

## 4. 提案するシステムレベル設計手法

### 4.1. 対象とするシステム LSI の特徴

提案するシステムレベル設計手法について述べる前に、この設計手法が対象とするシステム LSI の特徴を以下のように定義する。

- 大規模 / 複雑で、ハードウェアやソフトウェアなどの性質の異なる要素で構成されているチップであること
- 多機能化 / 高性能化が進み、システム全体での最適化が重要であること
- 多品種少量生産で、開発期間の短 TAT が非常に重要であること
- 上記の特徴を満足するために、IP の利用比率が非常に高いチップであること

### 4.2. 提案するシステムレベル設計フロー

#### 4.2.1. 概要

既存のシステムレベル設計フローは、図 4.2-1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。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、リスピンを起こす」

といったシステムレベル設計に関する様々な問題や課題を生み出す原因になっている。

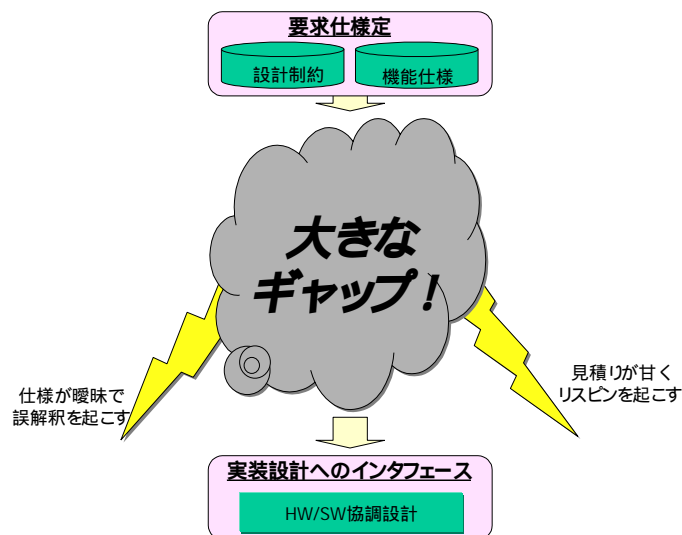


図 4.2-1 既存のシステムレベル設計フロー

そこで、提案する設計フローでは、このギャップを埋めるために、「要求仕様定義」から「実装設計」の間の設計フローを、図 4.2-2のように明確に定義していくことに

する。すなわち、このギャップを「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。

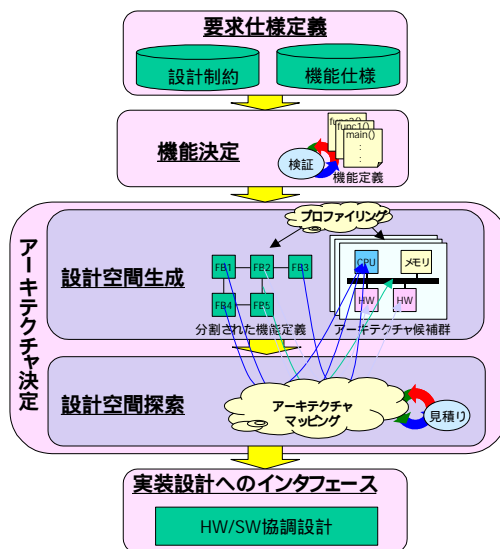


図 4.2-2 提案するシステムレベル設計フローの概念図

機能決定フェーズでは、要求仕様定義で決定したシステムの機能仕様をシステムレベル設計言語により定義（機能定義と呼ぶ）し、シミュレーションにより検証する。このフェーズでシステムの機能仕様とそれを表現した機能定義との妥当性が評価される。

アーキテクチャ決定フェーズでは、機能決定フェーズで検証された機能定義を実現するときに、どのようなシステム・アーキテクチャで実現するのかを決定する。このフェーズは、「設計空間生成」と「設計空間探索」の2つに分けることができる。

設計空間生成フェーズでは、後の設計空間探索の対象となる「分割された機能定義」と「アーキテクチャ候補群」を生成する。この作業は、機能決定フェーズでの機能定義から、「4.2.4 設計空間生成」で説明する「プロファイリング」を用いて収集した情報をもとに行われる。

設計空間探索フェーズでは、設計空間生成で作成された、分割された機能定義をアーキテクチャ候補群から選ばれたアーキテクチャにマッピングし、シミュレーション等でその性能を見積る。そして、このアーキテクチャ候補の選択とマッピングを繰り返し、与えられた「設計制約」を満たす最適なアーキテクチャを探索、決定する。最適アーキテクチャが決定されると、それをもとに、手動、あるいは、動作合成ツールやインターフェース合成ツールなどを用いて、RT レベルのハードウェア・コードや組み込み用ソフトウェア・コードが自動的に生成される。

以上をフローで示すと、図 4.2-3のようになる。

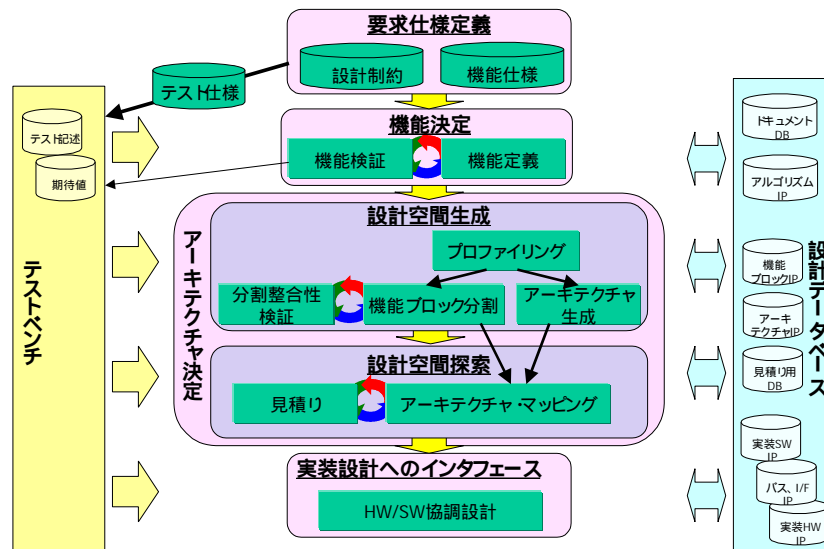


図 4.2-3 提案するシステムレベル設計フロー

この設計フローでは、要求仕様定義から導かれたテスト仕様や機能検証の結果を期待値として用いてテストベンチを作成する。それを設計フロー全体にわたって適用することにより、フロー全体での整合性を保証しようとする。

また、各設計フェーズでの IP や見積りデータベースなどを関連付けてまとめた「設計データベース」を用意することにより、フロー全体で過去の設計資産の再利用性を高めようとしている。

以上が、SLD 研究会が提案するシステムレベル設計フローである。この設計フローの各フェーズの詳細については、次節以降（4.2.2～4.2.8）で説明していく。

#### 4.2.2. 要求仕様定義

要求仕様定義フェーズでは、対象システムに対する要求項目として、図 4.2-4に示すような項目が定義される。

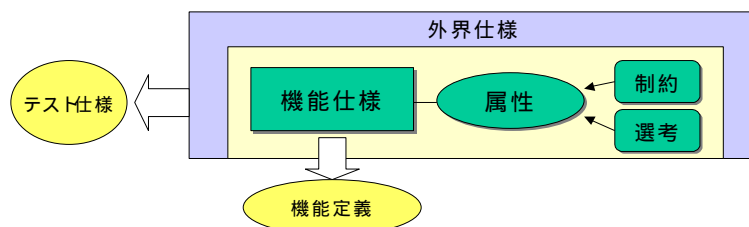


図 4.2-4 要求仕様定義

すなわち、そのシステムが存在する環境（外界）とシステムがどのように関連するかを示した「外界仕様」、システムが持つべき機能を定義した「機能仕様」、また、その機能仕様に対する属性として「制約」や「選考」などを定義する<sup>[1]</sup>。制約とはそのシステムが絶対に守らなければならない必須条件で、これを満たさないシステムは受け入れられない。これに対して選考とは望ましいが選択可能な条件である。すべての制約を満たすシステムは、与えられた要求仕様に対してどれも許容可能なシステムになる。しかし、あるシステムは、他のシステムよりも好ましいということがある。選考によって、設計者は許容可能なシステムを比較し、よりよいほうを選択することができる。そして、ここで定義された要求仕様をもとに、後工程で機能定義やテスト仕様の定義が行われる。

これらの要求仕様定義は、そのシステムを実際に設計する部門だけでなく、例えば、商品企画部門やカスタマーなどの間でも参照される。従って、このフェーズでは、非エンジニア系の人でも理解できるような、自然言語に近い表現形式の要求仕様定義言語をサポートしていくことが必要になると思われる。

#### 4.2.3. 機能決定

機能決定フェーズでは、図 4.2-5に示すように、要求仕様定義フェーズで決定したシステムの機能仕様をシステムレベル設計言語により定義（機能定義と呼ぶ）し、それをシミュレーションにより検証する。これにより、システムの機能仕様とそれを表現した機能定義との妥当性を評価することができる。

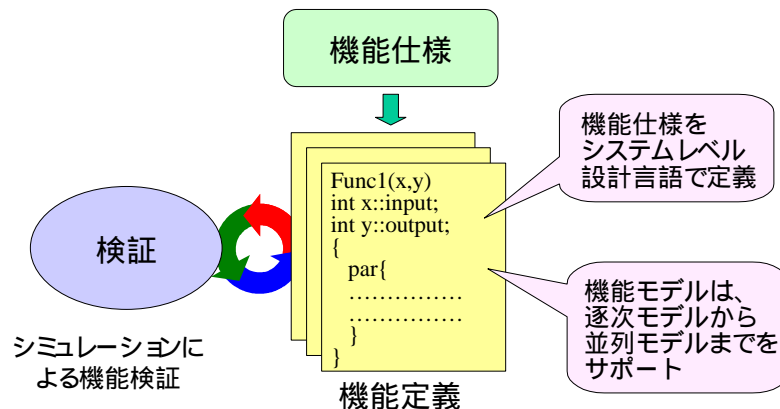


図 4.2-5 機能決定

機能定義はシステムレベル設計言語で記述されるが、その表現の対象となるモデルを機能モデルと呼ぶことにする。機能モデルは、最も単純な逐次（シーケンシャル）モデルから複数の機能を同時に動作させることを定義する並列（コンカレンシ）モデ

ルも含めて扱うことができるものとする<sup>1</sup>。このフェーズでの並列モデルの導入は、並列性を持った実際のシステムを表現しやすくするために導入する。すなわち、このフェーズの並列モデルでは、性能に関連する並列性の検討は行わず、機能のみに注目した検証を行う。これは、システムの性能に関する検討は、後の「アーキテクチャ決定」で、プロファイリングやアーキテクチャマッピングを用いた検証で行えるようにするためである。

#### 4.2.4. 設計空間生成

図 4.2-6に示すように、設計空間生成フェーズでは、機能ブロック分割とアーキテクチャ生成を行う。機能ブロック分割とは機能モデルを複数のブロックに分割、または統合することにより作成した各機能ブロックと各ブロック間の通信で定義されたモデル（「プロセスモデル」と呼ぶ）を作成することである。各機能ブロックは、最終的に、ハードウェア、またはソフトウェアのいずれかで実装されることになる。これらの機能ブロックが、逐次的に実行されるか、並列に実行されるかは処理の依存関係、およびシステムのアーキテクチャに依存する。一方、アーキテクチャ生成とは、機能モデルを実現するための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示すモデルを「アーキテクチャモデル」と呼ぶ。

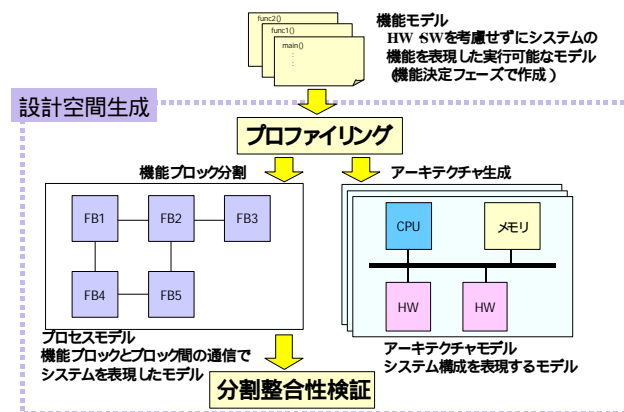


図 4.2-6 設計空間生成

<sup>1</sup> 機能モデルにおいて、並列モデルを導入するかどうかは、研究会内でも意見が分かれるところである。実際のシステムではそのほとんどが並列性を内在しており、並列モデルを導入したほうが、実際のシステムをモデル化することが容易になる。しかし、機能レベルで並列性をもつということは、性能に影響を与えるなんらかの要因を持つことになり、それが後段のアーキテクチャ決定フェーズでの選択肢を狭めることになる可能性がある。従って、研究会が提案する機能モデルで並列性を導入するかどうかは、さらに議論を行うべきである。しかし、時間的な制約からこの議論は今後の課題とし、本技術報告書では、システムの表現のしやすさを考慮し、機能モデルに並列性を導入する。そして、それを後段のアーキテクチャ決定フェーズで、性能に注目した並列性に切りなおし（並列な部分をマージして逐次化したり、逆に、逐次な部分を性能を考慮し並列化したり等）を行うということで話を進めていくことにする。

「アーキテクチャモデル」は、プロセッサ、メモリ、ハードウェアブロックやそれらの接続関係でシステム構成を表現した実装モデルである。この時点では、最適なアーキテクチャモデルを決めることはできないので、複数のアーキテクチャ候補を考えておき、後の設計空間探索フェーズでの見積り結果より最適なアーキテクチャを決定する。このように、機能モデルからプロセスモデルとアーキテクチャモデルを作成するためには、ブロック分割の粒度、分割の切れ目を決定するための判断基準が必要になる。また、機能モデルからアーキテクチャモデルを作成するためには、おおよその処理量と設計制約との対比が必要である。このように、機能ブロック分割とアーキテクチャ生成のための指標を得るために、「プロファイリング」と呼ぶ手法を利用する。また、「プロセスモデル」は、機能モデルと機能的に等価であることを検証する必要がある。また、分割が意図通りに行われたかを確認するために、各ブロックの処理量やブロック間の処理量を確認する必要がある。このようなプロセスモデルの確認は、「分割整合性検証」と呼ぶ処理で行う。

プロファイリングとは、機能モデルを動的あるいは静的に解析し、プロセスモデルやアーキテクチャモデルを作成するために必要となる指標を行うことである。このために抽出する情報とその利用方法の一例について表 4.2-1に示す。

表 4.2-1 プロファイリングで抽出する情報とその利用例

指標	抽出する情報の例	設計空間生成での利用方法
変数アクセス回数	<ul style="list-style-type: none"> <li>・Read/Writeの回数</li> <li>・アクセスされる箇所</li> <li>・同一変数のReadとWriteの距離</li> </ul>	<ul style="list-style-type: none"> <li>・アルゴリズム変更判断</li> <li>・分割単位の候補決定</li> <li>・メモリ レジスタの選択</li> </ul>
データ転送量	<ul style="list-style-type: none"> <li>・Read/Writeのデータビット数 * 実行回数</li> <li>・関数引数のデータ量</li> </ul>	<ul style="list-style-type: none"> <li>・分割単位の候補決定</li> <li>・HW / SW分割の候補決定</li> <li>・通信プロトコル選択</li> </ul>
演算量	<ul style="list-style-type: none"> <li>・行単位での演算回数</li> <li>・ブロック単位での演算回数</li> <li>・関数単位での実行回数</li> </ul>	<ul style="list-style-type: none"> <li>・分割単位の候補決定</li> <li>・HWやSW上での性能見積り</li> </ul>
統計解析	<ul style="list-style-type: none"> <li>・演算 / 制御命令の種類と実行回数</li> </ul>	<ul style="list-style-type: none"> <li>・プロセッサの選択</li> <li>・命令セットの追加</li> </ul>

プロファイリングで抽出する情報には、例えば、変数のアクセス回数やデータ転送量、演算量などがある。情報は、局所的にみる場合や大局的にシステム全体にわたって抽出する場合があります。例えば、システム全体を通してどの種の演算命令が多いかなどの統計解析を行うこともある。プロファイリングで得られた情報は、プロセスモデル、アーキテクチャモデルを作成するために利用される。処理量や通信量を計測することで、分割単位や接続方式を決める際の参考にできる。以下に、プロファイリングで抽出する情報とその利用例について説明する。

- 変数アクセス回数

ある変数のアクセス回数が非常に多いと、この部分が処理のボトルネックになる場合がある。この際、性能を満足するためには、変数を複数の変数に分割することで処理を分散させたり、アルゴリズムそのものを変更したり、アクセス回数を減らす工夫が必要となる。同一変数にアクセスする処理は、複数プロセスよりも同一プロセスに納めた方が良いので、同一ブロックに割り当てるなど機能ブロック分割の判断材料となる。また、同一変数へのアクセス間隔が短い場合、高速な処理が必要となり、メモリではなくレジスタでの実現を検討する必要がある。

- データ転送量

データ転送量の多い部分を 1 ブロックにまとめるなどの機能ブロック分割時の指標や分割した場合の通信プロトコルを選ぶ基準として使用される。

- 演算量

関数や命令毎の実行回数をカウントし、1 ブロックへの演算集中を避けるなどの機能ブロック分割時の指標や専用ハードウェア化を検討するなどのアーキテクチャ構成を検討する際の指標として使用される。

- 統計解析

処理を高速化するために、演算の使用頻度をカウントし、クリティカルな演算を高速化できるプロセッサを選択したり、プロセッサの命令セット追加を検討したりするために利用される。

#### 4.2.5. 設計空間探索

図 4.2-3 に示す設計空間探索フェーズでは、機能ブロック分割した結果をアーキテクチャにマッピングして見積りを行い、設計制約を満足する最適なアーキテクチャを選択することを行う。もう少し詳細に説明すると、図 4.2-3 に示した設計空間生成で作成されたプロセスモデルをアーキテクチャ候補群から選んだアーキテクチャモデルにマッピングし、シミュレーションに代表される動の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、性能、コスト（ゲートやメモリサイズ）、消費電力等といった設計制約の各種項目が要求を満しているか見積ることである。このアーキテクチャ候補群からの選択とマッピングを繰り返し実施することにより、与えられた「設計制約」を満たすアーキテクチャを探索し、最終的にシステムとしての満足解を決定する。

この見積り評価は、非常に高速かつ高精度に行うことが理想である。しかし、通常の Verilog-HDL や VHDL を利用した詳細シミュレーションを使っでの設計制約評価は、

非常に多くの探索時間がかかるため満足解を決定するのが困難となる。

この問題に対する解決案の一つとして、我々は、機能を表現したプロセスモデルの代わりに、見積り用の数値もしくは関数式で表現したトランザクションモデルと呼ばれるモデルを検討した。ここで言う見積り用の数値や関数式とは、性能を見積る場合は処理量、ゲート規模を見積るときはハードウェアのゲート数やソフトウェアのメモリサイズ、電力を見積るときはハードウェアのオン/オフの変化総量やプロセッサでの命令数等が考えられる。以下の説明では、これらの表現を便宜上、処理量という言葉を使って説明する。

トランザクションモデルは、プロセスモデルと同じ構造を用い、プロセスモデルで表現されていた各機能ブロックに設計制約毎の性能、コスト、消費電力値等の処理量を埋め込むことで表現する。実際の見積りは、各処理量と実現するアーキテクチャの関係によって決定される。本モデルの導入により、機能と設計制約を分離し、設計制約のみに着目した高速な見積りを可能にすることを目指している。

ここに、トランザクションモデルを用いた最も単純な見積り方法を、性能見積りを例に取り以下に説明する。

今、各機能ブロックの処理量が入出力に依存せず一定値で表すことができるトランザクションモデルを考える。単位時間当たりの処理能力は、アーキテクチャモデルにより得られるので、アーキテクチャにマッピングされた機能ブロックの処理時間見積りは、下記の算式で求めることができる。そして、システム全体の性能は、これらを加算することで算出可能となる。

$$\text{処理時間見積り} = \text{処理量} \div (\text{単位時間あたりの処理能力})$$

図 4.2-7 の例において、性能見積りを例にとって説明する。各機能ブロック FB1、FB2、FB3、FB4、FB5 の処理量を各々10 とする。ハードウェアで実現する場合の単位時間処理能力を 10、ソフトウェア（図では CPU にマッピング）で実現する場合の単位時間処理能力を 1 とした場合、ソフトウェアにマッピングする FB1、FB2 の処理時間見積りは、各々  $10 \div 1=10$ 、ハードウェアにマッピングする FB3、FB4、FB5 の処理時間見積りは、各々  $10 \div 10=1$  と見積れる。

今、各機能ブロックの処理順序について、FB2 は FB1 の処理完了を受けて動作し、FB3 は FB2 の処理完了を受けて動作するものとする。また、FB4、FB5 は、他の機能ブロックとは独立に処理可能とすると、処理時間見積りは図 4.2-8 のようになる。実際、処理時間だけ最小にするには全部ハードウェア化すれば良いが、コストや消費電力面とのトレードオフの関係からシステム全体として最適かどうかはわからない。また、時間面でネックになる FB1 と FB2 を平行に処理できるなら、処理時間は半分



以下になることが見積れる。実際には、見積りたい制約条件ごとのトランザクションモデルを用意し、システム全体として満足できる解をこのフェーズで探索することになる。

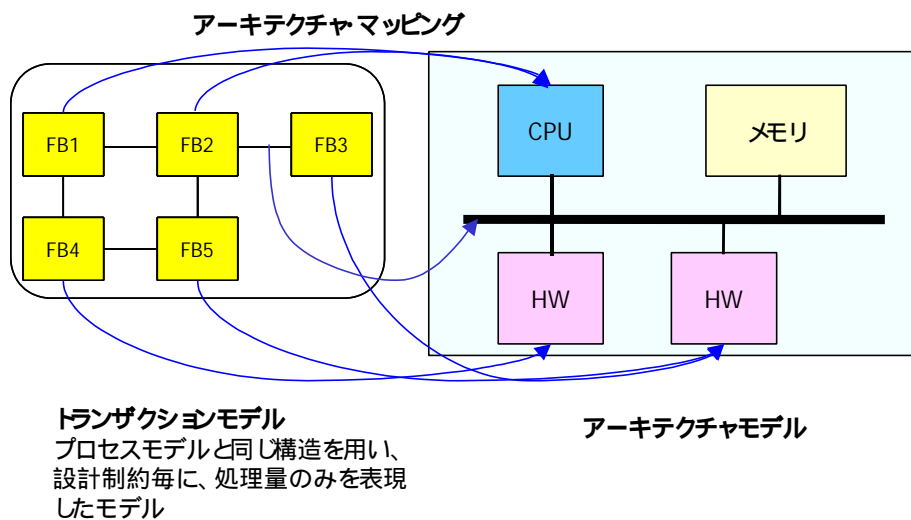


図 4.2-7 設計空間探索

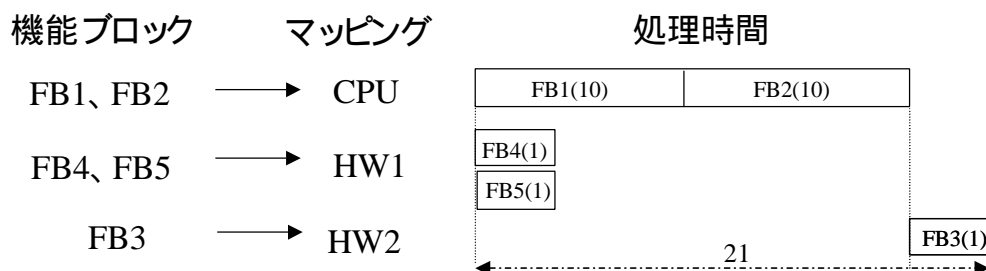


図 4.2-8 処理時間見積り例

前述の例で、粗い精度のトランザクションモデルを利用した見積りの例を示したが、さらに細かな精度で記述するトランザクションモデルも存在する。図 4.2-9 は、トランザクションモデルの精度を色々に変えた時の方法とその特徴を示したものである。先の例で示した性能見積りは、設計データを使った平均特性値を使って加算評価を行った例であるが、非常に高速に見積れる反面、処理データに依存した性能に関しては詳細に見積れていない。ソフトウェアの性能を見積る場合の一例として、命令セットレベルにまで落とすやり方が考えられるが、やはり組込みソフトウェアが完成し、ターゲットプロセッサまで完全に特定しないと精度が上げられないという問題は残る。ソフトウェアでは、Cadence 社の VCC システムの場合、ターゲットプロセッサ独立の仮想命令セットである VPM(バーチャルプロセッサモデル)を規定することで中間的

なレベルの見積りを行えるような工夫がされている。ハードウェアでは、SystemC2.0の場合、時間を考慮しない機能モデル UTF とサイクル精度の RTL モデルとの中間に、時間を考慮した TF や、バスサイクルレベルの BCA を導入して中間的な精度の見積りを工夫している。

このように、検証の精度と速度にはトレードオフの関係があるが、現実には、ある程度の精度を確保し、高速に見積れるような手法の確立が望まれる。また、探索レベルによって粗い精度で高速に見積って探索空間を限定し、高精度な見積りで絞り込んでいくといった方法も併せて検討していかなければならない。

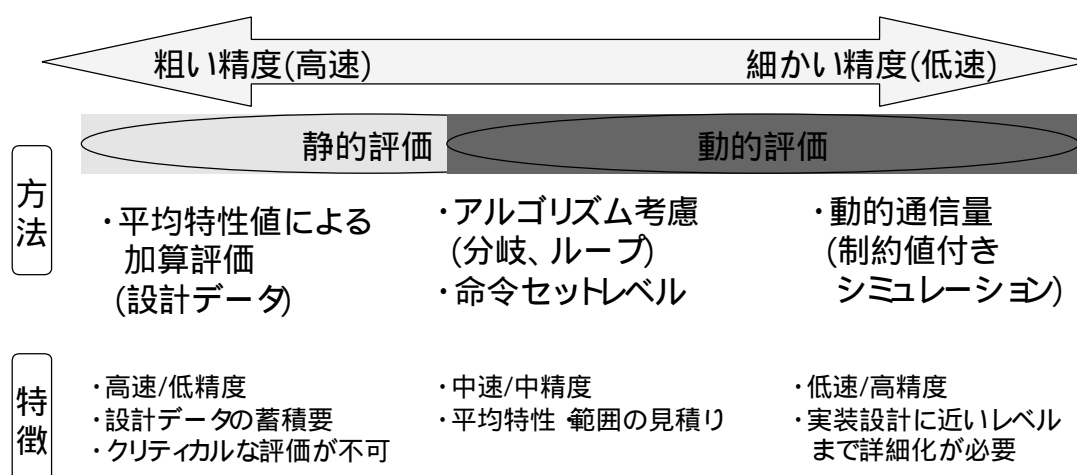


図 4.2-9 トランザクションモデルと精度

#### 4.2.6. 実装設計へのインタフェース

アーキテクチャが決定されると、ハードウェア部の設計、ソフトウェア部の設計に進む。また、ハードウェア部とバスとの接続部分や、デバイスドライバなどのインタフェース部分についての設計も行われる。ハードウェア部の設計は、動作合成と論理合成技術を使用することにより、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、さらにゲートレベルのモデルを自動生成することが可能となる。ソフトウェア部の設計は、RTOS によるタスク制御の設定、プロセッサに依存した最適化処理、ハードウェアとのインタフェース部の開発などを行い、正常動作を確認することからなる。インタフェース部の設計は、協調設計ツールにより自動生成できる場合もある。実装設計へのインタフェース部で開発された部分の動作確認には、ハードウェア部、ソフトウェア部、インタフェース部を含むシステムをハードウェア/ソフトウェア協

調検証ツールなどにより検証する。

#### 4.2.7. テストベンチ

システム LSI の機能検証、タイミング検証を動的に行うために、テストベンチが必要となる。まず、テストベンチは、テスト仕様書をもとにシステムレベル設計言語で記述される。そして、テストベンチを使用して、機能定義したシステムを機能決定フェーズで検証し、その結果を期待値として保管する。テストパターンと期待値は、アーキテクチャ決定や実装設計へのインタフェースなどでの検証に使用されるよう、再利用性を考慮する必要がある。テストベンチは、機能テスト用のパターン生成部、検証結果の正当性を確認するための結果確認部から構成される。これらの部分を記述するために、システムレベル設計言語は、パターン生成に必要な機能を記述するだけでなく、サイクル情報等のタイミング情報を記述できる必要がある。例えば、2 サイクル後から 5 サイクル後の間に、値が変化することを確認するなどの時相論理と呼ばれる時間概念の記述が必要となる。

#### 4.2.8. 設計データベース

システムレベル設計の効率化の手法として、過去の設計資産を利用した再利用設計が考えられる。ハードウェア開発では IP ベース設計、ソフトウェア開発ではオブジェクト指向技術を使った再利用設計が普及してきているが、上流のシステムレベル設計においても再利用設計による設計効率化が検討できる。我々は、設計データベースとしてどのようなデータを活用すれば良いか検討した。例として以下に、ドキュメント DB、アルゴリズム IP、機能ブロック IP、アーキテクチャ IP、見積り用 DB、実装ソフトウェア IP および実装ハードウェア IP を挙げてみた。これらを有機的に関連付けた設計データベースもまた重要になると考えられる。技術的に現在研究レベルであり検討結果を述べるにいたっていないが、今後の検討課題として提案設計フローに示した上記設計データについて述べる。

##### 【1】ドキュメント DB

過去の設計データの中の機能ブロック毎にドキュメント化されたデータベースである。再利用データのインデックスや機能概要が一別できるものが望ましい。再利用部品を修正して使う際、電子化された元データから修正すると関連部品を矛盾なくかつ更新忘れ等のミスも防止できる。規格化された仕様等は部品選択時に非常に有効となる。設計システムのフロントエンドシステムとリンクして検討していかなければならない。

## 【2】アルゴリズム IP

定式化された機能を実現する際、いくつかのアルゴリズムが考えられる。それらを機能評価用の IP として再利用することで機能決定フェーズにおける機能評価を効率化できる。

## 【3】機能ブロック IP

アルゴリズム IP が、定式化された機能の IP を指すものとするれば、機能ブロック IP は、設計対象の機能ブロック分けされたすべてに対してその用途を広げたものといえる。そのまま使うというより変更のためのベースデータとして再利用することで機能定義の効率化が図れる。

## 【4】アーキテクチャ IP

設計対象すべてのアーキテクチャを IP 化するのでは、再利用が困難になってしまうが、サブシステムレベルのアーキテクチャ流用を考える場合の IP 化は検討の余地がある。必要性も踏まえ検討していく必要がある。

## 【5】見積り用 DB

空間探索時に必要となるデータ群をさす。4.2.5.節でも述べたが、精度は低いが高速に見積れる特性データ、たとえば IP 毎に持つ遅延情報や消費電力情報やゲート数またはメモリサイズ等を静的データや特性関数で持ったものと、動の見積り用のシミュレーション時に算出するための特性データ等を有機的に保持することが必要となる。設計空間探索システムと連携したデータモデルや精度を変えられるようなシステムの検討と併せて実現化を検討していかなければならない。

## 【6】実装ソフトウェア IP

ソフトウェア開発を効率化するために、ソフトウェアを IP として部品化することは意味がある。ただし、最適化前のソフトウェアを IP 化すると処理速度の不足が問題になる場合がある。また、最適化後のソフトウェアを IP 化するとプロセッサに依存してしまうため流用化ができないことが問題となる。今後の研究が期待される。

## 【7】バス / インタフェース IP

バスやインタフェースに関する再利用化の部品であるが、有効性を含め今後検討する必要がある。

## 【8】実装ハードウェア IP

現在、IP ベース設計で使われている IP をさす。見積り DB や機能 IP と連携して拡張を考えていく必要がある。

### 4.3. ニーズ / シーズからの分析

本節では、提案したシステム設計フローが設計者のニーズ（「設計者ニーズからの課題」で述べた）を解決できるかについて検証する。また、現在の EDA 技術（「シーズとしてのシステム設計技術」で述べた）が、この設計フローをどこまでカバーできるかを検証することで、提案したシステム設計フローの妥当性を確認する。

#### 4.3.1. ニーズからの分析

設計者ニーズからの課題を設計フローにマッピングすると図 4.3-1 のようになり、ほとんどの課題が設計フローの各フェーズで解決できることが分かる。

以下では、2 章での課題分類毎に、提案設計フローでの解決策を述べる。

##### 【1】機能決定に関する課題解決策

仕様の曖昧さが残るという課題は、実行可能なシステムレベル設計言語を導入することで解決を目指す。システムレベル設計言語でシステム記述を行うことで、曖昧さが明らかになり、そこを除去する。さらに機能検証を行うことで、正しくシステム記述ができたことを確認できる。システム全体の機能仕様をリーズナブルな時間で検証できないという機能検証に対する課題に対しても、早期段階で機能定義をした後で機能検証を実行し、必要に応じて機能定義に反映することの繰り返しで問題を解決する。また、システム仕様の機能定義への記述漏れは、システム全体のシミュレーションを実行することで防げる。また、状態遷移図を記述して、その内容をチェックすることでもシステム仕様の記述漏れを回避できる。

一方、仕様が未確定な部分があっても設計を進めなければならない場合に、未確定部分を残せないという問題に対しては、他の部分に影響を与えないようなダミーモデルを用いたシミュレーションが可能性として考えられるが、実際には難しい問題である。また、未確定部分を残したまま設計を進めて、未確定部分の仕様が確定した時に、上流の設計工程に後戻りすることを避ける設計手法の確立が必要である。

##### 【2】アーキテクチャ決定に関する課題解決策

最適アーキテクチャを如何に探索、決定するかという課題は、以下のように解決できる。プロファイリング処理で得た情報を元に機能定義から機能ブロックおよびアーキテクチャの候補を生成し、その後、アーキテクチャマッピングと見積り処理を通じて、アーキテクチャ候補群から、設計制約を満たす最適なアーキテクチャを選択する。ここで、設計データベースに存在するアーキテクチャ IP と見積り用データベースを用いて、アーキテクチャマッピングと見積りを実行する。また、我々は、機能ブロックに対応させて、処理量のみを表現したトランザクションモデルを導入することで、見積り処理の高速化を図った。人手の見積り精度が甘いという課題には、より高精度な

見積りモデルの構築、および過去の設計事例データを活用したアーキテクチャ IP と見積りデータベースの充実で見積り精度を向上することが可能である。

### 【3】実装設計へのインタフェースに関する課題解決策

RTL 設計などの実装系へのパスが不明確で、設計者の工数が必要であるという課題に対しては、ハードウェア/ソフトウェアのインタフェース合成技術を適用することで解決を図れる。即ち、ハードウェア部分には動作合成および論理合成を用いて、システムレベル設計言語で記述したモデルを変換可能である。また、ソフトウェア部分については実装ソフトウェア IP を活用することで、設計者の工数を低減できる。

また、実装設計へのインタフェース実行後のモデルが要求仕様と合わないという問題については、実装設計前に用いたテストベンチを流用することで誤りの検出を行うようにする。ただし、上流から下流までを網羅的にカバーするテストベンチ生成手法は、今後の課題である。

### 【4】設計全体の課題解決策

設計全体を通してみた問題として、IP の組み込みやカスタマイズに関する問題は、IP 利用のために、IP のインタフェースの標準化、IP 合成技術の導入、機能から実装までの設計フロー全体に渡っての IP の充実を図ることが解決策となる。

ソフトウェア開発における協調検証シミュレータの性能不足という課題に対しては、パフォーマンスのネックとなっているハードウェアモデルの抽象度をあげることや、アクセラレータおよびエミュレータとのリンクなどが解決策として考えられる。

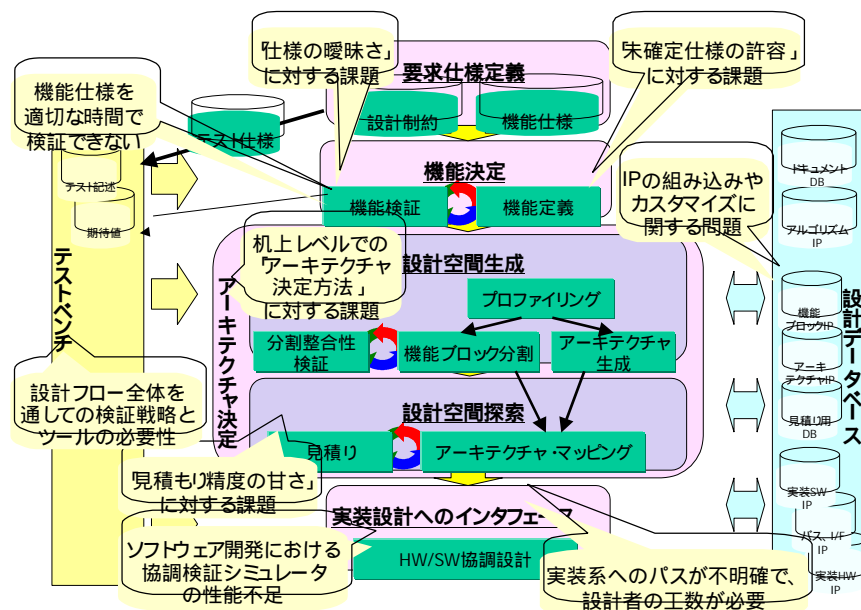


図 4.3-1 設計者ニーズから抽出した課題

#### 4.3.2. シーズからの分析

我々が提案する設計フローに対する、3章で述べたシーズとして調査した6つの現存システムの技術的サポート状況を表4.3-1に示す。

表では、システムが自動化を提供しているものを○で表わし、自動ではないがGUIなどで支援する機能をサポートしている場合は△とし、設計者を支援する機能はないがシステムを運用する際に必要な工程を□、システムの対象外である設計工程を×とした。

以下では、提案設計フローの工程毎に現存システムのサポート状況を検証し、提案フローの実現可能性を評価する。

##### 【1】要求仕様定義

現状では、要求仕様定義をサポートしているツールはなく、ユーザは各々のシステムに応じてシステム仕様および設計制約を作成しなければならない。この分野は、ソフトウェア設計手法の拡張およびその適用が期待される。

##### 【2】機能決定

機能検証については、全てのツールが機能記述をシミュレーションすることで実現している。しかし、その検証内容および機能定義をいかに記述するかは、明確になっておらず、機能定義手法の確立が必要である。

##### 【3】アーキテクチャ決定

###### 【3-1】設計空間生成

あるツールは、プロファイリングに近いことをサポートし、さらに機能定義から機能ブロック分割した結果であるプロセスモデルの変換までをサポートしている。アーキテクチャ生成については、アーキテクチャモデルを候補として提示する機能を有しているツールは存在するが、生成そのものをサポートするツールはない。設計空間生成の方法論の確立と合わせてツールの充実が望まれる。

ブロック分割したモジュール群で表わす機能が、分割前と機能的に一致していることを確認する整合性検証は、全てのシステムがシミュレーションをサポートしている。

###### 【3-2】設計空間探索

設計空間探索では、徐々にツールが整備されてきているが、パフォーマンス以外の消費電力やコストの見積りの実現、ハードウェア/ソフトウェアのいずれで実現するかの特ラードオフ評価技術、また、ある程度の精度を確保し高速に見積れる手法の確立とツールの実現が課題として残されている。

##### 【4】実装設計へのインタフェース

インタフェース合成については、ほとんどのツールが動作合成および論理合成技術

で合成できるハードウェア部分のインタフェース生成をサポートしている。

## 【5】テストベンチ

システムレベル設計でのテストベンチ作成に関しては、現時点ではサポートしているツールはほとんど無い。ランダムパターン発生機能を提供するシステムはあるが、テスト仕様や機能検証結果からそれに合致するテストベンチの生成を実行するツールの実現が望まれる。

以上の評価および考察から、我々の提案フローは、設計者ニーズから出された課題をほぼクリアでき、また、課題は残るものの、現状の技術の延長線上で実現できる可能性は十分にある。

表 4.3-1 技術（シーズ）マップ

設計工程		要素技術	ツールA	ツールB	ツールC	ツールD	ツールE	ツールF
要求仕様定義		システム仕様	×	×	×	×	×	×
		設計制約	×	×	×	×	×	×
機能決定		機能定義						
		機能検証						
アーキテク チャ決定	設計空間 生成	プロファイリング	×	×	×	×		
		機能ブロック分割						
		アーキテクチャ生成						
	設計空間 探索	分割整合性検証						
		アーキテクチャマッピング (性能見積り)	×					×
実装設計への インタフェース		インタフェース合成	×					
		HW/SW協調検証			×			
テストベンチ		テストベンチ作成						

: ツールにより自動化が実現されている  
 : 設計者を支援する機能がツールにある  
 : 設計者を支援する機能がツールにない  
 × : ツールとして対象としていない

## [ 参考文献 ]

- [1] D.C.ゴーズ、G.M.ワインバーグ 著、黒田純一郎 監訳、柳川志津子 訳 『要求仕様の探検学 - 設計に先立つ品質の作り込み - 』、共立出版株式会社 1994 年 6 月初版 9 刷発行