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

2008－SLDM－133（31）
2008／1／17

ハードコンシャス記述：ハードウェア化を目的としたC言語記述スタイル

Kaiyi Mao† 天野 英晴† 堤 聡† Vasutan Tunbunheng†

† 慶應義塾大学理工学部
〒223-8522 神奈川県横浜市港北区日吉3-14-1
E-mail: †mao@am.ics.keio.ac.jp

あらまし 組み込み機器の多機能化，複雑化に伴い，C言語ベースのハードウェア設計が広範囲に用いられるようになった．C言語ベースのハードウェア設計では，既存のC言語で記述された豊富なアルゴリズムをいかに再利用するかが，設計期間の短縮および生産性向上のための鍵である．ところが，C言語特有の自由度の高い記述や，ソフトウェア的な技法によって，既存のアルゴリズムやプログラムを直接にハードウェア記述用C言語に移植することが困難となっている．本研究では，ハードウェア化が容易なプログラミング作法，ハードコンシャス記述スタイルを提案する．まず，MiBenchからいくつかのアプリケーションをハードコンシャス記述に書き直し，その記述からHandel-CとBach-Cなどのハードウェア記述用C言語に変換した．次に，ハードコンシャス記述に書き換えるための変更点を分類し，記述の変更に要する作業時間を評価した．評価結果から，ハードコンシャス記述は，アルゴリズム設計とハードウェア設計の架け橋として利用でき，設計時間を短縮できる可能性があることを示した．
キーワード 高位設計，コーディング作法，ハードコンシャス記述

Hardware Consious Style: A C-Language Style for Hardware Design

Kaiyi MAO†, Hideharu AMANO†, Tsutsumi SATOSHI†, and Vasutan TUNBUNHENG†

† Faculty of Science and Technology, Keio University
3-14-1 Hiyoshi Kohokuku, Yokohama 223-8522 Japan
E-mail: †mao@am.ics.keio.ac.jp

Abstract In order to cope with recent complicated functions in multi-media applications, C-based behavioral design method has been becoming popular in embedded hardware design instead of RTL design. However, re-usability of existing C program codes and portability to hardware description have turned out to be an obstacle in such design methodologies. Due to wide variation of coding styles and highly abstracted programming models, it is almost impossible to transplant directly these kinds of codes to hardware C descriptions. In order to address this problem, we propose a new C language coding style called hardware conscious style. We rewrote some application programs from MiBench applying the hardware conscious style, and then, we converted them to hardware C languages such as Handel-C and Bach-C. We collected the key points in the modification and evaluated the converting time and the the number of rewritten source code lines. The result shows that our proposed hardware conscious style can be used as a standard description of algorithms, and easily translated in hardware C description.
Key words High-Level Design, Coding Style, Hardware Conscious Style

1. はじめに

近年，組み込み機器の多機能化により，音声，画像などのマルチメディア処理機能，ネットワーク接続機能，暗号化復号化機能など，高い処理能力が要求されるアプリケーションが増えている．しかし，組み込みシステムで用いられるプロセッサは，消費電力とコストの小さなCPUであり，一部のプログラムをハードウェア化し，アクセラレータとして用いることで特定の性能のみを改善する方法が一般的である．また，安価で大容量のFPGAの普及により，大規模なハードウェアを様々なアルゴリズムに合わせて設計が可能となったことから，FPGA上でハードウェアを実装する方法も一般化してきた．

一方，多くのハードウェア設計企業は，激しい競争を勝ち抜くために，より短時間で新製品を市場に投入することが必要と

なってきた．ハードウェア設計では，高位設計が広がり，RT レベル記述の代わりに動作レベル記述が普及しはじめている．動作レベル記述，特に C ベースのハードウェア記述は，ANSI-C に近い文法を持っていることから，既存の豊富なアルゴリズム資源を再利用しやすく，高位での検証もできる．そのため，製品の設計を短時間で行え，生産性を向上させることができると期待されている．

しかし，C ベースの設計手法により既存のアプリケーションを移植するためには，設計者が手動での修正を要する問題が数多く残っている．例えばハードウェア化すべき様々な組み込み用アプリケーションの多くは動的なメモリ割り付けや，積極的なポインタの利用などのソフトウェア的な技法を駆使して記述されている．さらに，最近の C++や Java など，プログラムを実行しないと変数の型やサイズが決まらないように動的に抽象化される傾向もある．そのため，これらのアプリケーションを直接ハードウェア記述用 C 言語に移植することが不可能となり，結局，ハードウェア設計者はもう一度頭から元々のアルゴリズムとデータ構造を理解し，一から書き直さざるを得ない場合がほとんどである．

我々は，SEC/IPA の委託研究としてハードウェア化を意識した C 言語の書き方の作法，即ち，ハードコンシャス記述について検討している [9]．ハードコンシャス記述は，容易に各種ハードウェア記述用 C 言語に変換が可能で，ハードウェア化を行うアルゴリズムをこの記述で蓄積すれば，共通の資産として用いることができる．ベンチマーク用の記述法としても有効である．本報告では，このハードコンシャス記述について紹介する．

2. 関連研究

最近 C ベース設計が普及しているが，ソフトウェアとハードウェアの分担を含め，システム全体を記述するシステム記述用言語と，既存の HDL より抽象度の高いレベルの単体ハードウェアの設計を行うハードウェア記述用言語に分けられる．前者の代表は System-C [2], SpecC [3]，後者は Handel-C [4], Bach-C [5], Impulse-C [6], Mitrion-C [7] など様々である．我々が今回対象とするのは，アクセラレータ用ハードウェアの記述であり，後者のハードウェア記述用 C 言語である．しかし，これらの言語には，それぞれ独特の制約があり，記述の統一は困難であり，いずれかの言語で標準化を行うのは現実的ではない．

近年，組み込みソフトウェアの分野では，IPA/SEC（情報処理推進機構・ソフトウェアエンジニアリングセンター）により，組み込みソフトウェア開発向けコーディング作法ガイドがまとめられた．このコーディング作法はソフトウェアの品質を向上させるため，C 言語の記述の自由度を抑え，「信頼性」「保守性」「移植性」「効率性」の四つの品質特性を考え，提案した組み込みソフトウェア開発向けコーディング作法である．

この記述作法は，ハードウェア記述向けではないが，組み込みプログラム開発者が，この作法を守ってアプリケーションを書けば，ハードウェア記述用 C 言語への移植が従来に比べ格段に容易になる．しかし，この作法はあくまで組み込みソフトウェアの開発用で，ハードウェアの記述に用いるためには拡張が必要となる．

3. ハードコンシャス記述

3.1 ハードコンシャス記述の位置づけ

我々は，SEC/IPA による組み込みソフトウェア開発向けコーディング作法を基にして，ハードウェア化が容易なプログラミング作法，Hardware Conscious Style（ハードコンシャス記述スタイル）をまとめ，これによるアルゴリズムの記述の蓄積を試みた．

ハードコンシャス記述スタイルは，ハードウェア化を意識した ANSI-C の書き方の作法であり，新しいハードウェア記述用 C 言語を提案しているのではない．しかし，ハードコンシャス記述で書かれたコードは，様々なハードウェア記述言語に多くの場合，直接的に変更可能と考えられる．

ハードコンシャス記述を，直接変換してできたハードウェア記述用 C 言語で，優れたハードウェアがすぐに生成されるわけではない．しかし，この記述は機能合成ツールの能力を最大限に生かすと共に，設計者によって設計上のトレードオフの検討を容易にし，デバイスと要求仕様に合わせた優れたハードウェアを最終的に設計するために助けとなるはずである．

3.2 ハードウェアの意識

ハードコンシャス記述では次の 3 点に注意して記述する．

- 変数のサイズ，符号を常に意識し，管理する．これらはハードウェアのコストに直結する．
- 単一メモリではなく，複数のメモリモジュールを利用するという考え方をする．すなわち，配列等のデータ構造は，基本的にそれぞれ独立の幅と深さを持ったメモリモジュール上に実現されると考える．
- 全ての資源は静的であり，動的に生成，変更することはできないと考える．

ポインタの問題などは，複数メモリモジュールの利用とサイズの管理を意識すれば自然に制限される．後はアルゴリズムをごく自然に記述する．

ハードコンシャス記述における制約は次の通りである．

- main 関数に対する引数の禁止．
- 変数の符号とサイズの指定．
- 異なった用途に用いる配列の統合の禁止．
- ポインタの使用制限．
- '++' 演算子，'--' 演算子，',' 演算子の禁止．
- 変数のスコープの制限．
- ポインタの二重構造，リンクリストの禁止．
- 初期化処理は，事前に計算し，初期値として設定．
- 構造体の配列の禁止．

具体的な記述は，一般的な C 言語による記述からハードコンシャス記述への変換例と共に後の節に示す．

4. MiBench の記述

ハードコンシャス記述法を固めるため，組み込み用アプリケーションベンチマーク集 MiBench [8] を，この記述法で書き換えている．MiBench は University of Michigan で開発された組み込み用アプリケーションのベンチマークで，Automotive and In-

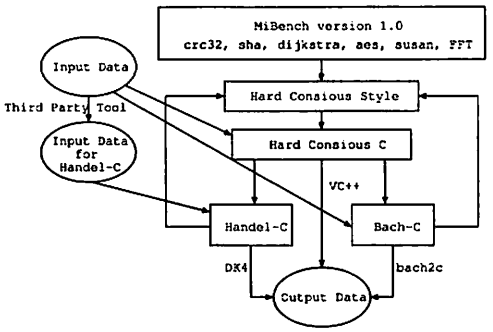

図 1 ハードコンシャス記述による記述と検証の流れ

dustrial Control, Consumer Devices, Office Automation, Network, Security, Telecommunication の五つの組み込み分野で，総計 25 個のプログラムが用意されている．ここでは，この中から，六つのプログラムを選んで，ハードコンシャス記述に書き直し，さらに，その記述からハードウェア記述用 C 言語に変換し，それぞれの主な修正点，修正時間の評価を行った．

4.1 実装したアプリケーション

- crc32 (telecomm): 32bit の CRC 計算するアルゴリズム．
- sha (security): 暗号化に用いるハッシュアルゴリズム．
- dijkstra (network): 最短経路の計算アルゴリズム．
- FFT (telecomm): 高速フーリエ変換のアルゴリズム．
- susan (automotive): 画像処理アルゴリズム (smoothing のみ)．
- aes (security): 共通鍵暗号方式ハッシュアルゴリズム．

4.2 対象ハードウェア記述用 C 言語

ここでは，ハードコンシャス記述から変換する対象のハードウェア記述用 C 言語として Handel-C と Bach-C を用いた．

Handel-C は Celoxica 社のハードウェア記述用 C 言語である．変換された記述は，デザインツール DK Design Suite 4 によってコンパイルした後，シミュレーション結果の確認を行う．バックエンドコンパイラには Visual C++を使用する．

Bach-C はシャープ社によるハードウェア記述用 C 言語である．変換された記述は，Bach System により，コンパイルして，シミュレーションを実行する．コンパイルコマンドは bach2c で，デバッグコマンドは bachdbg を使用する．

図 1 を示すように，MiBench から選んだ六つのアプリケーションをハードコンシャス記述に従って，書き直し，Visual C++でコンパイルして結果を確認する．さらに，書き直したソースから，Handel C と Bach-C のソースコードに書き直し，それぞれの開発環境でコンパイルした後，シミュレーション結果を確認する．

5. 主な修正点

5.1 ハードコンシャス記述への修正

本節では，MiBench のアプリケーション記述から，ハードコンシャス記述へ修正する際に行った主な修正点を紹介しつつ，ハードコンシャス記述における作法を具体的に説明する．

a) main 関数の引数の禁止

一般的な C アプリケーションには，argc と*argv[] など引数がよく使われているが，その引数はハードウェア記述用 C 言語で使用できないため，次のように削除した上に，戻り値を void に書き直す．

```
//int main(int argc, char *argv[])
void main(void)
```

また，main 関数本体で，これらの引数を使用しているコードの修正または削除の作業も必要になる．コマンドライン入力を引数で与えている場合は，この引数を変数か定数に書き直し，初期値を指定して固定する．

b) マクロ定義の修正

マクロ定義は，C 言語による記述でよく使われる一つの技法である．C のマクロ定義は自由度が高いので，そのままハードウェア記述言語に移植することが難しい．そのため，ハードコンシャス記述での制限と修正が必要である．修正方法は，場合によって異なるが，マクロ定義から変数や定数に変換，または関数に書き直すという方法がよく用いられる．次の例は，マクロ定義から関数に書き直したものである．

```
//#define word_in(x)      *(word*)(x)
UINT32 word_in(UINT8 *x, UINT8 base) {
    UINT32 ret;
    ret = (UINT32)(*((x)+3+base))<<24;
    ret = ret + ((UINT32)(*((x)+2+base))<<16);
    ret = ret + ((UINT32)(*((x)+1+base))<<8);
    ret = ret + (UINT32)(*((x)+base));
    return ret;
}
```

c) 変数のスコープの修正

ハードコンシャス記述では，変数のスコープに制限が加えられている．C 言語の自由度により，関数中のローカル変数のスコープが，ハードウェア記述用 C 言語のローカル変数のスコープと異なる場合がある．引数でデータのやり取りする方法は，ハードウェア記述 C 言語では制限が多い．特に，配列のデータの扱いに関する制限が一番多い．それは，この操作がメモリの制御に直結するためである．

d) 再帰呼び出し関数の禁止

再帰呼び出し関数はハードウェアで実装しても，一般的に効率が良くないため，ハードコンシャス記述ではその利用が禁止されている．このため，元のアプリケーションの再帰呼び出し関数を修正する．

e) データ変数の幅と符号の修正

一般的に，C 言語でアプリケーションを設計する際には，メモリの使用量を考慮して，データ変数の幅やデータの符号を考える．メモリの容量に余裕があれば，大きめのデータサイズを選択してもあまり問題はない．

だが，ハードウェア設計する際には，データのサイズがハードウェアのサイズに直接影響するため，適切にその選択を行い，

明示する必要がある．そのため，ハードコンシャス記述では，データ変数の幅と符号の指定を特に重要視している．

データ変数の幅と符号の指定は，次のように，マクロ定義によって既存のソフトウェア記述と両立できるようにしている．

```
typedef unsigned char UINT1;
...
typedef unsigned char  UINT8;
typedef unsigned short  UINT9;
...
typedef unsigned short  UINT16;
typedef unsigned long  UINT32;
```

次に，符号に関する修正も必要である．C 言語の記述中，int と宣言した場合でも，本当に符号付きでなければならない場合は少ない．例えば，多くの場合，for 文の回数をカウントするイテレータ変数は，符号付きの int で宣言されるが，本来は，符号無しで良く，ハードウェア化する場合，負の数にならないと分かれば，ハードウェア量を減らすことができる．このような，無駄な符号付き数に関して修正を行う必要がある．

f） データ変数のキャスト操作の修正

ソフトウェア記述では，変数の型が異なる場合でも，暗黙の変換ルールが成立する場合は，必ずしもキャストを行う必要がないが，ハードコンシャス記述では，異なる型を持つ変数間で演算を行う場合，変数のキャストを明示的に指定する必要がある．多くのプログラムで，暗黙のキャストが多用されているため，この変換作業に時間がかかる．

g） 構造体の修正

ハードコンシャス記述では構造体の配列を禁止している．例えば，次のコードで，配列 rgnNodes は 単一の物理メモリ上にフラットに並べると，rgnNodes[i].Cost[j] と rgnNodes[i].iDist が同時にアクセスできなくなってしまい，その結果，処理の並列性が制限される．

```
struct _NODE {
    UINT16 iDist;
    UINT16 iPrev;
    UINT16 Cost[NUM_COST];
};
struct _NODE rgnNodes[NUM_NODES];
```

このような場合，要素毎に独立の物理メモリを用いるのが最善の策である．つまり下記のように，別々の配列を定義し，これらが別々のメモリモジュールにマッピングされるようにする．

```
UINT16 rgnNodes_iDist[NUM_NODE];
UINT16 rgnNodes_iPrev[NUM_NODE];
UINT16 rgnNodes_Cost[NUM_NODE][NUM_COST];
```

rgnNodes0, rgnNodes1 など複数の構造体を扱う場合，記述のスマートさが失われるが，やむを得ないと考える．

Bach-C では構造体の配列自体が許されていない．Handel-C は可能だが，合成上の制約が大きい．構造体の配列を使わないことにより，実際のハードウェア実装に近い記述となり，変換時の手間を省くことができる．

h） ポインタの修正

ソフトウェアによる処理とハードウェアによる処理の根本的な違いは，ソフトウェアが全てを単一の広大なメモリ上で処理するのに対して，ハードウェアはデータを複数の限定された容量のメモリモジュールに分けて格納する点にある．

ハードコンシャス記述では，ポインタを完全に禁止してはいない．実際，全てのハードウェア記述用 C 言語でも，限定的にポインタの利用を許している．メモリのサイズが限定されており，かつ，幅もまちまちなのが複数存在するということを考慮することにより，ポインタの利用は自然に制限される．

ポインタが，通常のソフトウェアで便利なのは，単一メモリ空間上のどこでも，対象データのサイズを意識しないで相対的に指すことが可能であるためである．違ったサイズのデータに対してもキャストすることで扱うことができる．

ところが，ハードウェア化する場合には，物理的なメモリが複数別々に設けられ，そのサイズとデータ幅は，まちまちである．メモリのサイズが違えばアドレスの最大値も違うはずで，これを同じアドレスレジスタで扱うことは基本的にはあり得ない．したがって，ポインタのキャストは禁止されている．

ソフトウェア技法によくあるポインタの二重構造，リンクリストはハードコンシャス記述では禁止される．実際には，ポインタの二重構造やリンクリストは，それ自体ハードウェア化できないわけではない．一定の容量のメモリを設けて，その上で，アドレスレジスタと制御回路を生成して，ソフトウェアでの処理と同様に実現すれば良い．しかし，これをやると動作速度はメモリのアクセスに律速され，ソフトウェアに対してほとんど高速化できない．しかも，malloc のような動的なメモリ確保は全く不可能である．組み込み用途の処理において，多くの場合はリンクリストは通常の配列アクセスに変換可能である．

i） C 言語標準ライブラリ関数の書き直し

C 言語標準ライブラリ関数は，基本的に再定義すべき場合が多い．例えば，次のコードである．

```
void memcpy_bc(UINT8* dest, UINT8* src,
               UINT32 dest_i, UINT32 src_i,
                      UINT32 size) {
    UINT16 i;
    for(i = 0; i < size; i++)
        dest[dest_i+i] = src[src_i+i];
}
```

C 言語標準ライブラリ関数の memcpy に対して，この書き直した memcpy_bc には，dest_i と src_i という二つの引数が追加されている．Bach-C では，配列の先頭以外の要素へのポインタを使用して，その他の要素にアクセスすることができない．そのため，配列の先頭以外の複数の要素へアクセスするには，配列の先頭へのポインタと dest_i，src_i などのオフセットを利用して行う必要がある．

j） 複数命令構文の修正

ハードコンシャス記述では，インクリメント演算子，デクリメント演算子，カンマ演算子など，一つの構文で複数の命令が実行されるような記述に制限を加えている．これらは，場合によっては不必要な順序制約を課し，並列性を損ねることがある．このため，次のように修正する．

```
//*t++ = *f++
*t = *f; t++; f++;
```

また，カンマ演算子でつながっている，次のような二つの構文も修正する必要がある．

```
//ii=0, jj=0;
ii=0; jj=0;
```

5.2 ハードウェア記述用 C 言語への修正

本節では，書き直したハードコンシャス記述から，Handel-C や Bach-C などのハードウェア記述用 C 言語へ修正する場合の主な修正点を検討する．ハードコンシャス記述は，ハードウェア設計を考慮した記述スタイルのため，ハードウェア記述用 C 言語への変換は比較的容易である．

5.2.1 Handel-C への変換

Handel-C は，FPGA を用いたハードウェア設計に主に利用されており，最も普及しているハードウェア記述用 C 言語である．特徴としては，基本的に一クロックに一文ずつ実行される回路が合成され，ROM や RAM の宣言，クロックやリセットの記述など，ハードウェアを表現するための専用の記述方法が追加されている．ポインタについては扱える範囲が広いが，型変換などが厳密に要求される．

ここでは，ハードコンシャス記述から Handel-C への変換上問題になる部分のみ簡単に紹介する．

a） データサイズに関する修正

ハードコンシャス記述で，データサイズを表すためにマクロを使用しているため，元のソースコードを修正する必要はないが，マクロ定義を次のように，Handel-C の型定義に合うように修正する必要がある．

```
#define UINT1 unsigned int 1
...
#define UINT32 unsigned int 32
```

Handel-C では，データサイズのチェックは極めて厳格であり，配列の添字の範囲オーバー，サイズの異なった変数同士の演算等は全てエラーとなる．サイズ間の変換も厳格で，符号無し，符号付きデータ間は，ビット数が同じでないとキャストできない．ビット数が短くなる方向への変換は，次のように，ビット切出演算子 (< –) を使ってサイズを揃えてからキャストする．

```
UINT32 val1;
INT12 x;
x = (INT12)(val1<-12);
```

ビットが長くなる方向は，さらに面倒である．変換先が unsigned ならば，次の例のように，ビット連接演算子 (@) を使って拡張して，サイズを揃えてからキャストする．0 は任意の桁の 0 を表してくれるので一つだけ書けば良い．

```
INT32 val1;
UINT12 x;
val1 = (INT32)(0@x);
```

b） ポインタに関する修正

Handel-C は，ポインタに関しては比較的自由度が高く，ハードコンシャス記述を守っていれば直接変換することが可能である．

c） 入出力に関する修正

入出力はシミュレーション用と，それぞれのターゲットハードウェアに合わせた記述があり，本格的にターゲットハードウェア上に実装する場合は，記述を修正する必要がある．シミュレーション用には，次の例のように，ファイルを指定して入出力チャネルを宣言し，後は通常のチャネルに対する入出力同様の記述を行うことで可能である．ちなみに，? は入力，! は出力を示す．

```
chanin input_a with {infile = "input.txt"};
input_a?c;
chanout output_a with {outfile = "output.txt"};
output_a!(val);
```

5.2.2 Bach-C への変換

Bach-C は，Handel-C と比較して，変数の型変換を自動的に行ってくれるなど，型に関しては比較的制約が緩い．一方，ポインタについては，関数の仮引数としてのみしか宣言できないという制約があり，この点で Handel-C よりも厳格である．

a） データサイズに関する修正

データサイズに関しては，Handel-C と同様に，次のように Bach-C の文法に合うようにマクロ定義を変更するだけで，元のソースコードに手を入れる必要はない．

```
#define UINT1 unsigned int#1
...
#define UINT32 unsigned int#32
```

Bach-C は，異なる型の変数同士の変換について，Handel-C よりも制限が緩く，代入によって自動的に型変換が行われる．また，符号拡張も通常のキャスト操作で行われる．ハードコンシャス記述スタイルに沿っていれば，基本的には直接変換が可能である．

b） ポインタ，構造体の配列に関する修正

ポインタは，関数の仮引数としてのみ用いることが可能であ

表 1 評価結果

	HC-C conv. (days)	HW-C conv. (days)	Lines
crc32	3	1	-7
sha	3	1	+36
dijkstra	2	1	-11
FFT	2	1	-29
susan	9	4	-
aes	9	6	+66

り，これに対する演算の制限も大きい．また構造体の配列が許されていないため，構造体で表されるようなデータ構造は，あらかじめ複数の配列に展開する必要がある．

c） 入出力に関する修正

入出力は Handel-C と同様に，シミュレーション用と実際のハードウェア用の記述が用意されている．シミュレーション用には，次の例のように，ストリームチャネルに対してファイルを定義し，専用のライブラリ関数を使用して入出力を行う．

```
stream_in infile = "datain.txt" ;
org_w = getint(infile);
...
stream_out outfile = "dataout.txt" ;
putint(outfile, 10, 0, val);
putchar(outfile, '\n');
```

d） そ の 他

Bach-C では，extern キーワードがサポートされていないため，グローバル変数を定義することができない．そのため，予め，一つのファイルにすべての記述をまとめる必要がある．

6. 評　　価

六つのアプリケーションをハードコンシャス記述，ハードウェア記述用 C 言語に手作業で変換し，修正時間とコード量の増減について評価を行った．評価結果を表 1 に示す．修正時間は，元のソースコードからハードコンシャス記述へ変換した際の修正時間 (HC-C) と，ハードコンシャス記述に変換したものから，さらにハードウェア記述用 C 言語へ変換した際の修正時間 (HW-C) についてまとめた．

コード量の増減は，ハードコンシャス記述に書き直したコードと MiBench の元のコードとを比較した際の行数の差である．コード量を比較する際に，ヘッダーファイル，コメント，デバッグ用のコードは計算に入れていない．

修正時間の評価結果から，ハードコンシャス記述への修正時間が比較的長いが，これは，MiBench のコードのアルゴリズム，データ構造を理解する時間がかかるためである．ハードコンシャス記述は，ハードウェア設計向けに検討された記述手法であるため，ハードウェア記述用 C 言語への変換は，修正時間が短くなっている．

本研究は，ハードコンシャス記述の適用性を検討し，スタイルをまとめるため，MiBench のコードからハードコンシャス記述への修正を行ったが，実際の応用では，このステップが不要で，ハードコンシャス記述に従って，直接アルゴリズムを開発することになる．したがって，今回の評価では，ハードウェア記述用 C 言語への修正時間が重要である．

ハードコンシャス記述への修正時間と比較して，ハードウェア記述用 C 言語への修正時間が短いことは，一度ハードコンシャス記述に直してしまえば，様々なハードウェア記述用 C 言語への変換が容易となり，アプリケーション設計とハードウェア設計の架け橋となれることを示している．

コード量の評価結果から，sha と aes の実装ではハードコンシャス記述のコード量を増えるが，crc32，dijkstra，FFT の実装ではコード量を減ることが分かった．sha と aes のコードの増える部分は，C 言語の標準ライブラリ関数の memcpy と memset を書き直すのに必要なコード量である．crc32，dijkstra，FFT のコード量が減る部分は，main 関数の引数処理の箇所であり，アルゴリズム実体のコード量はあまり変化していない．

この評価から，組み込み用のアプリケーションをハードコンシャス記述で記述した場合でも，ソースコードの量の増加を憂慮する必要はないこと考えられる．

7. おわりに

本研究では，ハードウェア設計向けの C 言語記述スタイル，ハードコンシャス記述を紹介し，ハードコンシャス記述の有効性が明らかにした．

このスタイルの適用性を検証するため，複数のアプリケーションをハードコンシャス記述に書き直し，選んだ二つのハードウェア記述 C 言語，Handel-C および Bach-C に変換して，それぞれの主な修正点をまとめ，修正時間とコード量の増減について評価を行った．

今回の評価から，ハードコンシャス記述は，アルゴリズム設計とハードウェア設計の架け橋として利用でき，設計時間を短縮できる可能性があることを示した．また，コード量の増加は大きな問題にはならないことが分かった．

今の段階では，すべての作業を手動で行っているため変換に時間がかかる．我々の研究グループでは，現在，自動的に Handel-C と Bach-C へ変換できるトランスレータの開発を行っている．

文　　献

[1] 情報処理推進機構, ソフトウェアエンジニアリングセンター編：組み込みソフトウェア開発向けコーディング作法ガイド, 翔泳社, May 2006

[2] http://www.systemc.org

[3] http://www.cecs.uci.edu/ specc/

[4] Handel-C Language Reference Manual: Celoxa, 2005

[5] Bach-C 言語マニュアル, シャープ株式会社, 2004

[6] http://www.impulsec.com/

[7] http://www.mitrionics.com/

[8] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite", in IEEE 4th Annual Workshop on Workload Characterization, December 2001

[9] 天野 英晴, ハードコンシャス記述 C：ハードウェア化を目的とした C 言語記述作法, SEC journal no.10, p54-p61, May 2007