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

2008－SLDM－134／2008－EMB－8 (15)
2008 / 3 / 27

SDRプロセッサを対象としたソフトウェア自動最適化手法の提案

池田 裕司† 山崎 浩輔† 前山 利幸† 武内 良男†

† 株式会社 KDDI 研究所
〒 356-8502 埼玉県ふじみ野市大原 2-1-15
E-mail: †yj-ikeda@kddilabs.jp

あらまし 無線通信をソフトウェアで実現する SDR (Software Defined Radio) 技術は，ソフトウェアの更新で通信方式の変更を可能とする利点を持つ．しかし，無線通信ではリアルタイム処理の要求が厳しく，SDR プロセッサごとに無線通信方式のソフトウェアを最適化する必要がある．現状，ソフトウェアの最適化は手動で行われているが，手動による最適化は開発期間を必要とする上，最適化したソフトウェアは SDR プロセッサに強く依存し，他の SDR プロセッサには再利用できないという問題点がある．この問題点を解決すべく，本稿では，SDR プロセッサのハードウェアの特徴を活用した最適化を自動実行する手法について提案する．本手法は，最適化対象のソースコードに対し，SDR プロセッサの特徴を考慮した最適化の設計に必要なソースコードの解析情報を付加することを特徴とする．更に，本手法の有効性を示すべく，本手法に基づき動作するツールを試作し，性能を評価した．その結果，試作したツールを用いた最適化によって約 69% の処理時間削減効果が得られた．

キーワード SDR，ソフトウェア，最適化，SDR プロセッサ

A Proposal of the Automatic Software Optimization Method for SDR Processor

Yuji IKEDA†, Kosuke YAMAZAKI†, Toshiyuki MAEYAMA†, and Yoshio TAKEUCHI†

† KDDI R&D Laboratories Inc
2-1-15 Ohara, Fujimino-shi, Saitama,356-8502 Japan
E-mail: †yj-ikeda@kddilabs.jp

Abstract Software Defined Radio (SDR) technology has the advantage that it can realize the wireless communication system by software. However, to realize the real-time processing of the wireless communication system, the software should be optimized for SDR processor. In general, the software is optimized by the software developer. However, the optimization by the software developer needs much time, and moreover, the optimized software does not work well on the other SDR processor. So far we propose a method to optimize the software automatically. In the proposed method, the analytic information is added to the target software. Moreover, we make a prototype based on the proposed method and evaluate the performance. The result shows about 69% of the calculation time is reduced using the prototype.

Key words SDR, software, optimization, SDR processor

1. まえがき

近年，ソフトウェア無線 (Software Defined Radio) 技術が注目を集めている．SDR 技術は，変調方式や利用する周波数帯域の異なる複数の無線通信方式を，ソフトウェアのみで実現する技術である．SDR 技術により，ソフトウェアを更新するだけで新しい無線通信システムに対応することができ，新システム導入時におけるコスト削減が可能となる．

一般的に無線通信においては，リアルタイム処理に対する要求が厳しく，CDMA2000 1x EV-DO Rev.0 では，受信データの 1 単位である 1slot の復調処理を 1.67ms 以内に終了しなければならない．本要求を満たすには，積和演算を得意とする DSP (Digital Signal Processor) や，特定の演算を高速に処理可能なアクセラレータ，ソフトウェア開発者により自由にカスタマイズ可能な PLD (Programable Logic Device) などを組み込んだ SDR プロセッサの利用が効果的である．しかしながら，SDR プロセッサは各々が非常に独特かつ複雑なアーキテクチャを有しているため，その性能を最大限引き出すためには，ソフ

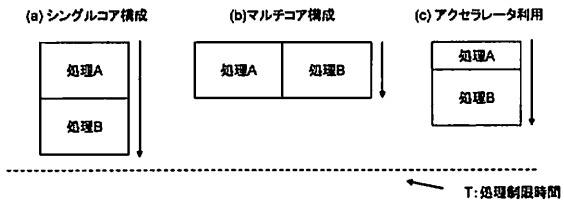

図 1　SDR プロセッサの特徴を活用した最適化の例

トウェアを SDR プロセッサのハードウェアの特徴に合わせて最適化する必要がある．現状，ソフトウェアの最適化は手動により行われているが，最適化には開発期間を必要とする上，最適化したソフトウェアは対象とした SDR プロセッサに強く依存し，他のプロセッサを利用する場合にはソフトウェアをソースコードから再設計しなければならない．

以上の点を鑑み，本稿では，SDR プロセッサのハードウェアの特徴を活用した最適化を自動実行する手法について提案する．提案手法は，C 言語などの高級言語で記述された最適化対象ソースコードに対して，最適化の設計に必要となるソースコード解析情報を予め付加することを特徴とする．また，提案手法に基づき動作するソースコード自動最適化ツールを試作し，性能を評価した．

以下，2 章において，SDR プロセッサを対象としたソフトウェア最適化について述べる．また，3 章において，提案するソフトウェア自動最適化手法の構成について述べる．更に，4 章において，本アーキテクチャに基づき動作するツールを試作し，性能を評価した結果を述べる．最後に，5 章において，まとめを述べる．

2. SDR プロセッサを対象としたソフトウェア最適化

2.1 SDR プロセッサの特徴を活用した最適化

無線通信では，リアルタイム処理の要求が厳しく，制限時間内に処理を終了させる必要がある．この厳しい要求を満たすためには，SDR プロセッサの利用が効果的である．

図 1 に SDR プロセッサの特徴を活用した最適化の例を示す．なお，処理 A と処理 B から構成される演算処理を実行させたものとする．同図 (a) は，シングルコア構成の SDR プロセッサを利用した場合を示したものである．シングルコア構成の場合，制限時間内に処理を終了させるために，演算コアの動作周波数は高速であることが望ましい．しかしながら，動作周波数の増加は消費電力の増大を招く．携帯電話のように容量が小さい場合，消費電力の増大は好ましくない．

同図 (b) は，マルチコア構成の SDR プロセッサを利用した場合を示したものである．マルチコア構成を利用し，処理を並列化させることで，制限時間内に処理を終了させることが可能となる．なお，マルチコア構成を活用する場合には，各コアに割り当てる演算処理の処理量や，各演算処理の実行順序を正確に把握し，適切に処理を割り振る必要がある．しかしながら，

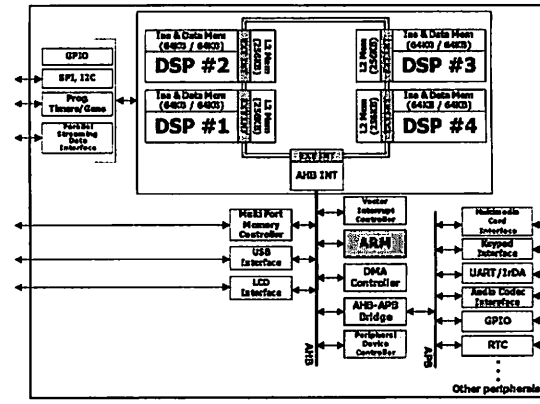

図 2　SB3010 のプロセッサ構成

無線通信で用いられる誤り訂正方式の一つで，復号処理を複数回繰り返すことにより性能改善を図るターボ復号は，N 回目の復号処理では $(N-1)$ 回目の出力結果を利用する．このような再帰的な演算では，処理の並列化は困難である．

同図 (c) は，処理 A 用のアクセラレータを有する SDR プロセッサを利用した場合を示したものである．アクセラレータを利用し，処理 A をアクセラレータに括り出すことで，処理時間を制限時間内に収めることが可能となる．無線通信では，FFT 演算のように，利用頻度が高いにも関わらず，処理量が非常に大きい演算が存在するため，専用アクセラレータを SDR プロセッサに搭載することは効果的である．しかしながら，専用アクセラレータを多数用いた場合，SDR ならではの柔軟性が損なわれてしまう．

以上述べたように，SDR プロセッサごとの特徴を考慮し，ソフトウェア最適化を行う必要がある．

2.2 最適化実施例

筆者らは，Sandbridge 社製 SDR プロセッサ SB3010 [4] を用いて，CDMA2000 1x EV-DO Rev.0 の端末受信機能を実装し，リアルタイム動作を確認した [2], [3]．SB3010 のプロセッサ構成を図 2 に示す．また，SB3010 の特徴の一例を以下に示す．

- CPU 内部に 4 つの DSP コアを有し，単方向のリングバスによって各コアが接続されている．
- 各コアは内部に 8 つのスレッドを有し，並列処理が可能
- 64KB の L1 データメモリ，256KB の L2 データメモリを各コアごとに搭載
- 外部に RAM，SDRAM など複数のメモリを搭載

本実装では，リアルタイム動作を実現するために，SB3010 のハードウェア上の特徴に合わせてソフトウェアの最適化を実施した．

まず，各コアにおける処理量が出来るだけ均一になるように演算処理の振り分けを行った．その後，メモリ割り当ての最適化を行った．メモリのアクセスに必要な時間は，L1 データメモリ，L2 データメモリ，外部メモリの順に短い．このため，変数や関数のアクセス頻度の解析を行い，アクセス頻度の高い変数や関数のメモリを L1 データメモリに優先的に配置した．ま

た，コア間のポーリング削減も併せて実行した．開発したソフトウェアでは，コア間での演算処理の同期を取るべくポーリングを実行する際に，メモリアクセスを頻繁に行うため，遅延が発生する．このため，ソフトウェアを再構成し，各コアの待ち時間削減を行った．

現状，このようなソフトウェア最適化は，ソフトウェア開発者によって手動で行われる．しかしながら，ソフトウェア最適化には多大な開発期間を要するだけでなく，最適化されたソフトウェアは対象のSDRプロセッサに強く依存し，他のSDRプロセッサに再利用できない問題点がある．

2.3 ソフトウェア自動最適化の問題点

手動による最適化の問題点を解決する手段として，ソフトウェアの最適化を自動的に実行するツールの開発がある[7]．自動最適化ツールは，SDRプロセッサのハードウェアアーキテクチャや制約の情報を外部から与えることで，SDRプロセッサの特徴を活用した形式にソフトウェアを自動最適化する．

一方，前節で述べたソフトウェア最適化の設計を行う上で，各演算処理の処理量や，処理全体の構成などの情報をソースコードから摘出することは必要不可欠である．ソフトウェア開発者が手動で最適化を実行する場合は，SDRプロセッサの特徴とソースコードを照らし合わせ，最適化に必要な情報をソースコードから摘出する．必要な情報の摘出には，ソフトウェア開発者の経験や判断が不可欠であり，ツールが自動的に摘出を行うことは非常に困難である．従って，SDRプロセッサの特徴を活用したソフトウェアの最適化を完全に自動化することはできない．

3. ソフトウェア自動最適化手法

ソフトウェア最適化を自動化する上での問題点を解決すべく，図3に示す手法を提案する．本手法は，C言語などの高級言語で記述されたソースコード（以下，汎用ソースコード）に対して，ソースコードの解析情報を付加することを特徴とする．ソースコード解析情報は，SDRプロセッサの特徴を活用した最適化の設計に必要となる情報である．本情報は，ソフトウェア開発者が予めソースコードを解析し，手動で与える．

ソースコード自動最適化ツールは，ソースコードに付加された解析情報と，対象とするSDRプロセッサのハードウェアアーキテクチャや制約などの特徴情報（以下，ハードウェア構成情報）を照らし合わせ，最適化を自動的に実行する．

提案手法により，最適化におけるソフトウェア開発者の作業は，ソースコード解析情報の付加のみとなる．解析情報の付加に必要な作業量は，手動でソフトウェアの最適化を実行する場合と比較して少なく，最適化に必要な期間を短縮することが可能となる．以下に，ソースコード解析情報とハードウェア構成情報の詳細を示す．

(i) ソースコード解析情報

以下にソースコード解析情報の一例を示す．

- 最適化対象となる演算処理単位の情報
- 演算処理の内容情報
- 演算処理の全体構成情報

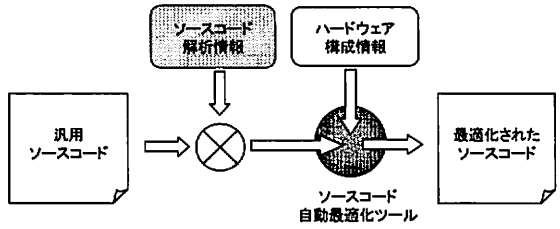

図3 提案手法の構成

- 変数の定義情報
- 演算処理の入出力変数情報
- 繰り返し回数情報

これらの情報を与えることで，以下に挙げる項目が可能となる．

- 最適化を適用すべき演算や，各コアに振り分ける演算処理単位の自動検出
- 専用アクセラレータの利用の可否
- 演算処理の全体構成を考慮した最適化の設計
- SDRプロセッサごとに異なる記述の独自変数の利用
- 入出力のポート数やバス幅，内部メモリ量を考慮した最適化の設計
- 演算処理の実行回数を考慮した最適化の設計

(ii) ハードウェア構成情報

以下に，ハードウェア構成情報の一例を示す．

- 独自変数型の情報
- アクセラレータの有無の情報
- コア構成情報
- バス幅，ポート数，内部メモリ量等のハードウェアの制約情報

4. 試作したツールによる提案手法の評価

提案手法に基づき動作するソースコード自動最適化ツールを試作し，評価を行った．なお，試作したツールではハードウェア構成情報をツール内部に予め内蔵しており，対象プロセッサ専用のツールとして動作する．

4.1 対象プロセッサ

本稿では，米国Stretch社製S5530を検討対象とする[9]．図4に，S5530のプロセッサ構成を示す．同図に示すように，S5530は，Instruction Set Extension Fabric（以下，ISEF）と呼ばれるPLDを有する．ISEFは，以下に挙げる特徴を有する．

- ユーザが定義可能な拡張命令ユニットが使用可能
- 128bitの独自変数WR (Wide Register) へ複数の変数をパック可能

複数の命令を拡張命令化し，単一の命令とすることで，処理時間を削減することが可能となる．また，WRを利用し，複数変数をパックすることで，処理の並列化が可能となる．例えば，ISEFへの入出力変数がshort型 (16bit) の場合，8個の変数をパックし，ISEFに渡すことで，8並列処理を行うことが可能となる．特に，上述した並列化による処理時間削減効果は大き

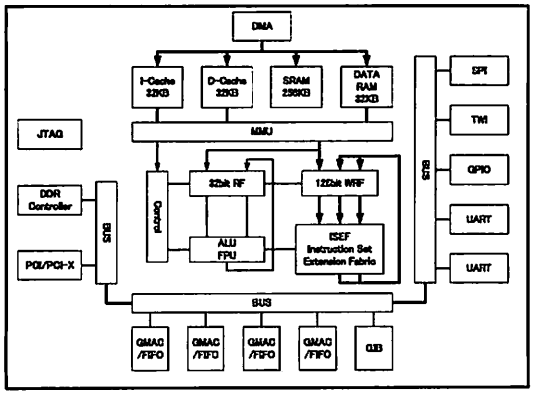

図 4　S5530 のプロセッサ構成

く，繰り返し回数の多い演算処理を ISEF に実行させることは非常に効果的である．従って，ISEF を効果的に利用するためには，繰り返し回数が多い演算処理の選出が必要である．

4.2　試作ツールを用いた自動最適化の構成

図 5 に，試作したツールを用いた自動最適化の構成を示す．まず，汎用ソースコードに対して，ソースコード解析情報を手動にて付加する．その後，ソースコード解析情報を付加した汎用ソースコード（以下，解析情報付加コード）をツールに入力し，S5530 用に最適化されたソースコードを得る．なお，S5530 用に最適化されたソースコードは，拡張命令を定義するコード（以下，拡張命令定義コード）と，拡張命令を呼び出し，演算処理を行うコード（以下，拡張命令呼び出しコード）により構成される．

4.3　解析情報付加コードの作成

C 言語で記述された汎用ソースコードに対して，ソースコード解析情報を付加することで，解析情報付加コードを作成する．図 6 に，解析情報付加コードの例を示す．図中太字が，付加したソースコード解析情報である．同図に示すように，各変数や演算処理に対して，変数の利用目的や演算処理の位置などを明示するタグを手動にて付加する．以下に，汎用ソースコードに付加するソースコード解析情報の例を示す．

- start：演算処理の開始
- alloc：演算に利用する変数
- init：変数の初期化
- loop-init：ループの開始条件
- loop-condition：ループの終了条件
- loop-renew：繰り返し時に実行される処理
- loop-end：ループの終了
- calc-input：演算の入力，calc-output：演算の出力
- calc-exe：演算処理
- end：演算処理の終了

これらの情報を与えることで，以下に挙げる項目が可能となる．

- 「start」および「end」のタグによって，最適化の対象範囲が検出可能となる．

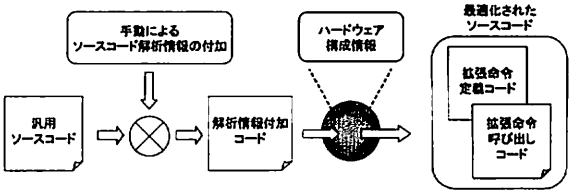

図 5　試作ツールを用いた自動最適化の構成

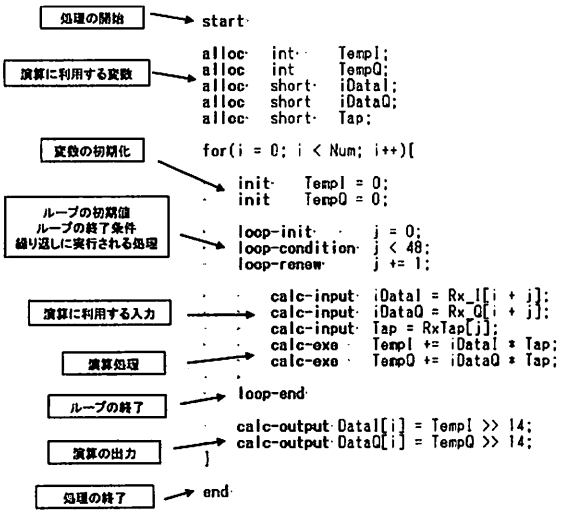

図 6　解析情報付加ソースコードの例

- 「alloc」のタグによって，演算に利用する変数を拡張命令内で定義することが可能となる．
- 「init」のタグによって，初期化が必要な変数を検出することが可能となる．
- 「loop-init」，「loop-condition」，「loop-renew」のタグによって，演算処理の繰り返し回数を検出することが可能となる．
- 「calc-input」や「calc-output」のタグによって，入出力のポート数やバス幅の制約を考慮することが可能となる．
- 「calc-exe」のタグによって，演算処理の位置を検出することが可能となる．

4.4　ハードウェア構成情報

S5530 用にソースコードを最適化する上で，考慮すべき特徴であるハードウェア構成情報を以下に示す．

- ユーザ定義の拡張命令を使用可能
- ISEF への入出力ポートの最大数は，入力=3，出力=2
- 入出力では任意の変数型を 128bit までパック可能

4.5　ソースコード自動最適化ツールの動作

ソースコード自動最適化ツールは，解析情報付加コードのタグを元に，ソースコードの自動最適化を行う．以下に，図 6 の解析情報付加コードから，各タグを検出した場合の動作について示す．また，図 7 に，作成された拡張命令定義コードと拡張命令呼び出しコードを示す．

(1) 解析情報付加コードから「start」のタグを検出した場合，最適化を開始する．
(2) 解析情報付加コードから「alloc」のタグを検出した場合，拡張命令内で利用する変数であると判断し，拡張命令定義コードに，変数を定義する．
(3) 解析情報付加コードから「init」のタグを検出した場合，初期化を行う変数があると判断し，初期化関数の定義および呼び出しを行う．
(5) 解析情報付加コードに「calc-input」，「calc-output」のタグがあった場合，ISEF との変数の受け渡しを行う記述を出力する．なお，このとき，「calc-input」のタグの個数から入出力ポート数の制約を考慮する．また，入出力変数の型から，ISEF へ同時に受け渡す変数の個数（パック数）を算出する．
(6) 解析情報付加コードから「loop-init」，「loop-condition」，「loop-renew」のタグを検出した場合，複数変数をパックし，並列処理を行うことで削減された繰り返し回数を算出し，記述を行う．
(7) 解析情報付加コードから「calc-exe」のタグを検出した場合，演算処理が存在すると判断し，拡張命令内の演算処理に演算内容をコピーする．
(8) 解析情報付加コードから「end」のタグを検出した場合，最適化を終了する．

4.6 性能評価

汎用ソースコードとして FIR フィルタ（48 タップ）を作成し，試作ツールの性能評価を行った．最適化を実行したソースコードの性能測定結果を表 1 に示す．比較対象として，手動で汎用ソースコードを最適化した場合，最適化を実行せず汎用ソースコードをそのまま利用した場合の結果も併せて示す．なお，処理時間削減率は，各方式を適用することで，最適化をしない場合と比較してどれだけ処理時間が削減されたかの割合を示したものである．

本結果より，提案した手法は，手動で最適化を行った場合よりも処理時間削減率が低いものの，十分な改善効果が得られていることがわかる．手動でソースコードを最適化する場合と比較して，総作業量が削減できることを考慮すると，提案した手法による自動最適化は本プロセッサに対して効果的であると言える．なお，自動最適化ツールの動作アルゴリズムの改良により，ツールによる最適化の処理削減率を手動により最適化の結果に近づけることが可能となるだけでなく，本ツールを FIR フィルタ以外の演算処理にも適用することが可能となる見込みである．

5. ま と め

本稿では，SDR プロセッサのハードウェアの特徴を活用したソフトウェアの最適化を自動実行する手法を提案した．本手法は，ハードウェアの特徴を活用する上で必要なソースコード解析情報を，最適化対象ソースコードに付加することを特徴とする．また，提案手法の有効性を確認すべく，本手法に基づき動作するツールを試作し，評価を行った．試作したツールを用いて FIR フィルタ処理の最適化を実行したところ，約 69%の処理削減効果が得られた．この削減効果は，手動で最適化を行った場合と比較して低いものの，最適化に必要な作業量が，提案手法により削減されることを考慮すると，提案した手法は本プロセッサに対して有効であると言える．

表 1 処理サイクル数

方式	自動最適化	手動による最適化	最適化なし
処理時間 [ms]	2.77	1.59	8.96
処理時間削減率 [%]	69	82	——

謝辞

研究を進めるにあたり日頃ご指導頂く KDDI 研究所秋葉所長ならびに野本執行役員に深謝する．

文 献

[1] 3GPP C.S0024 Ver.4.0, “cdma2000 High Rate Packet Data Air Interface Specification”
[2] Singo WATANABE, Yoshio KUNISAWA, Daisuke KAMISAKA and Yoshio TAKEUCHI, “A Software Radio Implementation of CDMA 1xEV-DO on a Single DSP Chip Designed for Mibile Handset Terminal”, IEEE VTC 2006 Fall.
[3] Daisuke KAMISAKA, Singo WATANABE, and Yoshio TAKEUCHI, “Study on Software Optimization for Sfotware Defined Radio using Multple-core DSP”, GSPx 2006.
[4] Sandbridge Technologies, http://www.sandbridgetech.com
[5] Andrew DULLER, Daniel TOWNER, Gajinder PANESAR, Alan GRAY and Willl ROBBINS, “picoArray technology: the tool’s story, ” Design, Automation and Test in Europe, 2005, p.106-111 Vol.3.
[6] John GLOSSNER, Danlel LANCU, Jin LU, Erdem HOKENEK and Mayan MOUDGILL, “A Software-Defined Communications Baseband Design, ” IEEE Communications Magazine p.120-128, January 2003.
[7] 山崎浩輔，渡辺伸吾，武内良男，“信号処理ソフトウェア可搬化技術に関する一検討，” 信学ソ大，B-17-19，Sep. 2007.
[8] WiMAX Forum, http://www.wimaxforum.org/home
[9] Stretch Inc, http://www.stretchinc.com

```
start

alloc   int     Temp1;
alloc   int     TempQ;
alloc   short   iData1;
alloc   short   iDataQ;
alloc   short   Tap;

for(i = 0; i < Num; i++){

    init    Temp1 = 0;
    init    TempQ = 0;

    loop-init       j = 0;
    loop-condition  j < 48;
    loop-renew      j += 1;

        calc-input  iData1 = Rx_I[i + j];
        calc-input  iDataQ = Rx_Q[i + j];
        calc-input  Tap = RxTap[j];
        calc-exe    Temp1 += iData1 * Tap;
        calc-exe    TempQ += iDataQ * Tap;

    loop-end

    calc-output Data1[i] = Temp1 >> 14;
    calc-output DataQ[i] = TempQ >> 14;
}

end
```

解析情報付加コード

```
static  se_sint<32>   Temp1;
static  se_sint<32>   TempQ;
static  se_sint<16>   iData1;
static  se_sint<16>   iDataQ;
static  se_sint<16>   Tap;

SE_FUNC void Init-func() {
    Temp1 = 0;
    TempQ = 0;
}

SE_FUNC void Calc-func(MR A, MR B, MR C) {
    int i;
    for (i = 0; i < 8; i++) {
        iData1 = A(16 * i + 15, 16 * i);
        iDataQ = B(16 * i + 15, 16 * i);
        Tap    = C(16 * i + 15, 16 * i);

        Temp1 += iData1 * Tap;
        TempQ += iDataQ * Tap;
    }
}

SE_FUNC void Output-func(MR *A, MR *B) {
    *A = Temp1 >> 14;
    *B = TempQ >> 14;
}
```

(2) allocのタグによる記述

(3) initのタグによる記述

(6) calc-inputのタグによる記述

(7) calc-inputのタグによる記述

(8) calc-exeのタグによる記述

(11) calc-outputのタグによる記述

拡張命令定義コード

```
MR  A, B, C;

for(i = 0; i < NUM; i++){

    Init-func();                        (3) initのタグによる記述

    for (j = 0; j < 48; j += 8) {       (4) loop-init,loop-condition,
                                            loop-renewのタグによる記述
        MRGET0INIT(&Rx_I[i + j]);
        MRGET0I(&A, 16);
        MRGET1INIT(&Rx_Q[i + j]);
        MRGET1I(&B, 16);
        MRGET2NIT(&RxTap[j]);
        MRGET2I(&C, 16);

        Calc-func(A, B, C);             (7) calc-inputのタグによる記述

    }                                   (9) loop-endのタグによる記述

    Output-func(&A, &B);
    MRS16I(A, &Data1[i]);               (11) calc-outputのタグによる記述
    MRS16I(B, &DataQ[i]);
}
```

拡張命令呼び出しコード

図 7　作成されたコード