# MeP における SoC 開発手法

**SoC Development Methodology with MeP**

**長田　貴夫**　　**松本　展**

■ OSADA Takao　　■ MATSUMOTO Nobu

**MeP（Media embedded Processor）[(1)]は，SoC（System on Chip）におけるデータ処理を開発するための新しいプラットフォームである。この開発の大部分は C 言語を用いて行われる。ユーザーは，応用に際しキャッシュサイズなどのコンフィギュレーションを選択し，必要なハードウェアを C 言語にて追加（ハードウェア拡張）する。**

**東芝は，このコンフィギュレーションとハードウェア拡張に対応するために，MeP インテグレータ [(2)] というツールを開発した。この MeP インテグレータは，コンフィギュレーションなどからコンパイラ，シミュレータなどの開発環境，RTL（Register Transfer Language），及び検証ベクトルを自動生成する[(3)]。更に，MeP インテグレータを用いたトップダウン設計フローを提案する。この開発手法を MPEG-2（Moving Picture Experts Group-phase 2）コーデック[(4)]に適用した結果，性能要求を満たす設計を効率的に成し遂げることができた。**

The media embedded processor (MeP) provides a new platform for developing system-on-chip (SoC) data processing units. The major part of the development is performed using C language. Users choose suitable configurations, such as the cache size, and add the hardware required for their application in C language (hardware extension).

To support such configuration and hardware extension, Toshiba has developed a tool called MeP Integrator. MeP Integrator generates software development tools, such as a compiler and a simulator, Register Transfer Language (RTL) descriptions, and verification vectors from the configuration. Furthermore, we have also proposed a new top-down design flow using MeP Integrator. As a result of applying this development methodology to an MPEG-2 codec, a design satisfying performance requirements was efficiently completed.

## 1　まえがき

MPEGのような画像圧縮／伸張アルゴリズムの開発などにより，デジタル民生機器やデジタル通信機器が発展した。一方，このデジタル民生機器やデジタル通信機器では，多くの機能が要求される。この多くの機能を満たすような大規模集積回路のSoCが，半導体プロセスの発展によって開発できるようになった。その結果，デジタル民生機器やデジタル通信機器のSoCは，多くの機能を含んだ複雑なものになっている。

**図1**に示すように，これらSoCは，マイクロプロセッサ（MPU），USB（Universal Serial Bus）やIEEE1394（米国電気電子技術者協会規格1394）のような標準インタフェースのIP（Intellectual Property），及びデータ処理機能を備えている。MPUやIPは，これらSoCに固有のものではなく，既存の設計が再利用される。しかし，データ処理はSoCに固有なものである。したがって，データ処理については新規設計の必要な場合が多く，開発上のボトルネックになっている。更に，このデータ処理には，通常，高い性能と柔軟性が必要である。そのため，ハードウェアとソフトウェアを組み合わせて実現する必要がある。また，このデータ処理では，開発結果の再利用と，ツールサポートが必要である。よって，単一のプラットフォームをベースとして開発することが望まれる。

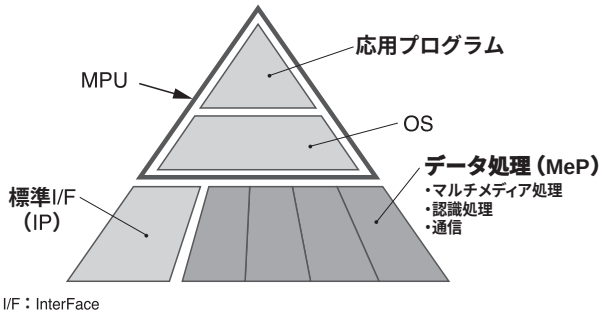


**図1．SoC 開発における MeP の位置づけ** ― MePは，SoCに固有のデータ処理機能を実現するために組み込まれる。

**Positioning of MeP in SoC development**

このデータ処理機能に，高い性能と柔軟性を実現するため，コンフィギュラブル プロセッサ MePを開発した。更に，MePのためのソフトウェア開発ツールとして，MeP インテグレータを開発した。このMeP インテグレータは，様々なコンフィギュレーションに対応し，かつ，開発結果の再利用を可能にする。

## 2 MeP コアの設計データと開発環境

MeP SoCを開発するためには，ユーザーが選択した構成に応じたLSIの設計データやソフトウェア開発ツールなどが必要になる（**図2**）。これらは，MePインテグレータというツールが自動的に生成・構築する。

このMePインテグレータを用いて，MeP SoCの構成を，テキストファイルやGUI（Graphical User Interface）にて対話的に指定する（**図3**）。

MePインテグレータによって自動的に構築される開発ツールの内容を，以下に示す。

⑴ Cコンパイラ，アセンブラ，標準ライブラリなど
⑵ C/C++レベルのシステムレベル シミュレータ
⑶ ソースコード デバッガ
⑷ JTAG（Joint Test Action Group）-ICE（In-Circuit Emulator），評価ボードなど（装置やボードは自動生成されない）

ここで，C/C++レベルのシステムレベルシミュレータとは，MePコアの命令セットシミュレータ，バスコントローラ，DMAC（Direct Memory Access Controller），ハードウェア拡張部のC/C++モデルをリンクしたシミュレーション環境である。ハードウェア拡張部のC/C++モデルについては，定められたAPI（Application Programming Interface）が用意されており，それに従ってモデルを作成すれば，MePインテグレータがMePコアの命令セットシミュレータなどと接続をする。このC/C++モデルはユーザー側で用意することが基本だが，アルゴリズム記述のC/C++から生成することが可能である。また，MePモジュールを複数含むマルチプロセッサ構成のシステムレベル シミュレータも生成できる。

更に，MePインテグレータは，MeP命令検証用のシミュレーションパターン，及びそれを実行するための検証環境も生成する。このシミュレーションパターンと検証環境は，MeP SoCの設計データそのものを検証するためのもので，その一部はユーザーにも提供される。

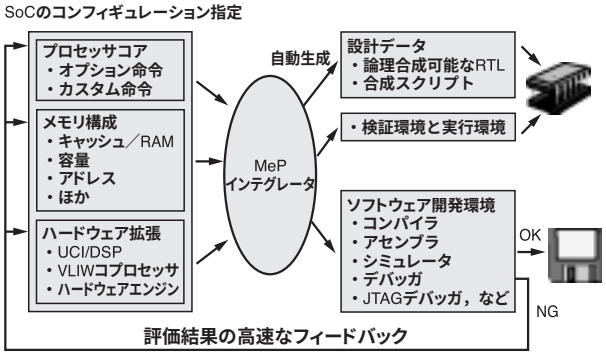


**図2．コンフィギュラブルなプロセッサの開発環境** ― MePインテグレータが，構成に応じた設計データやソフトウェア開発ツールなどを生成する。
**Development environment of configurable processor**

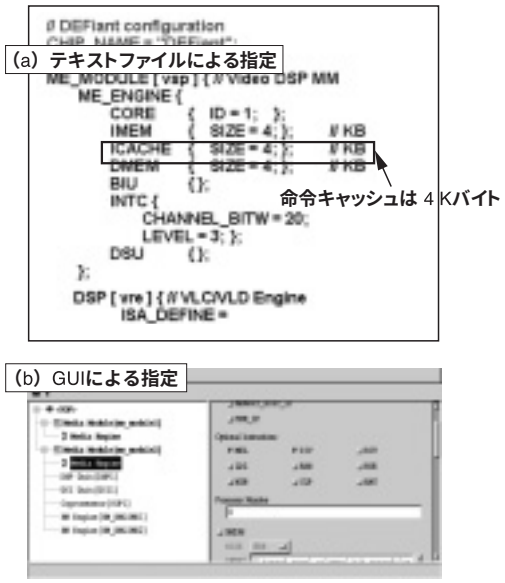


**図3．構成の指定** ― (a)では，ME_MODULE［vsp］以下に，VSPというMePモジュール構成が指定されている。例えば，図中の四角で囲まれた部分は，命令キャッシュが4 Kバイトであることを表している。(b)のメニューの左側の部分はハードウェア階層を表している。プルダウンメニューのコマンドなどにより，MePモジュールや他のハードウェアを追加することができる。
**Specification of configuration**

MePインテグレータは，また，この構成情報（コンフィギュレーション）からMePコアやバスインタフェース回路などのVerilogHDL（Hardware Description Language）のRTL記述，及び論理合成用スクリプトも生成する。

なお，ソフトウェア開発ツールについては，当社内で開発されたものを現在使用しているが，ほぼ同じ仕様に基づくGNU（GNU's Not Unix）ベースのMePインテグレータとソフトウェア開発ツール（米国RedHat社のGNUPro）の開発も進めている。

## 3 MeP の目指す設計フロー

### 3.1 トップダウン設計フロー

MePの目指すトップダウン設計フローは，C言語からSoCまでである（**図4**）。その設計フローは以下のとおりである。

⑴ グローバルな分割　C言語と抽象度の高いAPIを用いて，ハードウェアとソフトウェアの分割を行う。この分割は個々の性能見積りを行い，大量なデータ処理を行う部分のみハードウェアとして切り出す。残りをソフトウェアとして扱っておく。
⑵ システムシミュレーション　⑴で行った分割に対して，バスのトランザクションなどを考慮したシステム全体

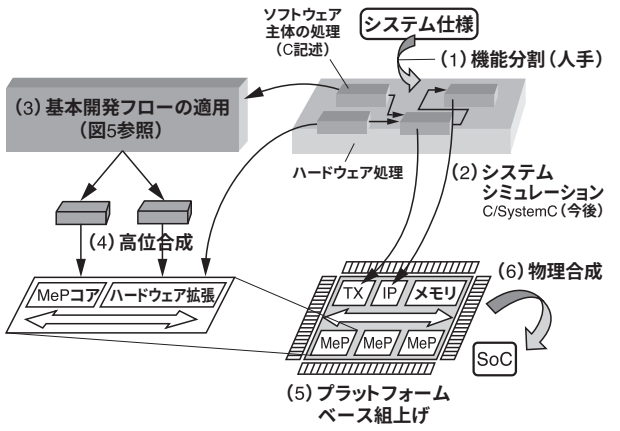


**図4．MeP の目指す SoC トップダウン設計フロー** ― ソフトウェア記述(C記述)からSoC設計までを行うフローを実現する。
**SoC top-down design flow with MeP**

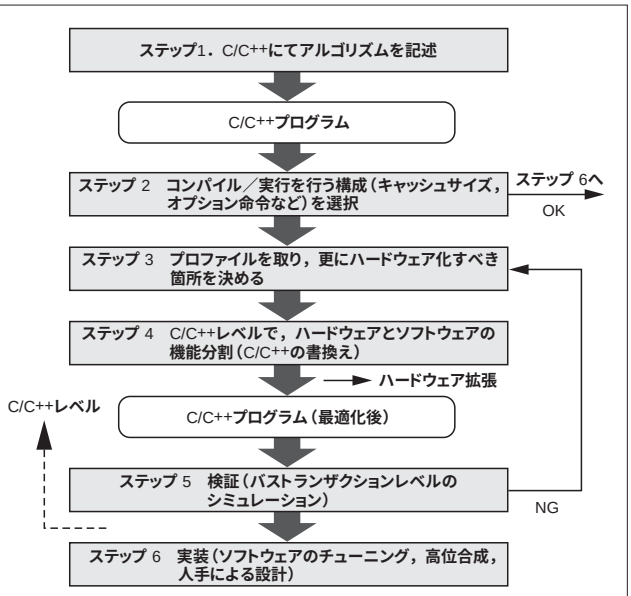


**図5．C/C++ からの開発フロー** ― 上流設計における大部分の工程(ステップ1～ステップ5)ではC/C++を用いる。
**Development flow from C/C++**

の性能見積りを行う。

⑶　局所的な分割　　(1)のソフトウェアの部分において，局所的な分割を行う。この局所的な分割は，要求性能が満たされるまでハードウェア化する箇所を切り出す。

⑷　MePモジュールのRTL設計とRTL検証　　高位合成などを用いたMePモジュールのRTL設計や，RTL検証を行う。

⑸　プラットフォームの組上げ　　マルチMePの組上げやIPの再利用など，バスを中心としたプラットフォームベースの組上げを行う。

⑹　物理合成　　タイミング収束の問題を解決するために，物理合成を行う。物理合成は，論理合成，フロアプラン及びレイアウト設計の一部である配置設計の工程を一貫して行う。

このトップダウン設計フローの特徴は，上流設計の大部分の工程をC/C++言語を用いて行うことである。

次に，MePインテグレータを用いた局所的な分割(基本開発フローの設計)について述べる。

### 3.2　ハードウェア拡張の基本フロー

MePを用いたシステムの開発フローは，大部分C/C++にて行われる(**図5**)。まず，C/C++でデータ処理のアルゴリズムを記述する(ステップ1)。次に，MeP用のC/C++コンパイラを用い，作成したC/C++プログラムをコンパイルする。そしてシステムレベルシミュレータを用いて実行し，その性能を評価する(ステップ2)。ここでユーザーは，キャッシュメモリのサイズやオプション命令のある／なしなど，構成をいろいろと変えて性能評価をする。各構成に対応したC/C++コンパイラやシステムレベル シミュレータは，前述のようにMePインテグレータによって自動生成される。構成を変えて実行した結果，必要な性能が満たされた場合は，すべてソフトウェアで実装できるということになる。そして，実装(ソフトウェアのチューニングなど)の工程へと進む。しかし，実際の開発ではそう簡単にいかない場合も多い。

構成を変えて実行しても必要性能を満たせなかった場合には，その一部の処理をハードウェアで実装して性能を改善する。その場合，まず，システムレベル シミュレータに付属するプロファイル機能を用いて，ハードウェア化すべき箇所，すなわち性能上のボトルネックとなっている部分(通常は，入力するC/C++プログラムのもっとも内側のループの一部)を抽出する(ステップ3)。

次に，その部分のC/C++プログラムを書き換えて，ハードウェアによる実装に変更する(ステップ4)。この作業が，MePモジュールのハードウェア拡張部の開発に相当する。そして，ハードウェアに変更した後，再度シミュレーションを実行して必要性能を満たしているかどうかを確認する(ステップ5)。ここでシステム性能を適正に評価するためには，バストランザクションを考慮したシミュレーションが必要になることに注意すべきである。

要求性能が満たされていた場合は実装(ステップ6)の工程に進むが，そうでない場合はプロファイル取得の作業(ステップ3)に戻り，更にハードウェア化すべき箇所を探す。

このように，必要な性能が満たされるまでハードウェア化する箇所を増やしていくと，最終的に必要性能を満足するようなハードウェア アーキテクチャと，その上で動作するソフトウェアを取得するとができる。なお，マルチタスクのプログラムの場合は，各タスクの必要性能を定め，以上の手順を各タスクについて繰り返すことになる。

## 4 ハードウェア拡張

MePのハードウェア拡張の設計方法について述べる。この方法は，ユーザー定義命令を追加する方法と，ハードウェアエンジンを追加する方法に大別される。ユーザー定義命令を追加する方法は，更に，UCI(User Custom Instruction)，DSP(Digital Signal Processor)，及びVLIW(Very Long Instruction Word)コプロセッサを用いる方法に分類されるが，いずれの方法でも，基本的に機械命令関数(Intrinsic Function)により行われる。

### 4.1 機械命令関数を用いた命令拡張

ユーザー定義命令を追加する方法の，機械命令関数について述べる。

**図6**は，コプロセッサ命令を用いてSIMD(Single Instruction Multiple Data)命令を定義し，高速化を図る例を示している。

命令拡張を行う場合は，命令拡張を行う部分を関数で記述し(C記述)，これを機械命令関数と呼ぶ。この機械命令関数に対する命令定義は，アーキテクチャ データベースで行う。アーキテクチャ データベースを読み込むMePインテグレータは，コンパイル環境を自動構築する。このコンパイル環境(コンパイラ)が，C記述の機械命令関数をアセンブラに変換する。コンパイラは更に，レジスタの割付けと，命令スケジューリングを行う。

ここで，アーキテクチャ データベースは，命令定義のほかに，SIMDの並列度やパイプラインの型などを記述している。

**ユーザー定義命令を関数形式で記述(機械命令関数)**

<Cのソース>

```
__pack short func (
    ⋮
    cpmov16_h(i, 1) ;
    cpadd_h(j, src2, i) ;
    cpsll_h(k, src1,1) ;
    cpor(l, j, k) ;
    }
```

コンパイラ

<コンパイル結果>

```
.x:
    cpmovi16.h $c10,1
    cpadd.h $c10, $c2, $c10
    cpsll.h $c11, $c1, 1
    ret
  + cpor $c0, $c11, $c10
```

**入力(アーキテクチャ データベース)**

```
CPMOVLA0.H : CRo :   //CRo <ーCCR
             11101_00000_00100_ooooo:
             {
             unsigned 8 i;
             expand(i = 0:3) {CRo@h [ i ] = (CCR (i+16)) [15 :  0] ; } ;
                  ⋮
             } :
             Cop : " SIMD = H, Group = CR_W_ACC0_R, Pipe = cro_w_acc0_r " :
             P0s :
```

**図6．ユーザー定義命令によるハードウェア拡張(機械命令関数)** ― ユーザー定義命令を用いたハードウェア拡張(UCI拡張，DSP拡張，VLIWコプロセッサ拡張)には機械命令関数を用いている。

**Hardware extension by user definition command (intrinsic function)**

MePインテグレータは，このアーキテクチャ データベースを用いて，コンパイラアセンブラ情報，C++パイプライン シミュレーションモデルや検証ベクトルなどを生成する。

### 4.2 ハードウェアエンジン

次に，ハードウェアエンジンを追加する方法には，人手でハードウェアエンジンを作成する方法と，自動生成する方法がある(**図7**)。

人手で作成する場合は図7(a)に示すように，APIに従ってハードウェアエンジンのC/C++のシミュレーションモデルを作成し，かつそのハードウェアエンジンを起動するようにソフトウェアも書き換える。なお，MePインテグレータのシステムレベル シミュレータには，ハードウェアエンジンのモデルを組み込むためのAPIが定義されている。

自動生成する方法では，図7(b)に示すように，シミュレーションモデルの作成以降の作業が自動化される。ユーザーは，C/C++プログラムのうちハードウェアエンジンで処理したい部分を関数としてくくり出す。そうすると，MePインテグレータがハードウェアエンジンを使えるようにソフトウェアを変換し，C/C++モデルを自動生成する。更に，高位合成ツールを利用してハードウェアエンジンのRTL記述を自動生成することもできる。

自動生成する方法では，ハードウェアエンジンのモデルを再度設計する必要がない。そのため，ハードウェア拡張の設計を繰り返し行え，最適なアーキテクチャを求めることが容易にできる。ただし，この方法では，処理できるC/C++プログラムの記述に制限がある。例えば，ポインタや構造体は使えない。

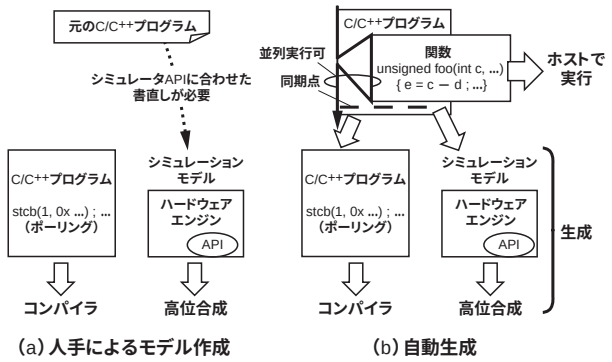


**図7．ハードウェアエンジンを用いる場合の作業** ― 人手でモデルを作成する方法と自動生成の方法がある。

**Work in case of using hardware engine**

## 5 実機におけるハードウェア拡張の効果

### 5.1 車載用画像認識装置

車載後方監視システムは，後方カメラにより後続車検出を

行っており，この後続車検出は，イメージ処理チップで処理する。このイメージ処理は，ひずみ補正，エッジ検出（ソーベルフィルタ），領域の追跡を行っている。これらの処理などを行い，車両を認識することができる。

今回の要求仕様では，MePの拡張機能なしの場合に比べ，7倍の処理速度が必要であった。そこで処理の高速化を行うために，3.2節で述べた基本開発フローを適用し，ハードウェアの拡張を行った。例として，垂直方向と水平方向のソーベルフィルタの処理に，VLIW命令を用いた結果を**図8**に示す。その結果，8画素の3×3フィルタ処理が，9サイクルで実行可能になった。

最終的に1フレーム（320×240，8ビットピクセル）のソーベルフィルタの処理性能が15倍，全体性能では6～7倍向上した。

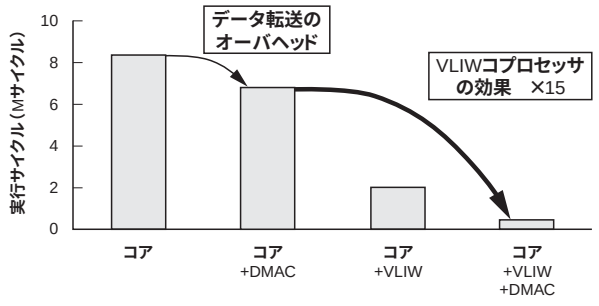


**図8．フィルタ処理の実行サイクル数** ― VLIW拡張を行うことにより，実行サイクルが15倍向上している。
**Number of filter processing execution cycles**

### 5.2　MPEG-2コーデック

今回のMPEG-2コーデックの開発には，トップダウン設計フローを適用している。このトップダウン設計フローに従って，ハードウェア拡張を行った結果を示す。

ハードウェア拡張では，グローバルな分割（3.1節の（1））と局所的な分割（3.1節の（3））に処理を分けている。グローバルな分割では，個々の性能見積りに基づいてDCTや量子化などの処理をハードウェアにした。また，局所的な分割では，まず，基本開発フローを用いてハードウェア拡張可能な箇所を求めた。**図9**に示すとおり，各処理に応じて違う処理を採用した。オーディオ処理にはVLIW命令を用い，ビデオデコーダの動きベクトル予測値にはハードウェアエンジンを，更に，ビデオエンコーダ処理のSIMDの部分にはUCI命令を用いた。このビデオエンコーダ処理では，単位Kゲート当たりの高速化が，グローバルな分割をした後の処理と比べ，10倍以上になった。これはグローバルな分割を行ったDCTなどのハードウェア化の後の，局所的な分割がUCI（SIMD）に適した処理であったためと考えられる。以上の結果，トップダウンフロー設計を用いたハードウェア拡張では，ハードウェアの拡張方法を使い分けることで効果が上がった。

| 処　理 | ハードウェア拡張 | ゲート数 | 高速化 |
|---|---|---|---|
| オーディオ処理 | VLIWコプロセッサ<br>・ALU／積和 | 38 K | 40 % |
| ビデオデコーダ | ハードウェアエンジン（自動生成）<br>・動きベクトル予測値（PMV）のハードウェア化 | 12 K | 19 % |
| ビデオエンコーダ | UCI<br>・2並列SIMD算術，論理，シフト，復号演算<br>・バイトシャッフル命令 | 3.1 K | 38 % |

ALU：Arithmetic Logic Unit　PMV：Predict Motion Vector
AAC：Advanced Audio Coding

**図9．ハードウェア拡張（ステップ3のみ）の効果** ― いろいろな拡張方法を使い分けることが効果的である。
**Effect of hardware extension (step 3 only)**

最終的に，これらのハードウェアの拡張方法を用いたMPEG-2コーデックの全体性能は，40倍向上した。

## 6　あとがき

MePのためのソフトウェア開発ツールMePインテグレータは，様々なコンフィギュレーションに対応する。更に，MePインテグレータを用いたトップダウン設計フローでは，ハードウェアの拡張方法を使い分けることによって，最適なハードウェアとソフトウェアを効率よく開発できる。

## 文　献


(1) （株）東芝．Media embedded Processor.
＜http://www.MePcore.com/＞，（参照2003-03-20）．
(2) Mizuno, A., et al. "Design Methodology and System for a Configurable Media Embedded Processor Extensible to VLIW Architecture". Proceeding of ICCD 2002. p.2－7.
(3) Kohno, K., et al. "A New Verification Methodology for Complex Pipeline Behavior". Proceeding of DAC 2001. p.816－821 .
(4) Ishiwata, S., et al. A Single-Chip MPEG-2 Codec Based on Customizable Media Microprocessor. IEEE Journal of Solid-State Circuits. **38**, 3, 2003, p.530－540.



**長田　貴夫**　OSADA Takao
セミコンダクター社　SoC研究開発センター　デジタルメディアSoC技術開発部。SoC上位設計の技術開発に従事。
SoC Research & Development Center



**松本　　展**　MATSUMOTO Nobu
セミコンダクター社　SoC研究開発センター　デジタルメディアSoC技術開発部主査。ソフトウェア開発環境，SoC上位設計の技術開発に従事。
SoC Research & Development Center