OKI Cベース SoC設計技術

岡田　敦彦　　児玉　秀賢
槇　和彦

従来SoC（System on Chip）を開発する工程は，システム上で実現したいアルゴリズムを用意し，SoCのアーキテクチャを決めてから，性能・コスト・消費電力の要求仕様を導き出すのが一般的である。しかしこの順番，すなわちSoCのアーキテクチャが決まった後で諸元を導出したのでは，差別化となる特徴を持ったLSIを開発するのは困難である。しかも，SoCのアーキテクチャ検討ができたとしても，さまざまなアーキテクチャを机上で見積るだけでは精度が低すぎて実際の性能と乖離が大きく，また，特定のアーキテクチャに絞ってFPGAボードを使って評価するにしても，さまざまなアーキテクチャを評価するには，膨大な時間がかかってしまうという問題点があった。

OKIでは以前より，高速・高精度な仮想プラットフォーム[1]や，短期間でアルゴリズムをハード化する技術[2]を構築し，C言語ベースの設計技術を蓄積してきた。そこでこの技術を適用することで，ユーザーの実現したいアルゴリズムからSoCのアーキテクチャを短期間のうちに探索し，要求仕様に合わせた差別化を図れるようにした。実際に我々は，現在までに特徴ある商品の量産に成功している（**図1**）。

本稿では，SoCのアーキテクチャ探索を中心に，**OKI**のC言語ベース設計技術を説明する。

MP3復号化LSI
「ML2841」
楽曲復号化時の消費電力従来比1/5に

MPEG-4符号化LSI
「ML86410」
クロック周波数を従来の数GHzから80MHzに低減

図1　**OKI**のCベースSoC商品

アーキテクチャ探索を中心とした設計フロー

OKIのC言語ベースのSoC設計は，ユーザーが望む諸元に基づいたアーキテクチャを短期間のうちに提示し，提示内容と乖離の無いLSIを提供することを実現している。**図2**にユーザーのアルゴリズムを実現するSoC設計フローを説明する。

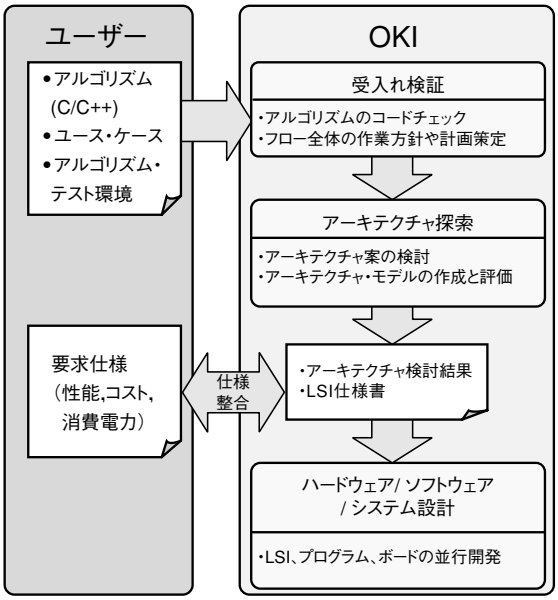

図2　C言語ベースSoC設計フロー

ユーザーからはSoCで処理したい「C/C++言語で記述されたアルゴリズム」と，そのアルゴリズムのSoC上での使い方「ユース・ケース」，そして，「アルゴリズム・テスト環境」を受け取り，開発に着手する。ここで重要なのは，ユース・ケースを明確に定義することである。ユーザーの目的・用途に特化してアーキテクチャを絞り込むため，アルゴリズムをシステム上でどのように使うかが重要になるからである。

（1）受入れ検証

初めに，受け取ったアルゴリズムのコードをチェック

し，フロー全体の作業方針とスケジュールを計画する。高速化のためのアセンブラ記述など，コードが特定の開発環境に依存していると，探索の基本となるアーキテクチャへの移植作業や後工程での最適化作業に大きく影響するため，十分なコードのチェックが必要である。

(2) アーキテクチャ探索

次に，このフローの中心となるアーキテクチャ探索工程となる。まずは入手したコードを探索用のアーキテクチャモデルへ移植して，初期性能を十分解析しておく。

この解析結果を元にアーキテクチャの探索方針を明確にし，アーキテクチャのモデル作成および評価を行う。評価は，機能・性能だけでなく消費電力やコストまで実施し，見積り精度向上を図る。

(3) ハードウェア／ソフトウェア／システム設計

アーキテクチャ探索工程が終ると，見積った仕様と，ユーザーの要求仕様を突き合わせる。

仕様の突き合わせで，ユーザーとの合意が取れれば，SoCのハードウェア設計とソフトウェア設計，そしてシステム設計を同時に開始する。この段階において，十分にアーキテクチャ検討が完了していること，およびプログラム開発が可能なシステムのエミュレーション環境が整っていることから，ハードウェア／ソフトウェア／システムの設計を並行して進めることができる。

アーキテクチャ探索工程は従来の設計手法よりも多くの工数がかかるが，開発全体の期間を見るとハードウェア・ソフトウェア・システム設計期間が短縮される効果のほうが大きい。我々の実設計においては，約30%の開発期間短縮を実現している[1)]。

アーキテクチャ探索を短期間化

さて，全体の開発期間が短縮されるとはいえ，アーキテクチャ探索工程に多くの工数を要することに変わりはない。そこで，我々は探索のための基本アーキテクチャモデルを開発することで，さまざまなアルゴリズムに柔軟に対応できるようにした。また，探索のための作業手順を体系化して，探索の効率化を図った。

(1) 探索のための基本アーキテクチャモデル

基本アーキテクチャモデルを**図3**に示す。このモデルの特長は，最小リソースで実現される汎用的なプラットフォームと，アルゴリズムを高速処理するための専用アクセラレータとを組み合せていることである。

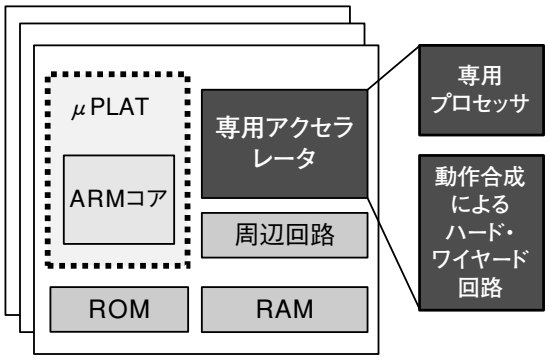

図3　基本アーキテクチャモデル

汎用的なプラットフォームは，「ARM*1) コア」を含む「μPLAT*2)」で構成されており，ユーザーはアルゴリズムを組み込んだアプリケーションを容易に開発することができる。専用アクセラレータは，さらに特徴的な演算を効率よく処理する専用プロセッサと，動作合成によるハード・ワイヤード回路の要素を配置している。専用プロセッサは，最小限のリソースで演算することができるため，低コストが要求される場合に適している。ハード・ワイヤード回路は，リソースを多数配置して並列化した演算を高速に処理できるため，高性能が要求される場合に適している。専用プロセッサとハード・ワイヤード回路といった特徴の相反する要素を組み合わせることで，コスト，性能，消費電力といったユーザーの要求仕様に応じたアーキテクチャ探索を行っている。たとえば，性能よりもコストに特徴を出したければ，専用プロセッサでの処理の比率を高くするし，コストよりも性能や消費電力に特徴を出したければ，ハード・ワイヤード回路での処理の比率を高くするアーキテクチャとしている。

このように，ユーザー要求に対して，柔軟に対応できる基本アーキテクチャモデルを用意している。

(2) 基本アーキテクチャモデルによる探索フロー

探索のための基本アーキテクチャモデルを用意しても，探索そのものに時間がかかっては事業として成立しないと考え，我々は探索のための手順を体系化することにした。**図4**（次ページ）に探索のフローを示している。

まずは，アルゴリズムのコードを専用アクセラレータに移植して，プロファイルを取得する。性能向上・消費電力低減・コスト低減の観点から，具体作業リストを作成し，探索プランを明確化しておく。その後，専用プロセッサでの探索と動作合成での探索を開始する。探索工程は往々にして“行き当たりばったり”になりやすく，探

＊1）ARM，AMBAは，英国ARM Ltd.の英国およびその他の国の登録商標です。　＊2）μPLATは沖電気工業株式会社の日本，ドイツ，英国における登録商標です。

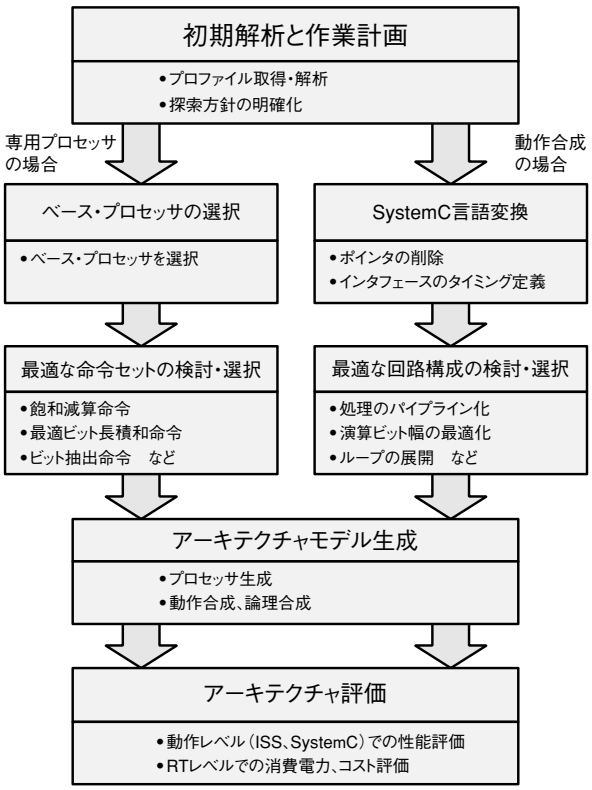

図4　専用アクセラレータの探索フロー

索のためのイタレーションが多数発生してしまう危険性がある。アルゴリズムのコード解析と探索方針を十分に明確にしておくことが重要となる。

専用プロセッサの場合，アルゴリズムの特徴に合わせたベース・プロセッサを起点として探索を開始する。たとえば，楽曲処理のための24ビット演算機能を備えたプロセッサや，2次元画像処理のためのSIMDプロセッサなどである。対象となるアルゴリズムの特徴に合わせたベース・プロセッサを用意することで短期間に目標性能を達成できる。

次に，ビットストリームの処理や任意のビット長の演算など，専用プロセッサに処理させたいコードから特徴的な処理を抽出し，専用命令案として検討する。アルゴリズムの特徴だけでなく，実際のコードの特徴から専用命令を追加することで，より繊細な探索が可能となる。

動作合成の場合，まずは動作合成可能な記述としてアルゴリズムをSystemC言語へ変換する。このとき，インタフェースのタイミングなどを定義する必要があるが，対象コードが変わるたびにインタフェースを作っていては工数が増大する。そこで，我々は動作合成のためのインタフェース仕様を定義し，SystemCへの変換と動作合成の工数を効率化している。

次に，リソースと処理のトレードオフを考え，処理のパイプライン化や演算ビット幅の最適化等を検討する。

検討結果がまとまったら，最適化案を動作レベルのモデルへ適用して，RTL（Register Transfer Level）へ展開する。そして，動作レベルのモデルは仮想プラットフォーム上に実装して性能を評価し，RTLモデルは消費電力・コストを評価する。

従来は，動作レベルからRTLへの展開作業を人手で実施していたため，膨大な時間を要しており，十分なアーキテクチャ探索ができていなかった。そこで，我々は市販のプロセッサ生成ツールおよび動作合成ツールを導入して，作業を自動化している。これらツールによる自動化により，作業工数や検証品質が格段に向上し，より多くのアーキテクチャの比較評価が可能となった。

探索のアプローチを体系化し，実作業を自動化することで，1カ月程度で探索を完了することを目指している。現在のところ，音声伸張アルゴリズムで約2週間，画像圧縮で約1カ月，画像認識で約1.5カ月と，短期間のうちに探索することができるようになった。

低消費電力化や小面積化を達成

今回のC言語ベース設計技術を実際のSoC開発に適用した事例を紹介する。

まずは，遠隔監視システム向けのMPEG-4圧縮LSI「ML86410」である（**図5**）。このLSIはMPEG-4圧縮の単機能であることと，データ処理の並列化が動作周波数の低減に有効であると考え，動作合成を使い全体をハード・ワイヤード化する方針とした。

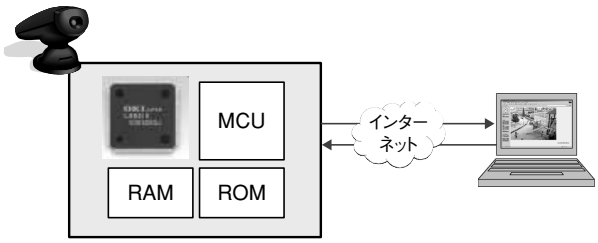

図5　遠隔監視システム

アーキテクチャ探索工程では，並列処理の段数などが異なる複数の構成を検討し，動作周波数とシステムの配信遅延を評価した（**表1**）。その結果，従来1GHzの汎用プロセッサを使って処理していた圧縮処理を，圧縮の品質を維持したまま81MHzまで低減することができた。

二つ目は，MP3伸長LSI「ML2841」の高機能版で，

表1　アーキテクチャ探索結果

	従来製品	要求仕様	アーキテクチャ探索結果			実LSI測定結果
			候補 1	候補 2	候補 3	
画素数	VGA	VGA	VGA	VGA	VGA	VGA
フレーム数（フレーム秒）	30	30	30	30	30	30
動作周波数（MHz）	1100	100以下	108／54	54／108	81／81	81／81
消費電力（mW）	5000	700以下	400	450	350	320
配信遅延（ms）	—	200以下	150	90	120	120
コスト（相対値）	—	1	1	1.1	0.9	0.88

＊ 複数の候補の中から、ユーザーの要求仕様にあったアーキテクチャ（表中候補３）を選択

AAC形式の音楽も再生できるLSIのアクセラレータである。複数の形式に対応する伸長LSIのため，リソースの共用に強みがある専用プロセッサを選択し，面積増加を抑えながら低消費電力を目指した。

図6にアーキテクチャ探索の結果を示す。楽曲データの伸長に必要な消費電力は10mWと小さい。これは，動作合成を使って完全ハード・ワイヤード化した場合の7mWと比べて，遜色のない値である。しかも，二つの圧縮方式それぞれの伸長回路を動作合成したアーキテクチャと比べると，約65%も小面積化している。

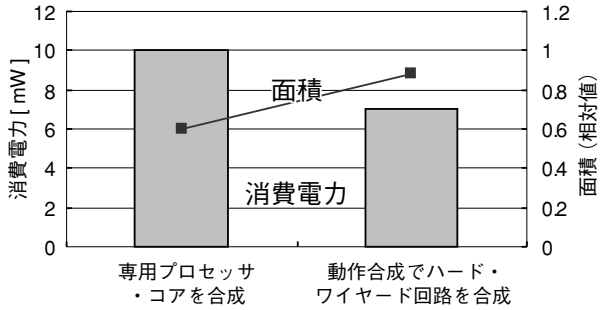

図6　専用プロセッサの効果

最　後　に

我々は，これまで述べてきたC言語ベース手法を適用し，画像や音声処理用SoCを開発してきた。MPEG-4圧縮LSI「ML86410」の開発では，クロック周波数を81MHzと大幅に低減し，高速な汎用プロセッサを必要としない安価なシステムを実現した。また，MP3／AAC伸長向けアクセラレータ開発では，複数の伸張機能に拡張したうえで，面積を最小限に抑えながら世界最小クラスの消費電力を達成することができた。

今後は，画像や音声の圧縮・伸張アルゴリズムだけでなく，暗号化および認証アルゴリズムに注力したSoCにも適用する。そして2〜3年後には，SoC全体のうち20〜30%程度はこの方式を適用して開発する計画である。◆◆

■参考文献

1）沖電気,システムLSIの開発期間を1/3削減，**OKI**プレスリリース，2004年11月29日
2）岡田敦彦 他，デジタル・オーディオLSIのCベース設計，沖テクニカルレビュー203号，Vol.72 No.3，P.44-47，2005年7月

●筆者紹介

岡田敦彦：Atsuhiko Okada. シリコンソリューションカンパニー カスタムビジネス本部 民生LSI設計第一部 民生MCU設計第二チーム
児玉秀賢：Hidetaka Kodama. シリコンソリューションカンパニー カスタムビジネス本部　民生LSI設計第一部
槇和彦：Kazuhiko Maki. シリコンソリューションカンパニー カスタムビジネス本部 民生LSI設計第二部

TIPS【基本用語解説】

C言語ベース設計
主にC/C++やSystemC言語でハードウエアモデルを記述し，システム全体を検証したり，RTLやネットリストへ展開したりする設計手法。抽象度が高く記述できるため，従来よりも検証時間が大幅に短縮される。

動作合成
SystemC言語で記述された“動作”を，HDLへ自動で変換する技術。アルゴリズムの規模にも依存するが，手作業なら6カ月かかるHDL変換作業が2カ月に短縮される。

プロセッサ生成
プロセッサ記述言語から，プロセッサモデル（シミュレーションモデル，HDL）および開発環境（コンパイラ，デバッガ）を自動で生成する技術。従来，6カ月以上かかっていた開発期間が，この技術により2カ月に短縮される。