효율적인 실시간 스펙트럼 연산을 위한 Pruned Generalized Sliding FFT 기법

하창읍*, 김완진*, 도대원**, 이동훈**, 김형남*
*부산대학교 전자전기공학과, **국방과학연구소

A Pruned Generalized Sliding FFT Method for Efficient Real-Time Spectrum Computation

Change-Eup Ha*, Wan-Jin Kim*, Dae-Won Do**, Dong-Hoon Lee**,
and Hyoung-Nam Kim*
*School of Electrical Engineering, Pusan National University
**Agency for Defense Development
hnkim@pusan.ac.kr

Abstract: 레이더나 소나와 같은 시스템에서는 수신된 신호로부터 다양한 정보를 획득하기 위해 실시간으로 수신 신호의 주파수 스펙트럼을 연산해야 할 필요성이 있다. 이를 위해 일반적으로 고속 퓨리에 변환 (fast Fourier transform, FFT) 알고리즘들이 사용되고 있으나, 정보 갱신 주기가 짧거나 일부 주파수 대역만이 요구되는 경우에는 연산량 측면에서 효율성이 감소하는 문제가 발생한다. 짧은 갱신주기 문제와 부분 스펙트럼 이용문제를 해결하기 위해 generalized sliding FFT (GSFFT)와 같은 sliding 기법과 transform decomposition (TD)과 같은 pruning 기법이 각각 제안되었으나, 두 가지 문제가 동시에 발생하는 경우에는 각각의 방법이 최고의 연산 효율성을 보장하지는 않는다. 이러한 문제를 해결하기 위해 본 논문에서는 기존의 GSFFT 와 TD 알고리즘의 장점을 결합시킨 Pruned GSFFT (PGSFFT)를 제안하였으며, 모의 실험을 통해 제안된 알고리즘이 기존의 sliding FFT 알고리즘이나 pruning 알고리즘들에 비해 적은 연산량을 가짐을 보인다.

Keywords: GSFFT, transform decomposition, pruning

Ⅰ. 서론

레이더나 소나와 같은 시스템에서는 목표물을 탐지 및 추적하기 위해 수신 신호로부터 방위정보와 거리, 도플러 천이와 같은 정보를 획득해야 한다. 수신된 신호로부터 정보를 추출하기 위해서는 시간 영역 또는 주파수 영역에서 신호를 처리해야 하나, 채널과 잡음 같은 왜곡요소로 인해 변형된 신호에서 정보를 추출하는 것이 어렵기 때문에 주파수 영역에서의 신호처리가 선호된다. 주파수 영역 신호처리를 위해서는 신호의 주파수 스펙트럼을 연산해야 하는데, 이를 위해 일반적으로 FFT (fast Fourier transform) 가 사용되고 있다. 그러나 FFT 알고리즘의 경우 스펙트럼의 갱신 주기가 짧아지면 이에 비례하여 연산량이 증가하므로 많은 H/W 자원이 요구되고 수신 신호의 특성이 알려져 있어 주파수 대역의 일부 정보만이 요구되는 경우에도 전대역 스펙트럼을 연산해야 하므로 불필요한 H/W 자원 낭비가 발생한다. 이러한 문제점들은 어뢰나 미사일과 같은 유도무기 체계와 같이 가용한 H/W 자원이 제한되어 있는 환경에서 시스템의 성능을 제한하는 요소로 작용할 수 있으므로, 실시간 처리 시스템을 구현하기 위해서는 반드시 해결되어야만 한다. Generalized sliding FFT (GSFFT)와 같은 sliding 기법과, transform decomposition (TD)와 같은 pruning 기법은 각각 짧은 갱신 주기와 국부 주파수 스펙트럼 연산 시 발생하는 문제에 대한 좋은 해결책이 될 수 있으나 [1],[2], 두 가지 문제가 동시에 발생하는 경우 최적화된 성능을 보장하지 못한다는 단점이 있다. 이러한 문제를 해결하기 위해 본 논문에서는 기존의 GSFFT와 TD 알고리즘을 결합시킨 pruned GSFFT (PGSFFT) 알고리즘을 제안한다. 제안된 알고리즘은 GSFFT와 TD의 장점을 동시에 활용할 수 있으므로, 갱신 주기가 짧고 국부 스펙트럼을 연산할 때에도 계산량을 감소시킬 수 있다. 본 논문의 구성은 다음과 같다. II장에서는 기존의 GSFFT와 TD 알고리즘에 대해 설명하고 III장에서 본 논문에서 제안하는 Pruned GSFFT(PGSFFT) 알고리즘을 소개한다. IV장에서는 기존의 sliding, pruning 알고리즘과 제안된 PGSFFT 알고리즘의 연산량을 비교, 분석하고, V장에서 본 논문의 결론을 맺는다.

본 연구는 2009년 국방과학연구소의 “광대역 소나의 신호처리 최적화 기법 연구”에 대한 연구용역으로 이루어졌음.

Ⅱ. GSFFT 와 TD 의 개요

1. Generalized Sliding FFT

GSFFT (generalized sliding FFT)는 윈도우가 이동하면서 갱신되는 샘플과 관련된 나비 (butterfly)만을 연산하고 나머지 부분은 메모리에 저장된 이전 결과 값을 이용함으로써, 연산량 측면에서 이득을 취할 수 있는 알고리즘이다 [2]. GSFFT에 대해 보다 자세하게 설명하기 위해 radix-2 FFT 알고리즘을 예로 들어 설명하면 그림 1과 같다. 그림 1에서 보듯이 윈도우의 크기 N, 즉 FFT의 길이는 8이며 갱신되는 샘플의 크기가 2라고 하면 이전 윈도우와 현재 윈도우를 연산할 때 중복되는 부분은 그림 1에 ‘A’로 표시된 부분이다. GSFFT에서는 그림 1과 같이 현재 연산을 수행할 때 중복되는 부분의 연산값을 메모리에 미리 저장된 값을 읽어 들여 사용하고, 연산을 수행한 후 다음 연산에 사

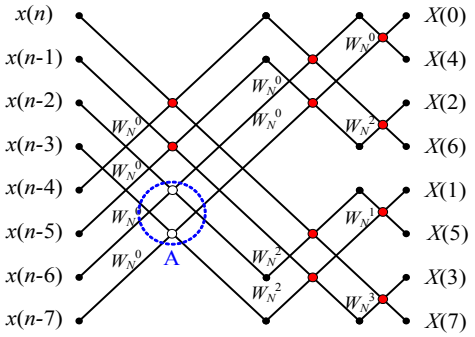

그림 1. radix 2 FFT (N = 8, M = 2).

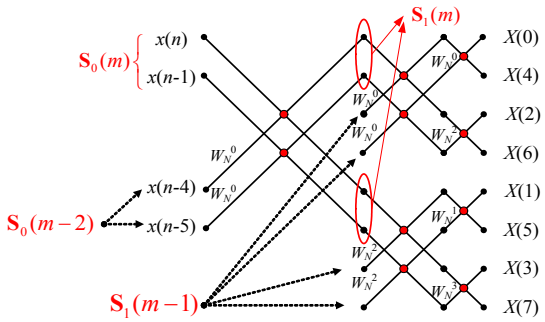

그림 2. Generalized Sliding FFT (GSFFT)

용될 값을 메모리에 저장하여 불필요한 연산을 제거한다. 이러한 과정을 반영하여 그림 1을 수정하면 그림 2와 같다.

그림 2를 설명하기 위해 m번째 윈도우 이동에 의해 갱신된 시간영역 샘플들을 다음과 같이 정의한다 [3].

$$\mathbf{U}(m) = [x(n)\ x(n-1)\ \ldots\ x(n-2^k+1)]^T . \quad (1)$$

여기서 k는 $M = 2^k$을 만족하는 정수이고 M은 갱신된 시간 샘플의 개수이다. 그림 2에서 $\mathbf{S}_i(m)$은 i번째 마디의 상태 벡터를 뜻하며, i 은 radix 2 FFT의 stage 인덱스를 나타낸다. 그리고 이전에 저장된 데이터를 사용하는 스테이지 수를 나타내는 인덱스 q를 $\log_2 N/M$으로 정의한다. m번째 시행에서 i 가 $0 \le i < q$ 인 경우, 즉 저장된 메모리를 사용하여 연산을 수행하는 경우 $\mathbf{S}_{i+1}(m)$은 다음과 같이 표현할 수 있다 [3].

$$\mathbf{S}_{i+1}(m) = \left(\mathbf{P}_{i,k}\left(\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\otimes \mathbf{I}_{2^{k+i}}\right)\right)\begin{bmatrix}\mathbf{S}_i(m)\\ \mathbf{V}_{i,k}\mathbf{S}_i(m-2^{q-i-1})\end{bmatrix} \quad (2)$$

$$\begin{cases}\mathbf{S}_0 = \mathbf{U}(m)\\ \mathbf{P}_{i,k} = \mathbf{P}_i \otimes \mathbf{I}_{2^k}\\ \mathbf{V}_{i,k} = \mathbf{V}_i \otimes \mathbf{I}_{2^k}\\ \mathbf{P}_i = [\mathbf{e}_{\text{odd}}\ \mathbf{e}_{\text{even}}]\\ \mathbf{V}_i = \text{diag}(W_N^{\sigma(0)}, W_N^{\sigma(1)}, \ldots, W_N^{\sigma(2^i-1)})\end{cases}, \quad (3)$$

여기서 $\mathbf{I}_2{}^k$는 $2^k \times 2^k$ 단위행렬이고 $\otimes$ 는 텐서 곱 (Kroneker product)을 의미한다. $\mathbf{P}_i$ 는 $2^{i+1}\times 2^{i+1}$ 치환행렬 (permutation matrix) 이고, $\mathbf{e}_{\text{odd}}$와 $\mathbf{e}_{\text{even}}$은 각각 $[\mathbf{e}_1\ \mathbf{e}_3\ \ldots\ \mathbf{e}_{2^{i+1}-1}]$ 와 $[\mathbf{e}_2\ \mathbf{e}_4\ \ldots\ \mathbf{e}_{2^{i+1}}]$ 이다. $\mathbf{e}_r$은 r번째 $2^{i+1}\times 2^{i+1}$ 단위 행렬의 열벡터 (column vector)를 나타낸다. $\mathbf{V}_i$는 twiddle factor $W_N^{\sigma(r)}$의 대각 행렬을 의미하며, 여기서 $\sigma(r)$는 r의 역비트 (reverse bit)를 취한 값이다. 그리고 $q \le i < k+q$인 경우, 저장된 값을 이용하지 않는 경우 $\mathbf{S}_i(m)$는 다음과 같이 표현되고 [3],

$$\mathbf{S}_i(m) = [S_{i,0}(m)\ S_{i,1}(m)\ S_{i,N-1}(m)]^T . \quad (4)$$

i+1번째 스테이지의 상태 벡터는 다음과 같이 계산된다.

$$\begin{bmatrix}\mathbf{S}_{i+1,t}(m)\\ \mathbf{S}_{i+1,t+2^{k+q-i-1}}(m)\end{bmatrix} = \begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}\mathbf{S}_{i,t}(m)\\ \mathbf{V}_{i,t}\mathbf{S}_{i,t+2^{k+q-i-1}}(m)\end{bmatrix}, \quad (5)$$

여기서 $\mathbf{V}_{i,t}$ 는 $W_N^{\sigma_i(t)}$이고 $\sigma_i(t)$는 $<t-1>_{N/2}{}^{i+1}$ 의 역비트를 취한 값이다. 그리고 $<t-1>_{N/2}{}^{i+1}$에서 $<>$는 modulo 연산자를 나타낸다. 위의 전개 과정을 이용하여 GSFFT에 필요한 연산을 구해보면 필요한 총 복소곱의 수는 $(N/2)\log_2 M + N - M$이다. 한 개의 복소곱은 2개의 실수 더하기와 4개의 실수 곱셈으로 구성되고, 나비 하나당 각각 4개의 실수 덧셈이 필요하므로 GSFFT에서 필요한 총 연산의 수는 $5N\log_2 M + 10N - 10M$이다.

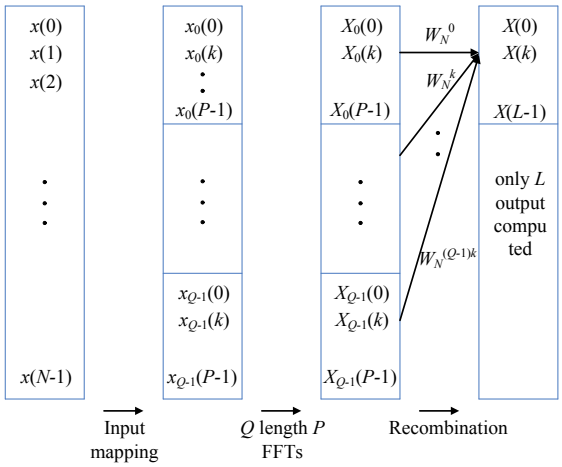

그림 3. transform decomposition의 블록다이어그램.

2. Transform Decomposition

TD (Transform Decomposition) 알고리즘은 특정 주파수 대역만을 연산하기 위해 제안된 pruning 알고리즘중의 하나로, Cooley-Tukey FFT 알고리즘의 분할정복법 (divide-and-conquer)에 그 기반을 두고 있다 [2]. TD 알고리즘은 그림 3에서 보는 바와 같이 3단계의 과정을 거쳐 수행되는데, 먼저 1단계에서는 입력 샘플들을 Q개의 P 블록으로 재정렬하고 다음 단계에서는 각 블록의 샘플들에 대해 DFT를 수행한다. 마지막 단계로 필요로 하는 주파수 대역만을 연산하기 위해 2단계에서 구한 DFT결과를 다시 DFT 연산을 통해 결합한다. TD 알고리즘의 세부적인 설명은 다음과 같다.

입력 샘플들을 Q개의 P 블록으로 재정렬하는 첫 번째 단계에서 시간 인덱스는 다음과 같이 표현된다.

$$n = Qn_1 + n_2, \text{ where } \begin{cases} n_1 = 0,1,\ldots,P-1 \\ n_2 = 0,1,\ldots,Q-1 \end{cases} \quad (6)$$

식 (6)을 이용하면 DFT식을 다음과 같이 변경할 수 있다.

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n) W_N^{nk} \qquad k = 0,1,\ldots,N-1 \\ &= \sum_{n_2=0}^{Q-1} \sum_{n_1=0}^{P-1} x(n_1 Q + n_2) W_N^{(n_1 Q + n_2)k} \\ &= \sum_{n_2=0}^{Q-1} \left[\sum_{n_1=0}^{P-1} x(n_1 Q + n_2) W_P^{n_1 < k >_p} \right] W_N^{n_2 k} \end{aligned} \quad (7)$$

여기서 $< >_p$ 는 modulo P 연산자를 뜻하며, 식 (7)은 다음과 같이 쓸 수 있고

$$X(k) = \sum_{n_2=0}^{Q-1} X_{n_2}(< k >_p) W_N^{n_2 k} \quad , \quad (8)$$

여기서 $X_{n_2}(< k >_p)$ 는

$$X_{n_2}(j) = \sum_{n_1=0}^{P-1} x(n_1 Q + n_2) W_P^{n_1 j} = \sum_{n_1=0}^{P-1} x_{n_2}(n_1) W_P^{n_1 j} . \quad (9)$$

이다. 식 (9)에서 인덱스 j는 0 에서 P – 1 사이의 값이다. TD의 두 번째 단계는 식 (9)을 이용한 P 포인트 DFT이며, 식 (9)를 연산하는 데 어떤 FFT알고리즘을 쓰는가에 따라 TD의 연산량이 결정된다. TD의 마지막 단계는 식 (9)의 결과와 식 (8)을 이용해 원하는 주파수 대역 결과를 결합하는 단계이다. 이 단계에서 Goertzel 알고리즘을 식 (8)에 적용하면 일반적인 DFT 연산을 사용하는 것보다 연산량을 반으로 줄일 수 있다 [4]. 식 (8)에 $m = Q - n_2 - 1$을 대입하면 식 (8)은 아래와 같이 $(W_N^k)^{j-1}$수열과 $X_{Q-j-1}(< k >)_p$수열의 컨볼루션 연산으로 변형된다.

$$\begin{aligned} X(k) &= \sum_{n_2=0}^{Q-1} X_{n_2}(< k >_p) W_N^{n_2 k} \\ &= \sum_{m=0}^{Q-1} X_{Q-m-1}(< k >_p)(W_N^k)^{Q-m-1} \end{aligned} \quad (10)$$

여기서 $y_k(j)$를 다음과 같이 두면

$$y_k(j) = \sum_{m=0}^{j-1} X_{Q-m-1}(< k >_p)(W_N^k)^{j-m-1} \quad (11)$$

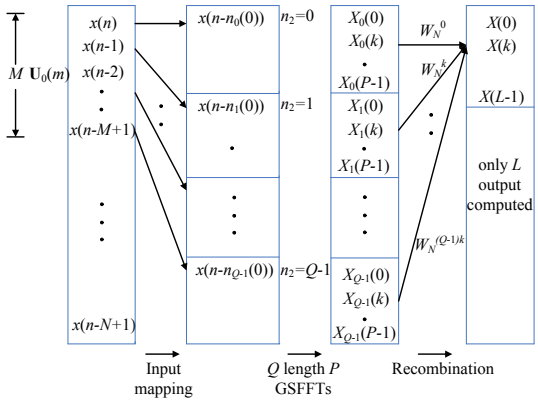

그림 4. Pruned generalized sliding FFT (PGSFFT).

식 (10)은 $y_k(j)$의 Q번째 출력이 된다.

위에서 언급한 내용을 토대로 TD의 연산량을 계산하기 위해 두 번째 단계와 세 번째 단계에서 각각 radix-2 FFT와 Goertzel을 사용하였다고 가정한다. 이때 필요한 주파수 빈 (frequency bin)의 개수가 L이라고 하면 총 $2N\log_2 P + 2LQ + 2L$번의 실수 곱셈과 $3N\log_2 P + 4L(Q-1) + 4L$의 실수 덧셈이 요구되므로, TD 알고리즘에 필요한 총 연산의 수는 $5N\log_2 P + 6LQ + 2L$가 된다.

Ⅲ. Pruned GSFFT

윈도우를 이동시키면서 원하는 주파수 대역만을 연산하기 위해서는 이전 윈도우와 현재 윈도우 사이의 중복된 연산들과 불필요한 주파수 빈을 계산하면서 발생하는 연산을 제거할 수 있는 알고리즘이 요구된다. 이러한 요구를 만족시키기 위해 본 논문에서는 TD의 P-point FFT가 다른 FFT 알고리즘으로 대체가 가능하다는 점에 착안하여, P-point FFT에 GSFFT를 사용함으로써 pruning 알고리즘과 sliding 알고리즘의 장점을 동시에 가지는 pruned GSFFT (PGSFFT) 알고리즘을 제안한다. 그림 4는 PGSFFT의 개념도를 나타내고 있으며, 그림 4에서 시간 색인 n_1과 n_2는 다음과 같이 정의된다.

$$n_{n_2}(n_1) = Qn_1 + n_2, \text{ where } \begin{cases} n_1 = 0,1,\ldots,M/Q-1 \\ n_2 = 0,1,\ldots,Q-1 \end{cases} \quad (12)$$

여기서 M은 시간샘플의 갱신 간격이고 Q는 TD의 P 블록의 개수이다. 이를 이용해 n_2번째의 P-point GSFFT의 입력을 다음과 같이 정의할 수 있다.

$$\mathbf{S}_0^{n_2}(m) = \left[x(n - n_{n_2}(0)) x(n - n_{n_2}(1)) \ldots x\left(n - n_{n_2}\left(\frac{M}{Q} - 1 \right) \right) \right]^T \quad (13)$$

각각의 P-point GSFFT 블록은 식 (2)와 식 (5)를 이용해 연산이 가능하며, P-point GSFFT 블록들에서 구해진 결과는 식 (9)의 $Xn_2(j)$와 동일하므로 식 (8)을 이용해 원하는 주파수 대역만을 연산할 수 있다.

표 1. PGSFFT 와 FFT 알고리즘과의 비교

Algorithm	Real operations
Cooley-Tukey [5]	$5N\log_2 N$
Transform Decomposition (Cooley-Tukey) [2]	$5N\log_2 P+\frac{6LN}{P}+2L$
Pruned FFT (Skinner) [6]	$5N\log_2 L+2N-2L$
GSFFT [3]	$5N\log_2 M+10N-10M$
PGSFFT	$5N\log_2\frac{MP}{N}+10N-10M+\frac{6LN}{P}+2L$

PGSFFT의 연산량은 다음과 같이 분석할 수 있다. 먼저 윈도우 길이가 P이고 입력이 Q/M인 Q개의 GSFFT는 총 $Q(5N\log_2 Q/M + 10N - 10Q/M)$번의 실수 연산이 필요하다 [3]. Q개의 GSFFT의 연산량과 Goertzel 알고리즘을 식 (11)에 적용하여 구현할 때 필요한 실수 연산의 총합은 다음과 같다 [2][4].

$$\mathrm{OPR}_{\mathrm{PGSFFT}}=5N\log_2\left(\frac{MP}{N}\right)+10N-10M+\frac{6LN}{P}+2L \quad (14)$$

Ⅳ. PGSFFT 의 연산 효율성

표 1에 PGSFFT 알고리즘의 연산량을 다른 알고리즘들과 비교하였다. Radix 2 FFT는 윈도우 사이즈 N에만 의존하고 GSFFT의 연산량은 시간 샘플의 갱신 간격 M에 의존한다. pruning FFT와 TD의 연산량은 연산하고자 하는 주파수 대역의 길이 L에 의존한다. 그림 5에 M과 L에 따른 표 1의 알고리즘들의 연산량을 모의실험을 통해 검증하였다. 모의실험 결과 M이 32보다 작고 L이 약 200샘플 이하일 때 PGSFFT의 연산량이 다른 어떤 알고리즘보다도 작음을 그림 5의 (a)와 (b)를 통해 알 수 있다. 하지만 그림 5의 (c)에서와 같이 M이 커질 수록 연산량이 pruned FFT의 연산량에 육박함을 알 수 있다.

Ⅴ. 결론

본 논문에서는 GSFFT와 TD의 장점을 결합한 PGSFFT 알고리즘을 제안하였다. 시간 샘플의 갱신 간격이 윈도우 길이의 1/8보다 작고 연산하고자 하는 주파수 대역의 길이가 윈도우 길이의 1/3보다 작을 때 제안된 PGSFFT를 이용하면 가장 효율적으로 스펙트럼을 연산할 수 있음을 알 수 있었다. 따라서 레이더나 소나와 같이 스펙트럼 갱신 주기가 짧고 특정 주파수 대역만을 운용하는 시스템에 제안된 알고리즘을 적용하면, 효율적인 스펙트럼 연산이 가능할 것으로 기대된다.

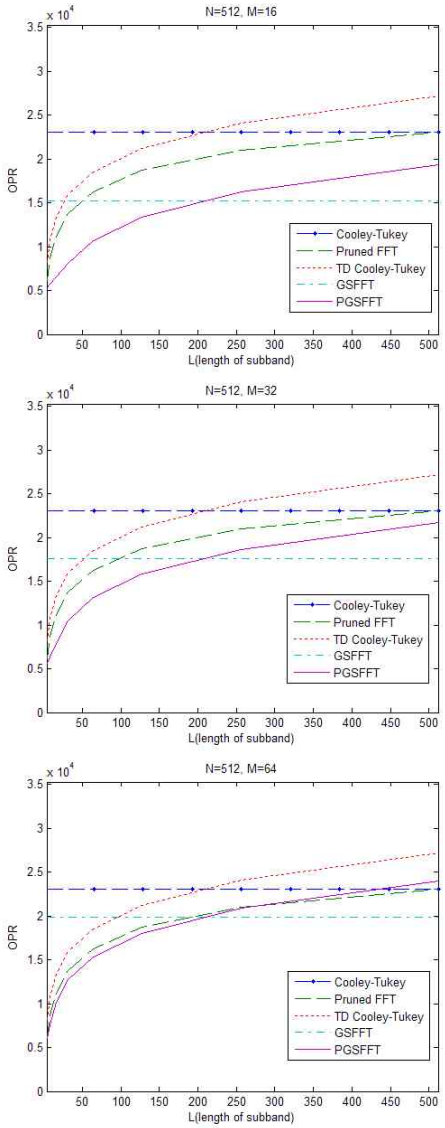

그림 5. M과 L에 따른 PGSFFT의 연산량 (N=512). (a) M=16, (b) M=32, (c) M=64.

참고문헌

[1] B. Farhang-Boroujeny, S. Gazor, "Generalized sliding FFT and its application to implementation of block LMS adaptive filters," *IEEE Trans. Signal Proc.*, vol. 42, no.4, 1994, 532-538.

[2] H. V. Sorensen, C. Sidney, "Efficient Computation of the DFT with Only a Subset of Input or Output Points," *IEEE Trans. Signal Processing*, vol. 41, no. 3, 1993, 1184-1199.

[3] S. Gazor, B. Farhang-Boroujeny, "A state space approach for efficient implementation of block lms adaptive filters," *Proc. ICCS/ISITA. conf. Commun. Syst.*, Singapore, 1992, 808-812.

[4] G. Goertzel, "An algorithm for the evaluation of finite trigonometric series," *American Math. Month., 65*, 1958, 34-35.

[5] A.V. Oppenheim, "*Discrete Time Signal Processing*," 2nd ed.(Upper Saddle River, NJ: Prentice-Hall, 1996).

[6] D.P. Skinner, "Pruning the decimation in-time FFT algorithm for computing a few DFT points,"*IEEE Trans. Acoust., Speech, Signal Processing,24*(2), 1976, 193-194.