다. 이때 VLSI 구현상 단점인 큰 radix PE를 적은 radix PE로 용이하게 구현하기 위해 Ⅲ절의 (5)에 나타낸 알고리듬을 이용한다. (5)를 구현하기 위한 일반적인 아키텍처를 설계 하기위해 다음과 같이 가정한다. 즉, n_k point FFT는 radix n_{k-1} PE에 의해 이루어지며 (단, $n_k=(n_{k-1})^{t_{k-1}}$), 각 radix n_m PE는 n_{m-1} PE (단, $n_m=(n_{m-1})^{t_{m-1}}$, m = 1, …, k-1)에 의해 연속적으로 분해되어 이루어진다. 위의 가정에 의한 표기를 바탕으로 아키텍처를 그림 6에 나타 내었다.

$\log n_{k-1} n_k$ 의 스테이지에서 단지 하나의 스테이지만 나타내었다. 그림 1에서 각각의 Fp는 교체 연산자 π (n/p,n)에 의해 같은 교체 구조를 갖는 그림 2의 radix p 버터플라이에 대응될 수 있다. 그러므로 Ⅳ절에서 보여진 대칭성은 그대로 보존되며 그림 3의 아키텍처 역시 이용될 수 있다. 그러므로 n_{k-1}배의 대칭성에 의해 한개의 버터플라이 스테이지는 n_{k-1}개의 radix n_{k-1} PE로 이루어져 있다. 각 radix n_{k-1} PE들은 radix converter와 radix n_{k-2} PE, radix n_{k-3} PE… 들로 (5)식을 이용하여 반복적으로 구현된다. Radix converter는 큰 radix PE를 적은 radix PE들로 구현하기 위한 하드웨어로써 그림 1에 나타난 T_q의 대각선 방향 entry를 곱하기 위한 multiplier와 FFT 행렬의 반복적인 분해를 위한 P^1 연산자에 대응되는 하드웨어인 local 데이타 shuffler로 구현할 수 있다. 각각의 radix n_m PE를 이루고 있는 radix n_{m-1} PE들은 모두 n_m point FFT를 계산하기 위해 사용되기 때문에 모두 같은 twiddle factor에 의해 구현될 수 있어 VLSI 구현상 매우 용이하다. 단지, radix converter 내에 있는 T_q의 entry들만이 다르다.

Ⅴ. 아키텍처 성능 평가

본 논문 그림 3과 그림 6에서 제안된 아키텍처의 성능을 수학적으로 평가하기 위해, 다음과 같은 VLSI 구현상의 일반적인 가정을 한다. 즉, 실시간 신호 처리를 고려하여, 입력은 clock cycle 당 하나씩 교체된 상태로 입력된다. 또한 FFT 시스템을 이루고 있는 PE는 two level pipelining [9] 되어 있다고 가정된다. 즉, 전체 시스템 뿐만 아니라, PE 자체도 파이프라인되어 있다고 가정한다. PE의 latency는 d이다. PE의 출력 cycle time(1/

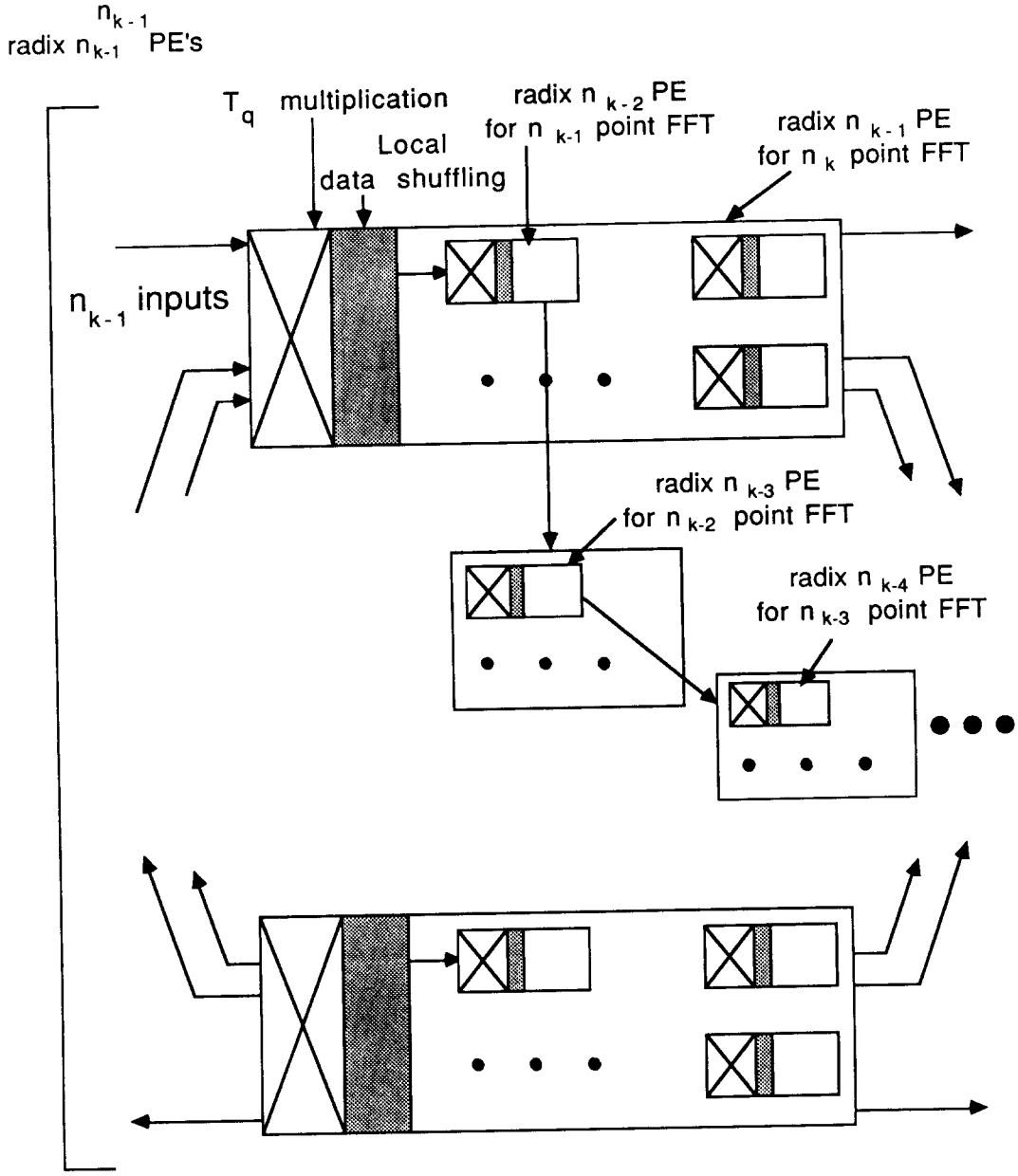

그림 6. 적은 radix PE로 큰 radix FFT를 구현하기 위한 아키텍처
Fig. 6. An architecture to implement a high radix FFT with lower radix PE's.

throughput). 즉 첫번째 버터플라이를 계산한 뒤, 다음 버터플라이를 파이프라인 방법으로 계산하는데 걸리는 시간은 r이다. 만일, PE 자체가 파이프라인 되어 있지 않다면, r = d가 된다. 먼저, 각 radix PE가 보다 적은 radix PE들로 이루어지지 않고, 한 개의 하드웨어로 이루어진 경우를 생각해 본다. 전체 FFT 시스템 latency의 계산은 첫번째 입력이 들어온 후 하나의 버터플라이 스테이지 전체가 계산되는 시간을 구하여 스테이지의 갯수 ($\log_p n$)를 곱하여 구할 수 있다. 우선 각 PE의 latency가 d이며 한 스테

이지 당 한개의 PE가 n/p^2 버터플라이를 병렬적으로 계산하므로, 나머지 $n/p^2 - 1$ 버터플라이를 전부 계산하기 위해서는 $(n/p^2 - 1)r$이 더 필요하다. 그러므로 전체 시스템의 latency, L은 (6)과 같이 나타낼 수 있다. 각, PE 내의 파이프라인 스테이지가 모두 작동하기 시작한 후 각 PE는 r마다 한개의 버터플라이를 계산하므로, 출력 cycle time, R과 throughput, T는 (7)과 같이 나타낼 수 있다. 그리고, 한개의 스테이지가 p개의 PE로 이루어지므로, 전체 시스템에 필요한 PE의 갯수, M은 (8)과 같이 나타낼 수 있다.

$$L = (\log_p n)(d + (n/p^2 - 1)r) \quad (6)$$

$$T = 1/R = p^2/(nr) \quad (7)$$

$$M = p \log_p n \quad (8)$$

그림 6에서 보여진 바와 같이 적은 radix PE들로 큰 radix 시스템을 이루는 아키텍처의 성능을 계산하기 위해 (6), (7) 그리고 (8)이 반복적으로 사용된다. Ⅳ절에서 쓰여진 일반적인 가정을 바탕으로 m = 1, …, k일때, n_m point FFT (또는 radix n_m PE)를 radix n_{m-1} PE에 의해 계산할 경우, latency (L_m), throughput (T_m) 그리고 가장 적은 radix PE들의 갯수 (M_m)는 (9), (10), (11)과 같이 나타낼 수 있다. (9)에서 LM_{m-1}은 radix n_{m-1} PE를 radix n_{m-2} PE에 의해 구현시 radix converter에 의한 latency를 나타낸다.

$$L_m = (\log_{n_{m-1}} n_m) \cdot (LM_{m-1} + L_{m-1} + (n_m/(n_{m-1})^2 - 1)R_{m-1}) \quad (9)$$

$$T_m = 1 / R_m = (n_{m-1})^2 / (n_m R_{m-1}) \quad (10)$$

$$M_m = \prod_{k=2}^{m} n_{k-1} \log_{n_{k-1}} n_k \quad (11)$$

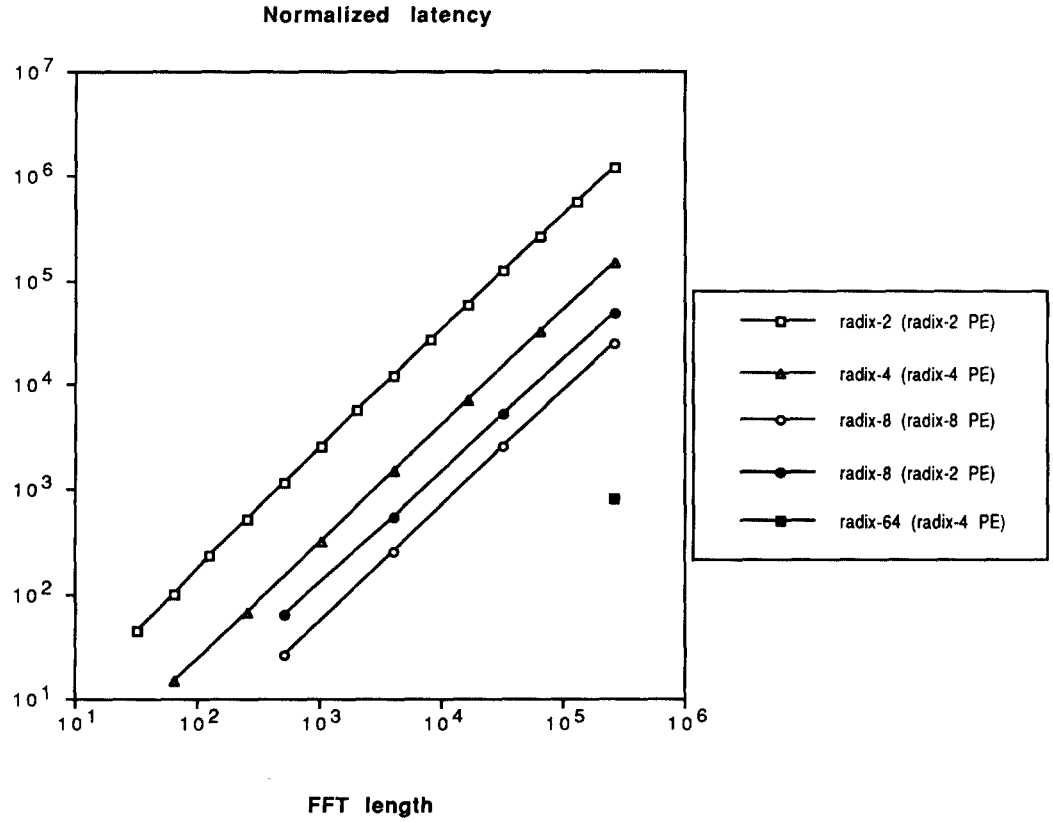

그림 7. a) Latency 비교
Fig. 7. a) The comparison of latencies.

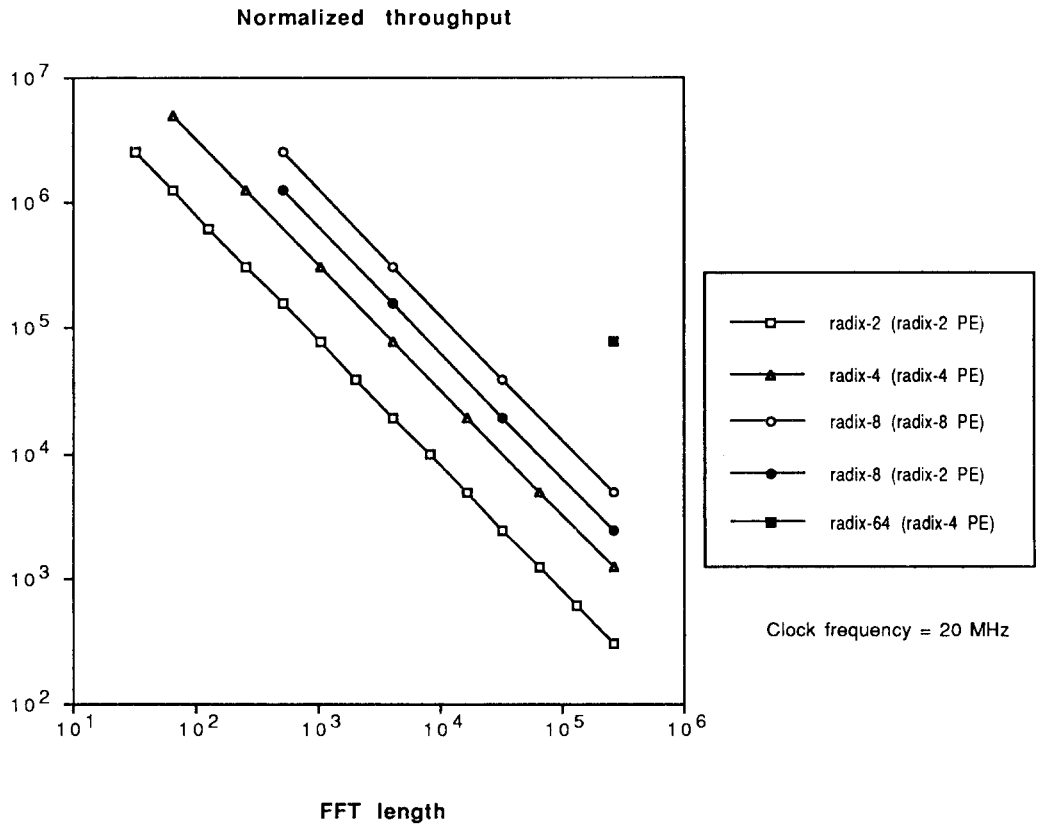

그림 7. b) Throughput 비교
Fig. 7. b) The comparison of throughputs.

(9)와 (10)에 나타낸 바와 같이 radix converter는 단지 시스템의 latency를 각 스테이지당 LM_{m-1}만큼 증가시키며, throughput에는 영향을 미치지 않는다. LM_{m-1}은 radix converter의 구현방법에 따라 다르나, 한개의 multiplier 지연시간 정도를 가지므로 시스템 성능에 큰 영향을 미치지 않음을 알 수 있다.

(6) - (11) 식을 바탕으로, 그림 7의 a), b), c)에서 각 경우의 latency, throughput, PE의 갯수에 대한 비교를 나타내었다. 여기서 radix 2, radix 4, radix 8 PE에 대해 d = 2, r = 1, LM_{m-1} = 2라고 가정되었다. 그림 7의 성능 평가 결과는 다음과 같이 설명할 수 있다. 즉, radix가 증가함에 따라 latency는 감소하고, throughput은 증가하여 계산 속도가 향상되며, radix가 감소함에 따라 하드웨어양이 감소한다. 또한 (5)를 바탕으로한 아키텍처 경우에 있어, radix 2 PE로 이루어진 radix 8 시스템의 경우 radix 8 PE로 이루어진 시스템과 거의 비슷한 latency와 throughput을 얻을 수 있다. Radix 4 PE에 의한 radix 64 시스템은 radix 4 PE로 이루어진 radix 4 시스템보다 우수하다. 또한 radix 4 PE보다 VLSI 구현이 어려운 radix 8 PE의 경우보다 더 좋은 성능을 얻을 수 있다. 그러나, 적은 radix PE들로 큰 radix FFT 시스템을 구현할 때 전체 PE의 갯수는 그림 7. c)에 나타낸 바와같이 증가한다. 그러나 간단하고 적은 PE의 반복 사용이 복잡한 PE보다 VLSI 구현상 간단함은 서술한 바 있다.

마지막으로 benchmark에 대해 논의해 보면 512 point FFT가 radix 8 PE로 구현되고, 각각의 PE는 radix 2 PE로 구성된다고 가정하자. Radix 2 PE에 대해서 시스템 clock은 20MHz이고, d = 2 clocks, r = 1 clock, radix converter에 의한 latency를 2 clocks라고 가정하자. 그러면 한 스테

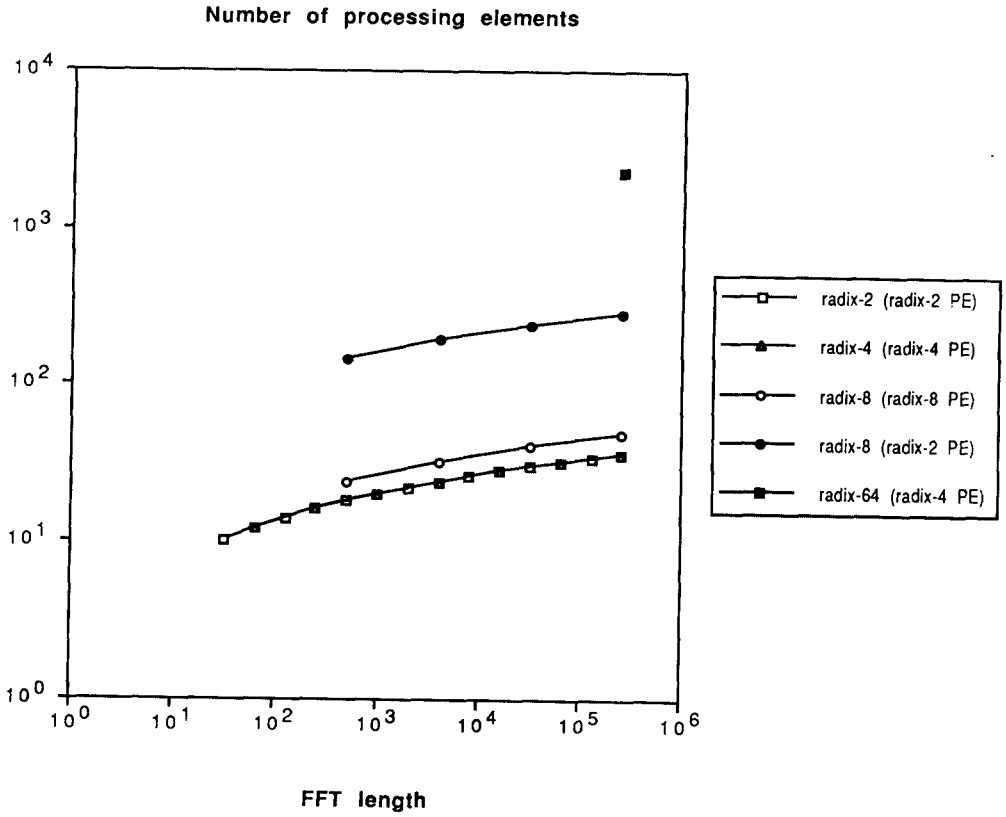

그림 7. c) PE수 비교
Fig. 7. c) The comparison of the numbers of PE's.

이지는 48개의 radix 2 PE로 이루어지고, latency는 33 us이며 0.8 us마다 연속적인 출력이 계산된다.

Ⅵ. VLSI 구현상의 장점

제안된 FFT 아키텍처의 VLSI 구현상 장점은 다음과 같이 요약될 수 있다.

1. 각 구현 환경에 맞추어 radix를 조정하여 FFT 계산의 병렬처리 정도를 조절함으로써, 최적의 하드웨어 양과 실행속도를 용이하게 얻을 수 있다. 이는 가장 경제성 있는 VLSI 구현을 얻을 수 있음을 의미한다.
2. Radix를 증가시켜 계산 속도를 향상시킬 경우 필요한 큰 radix 시스템을 간단한 적은 radix PE로 쉽게 구현할 수 있다.
3. 같은 radix에 의해 다른 FFT length를 계산할 경우에는 단지 twiddle factor memory와 데이타 shuffler만을 다시 프로그래밍하면 된다.
4. Pipelining에 의해 높은 throughput을 얻을 수 있다.
5. Global control line을 필요로하지 않고, 각 열에 있어 PE는 단지 인접한 열에 있는 PE하고만 연결되므로 interconnection이 간단하다.
6. 모든 PE들이 동일하고 간단하며 서로 독립적인 구조를 가지므로 WSI, WSP에 의한 구현시 faulty PE를 redundancy PE와 교환하기 편리하다. 그 결과 높은 수준의 fault tolerance를 갖는다. 그림 8은 스테이지 간의 interconnection을 프로그래밍함으로써 redundancy를 구현하기 위한 배치를 보여준다. 각 radix p 버터플라이 스테이지는 p개보

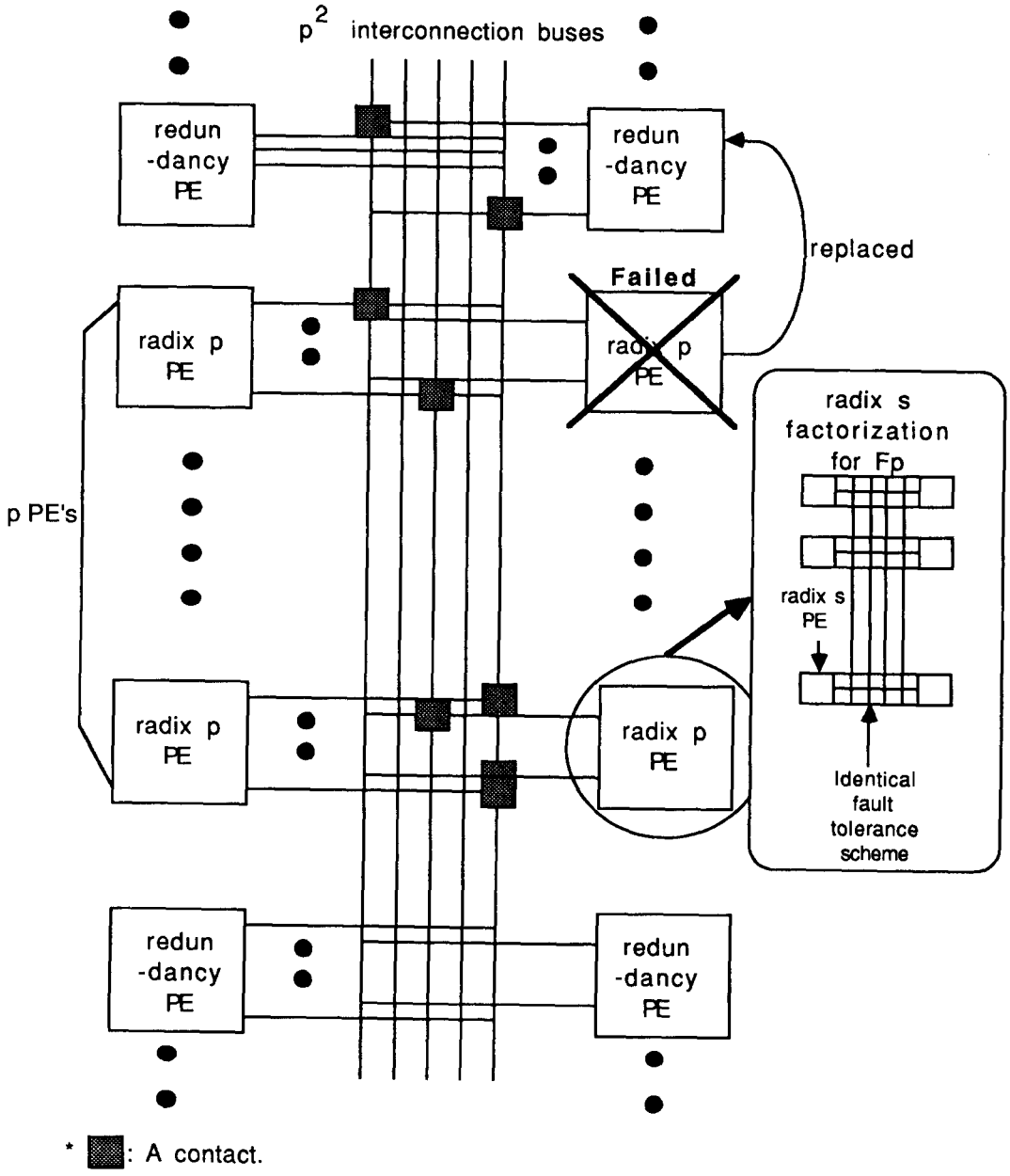

그림 8. Radix p, n point FFT 버터플라이 스테이지를 위한 fault tolerance 구조
Fig. 8. Fault tolerance structure for a radix p, n point FFT butterfly stage.

다 많은 PE로 구성되어 faulty PE는 redundancy로 대체된다. 같은 redundancy 구현 방법이 적은 radix PE에 의해 구현된 각각의 큰 radix PE 내에서 반복적으로 사용될 수 있다.

7. 실시간 신호 처리에 있어서, 입력이 raster scan order로 들어올 경우 PE의 출력 cycle time을 r clock이라 하면, p개의 PE가 병렬적으로 버터플라이 스테이지를 계산하므로 시스템은 매 r clock마다 p^2개의 입력을 계산한다. 그

러므로, 시스템과 입력 clock cycle time을 각각 ts, ti라 하면 (12)식이 성립한다.

$$t_s = (p^2/r) \cdot t_i \qquad (12)$$

그러므로, p와 r을 적절히 선택하므로써 t_s/t_i의 비를 적절히 조절시킬 수 있다. t_s/t_i의 비를 증가시키기 위해서는 p를 증가시켜야 하므로 high radix PE가 필요하나 (5)에서 설명된 방법을 이용하여 low radix PE 구현만으로 p를 쉽게 증가시킬 수 있다. t_s/t_i의 비를 증가시키므로써 입력 clock rate에 대한 FFT 시스템 clock rate를 늦추어, 넓은 면적에 다수의 PE로 구성되어 있는 시스템 내에서 모든 clock을 동기화시키는 문제 해결에 도움을 줄 수 있다.

8. 위에 서술된 장점들에 의해 특히 WSI, WSP에 의한 구현에 적합하다.

Ⅶ. 실험용 radix 2, 8 point FFT PE칩 설계

제안된 아키텍쳐의 VLSI 구현을 검증하기 위해 실험용 radix 2, 8 point FFT용 PE칩이 구현되었다. 우선 PE의 역할을 설명하기 위해 첫번째 버터플라이 스테이지의 timing을 그림 9에 나타내었다. 각 버터플라이 스테이지는 2배의 대칭성에 의해 2개의 PE에 의해 계산되며, 시스템 전체는 6개의 동일한 PE로 이루어진다. 따라서 radix 2, 8 point FFT 시스템은 부가적인 하드웨어 없이, 단순히 한번 설계된 PE를 중복 사용하여 구현될 수 있다. 각 node에 표시된 숫자들은 데이타가 각 node에 도착되는 timing을 나타낸다. 칩 면적의 제한에 의해 각 PE 칩의 구성 부분은 pipelining (two level pipelining)되지 않았으며, PE의 latency와 출력 cycle time은 1 clock으로 설계되었다. t = 2에서 두개의 PE는 입력을 병렬적으로 계산하기 시작한다. t = 3에서 첫번째 출력이 계산되고, t = 4에서 두번째 출력이 계산된다. 모든 다른 PE들도 같은 방법으로 계산한다.

그림 10에 제조된 PE칩의 세부 아키텍쳐를 나타내었다. 각 PE는 데이타 shuffler, 데이타 selector, 복소수 multiplier, adder, subtractor로 구성되어 있다. 각 PE는 두 그룹의 입력과 출력을 가지고 있으며, 입력과 출력은 각각 두개의 16 bit 데이타 (8 bit 실수와 8 bit 허수)로 구성된다. 우선, 데이타 shuffler가 계산될 2개의 버터플라이 입력을 잠시 저장한다. 데이타 select circuit이 2개의 입력 중 계산될 입력을 선택한다. Twiddle factor를 저장하기 위해서 programmable ROM이 사용되었다.

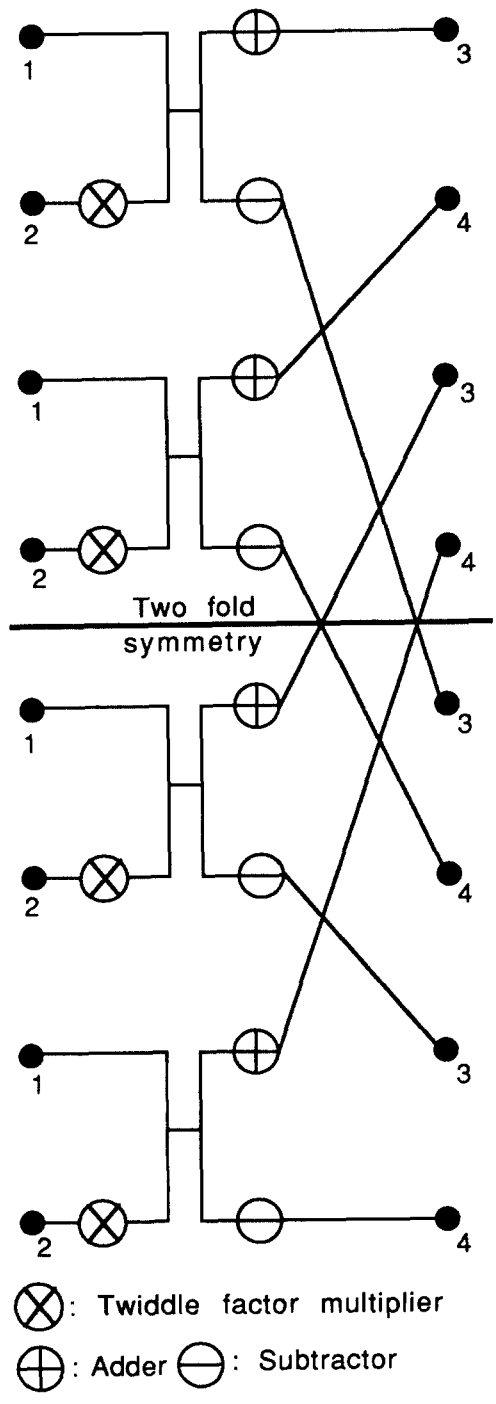

그림 9. Radix 2, 8 point FFT 버터플라이 계산 순서 및 하드웨어

Fig. 9. Computation sequence and hardware for a radix 2, 8 point FFT butterfly.

Radix 2, 8 point FFT 계산에 필요한 twiddle factor는 4 그룹으로 나눌 수 있으므로, ROM은 4개의 부분으로 구성되고, 각각의 PE를 위해서 한 부분이 선택된다. ROM access time에 의한 시간 지연을 줄이기 위해, 필요한 twiddle factor는 한 clock 먼저 fetch되어 ROM 출력 레지스터에 저장한다. 복소수 곱셈은 병렬적으로 네개의 multiplier에서 동시에 수행된다.

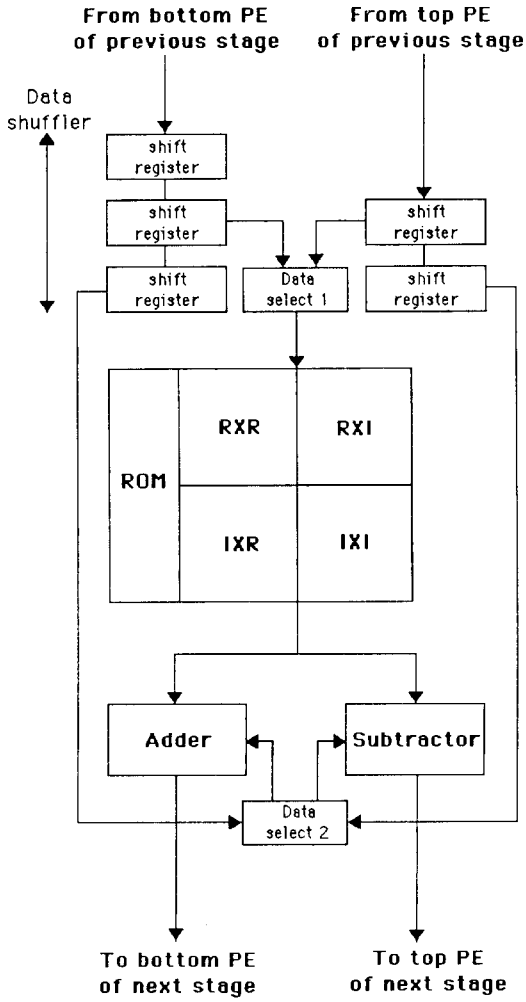

그림 10. Processing element 칩 구조
Fig. 10. The structure of a processing element chip.

그리고나서, 감산기와 가산기는 각각 x - Ay와 x + Ay를 계산한다. 이 절에서 설계된 radix 2 PE는 앞에서 설명한 바와 같이 radix 2보다 큰 radix, 8 point보다 큰 FFT length를 위한 시스템을 위한 PE로 용이하게 변형될 수 있다. PE 칩은 2 um, n well CMOS technology에 의해 만들어졌으며, 총 19000개의 트랜지스터로 이루어져 있다. 칩의 크기는 30.15 mm^2이며, I/O 핀은 총 64개로 pin grid array 패키지가 사용되었다. Layout시 Magic, simulation시 Irsim, Spice가 사용되었다. 칩의 사진을 그림 11에 나타내었다.

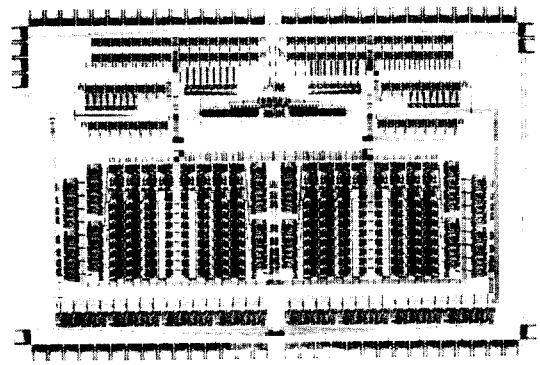

그림 11. 칩 사진
Fig. 11. Chip photograph.

Ⅷ. 결 론

Constant geometry FFT 알고리듬을 바탕으로 radix의 조절에 의한 병렬 처리 정도를 조절함으로써 하드웨어의 양과 계산속도를 쉽게 조정할 수 있으며, 간단한 PE를 중복사용하는 FFT 시스템이 제안되었다. 고속의 큰 radix 시스템을 구현할 경우 필요한 큰 radix PE를 동일한 적은 radix PE를 중복하여 구현하는 방법이 소개되었다. 제안된 아키텍처의 성능이 평가되고, 병렬 처리 정도에 의해 latency, throughput, PE 갯수가 비교, 검증되었다. 제안된 아키텍처의 VLSI 구현을 검증하기 위하여, 실험용의 raix 2, 8 point FFT PE칩이 설계, 제조되었으며, 그 결과가 논의되었다.

參 考 文 獻

[1] R. W. Linderman, C. G. Shephard, K. Taylor, P. W. Coutee, P. C. Rossbach, J. M. Collins, "A 70-MHz 1.2 um CMOS 16-point DFT Processor," *IEEE*

J. of Solid-State Circuit, vol. 23, No. 2, April, 1988.

[2] W. Liu, J. C. Duh, D. E. Atkins, "The design of a vector-radix 2D FFT chips," *IEEE Int.* Symposium on Comp. Arithmetic, pp. 231-236, 1985.

[3] K. Sapiecha, R. Jarocki, "Modular architecture for high performance implementation of FFT algorithm," *IEEE Int.* Symposium on Comp. Architecture, pp. 261-270, 1986.

[4] P. Bertolazzi, F. Luccio, VLSI: algorithms and architectures, Elsevier Science Publishing Co., Inc., New York, 1985.

[5] E. T. Chow, D. I. Moldvan, "Prime factor DFT parallel processor using wafer scale integration," *IEEE Int.* Symposium on Comp. Architecture, pp. 133-139, 1985.

[6] L. H. Jamieson, P. T. Mueller, H. J. Siegel, "FFT algorithm for SIMD parallel processing systems," J. of Parallel and Distributed Computing 3, pp. 48-71, 1986.

[7] Cooley, J. W., Tukey, J. W., "An algorithm for the machine calculation of complex fourier series," Mathematics of Computation, vol. 19, pp. 297-301, 1965.

[8] L. Rabiner, B. Gold, Theory and application of digital signal processing, Perntice Hall Inc., New Jersey, 1975.

[9] H. T. Kung, M. S. Lam, "Wafer-scale integration and two-level pipelined implementations of systolic array," J. of Parallel and Distributed Computing 1, pp. 32-63, 1984.

著者紹介

柳 在 熙 (正會員)

1963年 3月 3日生. 1985年 2月 서울대학교 전자공학과 학사. 1990年 2月 Cornell 대학교 전기공학과 공학박사 (SRAM, BICMOS 회로, FFT processor). 1990年 3月 ~ 1991年 3月 Texas Instruments, Dallas VHDL Lab. 연구원(64MEG DRAM설계). 1991年 3月 현재 홍익대학교 전자공학과 조교수. 주관심분야는 Image, Speech signal processing VLSI 아키텍쳐 및 시스템 설계, DRAM, SRAM 회로설계 등임.