解　説　論　文

無線通信用ＦＦＴプロセッサのハードウェア実装

Hardware Implementation of FFT Processors for Wireless Communications

吉澤　真吾 Shingo YOSHIZAWA
宮永　喜一 Yoshikazu MIYANAGA

アブストラクト　ディジタルテレビ放送，無線 MAN/LAN，移動体通信，UWB などの広帯域ディジタル無線通信システムでは OFDM(Orthogonal Frequency Division Multiplexing，直交周波数分割多重) が採用されている．OFDM 変復調は FFT(Fast Fourier Transform，高速フーリエ変換) による演算処理が不可欠であり，最近ではギガサンプル/秒の信号処理レートに対応した FFT プロセッサが開発されている．本稿は FFT アルゴリズムの解説から始まり，パイプライン FFT プロセッサのハードウェア実装手法を述べる．

キーワード　FFT，無線通信，OFDM，パイプライン FFT プロセッサ

Abstract　Orthogonal frequency division multiplexing (OFDM) is widely adopted in wideband digital wireless communication systems such as digital television broadcast, wireless LAN/MAN, mobile communication, and Ultra-Wideband (UWB). Operation processing of fast Fourier transform (FFT) is mandatory for OFDM modulation/demodulation. FFT processors which can support Giga samples per second (Gsps) in data processing have been developed recently. This paper starts from the explanation of FFT algorithm and describes hardware implementation of pipeline FFT processors.

Key words　FFT，Wireless communication，OFDM，Pipeline FFT processor

1. はじめに

ディジタルテレビ放送，無線 MAN/LAN，移動体通信，UWB(Ultra-Wideband) などに代表される広帯域ディジタル無線通信システムは，周波数利用効率やマルチパス干渉対策の観点から OFDM(Orthogonal Frequency Division Multiplexing: 直交周波数分割多重)[(1)]が採用されている．OFDM 変復調には離散フーリエ変換 (DFT: Discrete Fourier Transform) を用いるが，計算機上で高速計算を行う高速フーリエ変換 (FFT: Fast Fourier Transform)[(2)]は OFDM を無線機に実装するのに必要不可欠である．OFDM の原理，その応用技術やシステムは本誌の 2007 年 10 月号 (第 1 巻第 2 号) の解説論文[(3)]に掲載されているのでそちらを参照して頂きたい．

最近の無線通信システムは通信速度を高速化する要求に応じて無線伝送帯域幅を拡大しつつある．IEEE802.11 無線システムでは，IEEE802.11n 規格の帯域幅は最大 40MHz であるのに対して，後継の IEEE802.11ac 規格は帯域幅を最大 160MHz に拡張し，数 Gbit/s のデータ伝送速度に到達する．Ultra-Wideband (UWB) と呼ばれる超広帯域無線では MB-OFDM(MultiBand-OFDM) 方式が規格化されているが，その帯域幅は 528MHz にも達する．これらの無線通信システムを実時間動作させるには FFT 計算において数十 GOPS(Giga Operations Per Second) の演算処理性能が要求される．FFT 計算を高速化する手法はパーソナルコンピュータ，スーパコンピュータ，GPGPU，組込み用 CPU や DSP，専用ハードウェアなど様々なプラットホーム上で研究開発が進められている．ASIC(Application Specific Integrated Circuit) や FPGA(Field Programmable Gate Array) を対象とした専用ハードウェアによる FFT プロセッサ開発では，ギガサンプル/秒の信号処理レートに達する MB-OFDM 方式対応の FFT プロセッサ[(4),(5)]が開発されている．専用ハードウェアによる FFT プロセッサは高速性のみを追求するのではなく，無線受信機回路に実装する観点から FFT プロセッサの省面積化や低消費電力化も考えていく必要がある．そのため，FFT プロセッサの稼動効率をいかに高めるという課題，すなわち，無駄なデータ処理や演算処理を抑えていくことが重要な鍵となる．稼動効率の高い FFT プロセッサは既に 1970 年代頃から研究されており，多様な回路アーキテクチャが発表されている．その中でも通信処理でデータが逐次的に送られるという状況に適したパイプライン FFT プロセッサを取り扱う．

本稿は FFT アルゴリズムやパイプライン FFT プロセッサの基本設計を解説し，OFDM 方式を実現する上での FFT プロセッサの工夫を紹介する．また，本稿で解説したパイプライン FFT プロセッサの回路ソースコードは筆者の Web サイト[(6)]で

吉澤 真吾　正員　北見工業大学工学部電気電子工学科
E-mail　yoshizawa@islab.elec.kitami-it.ac.jp
宮永 喜一　正員：フェロー　北海道大学大学院情報科学研究科メディアネットワーク専攻
E-mail　miya@ist.hokudai.ac.jp
Shingo YOSHIZAWA, Member (Department of Electrical and Electronic Engineering, Faculty of Engineering, Kitami Institute of Technology, Kitami-shi, 090-8507 Japan), Yoshikazu MIYANAGA, Fellow, Member (Division of Media and Network Technologies, Graduate School of Information and Science Technology, Hokkaido University, Sapporo-shi, 060-0814 Japan).

電子情報通信学会　基礎・境界ソサイエティ
Fundamentals Review Vol.7 No.2 pp.1–8 2013 年 10 月
©電子情報通信学会 2013

公開している．

2. FFT アルゴリズム

2.1 原　　理

まず，離散フーリエ変換を以下の式で表す．

$$X(n) = \sum_{k=0}^{N-1} x(k)e^{-j2\pi nk/N} \quad n = 0, 1, ..., N-1 \qquad (1)$$

$x(k)$ は時間領域の離散信号であり，N サンプルのブロック化した信号に対して離散フーリエ変換を行い，周波数領域の離散信号 $X(n)$ を得る．$e^{-j2\pi nk/N}$ は回転因子 (Twiddle Factor) であり，簡略表記すると，

$$W = e^{-j2\pi/N} \qquad (2)$$

$$W^{nk} = e^{-j2\pi nk/N} \qquad (3)$$

となる．N=4 としたとき，式 (1) は

$$\begin{aligned} X(0) &= x(0)W^0 + x(1)W^0 + x(2)W^0 + x(3)W^0 \\ X(1) &= x(0)W^0 + x(1)W^1 + x(2)W^2 + x(3)W^3 \\ X(2) &= x(0)W^0 + x(1)W^2 + x(2)W^4 + x(3)W^6 \\ X(3) &= x(0)W^0 + x(1)W^3 + x(2)W^6 + x(3)W^9 \end{aligned} \qquad (4)$$

と書ける．行列表現すると，

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 \\ W^0 & W^2 & W^4 & W^6 \\ W^0 & W^3 & W^6 & W^9 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} \qquad (5)$$

W^{n+N}=W^n，$W^{n+N/2}$=$-W^n$ の関係を用いて整理する．N=4 の場合は $W^4 = W^0$, $W^6 = W^2$, $W^9 = W^1$, $W^2 = -W^0$, $W^3 = -W^1$ となるので，

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & -W^0 & -W^1 \\ W^0 & -W^0 & W^0 & -W^0 \\ W^0 & -W^1 & -W^0 & W^1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} \qquad (6)$$

式 (6) の 2 行目と 3 行目を入れ替えると

$$\begin{bmatrix} X(0) \\ X(2) \\ X(1) \\ X(3) \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & -W^0 & W^0 & -W^0 \\ W^0 & W^1 & -W^0 & -W^1 \\ W^0 & -W^1 & -W^0 & W^1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} \qquad (7)$$

回転因子を要素とする行列に規則的なパターンが現れるので，それぞれ，

$$\boldsymbol{A} = \begin{bmatrix} W^0 & W^0 \\ W^0 & -W^0 \end{bmatrix} \qquad (8)$$

$$\boldsymbol{B} = \begin{bmatrix} W^0 & W^1 \\ W^0 & -W^1 \end{bmatrix} \qquad (9)$$

とする行列で置き換えると以下の式で表現できる．

$$\begin{aligned} \begin{bmatrix} X(0) \\ X(2) \\ X(1) \\ X(3) \end{bmatrix} &= \begin{bmatrix} \boldsymbol{A} & \boldsymbol{A} \\ \boldsymbol{B} & -\boldsymbol{B} \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{A} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{B} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{I} \\ \boldsymbol{I} & -\boldsymbol{I} \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{A} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{B} \end{bmatrix} \begin{bmatrix} x(0)+x(2) \\ x(1)+x(3) \\ x(0)-x(2) \\ x(1)-x(3) \end{bmatrix} \\ &= \begin{bmatrix} W^0(x(0)+x(2)) + W^0(x(1)+x(3)) \\ W^0(x(0)+x(2)) - W^0(x(1)+x(3)) \\ W^0(x(0)-x(2)) + W^1(x(1)-x(3)) \\ W^0(x(0)-x(2)) - W^1(x(1)-x(3)) \end{bmatrix} \end{aligned} \qquad (10)$$

ただし，$\boldsymbol{I}$は 2×2 の単位行列，$\boldsymbol{0}$は 2×2 の零行列である．式 (6) の計算は複素加算が 12 回，複素乗算が 16 回であるのに対して，式 (10) は重複した計算部分を除くと複素加算を 8 回，複素乗算を 12 回に減らすことができる(注1)．FFT アルゴリズムは計算順序を変更することにより生じる重複箇所の計算を省くことによって高速計算を実現していることが分かる．

2.2 実 装 方 法

式 (10) を $W^0 = 1$ の関係から整理すると，

$$\begin{bmatrix} X(0) \\ X(2) \\ X(1) \\ X(3) \end{bmatrix} = \begin{bmatrix} (x(0)+x(2)) + (x(1)+x(3)) \\ (x(0)+x(2)) - (x(1)+x(3)) \\ W^0(x(0)-x(2)) + W^1(x(1)-x(3)) \\ W^0(x(0)-x(2)) - W^1(x(1)-x(3)) \end{bmatrix} \qquad (11)$$

式 (11) の計算順序をシグナルフローグラフ (SFG: Signal Flow Graph) で表したのが図 1 である．$\boldsymbol{x} = [x(0), x(1), x(2), x(3)]^T$ から $\boldsymbol{x}_1 = [x_1(0), x_1(1), x_1(2), x_1(3)]^T$ を計算するまでの処理をステージ 1 とし，残りの処理をステージ 2 とする．図 1 の太線矢印で示す計算は $x(0)$ と $x(2)$ から $x(0)+x(2)$ と $x(0)-x(2)$ を計算する．この計算をバタフライ演算 (Butterfly Operation) と呼び，ハードウェア実装ではバタフライ演算器と複素乗算器を組合せる構成が多く見られる．FFT アルゴリズムは上記のバタフライ演算を繰り返して実行する．

次に，バタフライ演算に与える入力と出力の組合せを考え

(注1)：$W^0 = 1$, $W^1 = j$ であるので式 (6) と式 (10) の実際の演算回数は更に少なくなる．

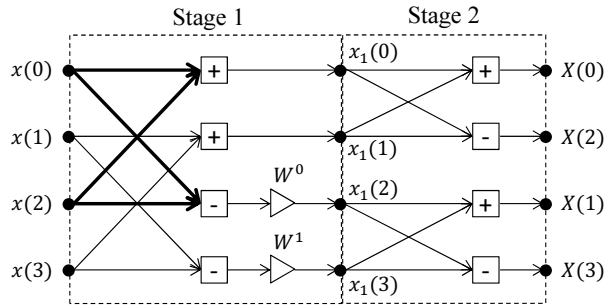


図 1　FFT シグナルフローグラフ

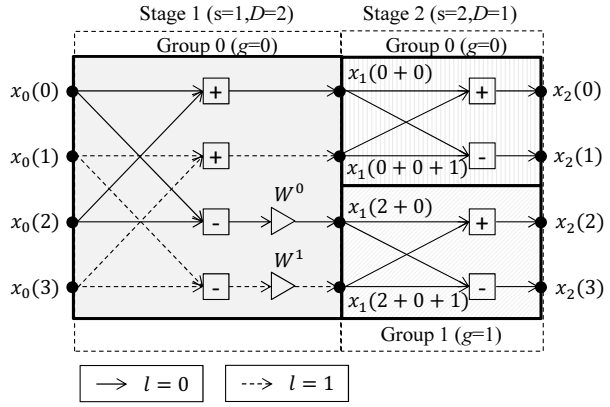


図 2　FFT シグナルフローグラフをグループ分けした場合

る．ステージ 1 の全てのバタフライ演算を一つのグループとして，ステージ数を 1 増えるごとにグループ数を 2 倍，バタフライ演算数を半分にして細分化していく．そのグループ分けしたものを図 2 に表す．ただし，ステージ 1 の入力は $\boldsymbol{x}_0 = [x_0(0), x_0(1), x_0(2), x_0(3)]^T$ と表記を変更している．バタフライ演算の入力となる二つの節点 (双対節点 (Dual Node) と呼ぶ) の間隔に注目するとステージが 1 増えるごとにその間隔は半分となる．双対節点間隔 D，ステージ番号 s，グループ番号 g，グループ内の番号 l，回転因子番号 p に対して，バタフライ演算と回転因子の乗算は以下の式で表すことができる．

$$\begin{aligned} x_s(2gD+l) &= \\ & x_{s-1}(2gD+l) + x_{s-1}(2gD+l+D) \\ x_s(2gD+l+D) &= \\ & \{x_{s-1}(2gD+l) - x_{s-1}(2gD+l+D)\}W^p \end{aligned} \quad (12)$$

各変数は以下の規則で変化する．

$$\begin{aligned} s &= 1, ..., \log_2 N \\ D &= N/2^s \\ g &= 0, ..., 2^{s-1} - 1 \\ l &= 0, ..., D-1 \\ p &= l \cdot 2^{s-1} \end{aligned} \quad (13)$$

図 2 の実線と破線の矢印はそれぞれ $l = 0, 1$ の場合であり，$x_1(2+0+1)$ は $2gD = 2$, $l = 0$, $D = 1$ を示している．図 2 と式 (12) と (13) に基づく MATLAB(若しくは Octave[7]) プログラムを図 3 に示す．MATLAB は配列の要素番号は 1 から始まるのが規則なので 7，8 行目の代入は “要素番号+1” で表していることに注意する．

```
for s=1:log2(N)
    D = N / 2^s; % 双対節点間隔
    for g=0:(2^(s-1)-1)
        for l=0:D-1
            p = l * 2^(s-1);
            W = exp((-2*pi*i)*p/N);
            X_(2*g*D+l+1)   =   X(2*g*D+l+1) + X(2*g*D+l+D+1);
            X_(2*g*D+l+D+1) = ( X(2*g*D+l+1) - X(2*g*D+l+D+1) ) * W;
        end
    end
    X = X_; % 次ステージ計算のための更新
end
```

図 3　MATLAB(Octave) プログラム

式 (11) から FFT アルゴリズム終了後にデータの並び替えが必要である．式 (5) の左辺 $\boldsymbol{X}$ と式 (11) の左辺 $\overline{\boldsymbol{X}}$ について，各要素番号を 2 進数で表すと

$$\begin{aligned} \boldsymbol{X} &= \begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} X(00) \\ X(01) \\ X(10) \\ X(11) \end{bmatrix} \\ \overline{\boldsymbol{X}} &= \begin{bmatrix} X(0) \\ X(2) \\ X(1) \\ X(3) \end{bmatrix} = \begin{bmatrix} X(00) \\ X(10) \\ X(01) \\ X(11) \end{bmatrix} \end{aligned} \quad (14)$$

となるから，$\overline{\boldsymbol{X}}$ の要素番号を 2 進数で表し，そのビット順序を逆転することにより $\boldsymbol{X}$ を得る．この処理はビット逆順 (Bit Reversal) と呼ぶ．以上説明した FFT アルゴリズムは FFT 点数 $N = 2^\gamma$ (γ は正整数) を前提とした基数 2 のアルゴリズムである．他の基数や基数を組み合わせた混合基数 (Mixed Radix) の FFT アルゴリズムは省略するので文献(8) を参照してして頂きたい．

3. パイプライン FFT プロセッサの基本設計

3.1 設計方針

前章で説明した FFT アルゴリズムを専用ハードウェアに実装する方法を述べる．最初に図 2 のシグナルフローグラフに従って演算器を直接構成する方法を提示し，その回路構成 (回路アーキテクチャ) を図 4 に示す．図の “CADD”，“CSUB”，“CMUL” はそれぞれ，複素加算器，複素減算器，複素乗算器を表し，“ROM” は回転因子データを供給する ROM(Read-Only Memory) を示す．N=4 の場合，FFT プロセッサは 4 入力，4 出力のデータ入出力を持つ．直並列変換 (Serial-Parallel (S/P))，FFT プロセッサ，並直列変換 (Parallel-Serial (P/S)) のタイミングチャートを図 5 に示す．図 5 の入力において，N サンプル

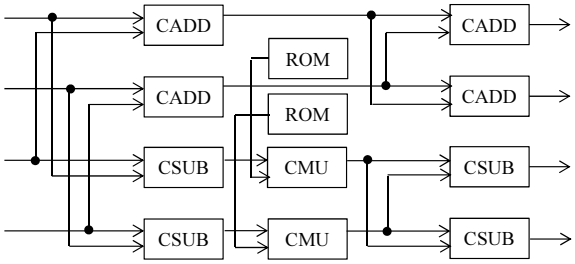
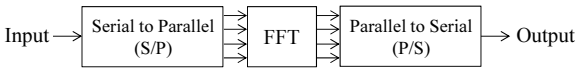


図 4　直接構成による回路アーテキクチャ



Cycle	0	1	2	3	4	5	6	7	8
S/P (Input)	$x^0(0)$	$x^0(1)$	$x^0(2)$	$x^0(3)$	$x^1(0)$	$x^1(1)$	$x^1(2)$	$x^1(3)$	$x^2(0)$
FFT				$X^0(0)$				$X^1(0)$	
				$X^0(2)$				$X^1(2)$	
				$X^0(1)$				$X^1(1)$	
				$X^0(3)$				$X^1(3)$	
P/S (Output)				$X^0(0)$	$X^0(2)$	$X^0(1)$	$X^0(3)$	$X^1(0)$	$X^1(2)$

図 5　直並列変換，FFT プロセッサ，並直列変換のタイミングチャート

の信号データを 1 個のブロックとして，連続するブロックをブロック番号 v を用いて $\boldsymbol{x}^v(v{=}1{,}2{,}...)$ と表す．図 5 の出力も同様に $\boldsymbol{X}^v$ と表す．通信用ディジタル回路は 1 クロックサイクルにつき 1 個のデータが流れるシリアルデータを扱う場合が多いため，直接構成 FFT プロセッサの前後に直並列変換と並直列変換を必要とする．タイミングチャートから FFT プロセッサは 4 サイクルのうち，3 サイクルは直並列変換の完了を待つので何も有効な動作をしていないことが分かる．全サイクルで有効な動作をしているときの稼動効率を 1 としたとき，FFT プロセッサの稼動効率は 1/4 である．すなわち，直接構成の稼動効率は $1/N$ であり，N が大きい場合には稼動効率の低下が顕著となる．

FFT 計算を高速化する目標のみならば直接構成で充分であるが，無線通信回路は装置の小形化やバッテリー駆動時間を長くするために回路規模削減や低消費電力化の工夫が求められる．無駄な処理や回路部分を少なくすること，すなわち，稼動効率をできるだけ 1 に近づけることのできる優れた回路アーキテクチャが FFT プロセッサ設計に求められる．

3.2 動作原理

パイプライン FFT プロセッサの基本となる R2SDF(Radix-2 Single-path Delay Feedback)[9] パイプライン FFT(以降，R2SDF と略す) の動作原理を説明する．R2SDF は図 6 で示すようにバタフライ演算器 (Butterfly Unit)，FIFO(First-In First-Out) メモリ，回転因子を乗算する複素乗算器で構成される．一度バタフライ演算器から出力されたデータを FIFO を経由してバタフライ演算器の入力に戻すのが特徴である．FFT 計算をステージごとに分割して並列処理を実行するのでパイプライン FFT と呼ばれる．

図 7 で示すようにクロックサイクルを経過しながら R2SDF

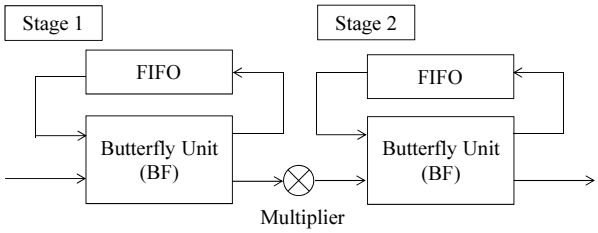


図 6　R2SDF

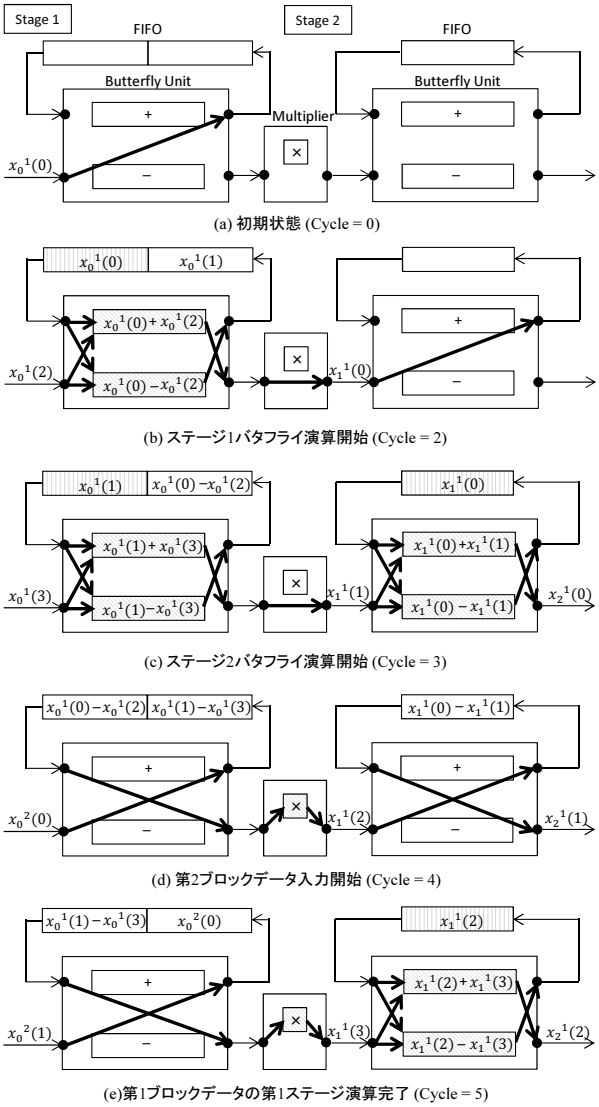
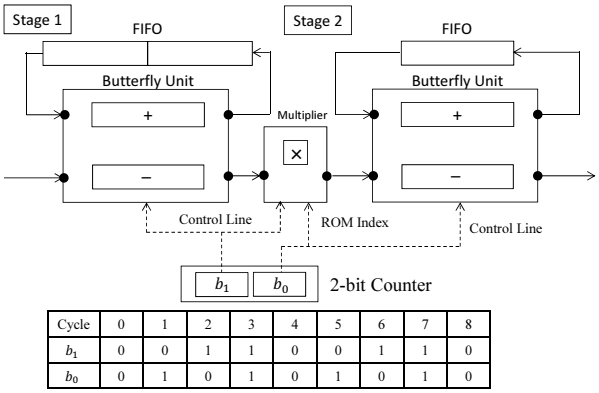


図 7　R2SDF の動作

の動作を観察してみる．ここで，$\boldsymbol{x}_s{}^v$ は図 2 のシグナルフローグラフに対応した第 v 番目ブロック，第 s ステージの途中計算データを示している．なお，$s = 0$ は FFT 入力データに対応する．図 7(a) は第 0 サイクル目の初期状態を表している．最初の FFT 入力データが $x_0{}^1(0)$ が入力され，ステージ 1 の FIFO メモリに格納される．図 7(b) の第 2 サイクル目でステージ 1 のバタフライ演算が開始する．ステージ 1 の FIFO メモリは 2 ワードで構成されており，1 サイクル経過するたびに右から左のワードにデータが移動する．ステージ 1 の入力データ $x_0{}^1(2)$ と FIFO メモリから出力される $x_0{}^1(0)$ を用いて，バタフライ演算

Cycle		0	1	2	3	4	5	6	7	8
Stage 1	Input	$x_0^1(0)$	$x_0^1(1)$	$x_0^1(2)$	$x_0^1(3)$	$x_0^2(0)$	$x_0^2(1)$	$x_0^2(2)$	$x_0^2(3)$	$x_0^3(0)$
	FIFO Output			$x_0^1(0)$	$x_0^1(1)$	$x_0^1(0)-x_0^1(2)$	$x_0^1(1)-x_0^1(3)$	$x_0^2(0)$	$x_0^2(1)$	$x_0^2(0)-x_0^2(2)$
	Butterfly			$x_0^1(0)+x_0^1(2)$ $x_0^1(0)-x_0^1(2)$	$x_0^1(1)+x_0^1(3)$ $x_0^1(1)-x_0^1(3)$	T	T	$x_0^2(0)+x_0^2(2)$ $x_0^2(0)-x_0^2(2)$	$x_0^2(1)+x_0^2(3)$ $x_0^2(1)-x_0^2(3)$	T
	Multiplier			T	T	W^0	W^1	T	T	W^0
Stage 2	Input			$x_1^1(0)$	$x_1^1(1)$	$x_1^1(2)$	$x_1^1(3)$	$x_1^2(0)$	$x_1^2(1)$	$x_1^2(2)$
	FIFO Output			T	$x_1^1(0)$	$x_1^1(0)-x_1^1(1)$	$x_1^1(2)$	$x_1^1(2)-x_1^1(3)$	$x_1^2(0)$	$x_1^2(0)-x_1^2(1)$
	Butterfly			T	$x_1^1(0)+x_1^1(1)$ $x_1^1(0)-x_1^1(1)$	T	$x_1^1(2)+x_1^1(3)$ $x_1^1(2)-x_1^1(3)$	T	$x_1^2(0)+x_1^2(1)$ $x_1^2(0)-x_1^2(1)$	T
	Output				$x_2^1(0)$	$x_2^1(1)$	$x_2^1(2)$	$x_2^1(3)$	$x_2^2(0)$	$x_2^2(1)$

図 8　R2SDF のタイミングチャート　“T” はデータの通り抜けを表す．



Cycle	0	1	2	3	4	5	6	7	8
b_1	0	0	1	1	0	0	1	1	0
b_0	0	1	0	1	0	1	0	1	0

図 9　R2SDF の制御

を行う．加算結果 $x_1{}^1(0)$ はステージ 1 から出力し，ステージ 2 の FIFO メモリに格納される．減算結果 $x_0{}^1(0)-x_0{}^1(2)$ はステージ 1 の FIFO メモリに格納される．図 7(c) の第 3 サイクル目でステージ 2 において入力 $x_1{}^1(1)$ と FIFO メモリ出力 $x_1{}^1(0)$ のバタフライ演算が開始する．このとき，FFT 計算結果 $x_2{}^1(0)$ が出力される．図 7(d) の第 4 サイクル目は第 2 ブロックの FFT 入力データ $x_0{}^2(0)$ が入力されるが，FIFO メモリに格納される処理は第 1 ブロックの場合と同様である．ステージ 1 の FIFO メモリから以前計算した減算結果 $x_0{}^1(0)-x_0{}^1(2)$ が出力されるのでこれに回転因子を乗算してステージ 2 の入力 $x_1{}^1(2)$ とする．図 7(e) の第 5 サイクル目で第 1 ブロックに対するステージ 1 の計算が終了する．6 サイクル目以降の動作はブロック番号 2 以降に対する図 7(b)〜(e) の繰返しである．

R2SDF のタイミングチャートを図 8 に示す．FFT 入力と計算結果出力のタイミングは図 5 と同じである．タイミングチャート内の “T” はデータの通り抜けであり，有効な演算を行っていない状態を表す．このことからステージ 1，ステージ 2 の稼動効率は両方とも 1/2 である．ステージ数が増えてもこの関係は変わらないので R2SDF の稼動効率は FFT 点数に関係なく常に 1/2 である．直接構成の稼動率は $1/N$ であるので，$N=4$ では R2SDF の稼動効率が 2 倍に上昇しているが，この効果は図 4 と図 7 の比較で複素演算器数が 10 から 5 と減少していることからも確認できる．また，演算処理とデータ通り抜け処理はステージ 1 は 2 サイクル，ステージ 2 は 1 サイクルおきに切

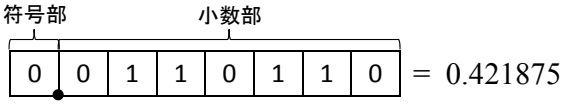


図 10　固定小数点フォーマット

り換わっているので，その切換制御は図 9 に示す 2 ビットカウンタの各桁ビット出力を用いて実現できる．

3.3　固定小数点演算

専用ハードウェアでは浮動小数点演算よりも回路規模を小さくできる固定小数点演算が利用される場合が多い．固定小数点演算は数値を表現できるダイナミックレンジが小さいため，桁あふれ (オーバフロー) による演算誤りが起きやすいことに注意しなければならない．FFT 計算は加減算と乗算のみで成り立つので図 10 に示す整数部を持たない固定小数点フォーマット (値 y に対して $|y|<1$) を用いるとオーバフロー回避が容易となる．上記の場合でオーバフローが発生するのは加減算の結果が絶対値で 1 以上になる場合のみであるので，バタフライ演算後の値を常に絶対値で 0.5 未満になるように調整する．すなわち，式 (12) において

$$\begin{aligned} x_s(2gD+l) &= \\ &\{x_{s-1}(2gD+l)+x_{s-1}(2gD+l+D)\}\cdot\frac{1}{2} \\ x_s(2gD+l+D) &= \\ &\{x_{s-1}(2gD+l)-x_{s-1}(2gD+l+D)\}\frac{W^p}{2} \end{aligned} \quad (15)$$

となるように 1/2 スケーリングを行う．ただし，1/2 スケーリングによる桁落ち (丸め誤差) が生じることも考慮する必要がある．

3.4　FIFO メモリ

FIFO メモリを LSI 上に実装するには，組込メモリ (SRAM 等) を用いる方法とシフトレジスタで代用する方法がある．FIFO メモリの記憶サイズが小さい場合は組込メモリよりも消費電力や回路面積が小さくなるシフトレジスタが優位であるが，記憶サイズが大きくなるつれてその優位度が小さくなり，ついには

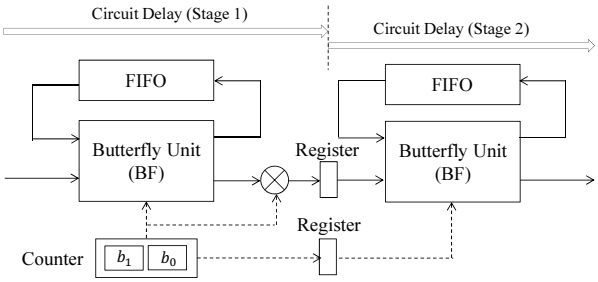


図 11　レジスタの挿入

表 1　FPGA 実装結果

FFT 点数	1024	512	256	128	64
動作周波数 (MHz)	77.4	78.6	79.1	79.7	81.0
ロジックエレメント数	6,154	4,468	3,401	2,653	2,091
ロジックレジスタ数	515	464	415	368	323
メモリサイズ (bits)	32,128	15,808	7,680	3,648	1,664
9 ビット乗算器要素数	72	64	56	48	40

逆転する．FPGA 上の実装では，FIFO メモリとして機能するシフトレジスタは組込メモリの一部として構成されるので実装方法による性能差を考慮しなくてもよい．

3.5　レジスタ挿入

FFT プロセッサのクロック動作周波数を上げるには内部演算器で発生する回路遅延に対してレジスタを挿入して分割する．図 6 にレジスタ挿入を行った場合を図 11 に示す．ステージ 1 とステージ 2 の間にレジスタを挿入することで回路遅延が分割され，その分だけ動作周波数が増加する(注2)．レジスタ挿入によりステージ 2 の処理は 1 サイクル分だけ遅れるのでカウンタ信号もレジスタを挿入してタイミングを合わせる必要がある．FFT 計算結果出力も 1 サイクル分遅れるので FFT プロセッサのレイテンシ (入力データを与えてから出力までにかかる時間) が 1 サイクル分増える．よって，図 6 のレイテンシが $N-1$ サイクルであるのに対して図 11 のレイテンシは N サイクルとなる．

3.6　回路設計と FPGA 実装結果

ハードウェア記述言語を使用して FFT プロセッサの回路設計を行う．回路設計の具体的な手順例は以下のとおりとなる．

- ディジタル回路の動作をモデル化したソフトウェアプログラムを作成し，アルゴリズムやデータフォーマット (固定小数点や整数型) の検証を行う．
- 上記のプログラムに回路シミュレーション用の入力データ (テストパターン) と照合用出力データ (期待値データ) を生成する機能を加え，回路シミュレーション用データを生成する．
- ハードウェア記述言語を使用して回路構造や動作を記述し，回路ソースコードを作成する．
- 回路ソースコードを検証するためのシミュレーション記述 (テストベンチ) を作成する．
- 回路ソースコード，テストベンチ，テストパターン，期待値データを回路シミュレータに与え，回路シミュレーションによる回路検証を行う．

FFT プロセッサ回路設計に使用したソフトウェアプログラム (MATLAB/Octave) や回路ソースコード (Verilog) を筆者の Web サイト[6]で公開しているので興味のある読者は参照して頂きたい．

表 1 に設計した回路の FPGA 実装結果を示す．FPGA デバイスは Altera CycloneIV EP4CG50CF23C6 を使用し，FPGA 合成や配置配線において Altera QuartusII 12.0sp2 を使用した．回路の動作周波数は 80 MHz 前後であり，IEEE802.11n 無線 LAN の OFDM 変復調を実時間処理するのに必要な信号処理レート (40 メガサンプル/秒) を上回る．FFT 点数によってメモリサイズが大きく変動するがこれは FIFO メモリサイズの増減に相当する．FFT 点数 1,024 に対する FPGA の回路リソース使用率が 12%程度であるので更に大きい FFT 点数の回路を実装することも可能である．

4.　OFDM 方式のための工夫

無線通信分野で OFDM 方式の実用化が進むにつれて，OFDM 方式対応の FFT プロセッサの研究開発が行われるようになった．以下，FFT プロセッサ設計における OFDM 方式のための工夫を紹介する．

4.1　回路規模の削減

Shousheng らの論文[10]ではパイプライン FFT プロセッサを回路アーキテクチャで分類しその特徴を述べているが，R2SDF の改良型として R2^{2}SDF を発表している．ある FFT ステージの複素乗算における回転因子が j 若しくは $-j$ になるように FFT アルゴリズムを変形することで，乗算のハードウェアコストがほぼ零になる (実数部と虚数部のデータ入れ替えるのみ) 改良を行っている．R2SDF は所要の複素乗算器数が $2(\log_4 N-1)$ であるのに対して，R2^{2}SDF は $\log_4 N-1$ に削減することができる．

Han らの論文[11]では複素乗算器のハードウェアコストを削減する方法として CSD(Canonical Signed-Digit) 表現を用いて乗算をシフト演算と加減算で置き換える方法が示されている．CSD 表現はある値を 2 の補数表記で表すが，例えば

$$23 = 1\cdot 2^5 + 0\cdot 2^4 + (-1)\cdot 2^3 + 0\cdot 2^2 + 0\cdot 2^1 + (-1)\cdot 2^0 \quad (16)$$

と表現する．上記の値に対する乗算は 2 進数によるシフト演算と加減算で置き換えることができる．

(注2)：クロック周期 T_s が回路遅延減少により $T_s/2$ と短くできるとき，動作周波数は $1/T_s$ から $2/T_s$ に増加する．

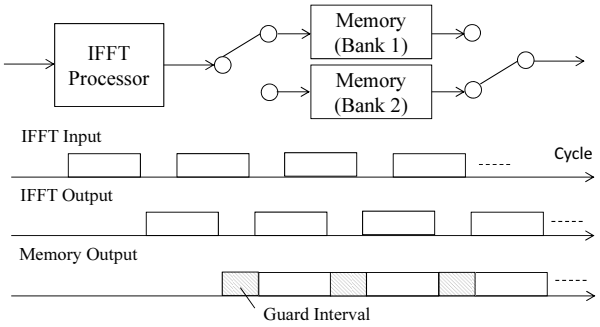


図 12　ガードインターバル処理方法

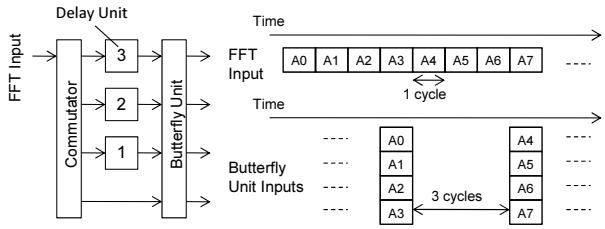


図 13　R4MDC

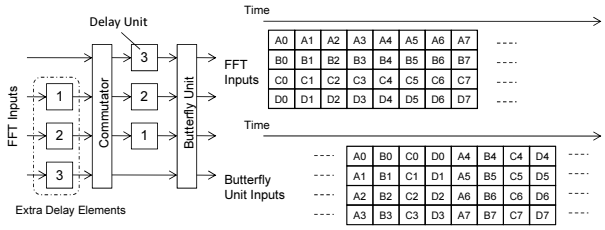


図 14　Multi-Channel MDC

4.2　ガードインターバル処理

OFDM 変調は逆離散フーリエ変換した後の時間領域信号の一部をコピーしてガードインターバルとして挿入する．図 5 や図 8 のタイミングチャートではブロックデータが連続して出力されるのでガードインターバルを挿入するためには何らかの工夫が必要となる．FFT プロセッサにガードインターバル挿入処理を対応させる工夫[12]を図 12 に示す．なお，逆高速フーリエ変換 (IFFT: Inverse FFT) は高速フーリエ変換における回転因子の符号を正負逆にするだけなので回路アーキテクチャは FFT プロセッサと同様である．IFFT プロセッサの入力はあらかじめガードインターバルを挿入することを考慮して入力待ち間隔を与える．この入力待ち間隔では IFFT プロセッサ内の FIFO メモリや制御カウンタを動作させないようにする．IFFT プロセッサの出力はブロックデータごとにメモリに記憶されるが，データの読出し前に次のブロックデータが書き込まれないようにメモリバンクを二つに分割する．このメモリ記憶処理は IFFT 計算後のビット逆順並び替え処理も兼ねている．メモリ出力ではブロックデータの一部を繰り返して読み出すことでガードインターバル挿入部分の間隔を埋めることができる．この方法は IFFT プロセッサの稼動効率が低下するがガードインターバル挿入のタイミング調整が不要であり，ガードインターバル処理対応として付加する回路が簡単になるという利点がある．

4.3　MIMO-OFDM 方式

FFT プロセッサの処理可能な信号伝送レート (スループット) を上げる手法として図 13 に示す R4MDC(Radix-4 Multi-path Delay Commutator)[13]がある．R4MDC は基数 4 の FFT アルゴリズムに基づき 4 入力 4 出力のバタフライ演算を行う．バタフライ演算器に四つのデータをそろえるため，一定のサイクル分データを遅らせる遅延器 (Delay Unit) とデータ並び替え器 (Commutator) を使用する．ただし，図 5 と同様に直並列変換を要するので稼動効率が 1/4 に低下する欠点がある．

最近の無線通信分野では MIMO(Multiple Input Multiple Output) 技術の普及が進んでいる．MIMO-OFDM 方式における FFT プロセッサは独立した複数のデータストリームを同時に処理する．Sansaloni らの論文[14]では図 14 に示すにように，R4MDC に複数データ入力と Delay Unit を追加するのみで複数のデータストリームを同時処理できる Multi-Channel MDC がスループット向上に有効であることを述べている．Multi-Channel MDC は直並列変換がデータストリーム間のデータ交換に置き換わることでバタフライ演算器の入力待ちサイクルが生ぜず，稼動効率がほぼ 1 となる長所を持つ．吉澤らの論文[15]で Multi-Channel MDC は R2SDF を 8 個並べる場合と比較して回路面積を約 1/3，消費電力を約 1/4 に削減できることを示している．

5. ま と め

本稿では割愛したがパイプライン FFT プロセッサのほかに，回路規模が小さく FFT 仕様や計算順序を柔軟に変更できるメモリ型 FFT プロセッサ[16]がある．メモリ型 FFT プロセッサは通信処理ほど高いスループットが要求されないマルチメディア処理で利用されている．昔はスーパコンピュータのみで実時間処理可能であった無線通信方式が現在ではスマートフォンや携帯電話で動作しているのはハードウェア実装分野の成果に依るところが大きい．ハードウェア実装の研究はそのような橋渡しをすることが使命であり意義であると言えるだろう．

文　献

(1) R. W. Chang and R. A. Gabby,“A theoretical study of performance of an orthogonal multiplexing data transmission scheme,” IEEE Trans. Commun., vol.16, no. 4, pp.529-540, Aug. 1968.
(2) E. O. Brigham and R. E. Morrow, “The fast Fourier transform,” IEEE Spectr., vol. 4, no. 12, pp. 63-70, Dec. 1967.
(3) 伊丹　誠, “OFDM の基礎と応用技術,” IEICE Fundamentals Review, vol. 1, no. 2, pp.35-43, Oct. 2007.
(4) Y.-W. Lin, H.-Y. Liu, and C.-Y. Lee, “A 1-GS/s FFT/IFFT processor for UWB applications,” IEEE J. Solid-State Circuits, vol.40, no.8, pp.1726-1735, Aug. 2005.
(5) S.-N. Tang, J.-W. Tsai, and T.-Y. Chang, “2.4-GS/s FFT processor for OFDM-based WPAN applications,”

IEEE Trans. Circuits Syst., vol.57, no.6, pp.451-455, June 2010.
(6) http://islab.elec.kitami-it.ac.jp/yoshizawa/fft/
(7) http://www.gnu.org/software/octave/
(8) E. Oran Brigham, 宮川 洋, 今井秀樹, (訳), “高速フーリエ変換,” 科学技術出版社, Dec. 1978.
(9) E. Wold and Alvin M. Despain, “Pipeline and parallel-pipeline FFT processors for VLSI implementations,” IEEE Trans. Comput., vol. c-33, no. 5, pp.414-426, May 1984.
(10) S. He and M.Torkelson, “A new approach to pipeline FFT processor,” IPPS 1996, pp. 766-770, April 1996.
(11) W. Han, T. Arslan, A. T. Erdogan, and M. Hasan, “Multiplier-less based parallel-pipelined FFT architectures for wireless communication applications,” ICASSP 2005, pp.45-48, March 2005.
(12) S. Yoshizawa, Y. Miyanaga, H. Ochi, Y. Itoh, N. Hataoka, B. Sai, N. Takayama, and M. Hirata, “300-Mbps OFDM baseband transceiver for wireless LAN systems,” ISCAS 2006, pp. 5455-5458, May 2006.
(13) E. E. Swartzlander, W. K. W. Young, and S. J. Joseph, “A radix-4 delay commutator for fast Fourier transform processor implementation,” IEEE J. Solid-State Circuits, vol. SC-19, no. 5, pp. 702-709, Oct. 1984.
(14) T. Sansaloni, A. Pérez-Pascual, V. Torres, and J. Valls, “Efficient pipeline FFT processors for WLAN MIMO-OFDM systems,” IEE Electronics Letters, vol. 41, no. 19, pp. 1043-1044, Sept. 2005.
(15) S. Yoshizawa and Y. Miyanaga, “Design of area- and power-efficient pipeline FFT processors for 8x8 MIMO-OFDM systems,” IEICE Trans. Fundamentals, vol.E95-A, no.2, pp.550-558, Feb. 2012.
(16) E. Cetin, R. C. S. Morling, and I. Kale, “An integrated 256-point complex FFT processor for real-time spectrum analysis and measurement,” IMTC 1997, pp. 96-101, May 1997.

（SIS 研究会提案，平成 25 年 6 月 12 日受付　7 月 8 日最終受付）

吉澤　真吾（正員）
平 13 北大・工・電子卒．平 15 同大学院工学研究科修士課程了．平 17 同大学院工学研究科博士後期課程了．平 18 同大学院情報科学研究科特任助手．平 20 同大学・情報・助教．以来，ＶＬＳＩアーキテクチャ，無線通信などの研究に従事．平 24 北見工大・工・准教授，現在に至る．博士 (工学)．平 19 年度学術奨励賞受賞．

宮永　喜一（正員：フェロー）
昭 54 北大・工・電子卒．昭 56 同大学院修士課程了．昭 59 米国イリノイ大客員研究員．昭 62 北大・工・電子講師．昭 63 同助教授．現在，北大大学院情報科学研究科教授．主として，並列信号処理，並列計算機アーキテクチャ，適応信号処理の研究に従事．工博．