社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2003－ＳＬＤＭ－111（20）
2003／10／24

オンチップマルチプロセッサシステムによる FFT並列処理に関する基礎検討

玉置　祥[†]　和田　知久[††]　神山　一弘[†††]

† 琉球大学大学院理工学研究科情報工学専攻
†† 琉球大学工学部情報工学科
††† 株式会社マグナデザインネット
E-mail: †sho@lsi.ie.u-ryukyu.ac.jp, ††wada@ie.u-ryukyu.ac.jp, †††kamiyama@magnadesignnet.com

あらまし　多種多様の通信プロトコルに柔軟に対応するために，ソフトウエアによりデジタル通信装置の動作を変更可能とするソフトウエアラジオ (SDR: Software-defined Radio) が提案されている．本研究では，リアルタイムでのデジタル通信のベースバンド処理は高い計算能力が要求されることを前提に，ソフトウエアラジオシステムのハードウエアプラットフォームとしてオンチップマルチプロセッサを想定し，C 言語を用いてシングル CPU 搭載の PC で動作するシミュレータを開発し，8K ポイント FFT を複数の仮想プロセッサで並列処理するための検討を行った．プロセッサ数による処理速度の相違について検討し，ボトルネックとなる部分の検証を行なうことで，オンチップマルチプロセッサシステムにおける FFT の高速処理に関する手法を提案する．

キーワード　オンチップ，マルチプロセッサ，並列処理，FFT

Basic examination about FFT parallel processing on On-chip multi-processor system

Sho TAMAKI[†], Tomohisa WADA[††], and Kazuhiro KAMIYAMA[†††]

† Information Engineering Course, University of the Ryukyus
†† Dept. of Information Engineering, University of the Ryukyus
††† Magna Design Net, Inc.
E-mail: †sho@lsi.ie.u-ryukyu.ac.jp, ††wada@ie.u-ryukyu.ac.jp, †††kamiyama@magnadesignnet.com

Abstract　The SDR(Software-defined Radio), which can change the operation of digital-communications equipment by software, is proposed to adapt to various communication protocols. In this research, it's assumed that on-chip multiprocessor system is used as hardware platform of software radio system since base-band processing of digital communication in real time requires high calculation capabilities. Therefore we developed the simulator which can run on single CPU computer using C language and we investigated to enable parallel processing of the 8k point FFT by two or more virtual processors. To propose a method for fast processing of FFT on on-chip multi-processor system, we inspected the differences of processing speed by number of MPU and bottle-neck of system.

Key words　On-chip, Multi-processor, Parallel processing, FFT

1. はじめに

多種多様な通信プロトコルに柔軟に対応するため，ソフトウエアによりデジタル通信装置の動作を変更可能とするソフトウエアラジオが提案されている．本研究では，デジタル通信におけるベースバンド処理に，オンチップマルチプロセッサを使用する際の実装方法について検討を行った．検討材料として，地上波デジタル放送やワイヤレス LAN などの大容量デジタル通信で採用されている直交周波数多重変調 (OFDM: Orthogonal Frequency Division Multiplexing) の変復調処理である FFT を想定し，実際に並列処理を行なうことによる高速化手法についての提案を行なう．

2. オンチップマルチプロセッサシステム

オンチップマルチプロセッサシステムは，複数のプロセッサをひとつのシリコンチップ上に配置して，協調・並列動作させることで，処理の高性能化を行わせることができる．また同時に，システムの低コスト化および小型化を実現することができる．図1に，マルチプロセッサシステムの構成図を示す．

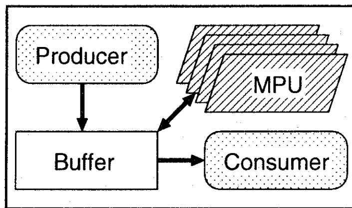


図1 マルチプロセッサシステムの構成図

2.1 構成と機能

このマルチプロセッサシステムの基本構成要素は，Producer, MPU, Consumer そして バッファメモリ (以後バッファと示す) の4つである．

ここでは，Procuder は外部システムからのデータ入力機能であり，Consumer は外部システムへのデータ出力機能をもつものである．MPU は，複数搭載されており，図1では4つの MPU を載せた場合が示されている．また，MPU は同一のものである必要はなく，例えば RISC プロセッサと DSP プロセッサの2種類を載せることも可能である．これらの要素間にバッファが配置されており，要素間のデータ転送を行う．上記に述べた機能により Producer と Consumer のバッファへのアクセスは，書き込みのみと読み込みのみとなっている．

4つの要素を協調動作させて，複数の MPU による並列実行を可能にするため，バッファをのぞいた各要素には，次に述べる2つの機能が備えられている．

a) Notify

バッファに，データが書き込まれたことを後続の要素に伝える．例えば，Producer が バッファに必要なデータをすべて書き込み終わった時に，MPU に対して Notify を出す．MPU は，Notify を受けてから動作を始める．

b) Release

バッファから，データをすべて読み込み，空き状態であることを他の要素に伝える．例えば，MPU がバッファのデータをすべて読み込み終わって，バッファが必要なくなった時に，Producer に対して Release を出す．Producer は，MPU からの Release を受けることで，バッファへの書き込みができるようになる．

2.2 動作の流れ

複数の MPU を並列動作させて，正常な処理を行わせるには，Notify と Release の2つの機能を使い，図2のフローチャートに示すような，動作の流れを作ることが必要である．

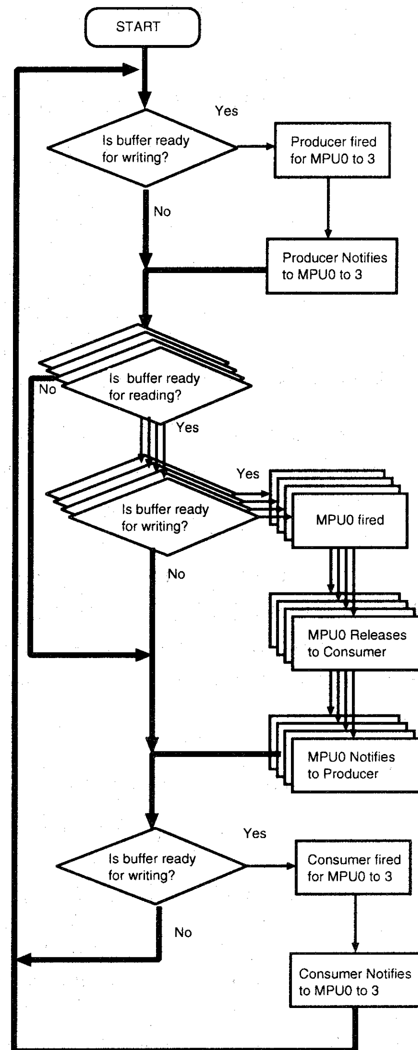


図2 フローチャート

まず，Producer はバッファへの書き込みが可能かを判断し，可能であるならば，バッファへのデータ書き込みを開始する．これは，MPUx(x は MPU 番号) からの Release を受けて動作することに相当し，システム起動直後などの初期状態においては，MPU からの Release は発行済とする．バッファへの書き込みが完了したら，そのデータを処理する MPU に対して Notify を出す．

つぎに，Notify を受けた MPU は，処理結果を書き込む領域がバッファに存在するかを確認する．これは，Consumer からの Release を受けたかどうかの確認である．バッファへの書き込みが可能であれば，処理を開始して，Producer からのデータをすべて読み込み終えて必要なくなれば，Release を発行する．処理を終えて，結果をバッファに格納終了したら Consumer に対して Notify を出す．

Consumer は，すべての MPU からの Notify を確認してか

ら処理を開始する．すべての処理を終えてから，MPU に対する Release を発行し，一連の動作の流れが完了する．

3. FFT の並列処理

通常の，並列処理ではない FFT と比べ，並列処理による FFT では，その実行時間は小さい．表 1 に，通常の 8192 点 FFT(以降 8kFFT と略す) と，2048 点 FFT(同じく 2kFFT と略す) を 4 つ並列処理して 8kFFT と等価な処理を行なった場合の MPU1 つあたりのバタフライ演算量を示す．表 1 から，単純にバタフライ演算量を比べる限り，2kFFT を 4 つ並列処理 (2kFFTx4 と略す) させた場合，MPU1 つあたりの演算量が減少するため，並列処理したとき、結果的に実行時間が短縮されて高速処理できることがわかる．

表 1 8kFFT と 2kFFTx4 のバタフライ演算量

	Radix-4 バタフライ	Radix-2 バタフライ	合計
8kFFT	10,240	4,096	14,336
2kFFTx4	4,608	1,024	5,632

3.1 8kFFT の並列処理

例として，8kFFT を並列処理する場合，単純に 2kFFT を 4 回行うだけでは，正しい結果は得られず，Producer または Consumer において特殊な処理が必要となる．まず，Producer で行う特殊処理を PreFFT とすると，例えば 2kFFTx4 のシステムでは，PreFFT として Radix-4 の 8kFFT を 1 回行う必要がある．つぎに，Consumer で行う処理を PostFFT とすると，同じく 2kFFTx4 のシステムでは，Radix-4 の 4 点 FFT を 2048 回行なう必要がある．1kFFTx8 では，Radix-8 の FFT を行なうことになる．512FFTx16 では，さらに複雑となり，Radix-16 となる．並列化をすすめて， MPU 数を増やした場合の PreFFT を表 2 に，PostFFT を表 3 に示す．

表 2 並列化数による PreFFT の違い

並列化数	PostFFT の内容
2	Radix-2 8kFFT 1 回
4	Radix-4 8kFFT 1 回
8	Radix-4 8kFFT 1 回 + Radix-2 2kFFT 1 回
16	Radix-4 8kFFT 1 回 + Radix-4 2kFFT 1 回

表 3 並列化数による PostFFT の違い

並列化数	PostFFT の内容
2	Radix-2 2 点 FFT を 4092 回
4	Radix-4 4 点 FFT を 2048 回
8	Radix-8 8 点 FFT を 1024 回
16	Radix-16 16 点 FFT を 512 回

3.2 2kFFTx4 のシステム

図 1 に，マルチプロセッサシステムの構成図を示したが，FFT を実装するにあたって，データフローの観点からブロック図を示すと図 3 のようになり，データと処理の流れは，以下のとおりである．

- Producer が必要なデータをバッファに書き込み
- 4 個の MPUx が 並列に 2kFFT を行なう
- 結果をバッファに書き込み
- Consumer が PostFFT を行なう

図 4(a) に，PreFFT や PostFFT をのぞいたタイミングチャートを示す．図 4(b) は，MPU を 8 つ使って 1024 点 FFT(以降 1kFFT) を 8 個にしたときのタイミングチャートである．また，凡例を図 4(c) に示す．各タイミングチャートは，上から Producer, MPU0,1,2 のチャートとなっている．

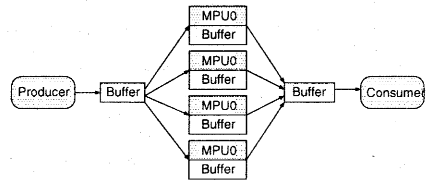
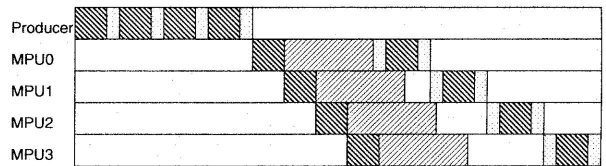
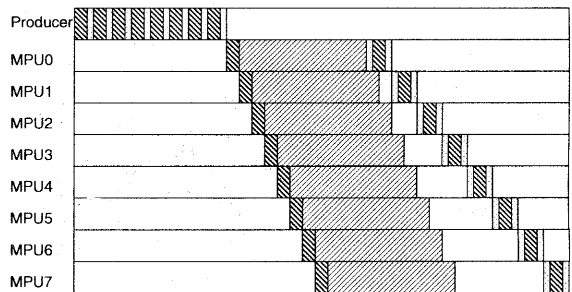


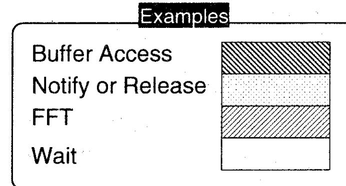
図 3 2kFFTx4 のブロック図



(a). 2kFFTx4 のタイミングチャート



(b). 1kFFTx8 のタイミングチャート



(c). 凡例

図 4 タイミングチャートと凡例

3.3 問 題 点

前述した，図 3 に示すシステムでは，Producer 側に 1 つ，Consumer 側に 1 つ，さらに MPU の作業用に 4 つのバッファメモリが必要であり、合計 6 つのバッファが必要である．またバッファは 1 ポートを想定していて、読み込みと書き込みを同

時に行えないため、Producer 側である MPU がデータの読み込みを行なっている間は，ほかの MPU は待機状態となる．同様に Consumer 側でも，同時に 1 つの MPU しかバッファに書き込めないため待機時間が多く発生する．このことは，図 4(a) と図 4(b) のタイミングチャートで明らかであり，大きな問題点である．

3.4 改良案の提案

問題点を解決するために，図 5 に示すようなシステムを提案する．これは，各 MPU の前後に 1 つずつの合計 8 枚のバッファを配した構造となっている．Producer は，MPU 1 つ分のデータを書き込むごとに Notify を出すため，各 MPU は自身宛の Notify を受け次第に処理を開始できる．また，処理結果を出力する際も，各 MPU は独立してバッファに書き込むことができる．このことで，MPU が FFT 処理を行なったあと，Consumer 側バッファへのデータ書き込みを行なう際に待機する必要がなくなる． MPU の作業用バッファとしては，Producer 側のバッファを使用することが可能であり、トータルのバッファ数の増加は 6 から 8 で少なく、メモリ容量的には (Q15 フォーマットで 48K バイトから 64K バイトへ) 節約することが可能である．

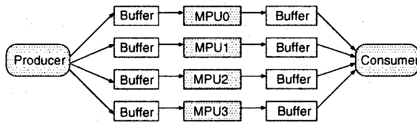


図 5 2kFFTx4 のブロック図 (提案システム)

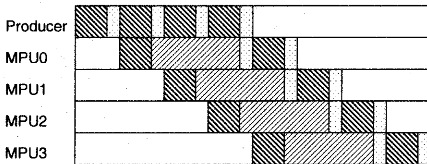


図 6 2kFFTx4 のタイミングチャート (提案システム)

4. シミュレーション

4.1 シミュレータ

本研究では，C 言語を用いて，シングル CPU での動作が可能なシミュレータを作成した．このシミュレータは，われわれが提案した 図 3 や図 5 に示すシステムの動作を実現するものである．ただし，実機やエミュレータと違い，本シミュレータには，いくつかの制限がある．

4.2 実行サイクル数の推定

本研究で提案しているマルチプロセッサシステムでは，表 4 に示すように，加減算や乗算などの FFT で使用する演算，Notify と Release にかかるオーバーヘッドやバッファアクセスといった，基本的な作業別の実行サイクル数を仮定した．

表 4 作業別単位実行サイクル数

	実行サイクル
加算	1
減算	1
乗算	1
Notify	100
Release	100
MPU バッファへの書き込み (Producer)	8
MPU バッファへの書き込み (MPU 作業用)	1
MPU バッファの読み込み (MPU 作業用)	1
Consumer バッファへの書き込み (MPU)	8
Consumer バッファの読み込み (Consumer)	1

4.3 実行速度

実行速度は，実行サイクル数の逆数で求めることができる．そこで，表 4 から改良前後での実行速度を求めた．

まず，図 7 に MPU 数ごとの FFT 処理速度のグラフを示す．ここでいう FFT 処理速度とは，MPU0 の FFT 処理開始から，MPU3 の FFT 処理終了までのサイクル数の逆数をいうが，図 7 のグラフでは，MPU 数に比例して FFT 処理速度は向上している．特に，4MPU から 8MPU での速度向上は著しく，2MPU から 4MPU でおよそ 1.8 倍の向上に比べ，2.2 倍となっている．16MPU においても速度低下はみられず，8MPU の約 1.5 倍の実行速度となっている．FFT 処理に関しては，MPU 数を増やすほど実行サイクル数は減少し，実行速度があがるといえる．なお，FFT の実行速度は，改良前後で変わらない．

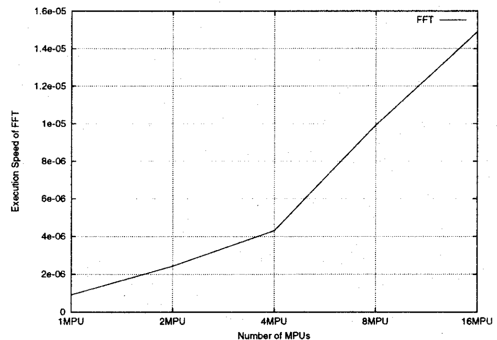


図 7 FFT の実行速度

次に，バッファアクセスや本研究で提案している Notify, Release などの機能を合わせたオーバヘッドについて，図 8 に実行速度のグラフを示す．比較のため，改良前後のシステムにおける実行速度のグラフを示す．MPU 数が増えるにつれて，やや鈍化はするが順調に実行速度は上がっているが，このまま MPU 数を増加させれば，やがて実行速度が頭打ちになることが予想できる．これは，バッファの数が増えることで，メモリ関係のオーバーヘッドが増加するためと思われる．

図 9 には，PreFFT を使用したシステムの全体的な実行速度のグラフを示し，図 10 に，PostFFT を使用したものを示す．両者ともに，4MPU までは実行速度をあげているが，図 10 の PreFFT システムでは，8MPU で 4MPU の実行速度を下回っ

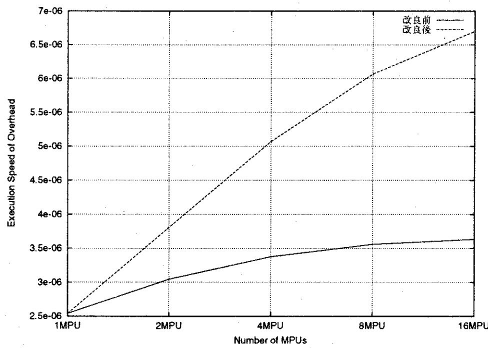


図 8　オーバーヘッドの実行速度

ている．この差は，図 11 の PreFFT と PostFFT の実行速度の違いからわかる．PreFFT は，2MPU の時点で PostFFT よりかなり高速であるが，4MPU で追い付かれ，8MPU 以降では逆に遅くなっている．つまり，MPU 数の増加により，FFT 処理やオーバーヘッドについては速度向上がみられたが，8MPU 時点において PreFFT の処理時間が FFT とオーバーヘッドの速度向上分を上回り，システム全体の速度低下を招いたということになる．

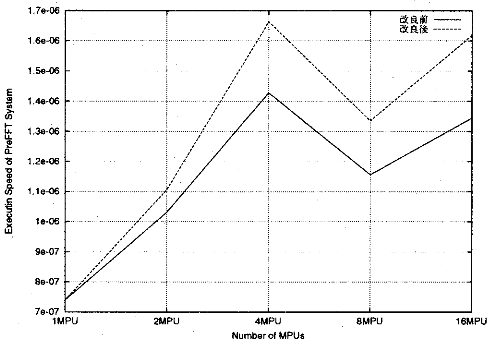


図 9　PreFFT を使用したシステム全体の実行速度

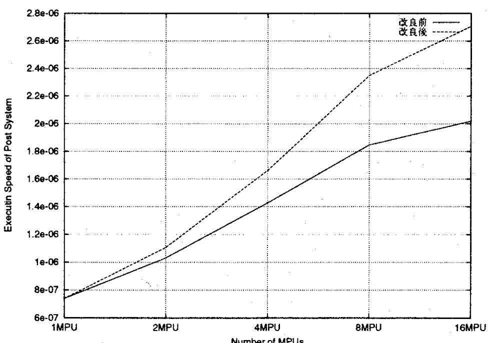


図 10　PostFFT を使用したシステム全体の実行速度

4.4　結　　果

ここで，PreFFT システムと PostFFT システムの実行速度を比較するため，改良後の両システムの実行速度を図 12 に示す．

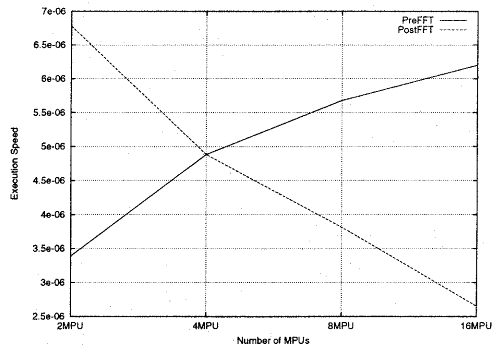


図 11　PreFFT と PostFFT の実行速度

4MPU までは，実行速度に差はないが，8MPU で PreFFT システムの実行速度が大幅に落ち，16MPU でも差が開いたままとなっている．よって，8kFFT を並列処理する場合，8MPU 以上では PostFFT システムの方が高速である．表 5 に，PreFFT を使用したシステムと PostFFT を使用したシステムの改良後の改善率を示すが，8MPU 以上では PostFFT システムが優れているのがわかる．

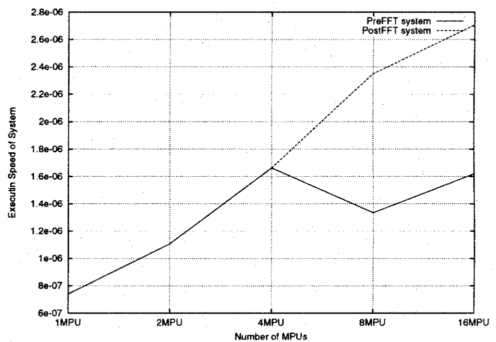


図 12　PreFFT と PostFFT の実行速度比較

表 5　PreFFT と PostFFT を使用したシステム全体の改善率

MPU 数	PreFFT (%)	PostFFT (%)
1	0.0	0.0
2	6.78063	6.78063
4	14.1243	14.1243
8	13.4187	21.4369
16	16.9273	25.4133

5.　ま　と　め

本研究では，オンチップマルチプロセッサシステムでの，FFT 並列処理に関する検討を行った．その結果，マルチプロセッサシステムのハードウェア的な構成としては，Producer, MPU, Consumer の各々の要素にバッファを配置するするものに比べ，MPU の入力側と出力側に 1 枚ずつ配する方が効率的であり，タイミングチャートと実行サイクル数の推定による検証で，表 6 に示すように待機時間に大幅な改善がみられた．

表 6　待機時間の改善率

MPU 数	改善率 (%)
2	20.0
4	33.4
8	41.3
16	45.8

また，8kFFT を行う場合について，MPU で FFT した結果に対して処理を行う PostFFT システムを採用することで，高速な処理を行えることを確認した．ただし，16MPU で Radix-16 が必要になるなど，アルゴリズムが複雑化するため，開発の容易性も考慮した検討が必要になる．

今後の課題としては，バッファの数と実行時間のトレードオフについてのさらなる検証を行い，MPU 数が増えることによるオーバーヘッド処理時間短縮の頭打ちについての打開策を検討することや，8MPU 以上における PreFFT の処理時間短縮と 16MPU 以上での PostFFT アルゴリズム検討などが考えられる．

また，シミュレータを作成して，実際に 8kFFT を並列処理することに成功しており，検討結果の信憑性を高めることができた．今後，実機のオンラインマルチプロセッサシステムによる開発が可能になれば，FFT を並列処理させて，今回の検討結果を実証したい．

文　献

[1] 前原崇章 (2002)「2 バンク RAM を用いた高速フーリエ変換回路の設計」琉球大学大学院理工学研究科情報工学専攻

[2] Thorsten Grötker, Stan Liao, Grant Martin, Stuart Swan(2002) "System Design with SystemC" Kluwer Academic Pub

[3] 平田富夫 (1998)「アルゴリズムとデータ構造」森北出版

[4] 三上直樹 (2000)「ディジタル信号処理の基礎」CQ 出版

[5] 「FFT 概略」URL: <http://momonga.t.u-tokyo.ac.jp/~ooura/fftman/ftmn1_23.html>