社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

FIR フィルタの LUT カスケードによる実現について

笹尾　勤†　井口　幸洋††　鈴木　隆広†††

† 九州工業大学 情報工学部 電子情報工学科　〒 820–8502 福岡県飯塚市大字川津 680–4
†† 明治大学 理工学部 情報科学科　〒 214–8571 川崎市多摩区東三田 1–1–1
††† 明治大学 大学院 理工学研究科　〒 214–8571 川崎市多摩区東三田 1–1–1

あらまし　本論文では, FIR フィルタのビット分散演算を表す関数として, n 入力 q 出力 DA 関数を定義する. 次に, DA 関数を k 入力 q 出力のセルを用いた LUT カスケードで実現する場合 (1). 単一の ROM で実現する場合に比べ必要メモリ量を大幅に削減できる; (2). LUT カスケードのセル数は高々 $\lceil \frac{n-k-1}{k-q} \rceil + 2$ である; (3). LUT カスケードのメモリの総ビット数は, 高々 $(\lceil \frac{n-k-1}{k-q} \rceil + 2)2^k q$ である; (4). LUT カスケードの構造は, 量子化ビット数 q, タップ数 N, セルの入力数 k に大きく依存する; ことを示す.

キーワード　ディジタルフィルタ, ビット分散演算, LUT カスケード, 関数分解, 二分決定グラフ, FPGA

LUT Cascade Realization of FIR Filter

Tsutomu SASAO†, Yukihiro IGUCHI††, and Takahiro SUZUKI†††

† Department of Computer Science and Electronics, Kyushu Institute of Technology
680–4, Kawazu, Iizuka, Fukuoka, 820–8502 Japan
†† Department of Computer Science, Meiji University
Kawasaki 214-8571, Japan
††† Department of Computer Science, Meiji University
Kawasaki 214-8571, Japan

Abstract　This paper first defines the n-input q-output DA function, which denotes the distributed arithmetic for a finite impulse response (FIR) filter. It shows a method to realize the DA function by using k-input q-output cells. The experimental results show that LUT cascade realizations require much smaller memory than the single ROM realization of the DA function.

Key words　Digital filter, Distributed arithmetic, LUT cascade, Functional decomposition, Binary decision diagram, FPGA

1. はじめに

ディジタルフィルタは, 信号処理における基本要素である [7]. ディジタルフィルタは, 有限長のインパルス応答を有する FIR (Finite Impulse Response) フィルタと, 無限長のインパルス応答を有する IIR (Infinite Impulse Response) フィルタとに分類できる. FIR フィルタは, 非再帰形で構成でき, 常に安定な動作を行うという特徴がある. また, 線形位相特性をもたせることが可能であり, 波形伝送などに利用できる. 反面, 急峻な遮断特性を得るためには, 高次のフィルタが必要であり, IIR フィルタに比べハードウエア量が増加する欠点がある.

FPGA 上に FIR フィルタを実現する際, 積和演算を分散演算 (Distributed Arithmetic: DA) という手法で, テーブル参照に置き換えて実行する方法が知られている [6], [12], [13]. テーブル参照には, FPGA 中の RAM ブロックを利用できる.

本論文では, FIR フィルタの分散演算を LUT カスケードという, メモリを直列に接続した回路で実行する方法を提案する. この方法では, 分散和演算を率直に 1 つのメモリで実現するよりも, はるかに少ないメモリ量で実現できる. LUT カスケードの大きさや構造は, 主として, FIR フィルタのタップ数 N, 出力のビット精度 q, LUT カスケードのセルの入力数 k, セルの出力数 q で定まる. 従って, 回路の大きさや性能の予測が容易である. 本手法は, FPGA のブロック RAM や, 専用 LUT カスケードチップ [11] を用いて FIR フィルタを実現する際に応用可能である.

2. FIR フィルタ

［定義 1］ **FIR フィルタ**とは,

$$\mathcal{Y}(n) = \sum_{i=0}^{N-1} h_i \cdot \mathcal{X}(n-i) \tag{1}$$

を計算する回路である. ここで, $\mathcal{X}(i)$ は, 時刻 i における入力 $\mathcal{X}$

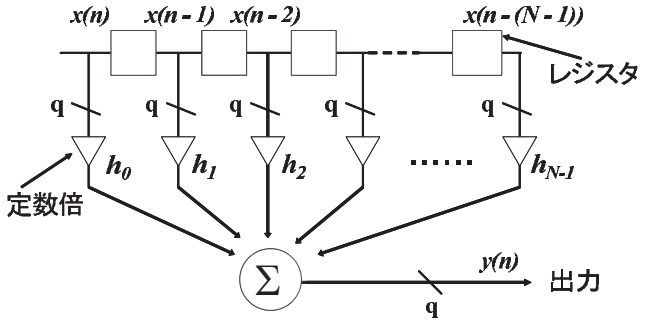

図 1　FIR フィルタ (並列実現).

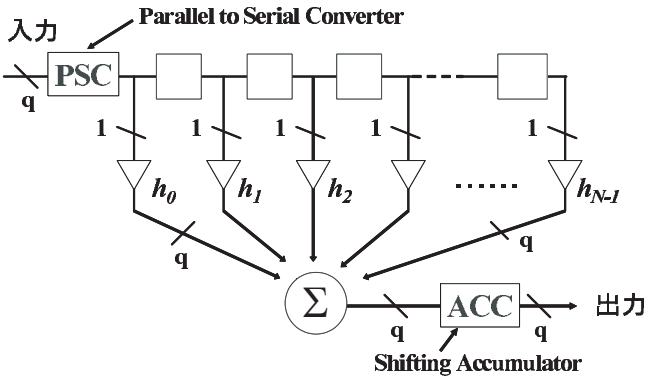

図 2　FIR フィルタ (直列実現).

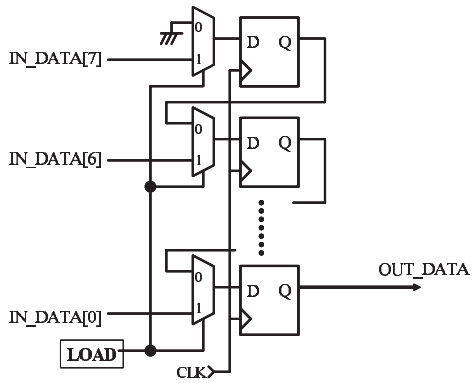

図 3　PSC の実現例.

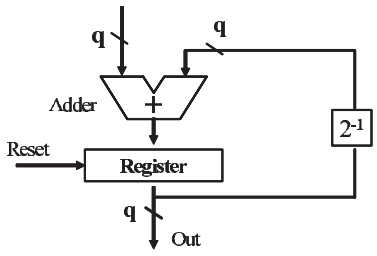

図 4　ACC の実現例.

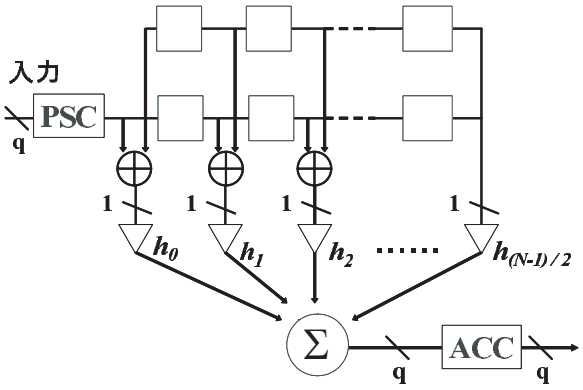

図 5　FIR フィルタ (直列対称実現).

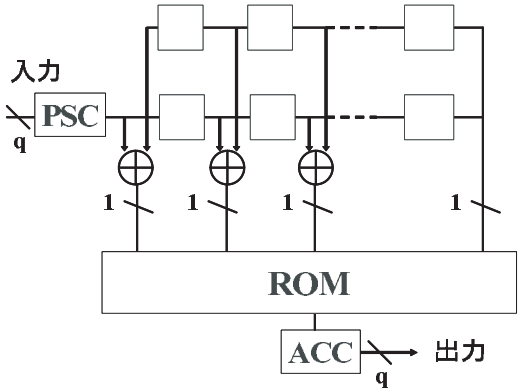

図 6　FIR フィルタ (直列対称 ROM 実現).

の値, $\mathcal{Y}(i)$ は, 時刻 i における出力 $\mathcal{Y}$ の値を表す(注1). また, h_i は q ビットの固定小数点で表現された数で**フィルタ係数**を表し, N は, フィルタの**タップ数**を示す(注2).

図 1 は, 式 (1) を直接実現する回路である. N 段の q ビットシフトレジスタの他に, q ビットの定数乗算器 N 個と, q ビットの数を N 個加算する回路から構成されており, ハードウエア量は多い. 図 1 の回路のハードウエア量を削減するために, q ビットの数をビット直列に計算するように変更した回路が, 図 2 である. ここで, PSC は並列直列変換器 (Parallel to Serial Converter) を表し, 例えば, 図 3 の回路で実現可能である.

また, この場合 $h_0, h_1, \ldots h_{N-1}$ への入力は, 0 または 1 の数となるので, 定数乗算器は, AND ゲートで置き換えが可能である. 図 2 の ACC はシフト累算器 (Shifting Accumulator) を表す. これは, シフト演算をしながら加算を繰り返す回路であり, 例えば, 図 4 の回路で実現可能である. この方式では, ハードウエア量は約 q 分の 1 に削減可能であるが, 処理時間は約 q 倍となる. 図 2 の組合せ回路の部分は N 入力 q 出力となるが, この部分は後で定義する DA 関数を実現する. 線形位相特性を満たす FIR フィルタでは, その係数が関係 $h_i = h_{N-i-1}$ を満たす. このようなフィルタを**対称フィルタ**という. このとき, 図 2 の回路は図 5 の回路に変更でき, ハードウエア量を削減できる. この場合, q ビットの数を $(N+1)/2$ 個加算する回路を用いればよい. また, 図 5 の $\oplus$ の記号は直列加算器を示す(注3). 図 6 は, 組合せ回路の部分を ROM で実現したものである. 例えば入力数が 3 の場合, ROM には表 1 に示す値を予め計算して貯えておく. これ

(注1)：本論文では, $\mathcal{X}$ や $\mathcal{Y}$ 等の変数は, フィルタで用いる信号値を表し, x_i は論理変数, X_1, X_2 は論理変数のベクトルを表す.

(注2)：一般に入力変数のビット数，フィルタ係数のビット数，出力のビット数は異なっても良い．しかし，本論文では簡単のため全て q ビットで表現されているものとする.

(注3)：直列加算の直前には Reset 信号を加えて, レジスタやフリップ・フロップの値を 0 にする必要がある.

表 1　DA 関数の例.

Address	Data
000	0
001	h_0
010	h_1
011	$h_0 + h_1$
100	h_2
101	$h_0 + h_2$
110	$h_1 + h_2$
111	$h_0 + h_1 + h_2$

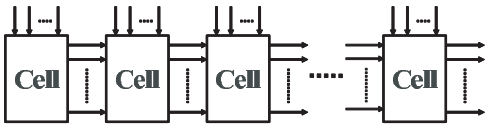

図 7　LUT カスケード.

表 2　分解表の例.

$\vec{X}_1 = (x_0, x_1, x_2)$

$\vec{X}_2 = (x_3, x_4)$	000	001	010	011	100	101	110	111
00	001	010	011	100	001	100	100	100
01	111	110	000	011	111	010	010	011
10	010	110	010	001	010	010	010	001
11	010	011	101	010	010	011	011	010

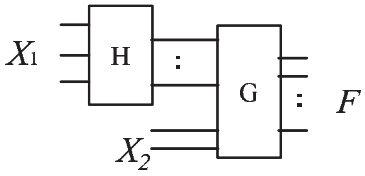

図 8　関 数 分 解.

は, 後で定義する DA 関数の例となっている. これは, **分散演算** (Distributed Arithmetic) とも呼ばれ, 多数の乗算器と多入力の加算器を 1 個のメモリで置き換えることができるので, 畳み込み演算を実現する際に多用されている [2], [3], [5], [6], [12], [13]. 但し, この方法を利用できるのは, 一方の入力が定数のときのみである.　FIR フィルタの場合には, フィルタ係数が一定であり, この手法を使用可能である. 一般に, ROM の入力数を n, 出力数を q とするとき, ROM に必要なメモリ量は, $2^n q$ ビットとなる.

3.　LUT カスケードと関数分解

本章では, LUT カスケードと関数分解の関係を述べる.

［定義 2］　**LUT カスケード**は, 与えられた論理関数を図 7 の回路構造で計算する. 各ブロックを**セル**といい, 各セルは任意の論理関数を実現する. また, 隣接するセル間を接続する信号線を**レイル**と呼ぶ.

［定義 3］　多出力論理関数 $\vec{F}(\vec{X})$ を $\vec{F} = g(h(\vec{X}_1), \vec{X}_2)$ の形で表現することを**関数分解**という. この時, $\vec{X}_1$ の変数を**束縛変数**, $\vec{X}_2$ の変数を**自由変数**という. ここで h は多値の関数である(注4).

［定義 4］　$\vec{X}_1$ の変数を列に, $\vec{X}_2$ の変数を行に対応させ, $\vec{F}(\vec{X}_1, \vec{X}_2)$ の値が, 対応する要素となる表を $\vec{F}(\vec{X}_1, \vec{X}_2)$ の**関数分解表**という. q 出力関数の場合, 各要素は, q ビットのベクトルとなる. 異なる列パターンの数を**列複雑度**という.

［例 1］　図 2 に 5 入力 3 出力論理関数の分解表の例を示す. ここで, $\vec{X}_1 = (x_0, x_1, x_2)$, $\vec{X}_2 = (x_3, x_4)$ である. また, 列複雑度は 5 である.　(例終り)

［定理 1］　関数分解 $\vec{F}(\vec{X}) = g(h(\vec{X}_1), \vec{X}_2)$ において, 分解表の列複雑度が μ のとき, 関数 $\vec{F}(\vec{X})$ は, 図 8 の回路構造で実現可能である. また, 二つのブロック H と G の間の信号線は, $\lceil \log_2 \mu \rceil$ 本あれば十分である.

［定理 2］　関数 $\vec{F}$ の変数を $\vec{X} = (x_0, x_1, \ldots, x_{n-1})$ とする. $\vec{X}_1 = (x_0, x_1, \ldots, x_i), \vec{X}_2 = (x_{i+1}, \ldots, x_{n-1})$ とするとき, 全ての $i (q < i < n-1)$ に関して, $\vec{F}(\vec{X}_1, \vec{X}_2)$ の分解表の列複雑度が 2^q 以下ならば, $\vec{F}$ は, k 入力 q 出力のセルを用いた LUT カスケードで実現可能である. ここで, $k > q$ である.

(証　　明)

$\vec{X}_1 = (x_0, x_1, \ldots, x_{k-1}), \vec{X}_2 = (x_k, \ldots, x_{n-1})$ として, $\vec{F}(\vec{X})$ を関数分解すると, 図 8 の回路が得られる. 二つのブロック間の信号線数を q とすると, G は $n-k+q$ 入力回路となる. G に対して, 再度関数分解を適用する. この際, 束縛変数がすべての中間変数 (即ち, H の出力信号線) を含むようにする. このとき, 分解表の列複雑度はやはり 2^q 以下となる.

以上のように関数分解を繰り返し適用することにより, 図 7 のカスケード回路を得る.　□

4.　DA 関数とその LUT カスケード実現

［定義 5］　n 入力 q 出力 **DA 関数** $\vec{F}(x_0, x_1, \ldots, x_{n-1})$ とは,

$$\sum_{i=0}^{n-1} h_i \cdot x_i \qquad (2)$$

の値を計算し, その結果を q ビットの 2 進数で表現する n 入力 q 出力論理関数である(注5). ここで, h_i は実数係数を表し, q ビットの 2 進数で表現されているとする. また, 数の表現には, 固定小数点表現を用い, 負数は 2 の補数表現を用い, 量子化ビット数には, 符号のための 1 ビットを含むものとする. h_i の二進表現を $(\vec{h}_i)_2$ とすると, $\vec{F}$ は,

$$\begin{aligned}
\vec{F}(1, 0, \ldots, 0) &= (\vec{h}_0)_2 \\
\vec{F}(0, 1, \ldots, 0) &= (\vec{h}_1)_2 \\
&\vdots \\
\vec{F}(0, 0, \ldots, 1) &= (\vec{h}_{n-1})_2
\end{aligned}$$

を満たすものとする.

［定義 6］　フィルタ係数 $h_0, h_1, \ldots, h_{n-1}$ を q ビットで量子化後加算して (量子化後加算) 生成した関数を **DA1 関数**という. 一

(注4)：文献 [1] では h が 2 値の場合のみ考案している. 文献 [4] でも $\vec{F}$ は単一出力関数の場合のみ考案している.

(注5)：非対称フィルタの場合は, $n = N$ であり, 対称フィルタの場合は, $n = (N+1)/2$ である.

方, フィルタ係数 $h_0, h_1, \ldots, h_{n-1}$ を実数として加算後, q ビットに量子化 (加算後量子化) して生成した関数を **DA2 関数**という.

DA1 関数も DA2 関数も定義 1 を満たすが, 一般には, DA1 関数と DA2 関数は異なる. DA1 関数は, 実現する際のハードウエア量が少なく, DA2 関数は FIR フィルタでの誤差が少ない. 次の補題は, q 出力 DA1 関数の分解表の列複雑度が高々2^q であることを示す.

［補題 1］ n 入力 q 出力 DA1 関数を $\vec{F}(x_0, x_1, \ldots, x_{n-1})$ とする. いま, $(\vec{X_1}, \vec{X_2})$ を $\vec{X} = (x_0, x_1, \ldots, x_{n-1})$ の分割, ただし, $\vec{X_1} = (x_0, x_1, \ldots, x_{i-1}), \vec{X_2} = (x_i, x_{i+1}, \ldots, x_{n-1})$ とする. $\vec{X_1}$ を束縛変数, $\vec{X_2}$ を自由変数とする $\vec{F}$ の分解表を考える. このとき, この分解表の列複雑度は高々2^q である.

(証　　明)

$i \leqq q$ のとき, 列複雑度は高々 2^q であるため, $i > q$ の場合を考える. 分解表の第 1 行目, 即ち, $\vec{X_2} = (0, 0, \ldots, 0)$ の行を考える. このとき, 分解表の各要素は q ビットのベクトルであるので, 異なる要素の数は高々 2^q 個である. 従って, $\vec{a}, \vec{b} \in \{0,1\}^i, \vec{a} \neq \vec{b}$ としたとき, $\vec{F}(\vec{a}, \vec{0}) = \vec{F}(\vec{b}, \vec{0})$ となる $\vec{X_1}$ への二つの割り当て $\vec{a}$, $\vec{b}$ に対応する二つの列が存在する. 次に, これらの二つの列の第 j 行目 $(j > 0)$ を考える. このとき, $\vec{X_2} = (x_i, x_{i+1}, \ldots, x_{n-1})$ の値を $\vec{c}$ とすれば, 定義 5 より, $\vec{F}$ は関係

$$\vec{F}(\vec{a}, \vec{c}) = \vec{F}(\vec{a}, \vec{0}) + \vec{F}(\vec{0}, \vec{c})$$
$$\vec{F}(\vec{b}, \vec{c}) = \vec{F}(\vec{b}, \vec{0}) + \vec{F}(\vec{0}, \vec{c})$$

を満たす. ここで, 記号 + は桁上がりを許す 2 進数の加算を示す. 従って, $\vec{F}(\vec{a}, \vec{c}) = \vec{F}(\vec{b}, \vec{c})$ が成立する. この関係は, すべての $j > 0$ で成立するので, $\vec{a}, \vec{b}$ に対応する二つの列のパターンが等しいことがわかる.

以上のことより, 分解表の列複雑度は高々2^q であることが証明できた. □

［例 2］ 表 3 に $n = 5$, $\vec{X_1} = (x_0, x_1, x_2)$, $\vec{X_2} = (x_3, x_4)$ の分解表の例を示す. 今, 分解表の各要素は 2 ビットのベクトルと仮定する. この時, 分解表には, 高々 4 つの異なったベクトルしか現れない. 分解表の第 1 行目, $(x_3, x_4) = (0, 0)$ の行を考える. 今, $(x_0, x_1, x_2) = (0, 1, 1)$ の列の要素と, $(x_0, x_1, x_2) = (1, 0, 0)$ の列の要素が等しいと仮定する. 即ち, $h_1 + h_2 = h_0$ とする. 次に, 分解表の第 2 行目, $(x_3, x_4) = (0, 1)$ の行を考える. DA 関数の性質より, 第 2 行目は, 第 1 行目に h_3 を加えたものである. 従って, $(x_0, x_1, x_2) = (0, 1, 1)$ の列の要素 $h_1 + h_2 + h_4$ と, $(x_0, x_1, x_2) = (1, 0, 0)$ の列の要素 $h_0 + h_4$ が等しい. 即ち, $h_0 + h_1 + h_3 = h_2 + h_3$ が成立する. 同様なことが, 分解表の第 3 行目と第 4 行目に対しても成立する. 即ち, $(x_0, x_1, x_2) = (0, 1, 1)$ の列の要素と, $(x_0, x_1, x_2) = (1, 0, 0)$ の列の要素が等しい. 従って, $(x_0, x_1, x_2) = (0, 1, 1)$ の列のパターンと, $(x_0, x_1, x_2) = (1, 0, 0)$ の列のパターンが等しい. つまり, 第一行目で二つの列が等しければ, 全ての行でその二つの列は等しい. 以上のことから, 分解表の列複雑度は高々 4 であることが証明できる. (例終り)

［定理 3］ q 出力 DA1 関数を実現する LUT カスケードのレール数は q 本あれば十分である. また, 各セルの入力数は $q+1$, 出力数は q あれば十分である.

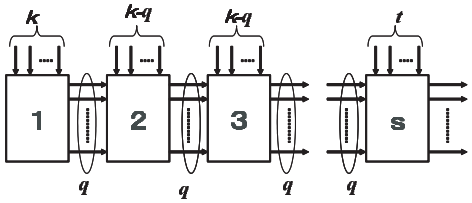

図 9　DA 関数のカスケード実現.

(証　　明)

補題 1 と定理 2 より, 明らかである. □

［定理 4］ n 入力 q 出力 DA1 関数を実現する LUT カスケードを k 入力 q 出力のセルで実現する場合, セル数は, $\lceil \frac{n-k-1}{k-q} \rceil + 2$ 個あれば十分である.

(証　　明)

定理 2 の方法で, DA1 関数をカスケード実現すると, 図 9 のようになる. このカスケードのレイル数を q, セル数を s とする. また, 最後のセルの入力数を t とすると, 関係 $k + (s-2)(k-q) + t = n, 1 \leqq t \leqq k-q$ が成立する. これより, $s \leqq \lceil \frac{n-k-1}{k-q} \rceil + 2$ を得る. □

k 入力 q 出力のセルのビット数は $q2^k$ であることより, 次の系を得る.

［系 1］ n 入力 q 出力 DA1 関数を LUT カスケードで実現する際, k 入力 q 出力のセルを用いた場合, メモリのビット数は $(\lceil \frac{n-k-1}{k-q} \rceil + 2)2^k q$ あれば十分である.

［補題 2］ n 入力 q 出力 DA1 関数を直接実現する ROM のビット数は $q2^n$ である.

［系 2］ n 入力 q 出力 DA1 関数を ROM で直接実現する場合のメモリ量と k 入力 q 出力のセルを用いて LUT カスケードを実現する場合のメモリ量の比は, およそ $(\lceil \frac{n-k-1}{k-q} \rceil + 2)2^{k-n}$ である.

系 2 で $k = q+1$ とおくと, 上の比は $(n-q)2^{q-n+1}$ となる. これより, $n-q$ の値が大きいとき, LUT カスケードを用いた場合のメモリの削減効果が高いことが分かる.

5. 実験結果

前章で得られた理論的結果を確認するために, 多数の FIR フィルタの DA 関数を LUT カスケードを用いて合成した. LUT カスケード設計に分解表を用いるのは能率が悪いので, 二分決定グラフ (binary decision diagram :BDD) を用いた [9], [10]. 設計パラメータは,

- フィルタの種類:(HPF, LPF, BPF, BEF)
- タップ数 N: (9,17,33)
- 量子化ビット数 q: (8,10,12,14)
- 窓関数: (カイザー窓関数, ハニング窓関数)
- DA 関数の種類: (DA1 関数, DA2 関数)

である. 従って, 全部で 192 通りの DA 関数を実現した. フィルタ係数の量子化法は, 下位桁が 0 の時切り捨て, 1 の時最下位ビットに 1 を加算である. 表 5 と表 4 に DA 関数を ROM で実現した場合と LUT カスケードで実現した場合の比較を示す. 但

表 3　DA1 関数の分解表の例.

$\vec{X}_H=(x_3,x_4)$ \ $\vec{X}_L=(x_0,x_1,x_2)$	000	001	010	011	100	101	110	111
00	0	h_2	h_1	h_1+h_2	h_0	h_0+h_2	h_0+h_1	$h_0+h_1+h_2$
01	h_4	h_2+h_4	h_1+h_4	$h_1+h_2+h_4$	h_0+h_4	$h_0+h_2+h_4$	$h_0+h_1+h_4$	$h_0+h_1+h_2+h_4$
10	h_3	h_2+h_3	h_1+h_3	$h_1+h_2+h_3$	h_0+h_3	$h_0+h_2+h_3$	$h_0+h_1+h_3$	$h_0+h_1+h_2h_3$
11	h_3+h_4	$h_2+h_3+h_4$	$h_1+h_3+h_4$	$h_1+h_2+h_3+h_4$	$h_0+h_3+h_4$	$h_0+h_2+h_3+h_4$	$h_0+h_1+h_3+h_4$	$h_0+h_1+h_2+h_3+h_4$

し, 紙面の都合により ($N=33$, カイザー窓関数) の場合のみを示す. 対称フィルタなので $n=(N+1)/2$ である. また, セルの入力数は, $k=q+1$, 出力数は q とした.

実験から次の結果を得た.

（1） 量子化ビット数 q が小さい場合, フィルタ係数 $h_0,h_1,\ldots,h_{n_1}$ を量子化すると, ゼロとなるものが多かった. 特に, DA1 関数の場合には, 列複雑度が大幅に小さくなった.

（2） LUT カスケードを用いると, 総メモリ量を大幅に削減できた. また, DA1 関数 (表 4) の場合には, さらに大幅に削減できた. DA2 関数（表 5) の場合, $q=8$ のとき 32 分の 1 に, $q=10$ のとき 16 分の 1 に, $q=12$ のとき 4 分の 1 に, $q=14$ のとき 2 分の 1 に削減できた.

（3） DA1 関数 (表 4) と，DA2 関数 (表 5) は，大きく異なった. DA1 関数の場合は，補題 1 に示すように，分解表の列複雑度は常に 2^q 以下となる. 特に, 表 4 の DA1 関数では, 分解表の列複雑度は 2^{q-2} 以下となった．一方，DA2 関数の場合, 表 5 の関数では列複雑度は 2^q 以下となったが, ハニング窓関数を用いたものでは 2^q を超える関数も存在した．フィルタ特性は，DA2 関数の方が優れたものとなる. ハードウエア量が一定の場合は, 量子化ビット数 q を増やし, DA1 関数を使用する方法もある.

（4） 図 5 の回路において, h_0 に対応する変数 x_0 を BDD の根側に, $h_{(N-1)/2}$ に対応する変数 $x_{(N-1)/2}$ を BDD の葉側に配置したものを初期順序とし, BDD を繰り返し改善を行うと, 小さなカスケードを得た. これは, 絶対値の小さなフィルタ係数に対応する変数が BDD の根に近くなるように配置すれば, 分解表の列複雑度が小さくなる事からも予測される.

（5） 量子化ビット数 q が大きいほど, また, タップ数 N が大きいほど, 列複雑度は大きくなった.

6. 結　　論

本論文では, FIR フィルタの組合せ回路の部分を表現する関数として DA 関数を定義した. さらに, 量子化後加算したものを DA1 関数, 加算後量子化してたものを DA2 関数と定義した. また, n 入力 q 出力 DA 関数を k 入力 q 出力のセルを用いた LUT カスケードで実現する場合

（1） LUT カスケードで実現することにより, 単一の ROM に比べ必要メモリ量を大幅に削減できる.

（2） LUT カスケードの構造は 主として, n, q, k に依存する.

（3） DA1 関数を実現する LUT カスケードは, DA2 関数を実現する LUT カスケードよりも小型になる.

（4） DA1 関数を実現する LUT カスケードのセル数は高々 $\lceil\frac{n-k-1}{k-q}\rceil+2$ である.

（5） DA1 関数を実現する LUT カスケードのメモリの総ビット数は, 高々$(\lceil\frac{n-k-1}{k-q}\rceil+2)2^k q$ である.

という結論を得た.

LUT カスケード実現に必要なビット数は, 量子化ビット数 q の増加に伴い, 指数関数的に増える. 従って, q の値が大きいときには, フィルタ係数 $h_0,h_1,\ldots,h_{n_1}$ を幾つかの群に分割し, 各群毎に和を計算し, 最後に加算器でその和を計算する実現法 [2] を用いるのが得策である. 各群の DA 関数や，最後の加算器は, LUT カスケードで実現可能である.

本論文では, ディジタルフィルタの分散演算のモデルとて DA 関数を定義したが, DA 関数は、このほか, DCT, DFT, 畳み込み演算にも利用可能である.

謝　　辞

本研究は, 一部, 文部科学省・科学研究費補助金, および, 文部科学省・北九州地域知的クラスター創成事業の補助金による. ご討論頂いた永山忍博士，LUT カスケード設計ツールで協力頂いた中原啓貴氏に感謝する.

文　　献

[1] R. L. Ashenhurst, “The decomposition of switching functions,” *In Proceedings of an International Symposium on the Theory of Switching*, pp. 74-116, April 1957.

[2] Actel Corporation, “CoreFIR: Finite implulse response (FIR) filter generator,” http://www.actel.com/ipdocs/CoreFIR_IP_DS. pdf

[3] R. J. Andraka, “FIR filter fits in an FPGA using a bit serial approach,” *Proceedings of the EE-Times 3rd Annual PLD design Conference and Exhibit*, March 1993.

[4] H. A. Curtis, *A New Approach to The Design of Switching Circuits*, D. Van Nostrand Co., Princeton, NJ, 1962.

[5] T-T. Do, H. Kropp, C. Reuter, and P. Pirsch, “A flexible implementation of high-performance FIR filters on Xilinx FPGAs,” *FPL 1998*, Estonia, Aug. 31 - Sept. 3, 1998, pp. 441-445.

[6] L. Mintzer, “FIR filters with field-programmable gate arrays,” *Journal of VLSI Signal Processing*, Aug. 1993, pp. 120-127.

[7] K. K. Parhi, *VLSI Digital Signal Processing Systems Design and Implementation*, John Wiley, New York, 1999.

[8] T. Sasao, *Switching Theory for Logic Synthesis*, Kluwer Academic

表 4　DA1 関数の ROM による実現と LUT カスケードによる実現の比較 (量子化後加算).

Filter Type (Cut Off Frequency)			HPF (5kHz)				LPF (5kHz)				BPF (5k-10kHz)				BEF (5k-10kHz)			
Window Function			Kaiser				Kaiser				Kaiser				Kaiser			
Tap [N]			33				33				33				33			
Quantize Bit [q]			8	10	12	14	8	10	12	14	8	10	12	14	8	10	12	14
ROM	#IN[n]		17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17
	#OUT[q]		8	10	12	14	8	10	12	14	8	10	12	14	8	10	12	14
LUT Cascade	#LUT		1	1	1	1	1	2	1	2	1	1	2	2	1	1	2	2
	Cell 0	#IN	8	9	12	15	6	6	13	10	8	10	6	7	7	11	7	9
		#OUT	8	10	12	14	8	4	12	8	8	10	4	6	8	10	6	7
	Cell 1	#IN						10		14			13	15			12	14
		#OUT						10		14			12	14			11	14
BDD の最大幅			56	229	278	3776	34	166	672	2697	40	115	220	2186	18	196	196	2894

表 5　DA2 関数の ROM による実現と LUT カスケードによる実現の比較 (加算後量子化).

Filter Type (Cut Off Frequency)			HPF (5kHz)				LPF (5kHz)				BPF (5k-10kHz)				BEF (5k-10kHz)			
Window Function			Kaiser				Kaiser				Kaiser				Kaiser			
Tap [N]			33				33				33				33			
Quantize Bit [q]			8	10	12	14	8	10	12	14	8	10	12	14	8	10	12	14
ROM	#IN[n]		17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17
	#OUT[q]		8	10	12	14	8	10	12	14	8	10	12	14	8	10	12	14
LUT Cascade	#LUT		7	5	3	2	7	5	3	2	8	4	3	2	8	4	3	2
	Cell 0	#IN	9	11	13	14	9	11	13	14	9	11	13	14	9	11	13	14
		#OUT	8	10	11	12	8	10	11	12	8	9	11	12	8	9	11	12
	Cell 1	#IN	9	11	13	15	9	11	13	15	9	11	13	15	9	11	13	15
		#OUT	8	10	11	14	8	10	11	14	8	9	11	14	8	9	11	14
	Cell 2	#IN	9	11	13		9	11	13		9	11	13		9	11	13	
		#OUT	8	9	12		8	9	12		8	9	12		8	9	12	
	Cell 3	#IN	9	11			9	11			9	11			9	11		
		#OUT	8	9			8	9			8	10			8	10		
	Cell 4	#IN	9	11			9	11			9				9			
		#OUT	7	10			7	10			8				8			
	Cell 5	#IN	9				9				9				9			
		#OUT	7				7				7				7			
	Cell 6	#IN	9				9				8				8			
		#OUT	8				8				7				7			
	Cell 7	#IN									9				9			
		#OUT									8				8			
BDD の最大幅			234	585	1930	4298	240	598	1931	3115	211	558	1647	3947	218	553	1782	3897

Publishers, 1999.

[9] T. Sasao, M. Matsuura, and Y. Iguchi, “A cascade realization of multiple-output function for reconfigurable hardware,” *International Workshop on Logic and Synthesis(IWLS01)*, Lake Tahoe, CA, June 12-15, 2001, pp. 225-230.

[10] T. Sasao and M. Matsuura, “A method to decompose multiple-output logic functions,” *41st Design Automation Conference*, San Diego, CA, USA, June 2-6, 2004, pp. 428-433.

[11] 田中克征, 吉住謙一, 中村和之, 笹尾勤, 松浦宗寛, Qin Hui, 井口幸洋, “LUT カスケードアーキテクチャによるプログラム可能 LSI の開発,” 電子情報通信学会, 第 2 種研究会・第 8 回システム LSI ワークショップ, ポスターセッション, 2004 年 11 月 30 日, pp. 315-318.

[12] S. Yu and E. E. Swartzlander, “DCT implementation with distributed arithmetic,” *IEEE Trans. on Computers*, Vol. 50, No. 9, Sept. 2001,pp. 985-991.

[13] S. A. White, “ Applications of distributed arithmetic to digital signal processing: a tutorial review,” *IEEE ASSP Magazine*, Vol. 6, No. 3, July 1989, pp. 4-19.