# 2 System Level Design

# システムレベル設計フローと設計言語

NECエレクトロニクス（株）
黒坂　均
hitoshi.kurosaka@necel.com
松下電器産業（株）
竹村 和祥
takemura.kazuyoshi@jp.panasonic.com
高知工科大学
橘　昌良
tachibana.masayoshi@kochi-tech.ac.jp

本章では，System on a Chip（以下，SoCと呼ぶ）[1]の設計をハードウェア設計者の観点から説明する（以下，ハードウェアをHW，ソフトウェアをSWと呼ぶ）．

図-1に示すように，HWの設計は，トランジスタの組合せによる設計（トランジスタレベル設計），トランジスタから作成したANDやORなどの論理素子を使用した設計（ゲートレベル設計），組合せ論理と記憶素子を意識したHW記述言語による設計（レジスタトランスファレベル設計（以下，RTLと呼ぶ））へと変遷してきた．さらに，SoCの大規模化，複雑化に対応するため，再利用設計，設計記述の抽象化と設計の自動化，HWとSWの同時設計により設計期間を短縮するための取り組みが行われている．再利用設計とは，使用実績のある設計部品を利用して設計を行うことを意味し，プラットフォーム設計手法[2),3)]が提案されている．また，設計記述の抽象化や設計の自動化についてはシステムレベル設計手法が提案されている．HWとSWの同時設計はHW/SWコデザイン[4),5)]として，従来から提案されていたが，システムレベル設計手法[1),3)]では従来手法より設計の早期段階からHWとSWの同時設計が可能となる．システムレベル設計手法は，アルゴリズムや方式を言語により記述することから始まり，既存のHW設計やSW設計へとつなげていく設計方法である．ここでは，システムレベル設計を実現する設計工程の流れ（システ

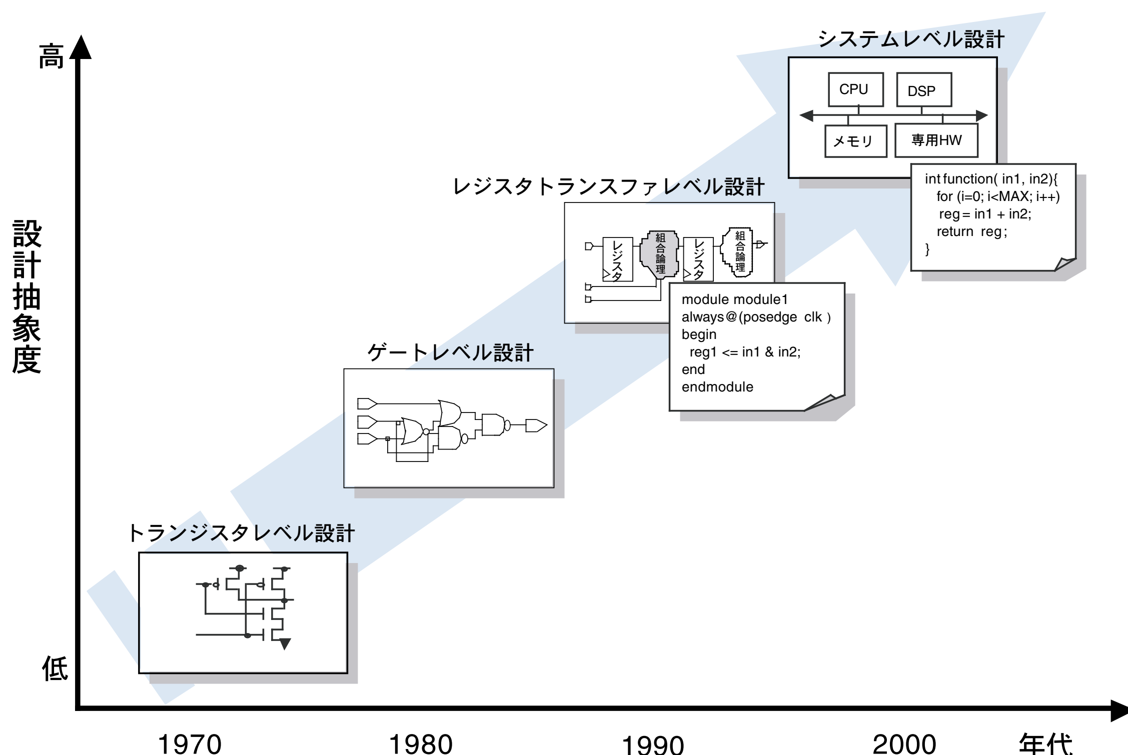



図-1　HW設計の変遷

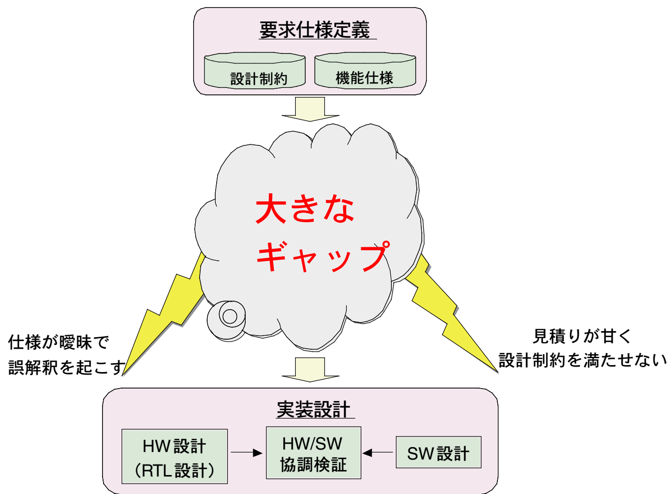



図-2　従来のSoC設計フロー

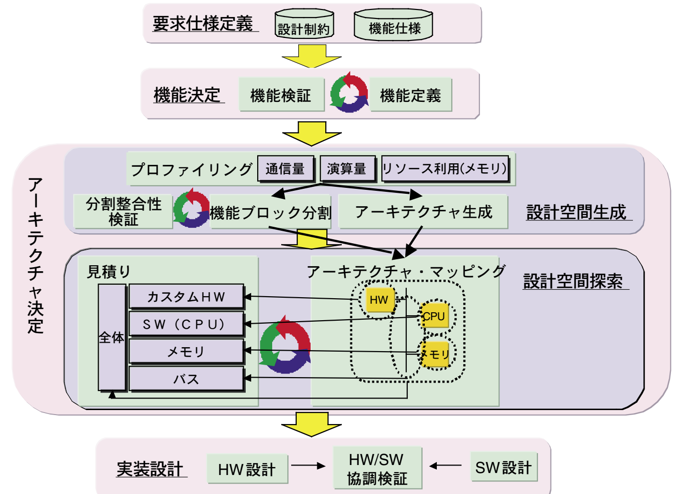



図-3　システムレベル設計フロー

ムレベル設計フロー）とそこに必要となる記述言語（システムレベル設計言語）について説明する．設計フローについては，さまざまな提案[6)～14)]があるが，ここではJEITA EDA技術専門委員会SLD研究会が提案した設計フロー[15)]を例に説明する．

## システムレベル設計フロー

SoCの設計は，SoCへの要求仕様の決定（「要求仕様定義」），実装方法の検討，HWの設計，SWの設計（「実装設計」）と進む．従来のSoC設計フローは，図-2のように，「要求仕様定義」フェーズと「実装設計」フェーズの間に大きなギャップがある．すなわち，この設計フェーズ間は自動化されていないため，設計者が仕様書を解釈し，実装を考えることになる．このギャップが，「仕様が曖昧で誤解釈を起こす」，「見積りが甘く，設計制約を満たせない」といったSoC設計上の問題を生み出す原因になっている．

そこで，このギャップを埋めるために，システムレベル設計フローでは，「要求仕様定義」から「実装設計」の間の設計フローを，図-3のように「機能決定」と「アーキテクチャ決定」の2つのフェーズで埋める．

### ■各設計工程の説明

#### (1)「要求仕様定義」フェーズ（ System Specification）

要求仕様とは，実現する機能や使用する規格を決める機能仕様とチップ面積，処理性能，消費電力などを決める設計制約を意味する．要求仕様定義とは，この要求仕様を設定することである．具体的な要求仕様は，SoCが搭載される製品の企画段階で決められる．決められた要求仕様は，製品の企画部門だけでなく設計部門でも共有できる表現形式で定義していく．現在，要求仕様は，自然言語（日本語や英語など）で記述される場合が多く，曖昧な表現や誤解釈による設計やり直しの原因にもなっている．仕様の曖昧さや漏れをなくすために，後述するUML（Unified Modeling Language）[27)]などの言語による定義も提案されている．

#### (2)「機能決定」フェーズ（ Functional Design ）

機能決定とは，要求仕様定義で決定した機能仕様と設計制約から実現する機能を決定することである．このフェーズではシステムレベル設計言語により実行可能な「機能モデル」を記述する（以下，機能定義と呼ぶ）．さらに，機能モデルをシミュレーションすることにより，機能仕様が実現されていることを検証する．このフェーズでは，SoCの実現方法については触れず，機能仕様を実現するための方式やアルゴリズムの検証に主眼を置く．また，計算精度についても検討する．たとえば，浮動小数点演算から固定小数点演算への変換やビット長の調整を行う．

#### (3)「アーキテクチャ決定」フェーズ（Architecture Design）

アーキテクチャ決定とは，機能決定フェーズで検証された機能仕様を，どのようなシステム・アーキテクチャで実現するかを決定することである．ここで述べるシステム・アーキテクチャとは，SoCを実現するHW構成やSW構成を意味する．HW構成は，CPUやDSPなどの汎用プロセッサ，メモリやメモリ制御回路，アプリケーションに特化した専用HWと各構成要素を接続するバスなどのことである．また，SW構成は，リアルタイムOS（RTOS），ミドルウェア，デバイスドライバなどHWに依存したSWなどのことである．従来のSoC設計手法では，システム設計者が，要求仕様を満たすアーキテクチャ構成を勘と経験をもとにした独自の方法で決定し，機能定義をHWで実現する機能とSWで実現する機能に分割していた．図-3に示すようにシステムレベル

設計手法では，これを「設計空間生成」と「設計空間探索」という2つのフェーズで実現する．

### (3-1)「設計空間生成」フェーズ

「設計空間生成」フェーズとは，機能ブロック間の通信量，機能ブロックのメモリ使用量や演算処理量を「プロファイリング」することで，「機能ブロック分割」と「アーキテクチャ生成」を行うことである．

<プロファイリング>

プロファイリングとは，変数のアクセス回数などのデータ通信量や演算の実行回数などの演算量を抽出することである．通信量や演算量を計測することで，機能ブロックの分割単位や機能ブロック間の接続方式を決める際の参考にする．たとえば，通信量の多い個所は同一ブロックとして統合する．また，高速性が必要なものは切り出し，並列化することにより最適な機能ブロック分割が可能になる．

<機能ブロック分割>

機能ブロック分割とは「機能決定」フェーズで定義した機能モデルを複数の機能ブロックに分割，または統合することを示す．さらに，分割した機能ブロック，および機能ブロック間の通信処理を行うモデルを作成する．機能ブロックを分割した際には，分割前後で機能が等価であることを検証（分割整合性検証と呼ぶ）する．各機能ブロックは，最終的に，専用HW，またはCPU上で実行するSWのいずれかで実装されることになる．

<アーキテクチャ生成>

アーキテクチャ生成とは，機能仕様を実現するシステムのアーキテクチャ構成を作成することである．このアーキテクチャ構成をモデル化したものを「アーキテクチャモデル」と呼ぶ．アーキテクチャモデルは，HW構成と各構成要素の接続関係と各構成要素の処理性能など性能見積りに必要な情報を持ったモデルである．この時点では，複数のアーキテクチャモデルをアーキテクチャ候補として作成しておく．

### (3-2)「設計空間探索」フェーズ

設計空間探索とは，機能モデルを動作できる複数のアーキテクチャ候補の中から最適なアーキテクチャを探索して選択することである．探索方法としては，各機能モデルをアーキテクチャモデルの構成要素へ割り付け（アーキテクチャマッピングと呼ぶ），処理性能や消費電力の見積り（詳細は，本特集の「低消費電力化設計と消費電力見積り」を参照）を行うことによって設計制約を満たす最適なアーキテクチャ構成を探索することができる．見積りでは，シミュレーションに代表される動的見積り手法や既存データの特性値や関数式を利用した静的見積り手法等を用いて，処理性能，コスト（たとえば，ゲート数やメモリサイズ），消費電力等の設計制約の各種項目を満たしているかを調べる．このようにアーキテクチャモデルと機能モデルを分離することで機能モデルを大きく変更することなくアーキテクチャマッピングを行える利点がある．

### (4)「実装設計」フェーズ（ HW/SW Implementation ）

実装設計とは，アーキテクチャを決定した後，専用HWの設計，CPUやDSPで実行するSW設計を行うことである．既存のSoC設計では，HW設計部門は，HW記述言語（以下，HDLと呼ぶ）で表現したRTLのモデルを論理合成と呼ぶ技術を用いて自動的にゲートレベルネットリストに変換することでHWの詳細化を行う．また，SW設計部門では，使用するCPUの動作をコンピュータ上で模擬した仮想シミュレータなどを利用してSWを開発している．開発したHWとSWを組み合わせて動作させるのは，チップを開発する前にHW/SW協調検証技術を使用して実現しているが，HW検証速度の遅さや検証環境準備に手がかかるなど問題があり，SoCを製造する前に十分な検証は難しかった．一方，システムレベル設計では，HWやCPUなどの設計モデルを既存設計より抽象度の高いモデルで表現できるため，高速なHWとSWの協調検証（詳細は，本特集の「ハードウェア／ソフトウェア協調シミュレーション技術」を参照）が可能となってきている．従来，製造後に検出したバグはSW変更で回避するか，HWの再設計・製造となっていた．システムレベル設計ではシステム検証が行える範囲が広がったことで，SoC製造前にバグの検出が行え，HWの修正も可能となる．HW設計は，動作合成（詳細は，本特集の「動作合成技術の動向」を参照）により，システムレベル設計言語で記述したモデルからRTLのモデルを自動生成し，以降，既存の設計フローにある論理合成技術によりゲートレベルのネットリストへと詳細化を行う．また，SW設計は，リアルタイムOS（RTOS）によるタスク制御，高級言語を処理するコンパイラやアセンブラコードの手書きによるCPU依存の最適化処理，デバイスドライバと呼ばれるHWとのインタフェース用SWの開発を行い，正常動作を確認する．

## システムレベル設計における検証

システムレベル設計における検証は，「機能決定」フェーズで行われる機能検証，「アーキテクチャ決定」フェーズで行われる機能ブロックを分割する前後の等価性検証（分割整合性検証），処理性能や消費電力などの設計制約を満たすかを見積るアーキテクチャ検証，「実

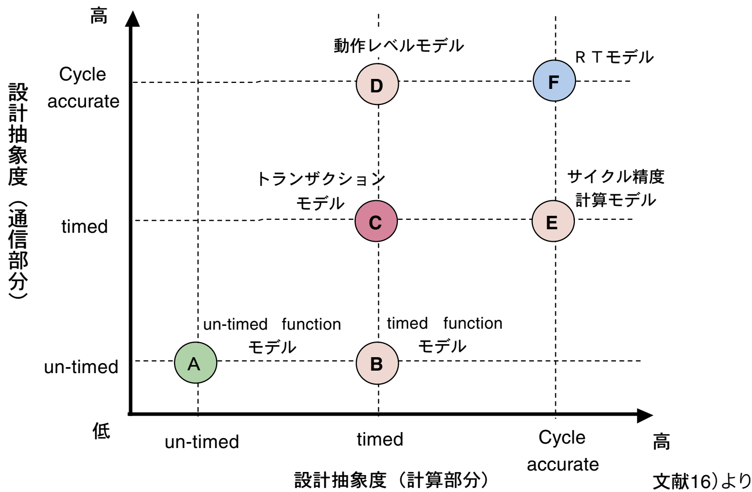



図-4　設計抽象度の分類例

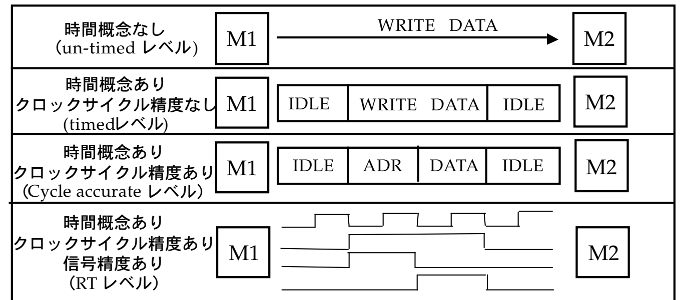



図-5　設計抽象度の例

装設計」フェーズで行われるHW設計やSW設計の正当性を確認するHW検証，SW検証，さらにHWとSWを含めたシステム全体の検証（HW/SW協調検証）からなる．「機能決定」や「アーキテクチャ設計」フェーズなど従来のSoC設計手法より早期の設計段階で機能バグや性能の未達成が検出できることで，設計の改善を行え，設計の後工程での問題検出を減らすことができる．

検証には，検証対象を表現した設計モデル，検証仕様，検証ツールが必要になる．設計モデルはシステムレベル設計言語で記述し，検証仕様はシステムレベル言語，テストベンチ記述言語，プロパティ記述言語，アサーション記述言語（詳細は，本特集の「アサーションベース検証」を参照），CPUやDSPなどのプロセッサ用のSW記述言語などにより記述する．

検証ツールは，シミュレーション技術による動的検証技術やフォーマル検証と呼ばれる静的検証技術が使用される．「機能決定」フェーズでは，アプリケーション分野別に，アルゴリズムを取り揃えた検証ライブラリが用意されているものもある．「アーキテクチャ決定」フェーズでは，バスの混雑度の観測やSWプログラムのプロファイラやデバック機能が必要となる．「実装設計」フェーズでは詳細なHWデバック機能が用意される．また，検証が目的を達成したかを確認する指標として検証カバレッジの導入，バスプロトコルなどの整合性を確認するための検証が行われる．検証は，設計フェーズの早期でデバックを行い，後工程での検証をなくすことが望まれるが，実際は，合成ツールの使用も含めた設計変更前後の検証，詳細なタイミング検証やデバイス依存のライブラリを使用した検証が必要になる．このため，設計モデルの精度，検証目的を把握した上で各設計フェーズでの検証を行うことが重要となる．

## 設計抽象度

設計モデルを記述することをモデリングと呼ぶ（詳細は，本特集の「システムのモデル化と計算モデル」を参照）．設計抽象度とは，モデリングの際に決める機能，動作，構造，通信方法，時間概念などの詳細度を示す．たとえば，実装する機能の網羅度，信号のビット幅や演算器の計算精度，機能ブロックの分割や機能ブロック間の接続表現の詳細度，機能ブロック間の通信方法に関する詳細度，時間概念の有無や精度などの詳細度がある．各情報の詳細度を高めれば，検証の詳細度や性能の見積り精度が高まるが，一方で設計モデル作成に時間がかかること，検証速度が遅くなることなどのトレードオフが生じる．したがって，各設計フェーズでの検証目的を考慮して設計抽象度を決めることが重要になる．

時間概念の抽象度は，たとえば図-4に示すように設計モデルの計算部分と通信部分をそれぞれ3つのカテゴリに分類できる．3つのカテゴリとは，時間概念がない抽象度（un-timedレベル），クロックサイクル精度はないが時間概念がある抽象度（timedレベル），クロックサイクル精度の時間概念がある抽象度（Cycle accurateレベル）である．最近，話題のトランザクションレベルモデルは，図-4のB，C，D，Eを呼ぶ[16]が，さらにF（信号のピン精度がない場合）を含む分類方法もある．

図-5では，設計抽象度の違いによるデータ転送の詳細度の違いを示す．un-timedレベルはデータを転送するのみだが，timedレベルになるとデータ転送する前後にIDLE処理が考慮される．さらに，Cycle accurateレベルでは，DATAが書き込まれるサイクルが実LSIの動作サイクルと一致する．RTレベルはサイクル精度についてはCycle accurateレベルと同様だが，信号端子情報がHWと一致する．「機能決定」フェーズではun-timedレベル，「アーキテクチャ決定」や「実装設計」フェーズでは，目的別にun-timedレベルからCycle accurateレ

ベルまでを用途に応じて使い分ける．un-timedレベルからCycle accurateレベルの設計モデルは，動作合成ツールにより，レジスタトランスファレベル（RTL）に詳細化される．

## システムレベル設計言語

システムレベル設計言語は，機能仕様や設計制約を記述するために使用する．システムレベル設計言語の使用により，以下のような利点が期待できる．

（A）設計の早期段階でシミュレーション可能なモデルで要求仕様を作成することにより，仕様誤りや誤解を取り除くことができる．

（B）設計抽象度の高い記述により，記述量が減る，検証速度が上がることにより，設計生産性があがる．

### ■システムレベル設計言語の要件

システムレベル設計言語には，システムレベル設計フローの各フェーズで求められる記述能力が求められる．たとえば，「機能決定」フェーズでは，アルゴリズムやビット精度の表現が可能なこと，「アーキテクチャ決定」から「実装設計」フェーズでは，HWを記述するためのデータ型，並列性，クロックなどの時間的な概念の表現や，SWを記述するための割り込み表現などをサポートすることが必要となる．また，設計抽象度別の表現能力，たとえば，通信部での配列表現，オブジェクト指向の継承概念を用いた抽象的な信号による接続，実デバイスと等価な信号単位での接続などの記述が必要となる．設計フェーズ別に異なる設計言語を使用することは設計者の負担にもつながるため，現在提案されている設計言語は，「機能決定」から「実装設計」までの工程を1つの言語で表現できることを意図したものが多い．

### ■言語動向[5), 15)]

従来，HW設計ではVHDL，Verilog-HDL[17)]を主とするHW記述言語（Hardware Description Language（HDL）），SW設計ではCやC++言語など標準的な設計言語が使用されてきた．しかし，設計対象の大規模化により，さらなる設計効率化と設計の再利用化が必要となり，標準化されたシステムレベル設計言語の重要性が増してきている．このような動きの中，1999年にはSystemC[18)]，SpecC[19)]などCやC++言語をベースとしたシステムレベル設計言語の標準化推進団体が発足した．また，2000年になりVHDL，Verilog-HDLなどのHW記述言語の標準化に深くかかわってきたVI（VHDL International）とOVI（Open Verilog International）の2つの団体が統合してAccellera[20)]を発足させ，SystemVerilog[21)]の標準化活動を行っている．

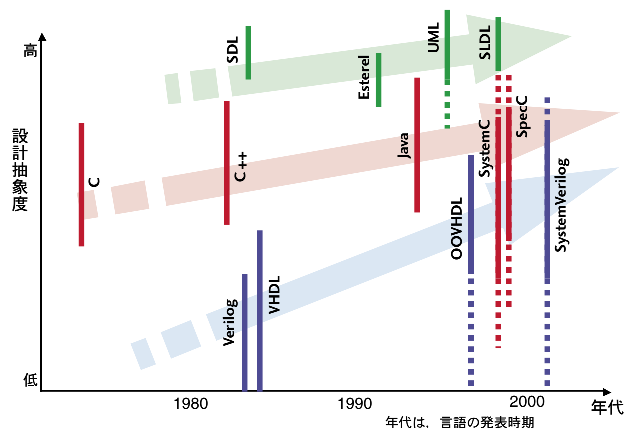



図-6　設計言語の変遷

図-6は，システムレベル設計への適用が試みられた既存の設計言語の動向を系統別に示した言語マップである．図-6の縦軸は，言語がサポートする設計抽象度，横軸は言語の発表時期を表している．システムレベル設計言語はアプローチ的に見て，大まかに（1）HW設計言語から派生した言語，（2）SW設計言語から派生した言語，（3）その他の新規言語の3つの系統に分類できる．全体的な傾向としては，年代とともに徐々に記述抽象度を広げ，適用範囲がシステム全体をカバーするように推移している．

#### （1）HW設計言語から派生した言語

VHDLやVerilog-HDLなど既存のHW記述言語を拡張してシステムレベルの記述を可能にした言語である．

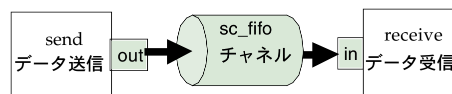
VHDLの言語拡張はIEEE DASC内で1996年頃から議論され，設計再利用性を高めることを目的にオブジェクト指向性を取り入れたOOVHDL（Object-Oriented Extensions to VHDL）[22)]の検討が行われてきた．

Verilog-HDLに関しては，2001年に規格改訂がVerilog2001として行われ，高い抽象レベルでシステムを記述するために必要な仕様が追加された．さらに2002年にSystemVerilog[21)]が提案されている．SystemVerilogでは，新たにモジュール間通信の抽象記述を可能とするインタフェースオブジェクト，設計モデルの検証に必要なテストベンチ構文の拡張，アサーション記述の追加を行っている．

#### （2）SW設計言語から派生した言語

CやC++言語をベースとした言語とJavaをベースとした言語が提案されている．どちらもHWを記述する仕組みを追加することにより，HW，SWの両方を記述できる言語となっている．

CやC++ベース言語は，既存のSWプログラム開発環




ブロック図

SystemC 記述

```
//   ブロック図に相当する接続構造を記述
#include < systemc.h >
#include " fifo.h "

int sc_main(int argc, char *argv[])
{
  sc_fifo<int> l_fifo;
  send  l_send("l_send ");
  receive l_receive("l_receive ");
  sc_clock clk("clk", 10, SC_NS, 5);

  l_send.out(l_fifo );
  l_send.clk(clk);
  l_receive.in(l_fifo );
  l_receive.clk(clk);

  sc_start(-1);
  return 0;
};
```

```
// send, receive の実装方式を記述
#include < systemc.h >
SC_MODULE(send ){
  sc_port< sc_fifo_out_if <int> > out;
  sc_in<bool>                     clk;
  void send_data ();

  SC_CTOR(send ){
    SC_THREAD(send_data  );   sensitive <<  clk.pos();
}};

void send::send_data (){
  int i=0;
  while(1){
    i++;
    out-> write(i);
    cout << "send data is " << i <<   endl;
}}

SC_MODULE(receive ){
  sc_port< sc_fifo_in_if <int> > in;
   sc_in<bool>                    clk;
  void receive_data ();

  SC_CTOR(receive ){
    SC_THREAD(receive_data  );   sensitive <<  clk.neg();
}};

void receive::receive_data (){
  while(1){
    cout << "received data is " <<  in->read() <<   endl;
}}
```

図-7　SystemCの例

境を利用できる点，多くのSW設計者やアルゴリズム開発者に利用されているため容易に習得できる点，動作合成ツールなどの開発ツールが他のシステムレベル設計言語に比べ進んでいる点などの利点がある．

C言語ベースのシステムレベル設計言語の研究は，1980年代後半に始まった．スタンフォード大学で開発されたHardware-C[23]，NECが開発したBDL[24]，シャープが開発したBach-C[25] などがある．1990年代後半になり，EDAベンダからもCやC++をベースにしたシステムレベル設計言語が提案されてきた．ANSI-C言語の仕様を拡張したものとしては，カリフォルニア大学アーバイン校で開発されたSpecC[19]， C++言語を拡張したものとしては，米国Synopsys社が中心になって開発したSystemC[18] がある．図-7にSystemCの記述例を示す．C++クラスライブラリで定義したモジュールと呼ぶ回路構造の記述，ポート，インタフェース，チャネルと呼ぶ通信定義の抽象的な記述が可能などの特徴がある．

一方，Javaベース言語としては，JavaTime[26]がある．Java言語は，Java仮想マシンとバイトコードによる標準的なプラットフォーム上で動作可能な点，ポインタの禁止による暗黙の並列性の識別と抽出が容易である点，マルチスレッドによる明確な並列性をサポートしている点などの利点があるが，C言語に比べ実行速度が遅い点が欠点である．

### (3) その他の言語

UML（Unified Modeling Language）[27] は1996年に3つのオブジェクト指向方法論を統合したモデリング言語として考案され，OMG（Object Modeling Group）により標準化が行われている．システムの機能と外界との関係を示すユースケース図，オブジェクトの構造を記述するクラス図，各オブジェクト間のメッセージのやりとりを記述するシーケンス図，状態遷移を表現するステートチャート図などを使用してシステムを記述する．元来，SW開発におけるシステム記述のための言語であったが，構造図を加えてHWを含めたシステムレベル記述に適用する動きがでている．

SDL（Specification and Description Language）[28] は，通信プロトコル仕様を記述するために開発された言語でITU-T（国際電機連合の電気通信標準化部門）の標準規格となっている．SDLからC言語，VHDLを生成するツール，UMLのグラフィカル記述を組み込み，解析，設計，テストをサポートするツールもある．

Esterel[29] は1980年代初期にリアルタイムシステムのプログラム言語として開発され，1990年代になりHW記述のための変更が行われた．POLIS[6] システムの言語として使用されている．

SLDL（Systems Level Design Language）[30] は，設計制約が記述できる点，複数の設計言語で記述されたシステムを取り込み，統合できる仕組み（Bridging Semantics）を持つ点が他の言語と異なる．Accelleraで活動を継続している．

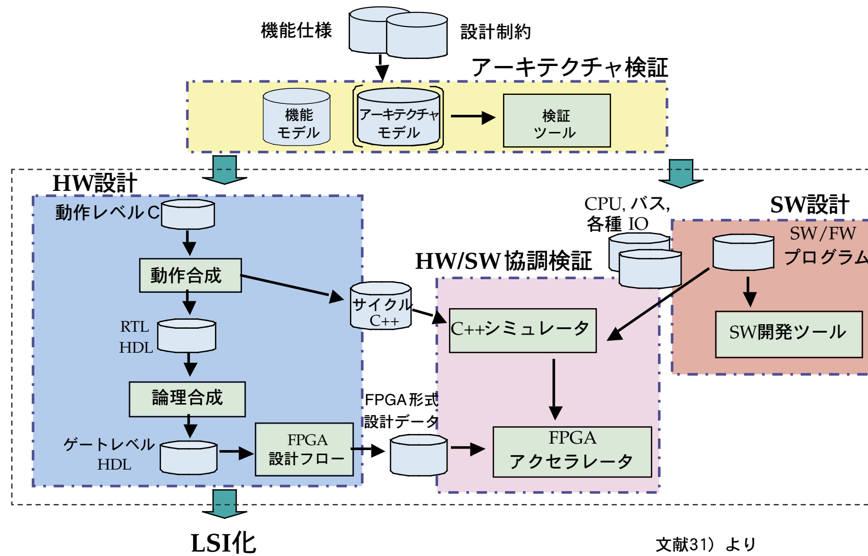



図-8　システムレベル設計フローの例

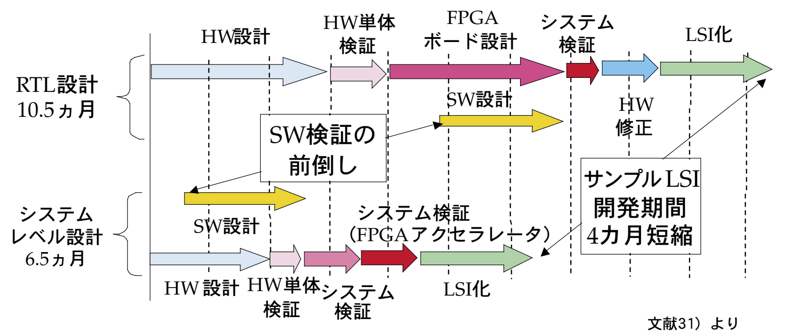



図-9　システムレベル設計の効果

## システムレベル設計の効果

ここ数年，画像コーデックや携帯電話端末向けなどのSoC設計にシステムレベル設計を適用した事例が報告されている[31)～33)]．たとえば，携帯電話端末のSoC開発では図-8に示すシステムレベル設計環境を使用し，HW設計開始からLSIサンプルチップができあがるまでの開発期間を従来のRTL設計手法に比べ4カ月短縮した[32)]．図-9が示すように，動作合成によるHW設計の効率化とSW検証の前倒しにより，LSI化の前にHWとSWを含めたシステム検証とデバックが行える利点がある．

図-8に示すように，この例では，従来行っていた机上のアーキテクチャ検証に加え，C++モデルとツールによる性能見積りを実施し，バス構成の違いによる画像と音声の複数処理時の性能比較を行った．また，HW設計は，動作合成ツールを使用し，CモデルからHW実装用のRTLモデルと検証用のクロックサイクル精度のC++モデルの自動生成を行うことでモデル作成時間を効率化した．さらに，C++シミュレータとFPGAボードを使用した高速シミュレータを使用し，既存のRTLモデルを含むシステム検証を高速に行うことでLSI製造前にSWを含むシステム検証を行うことができ，リスピン（設計ミスなどによるLSI製造のやり直し）なしにLSI設計を行えた．

別の事例としては，複数のプロセッサを含む複雑なSoCをSystemCでモデル化し性能確認，HW/SW協調検証をLSIサンプルチップができあがる前に実施し，50KCPS（1秒間に50Kサイクル実行）というHWエミュレータに迫る高速シミュレーションを達成した例[33)]などがある．

システムレベル設計の効果は，HW設計の効率化と検証の効率化にある．HW設計では，システムレベル設

計言語によるHWの記述量が従来のHDL記述に比べ少なくてすむこと，動作合成によりHDLの自動生成が可能なため，システムレベルとRTレベルのモデル作成の二度手間をなくせるなどの利点がある．また，検証については，RTLのHDL検証では時間がかかっていた画像処理などのHW検証を高速に行えること，さらにSWを含めたシステム検証を高速に行えるなどの利点がある．これにより，デバイスドライバなどHWに依存するSW検証が行え，LSI製造前にHWを修正することも可能となった．このような設計効率化を実現するためには，単にシステムレベル設計やツールを導入するだけではなく，適切な検証目的の設定，設計対象のモデル化技術，効率的な検証環境構築技術が重要になる．

## システムレベル設計の今後の課題

システムレベル設計は新しい設計方法であり，さらなる設計効率化の実現，本設計手法の普及や定着のためには以下に示すような課題がある．しかし，大規模化，複雑化するSoCを設計するためにはシステムレベル設計は必要不可欠な技術であり，産学の協力によりこれらの技術課題を解決していくべきと考えている．

**＜課題の例＞**

- 要求仕様定義から機能決定に至る設計手法の確立
- プロファイリング技術を中核とするアーキテクチャ設計技術の確立
- システムレベル設計言語の標準化とモデリングガイドラインの策定
- 動作合成技術やインタフェース生成技術の成熟と利用技術の展開
- システムレベルにおけるアサーション検証，フォーマル検証，HW/SW協調検証など検証技術の確立
- OS，ドライバSW，ミドルウェアなどSWも含めたシステムレベル検証技術の成熟と利用技術の展開
- レイアウト設計などデバイスに依存した設計を考慮した設計フロー構築

**謝辞**　本稿の執筆にあたって，JEITA EDA技術専門委員会SLD研究会の温兆祺，荒木大，齊藤博文，野々垣直浩，大塚直人，塚本泰隆，吉永和弘の各氏のご協力をいただきました．また，今井正治（大阪大学），吉田紀彦（埼玉大学），神戸尚志（近畿大学）の各先生のご協力をいただいたことに感謝いたします．

**参考文献**

＜システムオンチップの設計＞
1）麻生他：システムLSIのすべて，2000年5月，工業調査会．
＜プラットフォーム設計手法＞
2）Ferrari, A., Sangiovanni-Vincentelli, A. :System Design: Traditional Concepts and New Paradigms, ICCAD'99, pp.2-12（1999）.
3）Kentzer, K., Malik, S. et al. : System-Level Design: Orthogonalization of Concerns and Platform-Based Design, IEEE Trans. On CAD, pp.1523-1543（2000）.
＜HW/SWコデザイン＞
4）今井，松永他：特集「ハードウェア／ソフトウェア・コデザイン」，pp.604-632，情報処理，Vol.36, No.7（July 1995）.
5）Staunstrup, J., Wolf, W. et al: Hardware/Software Co-Design, Kluwer Academic Publishers（1997）.
＜システムレベル設計手法＞
6）POLIS:
http://www-cad.eecs.berkeley.edu/Respep/Research/hsc/abstract.html
7）OCAPI-XL: http://www.imec.be/design/ocapi/
8）Vanmeerbeeck, G., Schaumont, P. et al.: Hardware/Software Patitioning for Embedded System in OCAPI-xl, CODES'01（Apr. 2001）.
9）Chinook: http://www.cs.washington.edu/research/chinook/
10）SoC Environment（SCE）：
http://www.cecs.uci.edu/~cad/sce.html
11）TIMA System Level Synthesis : http://tima.imag.fr/SLS/
12）CoWare N2C: http://www.coware.com
13）CoCentric System Stdio: http://www.synopsys.com/products/cocentric_studio
14）Liao, S., Tjiang, S. et al.: An Efficient Implementation of Reactivity for Modeling Hardware in the Scenic Design Environment, DAC'97 .
15）JEITA EDA技術専門委員会SLD研究会：システムレベル設計手法の提案とシステムレベル設計言語の適用化検討，2001年5月，
http://eda.ics.es.osaka-u.ac.jp/jeita/eda/project/sld/public-j2001/
＜設計抽象度＞
16）Cai, L. and Gajski, D. :A Transaction Level Modeling: An Overview, First IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, pp.19-24, Newport Beach, CA（Oct. 2003 ）, http://www.ics.uci.edu/~cecs/conference_proceedings/isss_2003/cai_transaction.pdf
＜システムレベル設計言語＞
17）VHDL, Verilog-HDL:http://eda.org/
18）SystemC: http://www.systemc.org/
19）SpecC: http://www.specc.org/
20）Accellera: http://www.accellera.org/
21）SystemVerilog: http://www.systemverilog.org/
22）OOVHDL: http://www.vhdl.org/oovhdl/
23）Hardware-C:
Camposano, R. and Wolf, W. : High Level VLSI Synthesis, pp.127-151, Kluwer Academic Publishers（1991）.
24）BDL:
Wakabayashi, K.:C-based Synthesis Experiences with a Behavior Synthesizer “Cyber”，Proc. of DATE'99, pp.390-393（1999）.
25）Bach-C:
Kambe, T., Yamada, A. et al. :A C-based Synthesis System, Bach and Its Application, Proc. ASP- DAC2001, pp.151-155（2001）.
26）JavaTime: http://www-cad.eecs.berkeley.edu/~jimy/research/cadlunch.4.98/index.htm
27）UML：http://www.uml.org/
28）SDL：http://www.sdl-forum.org/
29）Esterel：http://www-sop.inria.fr/esterel.org
30）SLDL：http://www.sldl.org/
＜設計事例＞
31）プログラムがそのままチップになる，日経エレクトロニクス，pp.104-133（July 2002）.
32）黒坂，宮本 他：C言語ベース設計手法による携帯電話用アプリケーションLSI設計事例，NEC技報，Vol.56, No.4, pp.160-163（Apr. 2003）.
33）竹村：SystemCを利用したシステムシミュレータ開発事例，第3回日本SystemCユーザフォーラム2003（Jan. 2003）．http://www.systemc.org/projects/sitedocs/document/edsf_2003_Takemura

（平成16年4月6日受付）