論文94-31B-5-2

이차원 Constant Geometry FFT VLSI 알고리즘 및 아키텍쳐

(VLSI Algorithms & Architectures for Two Dimensional Constant Geometry FFT)

柳 在 熙*, 郭 珍 錫**

(Jae Hee You and Jin Suk Kwak)

要 約

본 논문에서는 이차원 FFT에 있어서 모든 버터플라이 스테이지가 동일한 구조로 이루어져 VLSI 구현에 매우 적합한, 교체된 입력과 정상 배열된 출력을 갖는 이차원 constant geometry FFT 알고리즘과 아키텍쳐가 소개된다. 또한, 교체된 입력과 교체된 출력을 가지나 VLSI 구현시 큰 면적이 필요한 global interconnection에 의해 연결되는 스테이지의 수를 반으로 감소시키는 이차원 constant geometry FFT 알고리즘과 아키텍쳐가 소개된다. 각 알고리즘은 한개의 스테이지에 대응되는 분해행렬의 행과 열을 교체시킴으로써 얻어진다. 또한, 위의 각 아키텍쳐에 있어서 non-recursive 또는 recursive pipeline 구조를 통하여 요구되는 시스템 구현 환경에 따라 다양하게 연산의 직·병렬 정도를 조절하는 방법이 논의된다. 고성능의 큰 radix FFT 시스템 구현을 용이하게 하며, 스테이지간의 interconnection 길이를 더욱 감소시키기 위해 작은 radix PE들로 큰 radix PE를 구현하는 방법이 설명되었다. 마지막으로, 각 아키텍쳐의 성능을 비교·평가하였다.

Abstract

A two dimensional constant geometry FFT algorithms and architectures with shuffled inputs and normally ordered outputs are presented. It is suitable for VLSI implementation because all butterfly stages have identical, regular structure. Also a methodology using shuffled FFT inputs and outputs to halve the number of butterfly stages connected by a global interconnection which requires much area is presented. These algorithms can be obtained by shuffling the row and column of a decomposed FFT matrix which corresponds to one butterfly stage. Using non-recursive and recursive pipeline, the degree of serialism and parallelism in FFT computation can be adjusted. To implement high performance high radix FFT easily and reduce the amount of interconnections between stages, the method to build a high radix PE with lower radix PE's is discussed. Finally the performances of the presented architectures are evaluated and compared.

* 正會員, 弘益大學校 電子工學科
(Dept. of Elec. Eng., Hongik Univ.)
** 準會員, 韓國電子通信研究所, 映像通信研究所
※이 논문은 1992년도 교육부 지원 한국 학술진흥재단의 자유공모과제 학술연구 조성비에 의하여 작성되었음.
接受日字 : 1993年 6月 21日

Ⅰ. 서 론

Fast Fourier Transform (FFT)은 공학의 거의 모든 분야에서 폭넓은 응용분야를 갖고 있다. FFT 알고리즘이 처음 소개된 이래로, 그 중요성으로 인해 효율적인 FFT 알고리즘의 구현을 위한 많은 노력이 수행되어 왔다. 최근에 실시간 신호 처리에 대한 요구가 증가하고, VLSI 기술이 고도로 발달함에 따라 같은 칩을 중복하기 용이한 VLSI의 장점을 최대한으로 이용할 수 있는 병렬처리 FFT 시스템 알고리즘이 특히 중요하게 되었다. VLSI 구현이라는 측면에서 기존 아키텍쳐의 특징을 살펴보면 다음과 같다. [1] 에서는 2-D FFT를, row-column transposition을 취하지 않고, 1-D FFT와 같이 linear array로 계산하였으며, Cooley-Tukey 알고리즘을 사용하여 스테이지간의 interconnection이 불규칙해지는 단점이 있다. [2] 에서는 systolic array를 이용하여 FFT 하드웨어를 설계하였으며, 엘리베이터와 비슷한 구조로 데이타 교체가 이루어진다. [3] 에서는 n point radix 2 FFT를 계산하기 위하여 한 스테이지당 n/2개의 processing element (PE)를 사용하였다. 각 스테이지 간에 serial-in parallel-out 또는 parallel-in serial-out shift 레지스터를 사용하여 latency가 증가된다. Pipeline시 속도는 빠르나, 비교적 많은 양의 하드웨어와 복잡한 interconnection이 스테이지 사이에서 요구된다. [4] 에서는 prime factor decomposition을 이용하여 비교적 융통성 있는 FFT 아키텍쳐를 구현하였으며, shift 레지스터를 사용하여 교체를 하였다. 그러나, 여러가지 다른 FFT length를 계산하기 위해서는 여러가지 다른 종류의 PE가 필요하며, 스테이지간의 interconnection이 불규칙하다. [5] 에서는 Single Instruction stream - Multiple Data stream (SIMD)을 사용하여 1-D와 2-D FFT를 계산하는 하드웨어를 제시하였다. 그러나, SIMD 시스템이 필요로 하는 control unit, 각 PE 사이의 interconnection network와 local memory가 PE 내부에 필요하며, VLSI에서 간단한 PE 칩을 여러번 사용할 수 있는 장점을 충분히 이용하지 못해 하드웨어 구현이 어렵다. 이 밖에도 많은 아키텍쳐들이 개발되어 왔으나, 대부분이 용이한 VLSI 구현으로 실시간 처리를 위한 응용 분야에는 부적당하며, 스테이지간의 불규칙한 interconnection으로 인해 복잡한 control logic을 필요로 하거나, 서로 다른 PE를 필요로 하는 등의 문제가 있다.

이에 본 논문에서는 먼저, Ⅱ절에서 모든 버터플라이 스테이지가 동일한 규칙적인 구조를 가지므로 연산에 필요한 제어 구조가 간단하며, 또한 같은 하드웨어의 중복 사용이 가능하여 VLSI 구현에 특히 적합한 이차원 constant geometry FFT (CGF) 알고리즘이 논의된다. 또한 이를 바탕으로 global interconnection에 의해 연결된 버터플라이 스테이지 수를 반으로 감소시켜 칩면적을 줄일 수 있는 알고리즘이 설명된다. Ⅲ절에서는 고성능 시스템을 위해 필요한 복잡한 큰 radix PE의 구현을 용이하게 하며, 전체 시스템의 global interconnection 복잡도를 더욱 감소시키기 위한 방법이 각각의 알고리즘에 대해 소개된다. Ⅳ절에서는 Ⅱ절의 알고리즘에 대하여 non-recursive 및 recursive 연산을 바탕으로 직병렬 정도가 조절 용이한 다양한 아키텍쳐의 구조가 소개된다. 마지막으로, Ⅴ절에서는 제안된 아키텍쳐에 대한 성능이 평가 비교된다.

Ⅱ. 이차원 CGF 알고리즘

이차원 CGF 알고리즘 유도에 기본이 되는 교체된 입력과 정상 배열된 출력을 갖는 radix p, N point 일차원 Cooley-Tukey FFT 행렬 알고리즘[6] 은 (1)과 같이 나타낼 수 있다. 식 표기에 필요한 기호와 정의, 그리고 수학식 유도에 필요한 보조정리는 Appendix A에 나타내었다. $N = p^t$라고 하면,

$$
\begin{aligned}
F_N P(p) &= A_t A_{t-1} \cdots A_1 \\
A_q &= \left(I_{N/L} \otimes F_p \otimes I_m\right)\left(I_{N/L} \otimes D_q\right) \\
D_q &= diag\left(I_m, \Delta, \cdots, \Delta^{p-1}\right) \\
\Delta &= diag(I_m, W_L, \cdots, W_L^{m-1}) \\
L &= p^q, m = L/p \\
P(p) &= \left(I_{N/p^t} \otimes \pi(p, p^t)\right) \cdots \left(I_{N/p} \otimes \pi(p, p)\right) \qquad (1)
\end{aligned}
$$

(1)에서 P(p)는 입력 교체 연산자이다. (1)의 결과를 바탕으로 하여 먼저 1절에서는 교체된 입력과 정상 배열된 출력을 위한 이차원 CGF 알고리즘 I이 소개되고, 2절에서는 교체된 입력과 교체된 출력을 갖으나 global interconnection에 의해 연결되는 스테이지의 수를 반으로 감소시키는 이차원 CGF 알고리즘 Ⅱ가 소개된다.

1. 이차원 CGF 알고리즘 I

이차원 입력 $x(n_1, n_2)$에 대해 이차원 출력 $X(k_1, k_2)$를 발생시키는 $N \times N$ point 이차원 DFT는 (2)와 같이 나타낼 수 있다.

$$X(k_1,k_2)=\sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1}x(n_1,n_2)\cdot\exp(2\pi in_1k_1/N)\cdot\exp(-2\pi in_2k_2/N)$$
$$k_1,k_2=0,1,\cdots,N-1$$
$$i=sqrt(-1) \qquad (2)$$

(2)로부터 이차원 FFT 행렬 알고리즘을 일차원 linear array 아키텍쳐로 연산하기 위하여 입력 열벡터 $(x(n))^T$와 출력 열벡터 $(X(k))^T$를 다음과 같이 배열한다.

$$(x(n))^T=(x(n_1N+n_2))^T$$
$$=(x(0,0),x(0,1),\cdots,x(0,N-1),$$
$$x(1,0),\cdots,x(N-1,N-1))^T$$
$$(x(k))^T=(X(k_1N+k_2))^T$$
$$=(X(0,0),X(0,1),\cdots,X(0,N-1),$$
$$X(1,0),\cdots,X(N-1,N-1))^T$$

그러면, 이차원 DFT 행렬, $F_N{}^2$은 (1)의 일차원 DFT 행렬, F_N의 Kronecker product에 의해 (3)과 같이 표현될 수 있다.

$$F_{N^2}=F_N\otimes F_N \qquad (3)$$

(3)은 다음과 같이 증명될 수 있다. (2)에서 출력 $X(k_1,k_2)$를 발생시키기 위해 입력 $x(n_1,n_2)$에 곱해지는 twiddle factor는 $\exp(-2\pi in_1k_1/N)\cdot\exp(-2\pi in_2k_2/N)$이다. 이는 일반적으로 (3)에서 일차원으로 배열된 출력 $X(k_1N+k_2)$ (이차원시 $X(k_1,k_2)$)을 발생시키기 위하여 입력 $x(n_1N+n_2)$ (이차원시 $x(n_1,n_2)$)에 곱해지는 twiddle factor, F_N의 k_1N+k_2 행, n_1N+n_2열의 값과 동일하다.

이제 (1)의 결과식을 이용하기 위하여 (3)의 양변에 $(P(p)\otimes P(p))$를 곱하고, Appendix A의 보조정리 6을 적용한다. $F_NP(p)$에 대하여 (1)의 일차원 Cooley-Tukey FFT 행렬 알고리즘을 대입한 뒤, 계속해서 보조 정리 6을 반복 적용하면 (4)가 얻어진다.

$$F_{N^2}=(P(p)\otimes P(p))$$
$$=F_NP(p)\otimes F_NP(p)$$
$$=(A_tA_{t-1}\cdots A_1\otimes I_N)(I_N\otimes A_tA_{t-1}\cdots A_1)$$
$$=(A_t\otimes I_N)\cdots(A_1\otimes I_N)\cdot(I_N\otimes A_t)\cdots(I_N\otimes A_1) \qquad (4)$$

(4)로부터 이차원 CGF 알고리즘은 다음과 같이 얻어질 수 있다. $1\le q\le 2t$에 대해 $L=p^q$, $m=L/p$라 하고, 새로운 행렬 H_q를 이용한다.

i) $1\le q\le t$에 대해, $H_q=I_N\otimes A_q$라 하면

$$H_q=I_N\otimes(I_{N/L}\otimes F_p\otimes I_m)(I_{N/L}\otimes D_q)$$
$$=(I_{N^2/L}\otimes F_p\otimes I_m)(I_{N^2/L}\otimes D_q)$$

ii) $t+1\le q\le 2t$에 대해, $q'=q-t$, $L'=p^{q'}$, $m'=L'/p$라 하고 $H_q=A_q'\otimes I_N$라 하면,

$$H_q=(I_{N/L'}\otimes F_p\otimes I_{m'})(I_{N/L'}\otimes D_{q'})\otimes I_N$$
$$=(I_{N^2/NL}\otimes F_p\otimes I_{m'}\otimes I_N)(I_{N^2/NL'}\otimes D_{q'}\otimes I_N)$$

이제 $t+1\le q\le 2t$에 대해 $D_q=D_q'\otimes I_N$이라 하면

$$H_q=(I_{N^2/L}\otimes F_p\otimes I_m)(I_{N^2/L}\otimes D_q)$$

그러면, i)과 ii)로부터 (5)가 얻어진다.

$$F_{N^2}(P(p)\otimes P(p))=H_{2t}H_{2t-1}\cdots H_1 \qquad (5)$$

(5)의 분해 행렬 H_q에서 $(I_{N^2/L}L\otimes F_p\otimes I_m)$은 차례로 보조정리 2, 1을 사용하여 다음과 같이 변형될 수 있다.

$$I_{N^2/L}\otimes F_p\otimes I_m$$
$$=\pi(p,N^2)^{(2t-q)^T}\pi(p,N^2)^{(2t-q)}(I_{N^2/L}\otimes F_p\otimes I_m)$$
$$\pi(p,N^2)^{(2t-q)^T}\pi(p,N^2)^{(2t-q)}$$
$$=\pi(p,N^2)^q(F_p\otimes I_{N^2/p})\pi(p,N^2)^{q^T}$$

위 결과를 이용하여 H_q를 다시 쓰면

$$H_q=\pi(p,N^2)^q(F_p\otimes I_{N^2/p})\pi(p,N^2)^{q^T}(I_{N^2/L}\otimes D_q)$$

이제, B_q와 C_{q-1}을 다음과 같이 놓는다.

$$B_q=(F_p\otimes I_{N^2/p})\pi(p,N^2)^T$$
$$C_{q-1}=\pi(p,N^2)^{(q-1)T}(I_{N^2/L}\otimes D_q)\pi(p,N^2)^{(q-1)}$$
$$=\pi(p,N^2)\pi(p,N^2)^{(2t-q)}(I_{N^2/L}\otimes D_q)\cdot\pi(p,N^2)^{(2t-q)^T}\pi(p,N^2)^T$$
$$=\pi(p,N^2)(D_q\otimes I_{N^2/L})\pi(p,N^2)^T \qquad \text{(보조 정리 4, 1)}$$

C_{q-1}은 대각행렬이고, B_q는 단지 radix p에 의해서만 구조가 달라지므로, radix가 결정될 경우 모든 스테이지는 동일한 구조로 이루어짐을 알 수 있다. 위 결과로부터, $V_q=B_qC_{q-1}$라 하면

$$V_q=(F_p\otimes I_{N^2/p})\pi(p,N^2)^T\pi(p,N^2)(D_q\otimes I_{N^2/L})\pi(p,N^2)^T$$
$$=(F_p\otimes I_{N^2/p})(D_q\otimes I_{N^2/L})\pi(p,N^2)^T$$

따라서 (6)의 이차원 CGF 알고리즘이 얻어진다.

$$F_{N^2}(P(p)\otimes P(p)) = H_{2t}\cdots H_1 = \pi(p,N^2)^{2t}\cdot V_{2t}\cdots V_1$$
$$= V_{2t}\cdots V_1$$
(단, $V_q = (F_p \otimes I_{N^2/p})(D_q \otimes I_{N^2/L})\pi(p,N^2)^T$
$L = p^q,\ m = L/p$

i) $1\le q\le t$에서
$$D_q = diag(I_{L/p}, \Delta, \cdots, \Delta^{p-1})$$
$$\Delta = diag(1, W_L, \cdots, W_L^{m-1})$$

ii) $t+1\le q\le 2t$에서
$$D_q = D_{q'} \otimes I_N,\ \ q' = q - t \qquad (6)$$

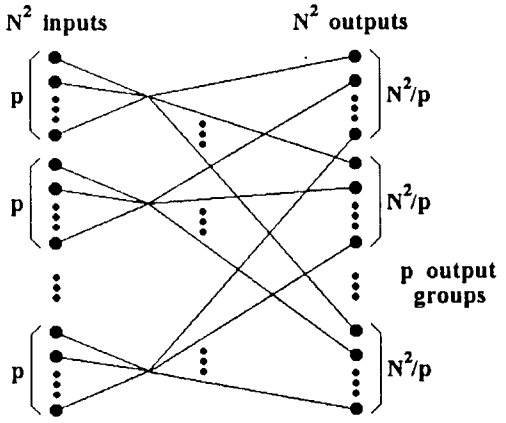


그림 1. 이차원 CGF 버터플라이 다이어그램 Ⅰ
Fig. 1. Two dimensional CGF Butterfly diagram Ⅰ.

그림 1에 (6)의 결과식을 이용한 교체된 입력과 정상 배열된 출력을 갖는 radix p, N×N point 이차원 CGF 버터플라이 다이어그램 Ⅰ이 보여진다. CGF 성질에 따라 모든 버터플라이 스테이지는 그림 1과 같은 동일한 구조를 갖는다. 이러한 스테이지간의 동일한 구조는 각 버터플라이 스테이지 간에 서로 다른 intercon nection 형태를 요구하는 기존의 Cooley - Tukey 알고리즘에 비해 VLSI 구현이 용이하며, 또한, 병렬처리 시스템의 구현에 있어 동일한 하드웨어를 반복해서 사용할 수 있는 장점이 있다.

2. 이차원 CGF 알고리즘 Ⅱ

이미 언급된 바와 같이 Ⅰ절에서 얻어진 CGF 알고리즘 I은 기존의 Cooley- Tukey 알고리즘에 비해 매우 규칙적인 장점이 있다. 그러나, 알고리즘 Ⅰ을 바탕으로 VLSI 구현시, 모든 스테이지의 출력은 global interconnection을 통해 다음 스테이지로 입력되므로 큰 칩 면적이 소모된다. 따라서, 본 절에서는 global interconnection에 의해 연결되는 스테이지 수를 반으로 감소시키는 CGF 알고리즘 II가 소개된다.

알고리즘 Ⅱ의 유도에 있어 필요로 되는 일차원 CGF 알고리즘[6] 은 (1)로부터 (7)과 같이 나타낼 수 있다.

$$F_N P(p) = V_t V_{t-1}\cdots V_1$$
단, $V_q = \pi(N/p, N)\cdot(I_{N/p}\otimes F_p)\cdot T_q$
$T_q = \pi(p,N)\cdot(D_q \otimes I_{N^2/L})\cdot\pi(p,N)^T$
$D_q = diag(I_m, \Delta_m, \cdots, \Delta_m^{p-1})$
$\Delta_m = diag(1, W_L, \cdots, W_L^{m-1})$
$L = p^q, m = L/p$ (7)

다시 Ⅰ절에서와 마찬가지로 (3)의 결과를 이용한다. (7)의 우변을 (3)에 대입한 뒤, 계속해서 Appendix A의 보조 정리 6을 반복 적용하면 t개의 분해행렬로 이루어진 (8)이 얻어진다.

$$F_{N^2}(P(p)\otimes P(p)) = F_N P(p)\otimes F_N P(p)$$
$$= (V_t V_{t-1}\cdots V_1)\otimes(V_t V_{t-1}\cdots V_1)$$
$$= (V_t\otimes V_t)(V_{t-1}\otimes V_{t-1})\cdots(V_1\otimes V_1) \qquad (8)$$

(8)에서 행렬 ($V_q\otimes V_q$)는 보조정리 6에 의해 다음과 같이 표현될 수 있다.

$$V_q\otimes V_q = \left(\pi(N/p,N)\cdot(I_{N/p}\otimes F_p)\cdot T_q\right)$$
$$\otimes\left(\pi(N/p,N)\cdot(I_{N/p}\otimes F_p)\cdot T_q\right)$$
$$= (\pi(N/p,N)\otimes\pi(N/p,N))$$
$$\cdot\left((I_{N/p}\otimes F_p)\cdot T_q\otimes(I_{N/p}\otimes F_p)\cdot T_q\right)$$
$$= (\pi(N/p,N)\otimes\pi(N/p,N))$$
$$\cdot(I_{N/p}\otimes F_p\otimes I_{N/p}\otimes F_p)\cdot(T_q\otimes T_q)$$
$$= (\pi(N/p,N)\otimes\pi(N/p,N))\cdot(I_{N/p}\otimes F_p\otimes I_N)$$
$$\cdot(I_{N^2/p}\otimes F_p)\cdot\left(T_q\otimes T_q\right)$$

위의 결과는 아키텍쳐 관점에서 다음과 같이 설명될 수 있다. 먼저 ($V_q\otimes V_q$)에 있어 교체 행렬 ($\pi(N/p,N)\otimes\pi(N/p,N)$)은 단지 우측 나머지 행렬에 의해 발생된 출력이 다음 스테이지로 입력되는 순서만을 결정하며, 대각 행렬 ($T_q\otimes T_q$)는 입력 데이타에 대한 곱하기 연산만을 수행한다. 그리고, 행렬곱 ($I_{N/p}\otimes F_p\otimes I_N$) · ($I_{N^2/p}\otimes F_p$)는 각 행렬에 대응되는 두 개의 버터플라이 스테이지로 구현될 수 있다. 이때, 두번째 행렬 ($I_{N^2/p}\otimes F_p$)은 연속된 p개 입력에 대해

연속된 p개 출력을 발생시키는 F_p를 대각선 방향에 가지므로 최소한의 interconnection 길이만으로 구현될 수 있다. 그러나, 첫번째 행렬 $(I_{N/p}\otimes F_p\otimes I_N)$은 서로 N 데이타씩 떨어져 있는 p개의 입력에 대해 역시 N 데이타씩 떨어진 p개 출력을 발생시키는 $(F_p\otimes I_N)$을 대각선 방향에 갖는다. 이러한 행렬 구조는 VLSI 구현시 FFT의 길이, N이 증가함에 따라 interconnection의 길이를 증가시키며, 따라서 큰 칩면적을 요구하게 된다. 이러한 단점은 $(V_q\otimes V_q)$의 행과 열을 적절히 교체함으로써 제거될 수 있으며, 결과적으로 (9)의 이차원 CGF 알고리즘 Ⅱ이 얻어진다. 증명 과정은 Appendix B에 나타내었다.

$$
\begin{aligned}
F_{N^2} &= \left(I_{N/p}\otimes\pi(p,N)^T\otimes I_p\right)\cdot G_t G_{t-1}\cdots G_1 \\
&\quad\cdot\left(I_{N/p}\otimes\pi(p,N)\otimes I_p\right)\cdot\left(P(p)\otimes P(p)\right) \\
\text{단, } G_q &= 0(p,N)\cdot\left(I_{N^2/p}\otimes F_p\right)T2_q\cdot\left(I_{N^2/p^2}\otimes\pi(p,p^2)\right) \\
&\quad\cdot\left(I_{N^2/p}\otimes F_p\right)T1_q \\
O(p,N) &= \left(I_{N/p}\otimes\pi(p,N)\otimes I_p\right)\cdot\left(\pi(p,N)^T\otimes\pi(p,N)^T\right) \\
&\quad\cdot\left(I_{N/p}\otimes\pi(p,N)^T\otimes I_p\right)\cdot\left(I_{N^2/p^2}\otimes\pi(p,p^2)\right) \\
T2_q &= \left(I_{N^2/p^2}\otimes\pi(p,p^2)\right)\cdot\left(I_{N/p}\otimes\pi(p,N)\otimes I_p\right) \\
&\quad\cdot\left(T_q\otimes I_N\right)\cdot\left(I_{N/p}\otimes\pi(p,N)^T\otimes I_p\right)\cdot\left(I_{N^2/p^2}\otimes\pi(p,p^2)\right) \\
T1_q &= \left(I_{N/p}\otimes\pi(p,N)\otimes I_p\right)\cdot\left(I_N\otimes T_q\right)\cdot\left(I_{N/p}\otimes\pi(p,N)^T\otimes I_p\right)
\end{aligned}
\tag{9}
$$

(9)는 아키텍처의 관점에서 다음과 같이 설명될 수 있다. 먼저, 행렬 $T2_q$, $T1_q$는 대각 행렬 $(T_q\otimes I_N)$, $(I_N\otimes T_q)$의 좌측에 곱해진 교체행렬의 transpose를 각각 우측에 곱해주었으므로 역시 대각행렬이다. 따라서 데이타 교체는 요구되지 않는다. 행렬 $(I_{N^2/p}\otimes F_p)$는 연속적인 p개의 입력을 받아들여 입력 위치와 동일한 출력 위치에 연속적인 p개의 출력을 발생시킨다. 따라서 단지 연속된 p개 데이타에 대한 interconnection만을 필요로 한다. 첫번째 $(I_{N^2/p}\otimes F_p)$에 의해 발생된 N^2개 출력은 교체행렬 $\pi(p,p^2)$을 대각성분으로 갖는 행렬 $(I_{N^2/p^2}\otimes(p,p^2))$에 의해 단지 연속된 p^2개의 데이타 사이에서만 교체되어 다음 버터플라이 연산을 위해 입력된다. 두번째 버터플라이 연산 역시 연속된 p개 데이타 내에서만 interconnection이 필요로 된다. 마지막으로 교체행렬 O(p, N)은 G_q에 의해 발생된 출력이 global interconnection에 의해 교체되어 다음 분해행렬 G_{q+1}의 입력이 되도록 한다. 분해 행렬 G_t에 의해 발생된 최종 출력은 교체행렬 $(I_{N/p}\otimes\pi(p,N)^T I_p)$에 의해 정상배열된 출력으로 교체된다. 즉, 분해행렬 G_q와 G_{q+1} 사이에는 global interconnection이 요구되나, G_q내에서는 단지 p^2개의 데이타 사이에서만 interconnection이 이루어진다. 따라서 전체 시스템에서 요구되는 global interconnection의 양은 단지 t개의 각 G_q 스테이지마다 global interconnection을 필요로 하므로 2t개의 스테이지간에 global interconnection을 필요로 하는 알고리즘 I에 비해 반으로 감소하며, 이는 칩면적을 크게 감소시킨다. 그림 2에 (9)의 이차원 CGF 알고리즘 Ⅱ을 바탕으로 하는 radix p, N×N point 버터플라이 다이어그램이 보여진다. 그림 2에서 G_q의 출력이 다음 스테이지로 입력되는 것을 보여주는 교체형태는 표기의 복잡함으로 인해 (9)의 출력 교체행렬 O(p,N)으로 나타내었다.

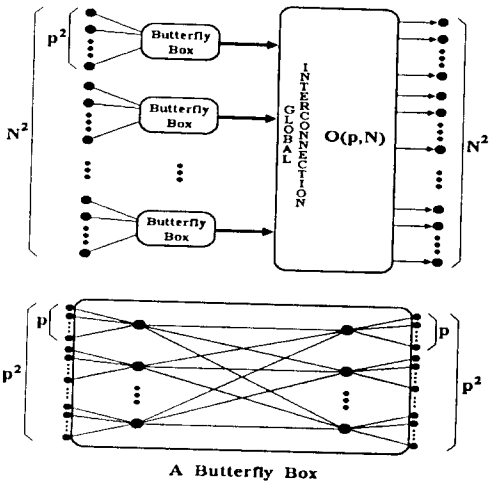


그림 2. 이차원 CGF 버터플라이 다이어그램 Ⅱ.
Fig. 2. Two dimensinal CGF Butterfly diagram Ⅱ.

Ⅲ. 작은 radix PE를 이용한 큰 radix PE의 구현 알고리즘

본 절에서는 이차원 FFT 시스템을 큰 radix PE로 용이하게 구현하기 위한 알고리즘이 소개된다. [6] 에서 일차원 FFT 시스템에 대하여 작은 radix PE로 큰 radix PE를 구현하는 방법이 소개되었다. 이 방법은 간단한 PE의 중복사용으로 용이한 시스템의 성능 향상, 계산속도와 하드웨어 양의 용이한 조절 이외에도 병렬 시스템의 구현시 칩면적을 증가시키는 global interconnection을 더욱 감소시켜 주는 효과를 제공할 수 있다. 먼저, Ⅱ.1 절 (6)의 알고리

즘에 대해, 그리고 Ⅱ.2 절 (9)의 알고리즘에 대해 위 방법의 적용이 각각 논의된다.

1. 작은 radix PE를 이용한 큰 radix PE의 구현 알고리즘 Ⅰ

아래에 (6)의 알고리즘을 구성하는 분해 행렬이 다시 보여진다.

$$V_q = \left(F_p \otimes I_{N^2/p}\right)\left(D_q \otimes I_{N^2/L}\right)\pi\left(p, N^2\right)^T$$

$$C'_q = \left(D_q \otimes I_{N^2/L}\right)\pi\left(p, N^2\right)^T$$

위에서 라 하면

$$V_q = \left(F_q \otimes I_{N^2/p}\right)\cdot C'_q$$

이제, 위 식에 보조 정리 1, 4, 2, 3을 차례로 적용하면

$$\begin{aligned} V_q &= \pi\left(p, N^2\right)^{(2t-1)}\cdot\left(I_{N^2/p}\otimes F_p\right)\cdot\pi\left(p, N^2\right)^{(2t-1)T}\cdot C'_q \\ &= \pi\left(p, N^2\right)\cdot\left(I_{N^2/p}\otimes F_p\right)\cdot\pi\left(p, N^2\right)\cdot C'_q \\ &= \pi\left(N^2/p, N^2\right)\cdot\left(I_{N^2/p}\otimes F_p\right)\cdot\pi\left(p, N^2\right)\cdot C'_q \end{aligned}$$

다시, $R_q = \pi(N^2/p, N^2)\cdot(I_{N^2/p}\otimes F_p)$, $T_q = \pi(p, N^2)\cdot C'_q$ 라 한다. T_q는 대각 행렬의 좌우측에 각각 $\pi(p, N^2)$과 $\pi(p, N^2)^T$를 곱해주었으므로 역시 대각 행렬이다. 그러면 V_q는 R_q와 T_q의 곱으로 (10)과 같이 표현된다.

$$V_q = R_q T_q = \pi\left(N^2/p, N^2\right)\cdot\left(I_{N^2/p}\otimes F_p\right)\cdot T_q \qquad (10)$$

(10)에서 V_q는 좌측에 $\pi(N^2/p, N^2)$, 우측에 대각 행렬 T_q, 그리고 가운데에 대각 방향으로 p point DFT 행렬 F_p를 갖는 블록 대각 행렬로 분해되었음을 알 수 있다. 위의 결과로부터, N point DFT 행렬 F_N이 p point DFT 행렬 F_p로 분해되는 과정을 반복적으로 적용함으로써 다시 F_p가 더 작은 radix s (단, $p=s^{t'}$) DFT 행렬 F_s로 분해될 수 있으며, 이는 요구되는 작은 radix DFT 행렬로 분해될 때까지 계속 적용될 수 있다. 따라서 큰 radix PE를 작은 radix PE들로 용이하게 구현할 수 있다. 그림 3에 작은 radix PE로 큰 radix PE를 구현하는 (10)의 원리를 행렬구조로 나타내었다.

2. 작은 radix PE를 이용한 큰 radix PE의 구현 알고리즘 Ⅱ

이차원 CGF 알고리즘 II에 있어 작은 radix PE를 이용하여 큰 radix PE를 구현하는 방법은 Appendix B의 (B.1)으로부터 유도될 수 있다. 편의상 Appendix B의 (B.1)을 다시 나타내었다.

$$\begin{aligned} G_q = &\left(I_{N/P}\otimes\pi(p,N)\otimes I_p\right) \\ &\cdot\left(\pi(N/p,N)\cdot\left(I_{N/p}\otimes F_p\right)\cdot T_q\otimes\pi(N/p,N)\right. \\ &\left.\cdot\left(I_{N/p}\otimes F_p\right)\cdot T_q\right)\cdot\left(I_{N/P}\otimes\pi(p,N)^T\otimes I_p\right) \end{aligned} \qquad (B.1)$$

이제 위 식의 우변에 대해 보조정리 6을 반복해서 사용한 뒤, 행렬 $(T_q\otimes T_q)$의 좌측에 단위행렬 $(I_{N^2}\otimes\pi(p,N)^T\otimes I_p)(I_{N^2}\otimes\pi(p,N)\otimes I_p)$를 곱하면 (11)이 얻어진다.

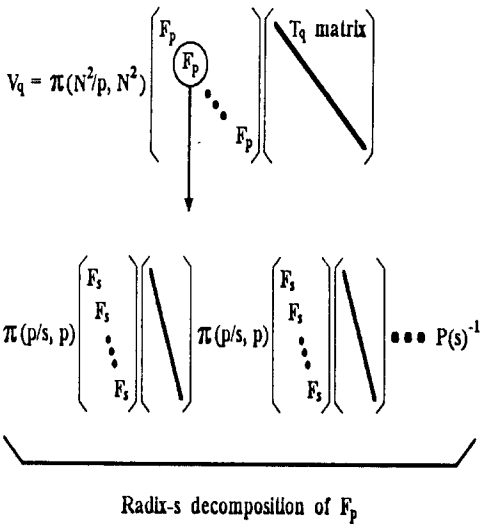


그림 3. 작은 radix PE로 큰 radix PE 구현을 위한 행렬 구조 표현 Ⅰ

Fig. 3. The matrix structure Ⅰ to construct a High radix PE by Low radix PE's.

$$\begin{aligned} G_q = &\left(I_{N/P}\otimes\pi(p,N)\otimes I_p\right)\cdot\left(\pi(N/p,N)\otimes\pi(N/p,N)\right) \\ &\cdot\left(I_{N/p}\otimes F_p\otimes I_{N/p}\otimes F_p\right)\cdot\left(T_q\otimes T_q\right)\cdot\left(I_{N/P}\otimes\pi(p,N)^T\otimes I_p\right) \\ = &\left(I_{N/P}\otimes\pi(p,N)\otimes I_p\right)\cdot\left(\pi(N/p,N)\otimes\pi(N/p,N)\right) \\ &\cdot\left(I_{N/p}\otimes F_p\otimes I_{N/p}\otimes F_p\right)\cdot\left(I_{N/P}\otimes\pi(p,N)^T\otimes I_p\right) \\ &\cdot\left(I_{N/P}\otimes\pi(p,N)\otimes I_p\right)\cdot\left(T_q\otimes T_q\right)\cdot\left(I_{N/P}\otimes\pi(p,N)^T\otimes I_p\right) \end{aligned} \qquad (11)$$

(11)에서 행렬 $(I_{N/p}\otimes\pi(p,N)\cdot(T_q\otimes T_q)\cdot(I_{N/p}\otimes\pi(p,N)^T\otimes I_p)$는 대각행렬이고, $(I_{N/p}\otimes\pi(p,N)\otimes I_p)\cdot(\pi(N/p,N)\otimes\pi(N/p,N))$는 버터플라이 연산 결과 발생되는 출력에 대한 교체연산을 행하는 행렬이다. 이제, 버터플라이 연산을 행하는 나머지 행렬을 Cq

라 하면, C_q는 다음과 같이 변형될 수 있다.

$$\begin{aligned} C_q &= \left(I_{N/P}\otimes F_p\otimes I_{N/p}\otimes F_p\right)\cdot\left(I_{N/P}\otimes\pi(p,N)^T\otimes I_p\right)\\ &= I_{N/p}\otimes\left(F_p\otimes I_{N/p}\right)\cdot\pi(p,N)^T\otimes F_p \qquad \text{(보조정리 6)}\\ &= I_{N/p}\otimes\pi(p,N)^T\cdot\left(I_{N/P}\otimes F_p\right)\otimes F_p \qquad \text{(보조정리 1)}\\ &= \left(I_{N/p}\otimes\pi(p,N)^T\right)\cdot\left(I_{N^2/P^2}\otimes F_p\right)\otimes F_p \qquad \text{(보조정리 6)}\\ &= \left(I_{N/p}\otimes\pi(p,N)^T\otimes I_p\right)\cdot\left(I_{N^2/P^2}\otimes F_p\otimes F_p\right) \text{(보조정리 6)} \end{aligned}$$

위 결과를 (11)에 대입하면, 작은 radix PE로 큰 radix PE를 구현하기 위한 알고리즘 Ⅱ이 (12)와 같이 얻어진다.

$$\begin{aligned} G_q = &\left(I_{N/P}\otimes\pi(p,N)\otimes I_p\right)\cdot\left(\pi(N/p,N)\otimes\pi(N/p,N)\right)\\ &\cdot\left(I_{N/P}\otimes\pi(p,N)^T\otimes I_p\right)\cdot\left(I_{N^2/P^2}\otimes F_p\otimes F_p\right)\\ &\cdot\left(I_{N/p}\otimes\pi(p,N)\otimes I_p\right)\cdot\left(T_q\otimes T_q\right)\left(I_{N/P}\otimes\pi(p,N)^T\otimes I_p\right) \end{aligned} \quad (12)$$

(12)에서 G_q는 두 p point DFT 행렬 F_p의 Kronecker product인 $(F_p\otimes F_p)$를 대각 방향으로 갖는 블록 대각 행렬 $(I_{N^2/p^2}\otimes F_p\otimes F_p)$와, 좌측의 교체 행렬 그리고 우측의 대각 행렬의 곱으로 나타난다. 즉, (12)는 (3)의 $(F_N\otimes F_N)$이 $(F_p\otimes F_p)$로 분해되었음을 보여준다. 따라서 (3)에서 (12)로의 분해 과정은 동일하게 $(F_p\otimes F_p)$에서 $(F_s\otimes F_s)$ (단, $p=s^{t'}$)로의 분해 과정에 반복적으로 이용될 수 있으며, 이는 단순히 Fp를 F_s로 분해하는 방식보다 규칙성과 PE의 복잡도, global interconnection의 감소라는 면에서

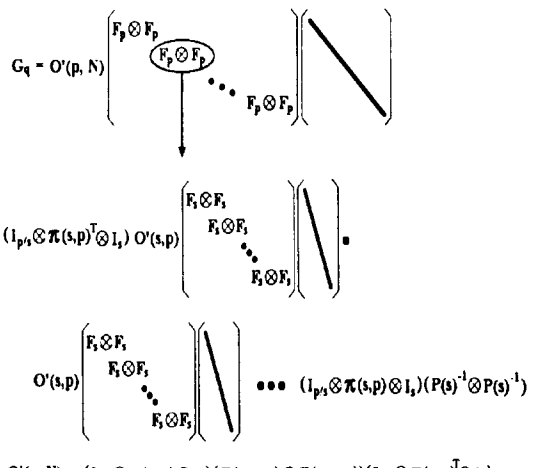


그림 4. 작은 radix PE로 큰 radix PE 구현을 위한 행렬 구조 표현 Ⅱ.

Fig. 4. The matrix structure Ⅱ to construct a High radix PE by Low radix PE's.

VLSI 구현에 있어 효율적이다. 또한 이 과정은 Fs보다 작은 길이의 DFT 행렬로의 분해로 연속적으로 적용될 수 있다. 그림 4에 작은 radix PE로 큰 radix PE를 구현하는 (12)의 원리를 행렬구조로 나타내었다.

Ⅳ. 이차원 CGF 아키텍처

본 절에서는 Ⅱ절에서 소개된 CGF 알고리즘 Ⅰ, Ⅱ를 바탕으로 하는 이차원 CGF 아키텍처 Ⅰ, Ⅱ가 소개된다. 아키텍처 Ⅰ과 Ⅱ는 모두 기존의 serial [1] 또는 parallel [3] 구조를 갖을 수 있다. Serial 구조의 경우 매우 적은 칩 면적을 필요로 하나, 고성능 시스템의 구현에는 부적당하다. 반면에 parallel 구조의 경우 고성능 시스템을 얻을 수 있으나 매우 큰 칩면적이 요구되는 단점이 있다. [6] 에서는 radix p 1차원 FFT에 대해 직 병렬 정도가 조절 용이한 아키텍처를 구현하기 위하여 모든 버터플라이 스테이지에 존재하는 p배의 대칭성을 이용하였다. 본 절에서도 이러한 p배의 대칭성을 확장, 적용하여 역시 직 병렬 정도가 조절 용이한 이차원 FFT를 위한 아키텍처 Ⅰ, Ⅱ가 소개된다. 또한, 이차원 FFT 연산에 있어 요구되는 모든 스테이지를 하드웨어로 구현하는 대신에 단지 몇개의 스테이지만을 구현한 뒤, 이를 반복 사용하는 recursive 구조의 아키텍처가 소개된다. 먼저, 1절에서는 이차원 CGF 알고리즘 Ⅰ을 바탕으로 하는 아키텍처 Ⅰ이 소개되고, 2절에서는 이차원 CGF 알고리즘 Ⅱ를 이용하는 아키텍처 Ⅱ가 소개된다.

1. 이차원 CGF 아키텍처 Ⅰ

그림 1에서 알 수 있듯이 radix p, N×N point 이차원 FFT를 위한 모든 버터플라이 스테이지의 출력은 각각 연속적인 N^2/p개의 데이타를 갖는 p개의 출력 그룹으로 나뉘어진다. 또한 각 그룹내의 데이타는 모두 동일한 교체형태를 갖는다. 즉, [6] 의 일차원 FFT 아키텍처의 구현에 이용된 p배의 대칭성이 알고리즘 Ⅰ을 바탕으로 한 모든 이차원 FFT 버터플라이 스테이지 Ⅰ에도 존재하며, 결과적으로 아키텍처 Ⅰ은 각 스테이지당 p개의 PE를 사용함으로써 용이하게 구현될 수 있다. 따라서 radix p 선택에 따라 FFT 연산의 직 병렬 정도, 또한 계산속도와 하드웨어 양의 조절을 용이하게 행할 수 있다. [6] 그림 5에 전체적인 아키텍처 Ⅰ의 구조가 보여진다. 아키텍처는 모두 동일한 구조를 갖는 $\log_p N^2$개의 스테이지로 구성되며, 입력측에는 입력 교체를 위한 데이타

shuffler가 있다.

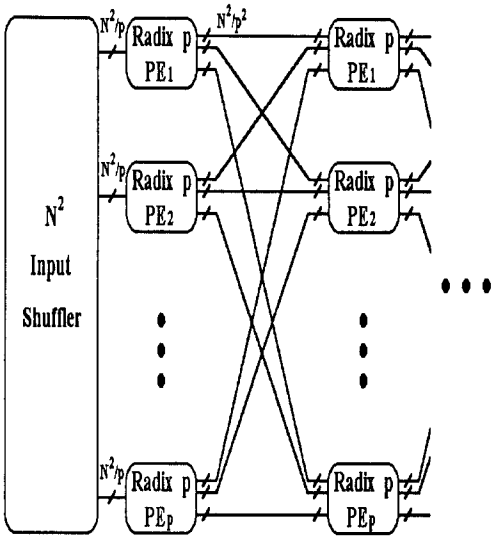


그림 5. 이차원 CGF 아키텍처 I

Fig. 5. Two dimensional CGF architecture I.

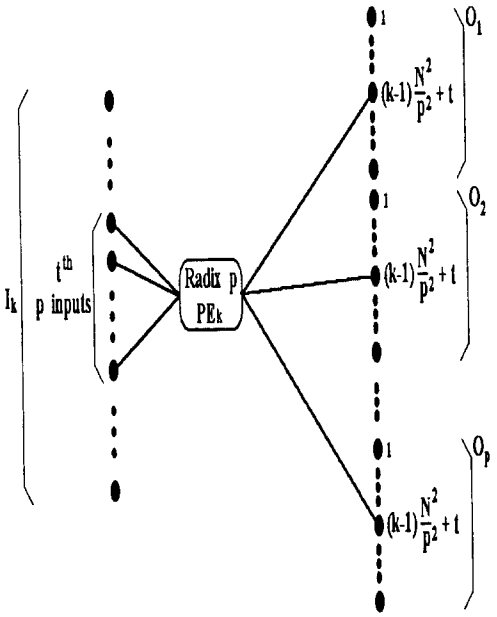


그림 6. 아키텍처 I에서 PE_k가 각 버터플라이를 처리하는 과정

Fig. 6. The procedure for PE_k to compute butterflies for architecture I.

이제, 각 스테이지의 PE가 입력을 처리하는 과정을 그림 6을 참고로 하여 설명한다. 먼저, 각 스테이지에는 p개의 PE가 존재하므로 N^2개의 입력과 출력은 각각 연속적인 N^2/p개의 데이타를 갖는 입력 그룹 $I_k(1\le k\le p)$와 출력 그룹 $O_j(1\le j\le p)$로 나뉘어진다. 이때, k번째 PE인 PE_k는 입력 그룹 I_k 내의 데이타를 p개씩 순차적으로 처리하게 되며, 그 과정은 다음과 같다. 일반적으로 k번째 입력 그룹 I_k 내의 $t(1\le t\le N^2/p^2)$번째 연속적인 p개 입력이 PE_k에 의해 버터플라이 처리된 결과 p개의 출력을 발생시키며, 이 p개의 출력은 모든 출력 그룹 $O_j(1\le j\le p)$내의 각 $(k-1)N^2/p^2+t$번째 위치로 하나씩 보내진다. PE_k가 k번째 N^2/p개의 입력을 처리하고 각 출력을 내보내는 과정은 전체 아키텍쳐를 구성하는 다른 모든 PE에 대해서도 동일하게 적용된다.

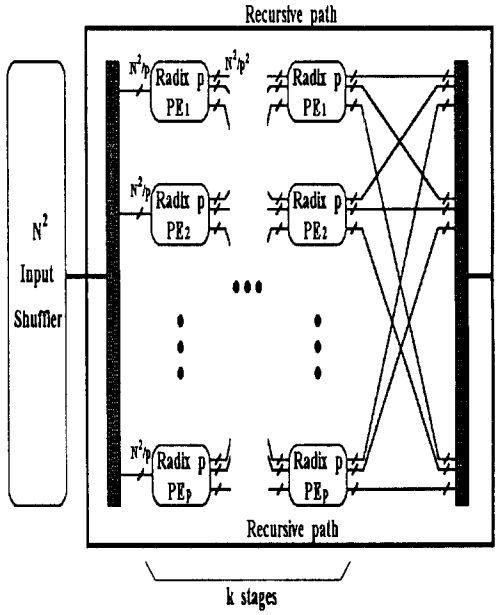


그림 7. Recursive pipeline 이차원 CGF 아키텍처 I

Fig. 7. Recursive pipeline two dimensional CGF architecture I.

그림 5의 아키텍처 I은 이차원 FFT 계산에 있어 모두 $\log_p N^2$개 스테이지의 구현을 요구한다. 그러나 모든 스테이지가 동일한 구조를 가지므로 이차원 FFT의 계산은 단지 k개의 스테이지만을 구현한 뒤 이 k개의 스테이지를 $(\log_p N^2)/k$번 반복해서 사용함으로써 행해질 수 있다. 이때 $k\neq(\log_p N^2)/2$인 경우에는 k개의 스테이지가 반복해서 이용됨에 따라 각각 다른 twiddle factor가 사용되어야 한다. 그런데, 식

(6)에서

$$V_{t+q} = V_q = \left(F_p \otimes I_{N^2/p}\right)\left(D_q \otimes I_{N^2/L}\right)\pi\left(p, N^2\right)^T,$$
$$L = p^q,\ 1 \le q \le t$$

이므로, 이차원 FFT 아키텍쳐 I의 처음 반과 이후의 반이 구조와 twiddle factor 면에서 완전히 동일함을 알 수 있다. 따라서, k = $(\log_p N^2)/2$개의 스테이지를 두번 사용하는 경우에는 동일한 twiddle factor가 이용될 수 있다는 장점이 있다. 그림 7에 k개의 스테이지만을 이용하는 아키텍쳐 I의 구조가 보여진다. 그림 7의 각 스테이지에서 각각의 PE가 입력을 처리하여 대응되는 출력을 발생시키는 과정은 그림 6에서 설명된 바와 동일하다.

2. 이차원 CGF 아키텍쳐 Ⅱ

아키텍쳐 Ⅱ의 구현에 있어서도 아키텍쳐 I의 경우와 같이 p배의 대칭성이 이용될 수 있다. 이미 언급한 바와 같이 출력단에 p배의 대칭성이 존재한다는 것은 N^2개의 출력 데이타가 각기 연속적인 N^2/p개의 데이타를 갖는 p개의 출력 그룹으로 나뉘어지며, 각 그룹내의 데이타 교체형태는 모두 동일하다는 것을 의미한다. (9)의 출력 교체행렬 O(p,N)에 의해 발생된 그림 2의 각 버터플라이 스테이지의 출력에도 이러한 p배의 대칭성이 존재한다는 것은 다음과 같이 설명될 수 있다. 편의상 출력 교체행렬 O(p,N)을 다시 나타내었다.

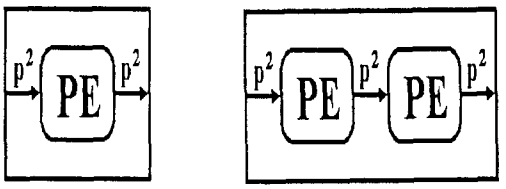
$$\begin{aligned} O(p,N) &= \left(I_{N/P} \otimes \pi(p,N) \otimes I_p\right)\cdot\left(\pi(p,N)^T \otimes \pi(p,N)^T\right) \\ &\quad \cdot\left(I_{N/P} \otimes \pi(p,N)^T \otimes I_p\right)\cdot\left(I_{N^2/P^2} \otimes \pi(p,p^2)\right) \\ &= \left(I_{N/p} \otimes \pi(p,N) \otimes I_p\right)\cdot\left(\pi(p,N)^T \otimes I_N\right)\cdot\left(I_N \otimes \pi(p,N)^T\right) \\ &\quad \cdot\left(I_{N/P} \otimes \pi(p,N)^T \otimes I_p\right)\cdot\left(I_{N^2/P^2} \otimes \pi(p,p^2)\right) \end{aligned}$$

설명의 편의를 위해 O(p,N)에 의한 교체는 두 단계로 고려하였다. 즉, 아래에 보여지는 i)의 교체행렬에 의해 첫번째 교체가 이루어지고, 단계 i)에 의해 교체된 데이타에 대해 ii)에 의한 두번째 교체가 수행된다.

$$\begin{aligned} i)\ &\left(I_{N/P} \otimes \pi(p,N)^T \otimes I_p\right)\cdot\left(I_{N^2/P^2} \otimes \pi(p,p^2)\right) \\ &= I_{N/P} \otimes \left((\pi(p,N)^T \otimes I_p)\cdot(I_{N/p} \otimes \pi(p,p^2)\right) \quad \text{(보조 정리 6)} \\ &= I_{N/P} \otimes \left((\pi(N/p,N)^T \otimes I_p)\cdot(I_{N/p} \otimes \pi(p,p^2)\right) \text{(보조 정리 3)} \\ ii)\ &\left(I_{N/P} \otimes \pi(p,N) \otimes I_p\right)\cdot\left(\pi(p,N)^T \otimes I_N\right)\cdot\left(I_N \otimes \pi(p,N)^T\right) \end{aligned}$$

i)과 ii)에서 각 교체행렬이 교체하는 데이타의 범위를 고려해보면, 단계 ii)의 첫번째 교체행렬 ($I_{N/p} \otimes \pi(p,N) \otimes I_p$)는 단지 pN ($\le N^2/p$)개의 연속된 데이타 블럭 내에서만 교체를 수행하므로, 결국 연속된 N^2/p 블럭 내에서 교체가 일어나, p배 대칭성에 영향을 주지 않는다. 단계 ii)의 두번째 교체행렬 ($\pi(N/p,N) \otimes I_N$)은 N^2개의 데이타를 N개씩 블럭 단위로 교체함으로써 p배의 대칭성을 제공한다.[6] 그리고, 단지 N개의 연속된 데이타 블럭 내에서만 교체를 수행하는 세번째 교체행렬 ($I_N \otimes \pi(p,N)^T$)은 두번째 교체행렬 ($\pi(N/p,N) \otimes I_N$)이 데이타를 N개씩 블럭 단위로 교체하기 때문에 역시 p배의 대칭성이 유지된다. 이제, 단계 i)에서의 교체 행렬을 고려하면 다음과 같다. 단계 i)의 결과식에서 교체행렬 ($\pi(N/p,N) \otimes I_p$) · ($I_{N/p} \otimes \pi(p,p^2)$)는 연속된 $pN(\le N^2/p)$개의 입력 데이타 블럭 내에서 교체를 수행하며 좌측 "$I_{N/p} \otimes$"에 의해서 모두 동일한 교체형태를 갖는 N/p ($\ge$p)개의 출력 그룹이 발생되는데, 이 출력 그룹의 수는 p의 멱승이 된다. 따라서 i), ii)에 의한 출력 교체행렬 O(p,N)은 출력단에 p배의 대칭성을 제공한다.

p배의 대칭성을 이용하는 아키텍쳐 Ⅱ의 전체적인 구조는 그림 5의 아키텍쳐 I과 유사하다. 차이점은 전체 스테이지의 갯수가 t개이며 한 스테이지의 출력 데이타가 교체되어 다음 스테이지로 입력되는 구조를 결정하는 출력 교체 행렬 O(p,N)이 다르다는 것이다. 이때, 그림 2에서 한개의 G_q에 대응되는 N^2/p^2 버터플라이 박스를 계산하기 위해 p배의 대칭성을 바탕으로 사용되는 p개의 하드웨어는 각각 그림 8(a), (b)의 구조를 갖을 수 있다.



(a) (b)

그림 8. 아키텍쳐 Ⅱ를 위한 PE의 구조
Fig. 8. The structure of PE's for architecture Ⅱ.

그림 8(a)에서는 단지 하나의 PE가 순차적으로 N^2/p^3의 버터플라이 박스를 처리한다. 이때 하나의 버터플라이 box 내에 있는 2p개의 버터플라이 연산은 다음과 같이 수행된다. 먼저, 그림 2에서 각 버터플라이 box 내의 왼쪽에 있는 p개의 버터플라이가 순차적으로 수행된 뒤, 그 결과 데이타를 이용하여

오른쪽에 있는 p개의 버터플라이가 순차적으로 수행된다. 결과적으로 요구되는 하드웨어 양은 최소가 되나, 시스템 성능은 낮다. 그림 8(b)는 두개의 동일한 PE가 직렬로 연결된 구조이다. 그림 8(b)의 구조가 사용될 때, 첫번째 PE는 그림 2의 N^2/p^3개 버터플라이 box 내 앞단의 N^2/p^3개 버터플라이를 순차적으로 처리하게 되며, 두번째 PE는 첫번째 PE에 의해 처리된 데이타를 이용하여 두번째 단에 있는 N^2/p^3개의 버터플라이를 순차적으로 처리하게 된다. 이러한 구조의 장점은 두번째 PE는 첫번째 PE가 가장 최초의 입력 데이타에 대해 단지 p개의 버터플라이를 처리할 때까지만 기다리고, 그다음부터 두개의 PE는 100% 동작할 수 있다는 것이다. 이는 그림 8(a) 구조에 비해 시스템의 latency를 크게 감소시킬 수 있다. 자세한 성능 평가 결과는 Ⅴ절에 나타내었다. 그림 9에 그림 8(b)의 구조를 바탕으로 radix 2, 4×4 point 이차원 FFT 연산을 위한 twiddle factor와 그 타이밍을 나타내었다.

아키텍쳐 Ⅱ역시 모든 스테이지가 동일한 구조를 가지므로 이차원 FFT의 계산에 있어 단지 k개의 스테이지만을 $(\log_p N)/k$번 반복해서 사용하는 아키텍쳐가 구현될 수 있다.

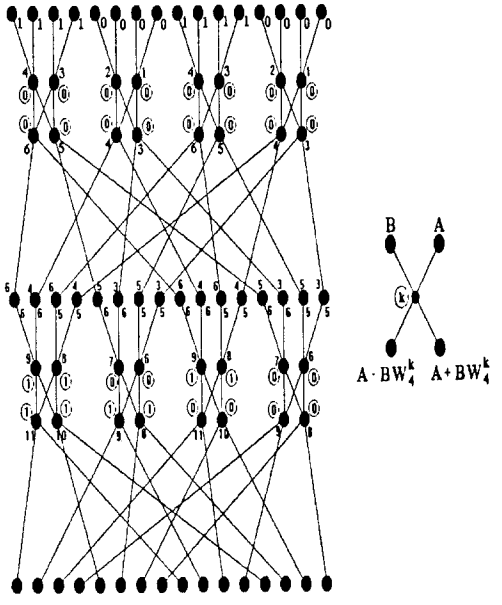


그림 9. 아키텍쳐 II에서 입 출력 타이밍 도 및 twiddle factors

Fig. 9. The input-output timing diagram and twiddle factors for architecture Ⅱ.

Ⅳ절에서 소개된 아키텍쳐는 위에 언급된 여러 특징 이외에도 [6] 에 소개된 계산 속도와 하드웨어 양의 조정 용이, pipeline 구조, fault tolerance 구현 용이, 시스템 클럭 속도 완화 등의 장점을 또한 가지고 있다. 마지막으로, 제안된 모든 아키텍쳐에 대해 Ⅲ절에서 소개된 작은 radix PE를 이용하여 큰 radix PE를 구현하는 방법이 또한 각각 적용될 수 있다. 즉, 작은 radix PE로 분해시 전체 아키텍쳐의 구조가 각각의 PE를 구현하기 위해 반복적으로 사용되어 질 수 있다. 이러한 특징은 복잡한 큰 radix PE의 VLSI 구현을 용이하게 하며, 큰 칩면적을 요구하는 global interconnection을 감소시킬 수 있는 장점이 있다.

Ⅴ. 성능 평가 비교

제안된 아키텍쳐의 성능을 수학적으로 평가하기 위해, 다음과 같은 VLSI 구현상의 일반적인 가정을 한다. ① 입력은 클럭 사이클 당 하나씩 교체된 상태로 입력된다 (raster scan order). ② 각 버터플라이 연산에 있어서 PE는 two level 파이프라인되어 있다고 가정된다. ③ PE의 latency는 d이다. 즉, pipeline 스테이지가 채워지지 않은 상태에서 첫번째 버터플라이를 계산하는 시간은 d이다. ④ PE의 출력 사이클 타임 (1/throughput), 즉 첫번째 버터플라이를 계산한 뒤, 다음 버터플라이를 파이프라인 방법으로 계산하는데 걸리는 시간은 r이다. ⑤ 모든 아키텍쳐에서 한 스테이지의 버터플라이가 모두 처리된 후 출력 데이타가 다음 스테이지로 입력된다고 가정한다.

위의 가정을 바탕으로 그림 5의 p배의 대칭성을 이용한 non-recursive 아키텍쳐 I에 대한 성능은 다음과 같이 계산된다. 먼저, 전체 FFT 시스템 latency의 계산은 첫번째 입력이 들어온 후 하나의 버터플라이 스테이지 전체가 계산되는 시간을 구한 후 전체 스테이지의 갯수를 곱하여 구할 수 있다. 우선 각 PE의 latency가 d이며 한 스테이지 당 한개의 PE가 N^2/p^2 버터플라이를 계산하므로, 나머지 N^2/p^2 - 1개의 버터플라이를 전부 계산하기 위해서는 $(N^2/p^2 - 1)r$이 더 필요하다. 그러므로, 전체 시스템의 latency, L_I은 (13)과 같이 나타낼 수 있다. 각 PE 내의 파이프라인 스테이지가 모두 작동하기 시작한 후 각 PE는 r마다 한개의 버터플라이를 계산하므로, 출력 사이클 타임 R_I과 throughput T_I는 (14)와 같이 나타낼 수 있다. 그리고, 한개의 스테이지 당 p개의 PE로 이루어지므로, 전체 시스템에 필요한 PE의 갯수, M_I은 (15)와 같다.

$$L_I = (\log_p N^2)(d+(N^2/P^2-1)r) \quad (13)$$

$$R_I = 1/T_I = (N^2 r)/p^2 \quad (14)$$

$$M_I = p \log_p N^2 \quad (15)$$

위의 결과를 이용하여, 단지 k개의 스테이지 만으로 이차원 FFT를 행하는 그림 7의 recursive pipeline 아키텍처 I에 대한 성능은 다음과 같이 계산된다. Recursive 경로에 의한 시간 지연을 무시할 때, 시스템의 latency, L_{Ir}은 L_I와 동일하게 (16)으로 얻어진다. 시스템의 출력 사이클 타임 R_{Ir}은 다음과 같이 얻어진다. 전체 시스템이 단지 k개의 스테이지만으로 이루어지므로 k개의 스테이지가 $(\log_p N^2)/k$번 이용되어야 한다. 처음 N_2개의 입력에 대해 k개의 스테이지가 $((\log_p N^2)/k - 1)$번 이용되고, throughput을 증가시키기 위해 마지막으로 recursive 경로를 통과한 데이타가 첫번째 스테이지에 의해 처리된 후 다음 N_2개의 입력 데이타를 처리한다고 하면, 이때 출력 사이클 타임 R_{Ir}는 (17)과 같다. 그리고 시스템에 필요한 PE의 갯수는 (18)과 같다.

$$L_{Ir} = (\log_p N^2)\cdot(d+(N^2/p^2-1)r) \quad (16)$$

$$\begin{aligned} R_{Ir} &= 1/T_I \\ &= k\cdot(d+(N^2/p^2-1)r)\cdot((\log_p N^2)/k-1)+(d+(N^2/p^2-1)r) \\ &= (d+(N^2/p^2-1)r)\cdot(\log_p N^2-k+1) \end{aligned} \quad (17)$$

$$M_{Ir} = p\cdot k \quad (18)$$

CGF 아키텍처 II에 대해서는 그림 8(b)의 구조를 고려하였다. 각 버터플라이 box 내의 local interconnection에 의한 delay는 없다고 가정한다. 그림 8(b) 구조 내의 첫번째 PE가 최초의 p개 버터플라이를 처리한 다음에, 두번째 PE는 첫번째 PE의 출력 데이타를 이용하여 delay없이 pipeline 방법으로 처리하므로, 한 스테이지의 latency는 하나의 PE가 순차적으로 N^2/p^2와 p개의 버터플라이를 처리하는 것과 동일하게 된다. 따라서, 아키텍처 I에서와 같은 방법으로 계산하면 전체 시스템의 latency, L_{II}는 (19)와 같이 얻어진다. 각 PE 내의 파이프라인 스테이지가 모두 작동하기 시작한 후 각 PE는 r마다 한 개의 버터플라이를 계산하므로, 출력 사이클 타임, R_{II}와 throughput, T_{II}는 (20)과 같다. 그리고, 한 개의 스테이지당 2p개의 PE로 이루어지고, 전체 스테이지가 $\log_p N$개이므로 전체 PE의 갯수, M_{II}는 (21)과 같다.

$$L_{II} = (\log_p N)\cdot(2d+(N^2/p^2+p-2)r) \quad (19)$$

$$R_{II} = 1/T_{II} = r(N^2/p^2+p) \quad (20)$$

$$M_{II} = 2p\cdot\log_p N \quad (21)$$

k개의 스테이지만으로 이차원 FFT를 행하는 recursive pipeline 아키텍처 II에서도 recursive 경로에 의한 시간 지연을 무시할 때, 시스템의 latency, L_{IIr}은 L_{II}와 동일하게 (22)로 얻어진다. 시스템의 출력 사이클 타임 R_{IIr}은 recursive pipeline 아키텍처 I에서와 같은 방법에 의해 (23)과 같이 얻어진다. 그리고 시스템에 필요한 PE의 갯수는 (24)와 같다.

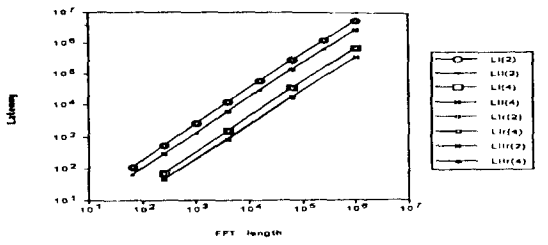
$$L_{II} = (\log_p N)\cdot(2d+(N^2/p^2+p-2)r) \quad (22)$$

$$\begin{aligned} R_{IIr} &= 1/T_{II} \\ &= (2d+(N^2/p^2+p-2)r)\cdot(\log_p N-k+1) \end{aligned} \quad (23)$$

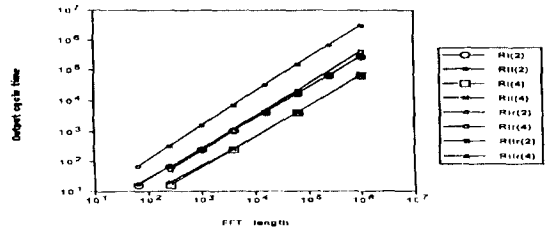
$$M_{IIr} = 2p\cdot k \quad (24)$$

III절의 작은 radix PE를 이용하여 큰 radix PE를 구현하는 방법을 적용했을 때의 성능 평가는 위의 성능 평가식을 참조하여 [6] 과 같이 구할 수 있으므로 여기서는 생략하였다. 그림 10(a), (b) 그리고 (c)에 (13)에서 (24)의 성능 평가 결과식을 바탕으로 d=2, r=1을 가정하여 각 아키텍처에 대한 성능 비교 결과가 보여진다. L, M, R은 각각 latency, 출력 사이클 타임, 전체 사용된 PE 갯수를 나타낸다. 아래첨자 I, II는 아키텍처 종류를, r은 recursive를 나타내며 이경우 총 스테이지의 절반의 하드웨어만이 사용되었다. 괄호 안은 사용된 radix를 나타낸다. 그림 10(a)에 두 아키텍처의 latency에 대한 비교 결과를 나타내었다. 아키텍처 II에서 latency는 모든 스테이지가 global interconnection으로만 연결되는 아키텍처 I의 약 1/2 정도를 나타내는 것을 알 수 있다. 또한 각 아키텍처에 있어 radix가 증가함에 따라 latency는 감소한다. Non-recursive와 recursive 아키텍처에서 latency는 각 radix와 아키텍처 종류에 따라 동일한 값을 갖는다. 반면에 그림 10(b)에서 알 수 있듯이 throughput은 아키텍처 I, II에서 거의 동일한 값을 나타내며, non-recursive 아키텍처가 recursive 경우에 비해 적은 출력 사이클 타임 (높은 throughput)을 나타낸다. 아키텍처 I, II의 PE 갯수는 그림 10(c)에서 알 수 있듯이 recursive 아키텍처는 non-recursive 경우에 비해 적은 PE 갯수를 필요로 한다. 결과적으로 radix가 증가함에 따라 latency는 감소하고 throughput은

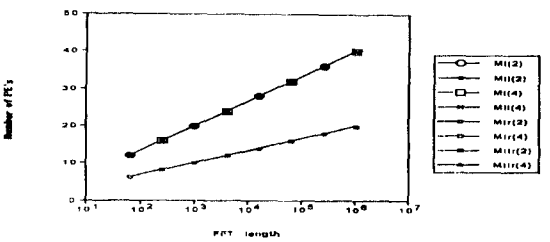
증가하며, 아키텍처 Ⅰ는 latency에 있어, 또한 interconnection 복잡도에 있어 아키텍처 Ⅰ보다 우수하고, non-recursive 아키텍처는 recursive 아키텍처보다 throughput이 높으나 더 많은 PE 갯수가 필요하다.



(a)



(b)



(c)

그림 10. (a) Latency 비교
(b) 출력 사이클 타임 비교
(c) 전체 PE 갯수 비교

Fig. 10. (a) The comparison of latencies.
(b) The comparison of output cycle times.
(c) The comparison of PE counts.

Ⅳ. 결 론

FFT 버터플라이 스테이지가 모두 동일한 구조를 가지므로, 같은 칩을 반복해서 사용하기 편리하며 VLSI 구현에 있어 매우 적합한, 이차원 constant geometry FFT 알고리즘 Ⅰ, 또한 이를 바탕으로 스테이지간의 interconnection 복잡도를 감소시킨 알고리즘 Ⅱ가 수학적으로 유도되었다. 각 이차원 constant geometry FFT 알고리즘은 동일한 radix로 이루어진 이차원 FFT 알고리즘을 구성하는 각 분해 행렬의 행과 열을 교체함으로써 얻어질수 있었다. 계산속도와 하드웨어 양의 조정 용이하고, global interconnection을 감소시키며 고성능 FFT 시스템의 용이한 구현을 위해 간단한 작은 radix PE로 큰 radix PE를 VLSI로 용이하게 구현할 수 있는 알고리즘Ⅰ, Ⅱ가 개발되었다. 개발된 이차원 constant geometry FFT 알고리즘 각각에 대해 시스템 성능과 하드웨어 양의 조절이 용이한 non-recursive 및 recursive pipelined 아키텍처 Ⅰ, Ⅱ가 제안되었다. 마지막으로, 제안된 각각의 아키텍처에 대한 성능이 평가, 비교되었다.

Appendix A

수학적 CGF의 개발을 위해 다음과 같이 정의한다.

정의 1. Kronecker product, ⊗는 다음과 같이 정의된다.

$$Amxn \otimes Bpxq = \begin{bmatrix} a_{00}B & a_{01}B & \cdots a_{0,n-1}B \\ \vdots & & \vdots \\ a_{m-1,0}B & a_{m-1,1}B & \cdots a_{m-1,n-1}B \end{bmatrix}_{mpxnq}$$

정의 2. diag(d) = diag(d_0, d_1, ⋯, d_{N-1})은 d_0, d_1, ⋯, d_{N-1}을 diagonal entry로 갖는 N X N diagonal matrix이다.

정의 3. FFT의 교체를 위해 N=pm 이라 하면,

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

$$(x(0),x(1),\cdots,x(N-1))\pi(p,N) = (x(0),x(p),x(2p),\cdots,x((m-1)p),x(1),x(p+1),\cdots)$$

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. : $\pi(p,N)^t = I_N$
보조 정리 5. : $(A\otimes B)\otimes C = A\otimes(B\otimes C)$
보조 정리 6. : $(A\otimes B)(C\otimes D) = AC\otimes BD$

Appendix B

G_q를 $(V_q\otimes V_q)$의 양쪽에 교체 연산자를 곱해 다음과 같이 두고, 보조정리 6을 반복해서 적용한다.

$$\begin{aligned}
G_q &= (I_{N/P}\otimes\pi(p,N)\otimes I_p)\cdot(V_q\otimes V_q)(I_{N/P}\otimes\pi(p,N)^T\otimes I_p)\\
&= (I_{N/p}\otimes\pi(p,N)\otimes I_p)\cdot(\pi(N/p,N)\cdot(I_{N/p}\otimes F_p)\\
&\quad\cdot T_q\otimes\pi(N/p,N)\cdot(I_{N/p}\otimes F_p)\cdot T_q)\\
&\quad\cdot(I_{N/P}\otimes\pi(p,N)^T\otimes I_p) \qquad (B.1)\\
&= (I_{N/P}\otimes\pi(p,N)\otimes I_p)\cdot(\pi(p,N)^T\otimes\pi(p,N)^T)\\
&\quad\cdot((I_{N/p}\otimes F_p)\cdot T_q\otimes I_N)\cdot(I_N\otimes(I_{N/p}\otimes F_p)\cdot T_q)\\
&\quad\cdot(I_{N/P}\otimes\pi(p,N)^T\otimes I_p) \qquad (\text{보조정리 } 3.6)\\
&= (I_{N/P}\otimes\pi(p,N)\otimes I_p)\cdot(\pi(p,N)^T\otimes\pi(p,N)^T)\\
&\quad\cdot(I_{N/p}\otimes F_p\otimes I_N)\cdot(T_q\otimes I_N)\\
&\quad\cdot(I_N\otimes I_{N/p}\otimes F_p)\cdot(I_N\otimes T_q)\cdot(I_{N/P}\otimes\pi(p,N)^T\otimes I_p)\\
&\qquad (\text{보조정리 } 6)
\end{aligned}$$

여기서 $(I_N\otimes T_q)$ $(I_{N/p}\otimes\pi(p,N)^T\otimes I_p)$를 대각행렬로 만들기 위해 $T1_q$를 다음과 같이 정의하고, 보조정리 6을 적용한다.

$$\begin{aligned}
T1_q &= (I_{N/P}\otimes\pi(p,N)\otimes I_p)\cdot(I_N\otimes T_q)(I_{N/P}\otimes\pi(p,N)^T\otimes I_p)\\
G_q &= (I_{N/p}\otimes\pi(p,N)\otimes I_p)\cdot(\pi(p,N)^T\otimes\pi(p,N)^T)\\
&\quad\cdot(I_{N/p}\otimes F_p\otimes I_N)\cdot(T_q\otimes I_N)\cdot(I_{N^2/p}\otimes F_p)\\
&\quad\cdot(I_{N/P}\otimes\pi(p,N)^T\otimes I_p)\cdot T1_q\\
&= (I_{N/P}\otimes\pi(p,N)\otimes I_p)\cdot(\pi(p,N)^T\otimes\pi(p,N)^T)\\
&\quad\cdot(I_{N/p}\otimes F_p\otimes I_N)\cdot(T_q\otimes I_N)\cdot(I_{N/P}\otimes\pi(p,N)^T\otimes I_p)\\
&\quad\cdot I_{N^2/p}\otimes I_pF_p)\cdot T1_q \qquad (\text{보조정리 } 6)\\
&= (I_{N/P}\otimes\pi(p,N)\otimes I_p)\cdot(\pi(p,N)^T\otimes\pi(p,N)^T)\\
&\quad\cdot(I_{N/p}\otimes F_p\otimes I_N)\cdot(T_q\otimes I_N)\\
&\quad\cdot(I_{N/p}\otimes\pi(p,N)^T\otimes I_p)\cdot(I_{N^2/p}\otimes F_p)\cdot T1_q \qquad (\text{보조정리 } 6)
\end{aligned}$$

다시 $T2_q'$를 $T1_q$와 같은 방법으로 다음과 같이 정의하면,

$$\begin{aligned}
T2_q' &= (I_{N/P}\otimes\pi(p,N)\otimes I_p)\cdot(T_q\otimes I_N)(I_{N/P}\otimes\pi(p,N)^T\otimes I_p)\\
G_q &= (I_{N/p}\otimes\pi(p,N)\otimes I_p)\cdot(\pi(p,N)^T\otimes\pi(p,N)^T)\\
&\quad\cdot(I_{N/p}\otimes F_p\otimes I_N)\cdot(I_{N/P}\otimes\pi(p,N)^T\otimes I_p)\\
&\quad\cdot T2_q'\cdot(I_{N^2/p}\otimes F_p)\cdot T1_q
\end{aligned}$$

위의 G_q에서 행렬곱 $(I_{N/p}\otimes F_p\otimes I_N)(I_{N/p}\otimes\pi(p,N)^T\otimes I_p)$은 다음과 같이 변형될 수 있다. 즉,

$$\begin{aligned}
&(I_{N/p}\otimes F_p\otimes I_N)\cdot(I_{N/P}\otimes\pi(p,N)^T\otimes I_p)\\
&= (I_{N/p}\otimes F_p\otimes I_{N/p}\otimes I_p)\cdot(I_{N/P}\otimes\pi(p,N)^T\otimes I_p) && (\text{보조정리 } 6)\\
&= (I_{N/p}\otimes F_p\otimes I_{N/p})\cdot(I_{N/P}\otimes\pi(p,N)^T)\otimes I_p && (\text{보조정리 } 6)\\
&= (I_{N/p}\otimes(F_p\otimes I_{N/p})\cdot\pi(p,N)^T)\otimes I_p && (\text{보조정리 } 6)\\
&= (I_{N/P}\otimes\pi(p,N)^T\cdot(I_{N/p}\otimes F_p))\otimes I_p && (\text{보조정리 } 2.1)\\
&= (I_{N/P}\otimes\pi(p,N)^t)\cdot(I_{N^2/p^2}\otimes F_p)\otimes I_p && (\text{보조정리 } 6)\\
&= (I_{N/P}\otimes\pi(p,N)^T\otimes I_p)\cdot(I_{N^2/p^2}\otimes F_p\otimes I_p) && (\text{보조정리 } 6)
\end{aligned}$$

$\pi(p,p^2)\cdot\pi(p,p^2)=I_{p^2}$임을 이용하여 행렬 $F_p\otimes I_p$의 우변에 곱한 뒤, 보조정리 6. 2. 6을 연속해서 적용한다.

$$\begin{aligned}
&(I_{N/p}\otimes\pi(p,N)^T\otimes I_p)\cdot(I_{N^2/p^2}\otimes(F_p\otimes I_p)\cdot\pi(p,p^2)\cdot\pi(p,p^2))\\
&= (I_{N/p}\otimes\pi(p,N)^T\otimes I_p)\\
&\quad\cdot(I_{N^2/p^2}\otimes(F_p\otimes I_p)\cdot\pi(p,p^2))\cdot(I_{N^2/p^2}\otimes\pi(p,p^2)) && (\text{보조정리 } 6)\\
&= (I_{N/p}\otimes\pi(p,N)^T\otimes I_p)\\
&\quad\cdot(I_{N^2/p^2}\otimes\pi(p,p^2)\cdot(I_p\otimes F_p))\cdot(I_{N^2/p^2}\otimes\pi(p,p^2)) && (\text{보조정리 } 2.1)\\
&= (I_{N/p}\otimes\pi(p,N)^T\otimes I_p)\\
&\quad\cdot(I_{N^2/p^2}\otimes\pi(p,p^2)\cdot(I_{N^2/p}\otimes F_p))\cdot(I_{N^2/p^2}\otimes\pi(p,p^2)) && (\text{보조정리 } 6)
\end{aligned}$$

위의 결과를 이용하여 G_q를 다시 쓰면

$$\begin{aligned}
G_q &= (I_{N/p}\otimes\pi(p,N)\otimes I_p)\cdot(\pi(p,N)^T\otimes\pi(p,N)^T)\\
&\quad\cdot(I_{N/p}\otimes\pi(p,N)^T\otimes I_p)\\
&\quad\cdot(I_{N^2/p^2}\otimes\pi(p,p^2))\cdot(I_{N^2/p}\otimes F_p)\cdot(I_{N^2/p^2}\otimes\pi(p,p^2))\\
&\quad\cdot T2_q'\cdot(I_{N^2/p}\otimes F_p)\cdot T1_q
\end{aligned}$$

마지막으로, $T2_q$와 $O(p,N)$을 각각

$$\begin{aligned}
T2_q &= (I_{N^2/p^2}\otimes\pi(p,p^2))\cdot T2_q'(I_{N^2/p^2}\otimes\pi(p,p^2))\\
O(p,N) &= (I_{N/p}\otimes\pi(p,N)\otimes I_p)\cdot(\pi(p,N)^T\otimes\pi(p,N)^T)\\
&\quad\cdot(I_{N/p}\otimes\pi(p,N)^T\otimes I_p)\cdot(I_{N^2/p^2}\otimes\pi(p,p^2))
\end{aligned}$$

라 하면,

$$G_q = O(p,N)\cdot(I_{N^2/p}\otimes F_p)\cdot T2_q\cdot(I_{N^2/p}\otimes\pi(p,p^2))\cdot(I_{N^2/p}\otimes F_p)\cdot T1_q$$

증명 끝.

參 考 文 獻

[1] W. Liu, T. Hughes, and W. T. Krakow, " The design of a vector - radix 2D FFT chip," *IEEE Int.* Symposium on Comp. Arithmetic, pp. 231-236, 1985.

[2] T. Willey, R. Chapman, H. Yoho, T. S. Durrani, and D. Preis, "Systolic implementation for deconvolution, DFT and FFT," *IEEE Procedings*, vol. 132 pt. F, No. 6, Oct. 1985.

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

[4] 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.

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

[6] 유 재희, "계산 속도와 하드웨어 양이 조절 용이한 FFT Array Processor 시스 템," 대한전자공학회 논문지, 제 30권, 제 3호, 1993.

[7] W. A. Perera, "Architecture for multiplierless Fast Fourier Transform hardware implementation in VLSI," IEEE Trans. on Acoustics, Speech, and Signal Processing, vol. ASSP-35, No. 12, Dec. 1987.

[8] R. W. Linderman, C. G. Shephard, K. Taylor, P.W. Coutee, P. C. Rossbach, J. M. Collins, and R. S. Hauser, "A 70-MHz 1.2- m CMOS 16-point DFT processor," *IEEE J. of Solid State Circuits*, vol. 23, No. 2, Apr. 1988.

著 者 紹 介

柳 在 熙(正會員)

1963年 3月 3日生. 1985年 2月 서울대학교 전자공학과 공학사. 1990年 2月 Cornell 대학교 전기공학과 공학박사(SRAM, BIC-MOS 회로, FFT processor). 1990年 3月 ~ 1991年 3月 Texas Instruments, Dallas VDL Lab. 연구원 (64MEG DRAM설계). 1991年 3月 현재 홍익대학교 전자공학과 조교수. 주관심 분야는 Image, Speech signal processing VLSI 아키텍처 및 시스템 설계, DRAM, SRAM 회로설계 등임.

郭 珍 錫(准會員)

1967年 10月 27日生, 1992年 2月 홍익대학교 전자공학과 공학사. 1994年 2月 홍익대학교 전자공학과 공학석사(VLSI 알고리즘 및 아키텍처). 1994年 2月 한국전자통신연구소(ETRI) 연구원. 주관심 분야는 DSP 알고리즘 및 아키텍처, 영상처리 등임.