2001年度 第14回 信號處理合同學術大會 論文集 第14卷 1號

Fast Jacket Transform의 VLSI 아키텍쳐

유경주, 홍선영, 이문호, 정진균
전북대학교 전자정보공학부

VLSI Architecture of Fast Jacket Transform

Kyung-Ju Yoo, Sun-Young Hong, Moon-Ho Lee, Jin-Gyun Chung
Div. of Electronic & Information Engineering, Chonbuk National University
e-mail: raceyou@vlsidsp.chonbuk.ac.kr

요 약

Walsh-Hadamard Transform은 압축, 필터링, 코드 디자인 등 다양한 이미지처리 분야에 응용되어왔다. 이러한 Hadamard Transform을 기본으로 확장한 Jacket Transform은 행렬의 원소에 가중치를 부여함으로써 Weighted Hadamard Matrix라고 한다. Jacket Matrix의 cocyclic한 특성은 암호화, 정보이론, TCM 등 더욱 다양한 응용분야를 가질 수 있고, Space Time Code에서 대역효율, 전력면에서도 효율적인 특성을 나타낸다 [6],[7]. 본 논문에서는 Distributed Arithmetic(DA) 구조를 이용하여 Fast Jacket Transform(FJT)을 구현한다. Distributed Arithmetic은 ROM과 어큐뮬레이터를 이용하고, Jacket Matrix의 행렬을 분할하고 간략화하여 구현함으로써 하드웨어의 복잡도를 줄이고 기존의 시스톨릭한 구조보다 면적의 이득을 얻을 수 있다. 이 방법은 수학적으로 간단할 뿐 만 아니라 행렬의 곱의 형태를 단지 덧셈과 뺄셈의 형태로 나타냄으로써 하드웨어로 쉽게 구현할 수 있다. 이 구조는 입력데이타의 워드길이가 n일 때, $O(2n)$의 계산 복잡도를 가지므로 기존의 시스톨릭한 구조와 비교하여 더 적은 면적을 필요로 하고 FPGA로의 구현에도 적절하다.

Ⅰ. 서 론

트랜스폼 기법은 이미지, 신호처리등 여러 분야에서 유용하게 사용되어진다. DSP 응용의 여러 분야에서 신호 파라미터는 푸리에 전력 스펙트럼을 사용하여 측정되어진다. 푸리에 변환 계산은 비교적 복잡하고 하드웨어 효율면이나 파라미터 측정의 정확성면에서 중요하게 사용되어진다 [5]. 이러한 분야에 Fast Hadamard Transform(FHT)이 사용되어지며 이는 적응형 필터나 DFT 스펙트럼 필터구현을 위한 DFT 계산에도 효율적이라고 알려져 있다.

일반적인 주파수 영역에서의 FIR 필터는 왈쉬 주파수 영역으로 쉽게 변환되어질 수 있고 유사하게 유한 임펄스 응답 필터 구현에도 이러한 결과를 이용할 수 있다. 1D FHT를 기반으로 2D FHT를 구현하게 되면 손실없는 이미지 압축이 가능하게 된다 [5]. FHT는 직교 트랜스폼이므로 단지 덧셈과 뺄셈만으로 구현이 가능하고 sinusodial 같은 트랜스폼보다 더 빠른 성능을 보이며 DFT와 유사하게 행렬- 벡터곱으로 나타낼 수 있다 [2]. 즉, N 입력일 경우 $Nlog_2N$ 덧셈과 뺄셈으로 고속 알고리즘을 가지게 된다 [3],[4].

FHT는 DFT와 유사하고 순환적인 구조를 가진다. 하다마드 트랜스폼의 시스톨릭한 배열형태의 칩은 South California University에 의해 설계되어지고 MOSIS가 제작하였다 [3]. 이 칩은 $2N-1$ 클럭 사이클, Latency N이 필요하지만 배열에서 덧셈구현은 word level 이므로 시간은 log_2N+n 에 비례하여 증가한다. n은 입력 샘플의 워드 길이이다. 새롭게 제안된 비트 레벨 시스톨릭 구조의 속도를 증가시키기 위한 기법이 [2]에 제안되었다. 이것은 latency는 $N(n+log_2N)$이고, $(2N-1)(n+log_2N)$의 클럭 사이클을 가진다.

Jacket Transform은 Hadamard Transform을 기본으로 확장한 것이다. Jacket Transform은 행렬의 원소에 가중치를 부여함으로써 Weighted Hadamard Matrix라고 한다. Jacket Matrix의 cocyclic한 특성은 암호화, 정보이론, TCM 등 더욱 다양한 응용분야를 가질 수 있고, Space

Time Code에서 대역효율, 전력면에서도 효율적인 특성을 나타낸다.

ROM based DA라 불리는 기존의 DA는 미리 계산된 데이타를 발생시키기 위하여 bit level 의 내적으로 다양한 입력을 분해시킨다. ROM based DA는 미리 계산되 데이타를 롬에 저장하고 그것을 규칙적인 형태로 이용함으로써 VLSI 구현시 면적면에서 효율적이다. 하지만 inner product 의 크기가 증가할때 ROM 면적은 지수적으로 증가하게 되고 면적 효율이 떨어지게 된다 [1]. DA based ROM를 사용하게 되면 효율적인 구현이 가능하게 된다. 기본적으로 여러 ROM 시퀀스와 덧셈, 뺄셈, 입력 데이타의 쉬프트 연산이 필요하다. 이러한 모든 함수들이 FPGA에 효율적으로 매핑된다 [8]. 본 논문의 목적은 FJT의 빠른 연산을 효율적인 구조로 나타내는 것이다. 즉 ROM의 크기를 줄이고 덧셈과 뺄셈 연산의 수를 최소화시켜 계산 속도를 빠르게 하는 작업이 필요하다. 2절에서는 수학적인 모델을 제시하고 3절에서는 DA와 Sparse Matrix 기법을 이용한 구현에 관하여 논하고 4절에서 제안된 구조를 보여주며 5절에서 결론을 맺는다.

II. 수식적인 배경

일반적으로 Hadamard Matrix는 다음과 같이 반복적으로 적용된다.

$$H_2=\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad \text{and} \quad H_{2N}=\begin{bmatrix} H_N & H_N \\ H_N & -H_N \end{bmatrix}$$

여기에서 H_N은 $N\times N$의 Hadamard Matrix이다. 나아가서 Transform 길이 N이 2의 지수의 형태라면, 고속 푸리에 변환과 비슷한 고속자켓변환(FJT)은 고속의 계산을 할 수가 있다. 입력 데이터와 변환된 데이터를 각각 사이즈가 N인 두 개의 벡터를 X와 Y라 하자. 그러면 Y는 다음과 같이 구할 수 있다.

$$Y=H_N X \tag{1}$$

다시 정리를 하게 되면,

$$Y_i=\sum_{k=0}^{N-1} H_{N,ik} X_k \tag{2}$$

$\{X_k\}$를 (3)식과 같이 두 부분으로 나눌 수 있다.

$$X_k=-x_{k,n-1}+\sum_{m=1}^{n-1} x_{k,n-1-m}2^{-m} \tag{3}$$

여기에서 n이 워드의 길이일 때, $x_{k,m}$은 x_k의 m번째 비트이고 $x_{k,n-1}$은 부호비트이다.
(3)식을 (2)식에 대입을 하게 되면,

$$\begin{aligned} Y_i &= \sum_{k=0}^{N-1} H_{N,ik}\left(-x_{k,n-1}+\sum_{m=1}^{n-1} x_{k,n-1-m}2^{-m}\right) \\ &= -\sum_{k=0}^{N-1} H_{N,ik}x_{k,n-1}+\sum_{m=1}^{n-1}\left(\sum_{k=0}^{N-1} H_{N,ik}x_{k,n-1-m}\right)2^{-m} \end{aligned} \tag{4}$$

H_{n-1-m}과 H_{n-1}을 다음과 같이 정의한다.

$$\begin{aligned} H_{n-1-m} &= \sum_{k=0}^{N-1} H_{N,ik}x_{k,n-1-m} \quad (m\neq 0) \\ H_{n-1} &= -\sum_{k=0}^{N-1} H_{N,ik}x_{k,n-1} \end{aligned} \tag{5}$$

최종적인 출력은 다음과 같이 구할 수 있다.

$$Y_i=\sum_{m=0}^{n-1} H_{n-1-m}2^{-m} \tag{6}$$

H_m은 $x_{k,m}$값에 의존하며 2^N개의 값을 가지게 된다. 그러므로, 예상되어지는 값들을 미리 계산을 하여 ROM에 이 값들을 저장해 놓는다. N비트의 입력들 $(x_{1,m}, x_{2,m}, \cdots, x_{N,m})$은 ROM의 주소로 작용하여 지정된 H_m의 값을 가져오게 된다.

III. 1-D FJT 구현 방법

(1)식과 같은 방법으로 직접 구현을 하게 되면($N=8$) 56개(N(N-1))의 덧셈과 뺄셈을 필요로 한다. 그러나 FJT에 사용되는 덧셈과 뺄셈 연산의 수는 24개로 감소된다. FJT 행렬 계수의 대칭적인 특징과 (7), (8)식과 (9)식에서와 같이 행렬을 분할하는 방법을 이용하면, 수식연산의 수를 감소시키고 계산 과정의 속도를 증가시킬 수 있다.

$$\begin{aligned} Y &= H_N X \\ &= H_N^4 H_N^2 X \\ &= H_N^4 T \end{aligned} \tag{7}$$

여기에서 H_N^4와 H_N^2는 일률적으로 각각 4개와 2개의 0이 아닌 값을 가지고 있는 행렬이다.

$$\begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \\ Y_5 \\ Y_6 \\ Y_7 \\ Y_8 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 \\ 1 & 0 & 1 & 0 & -1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 & 0 & -1 & 0 & -1 \\ 1 & 0 & -1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 & 0 & -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \\ X_8 \end{bmatrix} \quad (8)$$

$$\begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \\ Y_5 \\ Y_6 \\ Y_7 \\ Y_8 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 \\ 1 & 0 & 1 & 0 & -1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 & 0 & -1 & 0 & -1 \\ 1 & 0 & -1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 & 0 & -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \\ T_7 \\ T_8 \end{bmatrix} \quad (9)$$

$$\begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \\ T_7 \\ T_8 \end{bmatrix} = \begin{bmatrix} X_1 + X_2 \\ X_1 - X_2 \\ X_3 + X_4 \\ X_3 - X_4 \\ X_5 + X_6 \\ X_5 - X_6 \\ X_7 + X_8 \\ X_7 - X_8 \end{bmatrix} \quad (10)$$

이것은 DA의 원리를 이용하여 수행할 수 있는 FJT 계산의 2단계와 3단계에서 요구되는 덧셈과 뺄셈의 수를 표현한 것이다. (9)식은 본래 하다마드 행렬 벡터 연산이다. FJT는 하다마드 행렬 내부에 가중치를 부가한 것이다. (9)식을 이용하여 표현하면 다음과 같다.

$$\begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \\ Y_5 \\ Y_6 \\ Y_7 \\ Y_8 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & -i & 0 & i & 0 & -1 & 0 \\ 0 & 1 & 0 & -i & 0 & i & 0 & -1 \\ 1 & 0 & i & 0 & -i & 0 & -1 & 0 \\ 0 & 1 & 0 & i & 0 & -i & 0 & -1 \\ 1 & 0 & -1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 & 0 & -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \\ T_7 \\ T_8 \end{bmatrix} \quad (11)$$

(11)식은 본래 입력벡터 X의 각각의 4개의 성분과 미리 정의된 4개의 계수값의 하나와 곱해서 출력 벡터 Y의 각각의 성분을 구하는 행렬 벡터 계산이다. 이러한 방법에서

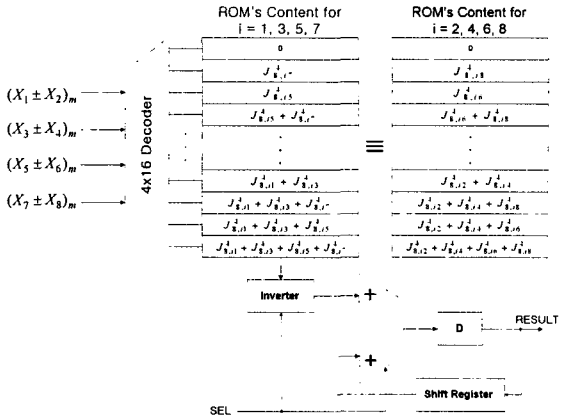

그림 1. Distributed Arithmetic ROM과 어큐뮬레이터 구조

행렬의 형태가 각각의 두 개의 연속되는 행렬이 비슷한 형태의 계수를 가지고 있다. 이것은 $(T_{2i+1})_m$과 $(T_{2i+2})_m$으로 주소로 사용되는 $(2i+1)$의 ROM 테이블과 $(2i+2)$의 ROM 테이블의 내용이 같아서 하나의 ROM으로 사용하여 표현할 수 있다. 그림1은 (4)식, (5)식과 위에서 설명한 J_8^4로부터 계산된 각각의 $i(i=1,2,\cdots,8)$에 의한 ROM 테이블에 따라서 ROM과 어큐뮬레이터를 이용하여 구현한 것이다. ROM에 들어가는 내용은 입력벡터에서 주소번지로 발생될 수 있는 모든 경우를 고려하여 ROM에는 각 행에 해당하는 16개의 내용을 미리 저장해 놓은 것이다.

Ⅳ. 1-D FJT 아키텍처

1-D FJT 구조는 그림2에서 보듯이 변환기로부터 비트 시리얼 형태로 회로에 8개의 입력이 들어간다. 4개의 별도의 RAC 블럭은 다음과 같은 8개의 변환을 계산하고 비트시리얼 형태의 덧셈과 뺄셈의 나비구조가 (9)식의 행렬의 입력원소를 발생시키는데 사용되어진다.

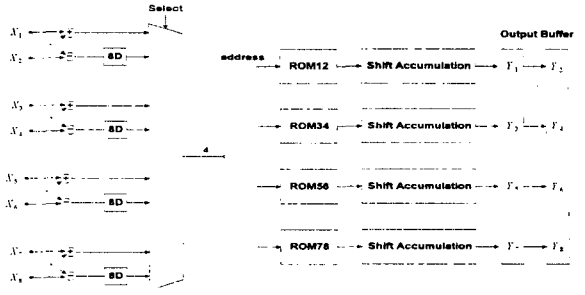

그림 2. 8-포인트 FJT(Fast Jacket Transform) 구조

처음 8-사이클동안 각각의 홀수항이 비트 패러렐 형태로 출력된다. 두번째 8-사이클 동안은 각각의 짝수항이 비트 패럴렐 형태로 출력되어진다. 이는 비트 시리얼 덧셈과 뺄셈의 워드 길이가 n보다 작거나 같을때 입증되어진다. 그렇지 않다면, 연산을 수행하는 동안 8-클럭 지연 외에 또 다른 지연이 있어야 한다. 이것은 부호비트를 포함하여 4비트로 나타내어지는 ROM의 최대값을 가지는 비트 패러렐 데이타를 포함한다. 만약 주어지는 가중치가 크다면 비트수를 최대치를 포함할 수 있도록 비트수를 늘려주어야 한다. RAC블록 내부의 인버터 신호는 매 n-사이클 마다 ROM의 내용을 2의 보수 형태로 계산하기 위해 사용되어지고, Select 신호는 n-사이클 주기를 가지면서 짝·홀수 입력을 선택하기 위해 사용되어진다. 여기에서 사용된 셀렉터는 멀티플렉서(8→4)이고 셀렉터가 동작하는 과정은 $m=0$에서부터 $m=(2\times n)-1$까지 처리되어진다.

예제 1 : (11)식을 이용하여 값을 계산하는 것과 그림 2와 같은 방법으로 구현하여 출력되는 결과를 비교하면 다음과 같다(단, (11)식에서 가중치는 $i=2$라고 가정). 예를 들어 Y_3 에 관해 벡터 내적에 의한 값을 계산 해보면

$$\begin{aligned} T_1 &= X_1 + X_2 = 00010101 \\ T_3 &= X_3 + X_4 = 00001111 \\ T_5 &= X_5 + X_6 = 00110011 \\ T_7 &= X_7 + X_8 = 11001100 \end{aligned}$$

이라고 할 때,

$$\begin{aligned} Y_3 &= T_1 + (-T_3) + T_5 + (-T_7) \\ &= 00010101 - 00001111 + 00110011 - 11001100 \\ &= 01101101 \end{aligned}$$

이다. 벡터내적에 의한 계산과 그림 3의 결과와 같음을 알 수가 있다.

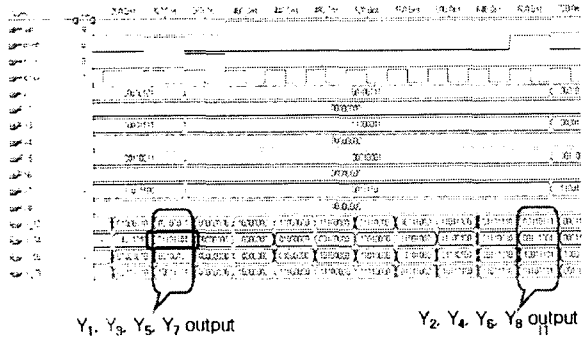

그림 3. 시뮬레이션 결과파형

V. 결 론

트랜스폼 기법은 압축, 필터링, 코드 디자인 등의 이미지 처리와 신호처리 등의 여러 분야에 유용하게 사용되어진다. 이러한 이유로 해서 FJT의 중요성 때문에 이미지와 신호처리에 응용된다. 본 논문에서는 Distributed Arithmetic을 이용해서 FJT 구조를 구현하였다. 이 구조는 입력 데이타의 워드길이가 n일 때, $O(2n)$의 계산 복잡도를 가지게 된다. 본 논문의 제안된 구조는 기존의 구조와 비교해 볼 때 FPGA 구현이 용이하고, 규칙적인 구조이며, 하드웨어 면적이 적고, 계산시간이 짧다.

참 고 문 헌

[1] K.Parhi " VLSI digital signal processing systems Design and implementation" John Wiley & Sons, Inc. USA, 1999.

[2] L.Chang and M.Chang, "A bit level systolic array for Walsh-HadamardTransforms." *Signal Processing Vol 31. pp341-347*, 1993.

[3] S.Y.KUNG, "VLSI Array Processors." *PRENTICE HALL, USA*, 1988

[4] M.H.Lee and M.Kav도, "Fast Hadamard Transform based on a Simple Matrix Factorization." *IEEE Trans. Acoust. Speech vol. ASSP-34, no.6. pp.1666-1667*, 1986.

[5] S.Rahardja and B.J.Falkowski, "Family of Unified Complex Hadamard Transforms." *IEEE Trans. Circuits and Systems II:Analog and Digital Signal Processing. Vol 46. No8*, Aug, 1999

[6] Moon Ho Lee, "A New Reverse Jacket Transform based on Hadamard Matrix." *ISITA 2000. IEEE International symposium on Information Theory, Sorrento (Italy)*, June 25-30 2000.

[7] Moon Ho Lee, "A New Reverse Jacket Transform and Its Fast Algorithm" *IEEE, Trans. on Circuits and System Vol. 47. no. 1* 2000. 1

[8] An FPGA based Walsh Hadamard transforms Amira, A.; Bouridane, A.; Milligan, P. *ISCAS 2001. The 2001 IEEE International Symposium on , Vol 2 . 2001 pp 569 -572.*