組込みシステムの開発における協調検証手法と技術者教育への適用実践

清尾, 克彦

https://doi.org/10.15017/1866322

出版情報：九州大学, 2017, 博士（工学）, 課程博士
バージョン：
権利関係：

組込みシステムの開発における協調検証手法と技術者教育への適用実践

2017 年 8 月

清尾克彦

目次

/114

表目次

図目次

概要

1990 年代後半から 2000 年代にかけて、半導体技術の進歩とデジタル化の進展によりコンピュータの高性能化・低価格化・低消費電力化が進み、あらゆる分野の電子機器や制御機器にコンピュータ機能が組み込まれて製品が実現されるようになってきた。このように、コンピュータ機能を組み込んで実現される電子機器や制御機器は総称して「組込みシステム」と呼ばれる。

組込みシステムは応用分野ごとに性能・コスト・納期・消費電力などに関して最適なバランスになるようにハードウェアとソフトウェアによる協調設計（コデザイン）により実現される。ハードウェアは半導体の進歩で 1 チップのシステム LSI で実現されるようになり、組込みソフトウェアと組み合わせて事前に検証をしっかり行なうことが必要となってきている。また、組込みシステムは時代の進展とともにいろいろな機能をハードウェアとソフトウェアで積み重ねて大規模化・複雑化した複合システムであり、内部はますますブラックボックス化してきている。

競争が激しい最先端の分野では、独創的なアイデアを含んだ新しい組込みシステム製品の開発が求められており、ニーズを満たし不具合のない組込みシステムを如何に短期間で実現するかが重要になってきている。このような組込みシステムの開発における問題点を分析してみると大きく設計、検証と技術者育成の 3 つの課題に分類できる。

設計については、再利用可能な設計資産（IP）の活用、IP の搭載を容易にするプラットフォームベース設計の採用や、抽象度の高い表現ができる高位言語の採用と高位合成ツールの利用により、設計の生産性向上と品質向上などが実現されつつある。

検証については、独創的なアイデアがニーズを満たしているかを評価したり、設計の不具合を取り除くために、システム全体を繰り返し動作させてみる必要があり、如何に効率良く評価していくかが大きな課題となっている。設計の上流ステージでのシステム機能を高位言語で記述した動作モデルによる評価・検証、その後のアーキテクチャ設計ステージでのハードウェアとソフトウェアによる協調検証、設計の下流にあたる実機に近い動作環境ステージでの人間の五感による評価・検証によって問題ないことが確認されてはじめてシステム LSI のモノづくりに入ることができる。

従来のやり方では、各設計ステージにおいて、設計モデル（コンピュータで実行できるように表現した設計対象）の抽象度に応じた検証環境を使ってバラバラに評価が行なわれてきた。そのため、設計の上流から下流まで効率良く検証を行なうことができなかった。また、ハードウェアとソフトウェアによる協調検証が必要になることから、そのインタフェースを各検証ステージでどのように実現するかも課題であった。

本研究では、組込みシステム開発における上流でのシステムレベルシミュレーションによる検証ステージから、下流の実機に近い環境での検証ステージまで、検証環境の統合と共通テストベンチの適用、および、コンポーネント論理バスアーキテクチャによるハードウェアとソフトウェアのインタフェースの統一により、階層的に効率良く繰り返し検証を行うトップダウン協調検証手法を提案する。トップダウン協調検証手法をマルチメディア系の 2 つの事例に適用し、各検証ステージにおいて実用レベルの性能で実行し、階層的に効率よく繰り返し検証できた成果を報告する。

技術者育成においては、最先端の組込みシステムの新規開発で、性能・コスト・消費電力などの目標を満足できるように、ハードウェアとソフトウェアの協調設計により、最適な分担を実現するシステムアーキテクトの育成が求められている。既に、組込みシステム分野では、システム LSI 開発の立場からのハードウェアとソフトウェアを含めた技術者の育成や、組込みソフトウェアに重点をおいた技術者の育成が行われているが、組込みシステム開発の課題を踏まえた技術者育成の取り組みも必要であると考えられる。

本研究では、組込みシステム開発の課題の分析や経験から抽出した「全体を俯瞰しながら基礎から応用へのスパイラルアップの取り組み」、「領域間のインタフェースの両面からの取り組み」、「ブラックボックス化した技術の階層的なモデリングの取り組み」、および、「ハードウェアとソフトウェアの最適な組み合わせを評価・選択する協調設計の取り組み」という 4 つのアプローチに基づいて、ハードウェアとソフトウェアの基本、およびインタフェースである命令セットを理解した上で、トレードオフを行えるシステムアーキテクトを育成するためのカリキュラムを提案する。実際にサイバー大学や岩手県立大学で実践した成果を報告する。

2017 年現在の IoT 時代を迎え、モノつくりの考え方が組込みシステム単体によるプロダクト指向から、ネットワークを介して、クラウドで評価・分析して付加価値を提供するサービス指向に変わってきている。IoT 時代では独創的なアイデアの創出が求められており、デザイン思考のアプローチが提案され、プロトタイピングによる評価検証による素早い繰り返しプロセスが提唱されている。本提案の「トップダウン協調検証手法」による効率の良い繰り返し検証のアプローチは IoT 時代と言われる 2017 年の今の視点から見ても役に立つと考えられる。また、IoT 時代の技術者育成について 2017 年現在取り組み中であるが、今回提案した 4 つのアプローチがクラウドコンピューティンと通信ネットワークを含めた IoT システムの教育へ拡張するのに有効と考えている。

第１章　序論

本論文は、組込みシステムの開発における協調検証手法と技術者教育への適用実践についての研究に関してまとめたものである。

本章では、研究の背景、組込みシステム開発の課題、関連研究と本研究の位置づけ、研究の概要、および論文の構成について述べる。

1.1　研究の背景

1990 年代後半から 2000 年代にかけて、半導体技術の進歩とデジタル化の進展によりコンピュータの高性能化・低価格化・低消費電力化が進み、あらゆる分野の電子機器や制御機器にコンピュータ機能が組み込まれて製品が実現されるようになってきた。このように、コンピュータ機能を組み込んで実現される電子機器や制御機器は総称して「組込みシステム」と呼ばれ、デジタル情報家電や通信機器、自動車制御機器など産業の基幹製品を占めるようになってきている。

組込みシステムはハードウェアとそれを動かすソフトウェアで構成されることから、応用分野の特性に応じてハードウェアとソフトウェアでどのように機能を分担して実現するかということが、製品の性能・コスト・納期・消費電力などを決める重要なファクタとなってきている。このように、製品にとって性能・コスト・納期・消費電力などが最適なバランスになるようにハードウェアとソフトウェアの分担を決めていく設計は協調設計（コデザイン）と呼ばれている。協調設計は製品の良し悪しを決める重要な位置づけにあり、製品全体に精通したシステムアーキテクトとよばれる人材が対応している。

初期（1990 年以前）の組込みシステムでは、実現する機能もシンプルで、まずはハードウェアを開発してから、そのハードウェアの上でソフトウェアを開発するやり方が行われてきた。ソフトウェアの開発中に機能やハードウェアの不具合が発見された場合は、ハードウェアを比較的容易に改修することができ、ソフトウェアを変更することで、組込みシステム全体の開発への影響は必ずしも大きくならずに対応することができた。

しかし、その後のムーアの法則による半導体の集積度の向上により、組込みシステムのハードウェアの大部分を 1 チップで実現するシステム LSI 時代（1990 年代後半以降）を迎えるようになった。市場ニーズとのミスマッチやハードウェアとソフトウェアから構成される組込みシステムの不具合により、システム LSI を作り直すことになると膨大な対策費用と機会損失が発生してしまうことから、LSI を作成する前に評価や検証により問題点や不具合を取り除く必要がでてきた。

また、組込みシステムは、時代の進展とともにいろいろな機能をハードウェアとソフトウェアによる複合技術で積み重ねてきており、最先端の組込みシステムはますます複雑なシステムになってきている。このように技術の階層が深くなるに従い、その内部のブラックボックス化が進んできている。

本研究は、このような 1990 年代後半から 2000 年代における時代背景のもとに、組込みシステム開発を対象に行なわれたものである。しかし、2014 年頃からの IoT（Internet of Things：モノのインターネット）時代を迎え、モノつくりの考え方が組込みシステム単体を販売するプロダクト指向から、組込みシステムをインターネットに接続してクラウドで評価・分析して付加価値を提供する IoT システムによるサービス指向に大きく変わろうとしている。このような状況を踏まえ、IoT 時代と言われる 2017 年の今の視点から、1990 年代後半から 2000 年代にかけての本研究の取り組みに対して評価・考察を加えている。

1.2　組込みシステム開発における課題

1990 年代後半から 2000 年代にかけて、競争が激しい最先端の組込みシステムとして、音や映像を扱うマルチメディア系やものの動きを制御する制御システム系の分野が注目されている。特に、独創的なアイデアを含んだ新製品の開発において、ニーズを満たし不具合のない組込みシステムを如何に短期間で実現するかが重要になってきている。このような組込みシステム開発の 1990 年代後半から 2000 年代おける問題点を分析してみると大きく設計と検証と技術者育成の３つの課題に分類される。

設計については、再利用可能な設計資産（IP）の活用、IP の搭載を容易にするプラットフォームベース設計の採用や、抽象度の高い表現ができる高位言語の採用と高位合成ツールの利用により、設計の生産性向上と品質向上など改善が既に進んできている。

それに対して、独創的なアイデアがニーズを満たしているかの評価や、ハードウェアとソフトウェアによる協調設計の不具合を取り除くための検証については、システム全体を動作させてみる必要があり、如何に効率良く実現するかが大きな課題となってきている。設計の上流ステージでシステム機能を高位言語で記述した動作モデルによる評価・検証、その後のアーキテクチャ設計ステージでハードウェアとソフトウェアによる協調検証、設計の下流にあたる実機に近い動作環境ステージでの人間の五感による評価・検証によって問題ないことが確認されてはじめてシステム LSI のモノづくりに入ることができる。従来は各ステージでそれぞればらばらな環境で評価・検証が行われてきたが、設計の上流から下流まで、繰り返しながら評価ができるトップダウン的な検証の取り組みが求められている。

また、複雑化してきた組込みシステムの新規開発において、応用製品として性能・コスト・消費電力などの目標を満足できるように、ハードウェアとソフトウェアによる最適な分担を実現する協調設計がますます重要になってきている。特に、独創的な機能を新たに開発する場合には、ブラックボックス化した内部を含めて、新しい視点でブレークスルーを図ることができるシステムアーキテクトの育成が大変重要になる。既に、組込みシステム分野では、組込みソフトウェアやシステム LSI 開発者の立場から、いろいろな取り組みが行われているが、組込みシステム開発の課題を踏まえて体系的な技術者育成の取り組みが求められている。

IoT 時代と呼ばれる 2017 年においては、モノつくりの対象がプロダクト指向の単体の組込みシステムから、組込みシステムと通信ネットワークとクラウドコンピューティングから構成される IoT システムというサービス指向の複合システム（SoS : System of Systems）に進化しつつある。このような IoT 時代は、第 4 次産業革命の到来と言われており、従来の延長線上とは異なる新しい取り組みが期待され、複雑なシステムの検証や付加価値創造に対応できる技術者育成が求められている。

1.3　関連研究と本研究の位置付け

本研究の全体像と位置づけを図 1.1 に示す。独創的なアイデアを取り込んだ組込みシステム製品開発を対象に、組込みシステム開発における課題分析を行ない、抽出された 2 つ課題（「検証の課題」と「技術者育成の課題」）を解決するための 2 つの研究（「トップダウン協調検証手法の提案と実践評価」と「複合領域での教育手法をベースとした組込みシステム技術者教育の提案と実践評価」）を示している。

1990 年代後半から 2000 年代にかけて、独創的なアイデアを含んだ組込みシステムの開発では、設計の上流ステージでシステム機能を高位言語で記述した動作モデルによる評価・検証が行なわれてきた。続くアーキテクチャ設計のステージでは、ハードウェアとソフトウェアによる協調設計・協調検証が、設計の下流にあたる実機に近い動作環境ステージでは、人間の五感による評価・検証が、階層的に行なわれきた。このようにシステム全体を動作させて評価する場合には、コンピュータ上でのシミュレーションや実機に近い動作環境で対象システムを実際に動かして評価を行なう動的な検証手法が使われている。従来のやり方では、各設計ステージにおいて、設計モデル（コンピュータで実行できるように表現した設計対象）の抽象度に応じた検証環境を使ってバラバラに評価が行なわれてきた。そのため、設計の上流から下流まで効率良く検証を行なうことができなかった。また、ハードウェアとソフトウェアでの協調検証が対象となることから、ハードウェアとソフト

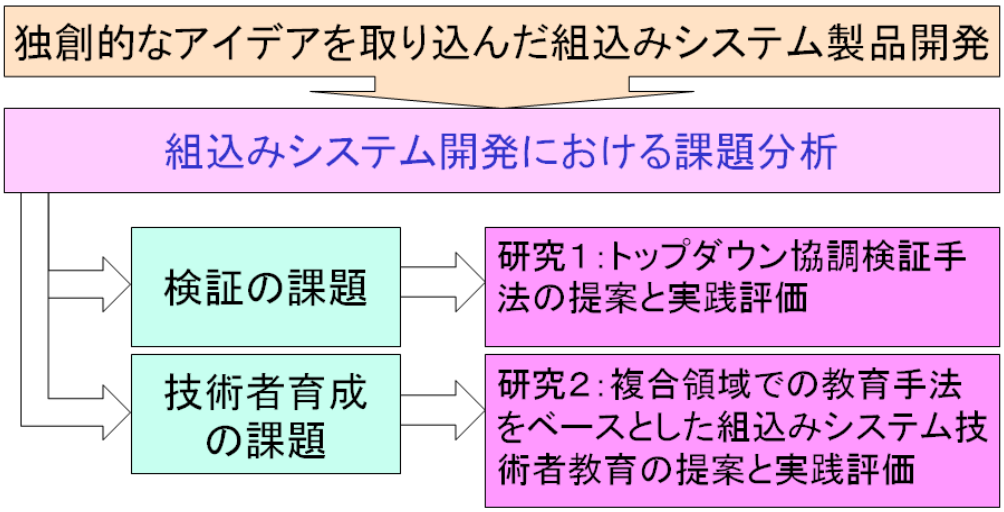


図 1.1 組込みシステム開発における課題分析と本研究の位置づけ

ウェアのインタフェースを設計の上位のステージから下位のステージまでどのように表現して検証を進めるかも重要な課題である。従来のやり方では、ある特定のステージにおけるインタフェース部分の合成や仮想CPUの命令セットとの協調シミュレーションについての取り組みが主であった。このため、設計の上流から下流にかけて、インタフェースの実現方法が異なると、変更が必要となり、検証作業の効率は良くなかった。本研究では、設計の最上流ステージから最下流の実機に近い動作環境ステージまで、繰り返し検証を行なうことができるように、検証環境の統合と共通テストベンチの適用、および、ハードウェアとソフトウェアのインタフェースの統一を図ったトップダウン協調検証手法を提案し、マルチメディア系の映像圧縮伸張システムなどを題材にその有効性を評価した。

組込みシステム開発において、組込みシステムが大規模化・複雑化し、技術のブラックボックス化が進むにつれて、ハードウェアとソフトウェアによる協調設計や協調検証を行う技術者の育成が今まで以上に重要になってきており、いろいろな機関で取り組みが行なわれてきた。これらの取り組みでは、システム LSI 開発の立場からのハードウェアとソフトウェアを含めた技術者の育成や、組込みソフトウェアに重点をおいた技術者の育成がメインで、それぞれ大勢の技術者を育成し大きな成果を上げてきた。本研究では、組込みシステム開発における課題の分析や経験を踏まえて、協調設計・協調検証手法を実践できるシステムアーキテクトを育成するための複合領域での教育手法を提案し、それに基づいた組込みシステム技術者育成のカリキュラムを提案・実践し、その有効性を評価した。

なお、IoT 時代と言われる 2017 年の今の視点から見ると、今回の提案の考え方は、組込みシステム開発に留まらず、IoT システム開発における課題の解決のヒントになると考えている。

IoT 時代では独創的なアイデアの創出が求められており、デザイン思考[黒川 12]のアプローチが提案され、プロトタイピングによる評価検証による素早い繰り返しプロセスが提唱されているが、本提案の「トップダウン協調検証手法」のアプローチはその先駆けの取り組みともいえる。また、技術者育成における提案についても、IoT 時代のクラウドコンピューティングと通信ネットワークを含めた複合領域の教育へ拡張する場合のベースになると考えている。

1.4 研究の概要

以下に、２つの研究内容の概要を示す。

なお、本研究は、1990 年代後半から 2000 年代にかけての組込みシステムの検証の課題に関する解決策の提案であるが、IoT 時代と言われる 2017 年の今の視点から見た評価も加えている。

（研究１）トップダウン協調設計検証手法の提案と評価

組込みシステム開発における上流のシステムレベルシミュレーションの検証ステージから、ハードウェアとソフトウェアによる協調シミュレーションの検証ステージ、さらに下

流の実機に近い環境でのハードウェアとソフトウェアによる協調エミュレーションの検証ステージまで、検証環境の統合と共通テストベンチの適用、および、コンポーネント論理バスアーキテクチャによるハードウェアとソフトウェアのインタフェースの統一により、階層的に効率良く検証を行うトップダウン協調検証手法を提案する。

マルチメディア系の2つの事例に適用して評価を行なった。1つ目の事例では、九州大学安浦研究室との共同研究の成果物である自製の ASAP（Application Specific Adaptable Processor）[Sha96]を使った線描画システムを対象に、協調シミュレーションを適用した。2 つ目の事例では、調達した 32 ビットプロセッサ（M32R/D）を使ったマルチメディア系の組込みシステム（映像圧縮処理システム）を対象に、上流のシステムレベルでのアルゴリズムの評価から、ハードウェアとソフトウェアによる協調シミュレーション、さらに下流の人間の感性による実機レベルの協調エミュレーションによる評価まで、階層的にトップダウン協調検証手法を適用した。各検証ステージにおいて実用レベルで検証することができ、階層的に効率よく繰り返し検証をすすめることができた[Seo97][YaM98][Seo98]。

（研究 2）複合領域での教育手法をベースとした組込みシステム技術者教育の提案と実践評価

複合領域である組込みシステムの開発において協調設計手法の重要度が増す中で、協調設計を実践できるシステムアーキテクチャの育成が急務となってきている。本研究では、組込みシステム開発の課題の分析や経験から抽出した「全体を俯瞰しながら基礎から応用へのスパイラルアップの取り組み」、「領域間のインタフェースの両面からの取り組み」、「ブラックボックス化した技術の階層的なモデリングの取り組み」、および、「ハードウェアとソフトウェアの最適な組み合わせを評価・選択する協調設計の取り組み」という 4 つのアプローチに基づいて、ハードウェアとソフトウェアの基本、およびインタフェースである命令セットを理解した上で、トレードオフを行えるシステムアーキテクトを育成するためのカリキュラムを提案する。実際にサイバー大学や岩手県立大学で実践した成果を報告する[清尾 11A][吉田 11A]。

1.5　論文の構成

本論文は、以下の 5 章で構成される。なお、1990 年代後半から 2000 年代にかけての課題が対象であるが、IoT 時代と言われる 2017 年の今の視点からの評価も示している。

第 1 章では、本論文の研究の背景と組込みシステム開発の課題を述べ、課題解決のために行なった研究の概要について述べた。

第 2 章では、組込みシステムの開発における現状と課題について幅広い視点から整理を行う。特に独創的なアイデアを取り入れた組込みシステムを新たに開発する場合における、検証の課題、および、技術者育成の課題を示す。

第 3 章では、組込みシステムにおけるトップダウン協調設計検証手法を提案し、マルチメディア系組込みシステムなど 2 件の事例を対象にトップダウン協調検証手法を適用した

成果について考察する。

第 4 章では、複合領域である組込みシステムにおける協調設計を主導できる技術者（システムアーキテクト）の教育手法を提案し、それに基づいて作成したカリキュラムの概要と適用結果について考察する。

第 5 章では本研究の成果をまとめると共に、今後の課題と取り組みについて述べる。

第 2 章　組込みシステム開発の現状と課題

本章では、1990 年代後半から 2000 年代にかけての最先端の組込みシステム開発における現状と課題を示す。組込みシステム開発において、ものづくりの成果物である応用目的に最適化された組込みシステム製品（プロダクト）、いろいろな技術を組み合わせた複雑系の組込みシステムを開発する手順（プロセス）、組込みシステムの開発を目標通りに進めていくプロジェクト、プロジェクトに参画する技術者（ピープル）に必要なスキルについて議論していく[Bra04]。

本章では、まず、組込みシステムとは何かを説明し、組込みシステム開発の発展の変遷、および、組込みシステム製品の技術成熟度の変遷について述べる。

そのあと、プロダクトの視点から、組込みシステムがいろいろな機能の集成体で複雑系のシステムであることを示す。組込みシステムの機能をハードウェア（ここではシステム LSI）とソフトウェア（組込みソフトウェア）の組み合わせで実現し、性能・コスト・消費電力など相反する要因をトレードオフしながら応用目的に最適な解をもとめていく協調設計（コデザイン）について説明する。また、組込みシステム製品ではシリーズものとして、設計資産を有効に再利用しながら拡張開発などを行うことから、製品としてのライフサイクルの視点から考える必要がある。

次に、プロセスの視点から、組込みシステムの開発の流れを説明し、特にハードウェアとソフトウェアによる協調設計と協調検証について述べる。

さらに、プロジェクトの視点から、大規模化・複雑化が進む組込みシステムの開発ではプロジェクト管理が重要になってきたことを示す。

最後に、組込みシステム開発のプロジェクトに参画する技術者に求められるスキルについて述べる。

上記の 4 つの視点から述べた現状を踏まえて、組込みシステム開発における課題を、検証と人材育成の観点からまとめる。

なお、前述したように IoT 時代を迎え、組込みシステムを取り巻く状況が大きくかわりつつある。モノつくりの考え方が組込みシステム単体を販売するプロダクト指向から、ネットワークで接続されたクラウドで評価・分析して付加価値を提供するサービス指向に大きく変ろうとしている。このような現状を踏まえて、IoT 時代の組込みシステムの現状と課題についても言及する。

2.1　組込みシステムとは

各種の機械や機器に組み込まれて、それを制御するためのコンピュータシステムは、組込みシステムと呼ばれている。一般に PC（パソコン）は汎用のコンピュータシステムとして、組込みシステムと区別して考えられるが、PC 内部の IO 制御装置は専用のコンピュータが組み込まれた組込みシステムといえる。また、汎用のコンピュータである PC を使っ

て実現した ATM（Automated Teller Machine：自動出納機）は専用目的の組込みシステムといえる。このように、組込みシステムは、組み込まれるコンピュータが専用目的で使われるかどうかに依存していると考えられる。

組込みシステムの主な特性をまとめると以下の通りである（図 2.1 参照）。

①製品の目的に最適化された専用化されたシステムである。新規開発の場合は、ハードウェアとソフトウェア、および開発環境の同時開発が必要となる。また、開発環境と実行環境が異なる。

②大きさや重さ、コスト、消費電力、開発期間などの厳しい制約条件を満足するように機能・性能などを実現する必要がある。

③人命や社会活動に影響を与えるシステムに組み込まれて使用される場合には、高い信頼性・安全性が求められる。

④定められた時間内に処理を終える必要がある場合には、リアルタイム性が要求される。

なお、組込みシステム製品の応用分野ごとに重視される特性は異なる。図 2.1 に示すように、携帯機器では矢印が示す低コスト化・低消費電力化・ハードウェア制約・期間短縮という厳しい制約条件が重視され、自動車機器では動作環境・高い信頼性・リアルタイム性が重視される。

また、組込みシステムはコンピュータ機能が組み込まれ、ハードウェアとソフトウェアという異なる技術を組み合わせて実現されることから、複合領域の世界といえる。組込みシステムの開発では、性能・コスト・消費電力等の相反する制約がある状態（この状態をトレードオフという）で、要求される機能を満足するようにハードウェアとソフトウェアで分担して実現することになる。

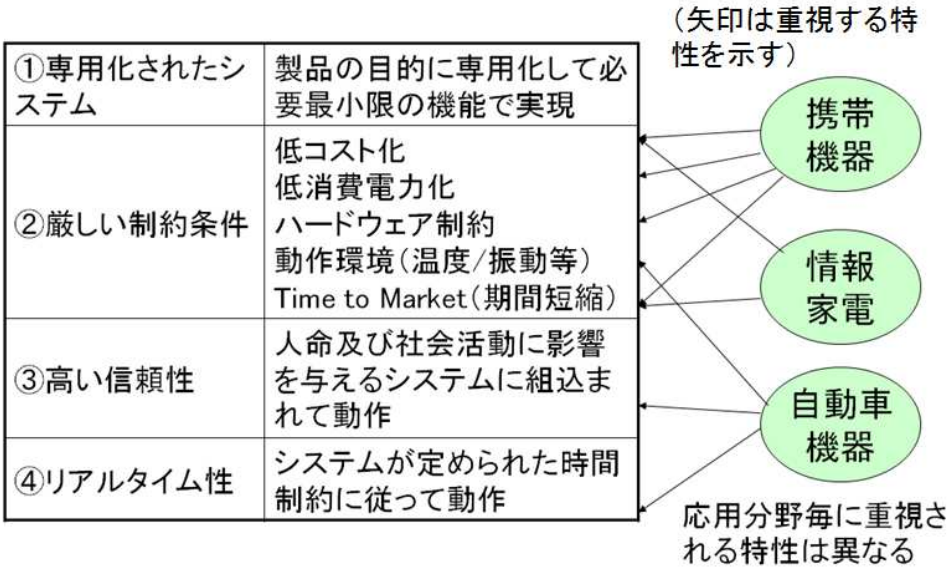


図 2.1　組込みシステムの主な特性

2.2　組込みシステム開発の変遷

情報通信社会の発展動向を図 2.2 に示す。ここでは、1960 年代から 2010 年代までの範囲を扱っている。この中で、組込みシステムはコンピュータの小型化・低コスト化・低消費電力化に伴い、あるゆる分野の機器で使われるようになってきている[Wol07] [Wol09]。2000 年には出荷されているコンピュータ（4 ビットから 64 ビットまで）の数の 98%が組込みシステムとして使われているといわれている[Ten00]。また、最近（2010 年以降）では、センサやアクチュエータを搭載した組込みシステムとネットワーク技術、クラウド技術を融合した IoT（Internet of Things）/M2M（Machine-to-Machine）が注目されている[清尾 13A][電気 16]。

組込みシステムの起源は定かではないが、1960 年代の汎用コンピュータの中央処理装置や入出力処理装置などは、制御回路の実現手段としてマイクロプログラム（ファームウェアとも呼ばれた）制御方式が使われていた。これらの制御回路は、専用目的の独自の命令セットを持ったコンピュータ機能であり、組込みシステムの草分けと言える。

炊飯器の例では、1980 年代にタイマー機能のついたマイコン制御の電気炊飯器が出現し、1990 年代には IH（Induction Heating）式が登場し、好みの炊き加減などを調整できる多機能化が進んだ。2000 年代から高級志向できめ細かな制御によりおいしいご飯が炊けるようになってきている。

また、国内の携帯電話では 1987 年に最初に販売されたが当時の重量は 900g もあった。1990 年代半ばあたりでデジタル化が進み、半導体の高集積化・低消費電力化により重量が 200g を割り 100g に近くなるに従って急速に普及をはじめた。2000 年代に入ると第三世代

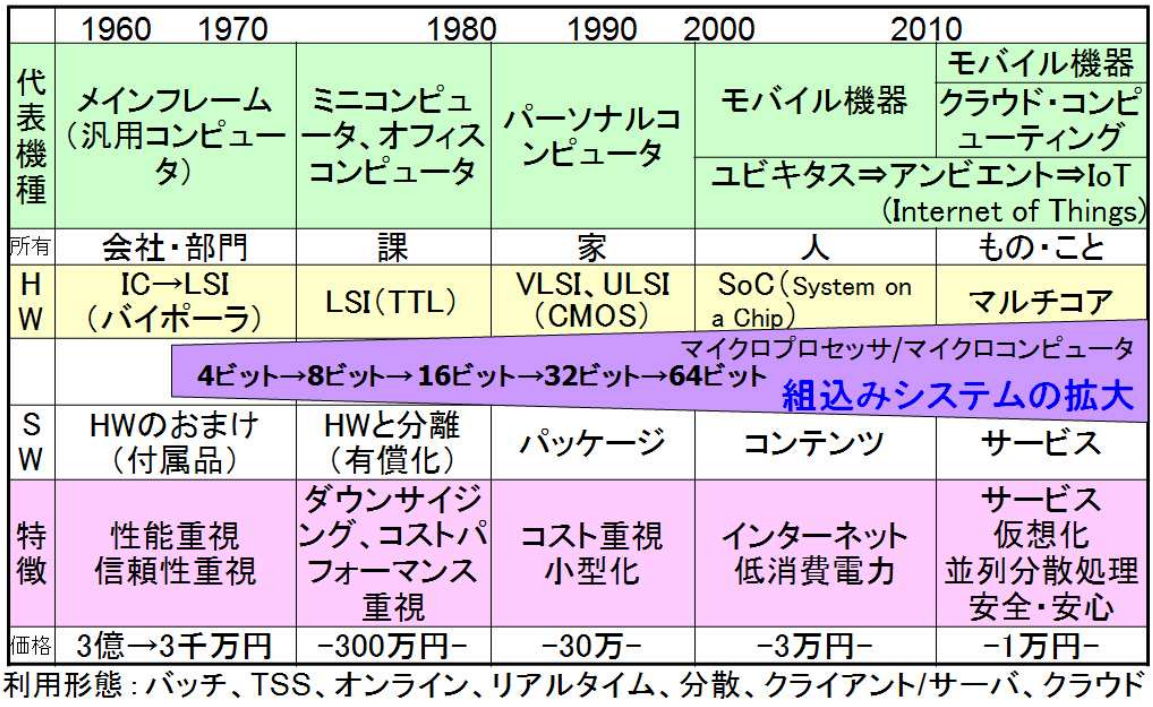


図 2.2　情報通信社会の発展動向

が登場しインターネットとの融合が行われ、2007 年以降はスマートフォンの普及が進んで、まさに PC の機能が手のひらで実現されるようになってきている。

自動車では、排ガス規制の関係で1990年代にエンジン制御の電子化がはじまり、その後2000 年代にかけてエアバックや ABS（Antilock Brake System）の電子化が進み、現在では車内の情報系（カーナビ、ETC : Electronic Toll Collection など）や画像認識などによる予防安全系の取り組みで進んでいる。2010 年代の高級車では、100 個程度のコンピュータ機能が搭載されていると云われている。

このように組込みシステムは、ムーアの法則による半導体の微細化とソフトウェア化によって、いろいろな電子機器をより洗練された知的なシステム（スマートシステム）へと変身させてきた。

特に IoT 時代（当初は IoT の代わりに M2M という用語が使われていた）を迎え、組込みシステムはユーザに近いエッジ側のセンサとアクチュエータを搭載した機器やクラウドとの中継を行なうゲートウェイ機器として位置づけられている。組込みシステムは従来のプロダクトとしての価値提供から、クラウドコンピューティングと通信ネットワークにより接続され、収集したデータ（ビッグデータ）を評価・分析して、その結果を実世界にフィードバックすることで、ユーザに価値を提供するサービス指向に大きく変わろうとしている。IoT 時代の組込みシステムについては、2.8 節であらためて論じる。

2.3　組込みシステム製品の技術成熟度の変遷

組込みシステム製品の実現方法（コンピュータアーキテクチャ）は、その応用分野の技術の成熟度に応じて変化していく。主な組込みシステム製品の技術的成熟度と実現方法の変遷のイメージ（2007 年時点）を図 2.3 に示す。縦軸に製品に使われている技術の成熟度（ここでは技術の完成度の意味合い）を示し、横軸に成熟度が低い方から黎明期、発展期、成熟期、安定期の 4 つに期間に分けている。組込みシステムではないが、PC は約 20 年前（1980 年後半）は発展期にあったが現在（2007 年時点）は安定期にある。携帯電話は 15 年前（1990 年前半）は発展期にあったが現在（2007 年時点）は成熟期にあり、自動車の電子化は現在（2007 年時点）発展期にあると位置付けられる。

技術が確立される黎明期には、新たな市場への参入を目指して、いろいろなオリジナリティを持ったアーキテクチャが出現するようになる。次の発展期には、市場の拡大を目指して、それぞれのアーキテクチャが高度化・洗練化されていく。この時期までは、一般に自社を中心に主要部分を自ら開発する垂直統合型に近いビジネスモデルになる。ハードウェアとソフトウェアによる協調設計によるトレードオフが最も行われる時期でもある。成熟期になると、市場での競争が激しくコスト競争となり、すべてを自社だけで対応できなくなってくる。ビジネスモデルも水平分業型に移行し、インタフェースの標準化とともに、効率よく機種展開するためにプラットフォーム化が進んでいく。安定期になると、コスト競争を含めビジネスに勝ち残ったアーキテクチャ（プラットフォーム）が業界標準として

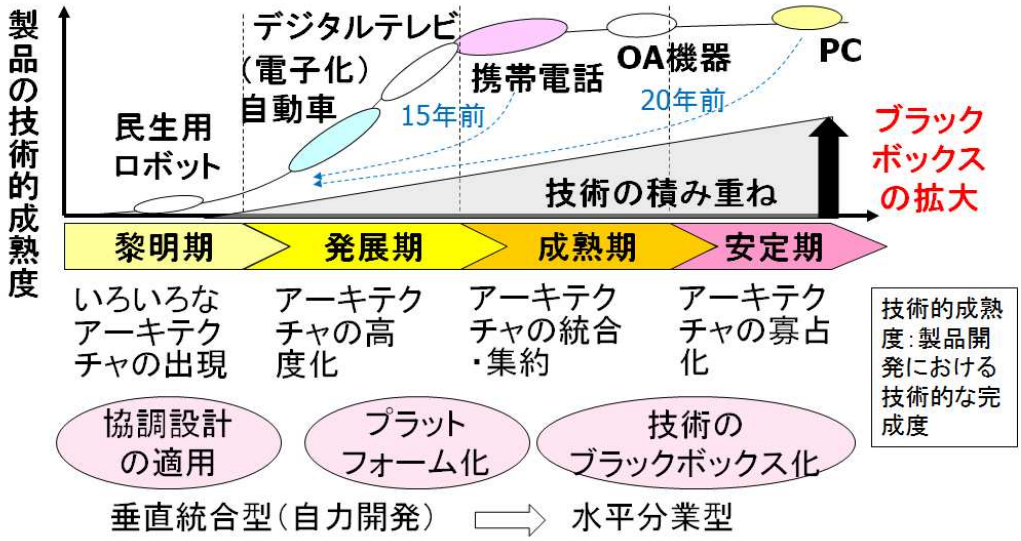


図 2.3　組込みシステム製品の技術的成熟度の変遷

（図 2.3 は、産業構造審議会情報経済分科会、第１１回情報サービス・ソフトウェア小委員会、「イノベーションの促進について」、2007 年[産業 07]より引用し筆者が改変）

固定化されるようになる。

組込みシステム製品は、黎明期、発展期、成熟期、安定期を通じて、いろいろな技術を積み重ねた結果として実現されており、その内部はブラックボックス化が拡大してきている。

黎明期から参画してきた技術者は、新しい技術を少しずつ積み上げながら、発展期・成熟期・安定期を乗り越えてきた場合が多く、ブラックボックス化された技術をしっかりと身に付けている。それに対して、後半から参画する若手技術者にとって、ブラックボックス化された技術の階層が深く、それを短時間で習得することは大変厳しい状況になってきている。

しかし、次世代を目指して大きなブレークスルーを図る場合には、このブラックボックス化した世界を如何に乗り越えていくかが重要になる。また、複雑化した組込みシステム製品の不具合の分析などでは、ブラックボックス化した内部の理解も必要となる。このように階層化された技術の内部を理解し、独創的な分野で活躍できる人材（システムアーキテクト）の育成が重要であり、それに必要な体系的な教育手法の充実が求められている。

2.4　組込みシステムのプロダクト

プロダクト（製品）に関して、以下の 4 つの視点から述べる。

① 組込みシステムは、AV（Audio Visual）・制御・通信などの応用技術分野を対象に、いろいろな異なる技術の組み合わせにより実現される多様な技術の集成体といえる。

② 組込みシステムは、ハードウェアとソフトウェアによる協調設計で実現されることから、コンピュータアーキテクチャの実現方法（ハードウェアとソフトウェアのインタフェース）は多様である。

③ 組込みシステムは、専用システムとして応用分野に特化した形で、トレードオフの関係にある性能・コスト・消費電力等の制約条件を満足するように、ハードウェアとソフトウェアの分担で実現する必要がある。

④ 組込みシステム製品の開発では、新規開発にかかる開発費用が膨大になることから、単に１回だけ開発するのではなく、製品開発のロードマップに沿って、改良開発や派生品開発を繰り返しおこない、製品としてのライフサイクル全体を通して費用を回収し利益を出せるように、IP による設計資産の再利用を含めた効率の良い実現方法を考えることが必要である。

2.4.1 組込みシステム製品は多様な技術の集成体

組込みシステムの概念的な構成を図 2.4 に示す。図 2.4 の中心部分がコンピュータシステムであり、ハードウェア（入力＋出力＋制御＋演算＋記憶）と記憶に格納されたソフトウェア（プログラムとデータ）で組込みシステムの機能を実現している。対象となる組込みシステムの要求仕様を満足するように、コスト・性能・消費電力などの相反する条件の中でトレードオフをおこない、ハードウェアとソフトウェアの分担を実現する協調設計が求められる。

マルチメディア系の処理では、マイクからの音声やカメラからの画像のアナログ入力信号をデジタルデータに変換してから、データ処理や圧縮伸張処理などの演算を行い、加工された音声や画像のデジタルデータをそれぞれアナログ信号に変換して、音声をスピーカ

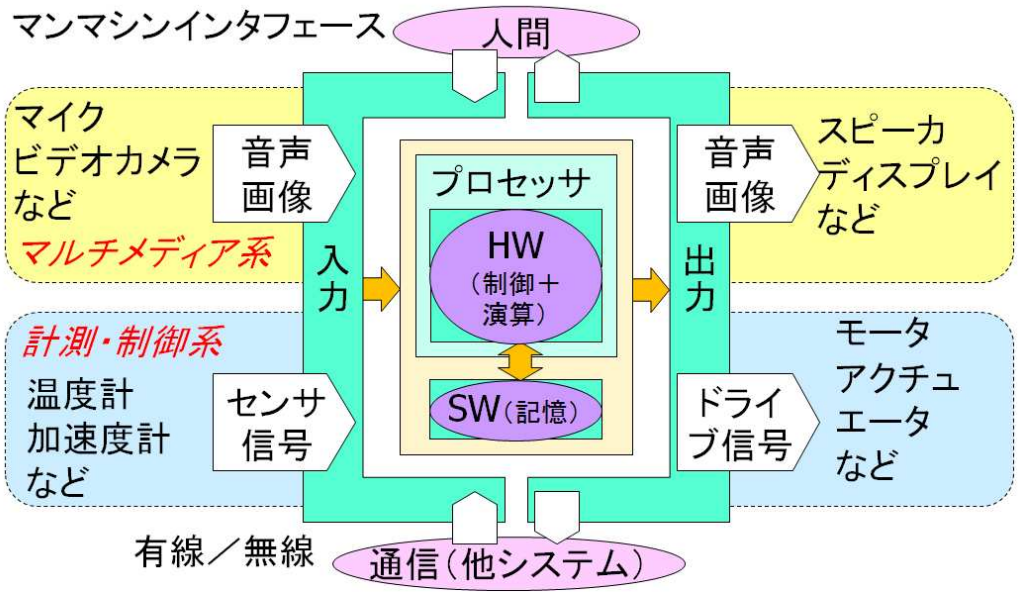


図 2.4 組込みシステムの概念的な構成

に、画像をディスプレイに出力することが行われる。

計測・制御系の処理では、温度や加速度などのセンサからのアナログ信号をデジタルデータに変換してから、制御アルゴリズムに基づいて演算を行い、制御されたデジタルデータをアナログ信号に変換して出力し、モータなどのアクチュエータを駆動することが行われる。一般に計測・制御系では、駆動した結果をセンサで測定し、アクチュエータを操作して目標の状態になるように制御が行われる。

また、組込みシステムは、専門家でない一般の人が取り扱うことから、人間にやさしい（使い勝手の良い）マンマシンインタフェースが必要となる。

さらに、組込みシステムが単独での使用から、有線や無線でネットワークに接続された形で利用されるようになってきており、他システムとの通信機能も必要とされている。

携帯電話の例をみると、マルチメディア系としてカメラ等による画像処理や音声処理機能が、計測・制御系として各種センサやバイブレータ用のモータなどが、人間とのインタフェースとなる高精度なディスプレイや、本来の機能である通信機能がそれぞれ盛り込まれて、ハードウェアとソフトウェアで分担して実現されている。

このように、組込みシステムは異なる技術を組み合わせて実現される複合領域のシステムであり、応用分野の基本技術を含めいろいろな異なる技術の理解・習得が必要となる。特に、ハードウェアとソフトウェアによる協調設計においては、トレードオフを行うのに必要な両方の技術の習得が必要になる。

また、組込みシステムの規模や適用範囲の急速な拡大により、複合領域としての複雑さがますます増大する中で、システム LSI の実現においてニーズを満足し、不具合を取り除くための検証が大きな課題である。特に、人間の目による画像評価や耳による音声評価が必要なマルチメディア系や、実際の動特性評価が必要な制御系の検証においては、実機レベルに近い環境での評価が必要である。

2.4.2 ハードウェアとソフトウェアの協調設計

組込みシステムのアプリケーションプログラムを処理するコンピュータシステムの構成（コンピュータ・アーキテクチャ）を図 2.5 に示す。ハードウェアとソフトウェアの協調設計における機能分担の方式として、水平分割方式と垂直分割方式の 2 通りが存在する[Ern98]。

(1) 水平分割（Horizontal Partitioning）方式

アプリケーションプログラムの処理の目的（高速化、コードサイズの削減など）に適した命令セットをもつプロセッサを実現し、それに対応したコンパイラやシミュレータなどの開発環境を構築するものである。一般的には、基本部（固定部と可変部から構成）と拡張部から構成される。トップダウン協調設計システムの構築で開発した ASAP[Sha96]が該当する。また、九州大学の Soft-Core Processor and Valen-C[Yas95][Yas98]や大阪大学の

ASIP[今井 02]などがこれに相当する。ハードウェアとソフトウェアのインタフェースは、基本部＋拡張部の命令セットになる。

(2)　垂直分割（Vertical Partitioning）方式

アプリケーションプログラムを構成する機能モジュール群をハードウェアとソフトウェアで分担して実現する方法である。高位言語（SystemC[Sys02]など、C 言語や C++をベースとしたシステムの機能を記述する言語）で記述されたハードウェア部分は、高位合成により RTL（Register Transfer Level）の回路に変換される。また、ハードウェアとソフトウェアのインタフェース部分は、一般にバスを介した動作シーケンスにより実現され、インタフェース合成ツールまたは人手により、ソフトウェア側のドライバなどやハードウェア側のインタフェース回路などが生成される。

プロセッサ部分については、基本部のみの場合には一般に既存のプロセッサを調達して使うことが多く、プロセッサの設計・検証モデルを入手できるかは調達先に依存する。前述の水平分割方式で、アプリケーションを効率よく実現できるように命令を拡張する場合には、プロセッサ部分を自分で作成することになるので、プロセッサの設計・検証モデルは容易に入手することができる。

このように、組込みシステムはハードウェアとソフトウェアにより実現されることから、協調設計・協調検証において、ハードウェアとソフトウェアのインタフェース、および、プロセッサのモデリングをどのように取り扱うかが課題である。

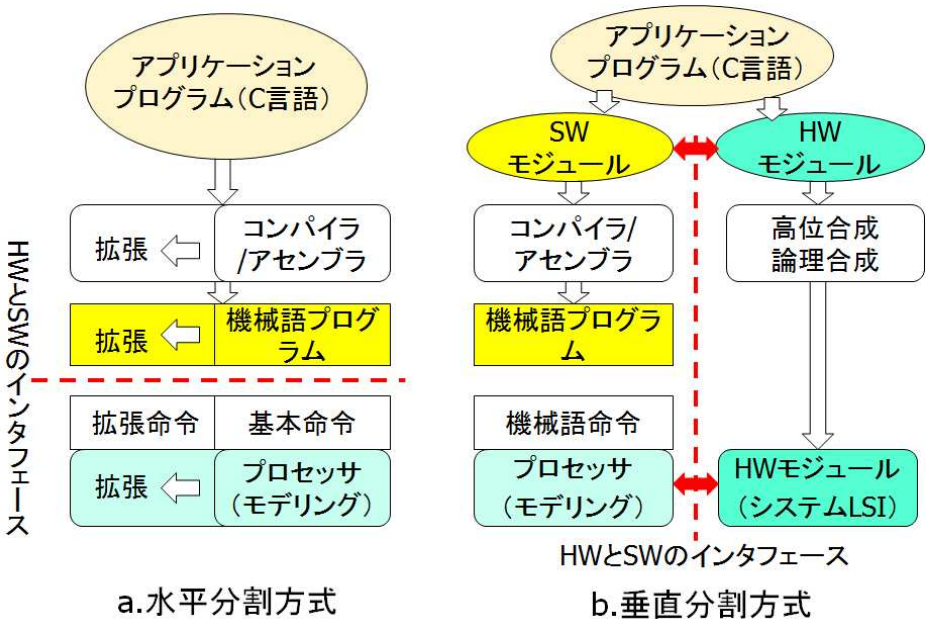


図 2.5　組込みシステムのコンピュータ・システムの構成

2.4.3　組込みシステムにおけるトレードオフ設計

組込みシステム製品の開発では、アーキテクチャ設計の段階において、いくつかの実現方法の選択肢の中から、いくつかの相反する項目を評価して、全体としてあらかじめ設定した条件を満足した解を求めるトレードオフ設計が行われる。組込みシステムにおけるトレードオフ設計の概要を図 2.6 に示す。この例では、トレードオフ項目として、性能・コスト・消費電力・柔軟性（変化への対応のしやすさ）などを取り上げる。以下、図 2.6 に示した 3 つの実現方法（(1)、(2)、(3)）について述べる。

(1) プラットフォームと IP の選択

組込みシステム製品の特性に合うように、基本となるプロセッサやバスなどのハードウェア構成を定め、その上で動作する組込み OS などの基本的なソフトウェア構成を定めたものをプラットフォームと呼ぶ。なお、ハードウェア構成部分をハードウェア・プラットフォームと呼び、ソフトウェア構成部分をソフトウェア・プラットフォームと呼ぶ。

再利用可能な付加価値の高い設計資産のことを IP（Intellectual Property）と呼ぶ。設計生産性向上の観点から、IP の活用が注目されている。競争領域でない IP については、外部の IP ベンダーから調達することが多い。逆に、差異化を図るために新たに開発した機能を自らの IP として有効活用することにより、付加価値の高い製品開発を継続することが可能になる。

組込みシステム製品をシリーズ化して開発を行う場合には、開発コストの低減、開発期間の短縮などから、プラットフォームの選定と IP の活用が重要となる。

(2) ハードウェアとソフトウェアの分担

新規に開発するシステム機能の各動作部分をハードウェアまたはソフトウェアで実現

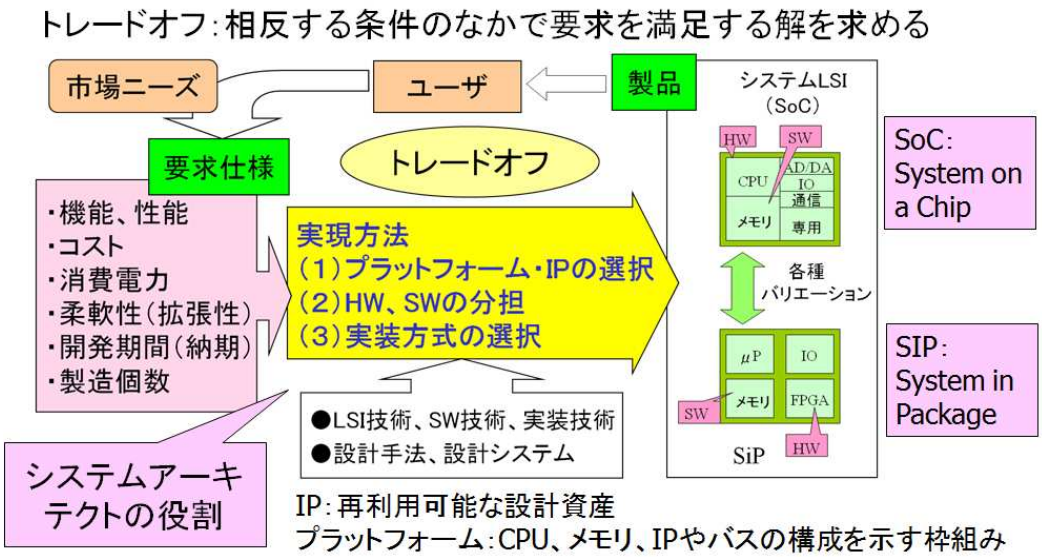


図 2.6　トレードオフ設計の概要

し、通信部分をハードウェアとソフトウェア、ハードウェア同士、ソフトウェア同士の間のインタフェースとして実現する。ハードウェアとソフトウェアのトレードオフにおける主な評価項目として、性能・コスト・消費電力・柔軟性などが使われる。コスト・柔軟性を優先するのであればソフトウェアで、性能・低消費電力を優先するのであればハードウェアで実現するのが一般的である。トレードオフの対象となる機能モジュールについて、それぞれハードウェアで実現する場合とソフトウェアで実現する場合について性能・コスト・消費電力の見積もりを行い、あらかじめ設定した条件を満足する組み合わせの中から総合的に判断して分担を決める。

図 2.7 に実行時間（性能の逆数）とコストによるトレードオフ評価の例を示す。A はコスト最優先ですべてソフトウェアで実現した場合で、C は性能最優先で大部分をハードウェアで実現した場合である。B は、システム全体の性能評価でボトルネックとなっているソフトウェア部分をハードウェア化することにより、あらかじめ設定された最適解の候補エリアの中におさまった場合である。

(3) ハードウェアの実装方式の選択

ハードウェアとして実現する場合、ASIC（Application Specific Integrated Circuit）または FPGA（Field Programmable Gate Array）で実現したり、SoC（System on a Chip）または SiP（System in Package）で実装したりする選択肢がある。開発の初期は、変更の可能性が高いことから、より柔軟性のある FPGA や SiP で実現し、システムが安定して量産化への移行段階ではコスト・パフォーマンスの高い ASIC や SoC 化に移行することも行われる。

実際の開発では、(1)(2)(3)のトレードオフは独立に行われるのではなく、並行して実施される。また、評価項目として、ここでは性能・コスト・消費電力・柔軟性をあげたが、実際の開発では開発工数（開発期間）、信頼性なども考慮して行われる。

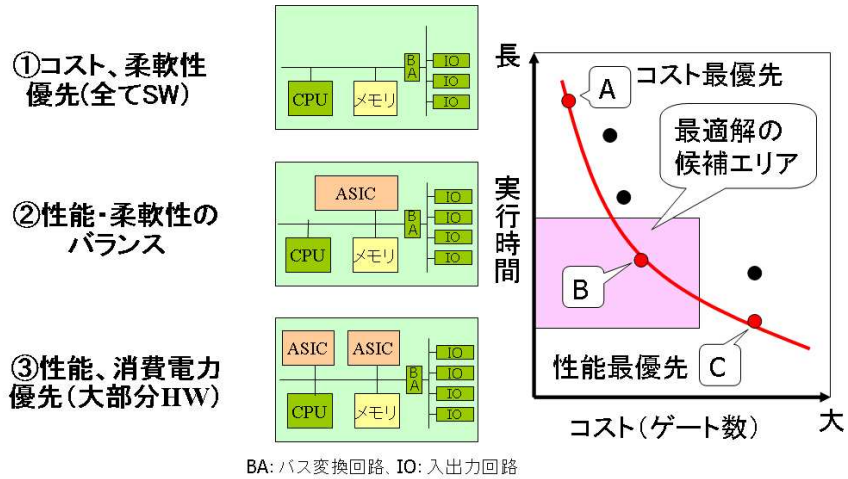


図 2.7　実行時間（性能の逆数）とコストによるトレードオフ評価の例

このトレードオフの結果が、その組込みシステムの価値を決めることから、関連する技術をマスターし経験を踏まえて判断できるシステム・アーキテクトの育成が必要である。

また、設計の上流での見積りを支援する設計システムの充実も重要である。

2.4.4　組込みシステム製品のライフサイクル

組込みシステム製品の開発は、図 2.8 に示すように、1 回だけの開発ではなく、製品展開のロードマップに基づいて、継続的にシリーズ化として開発が繰り返されていくのが一般的である。シリーズ内の製品展開においては、繰り返し行われる改良開発や派生品開発の効率化の観点から、ハードウェアとソフトウェアのプラットフォーム化と IP の活用が行われるようになってきた。図 2.8 の新世代である A シリーズの一連の開発において、最初の A1 の開発においてプラットフォームを確立し新規に開発した部分を IP 化（図 2.8 の A1 の 1～4）していく。次の開発 A2 では、4 の IP を改良して 41 とし、残りは以前の IP をそのまま活用して、短期に性能向上を狙う。さらに、IP の一部を高性能化したり、IP の一部を削除したりして、高機能品から低機能品までの派生品開発を繰り返し行うことによりシリーズ製品の寿命を延ばす取り組みが行われる。また、このようなシリーズ内の製品展開（A シリーズ）と並行して、技術的なブレークスルーを行い、次世代開発（B シリーズ）の立ち上げが行われていく。

組込みシステム製品のシリーズ開発は、図 2.9 の上側に示すように、最初の製品企画から、設計・試作・量産の開発サイクルによる何回かの製品展開を経て、最終的に開発終了・保守フェーズに入り、そのシリーズ製品の一生が終わることになる。このように組込みシステム製品のライフサイクルの観点から、効率の良い実現方法を考えることが重要である。

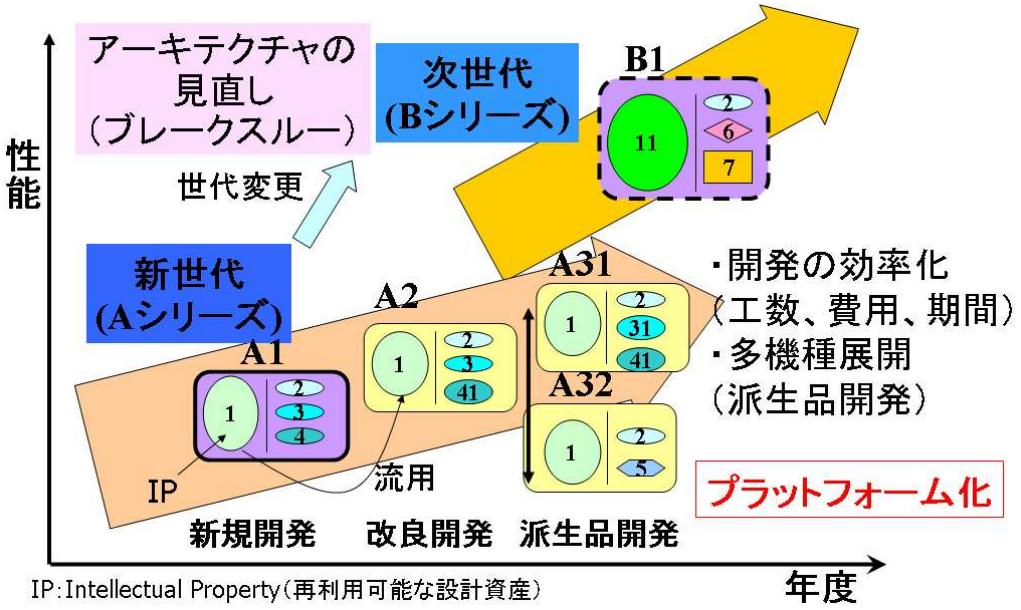


図 2.8　組込みシステム製品の開発ロードマップ

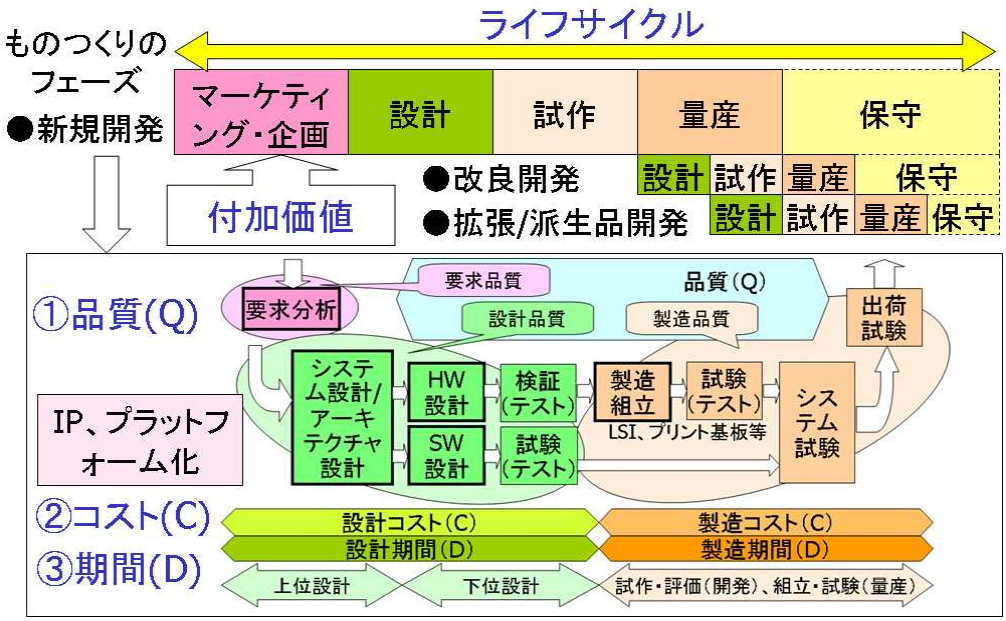


図 2.9　組込みシステム開発のライフサイクルと開発プロセス

開発における重要な管理項目である QCD（Quality、Cost、Delivery）の観点から開発プロセスやライフサイクルの関係を図 2.9 の下側に示す。

①の製品の品質については、顧客のニーズを満足しているかどうかの要求品質と設計が仕様どおりに作られているかどうかの設計品質と製品が不良なく作られているかの製造品質により決まる。特に独創的な製品開発のときには、ニーズを満足しているかの要求品質が特に重要となる。また、大規模化・複雑化が進むにつれ、設計品質の確保が難しくなり、検証手法の充実が必要である。

②の製品のコストについては、設計コストと製造コストから算出される。設計コストについては新規開発部分と再利用部分（IP）に分けそれぞれの出荷個数を考慮して製品コストに反映する必要がある。出荷数が少ない場合にはコスト全体に占める設計コストの割合が高くなり、出荷数が多い場合には製造コストの占める割合が高くなるので、出荷数に応じて、適用する設計方法や実装方法を検討する必要がある。

③の製品の納期については、設計期間と製造期間から算出される。競争の激しい分野では、市場投入時期が遅れると売上機会（先行利益）を逃してしまうことから、できるだけ短い期間で納期が遅れないように品質を確保した上で出荷する必要がある。大規模化・複雑化にともない、設計と検証の効率化（生産性向上）が大きな課題である。

図 2.10 に組込みシステム製品のライフサイクルにおける実現方法の変化を示す。組込みシステム製品のライフサイクルにおいて、協調設計のトレードオフの判断基準が異なることにより、実現方法が異なってくる。①の新規開発時の試作評価や初期開発では、早期に

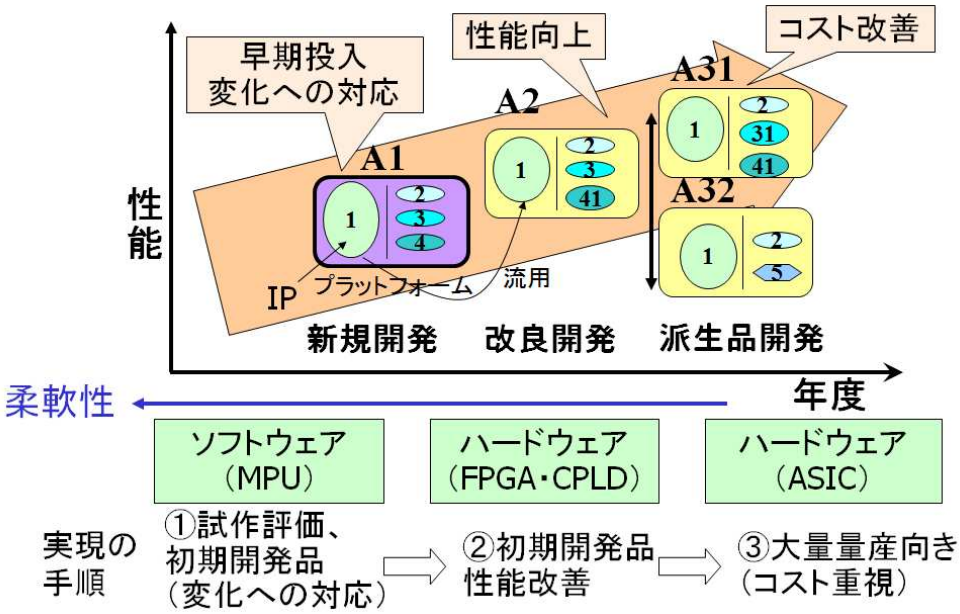


図 2.10　ライフサイクルにおける実現方法の変化

製品レベルの動作が必要であることから変化への対応が容易なソフトウェアで実現し、②の初期開発品の改良開発では仕様や設計品質が安定してきたので FPG A により性能向上をはかり、③の大量量産向けでは ASIC 化によるコスト削減を図るようなことが行われる。

例として、静止画圧縮伸張アルゴリズム（国際標準規格）である JPEG について、ハードウェアとソフトウェアのトレードオフがどのように行われてきたかを図 2.11 に示す。

JPEG のような国際標準規格では、基本仕様のアルゴリズムが C 言語で記述（定義）されることが多い。

①規格が制定されたころは、まだ規格自体にあいまいなところがあったり、実装方法の経験が少ないことから性能を犠牲にしてソフトウェアで実現されることが多い。

②規格が安定し、実装方法もいろいろ研究されてくると、性能向上を実現するためにハードウェア化が行われるようになる。

③プロセッサの性能が向上すると、ソフトウェアでもかなり高速に処理できるようになり、ハードウェア化によるコストを削減するために、ソフトウェアで実現されるようになる。

④微細化が進み、チップ面積に余裕ができ、低消費電力が求められるようになると、ハードウェア化が行われるようになる。

⑤現状では、低消費電力化を求められる組込みシステムではハードウェアで実現され、プロセッサの性能が高い汎用系のシステムではソフトウェアで実現されている。

このように、同じ機能でも、適用する製品の特性やライフサイクルにおける状況に応じて、ハードウェアまたはソフトウェアで実現されることになる。このように、製品のライフサイクルにおいて、それぞれ適した実現方法がとれるように、製品のライフサイクル全体を見通して判断できるようになることが必要である。

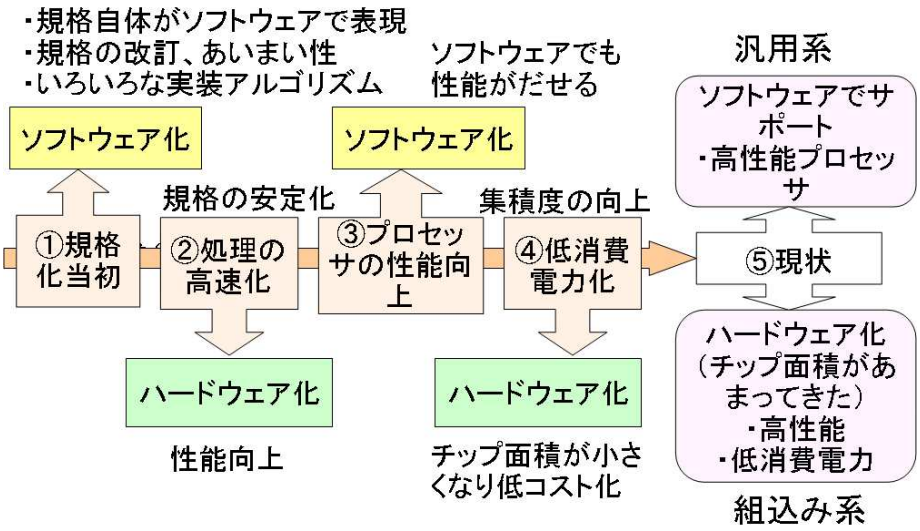


図 2.11　JPEG（静止画圧縮伸張）の HW と SW のトレードオフ例

2.5　組込みシステムにおける開発プロセス

組込みシステムの大部分の機能をシステム LSI とその上で動作する組込みソフトウェアで実現するためには、設計の上流で品質の作り込みを行うと共に、LSI 製造までに不具合を取り除くようにシステムレベルで評価・検証を行う必要がある。しかも、競争の激しい領域ではタイムリーな開発が必要である。従来のようなハードウェアとソフトウェアの分担開発や、LSI 部品を開発するというアプローチでは対応できず、システム全体を見通した効率の良い設計手法と検証手法を確立する必要がある。組込みシステムが、大規模・複雑になるに従い、単にプロダクト（ハードウェアとソフトウェア）の品質だけではなく、要求分析からシステム設計、ハードウェア/ソフトウェア設計、ハードウェア/ソフトウェア実装、テスト、保守までの開発プロセスが正しく行われることが重要となる。

独立行政法人　情報処理推進機構（IPA）のソフトウェアエンジニアリングセンター（SEC）がまとめた「組込みソフトウェア向け開発プロセス」[情報 07B]を参考に、ハードウェア向けの開発プロセスを加えた組込みシステムの開発プロセスの構成を図 2.12 に示す。

従来のまだ規模が小さかった組込みシステムの開発においては、ベテラン技術者が経験に基づいて、応用分野に適したプロセッサ（マイクロコンピュータ）を選定し、ハードウェア設計者が論理設計や基板設計を行い、その仕様に基づいてソフトウェア技術者が組込みソフトウェアを開発してきた。後で不具合がみつかれば、基板を修正したり、ソフトウェアを修正することで、なんとか対応ができた。

しかし、半導体の進歩によりシステム LSI 化が進み、対象規模が拡大し、複雑さが増大するに従い、開発の後半での変更は品質・コスト・納期などの観点から不可能になってきている。このような課題を解決し、付加価値の高い製品を実現するためには、開発の上流

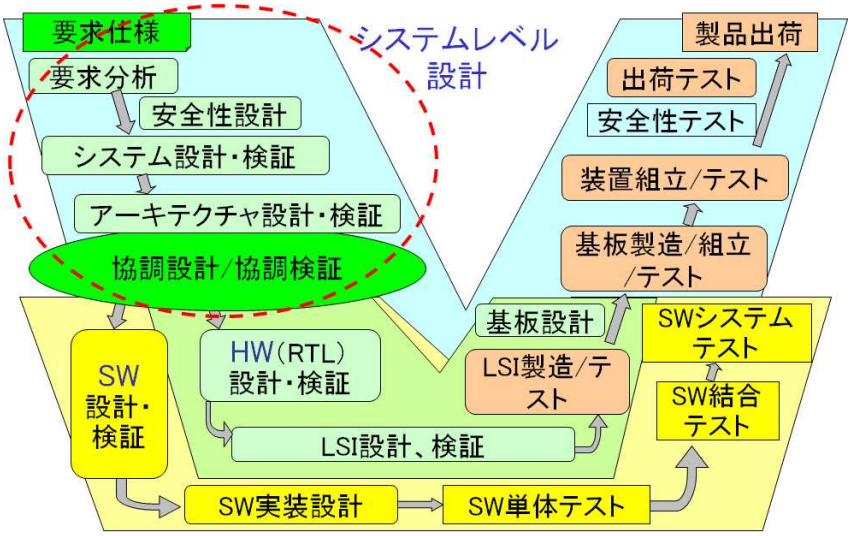


図 2.12　組込みシステムにおける開発プロセス

（IPA の「組込みソフトウェア向け開発プロセス」にハードウェア向け開発プロセスを追加）

のシステムレベル設計において、要求仕様をシステム仕様に落とし込み、システム仕様をハードウェアとソフトウェアで最適に分担して設計・評価を行う協調設計・協調検証が重要となる。

既に、このシステムレベル設計・協調設計・協調検証の分野では、1990 年代の前半から、多くの研究や報告が行われている。協調設計関連として、協調設計を実現する方法として水平分割方式と垂直分割方式があり、前者ではアプリケーションに最適なプロセッサの生成方法とそれに必要なコンパイラの自動生成の研究が、後者ではハードウェアとソフトウェアのインタフェースとプロセッサのモデリングについての研究などが行なわれている[Tho93] [Mic94] [Chi94] [Sat94] [Wol94] [Mic95] [今井 95] [安浦 95] [宮崎 95] [Ada96] [Mic96] [Mic97] [Sta97] [Ern98] [今井 98] [Mic01] [Vah03] [Wol03] [Mit09] [Sch10]。協調検証関連として、プロセッサの検証モデルと周辺のハードウェアの検証モデルを連動させてシミュレーションを行なう方法がいろいろ研究されている[Row94] [Gho95] [Pas97] [野々04]。協調合成・高位合成関連として、ハードウェアとソフトウェアのインタフェース部分のソフトウェアドライバとハードウェアインタフェース回路の合成方法の研究が行なわれている[Gup93] [Hua95]。システムレベル設計関連として、協調設計・協調検証・協調合成を含めたシステムレベルでの取り組みが報告されている[Bal03] [Hen03] [Lee03] [今井 04] [黒坂 04A] [黒坂 04B]、

開発プロセスにおける協調設計・協調検証の位置づけを図 2.13 に示す。

要求分析では、市場ニーズの要求分析から要求仕様をまとめる。要求仕様については、従来は文章で表現されていたが、最近は UML（Unified Modeling Language）による記述

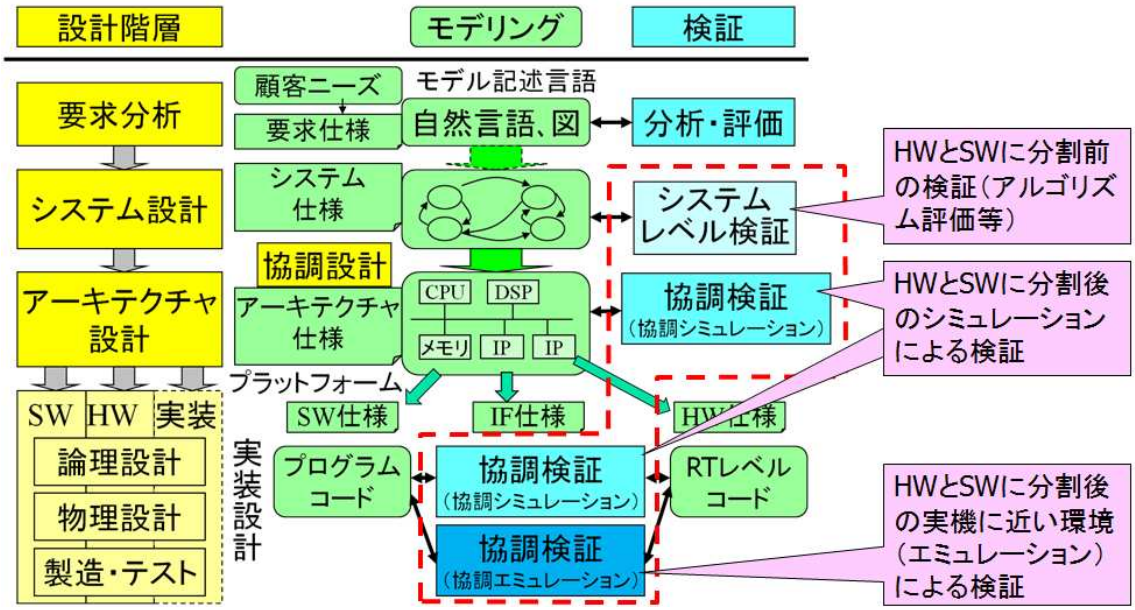


図 2.13　開発プロセスにおける協調設計・協調検証の位置づけ

も行われるようになってきている。

システム設計では、その要求仕様を満足するように、システムを実現するためのシステム仕様をまとめる。システム仕様の記述には、SystemC や C 言語などの高位言語が使われる。この段階では、まだハードウェアとソフトウェアの分担を意識せずに、主にアルゴリズムの最適な実現方法を評価・検証しながら、いくつかの動作モジュールとそれらの間の通信による解を求めることになる。また、Matlab などのモデル記述言語を使って、設計対象（コントローラモデル）とそれにより制御される機器や環境（プラントモデル）をモデル化して、システムの系全体を評価しながら、システム設計を行うモデルベース設計も行われている。この領域は、制御技術分野も含んだ世界であり、広義の協調設計分野といえる[Koi95]。

アーキテクチャ設計では、いろいろな実現方法の選択肢の中から、相反する要件（性能、コスト、消費電力、柔軟性、開発期間など）を評価しながら、対象製品に最適な解を求める設計探索が行われる。

最初に、システム機能としてどの程度のデータ演算量やトラフィック量が必要かを解析するプロファイリングが行われる。このプロファイリング結果に基づいて、性能・コスト・消費電力等の制約条件を満足するように、ハードウェアとソフトウェアの分担を実現する協調設計が行われる。

まず、ベースとなるプラットフォーム（搭載するプロセッサ・メモリ・バス等）を選択する。次いで、動作モジュールをハードウェア（IP/専用回路）またはソフトウェアに割り当てる。また、動作モジュール間の通信部分については、ハードウェアとソフトウェアとのインタフェース回路やハードウェアとの問答を行うドライバ・ソフトウェアに置き換え

ていく。

ハードウェア部分については高位合成により RT レベルに変換され、ソフトウェア部分については高位言語プログラムに変換（ソフトウェア合成）される。インタフェース部分については、インタフェース合成により自動的に変換することも可能になってきている。これらを総称して、協調合成ともいわれる[Mic95]。

また、システム設計では、高位言語でモデル化された内容が仕様どおりであるかをシステムレベルシミュレーションにより確認が行われる。アーキテクチャ設計では、ハードウェアとソフトウェアに分割されたモデルを組み合わせた協調検証が行われる。この段階の協調検証はシミュレーションベースで実行される協調シミュレーションで、分割されたハードウェアモデルと使用するプロセッサモデルの抽象度に応じて、いろいろな実現方法が存在する。

アーキテクチャ設計後は、ハードウェアとソフトウェアの設計が並行して実施されるので、両者を組み合わせて全体として正しく動作するかの協調検証が行われる。この段階の協調検証には、シミュレーションベースでおこなう協調シミュレーションとエミュレータによる協調エミュレーションがある。協調シミュレーションでは、実際のプロセッサがまだ存在しないときに、プロセッサをモデル化して、ソフトウェアの検証に使うことが多い。協調エミュレーションでは、実機に近い環境で応用ソフトウェアを含めた高速検証ができ、人間の感性による評価が可能となる。また、顧客に製品に近い動作を示すこともできる。

LSI の設計規模拡大に対して、プラットフォームや IP を再利用することにより、設計工数の増加を抑えることができるが、検証については再利用したところも含めて全体を動作させる必要があるため、検証規模や検証工数の増加を抑えることが難しい。また、システム LSI 化により、不具合は大きな影響を与えることから、検証環境の充実は非常に重要である。

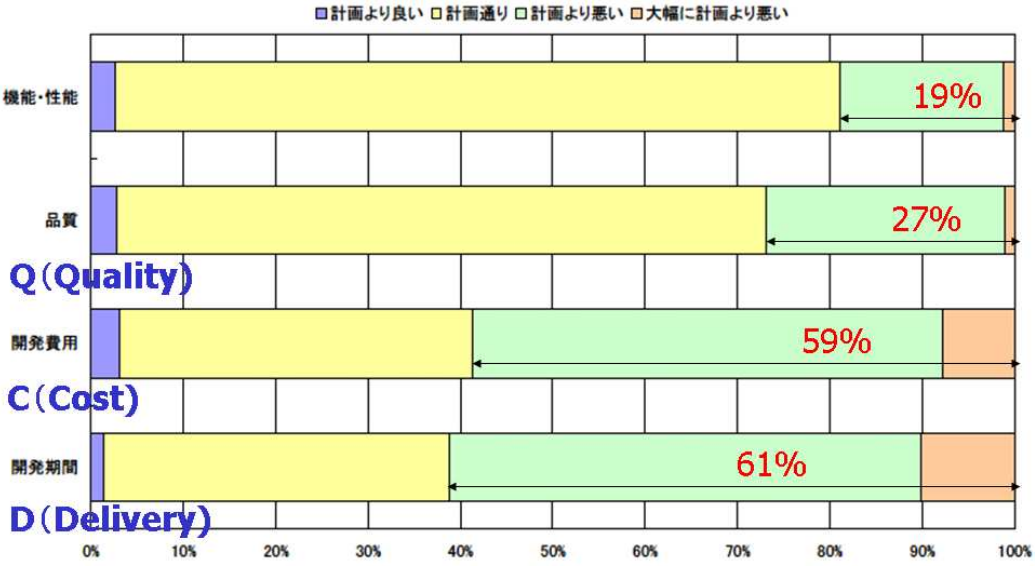
2.6 組込みシステム製品開発におけるプロジェクト

開発プロジェクトの規模が大きくなるほど、QCD の目標を実現するために、PDCA（Plan、Do、Check、Act）のサイクルをまわし、プロジェクトをマネジメントしていくことが重要となる。多くの組込みシステム製品の開発は、プロジェクトリーダー、システムアーキテクト、ハードウェア技術者やソフトウェア技術者を含む関係者（ステークホルダ）によるプロジェクト体制で行われる。組込みシステムの開発プロジェクトでは、ハードウェア技術者とソフトウェア技術者が、お互いのインタフェースを誤解のないように理解して、トラブルなく目標を達成することが求められている。

経産省と IPA が共同で行なった 2007 年版 組込みソフトウェア産業実態調査[経産 07]によれば、組込みシステムの開発計画に対する開発プロジェクトの結果は、図 2.14 のような状況であったと報告されている。機能・性能の面では 19%が目標を未達成であるのに対し

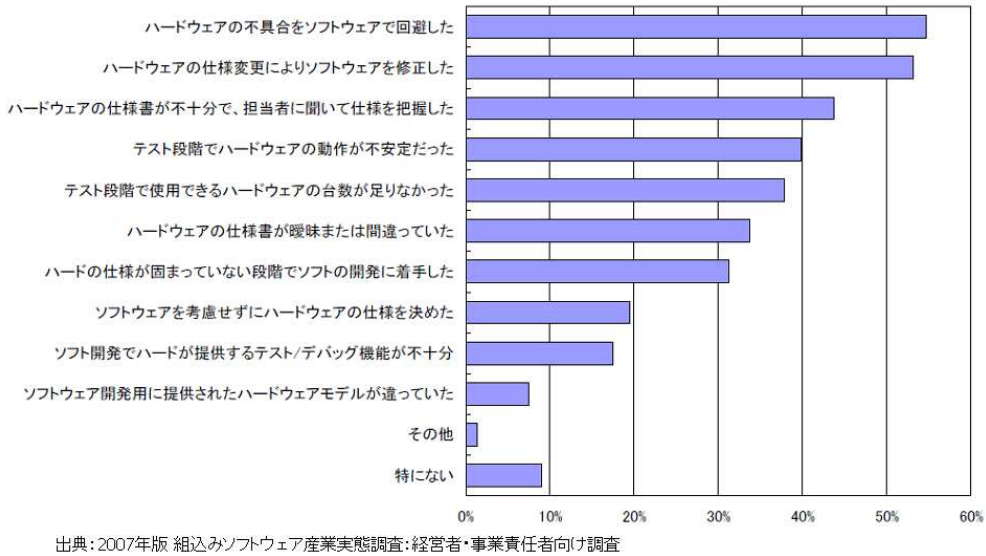
て、品質（Q）面では 27%が、コスト（C）面では 59%が、納期（D）面では 61%が目標を未達成となっており、特に納期遅れが問題である。競争の激しい分野では、納期の遅れが損益に大きく影響するので、不具合やトラブルにより開発期間が延びないように、プロジェクトをマネジメントして行くことが重要である。

また、組込みソフトウェアの開発でハードウェアに起因して直面した課題について図 2.15 に示す。組込みシステムの開発においては、ハードウェアとソフトウェアの分担とその間のインタフェース仕様を決める協調設計が重要であり、ハードウェア部門とソフトウェア部門が連携して取り組むことが必要であることを示している。



出典：2007年版 組込みソフトウェア産業実態調査　https://sec.ipa.go.jp/download/200706es.php

図 2.14　開発計画に対するプロジェクトの結果



出典：2007年版 組込みソフトウェア産業実態調査：経営者・事業責任者向け調査

図 2.15　ハードウェアに起因して直面した課題

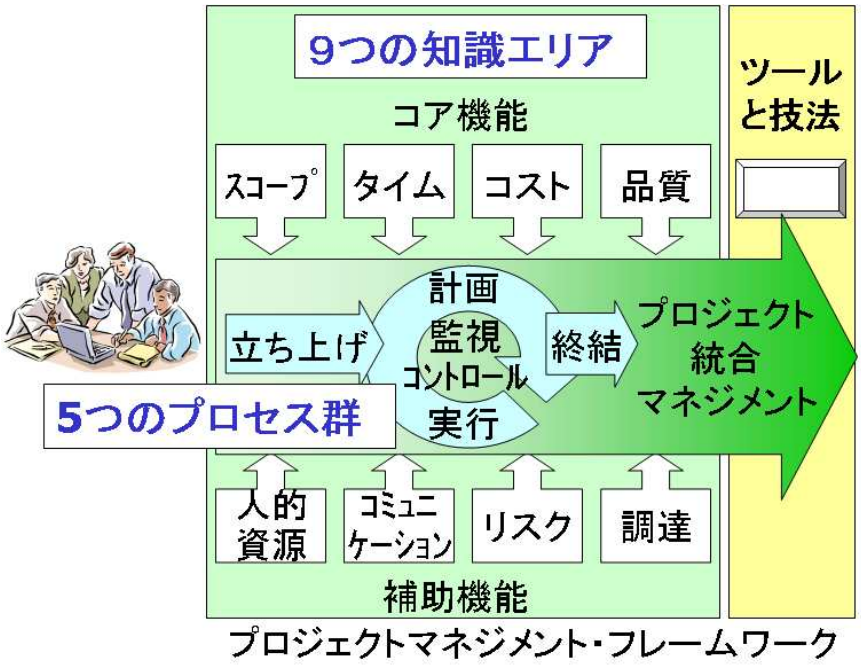


図 2.16　プロジェクトマネジメントの概要

組込みシステムの大規模な開発プロジェクトの推進においては、図 2.16 に示すような PMBOK（Project Mangement Body Of Knowledge）に基づいた取り組みが必要である。2000 年代の PMBOK は、9 つの知識エリアと 5 つのプロセス群とツールと技法から構成されている。9 つの知識エリアは、プロジェクトを進める上で必須の機能（コア機能）であるスコープ・タイム・コスト・品質と、補助機能である人的資源・コミュニケーション・リスク・調達と、統合から構成されている（なお、2013 年の第 5 版でステークホルダーが追加されて 10 の知識エリアとなっている）。組込みシステム製品のユーザを含む関係者（ステークホルダー）の要件や期待に応えるように、プロジェクトの対象範囲（スコープ）を明確に定義し、目標とする品質（Q）・コスト（C）・タイム（納期：D）の実現を目指して、人的資源・コミュニケーション・リスク・調達および統合のマネジメントを計画的に行い、立ち上げ→計画→監視・コントロール→終結の 5 つのプロセス群を回していくことが必要になる。

複合領域にある組込みシステムの開発は、大規模化・複雑化が進み、図 2.14 に示したように、目標通りに開発を完了することが難しい面があるが、プロジェクトを正しく進めるためのスキルや仕組みが不足している場合も多い。プロジェクトマネジメントに関するスキルに関しては、PMBOK のような知識レベルでの学習だけでは十分でなく、実際のプロジェクトを模擬した PBL（Project Based Learning、Problem Based Learning）型の演習を通じて経験しながら修得していくことが必要になっている。PBL については、チームにより目標を達成することから、コミュニケーション力の育成などの基本的な能力育成もかねて、高等教育機関や企業研修の場で行われるようになり、成果を上げてきている。なお、

負荷が特定の人に偏ったり、チームの編成の仕方や評価方法についていろいろな課題が指摘されている。

2.7　組込みシステムの開発に必要なスキル

プロジェクトを構成する技術者は、その担当職務に必要となる技術やスキルを習得しておく必要がある。組込みソフトウェア開発力の強化の一貫として、経産省・IPA により制定された組込み技術スキル標準（ETSS、Embedded Technology Skill Standard）[情報 09A]をベースに考える。ETSS は、IPA の SEC において 2005 年に策定され、スキル標準、キャリア標準、教育研修標準から構成されている。

組込みシステム技術者に必要な技術やスキルは、多様であることから、体系的な教育カリキュラムを設定して教育を行う必要がある。特に、付加価値の高い競争力のある組込みシステム製品の実現のために、ハードウェアとソフトウェアに精通したシステムアーキテクトの育成が求められている。

ユーザが満足する組込みシステム製品を開発するためには、開発に必要な業務を担当するそれぞれの技術者（ピープル）がそれぞれの立場・役割（職務）で、対象となる組込みシステム製品（プロダクト）を高い品質で作り込み、正しい組込みシステム開発プロセスをまわし、目標とする QCD を達成するようにプロジェクトを動かしていく必要がある。このように組込みシステム技術者が振舞うためには、それぞれの職務にふさわしいスキルセットを適切なレベルで保持して、活用することが必要になる。

組込みシステムが大規模化・複雑化するに従い、設計生産性の向上や品質向上が求められるようになり、従来の自己流ではなく、標準的なスキル体系に基づいた技術者育成が求められている。

ETSS のスキル標準のフレームワークでは、必要とするスキル群を 3 つのスキルカテゴリ（技術要素、開発技術、管理技術）に分類し、スキル粒度として最上位の抽象的な第一階層からより詳細な第三階層まで階層的に展開し、最下層のスキルに対して必要なスキルレベル（初級、中級、上級、最上級）を設定している。

組込みスキル標準 ETSS をベースに組込みシステム開発に必要となるスキル体系（ETSS スキル基準第一階層）をまとめたものを図 2.17 に示す。ETSS は 3 つのスキルカテゴリである技術要素、開発技術、管理技術から構成されている。ここで、技術要素がプロダクトに、開発技術がプロセスに、管理技術がプロジェクトに対応している。

ETSS では、技術要素として、以下に示すように 7 つの第 1 階層（コロンの後に第 2 階層を示す）について例示している。

① 通信：有線、無線、放送、インターネット
② 情報処理：情報入力、セキュリティ、データ処理、情報出力
③ マルチメディア：音声、静止画、動画、統合
④ ユーザインタフェース：人間系入力、人間系出力

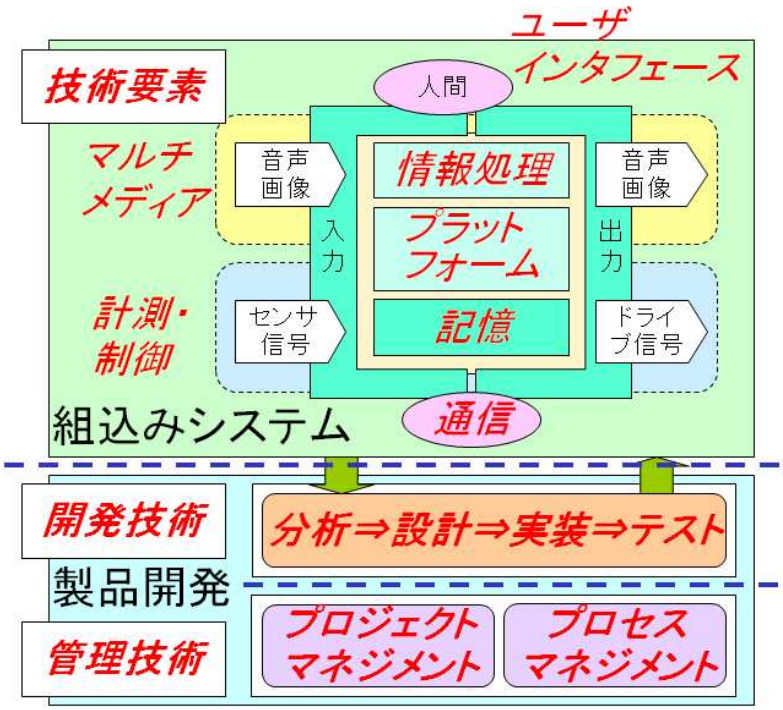


図 2.17　組込みシステム開発に必要なスキル体系（ETSS スキル基準第一階層）

⑤ 記憶（ストレージ）：メディア、インタフェース、ファイルシステム

⑥ 計測･制御：理化学系入力、計測・制御処理、理化学系出力

⑦ プラットフォーム：プロセッサ、基本ソフトウェア、支援機能

ETSS は、組込みソフトウェア技術者向けを想定しているが、組込みシステムとしての要求分析からシステム設計の上流も対象にしていることから、技術要素（プロダクト）はそのまま組込みシステム技術者にも適用可能であると考える。

開発技術（プロセス）については、ハードウェアとソフトウェアでは大きく異なることから、ハードウェアについては、新たな定義が必要となる。九州大学の「システム LSI 設計人材育成実践プログラム（QUBE）」では、システム LSI 設計技術のスキルマップを提案し、教育効果の測定やカリキュラムの改善への取り組みが検討されている[林田 08]。

管理技術（プロジェクト）の基本形は、組込みシステム全体で共通と考えられる。

ETSS は、本研究の 1 つである「複合領域での教育手法をベースとした組込みシステム技術者教育の提案と実践評価」でのカリキュラムの検討のベースとなっている。第 4 章の 4.2.1 項で ETSS のスキル標準と本研究での対応を示している。

2.8　IoT 時代の組込みシステム

マイケル・ポーターは、「IoT 時代の競争戦略」[マイ 15]で、IoT を「第 3 の IT 化の波」と位置付けて、モノづくりの世界が製品プロダクトの販売からサービスの提供に大きく転換すると分析し、組込みシステムの事業領域の変遷を農業用トラクターを例に図 2.18 のよ

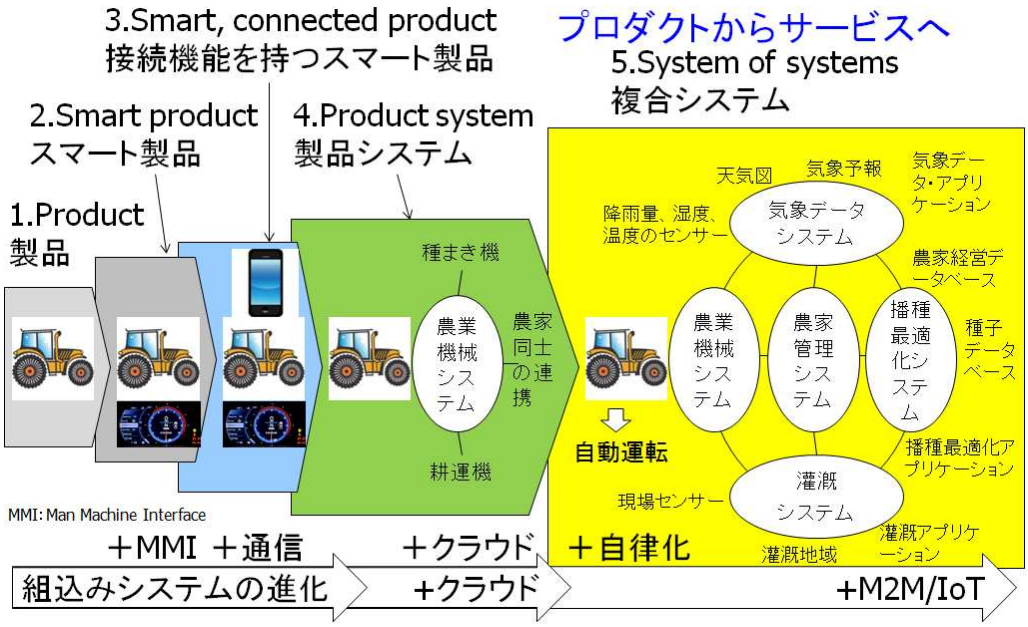


図 2.18　組込みシステム事業領域の変化（プロダクトからサービスへ）[マイ 15]

うに示している。従来の組込みシステムでは、農業用トラクター単体の製品（プロダクト）として販売されてきた。それに対して、パネル表示が見やすくなり（スマート製品化）、さらに通信機能でスマートフォンに各種情報を伝えることができるようになってきた。IoT 時代を迎え、農業用トラクターに設置されたセンサ情報などが、インターネットを通じて、クラウド上に送信され、農業機械システムとして稼働状況の見える化や他の農業機器との連携が可能になってきた。さらに、クラウド上での農業関連システムとの連携による複合システム（System of Systems）に進化して、従来の製品単体の販売モデルから農家管理システムや播種最適化システムなどのサービスを提供するモデルに変わると予測されている。究極的には位置情報などや周囲の環境情報からトラクターを無人で自動運転をするなどの自律化も期待されている。

このように、IoT システムは、組込みシステムとクラウド上のアプリケーションと通信ネットワークにより構成される複合システムといえる。通信ネットワーク化された組込みシステムにおいて、所有からシェア（共有）することに価値が移るとビジネスモデルも大きく変わると予想されている。このような IoT 時代を迎え、組込みシステムの位置づけも大きく変わってきていると言える。

IoT システムとその中での組込みシステムの位置づけを図 2.19 に示す。組込みシステムはユーザ側に近いエッジ側に位置づけられ、アクセスネットワーク（広域通信網）を介して、クラウドと接続されている。組込みシステムはセンサやアクチュエータを取り扱うデバイス群とそれらをエリアネットワーク（狭域通信網）で接続して全体を制御するゲートウェイから構成されている。自動車でいえば、デバイスは各種センサユニットやアクチ

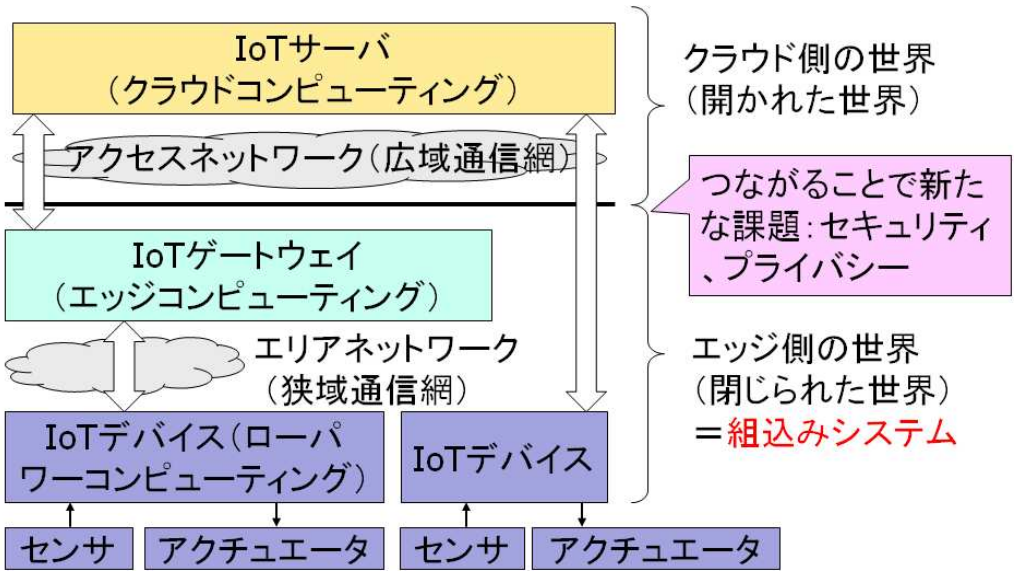


図 2.19　IoT 時代における組込みシステムの位置づけ（IoT システムの全体像）

ュエータユニットに、エリアネットワークは車内ネットワークの CAN バス等に、ゲートウェイについてはエンジン制御ユニット等に相当する。

携帯電話などの通信系機器を除いて、組込みシステムがアクセスネットワーク（広域通信網）でクラウドにつながることはほとんどなかった。IoT 時代を迎え、組込みシステムが通信ネットワークでクラウドにつながるようになってくると、以下のような 4 つの新しい課題が出てくると考えられる。

・1 つ目は、大量のデバイスからのデータ量が少ないセンサデータを如何に低コスト・低消費電力で通信を行なうか。
・2 つ目は、収集したセンサデータなどをどこでコンピューティング処理するか。
・3 つ目は、データのビッグデータ化と人工知能（機械学習）の実用化に対応してコンピューティング能力を如何に提供するか。
・4 つ目は、クラウドにつながることで今まで組込みシステムで考えてこなかったセキュリティやデータの利活用とプライバシーについてどのように扱っていくか。

以下、順番にそれぞれの課題について説明する。

(1) 低コスト・低消費電力な通信ネットワーク

図 2.19 に示すように、狭い領域に設置されたデバイスからのセンサデータをエリアネットワーク経由ゲートウェイに集約してから、アクセスネットワーク経由でクラウド上のサーバに送信する場合と、組込みシステムからアクセスネットワーク経由で直接サーバに送信する場合がある。

このエリアネットワークとアクセスネットワークから構成される IoT ネットワークの全

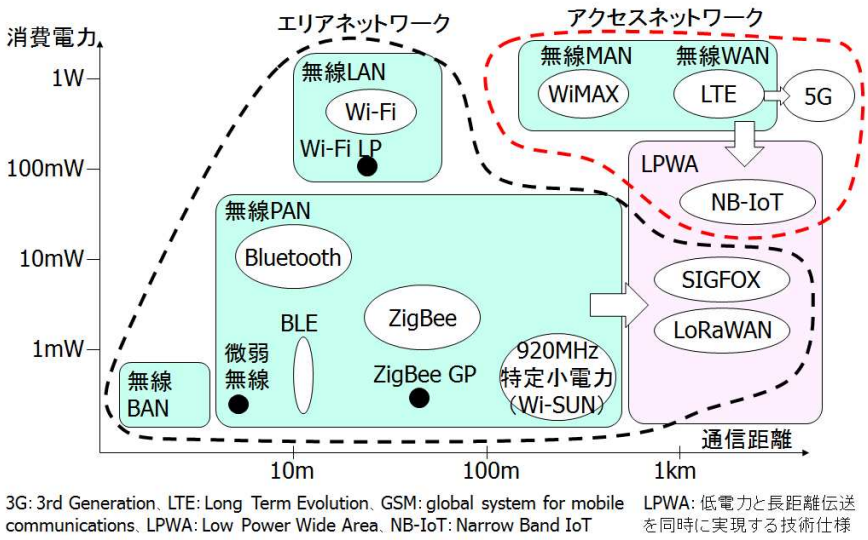


図 2.20　IoT ネットワークの全体像

体像を図 2.20 に示す。エリアネットワークで低消費電力なネットワークは BLE、ZigBee、Wi-SUN が該当する。アクセスネットワークでは 3G/LTE の通信モジュールでの消費電力は小さくなってきているが限界があり、現状では長い通信距離と低消費電力の両方を両立させた LPWA（Low Power Wide Area）ネットワークの SIGFOX、LoRAWAN、NB-IoT が注目されている。

また、アクセスネットワークでの通信プロトコルについても低消費電力化を指向した軽量のプロトコルである CoAP/6LoWPAN や MQTT が注目されている。

実際に低コスト・低消費電力の通信ネットワークを実現するために、それぞれの通信方式や通信プロトコルを評価して、最適な通信システムを構築していく必要がある。

(2) エッジコンピューティングとクラウドコンピューティング

組込みシステムからのセンサデータなどは、通常はクラウド上のサーバに送信され、クラウド上に蓄積されたビッグデータに対して、統計解析や機械学習などの分析処理が行なわれる（クラウドコンピューティング）。クラウドに送信して処理して結果を得るまでのレスポンスタイムは 10ms 以上かかる。リアルタイム性を要求される場合や、アクセスネットワークでのトラフィックがネックになる場合やセキュリティの観点から、クラウドではなく閉じられた世界であるゲートウェイの位置づけで分析処理が行なわれるケースがある（エッジコンピューティング）。

また、外部電源から電力が供給されず遠隔に配置されるデバイスの場合には、徹底した低消費電力化を図る必要がある（ローパワーコンピューティング）。この場合は、コンピュータ能力の少ないマイコンで動かす必要があり、コンピュータパワーを必要とする処理は、よりコンピュータパワーのあるゲートウェイやサーバで処理することになる。

このように IoT システムでは、アプリケーションに応じて、コンピューティングの階層を使い分ける必要がある。

(3) コンピューティング能力の提供方法

ビッグデータの統計解析や機械学習などの分析処理では、高性能な並列処理計算が必要とされる。これを実現するために当初は汎用高性能 CPU が使われてきたが、最近では汎用グラフィック処理ユニットである GPGPU が使われるようになってきた。さらに低消費電力で性能向上を目指して FPGA によるカストマイズされた処理が注目されている。コンピューティング能力の実現方法として、歴史的には汎用化と専用化の繰り返しが行なわれてきたが、現在の IoT 時代は専用化の流れに入りつつあるともいえる。

(4) セキュリティおよびデータの利活用とプライバシー

従来、通信機器以外の組込みシステムは外部とネットワークにつながれず、セキュリティについてはあまり考慮されてこなかった。IoT 時代を迎え、インターネットなどでクラウドに接続されるようになると、従来の Web システムでのセキュリティとは異なった対応が必要になる。人を介さないでネットワークに接続されること、低消費電力化による低性能なマイコンでの実現など課題も多く、実システムではセキュリティ問題の解決が必要となる。また、豊富なデータを所有することがサービスの品質向上に貢献し価値の源泉になると言われている。プライバシー問題を回避するために匿名化処理を行なうことにより、データをうまく利活用することが望まれている。

2.9 組込みシステム開発における課題

2.1 節から 2.7 節まで述べてきた 1990 年代後半から 2000 年代にかけての組込みシステムの現状を分析すると、設計の課題、検証の課題、および、技術者育成の 3 つの課題に分類される。この中で設計の課題については、設計手法の革新を狙いとした「トップダウン協調設計システム」の取り組みなどを通して解決に取り組んできた[清尾 99A][遠藤 00]。また、IP の活用やプラットフォームベース設計、抽象度の高い高位言語と進化した高位合成ツールの活用により、設計の生産性は大きく向上してきている。このように設計の課題は大きく改善されてきているので、残された検証と技術者育成についてそれぞれの課題と解決策について以下に述べる。

2.9.1 検証の課題とその解決策

検証の課題をまとめると以下のようになる。

- 組込みシステムにおける協調設計では、ハードウェアとソフトウェアの分担を性能・コストなどの観点からトレードオフを行い、LSI 作りの前に事前に評価・検証を繰り返し行って、最適な分担を決めておく必要がある。

- 組込みシステム自体が 1 チップのシステム LSI で実現されるようになり、不具合による LSI の作り直しは、開発コストの増大や開発期間の大幅な遅延を招くことになる。LSI 作りを行う前に、事前に不具合がないかを、システムレベルでハードウェアとソフトウェアを含めて検証を行うことが必要となる。このようなソフトウェアを含めた検証は協調検証と呼ばれている。
- 独創的なアイデアを取り込んだ組込みシステムの開発では、LSI 作りの前に、実機に近い環境でハードウェアとソフトウェアを動作させて人間の五感による評価や動くものによる実証評価を繰り返すことも必要である。
- 組込みシステムの大規模化・複雑化に対して、システム LSI の設計では再利用可能設計資産である IP を活用し、新しい追加機能のみを高位設計言語を使って記述し高位合成ツールを使って処理すれば、対処可能になってきているが、検証に関しては、再利用された IP を含めてシステム全体の検証が必要となり、検証規模・検証コストは拡大する一方である。
- 競争が激しい分野では短期開発が求めれ、検証の効率化が求められている。

このような課題に対して、今回対象とする組込みシステムに関する検証に求められる要求をまとめると以下にようになる。

① 組込みシステムのいろいろな機能をハードウェアとソフトウェアで分担して実現する場合、まず最初に対象となるいろいろな機能を高位言語で表現したモデルを作成し、正しい機能がモデル化されたかを検証する。その後、ある機能のモデルをハードウェアにした時とソフトウェアにした時の性能・コストなどを評価し、全体として満足解が得られるまで繰り返し評価を行う必要がある。

② ハードウェアとソフトウェアの分担が決まったら、その間のインタフェースの実現方式を決め、ハードウェアとソフトウェアによる協調検証により、システム全体としての動作を検証する必要がある。

③ 独創的なアイデアを取り込んだ組込みシステムの開発では、人間の五感による評価や動くものによる実証評価が必要であることから、実機に近い環境でハードウェアとソフトウェアを動作させる協調検証も行う必要がある。この段階で不具合が見つかった場合には、最初の①の段階に戻って、改善を行う必要があり、①から③までの繰り返し検証作業を LSI を作る前に効率良く行う必要がある。

従来の検証手法では、①から③まで、個別に検証が行われてきた。今回のように①から③までを、繰り返し効率良く検証を行うためには、トップダウンにハードウェアとソフトウェアを組み合わせて効率良く検証を行う協調検証手法が期待される。

本論文では、第 3 章でトップダウン協調検証システムを提案し、マルチメディア系組込みシステム（映像圧縮処理システムなど）2 件の事例に適用した成果について論じる。

2.9.2　技術者育成の課題とその解決策

技術者育成の課題をまとめると以下のようになる。

- 組込みシステムは、応用分野に対して関連するいろいろな技術を統合して実現されることから、幅広いスキルの修得が必要である。
- 組込みシステムは、ハードウェアとソフトウェアという異なる技術で実現される複合領域にあることから、両方のスキルを修得しておく必要がある。
- 組込みシステムは、過去の技術の積み重ねの上に構築されており、内部のブラックボックス化が進んでいる。創造的な製品を開発するなどブレークスルーを図る場合や、複雑なトラブルを解決する場合などは、内部の基本的な原理・原則を理解しておくことが求められる。
- 組込みシステムの価値は、ハードウェアとソフトウェアの分担を決めるアーキテクチャ設計までの段階で決まることから、性能・コストなどの相反する制約条件のもとで要求事項を満足するようにハードウェアとソフトウェアの協調設計により実現を図るシステムアーキテクトの役割は大変大きい。

特に、独創的な機能を新たに開発する場合には、ブラックボックス化した内部を含めて、新しい視点でブレークスルーを図ることができるシステムアーキテクトの育成が大変重要になる。既に、組込みシステム分野では、組込みソフトウェアやシステム LSI 開発者の立場から、いろいろな取り組みが行われているが、前述の課題を解決するためには、組込みシステム開発の課題を踏まえて体系的な技術者育成の取り組みが必要と考える。

筆者はいろいろな活動に参画した経験を踏まえて、第 4 章において、複合領域である組込みシステム分野の技術者育成手法について提案し、その実施成果を論じる。

2.9.3　組込みシステムを包含した IoT システム開発について

組込みシステムを包含した現在の IoT システムの開発においても、1990 年代後半から 2000 年代と同じように設計の課題、検証の課題、および、技術者育成の課題があると考える。組込みシステムの技術に加えて、さらに通信ネットワークとクラウドコンピューティングに関する技術を組み合わせて、サービスとしてのアプリケーションにふさわしい IoT システムを構築するには、幅広い技術力と全体を見通せる能力を必要とする。

IoT システムの開発においては、新しい価値の創出が期待されており、その開発方法として迅速にプロトタイプ作成を繰りかえし行なう「デザイン思考」のアプローチが注目されている[黒川 12]。何回かのプロトタイプ作成を経て、アプリケーションに最適になるように、構成要素であるハードウェア、ソフトウェア、および通信ネットワークを組み合わせて実現していくことになる。これは、組込みシステムにおける「ハードウェア」と「ソフトウェア」による協調設計に「通信ネットワーク」を組み込んで拡張したことにも相当する。

また、ポスト「京」プロジェクトおいて「スーパーコンピュータ開発」と「アプリケーション開発」の協調設計（コザイン)」の必要性が提唱されており[ポスト 17]、アプリケーションにとって最適になるように取り組む協調設計は今後とも必要なアプローチと考えられる。

設計の課題については、インタフェースの標準化、機能のモジュール化や API（Application Program Interface）化が進み、プロトタイプ事例[秋山 16]やノウハウの公開サイトが充実してきており、使い方を理解すれば比較的容易にシステムを構築できるようになってきている。

それに対して、検証の課題については、IoT システムによる新たな価値創出が期待されており、その効果を早期に確かめるために、「デザイン思考」のアプローチとしてプロトタイム作成・評価の頻繁な繰り返しが期待されている。

また、技術者育成の課題については、組込みシステム技術に加えてさらに幅広い技術を習得する必要があり、さらに新しい価値を創造し、評価する能力を持つことが期待されており、技術者育成の取り組みはますます重要になってきている。

本節での議論は本論文の主体ではないが、後の章において、IoT 時代と言われる 2017 年の今の視点からの考察を加えている。

2.10　まとめ

1990 年代後半から 2000 年代にかけての最先端の組込みシステム開発における現状と課題について、ものづくりの成果物である応用目的に最適化された組込みシステム製品（プロダクト）、いろいろな技術を組み合わせた複雑系の組込みシステムを開発する手順（プロセス）、組込みシステムの開発を目標通りに進めていくプロジェクト、プロジェクトに参画する技術者（ピープル）に必要なスキルの観点から整理を行なった。

主な課題として、設計の課題と検証の課題と技術者育成の課題の 3 つが考えられるが、設計の課題については、IP の活用やプラットフォームベース設計、抽象度の高い高位言語と進化した高位合成ツールの活用により、大きく改善されてきている。

本研究では残された検証の課題と技術者育成の課題を対象として、課題の分析と解決策を検討した。検証の課題に対しては、設計の上流から下流までハードウェアとソフトウェアを組み合わせて効率良く検証を行うトップダウン協調検証手法を提案し、第 3 章で具体的な研究内容について論じる。技術者育成の課題については、今後重要な役割を担うと期待されているシステムアーキテクトの育成を目標に、複合領域である組込みシステム分野の技術者育成手法について提案し、第 4 章で具体的な研究内容について論じる。

また、IoT 時代と言われる 2017 年の今の視点から、組込みシステムを包含した IoT システムの開発における現状と課題についても考察を行ない、検証の課題と技術者育成の課題の解決が重要であることを示した。

第 3 章　トップダウン協調検証手法の提案と適用評価

本章では、1990 年代後半から 2000 年代にかけての組込みシステムにおける検証の課題に対して、設計の上流におけるシステムレベルの検証ステージから、ハードウェアとソフトウェアによる協調シミュレーションの検証ステージを経て、実機に近い環境での協調エミュレーションの検証ステージまで、効率よく繰り返しながら検証を行っていくことができるトップダウン協調検証手法を提案し、その適用評価について論じる[Seo97][Seo98][YaM98][安田 98][安田 99]。

3.1 はじめに

組込みシステムの最初のシステム設計の段階で、ターゲットシステムのアーキテクチャとアルゴリズムが、ターゲットシステムに求められる機能を満足するか評価される。評価の結果が満足されるまで繰り返し変更が行われる。満足な結果が得られると、組込みシステムはハードウェアとソフトウェアに分割される。このハードウェアとソフトウェアによる協調設計の段階では、最初はともに C 言語などの抽象的なレベルで記述したモデルを使って、ハードウェアとソフトウェアによる協調シミュレーションにより機能と大まかな性能の評価が行われる。設計の詳細化が進むにつれ、順次より詳細なレベルで表現されたモデルに置き換えて、より精度の高い評価が行われる。さらに、実時間による評価や人間の感性による評価が必要な組込みシステムにおいては、実機に近い環境でのハードウェアとソフトウェアによる協調エミュレーションによる評価が必須となる。特に、独創的なマルチメディア系や制御系などの組込みシステムを開発する場合には、設計の上流から実機に近い環境まで、トップダウンで繰り返しながら検証を効率よく短期間で進めていくことが求められる。これを実現するためには、従来各設計の段階ごとにバラバラに進められてきた検証手法に対して、検証環境の統合や共通テストベンチの適用とともに、ハードウェアとソフトウェアのインタフェースと、ハードウェアとソフトウェアをそれぞれどのように表現して検証モデルを作成していくかというモデリング手法を明確にする必要がある。

ハードウェアとソフトウェアのインタフェースとモデリング手法に関しては、チャネルベースコンポーネントアーキテクチャ[Lin96]やバーチャル CPU アプローチ[Sch96]で報告されている。前者では、コンポーネントの間のインタフェースとその動作仕様は、システムレベルで協調合成を行うために定義されている。後者のアプローチでは、CPU のモデルと他のコンポーネントとのインタフェースは、協調シミュレーション環境を実現するために定義されている。

本章では、検証環境の統合と共通テストベンチの適用、および、コンポーネント論理バス（CLB：Component Logical Bus）アーキテクチャによるハードウェアとソフトウェアのインタフェースの統一により、設計の上流から下流に向けて各検証ステージで評価を繰り返しながら効率良く検証を行っていくトップダウン協調検証手法を提案する。

3.2 節で、組込みシステムにおける検証の課題を述べ、3.3 節で階層的な検証における現状と課題を整理し、3.4 節で提案するトップダウン協調検証手法の概要を述べる。3.5 節で提案したトップダウン

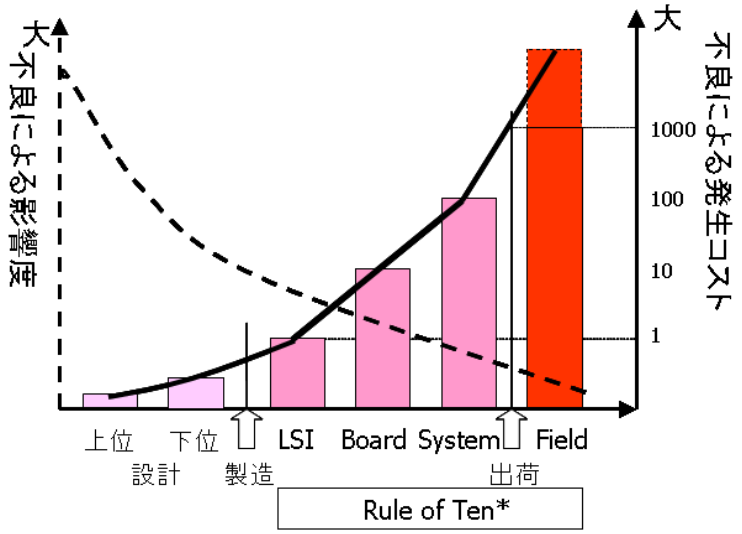
協調検証手法を評価するために構築した 2 つの事例の概要とその評価結果について述べる。3.6 節で考察を行い、最後の 3.7 節でまとめを行う。

3.2　組込みシステムにおける検証の課題

組込みシステムの開発は、要求分析、システム設計、アーキテクチャ設計、ハードウェア設計＋ソフトウェア設計のように設計の上流から下流へと進んでいく。各設計フェーズで不具合が残ったまま下位のフェーズに移行すると多大の影響を及ぼすことから、それぞれのフェーズで不具合を取り除くための検証が必要である。特に、機能の大部分がシステム LSI で実現される場合には、製造フェーズに入る前に不具合を取り除くことが必須である。

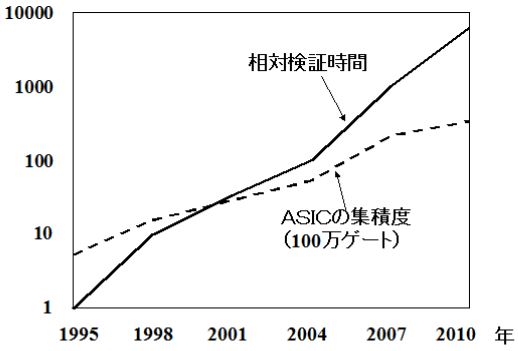
図 3.1 に開発の工程において発見される不良による影響度と発生コストを示す。横軸に設計・製造・出荷の流れを示す。左側の縦軸は不良による影響度を表し破線で示す。右側の縦軸は不良による発生コストを表し実線で示す。設計不良を見逃して、製造の後工程で設計不良が見つかる程、その不良による対策コストは膨大なものになる。製造工程での不良発見による発生コストは、10 倍の法則（Rule of Ten）に従うと言われている[AMB99]。LSI 段階で見つかったときの発生コストを 1 と仮定すると、モジュールの段階では 10、製品段階では 100、出荷後で 1000 ものコストがかかることを意味している。

特に、システム LSI を含んだ組込みシステム製品を開発する場合には、要求品質や設計品質が十分でなく不良が後工程で見つかると、不良による発生コストと開発遅延による売上の機会損失は膨大なものになり、その組込みシステム製品の事業にも影響を及ぼす可能性が高い。設計上流でのシステムレベルでの評価、アーキテクチャ設計段階でのハードウェアとソフトウェアによる実現における機能や性能の検証・評価、さらに実機に近い環境での実際のデータを使ったシステムレベルの検証・評価が必要である。



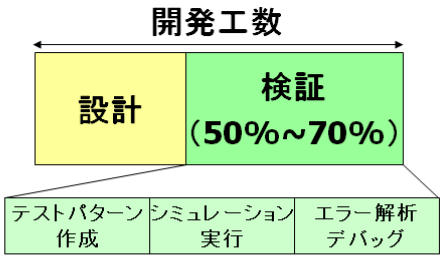
＊出典：T. AMBLER, B. BENNETTS, "Test and the Product Life Cycle," IEEE Design & Test of Computers, July-September 1999, Figure1(p.21) より引用

図 3.1　不良による影響度と発生コスト



「LSI設計力強化に向けた共同活動の提言」半導体産業研究所＋日本電子機械工業会(1998/3)、International Symposium on Physical Design(April,1997)

(a)検証時間の急速な増大



(b)検証工数の占める割合が増加

表. ITRS2007(＋2005)による予測工数	2005	2007	2009	2012	2015	2018
設計工数(SoC)(Mtr/年/10人チーム)(注)	3.3	5.4	10.6	24.6	73.4	113
検証工数(SoC)(Mtr/年/10人チーム)	4.7	8.0	17.6	39.8	91.8	210.9

(注)設計工数については改訂されておらず、ITRS2005で記載されている以前の値のままを表示

nn.n: 解決策が知られていない

http://www.itrs.net/Links/2007ITRS/Home2007.htm(2009/02/17 現在)より抜粋し編集

(c)ITRS2007による予測工数

図 3.2　検証時間、検証工数の推移

また、組込みシステム製品（システムLSI）の大規模化が進んでおり、検証にかかるコスト（検証時間）は図 3.2(a)に示すように設計規模の拡大に比べ急激に増加している。設計コストについては、プラットフォーム化+IP化による再利用設計と高位言語+高位合成により改善される方向にあるが、検証は設計で再利用された部分も含めて全体を扱うために、検証時間（図 3.2(a)）と検証工数（図 3.2(c)）は増え、全体の開発工数に占める割合も増加（図3.2(b)）している。DVCon2011のキーノートスピーチ[Rhi11]では検証のプロジェクト全体に占める割合は55%と言われている。検証規模の拡大に対して検証を高速に実行できる環境が必要になる。設計の上流から下流までのそれぞれの検証段階で、検証目的に対応して実用的なレベルで検証・評価に耐える検証環境の構築が求められる。

特に、競争の激しいマルチメディア系や制御系などの分野で、独創的な製品を新たに開発する場合には、新しいアルゴリズムの考案から、ハードウェアとソフトウェアによる実現方法についてトレードオフを行い、最終的に組込みシステムとして実機に近い環境で目標とする機能・性能を持っているかを確認するまで、繰り返し階層的に検証することが必要とされる。

3.3.　階層的な検証における現状と課題

設計の上流でのシステムレベルの検証ステージから、実機に近い環境による検証ステージまで、検証は階層的に行なわれる。最初に、検証ステージの階層を説明し、従来の階層的な検証の取り組みの課題を示す。

3.3.1　階層的な検証における現状

階層的な検証の全体像を図3.3に示す。システムレベルシミュレーションの検証ステージと、ハー

ドウェアとソフトウェアによる協調シミュレーションの検証ステージと、実機に近い環境でのハードウェアとソフトウェアによる協調エミュレーションの検証ステージの 3 つ検証ステージから構成される。協調シミュレーションの検証ステージは、ハードウェアとソフトウェアのモデル化の抽象度に応じて、さらに 4 つの検証ステージに分類される。

(1) システムレベルシミュレーションの検証ステージ

システム仕様を実現するためのキーとなるアルゴリズムやアーキテクチャを事前に評価・検証する階層である。対象のアプリケーションを C 言語などの高位言語でモデル化し、高速なシミュレーションにより、結果をビジブルに表示しながら、設計の上流でトレードオフ評価（デザインレビュー）ができることが重要である。C 言語や Matlab や Statemate などでシステムの動作を表現したシステムモデルが使われる。

(2) 協調シミュレーションの検証ステージ

システムレベルシミュレーションによる評価の結果、仕様がある程度決まった段階でハードウェアとソフトウェアの分担を決める。高性能なパソコンやサーバ上でハードウェアとソフトウェアをモデル化して、ハードウェアとソフトウェアの協調シミュレーションにより、機能の検証とシステム的な性能評価を行う。

この階層では、モデルの抽象度に対応して、さらに 4 つの検証ステージに分けられる。

①高位言語モデル

システムを構成する機能モジュールをターゲットとなるプロセッサに依存しない C 言語などの高位言語で表現した高位言語モデルを使って協調シミュレーションを行う。協調シミュレーションは、

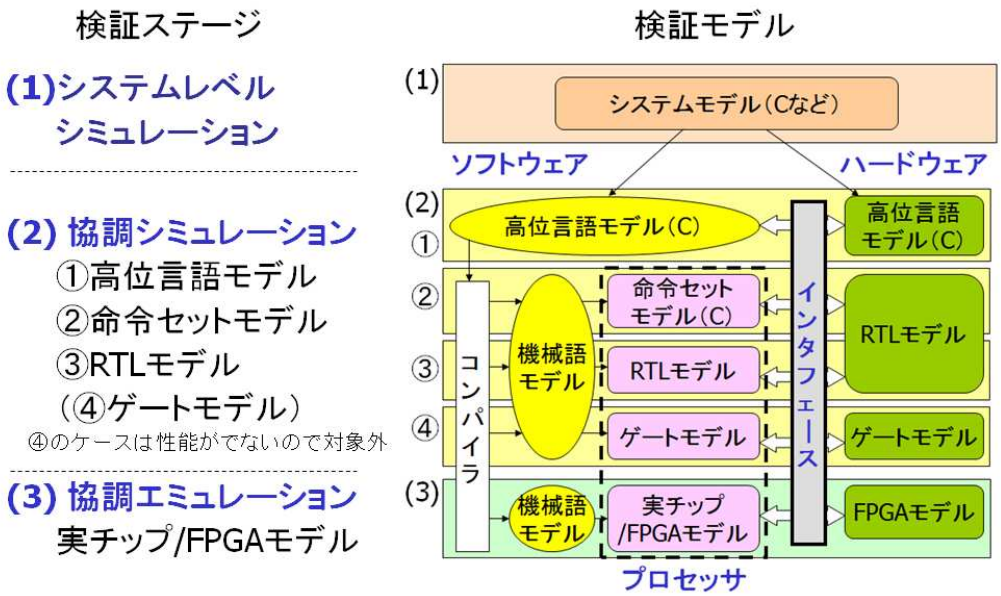


図 3.3　階層的な検証の全体像

ターゲットのプロセッサアーキテクチャが決定される前に開始することができる。ソフトウェアはシミュレーションを実行するホストマシンの高位言語モデルとして実行される。ハードウェアの高位言語モデルと組み合わせて、高速に協調シミュレーションを実行することが可能である。 主に、ハードウェアとソフトウェアで分担した機能の検証を行う。

②命令セットモデル

ターゲットとなるプロセッサの命令動作を C 言語などで記述した命令セットモデルを用いて協調シミュレーションを行う。ソフトウェアとして実行される高位言語モデルはターゲットプロセッサの機械語に変換され機械語モデルが生成される。この機械語モデルがターゲットプロセッサの命令セットシミュレータによって解釈され、実行される。残りのハードウェアは、RT レベルで表現された RTL モデルが使われる。命令セットシミュレータが、キャッシュやパイプライン動作を正確に模擬できる場合には、性能の評価を行うことができる。

③RTL モデル

ターゲットプロセッサと残りのハードウェアを RT レベルで表現した RTL モデルを用いて、クロックに同期してサイクルベースで協調シミュレーションを行う。この協調シミュレーションでは、クロックの精度で正確なタイミングの評価が可能であるが、シミュレーションの実行により多くの時間が必要である。なお、プロセッサの RTL モデルを取得することができないときは、この段階で協調シミュレーションを実行することはできない。

④ゲートモデル

プロセッサと残りのハードウェアをゲートレベルで表現したゲートモデルを用いて、タイミング精度の協調シミュレーションを行う。シミュレーションは、正確なタイミングで行われるが、実行時間が非常に長くなるので、この段階での協調シミュレーションは一般に実施されない。

(3) 協調エミュレーションの検証ステージ

ハードウェアとソフトウェアの協調シミュレーションの速度は実機に比べて 10^{-6} から 10^{-4} 程度になる。マルチメディア系や通信系のような人間の感性による評価が必要なものや応用ソフトウェアを含めた組込みシステムの実動作に近い検証・評価等については、実機評価機（ブレッドボード）を作成して行うのが一般的である。特にシステム LSI が出来る前に組込みソフトウェアを動作させて検証するときには必須である。また、顧客獲得のためには、実際に動作する試作モデルを見せることがますます重要になってきている。

従来の実機評価機の作成はその都度基板作成を伴うことから基板設計の開発コストや開発期間が必要になる。また、変更への追随が大変で、複数の実機評価機が必要な場合に同じ状態を保つことも難しくなることから、FPGA などのプログラマブル素子を活用したエミュレーション技術が注目されている。

エミュレーションでは、サーバやパソコン上の設計データをネットワークからこれらのプログラマブル素子にマッピングすることにより、容易に実機に近い形で動作させることが可能となる。適用にあたっては、現状の FPGA などのプログラマブル素子の性能上

表 3.1　各検証ステージでの評価内容

検証ステージ		評価内容
(1)システムレベルシミュレーション	システムモデル	アルゴリズムの評価（C言語） （本研究ではC言語を採用）
(2)協調シミュレーション	①高位言語モデル	ハードウェアとソフトウェアの機能の評価
	②命令セットモデル	ハードウェアとソフトウェアの性能の評価
	③RTLモデル	主にハードウェアのタイミング評価（実行時間がかなりかかる）
今回適用外	④ゲートモデル	正確なタイミング評価→実行時間がかかるので、静的タイミング検証を適用
(3)協調エミュレーション	実チップ/FPGAモデル	実機に近いシステムレベルの評価 人間の感性による評価、プロトタイプの提示

1MHz から 20MHz の速度でしか動作できないことを前提に、使い方を工夫する必要がある。

ターゲットプロセッサについては、その論理合成可能な RTL モデルが存在すればプログラマブル素子にマッピングした FPGA モデルが使われる。存在しない場合には実チップを使った実チップモデルが使われる。残りのハードウェアは論理合成可能な RTL モデルをプログラマブル素子にマッピングした FPGA モデルが使われる。実際のシステムに近い環境で高速な検証が可能である。

各検証ステージでの評価内容をまとめると、表 3.1 のようになる。

3.3.2　階層的な検証における課題

競争の激しいマルチメディアや制御機器で、独創的な製品を開発する場合には、プロトタイプを何回もつくりながら、設計の上流から下流まで、繰り返し評価・検証を行なう必要があるが、現状では以下の課題が存在する。

①従来の階層的な検証では、各階層の検証ステージ毎に検証モデルの表現方法が異なることから検証環境も異なり、また、検証に必要な入出力信号を扱うテストベンチもばらばらで、上流から下流まで繰り返し検証を行なうことが容易ではなかった。

②従来の階層的な検証では、各階層におけるハードウェアとソフトウェアのインタフェースが異なる場合があり、階層をまたがって検証する度にインターフェース部のモジュールの変換が必要になっていた。

③協調シミュレーションや協調エミュレーションの検証ステージでは、プロセッサを実行するための検証モデルの入手性が大きな問題である。今回は入手できるプロセッサを対象にすることで対応することにした。

このような課題に対する解決策として、次の 3.4 節でトップダウン協調検証手法を提案する。

3.4.　トップダウン協調検証手法の提案

競争の激しいマルチメディアや制御機器で、独創的な製品を開発する場合における階層的な検証の課題を解決するために、設計の上流における検証ステージから下流の実機に近い環境による検証ステージまで効率良く繰り返し検証できるトップダウン協調検証手法を提案する（図 3.4 参照）。

(a)検証環境の統合と共通テストベンチの適用

システムレベルシミュレーションと 3 階層の協調シミュレーションの検証環境を、高位言語（C）モデルと RTL モデルを混在して検証できる協調シミュレーション環境に統合し、システムレベルシミュレーションの検証ステージから、実機に近い環境での協調エミュレーションの検証ステージまで共通テストベンチを適用することにより、効率よい繰り返し検証を実現する。

(b)コンポーネント論理バスアーキテクチャによるインタフェースの統一

ハードウェアとソフトウェアのインタフェースを、コンポーネント論理バスアーキテクチャに統一することで、インターフェース部におけるモジュールの独立性を高め、階層をまたがる検証の効率化を実現する。

3.4.1　検証環境の統合と共通テストベンチの適用

各検証ステージでは、検証の対象となるソフトウェア部分とハードウェア部分をそれぞれコンピュータで検証できるようにモデル化する必要がある。3.3.1 節で説明したように、システムレベルシミュレーションと協調シミュレーションの階層におけるモデル化では C 言語モデルと RTL モデルが使われるが、従来の検証ではそれぞれ異なった環境で提供されてきた。今回シミュレーション環境として、C 言語モデルと RTL モデルの両方を扱える

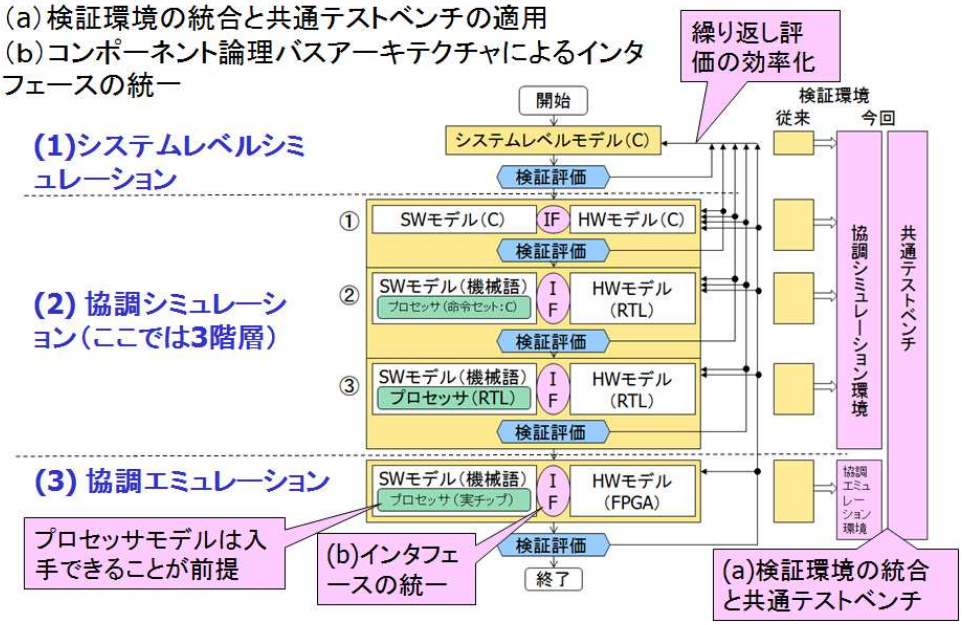


図 3.4　トップダウン協調検証手法の提案

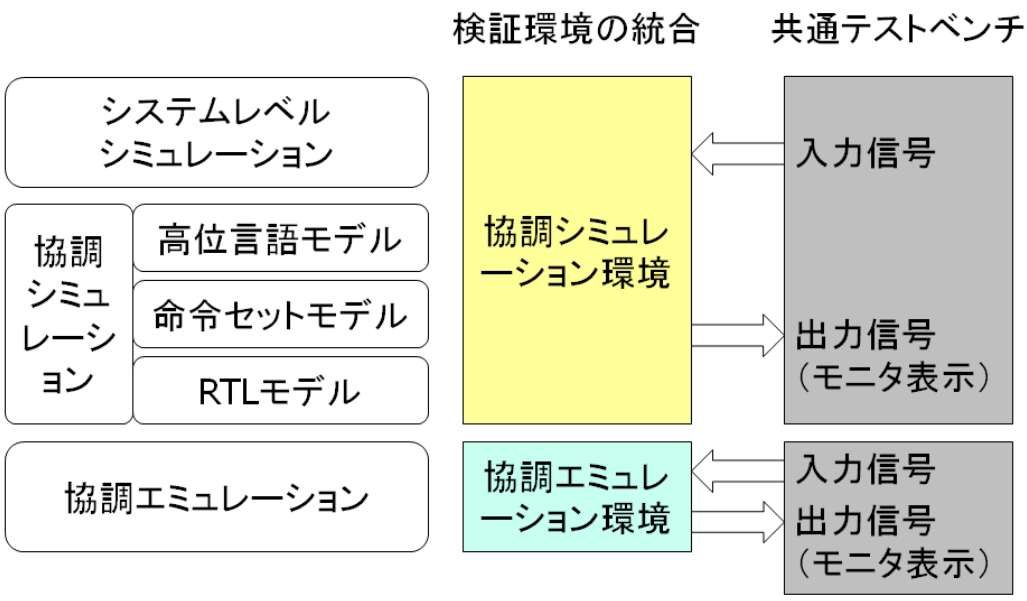


図 3.5　検証環境の統合と共通仕様のテストベンチの適用

協調シミュレーション環境に統合する。また、検証のための外部入力信号の生成や外部出力信号の表示などを扱う共通テストベンチを適用する（図 3.5 参照）。このように、シミュレーション環境の統合や共通仕様のテストベンチの適用により、検証につかうテストデータ（入出力信号データ）の共通化などが可能になり、システムレベルシミュレーションから、実機に近い協調エミュレーションまで、何回も繰り返し検証する作業の効率化を実現することができる。

3.4.2　コンポーネント論理バスアーキテクチャによるインタフェースの統一

ハードウェアとソフトウェアのインタフェースとして、従来は、検証シミュレーションの命令セットモデルの検証ステージ以降にハードウェアの実装と同じメモリマップド IO 方式を適用し、それ以前の高位言語モデルの検証ステージでは分割に応じて自由なインタフェースを定義する場合が多かった。そのため、高位言語モデルの検証ステージから下位の命令セットモデルの検証ステージに移行するときに、インタフェースを変更しそれに対応したハードウェアとソフトウェアのモジュールの変更が必要であった。

今回の提案では、高位言語モデルの検証ステージでも、インタフェースとしてメモリマップド IO 方式を適用することにより、協調検証シミュレーションの検証ステージから、協調エミュレーションの検証ステージまで、一貫したインタフェースによりインタフェース部分のモジュール化を図り、効率の良い繰り返し検証ができることを狙っている。

今回提案したメモリマップド IO 方式に統一したインタフェースについてコンポーネント論理バスアーキテクチャと定義した（図 3.6）。コンポーネント論理バスでは、システムバス上のメモリアドレス空間に割り付けられた Control レジスタ、Data レジスタ、および Status レジスタを介して、プロ

セッサ上で実行されるソフトウェアとワイヤードロジックで実行されるハードウェアの間で情報のやり取りを行う。

システムバスは、アドレス信号、データ信号、およびコントロール信号から構成される。アドレス信号で、Control レジスタ、Data レジスタ、および、Status レジスタを選択し、コントロール信号で、ソフトウェアからハードウェア側に書き込むのか、ハードウェア側からソフトウェア側に読み込むのかを指定する。データ信号を使って書き込むデータや読み込むデータを送受する。

ソフトウェアとハードウェアの間のデータ交換は、次のように行われる。ソフトウェアが、Data レジスタにデータを書き込んだあと、Control レジスタに起動の指示を書き込むことにより、ハードウェアを起動し、データを転送する。

ハードウェアにおける処理が完了すると、処理データを Data レジスタに書き込んだあと、処理の結果の状態を Status レジスタに書き込む。ソフトウェアにハードウェアの処理の終了を通知する方法には、割り込み方式とポーリング方式があるが、今回の研究ではポーリング方式を採用している。なお、Control レジスタ、Data レジスタ、Status レジスタの構成やビット数などは、ハードウェア側で実現する処理内容に依存する。

今回の提案は、このコンポーネント論理バスアーキテクチャを実装レベルだけではなく、設計の上流の協調シミュレーションの階層にも適用することで、トップダウンで検証を繰り返し効率よく行うことを狙ったものである。すなわち、協調シミュレーションから協調エミュレーションまで、異なる検証ステージでのモデルの置換やハードウェアとソフトウェア間のモデルの組み合わせにおいて、わずかな変更で移行できることが可能になることが期待できる。なお、協調エミュレーションでは、コンポーネント論理バスアーキテクチャをそのまま実装することで実現できるが、協調シミュレーションの階層においては、模擬的に実現することが必要になる。協調シミュレーションにおけるコンポーネント論理バスのインタフェースの実現方式として、図 3.7 に示すように共有メモリ方式、信号イベ

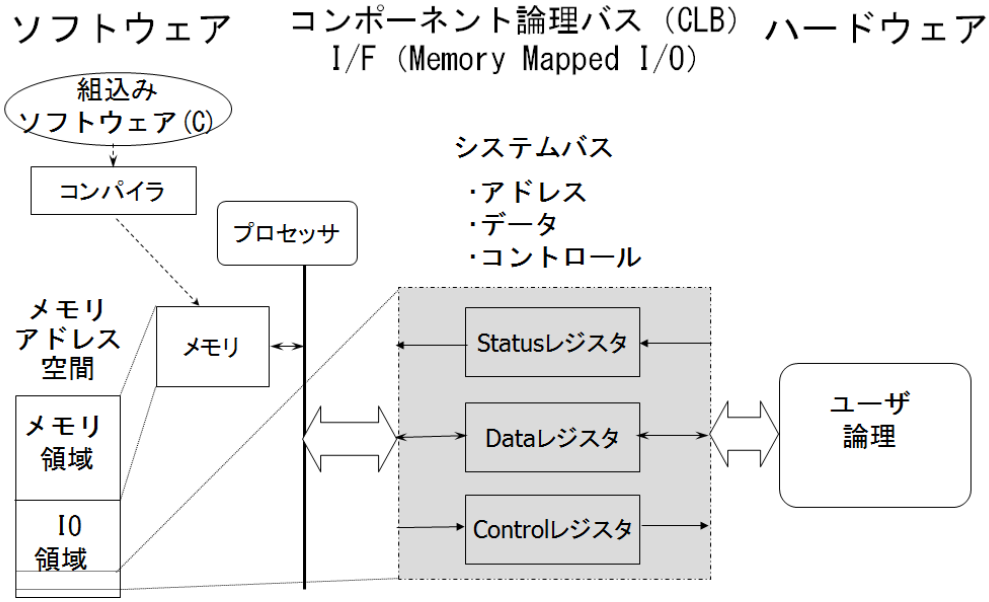


図 3.6　コンポーネント論理バス（CLB）アーキテクチャの構成

ント方式およびプロセス間通信方式の 3 つの方式を考える。

(1)　共有メモリ方式

高位言語レベルの検証ステージでは、図 3.7(1)に示すように共通メモリを用いて実現する。I / O 領域は、メモリアドレス空間の一部として割り当てられ、ソフトウェアとハードウェアがこの空間をアクセスすることでデータが交換される。

ソフトウェアは、共通メモリの IO 領域に割り当てられた Data レジスタにデータを書き込む。一連の書き込み操作が完了すると、起動イベントを Control レジスタに書き込み、ハードウェアに制御を渡す。ハードウェアは制御が渡されると、共通メモリ空間に割り当てられたハードウェアのレジスタから、必要なデータを読み込み、コントロールレジスタで指定された処理を行う。処理が終了すると、処理結果や状態を IO 領域に割り当てられた Data レジスタと Status レジスタにそれぞれ書き込む。ソフトウェアは、ハードウェアから制御が戻ってくると、共通メモリ空間に割り当てられている Data レジスタや Status レジスタを読み取り、ハードウェアの処理結果や状態を取得する。

(2) シングルプロセスでの信号イベント方式

プロセッサのモデル（命令セットモデル/RTL モデル）とハードウェアの RTL モデルを 1 つのプログラム（シングルプロセス）でシミュレーションできる場合には、図 3.7(2)に示すように 1 つのプログラム（シングルプロセス）での信号イベント方式で実現する。

ソフトウェアは機械語の書き込み（Store）命令により、必要なデータをイベントメモリの IO 領域に割り当てられた Data レジスタに書き込む。すべての必要なデータを書き込んだ後に、ハードウェアを起動する信号（信号イベント）を Control レジスタに書き込む。ハードウェアは、起動信号の信号イベントを受け取った後、IO 領域の Data レジスタの内容を読み込み、Control レジスタで指定さ

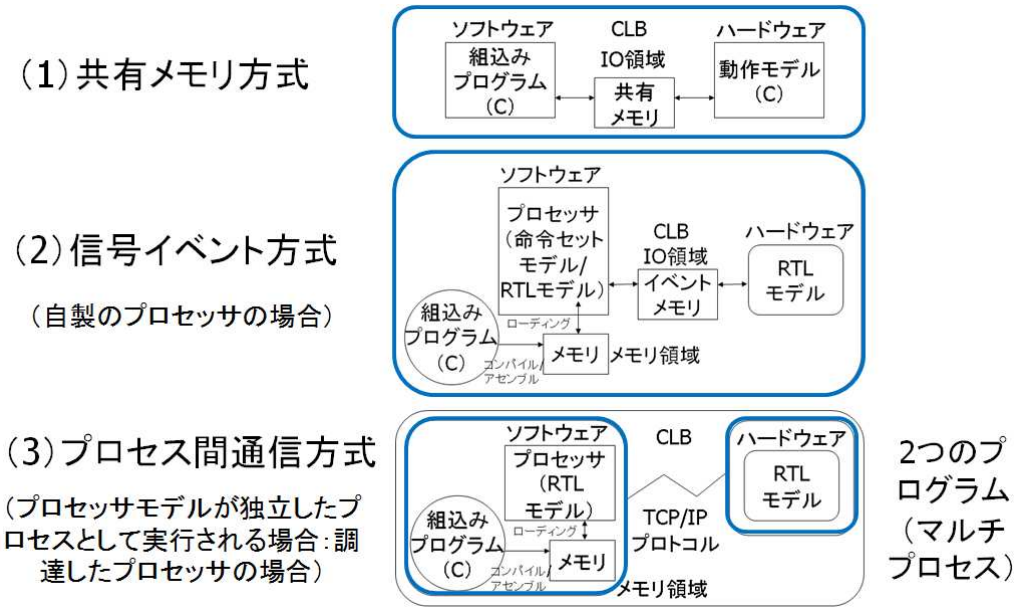


図 3.7　協調シミュレーションにおけるインタフェースの実現方式

れた処理を行う。ハードウェアは、処理を終了すると、処理結果や状態を IO 領域に割り当てられた Data レジスタと Status レジスタにそれぞれ書き込む。、Status レジスタには、終了の信号イベントが含まれている。ソフトウェアは Status レジスタの終了フラグにより、ハードウェアの処理が終わったことを検知する。

(3) マルチプロセスでのプロセス間通信方式

プロセッサのモデル（命令セットモデル/RTL モデル）とハードウェアの RTL モデルがそれぞれ異なるプログラム（マルチプロセス）でシミュレーションされる場合には、図 3.7(3)に示すようにプロセス間通信方式で実現する。プロセッサのモデルが他部門で開発されており、ハードウェアのモデルのシミュレータと同じプログラム（シングルプロセス）で実行できない場合に適用される方式である。

プロセッサのモデルとハードウェアの RTL モデルの間の IO 領域を通したデータ通信は、ソケットインタフェースやリモートプロシージャコールなどを用いて実現される。

3.5 マルチメディア系組込みシステムによる適用評価

本節では、提案したトップダウン協調検証手法を評価するために、2 件のマルチメディア系組込みシステムを対象に適用した。1 件目では、比較的簡単な線描画システムを対象に、ハードウェアとソフトウェアによる協調シミュレーションの検証ステージに適用して評価を行った。2 件目では、より複雑な映像圧縮処理システムを対象に、システムレベルシミュレーションから、ハードウェアとソフトウェアによる協調シミュレーション、さらに、設計の下流の実機に近い形で実動作と人間の感性による評価が可能になる協調エミュレーションの検証ステージまで適用して評価を行った。

最初に、対象となる組込みシステムを設計・検証するために使用した協調設計・検証ツールとプロセッサのモデリングについて説明する。続いて、線描画システムによる適用評価について述べ、最後に画像圧縮伸張システムによる適用評価について述べる。

3.5.1 協調設計・検証ツール

本提案を実現するのにふさわしいツールとして、ハードウェアとソフトウェアの協調設計・協調シミュレーションでは米国 HP 社で開発された Tsutsuji を、ハードウェアとソフトウェアの協調エミュレーションでは米国 Aptix 社のシステム MP4 を採用した。

(1) Tsutsuji の概要

Tsutsuji は、米国 HP（ヒューレットパッカード）社の研究所で 1993 年に開発され、その後 1996 年に図研から Vps（Virtual Prototyping System）として販売された[Cul93][ZUK96] [田中 11]。

Tsutsuji は、高位言語（C）や RTL など多様な設計記述形式をサポートし、高速な論理合成機能とビジブルな仮想計測器環境を持った高速なシミュレーション機能を備えており、

協調シミュレーション環境を構築するのに最適であった。

Tsutsuji の実行画面の例を図 3.8 に示す。モジュール図は回路図表記(信号線はベクタ表記）で階層的に表現できる。ブロックの中は HW 記述言語（LDF）でも記述が可能である。また、ステップ実行の画面では、1 ステップ毎にレジスタの値がどのように変化していくかを観察することができる。さらに、仮想計測器環境では、仮想的なつまみで値を設定したり、仮想的なディスプレイに波形を表示したりすることができる。

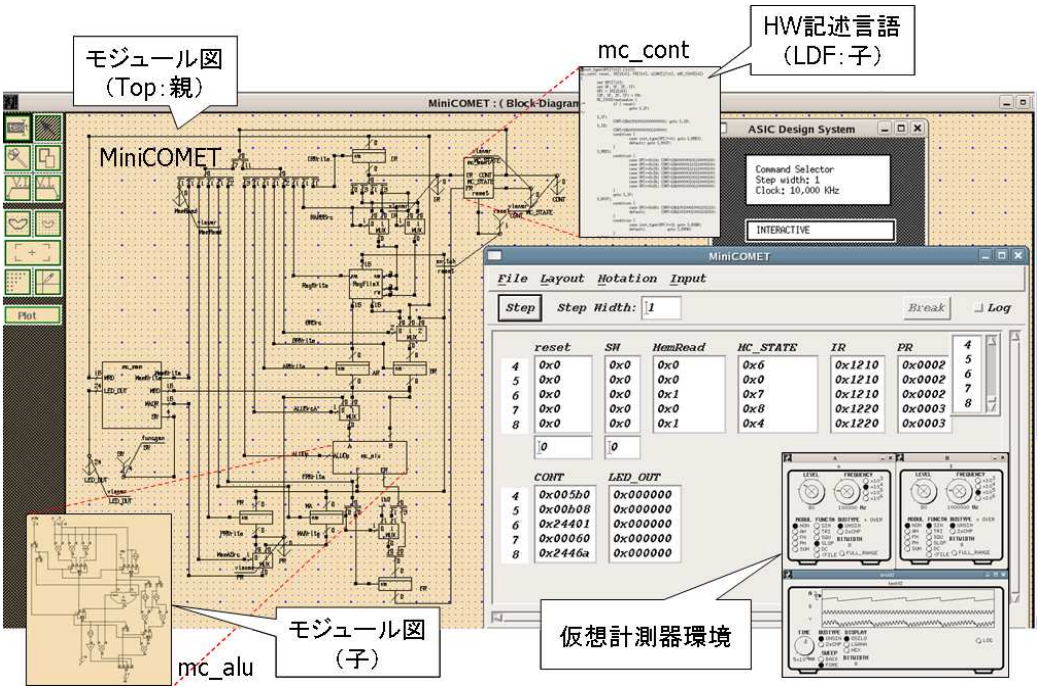


図 3.8　Tsutsuji の実行画面例

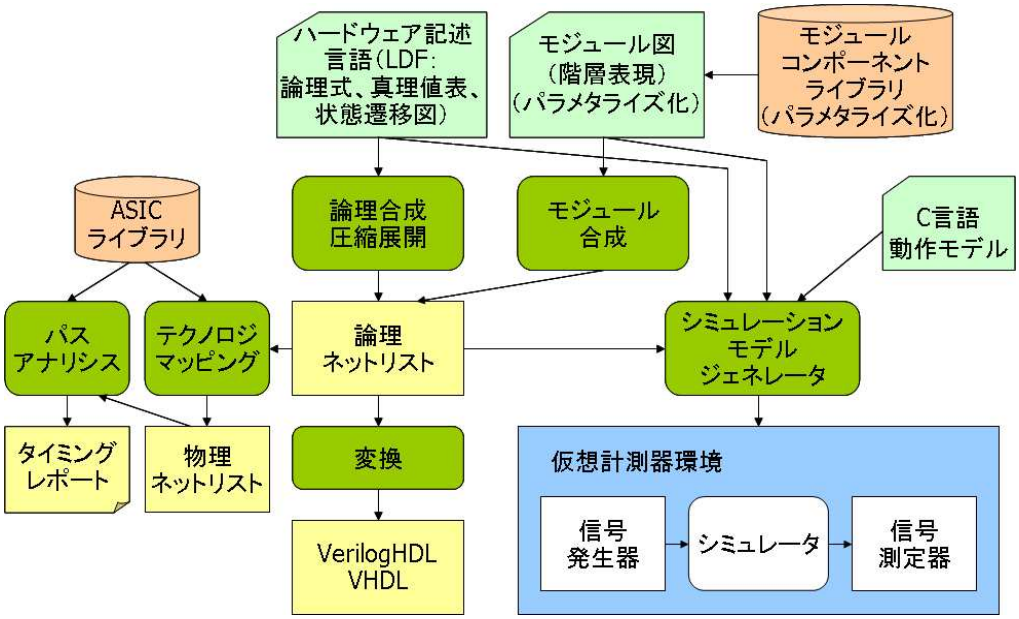


図 3.9　Tsutsuji の開発環境

Tsutsuji の開発環境を図 3.9 に示す。設計対象を階層記述できるモジュール図とハードウェア記述言語（LDF）で記述し、C 言語動作モデルと組み合わせて、シミュレーションモデルを生成し、仮想計測器環境のもとで高速なシミュレーションを行うことができる。また、論理合成・圧縮展開機能やモジュール合成によってゲートレベルの回路を生成し、ネットリスト変換により既存の Verilog や VHDL に接続することができる。さらに、ASIC ライブラリを参照し、パス解析によりタイミングレポートを作成することができる。

(2)　協調エミュレーションツール：システムエミュレータ（MP4）の概要

米国 Aptix 社（図研が販売）のシステムエミュレータの構成を図 3.10 に、プロセッサ M32R/D の実チップを搭載したアダプタボードの例を図 3.11 に、開発環境を図 3.12 に示す。1990 年代後半において協調エミュレーション環境を構築するのに最適なシステムであった。

図 3.10 のシステムエミュレータ MP4 には、論理情報をプログラム可能な FPGA（Field Programmble Gate Array）と接続機能をプログラム可能な FPIC（Field Programmable Interconnect Component）が搭載されている。図 3.11 に示すようなプロセッサ等の実チップやインタフェース回路を搭載したアダプタを開発し、MP4 に搭載することができる。図 3.12 の HDL 記述された設計データとピン割り付けデータから、FPGA と FPIC の書込みデータを作成し、ダウンロードすることで、システムエミュレーションを実現する。

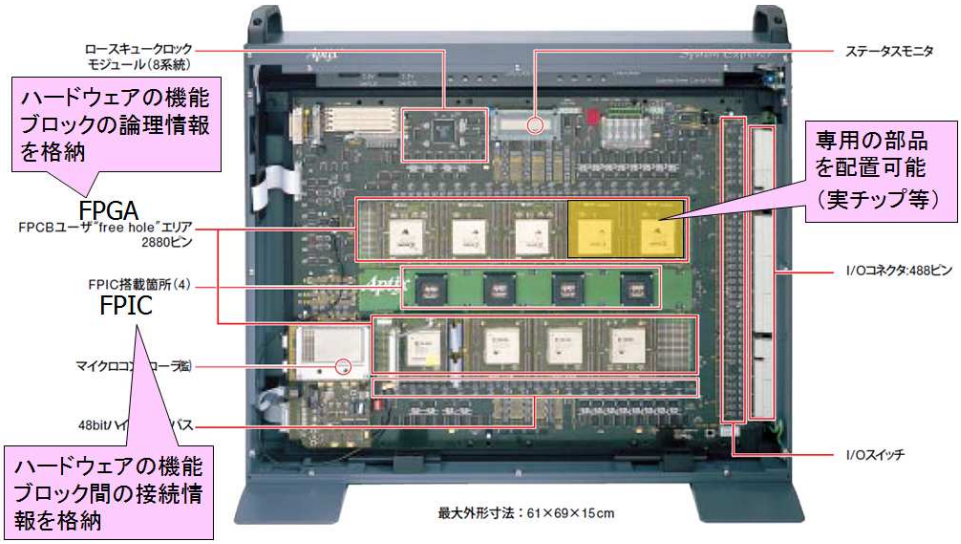


図 3.10　Aptix 社システムエミュレータ MP4 の構成

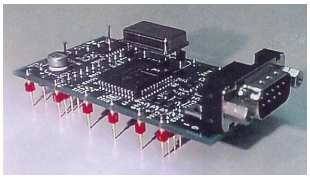


図 3.11　システムエミュレータのアダプタ（自製）の例

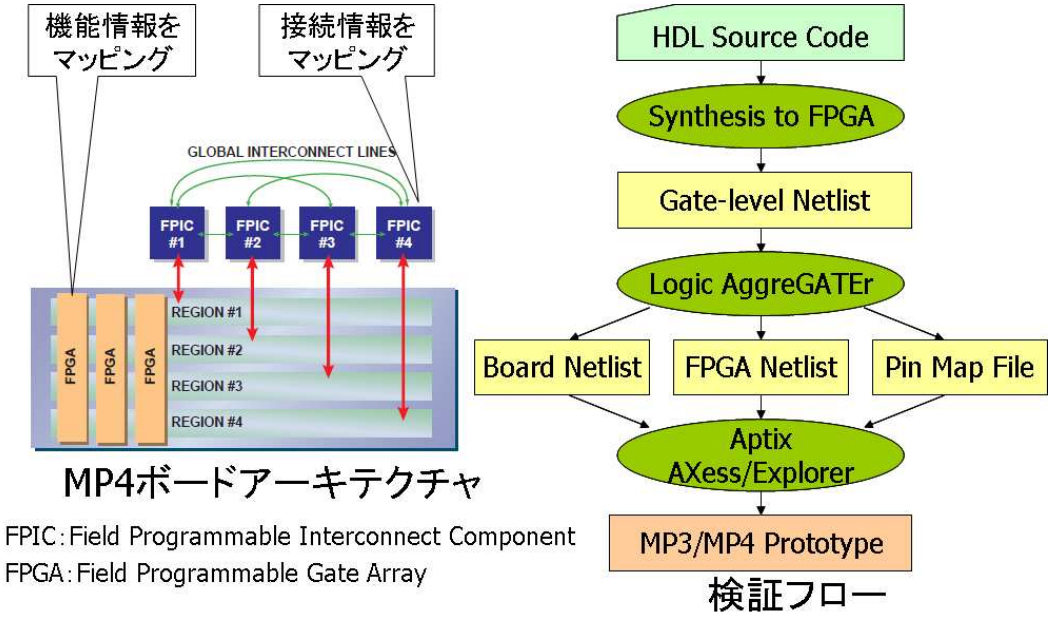


図 3.12　システムエミュレータの開発環境

3.5.2　プロセッサのモデリング

協調検証シミュレーションの命令セットモデルの検証ステージ以降では、使用するプロセッサの検証モデルが必要になる。検証モデルは、検証ステージに適切なものを使用する必要があり、如何にしてそれらを入手するかが課題である。今回の提案では、トップダウン協調検証手法の各検証ステージにおいて、プロセッサの検証モデルの位置づけを明確にして、検証モデルの入手性を考慮して協調シミュレーションと協調エミュレーションに取り組んだ。

(1) 協調シミュレーションの検証ステージ

協調シミュレーションの段階におけるプロセッサのモデリングについて表 3.2 に示す。

高位言語モデルでは、プロセッサは意識せず、ハードウェアとソフトウェアの C 言語モデル同士を協調シミュレーション環境で実行することになる。高速な実行が期待できる。

命令セットモデルでは、プロセッサの命令セットを C 言語モデルで表現する。ASAP のような自社製のプロセッサであれば、C 言語による命令セットモデルの入手性は問題ない。調達プロセッサを利用する場合には調達先からの C 言語による命令セットモデルの入手性に依存する。本研究では、同じ企業であったことから入手することができた。三菱電機製の 16 ビットマイコン M16C を使ったプロセッサの命令セットシミュレータの例が報告されている[Gho95]。

RTL モデルでは、自製のプロセッサの場合は問題ないが、調達の場合は同一企業内でも入手が困難な場合が多い。

表 3.2　協調シミュレーションにおけるプロセッサのモデリング

検証ステージ		プロセッサのモデリング	モデルの入手性	
			自製	調達
(2)協調シミュレーション	①高位言語モデル	プロセッサは意識しない。ホスト上でCモデル同士で検証	プロセッサは意識しない	
	②命令セットモデル	命令セットをCモデルで表現。なお、命令セットのクロック数を評価可能	可	比較的容易
	③RTLモデル	RTLレベルのHDLで表現。クロックレベルの正確なタイミングを評価可能	可	困難
			例えば ASAP	例えば M16C、M32R/D

ASAP：Application Specific Adaptable Processor
M16C：三菱電機製16ビットマイコン
M32R/D：三菱電機製DRAM内蔵32ビットプロセッサ（命令セットシミュレータは別プロセスとして提供）

(2)　協調エミュレーションの検証ステージ

協調エミュレーション環境は以下のようにして構築される。

・プロセッサのモデルは、HDL 記述されたものがある場合は、HDL 記述のモデルを論理合成し FPGA にマッピングしたものが使われる。HDL 記述したものがない場合には実チップが使われる。

・アプリケーションを記述したＣプログラムは、ターゲットとなるプロセッサの機械語にコンパイルされ、バイナリーデータが書き換え可能メモリに格納される。

・ハードウェアのモデルは HDL 記述したものを論理合成し FPGA にマッピングされる。

・部品間の接続情報は、プログラマブルなスイッチアレイ素子にマッピングされる。

・アナログ映像信号を A/D 変換によりデジタル信号に変換する入力信号アダプタや、デジタル信号を D/A 変換により外部のディスプレイに表示する出力信号アダプタにより、リアルな共通テストベンチを提供する。

協調エミュレーションにおけるプロセッサのモデリングについて表 3.3 に示す。

実チップモデルでは、プロセッサの本物のチップをそのまま利用する。チップをエミュ

表 3.3　協調エミュレーションにおけるプロセッサのモデリング

検証ステージ		プロセッサのモデリング	モデルの入手性	
			自製	調達
(3)協調エミュレーション	実チップモデル/FPGAモデル	実際のチップを使ったモデル	製造後可	可
		HDLで表現されたモデルから論理合成して作成したFPGAモデル	可	困難
			例えば ASAP	例えば M16C/M32R

レータに搭載するアダプタの開発が必要である。

FPGA モデルでは、HDL 記述されたプロセッサのモデルを論理合成して FPGA にマッピングすることで実現できる。自製のプロセッサの場合は問題ないが、調達したプロセッサの場合は一般に HDL モデルの入手が困難なので実現は難しい。

3.5.3　線描画システムによる評価

1 つ目の事例では自製したプロセッサを使った線描画システムを対象に、協調シミュレーションにおけるトップダウン協調検証手法についての評価を行った。協調シミュレーションにおける 3 つの検証ステージ（高位言語モデル、命令セットモデル、RTL モデル）での実現方法と効率よく検証できるかの評価が中心である[Seo97][YaM98]。

(1)　線描画システムの構成

線描画システムの構成を図 3.13 に示す。ソフトウェア部は、プロセッサとメモリで構成されている。メモリにロードされた線描画制御プログラムが、プロセッサ上で実行され一連の線の描画が行われる。ハードウェア部は、与えられた 2 点間の直線を高速に描画する線描画エンジンとライン表示器で構成されている。

この評価で使用するプロセッサとして、トップダウン協調設計システムを構築するときに開発した自製の 16 ビットの ASAP（Application Specific Adaptable Processor：アプリケーション志向適応型プロセッサ）を使用した[Sha96] 。ASAP は、アプリケーションプログラムで必要とする CPU のビット幅とレジスタファイルの数をパラメータとして入力し、開発したプロセッサ高位合成ツール[Sha96]によって生成されるプロセッサである 。今回の実験では、合成された命令セットモデルと RTL モデルを使用している。参考までに、16 ビット ASAP の規模は 5.3K ゲートで、線描画エンジン部のハードウェア規模は 1.5K ゲートである。

図 3.13 において、始点の XY 座標は始点レジスタ（X0、Y0）にセットされ、終点の XY 座標は終点レジスタ（X1、Y1）にセットされる。その後、スタートフラグである EN がセットされると、線

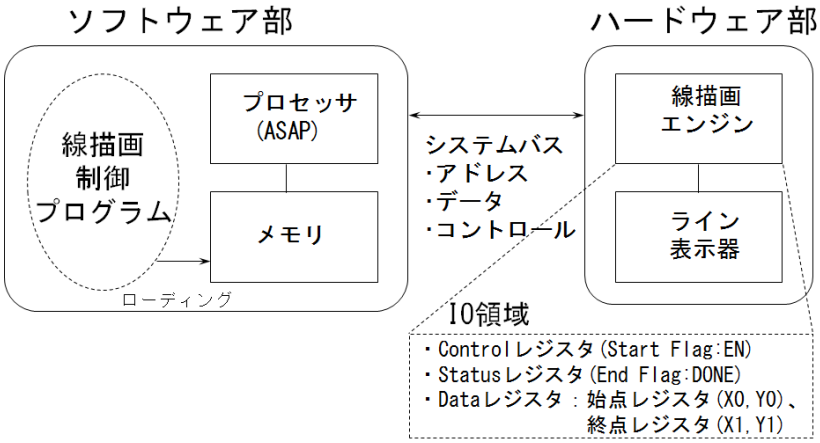


図 3.13　線描画システムの構成

描画エンジンが起動される。線描画エンジンが１本の線を描画し終えたとき、DONE フラグがセットされる。

(2) 適用結果と評価

提案したトップダウン協調検証フローに従い、プロセッサに依存しない高位言語モデル、プロセッサに依存する命令セットモデル、さらに RTL モデルを作成し、各々の評価を行なった。

プロセッサの命令セットモデル/RTL モデルと線描画エンジンの RTL モデルは同じシミュレーション環境で実行できることから、シングルプロセスでのシミュレーション環境を適用した。また、マルチプロセス環境でどの程度の性能が得られるかを評価するために、プロセッサの RTL モデルと線描画エンジンの RTL モデルとの間のコンポーネント論理バスインタフェースをプロセス間通信で実現する場合についても評価を行なった。

図 3.14 に線描画システムのコンポーネント論理バスの構成例を示す。

図 3.14 の組込みプログラム例に示すように、ソフトウェア部における高位言語モデルは、スタートアップイベント（ハードウェア部のモデル呼び出し）を取り除くといったわずかな修正で、命令セットモデルに置き換えることができた。

命令セットモデルから RTL モデルへの移行では、モデルを修正すること無しにハードウェア部の RTL モデルと結合ができた。今回の提案方式を使うことにより、検証ステージ間の移行を効率よく行うことができた。

協調シミュレーションの性能を、下記の 4 ケースで計測した。

① 高位言語モデル：(a)共有メモリ方式
② 命令セットモデル：(b)信号イベント方式
③ RTL モデル（シングルプロセス）：(b)信号イベント方式
④ RTL モデル（マルチプロセス）：(c)プロセス間通信方式

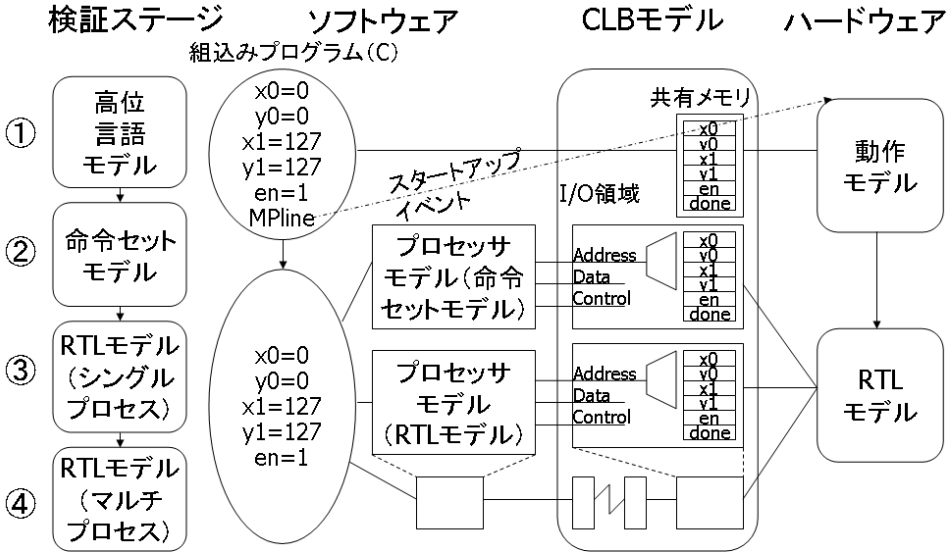


図 3.14　線描画システムのコンポーネント論理バス（CLB）モデルの構成例

協調シミュレーションの実行結果を表 3.4 に示す。この表では、RTL モデル（シングルプロセス）でのシミュレーション時間を 1 の単位として他の時間を正規化している。

高位言語モデルでの協調シミュレーションでは、ソフトウェアの C 言語モデルとハードウェアの C 言語の動作モデルを一緒にホストマシンで直接コンパイルし実行した。

命令セットモデルでは、C 言語で書かれた ASAP 命令セットシミュレータとハードウェア部の RTL モデルを高速にシミュレーションできる Tsutsuji サイクルベースシミュレータ[Cul93]を統合して、シングルプロセス環境で協調シミュレーションを行った。ASAP と Tsutsuji を使った命令セットモデルの協調シミュレーションの実行例を図 3.15 に示す。

表 3.4　線描画システムにおける協調シミュレーションの実行結果

<table>
<tr><th rowspan="2" colspan="2">検証
ステージ</th><th colspan="3">モデリング</th><th rowspan="2">協調検証
相対性能
（③ RTL
モデル=1）</th></tr>
<tr><th colspan="2">SW</th><th>インタフェース</th><th>HW</th></tr>
<tr><td colspan="2">①高位言語
モデル</td><td colspan="2">Cモデル</td><td>(a)共有メモリ方式</td><td>Cモデル</td><td>30</td></tr>
<tr><td colspan="2">②命令セット
モデル</td><td rowspan="3">微修正のCモデル
→ターゲットの機械語</td><td rowspan="2">命令セットモデル</td><td rowspan="2">(b)信号イベント方式</td><td rowspan="3">RTL
モデル</td><td>1.3</td></tr>
<tr><td>③</td><td rowspan="2">RTL
モデル</td><td>1</td></tr>
<tr><td>④</td><td>RTL
モデル</td><td>(c)プロセス間通信</td><td>1/370</td></tr>
</table>

同一プロセスであれば高位言語モデルに対して1/23程度で実行可能（性能評価可能）

プロセス間通信のオーバヘッドが大きい

AP: アプリケーションプログラム

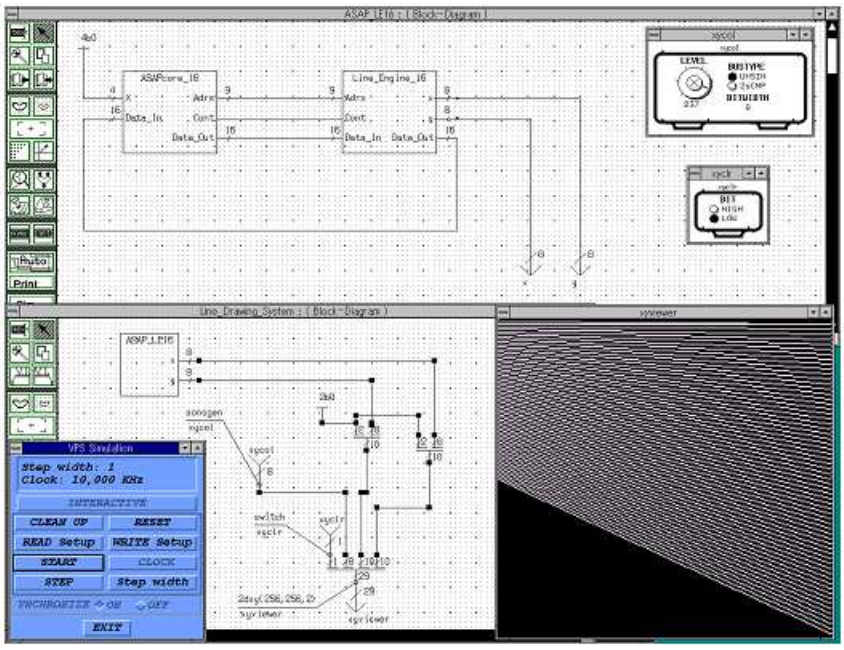


図 3.15　線描画システムの協調シミュレーション実行例

RTL モデル（シングルプロセス）では、ASAP の RTL モデルとハードウェアの RTL モデルを使って Tsutsuji のシングルプロセス環境でシミュレーションを行った。

RTL モデル（マルチプロセス）では、ASAP 命令セットシミュレータと Tsutsuji を使い、Tsutsuji は、TCP/IP に基づくプロセス間通信を使って ASAP 命令セットシミュレータと同期をとりながらマルチプロセス環境で実行した。

高位言語モデルでは、すべてのシミュレーションは C 言語を使ってホストマシン上で直接実行される。RTL モデル（シングルプロセス）の協調シミュレーションと比べ、30 倍高速に実行できた。

命令セットモデルと RTL モデル（シングルプロセス）での性能は、プロセッサの回路規模とハードウェア部の回路規模に依存する。この例では、命令セットモデルの協調シミュレーション速度は RTL モデルと比べ、1.3 倍高速になった。今回の評価においては、ハードウェアは同じ条件でかつプロセッサモデルも高速なサイクルベースシミュレータが使われているため、性能においてそれほど差異が出なかったと考えられる。

RTL モデル（シングルプロセス）と RTL モデル（マルチプロセス）の性能を比較すると、プロセス間通信のオーバーヘッドが大きく、自製のプロセッサを使う場合は、シングルプロセスで協調シミュレーションできる検証環境を構築することが重要である。

本事例では、協調シミュレーションの検証ステージにおいて、シングルプロセスの環境において高速なシミュレーションが可能であり、ハードウェアとソフトウェアのインタフェースとしてコンポーネント論理バスを適用することにより、上流の高位言語モデルから、命令セットモデル、さらに RTL モデルまで、効率よく繰り返し実行できることがわかった。

3.5.4 映像圧縮処理システムによる評価

2 つ目の事例として、マルチメディア系の典型的な例である映像圧縮処理システムを対象とした。この分野では、いろいろな変化に対応でき、かつ最新の規格を盛り込むため、従来すべて ASIC で対応してきたものを、プロセッサ上で実行される組込みソフトウェアと専用のカスタム回路の組み合わせにより実現する方法がいろいろ研究されている[Tho93] [Hol96]。コスト・性能・消費電力・柔軟性等の面からハードウェアとソフトウェアによるバランスのとれた組込みシステムを実現する必要があるが、規模が拡大し複雑になるほど、評価・検証・デバッグ環境として、ハードウェアとソフトウェアによるトップダウン協調検証が重要となる。

本事例では調達した 32 ビットプロセッサ（M32R/D）を使った映像圧縮処理システムを対象に、システムレベルシミュレーションから、協調シミュレーション、協調エミュレーションまで、提案したトップダウン協調検証手法を適用し評価を行なった。設計の上流から下流までの 4 つの検証ステージ(システムモデル、高位言語モデル、命令セットモデル、実チップモデル）において、実用レベルで効率よく繰り返し検証できるかの評価が中心である[Seo98][安田 99]。

(1)　映像圧縮処理システムの構成

映像圧縮処理システムの構成を図 3.16 に示す。主な機能ブロックは、システム全体を制御するプロセッサ(RISC)、映像データを圧縮処理する専用信号処理回路、入力デバイスからの映像信号を処理する入力処理回路、圧縮した映像データをビットストリームとして出力するビットストリーム出力処理回路、映像データを保持するフレームメモリ回路、プログラムを格納するプログラムメモリ回路（図には記載していない）等である。この中で、映像データを圧縮処理する専用信号処理回路については、製品目標に応じて、ハードウェアとソフトウェアによる組み合わせにより実現されることになる。また、圧縮伸張で新しいアルゴリズムを適用する場合には、アルゴリズムの妥当性を出来るだけ早い段階で実機に近い環境で人間の感性により確認する必要がある。

(2)　検証ステージ

図 3.17 に提案したトップダウン協調検証手法において今回適用した検証ステージを示す。上流から①アルゴリズム評価のためのシステムレベルシミュレーション、②機能評価のためのハードウェアとソフトウェアによる高位言語モデル協調シミュレーション、③性能評価のためのハードウェアとソフトウェアによる命令セットモデル協調シミュレーション、及び、④実機に近い環境での評価のためのハードウェアとソフトウェアによる実チップモデル協調エミュレーションの４つの検証ステージを適用した。なお、RTL モデルとゲートモデルは、性能が遅くなることから対象としなかった。また、ハードウェアとソフトウェアのインタフェースにコンポーネント論理バスアーキテクチャを適用すると共に、外部入出力を模擬した共通テストベンチを開発して、各検証ステージ間で一貫した効率の良い協調検証環境の実現を図った。

① **システムモデル**：図 3.17 の①に示すように、対象アプリケーションを実現するためのアルゴリズムやアーキテクチャを評価するために、システム全体を C 言語によりモデル化を行い、システムレベルシミュレーションにより評価を行う。

② **高位言語モデル**：図 3.17 の②に示すように、ハードウェアとソフトウェアのインタフ

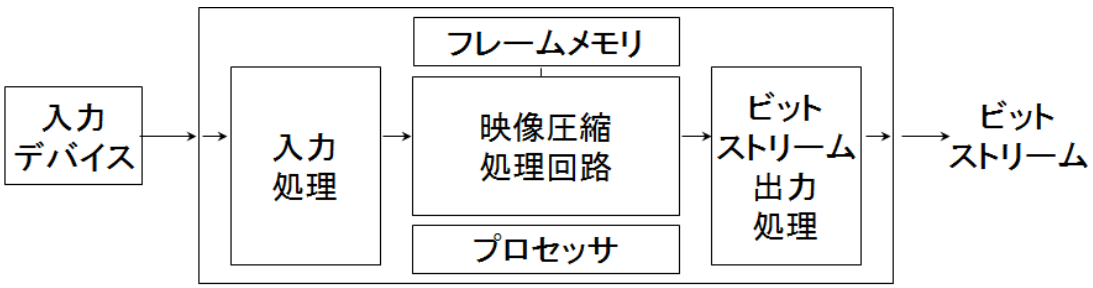


図 3.16　映像圧縮処理システムの構成例.

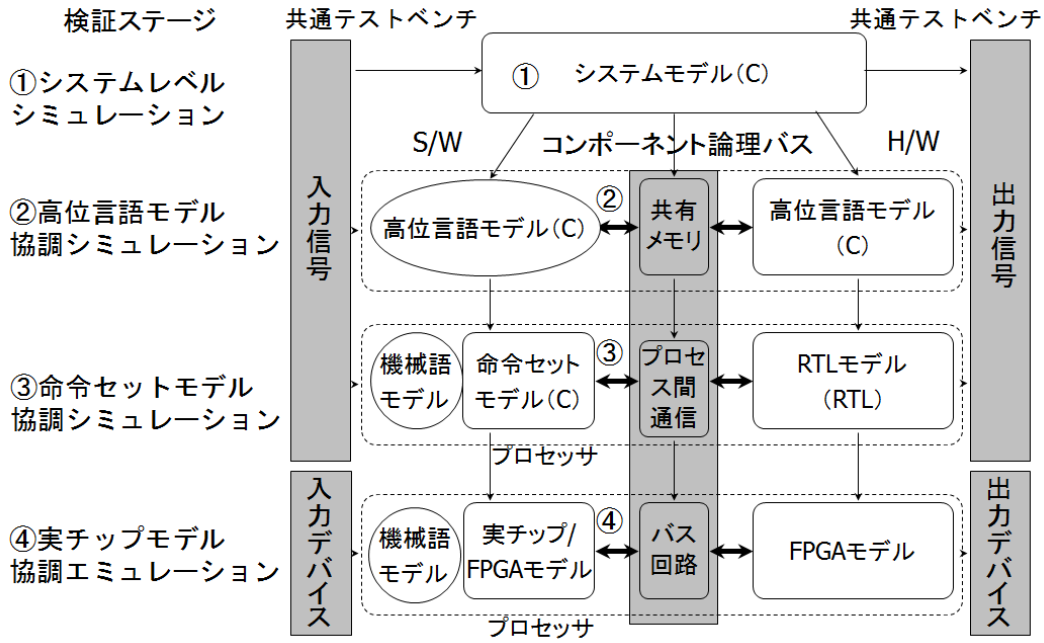


図 3.17　トップダウン協調検証システムにおける検証ステージ

ェースが確定し、高位言語（例えばC言語）で記述された組込みソフトウェアとハードウェア（C モデル）による協調シミュレーションにより、分担機能の検証・評価を行う。組込みソフトウェアは、ターゲットとなるプロセッサとは独立に、検証を行うホストマシン上で直接実行することで高速な処理が可能となる。

③　**命令セットモデル**：図 3.17 の③に示すように、高位言語で記述されたアプリケーションプログラムをターゲットとなるプロセッサの機械語にコンパイルし、プロセッサの命令セットモデルと、ハードウェアの RTL モデルによる協調シミュレーションにより、実行サイクルの算出による性能評価を含めてより詳細な検証を行う。プロセッサは社内調達のプロセッサ（M32R/D）を使い、キャッシュやパイプライン動作まで模擬できる命令セットシミュレータを使用する。

④　**実チップモデル**：図 3.17 の④に示すように、マルチメディアの世界では、人間を対象にした映像・音声処理が中心となることから、感性による評価が重要になる。このため、実時間に近い環境での評価が必要であり、システムエミュレータによる協調エミュレーションが重要となる。今回の協調エミュレーションでは、ターゲットのプロセッサ（M32R/D）の RTL モデルが入手できなかったため、実チップモデルを利用する。

このように４つの検証ステージをトップダウンに実行し、各検証ステージで発見された不具合を前の設計フェーズにフィードバックすることにより、早期にシステム全体の品質の評価が可能となる。

(3)　協調検証の実現方法

各検証ステージでの実現方法について述べる。

①　システムレベルシミュレーション

図 3.18 にシステムレベルシミュレーションの例を示す。

システム全体をＣ言語によりモデル化し、シミュレーションを行うホストマシンでコンパイルして実行することにより、高速なシミュレーション環境を実現できる。また、仮想計測器環境をサポートする共通テストベンチの構築により、実際の画像データを活用して検証できる環境を提供している。

②　高位言語モデルによる協調シミュレーション

図 3.19 に高位言語モデルによる協調シミュレーションの例を示す。

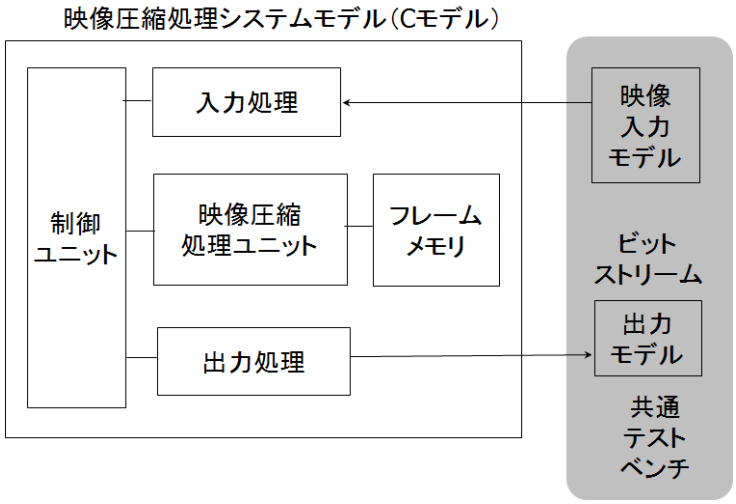


図 3.18　システムレベルシミュレーション.

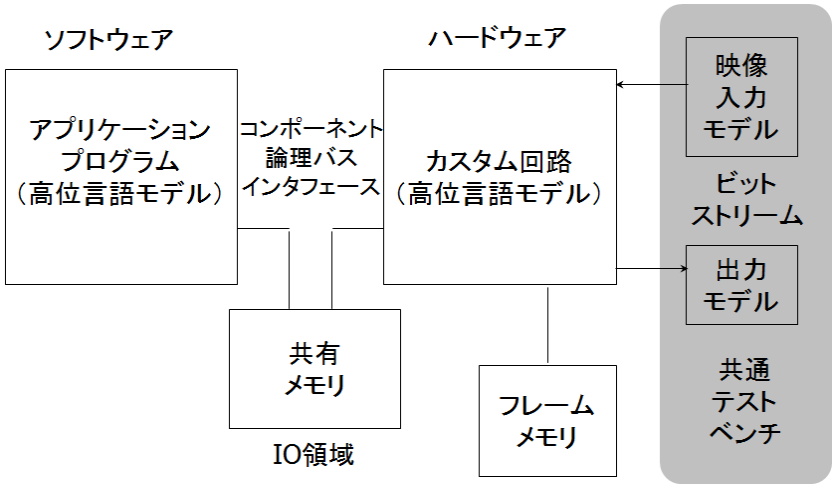


図 3.19　高位言語モデルによる協調シミュレーション

アプリケーションを記述したCプログラム（高位言語モデル）を、ホストマシンの機械語にコンパイルして直接実行することにより、プロセッサに依存せずに高速にシミュレーションができる。カスタム回路部も、C 言語で記述された高位言語モデルであり、同じようにコンパイルして、シングルプロセス上で高速にシミュレーションができる。コンポーネント論理バスアーキテクチャによるデータのやり取りは、カスタム回路部（専用映像圧縮回路）のレジスタやメモリを、メモリアドレス空間の IO 領域に割り付けて行う。実現方法としては、IO 領域を共有メモリに定義し、ここを双方でアクセスすることによりデータの送受を行う。システムモデルによるシミュレーションと同一の共通テストベンチを活用できる。

③　命令セットモデルによる協調シミュレーション

図 3.20 に命令セットモデルによる協調シミュレーションの例を示す。
対象となるアプリケーションを記述したCプログラムをターゲットとなるプロセッサの機械語にコンパイルして、そのプロセッサの命令セットシミュレータによりインタプリティブに実行する。カスタム回路部のハードウェアは RT レベルで表現し、サイクルベースでシミュレーションを行う。両方のシミュレータがシングルプロセス上で実行できないので、コンポーネント論理バスインタフェースをプロセス間通信の仕組みで実現し、データの送受を行う。プロセッサの命令セットモデルをキャシュやパイプライン動作まで模擬することにより、組込みソフトウェアの実行ステップを正確に把握でき、システムの性能を評価することが可能となる。システムモデルによるシミュレーションと同一の共通テストベンチが利用できる。

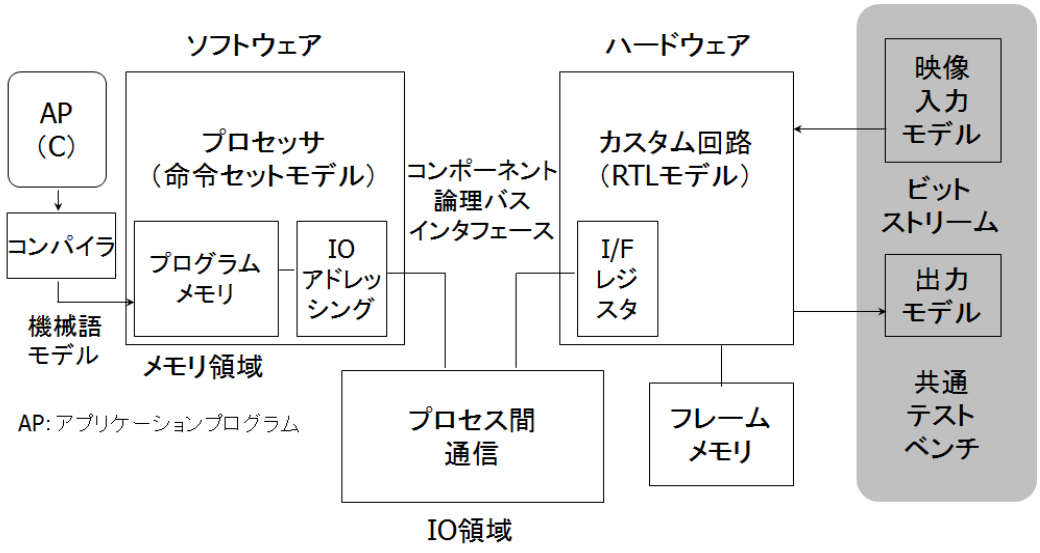


図 3.20　命令セットモデルによる協調シミュレーション

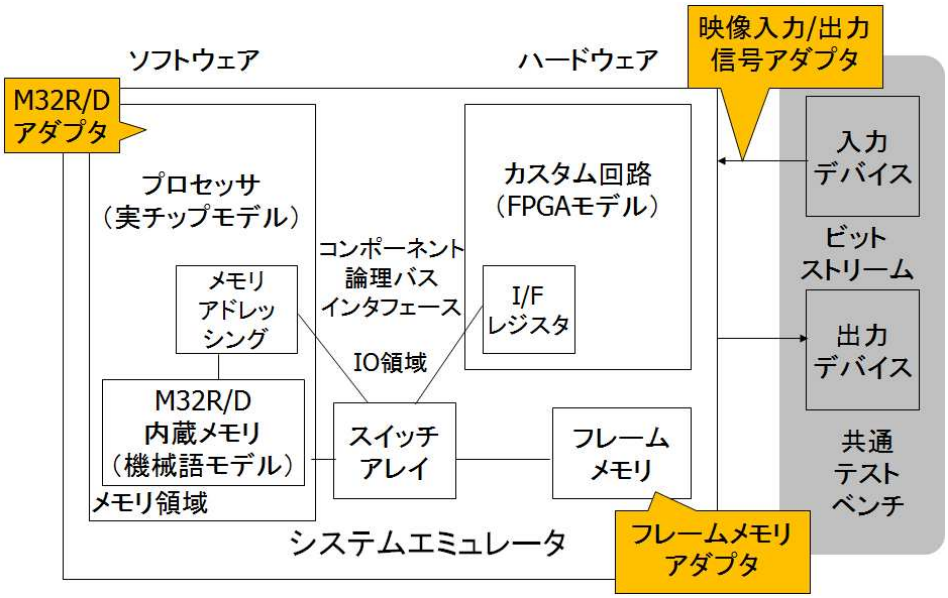


図 3.21　実チップモデルによる協調エミュレーション

④　実チップモデルによる協調エミュレーション

図 3.21 に実チップモデルによる協調エミュレーションの例を示す。

協調エミュレーション環境を以下のように構築する。

・プロセッサのモデルである実チップを搭載する M32R/D アダプタを作成する。

・アプリケーションを記述したＣプログラムはターゲットとなるプロセッサの機械語にコンパイルして、バイナリーデータを M32R/D 内蔵メモリに格納する。

・カスタム回路の RTL モデルを論理合成し、FPGA に分割してマッピングする。

・映像データを格納するフレームメモリアダプタを作成する

・部品間の接続情報は、プログラマブルスイッチアレイ素子にマッピングする。

・アナログ映像信号をデジタル信号に A/D 変換する映像入力信号アダプタや、デジタル信号を外部のディスプレイに表示するアナログ信号に D/A 変換する映像出力信号アダプタを作成し、リアルな共通テストベンチを実現する。

(4)　適用内容と評価手順

映像圧縮処理システムにトップダウン協調検証手法を適用し、その効果を確認した。図 3.22 に適用した映像圧縮処理システムのブロック図を示す。プロセサの制御のもとに、アナログ映像信号入力をデジタル信号に変換し、一旦フレームメモリに格納し、映像圧縮処理回路により動き予測を行い映像データの圧縮を行う。その後、ソフトウェアにより、符号化による圧縮を行い、仕様で規定されたビットストリームの信号列に変換する。

この事例では、プロセッサとして DRAM 内蔵 RISC プロセッサ(M32R/D)を社内調達した。命令セットシミュレータは提供されたが、RTL モデルは入手できなかった。映像圧縮処理専用回路としてプロセッサの指示で動作を行う映像圧縮プロセッサを開発した。専用

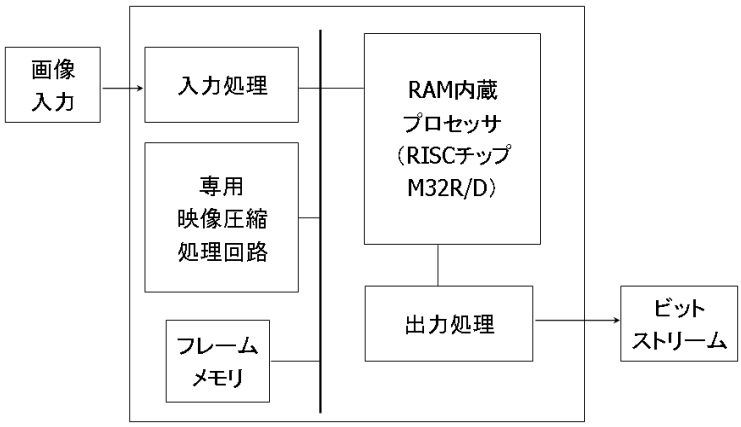


図 3.22　映像圧縮処理システムのブロック図

回路の規模は 50k ゲートであった。映像処理用に専用のフレームメモリを使い、システム制御のソフトウェアは M32R/D 内蔵の RAM を使って実行した。

トップダウン協調検証手法で実施した評価の手順について、以下に説明する。

①　システムレベルシミュレーション

映像圧縮のアルゴリズムをすべてＣ言語で記述し、アルゴリズムによる映像圧縮率と映像品質及び演算量の評価を行った。

②　高位言語モデルによる協調シミュレーション

ソフトウェア部分もハードウェア部分も C 言語によりモデル化を行った。ハードウェア部を制御するレジスタやデータを格納するバッファ群を共有メモリ空間に割り付けることにより、ハードウェアとソフトウェア間のデータのやりとりを実現した。すべて、C プログラムで表現され、ホストマシンで直接実行できるために高速にシミュレーションを行うことができた。ハードウェアとソフトウェアによる機能的な分担の検証を行なった。

③　命令セットモデルによる協調シミュレーション

C 言語で記述された組込みソフトウェアを調達したプロセッサ M32R/D の機械語にコンパイルし、ターゲットとなるプロセッサのキャシュやパイプライン動作の模擬を行う命令セットシミュレータを使ってインタプリティブに実行した。RT レベルでモデル化されたハードウェアはサイクルベースシミュレータで実行した。両方のシミュレータをシングルプロセスで実行できないため、プロセス間通信を使ってデータの送受を行なった。命令セットモデルによる協調シミュレーションの実施例(Tsutsuji を使用)を図 3.23 に示す。実行サイクルによる性能評価を中心に検証を行った。

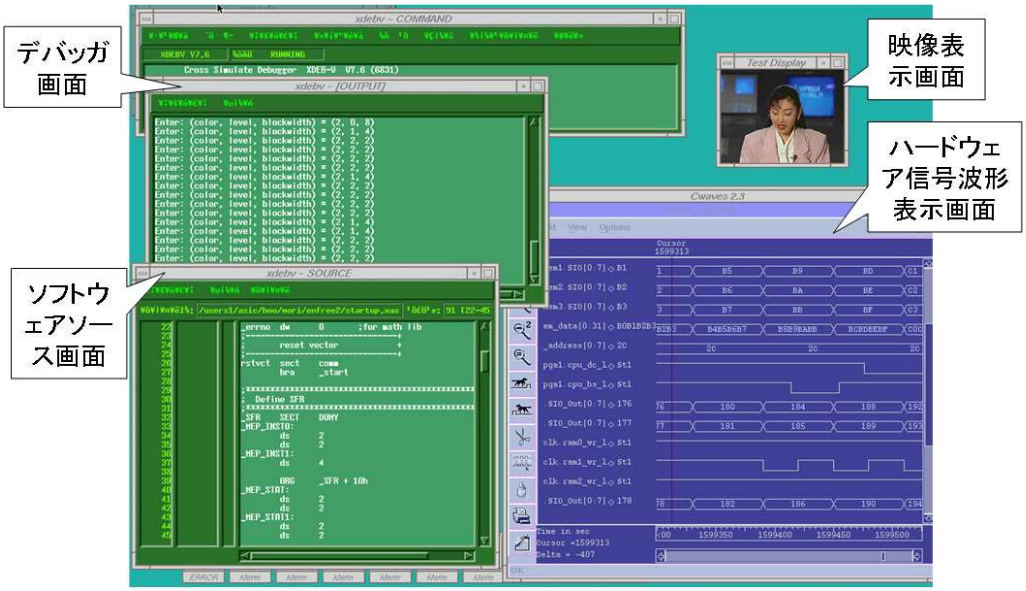
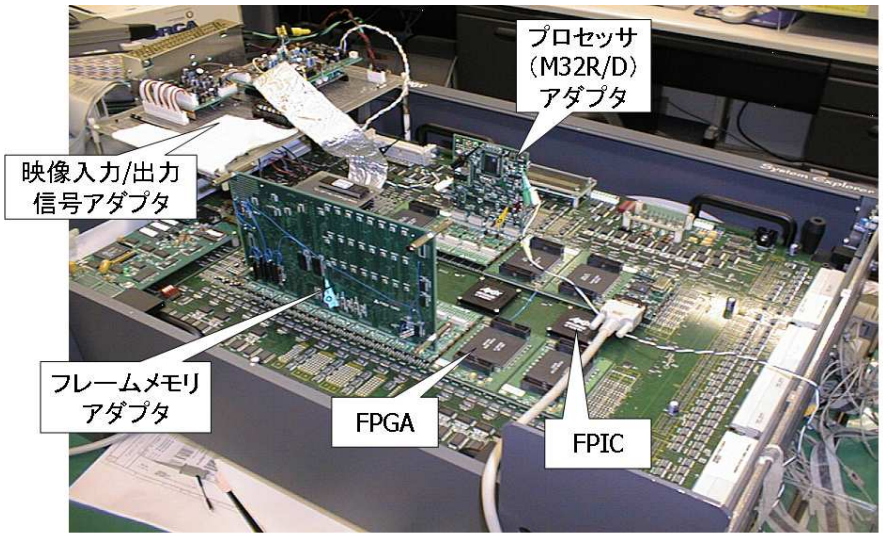


図 3.23　映像圧縮処理システムの協調シミュレーション実行画面例

④　実チップモデルによる協調エミュレーション

システムエミュレータ MP4[Apt96]上に、作成したプロセッサの M32R/D アダプタ、フレームメモリアダプタ、および、映像入力/出力信号アダプタを搭載した。プログラムはプロセッサ M32R/D の内蔵メモリに格納した。映像圧縮処理回路の論理回路を 4 石の FPGA（最大 10k ゲート搭載可能）にマッピングし、これらの部品間の接続情報をプログラマブルスイッチアレイ(FPIC)にマッピングした。当初 1Mhz で動作させ、最終的に 4Mhz までチューニングを行い、実機に対して 1/5 の速度でエミュレーションを可能とした。実際の映像を入力しながら、動きを伴った映像圧縮処理の評価を可能とした。協調エミュレーションの実施例を図 3.24 に示す。



当初1MHz→チューニングして4MHzで動作

図 3.24　　映像圧縮処理システムの協調エミュレーションの実行例

(5)　適用結果と評価

映像圧縮処理システムに適用したトップダウン協調検証の実行結果と、繰り返し検証の効率化の評価について説明する。

①トップダウン協調検証の実行結果と評価

トップダウン協調検証の実行結果を表 3.5 に示す。表 3.5 では、実機での速度を 1 の単位として、それぞれのシミュレーション速度を相対的に示している。

・システムモデルでは、すべてホストマシン上で直接実行できることから、ホストマシンの性能に依存するが、実機の 1/12.5 という高性能で評価でき、アルゴリズムの評価を繰り返し行うのに有効であった。

・高位言語モデルでの協調シミュレーションは実機の 1/125 の性能で評価できたので、ハードウェアとソフトウェアによる分担機能の評価を繰り返し行うのには十分に効果があった。

・命令セットモデルでの協調シミュレーションでは、クロック単位のタイミングを考慮したサイクルベースの協調シミュレーションであったが、プロセス間通信のオーバーヘッドのため、性能は実機の 1/250,000 となった。繰り返し実行するには無理があるが、正確な性能評価をすることができた。

・実チップモデルによる協調エミュレーションは実機の 1/10 の性能で、人間の目視による動画の画質評価と詳細な性能評価として、大変有効であった。また、組込みソフトウェアのコンカレントなデバッグが可能となり、システム全体の開発期間短縮に貢献できた。

表 3.5　映像圧縮処理システムのトップダウン協調検証の実行結果

<table>
<tr><th rowspan="2">検証
ステージ</th><th colspan="3">モデリング</th><th rowspan="2">協調検証
相対性能
（実機=1）</th></tr>
<tr><th colspan="2">SW</th><th>インタフェース</th><th>HW</th></tr>
<tr><th></th><th>AP</th><th>プロセッサ</th><th></th><th></th></tr>
<tr><td>①システムモデル</td><td colspan="3">Cモデル</td><td>0.08（実機の1/12.5）</td></tr>
<tr><td>②高位言語モデル</td><td colspan="2">Cモデル</td><td>共有メモリ方式</td><td>Cモデル</td><td>0.008（実機の1/125）</td></tr>
<tr><td>③命令セットモデル</td><td rowspan="3">Cモデル→ターゲットの機械語</td><td>命令セットモデル</td><td>プロセス間通信</td><td>RTLモデル</td><td>4×10^{-6}（オーバヘッド大）</td></tr>
<tr><td>④実チップモデル</td><td>実チップモデル</td><td rowspan="2">メモリマップド方式</td><td>FPGAモデル</td><td>0.1（実機の1/10）</td></tr>
<tr><td>実機</td><td>実チップ</td><td>ASIC</td><td>1（20MHz）</td></tr>
</table>

AP：アプリケーションプログラム

②繰り返し検証の効率化の評価

繰り返し検証の効率化に関する定性的な評価を図 3.25 に示す。

繰り返し検証が実用レベルで効果を発揮するためには、検証時間そのものが実用に耐えられる必要がある。また、各検証ステージでの検証環境の準備が効率良く行なえること、各検証ステージでのモデリングが効率良く行なわれることが必要である。

(a)　検証環境の準備時間

シミュレーション環境の統合により検証環境の準備時間が減少した。また、共通テストベンチの適用により、協調シミュレーション（高位言語モデル）以降の検証ステージで同じテストデータが使えることから準備作業の軽減につながった。

(b)モデリングの時間

ハードウェアとソフトウェアのインタフェースとしてコンポーネント論理バスアーキテクチャを採用することにより、インタフェース部分のモジュール化が可能になり、協調シミュレーション（命令セットモデル）以降のモデリング時間を軽減できた。

3.6　考察

自製したプロセッサ（ASAP）を使用した線描画システムと、調達したプロセッサ（M32R/D）を使用したマルチメディア系の組込みシステムである映像圧縮処理システムを対象に、提案したトップダウン協調検証手法を適用した。今回提案したトップダウン協調検証手法適用による効果を図 3.26 に示す。

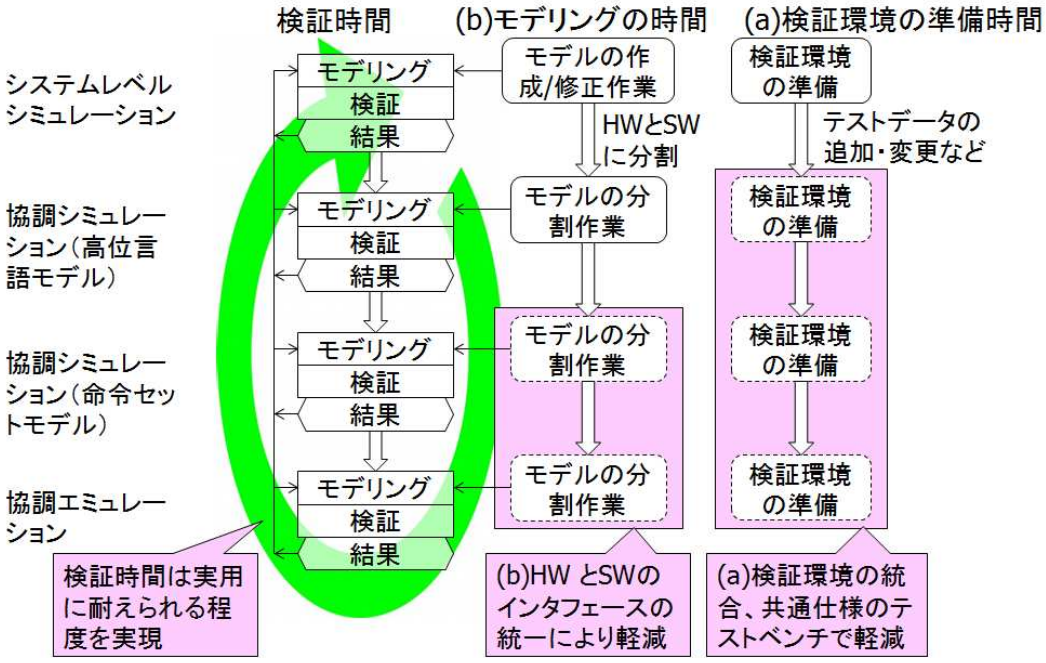


図 3.25　繰り返し検証の効率化

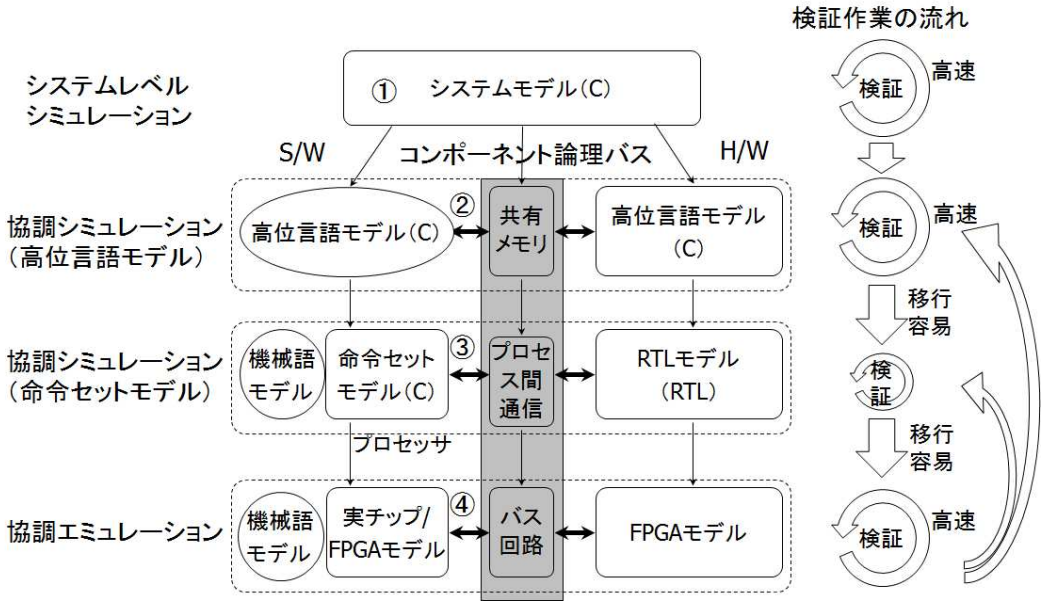


図 3.26　トップダウン協調検証手法の適用による効果

実機評価が必要な独創的な組込みシステム製品開発を行う場合には、以下の手順で効率よくタイムリーに評価・検証を行う必要がある。

(1) 独創的なアルゴリズムなどを設計の上流でシステムレベルシミュレーションにより繰り返し評価・検証行う。

(2) ハードウェアとソフトウェアの組み合わせで実現したシステムの機能と性能について評価・検証を行う。

(3) 実機に近い環境で、実動作による評価・検証や人間の感性による評価を行う。満足解が得られない場合は、前のステージに戻って設計を見直して、繰り返し評価・検証を行う。

本研究では、これらの課題を実現するために、従来各検証ステージ毎にバラバラであった検証環境を統合し、共通テストベンチを適用した。また、ハードウェアとソフトウェアとのインタフェースとしてコンポーネント論理バスアーキテクチャを採用し、協調シミュレーションから協調エミュレーションまでの各検証ステージに適用した。また、協調検証ではプロセッサのモデル化が重要になることから、自製の場合と調達の場合における各検証ステージでのモデルの実現方法を明確にした。

2 件の事例に適用した結果をまとめると、以下の効果を得ることができたと考えられる。

(1) システムレベルシミュレーションでは、高速なシミュレーションにより、繰り返し効率よく評価・検証ができた。

(2) (1)の評価・検証された機能をハードウェアとソフトウェアで分担し、インタフェースとしてコンポーネント論理バスアーキテクチャを採用し、これ以降の検証ステージに

すべて適用した。インタフェースの実装方法は、いろいろ存在したが、実装特有な変更部分は少なく、上位から下位のステージへの移行をそれぞれ容易に行うことができた。

(3) 高位言語モデルの協調シミュレーションでは、インタフェースを共有メモリ方式で実装し、ハードウェアとソフトウェアをともに高位言語（C）でモデル化し、ホストマシンでコンパイルしてシングルプロセスで実行することで高速なシミュレーションを実現できた。共通テストベンチとの組み合わせで、繰り返し効率よく評価・検証することができた。

(4) 命令セットモデルの協調シミュレーションでは、インタフェースをプロセス間通信でしか実現できなかったために、オーバーヘッドが大きかったが、ハードウェアは RTL モデルでソフトウェアは機械語モデル（プロセッサは命令セットモデル）により、大まかな性能評価をすることができた。頻繁な繰り返し実行は無理があるが、次の協調エミュレーションへの移行は、インタフェース部分の変更が少なく容易に移行することができた。

(5) 協調エミュレーションでは、プロセッサを実チップモデルまたは FPGA モデルにマッピングし、残りのハードウェアを FPGA モデルにマッピングすることにより、実機に近い環境で繰り返し効率よく評価・検証することができた。

以上により、提案したトップダウン協調検証手法は、独創的な組込みシステム製品の開発を効率よく評価・検証することに貢献できると判断できる。

自製のプロセッサでは、プロセッサのモデルとハードウェア部のモデルをシングルプロセス上で実行することが可能で、協調シミュレーションを効果的に実現可能である。それに対して調達したプロセッサではモデルの入手が困難な場合が多く、命令セットモデルを入手できた場合でも、プロセッサの命令セットシミュレータとハードウェアモデルのシミュレータをマルチプロセスで動かす必要がある場合が多い。今回の事例では、コンポーネント論理バスでのデータ授受にプロセス間通信を使ったが、そのオーバーヘッドは非常に大きく実用上大きな課題である。協調シミュレーションではシングルプロセスで実行可能な環境を作り込むことが重要であり、同じシミュレータに組み込めるプロセッサの命令セットモデルや RTL モデルの入手が必須である。

3.7 まとめ

マルチメディア系や制御系のように実機評価が必要な組込みシステムの開発に対して、検証環境を統合し、共通テストベンチを適用し、ハードウェアとソフトウェアのインタフェースとしてコンポーネント論理バスアーキテクチャを採用したトップダウン協調検証システムを提案した。新しいアイデアのアルゴリズムをシステムレベルシミュレーションで評価・検証を行う検証ステージから、ハードウェアとソフトウェアに分割して協調シミュ

レーションにより機能と性能の評価・検証を行う検証ステージを経て、実機に近い環境で協調エミュレーションにより実機動作と人間の感性による評価・検証を行う検証ステージまで、繰り返し効率よく検証できることを示した。このようなトップダウン協調検証手法を取り入れるこれにより、新しい独創的な組込みシステムを、各検証ステージで不具合を取り除きながら、効率よく高品質に開発できると考える。

組込みシステムの大規模化・複雑化は今後も進むと予想される。設計の課題については、再利用可能な資産（IP）の活用と IP を統合しやすくするプラットフォームベース設計の適用、さらに抽象度の高い高位言語の採用と高位合成ツールの適用により改善されてきている。それに対して、検証の課題は IP を含んだシステム全体をものづくりの前に動作させて評価・検証が必要となることから、規模の拡大に伴い難しさが増大する傾向にある。そのために、本研究で提案したような、ハードウェアとソフトウェア間のインタフェースを統一して、それぞれの機能をモデル化することにより、より効率のよい検証環境を実現することは有用なアプローチと考えられる。

特に、IoT 時代が到来し、組込みシステムが通信ネットワークでクラウドに接続され、クラウド上のアプリケーションと連携してサービスを提供する大きな変革期を迎えており、如何に価値を創造するかが求められている。この価値を創造するためのアプローチとして「デザイン思考」が注目されており、その骨子はユーザの目線に立って「プロトタイピングによる迅速な繰り返し評価」であると言われている。今回提案した「トップダウン協調検証手法」は、効率良くシステムの評価検証を繰り返すという観点で、IoT 時代でも有効な手法と考えられる。

第 4 章　複合領域での教育手法をベースとした組込みシステム技術者教育の提案と実践評価

4.1　はじめに

既に述べてきたように、1990 年代後半から 2000 年代において組込みシステムはデジタル情報家電や通信機器、自動車関連機器を含めて基幹製品のかなりの部分を占めるようになってきた。このような組込みシステムは、ハードウェアとソフトウェアを組み合わせてそれぞれの分野に適した製品として実現されている。半導体技術やデジタル化の進展にともない、組込みシステムは、異なる技術を統合してシステム化が行われ、ますます大規模化・複雑化してきている。複数の技術を組み合わせて、応用分野に最適なシステムを実現することがますます重要になってきており、これらを実現できる技術者としてシステムアーキテクトが求められている。

本章では、1990 年代後半から 2000 年代を念頭に、4.2 節で組込みシステムに必要なスキルと組込みシステム技術教育の現状について説明し、4.3 節で組込みシステム技術教育の問題点を論じる。次に、4.4 節で組込みシステム技術者の教育手法を 4 つの視点から提案し、4.5 節で提案に基づいて開発したカリキュラムの概要と実践評価した結果について論じる。4.6 節で考察とまとめを行う。さらに IoT 時代と言われる 2017 年の今の視点からの考察も行なう。

4.2　組込みシステムにおける必要なスキルと組込みシステム技術教育の現状

4.2.1　組込みシステム製品開発で必要となるスキル

第 2 章の 2.6 節で説明したように、経産省・IPA では、組込みソフトウェア開発力強化を狙いとして、SEC を設立し、2005 年に「組込み技術システム標準」ETSS を策定した。ETSS はスキル標準、キャリア標準、教育研修標準から構成されている。

スキル標準のフレームワークでは、必要とするスキル群を 3 つのスキルカテゴリ（技術要素、開発技術、管理技術）に分類し、スキル粒度として最上位の抽象的な第一階層からより詳細な第三階層まで階層的に展開し、最下層のスキルに対して必要なスキルレベル（初級、中級、上級、最上級）を設定している。

組込みスキル標準 ETSS をベースに組込みシステム開発に必要となるスキル体系をまとめたものを図 4.1 に示す。図 4.1 の左側に、ETSS の 3 つのスキルカテゴリである技術要素、開発技術、管理技術を示す。図 4.1 の右側に、ETSS のスキル体系に対応した、本論文の取り組みの位置づけを示す。プロダクトは ETSS の技術要素に、プロセスは開発技術に、プロジェクトは管理技術に対応させている。

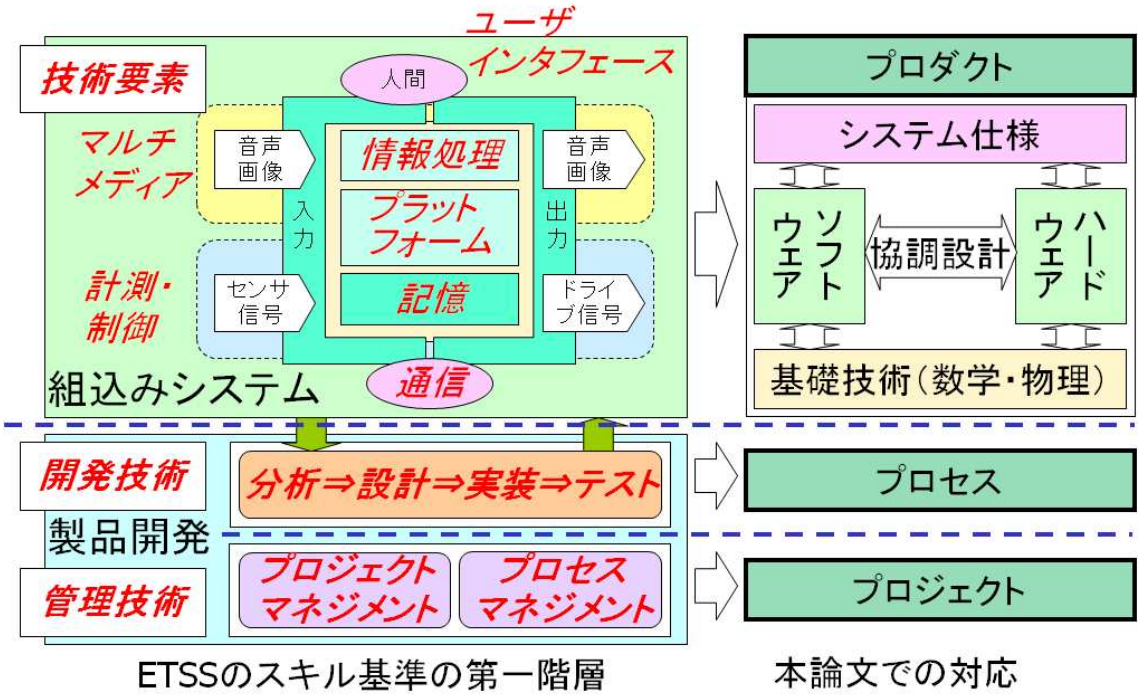


図 4.1　組込みシステム開発に必要なスキル体系と本論文での対応

(1) 技術要素：プロダクトの実現方法に関するスキル

ETSS では、技術要素として、以下に示すように 7 つの第 1 階層（コロンの後に第 2 階層を示す）について例示している。

① 通信：有線、無線、放送、インターネット
② 情報処理：情報入力、セキュリティ、データ処理、情報出力
③ マルチメディア：音声、静止画、動画、統合
④ ユーザインタフェース：人間系入力、人間系出力
⑤ 記憶（ストレージ）：メディア、インタフェース、ファイルシステム
⑥ 計測･制御：理化学系入力、計測・制御処理、理化学系出力
⑦ プラットフォーム：プロセッサ、基本ソフトウェア、支援機能

ETSS は、組込みソフトウェア技術者向けを想定しているが、組込みシステムとしての要求分析からシステム設計の上流も対象にしていることから、技術要素（プロダクト）はそのまま組込みシステム技術者にも適用可能であると考える。

(2) 開発技術：開発プロセスに関するスキル

開発の上流から下流までの流れに従って行う開発工程に関するスキルに相当する。各工程内で必要となるスキルと工程間のインタフェースを理解して進める必要がある

開発技術（プロセス）については、ハードウェアとソフトウェアでは大きく異なることから、ハードウェアについては、新たな定義が必要となる。九州大学の QUBE では、システム LSI 設計技術のスキルマップを提案し、教育効果の測定やカリキュラムの改善への取

り組みが検討されている[林田 08]。

(3) 管理技術：プロジェクトマネジメントに関するスキル

管理技術（プロジェクト）の基本形は、組込みシステム全体で共通と考えられる。開発を進めていくときのプロジェクトマネジメントに関するスキルに相当する。プロジェクトでは、関係者が参画し、PDCA のサイクルをまわして、製品としての QCD の目標を実現していく。

図 4.1 の右側に示すように、組込みシステムは製品毎に要求仕様から導き出されたシステム仕様を満足するように協調設計によりハードウェアとソフトウェアの最適な組み合わせが実現される。この協調設計による開発プロセスにおいて、階層的により抽象的なモデルから詳細なモデルに変換しながら設計が行われていく。この過程で、解析・評価を行うためのモデリング技術やそれらのベースとなる数学や物理などの基礎技術も必要となる。

このような協調設計のアプローチで、プロダクトやプロセスで必要となるスキルを修得するための体系的な教育手法について検討する必要がある。

4.2.2 組込みシステム分野の技術者育成の現状

1990 年代から組込みシステムがいろいろな分野で使われるようになり、2000 年にはコンピュータ（4～64 ビット）の年間出荷台数（約 80 億個）の約 98%は組込みコンピュータとして使われているとの報告があり、従来からのコンピュータ科学の教授法の見直しが指摘されている[Ten00]。

組込みシステムは応用分野に特化してハードウェアとソフトウェアを組み合わせて実現されていることから、組込みシステムの開発を担う技術者には、応用分野の知識とともにハードウェア技術やソフトウェア技術などの異なる技術を組み合わせて分析・評価を行う能力が必要となる。

このように組込みシステムの分野は、既存のコンピュータ科学（CS：Computer Science）と電子工学（EE：Electronic Engineering）のカリキュラムの両方に跨っており、この文化のギャップを埋める新たな取り組みの必要性が指摘されている[Hen07]。

最近のさらなる適用分野の拡大、規模の拡大、複雑さの増大から、組込みシステム技術者の量と質の拡大が求められており、組込みシステム分野の技術者育成に対して図 4.2 に示すように、いろいろな角度からの取り組みが行われている。

特に、組込みシステム製品の価値を決めるシステムアーキテクトの育成が求められている。

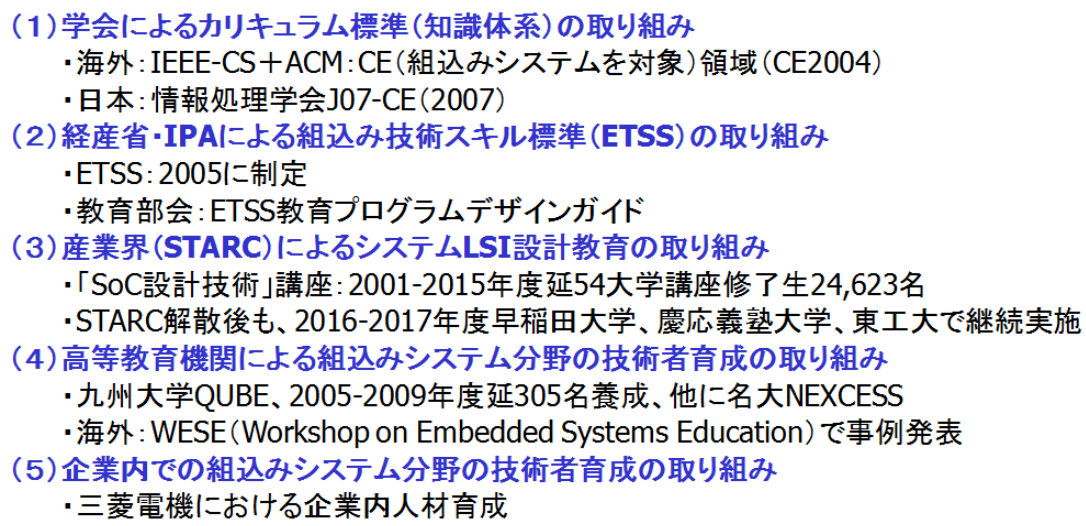


図 4.2　組込みシステム分野の技術者育成の現状

(1)　学会によるカリキュラム標準（知識体系）の取り組み

組込みシステム分野は、異なる技術にまたがった複合領域であることから、従来のコンピュータ科学（CS）と電子工学（EE）の学科では対応できないことが指摘されるようになり、2000 年初期から、組込みシステム分野の教育についての議論が行われるようになった。

米国では、IEEE-CS と ACM がタスクフォースを組み、2001 年に CS 領域に対するカリキュラム CC2001 を公表した。2002 年に IS（Information System）領域のカリキュラム、2003 年に SE（Software Engineering）領域のカリキュラム、2004 年に組込みシステムを意識した CE（Computer Engineering）領域のカリキュラム（CE2004）[Joi04]、2005 年に IT（Information Technology）領域のカリキュラムと、標準カリキュラムを整備し、最終的に CC2005 を策定した。

国内では、情報処理学会が米国の CC2005 の発行を受けて、日本の立場から検討を行い、2007 年度に情報専門学科カリキュラム標準 J07 を発表した[筧 08]。この中で、コンピュータエンジニアリング領域（J07-CE）[大原 08]が組込みシステム分野に対応している。J07-CE は米国の CE2004 を参考にしているが、ハードウェアの比重を組込みソフトウェアにやや移した内容となっている。

(2)　経産省・IPA による組込みシステムスキル標準の取り組み

日本の強みとする組込みシステム製品の開発において、大規模化・複雑化に対応して、組込みソフトウェアの品質問題や組込みソフトウェア技術者の不足が顕在化し、ソフトウェアのものづくり力の強化が指摘されるようになってきた。

経産省・IPA により、SEC に組込みソフトウェア開発力強化推進委員会が 2004 年（準

備会は 2003 年）に設立され、組込み技術スキル標準（ETSS）が 2005 年にを策定された[渡辺 06] [情報 09A]。筆者は開始時から委員および教育部会主査として参画し、ETSS における教育研修標準の策定に取り組んだ[経済 07]。また、この活動における成果物として「組込みスキル標準 ETSS 教育プログラムデザインガイド」を主査としてとりまとめた[情報 09B]。

また、経産省は、2007 年に企業や大学で実践されている組込みシステム分野の技術者育成の取り組みの概要を「組込み技術者教育ベストプラクティス集」として公開している[情報 07A]。この中で、九州大学の QUBE や名古屋大学の NEXCESS や筆者が参画していた㈱ゼネテック システム技術者養成センターの活動などが紹介されている。

(3) 産業界（STARC）によるシステム LSI 設計教育の取り組み

シリコン系半導体研究やシステム LSI 設計教育に関する日本の大学と半導体産業界のギャップを埋めるために、半導体業界各社が協力して大学の研究活動を支援しシステム LSI 設計技術者の教育を推進する組織として、㈱半導体理工学研究センター（STARC）が 1995 年に設立された。システム LSI 設計技術者の教育については、2000 年度から SoC 設計技術者（SoC アーキテクト）の育成とそのための環境作りに取り組み、2001 年度から大学向けの「SoC 設計技術」講座（SoC 概要編、LSI 設計編、組込みソフトウェア編、システム設計編）が開始された。その後、講義の他に実習講座も行われ、教育内容も継続的に改善を図りながら 2015 年度まで続けられた[小澤 03] [橘詰 06][小池 12]。過去 10 年間の実績は、51 大学での講座修了生 15,337 名、実習修了生 1,849 名となっている[小池 11]。筆者は、システム設計編の取りまとめおよび第 1 章の「組込みシステムとその概要」と第 6 章の「機能検証技術」を担当してきた。なお、STARC が 2016 年 5 月に解散した後も、現在（2017 年度で 17 年目）まで自主的に早稲田大学、慶応義塾大学、および、東京工業大学で講義が行なわれ、筆者も非常勤講師を務めている。

(4) 高等教育機関による組込みシステム分野の技術者育成の取り組み

国内での主な取り組みを以下に示す。

北九州地域において、産官学が連携して、システム LSI の設計開発拠点となり、システム LSI 設計技術者（高度なレベルの人材）を育成するための教育システムの確立を目指して、「福岡県システム LSI 設計開発拠点推進会議」が 2001 年 2 月に発足した。この会議のもとに「シリコンシーベルト福岡」プロジェクトが進められ、九州大学に研究拠点として「システム LSI 研究センター」が設立され、「福岡システム LSI カレッジ」と連携してシステム LSI 設計の社会人教育の取り組みが行われてきた。これらの取り組みを融合して、2005 年度に「システム LSI 設計人材育成実践プログラム」QUBE が開設された。このプログラムは、ハードウェアとソフトウェアを含むシステムを見渡す高い能力を有するシステム LSI 設計技術者の育成という企業ニーズに応える教育プログラムとなっている[築添 06]

[Yas06] [Tsu06]。2009 年度までの 5 年間に延べ 305 人の社会人を養成している（受講者延べ 1,451 人、合格者延べ 1,319 人、修了者延べ 366 人）[築添 10]。

また、名古屋大学では、社会人向けの「組込みソフトウェア技術者人材養成プログラム」NEXCESS を 2004 年度に開設した[山本 06] [Yam05]。組込みソフトウェア開発の理論から、社会人教育に強く要求されるスキル育成を達成するために、教育コースの体系化、育成スキルの明確化、学習者のスキル育成への取り組み姿勢の評価を行なっている。2008 年度までの 5 年間で社会人の修了者は延べ 1,224 人となった[名古 09]。なお、筆者は NEXCESS アドバイザリ委員会の委員として、NEXCESS の取り組みを支援してきた。

両大学共に、プロジェクト終了後は、新しい体制で社会人向けの教育が行われている[平川 12][石田 12][山本 12]。

なお、その他の高等教育機関での活動については、前述の「組込み技術者教育ベストプラクティス集」に記載されている[情報 07A]。

その後の取り組みとして、「分野・地域を超えた実践的情報教育協働ネットワーク（enPiT1：大学院生向け）」プロジェクト（2012 年度～2016 年度）[enP17A]）に続いて、「成長分野を支える情報技術人材の育成拠点の形成（enPiT2、学部生向け）」プロジェクト（2016 年度～2020 年度）[enP17B]が展開されている。enPiT の特徴は、ビッグデータ・AI 分野（中核拠点：大阪大学）とセキュリティ分野（運営拠点：東北大学）と組込みシステム分野（運営拠点：名古屋大学）とビジネスシステムデザイン分野（運営拠点：筑波大学）の 4 分野で構成され、それぞれの運営拠点/中核拠点校を中心に連携校とネットワークを形成して運営されていることである。まさに、IoT 時代にふさわしい形で、人材育成の取り組みが行なわれていると言える。

海外での主な取り組みを以下に示す。

組込みシステムが従来の簡単なコンピュータによる入力ー出力タスクを主体としたものから、高性能なマイクロプロセッサベースで入出力に加えて多くの計算機能を実現するものに変化してきており、次世代の組込みコンピューティングのコースは、部品の議論からシステムの解析や設計の議論に移るべきとの指摘がなされている。部品が複雑になるに従い、メソドロジー（方法論）がますます重要となっている[Wol00]。

その後、2005 年から毎年 Workshop on Embedded Systems Education(WESE)が開催されるようになり、日本の上記の 2 大学を含め、米国のカリフォルニア大学バークレイー校などの大学での組込みシステム分野を対象とした人材育成の取り組みが報告されている[Jac05][Yam05] [San05][Tsu06]。また、組込みシステム関連分野でカーネギメロン大学の組込みシステム教育の取り組みが報告されている[Koo05]。

(5) 企業内での組込みシステム分野の技術者育成の取り組み

一例として三菱電機での取り組みを示す。筆者は 2002 年度から 5 年間、企業内の人材育

成を行う技術研修センター（後に人材開発センターと改称）に在籍し、「情報・ソフトウェア教室」と「半導体・デバイス教室」の教室長として、該当分野の技術者育成をとりまとめてきた。

半導体分野の技術者育成では、トップダウン協調設計・協調検証システムを使って組込みシステム（システム LSI）開発を行うための技術ゼミナール（講義演習）などを企画し運営してきた。また、情報・ソフトウェア分野では、ETSS を参考に組込みソフトウェア開発関連と、PMBOK を参考にソフトウェアプロジェクトマネジメント関連の講座・演習の充実に重点的に取り組んだ[清尾 06A]。その後、2 年間株式会社ゼネテックで組込みシステム技術者育成に携わった[ゼネ 07]。

学会、公的機関、産業界、高等教育機関、企業での現状における取り組みをまとめると、組込みソフトウェアやシステム LSI を中心に展開にしているものが多い。ハードウェアとソフトウェアから構成される組込みシステムが大規模化・複雑化し、技術のブラックボック化が進む中で、独創的な組込みシステム製品の開発を切り開く技術者（システムアーキテクト）を育成するためには、組込みシステム技術教育の問題点を分析して体系的な取り組みが必要であると考える。

4.3　組込みシステム技術教育の問題点

複合領域としての組込みシステムの開発において、ハードウェアとソフトウェアの互いに異なる技術の組み合わせでプロダクトを最適に構築し、分析からテストまでのプロセスをまわし、関連する職務を分担しながら PDCA サイクルでプロジェクトを進めていく必要がある。このような組込みシステムの技術教育において、以下のような４つの問題点があると考える。

(1)基礎から応用までのステップが明確でない。

全体像として、図 4.1 に示すように、プロダクトの実現に必要な組込みシステムの技術要素の全体、開発の上流から下流までの全体の開発プロセス、および、プロジェクトの開始から終了までの PDCA の流れを理解する必要がある。複合領域である組込みシステムの全体像を 1 回の学習で修得することは難しい。また、ハードウェア技術やソフトウェア技術にしても同様である。最初は小規模なものや基本的なものを対象にはじめ、何回か繰り返しながら規模や範囲を拡大し経験を積み重ねて修得していくことが必要である。

(2)プロダクトとプロセスの両面のインタフェースに関する教育が不足している。

複合領域である組込みシステムの開発において、プロダクト、プロセスやプロジェクトにおいて、いろいろなインタフェースが存在する。インタフェースのいずれかの側にいる立場だけで考えると、思考の範囲が限定的になり、全体の動作を理解したり、システム全体として最適化を考えたり、トラブルの原因を見極めたりすることが難しくなる。このような課題を克服するには、インタフェースの両面からの理解が効果的であると考える。

(3)技術の積み重ねが不足している技術者にとって、技術の進歩に対応した取り組み方法の

教育が不足している。

現在の組込みシステムを構成する技術は、毎年出現する新しい技術の積み重ねによって、ますます複雑になり、ブラックボックス化が進んでいる。過去から何十年にわたって製品開発を担当してきたベテラン技術者は、製品を開発するたび毎に新技術を吸収し、既に蓄積されている技術に加えながら成長していくことができた。しかし、若い技術者にとっては、技術のスタート時点にさかのぼって、積みあがった技術を同じように体験的に学ぶことはできない。過去から積み上げられた技術をすべてブラックボックスにして、表層的な理解のまま、実際の開発に立ち向かえば、いずれ限界に突き当たり立ち往生する可能性が高い。限られた時間の中で、効果的に技術を習得していくための取り組み方法を考える必要がある。

(4)ハードウェアとソフトウェアのトレードオフに関する教育が不足している。

組込みシステムの開発の中で、製品としての価値を決める重要なことは、異なる技術を組み合わせて実現可能な複数の選択肢の中から、トレードオフにより、製品にとって最適な組み合わせを選択することである。特に、規模の拡大・複雑化に伴い、ハードウェアとソフトウェアによる最適な組み合わせを実現する協調設計でのトレードオフに関する教育の充実が必要である。

4.4　組込みシステム技術の教育手法の提案

4.4.1　組込みシステム技術教育の問題点に対する解決策の提案

4.3 節で指摘した問題点に対して、それぞれ対応する解決策を提案する。

(1) 全体を俯瞰しながら基礎から応用へのスパイラルアップの取り組み

複合領域である組込みシステムの全体像を 1 回の学習で修得することは難しい。まず、最初は、ハードウェアとソフトウェアともに、小規模なものや基本的なものを対象に全体が俯瞰できるように、開発のプロセスをまわす。1 度開発のサイクルを経験すると、少しずつ全体像が見えてくる。次第に規模や複雑さを拡大していくスパイラルアップの取り組みで、何回か繰り返しながら経験を積み重ねることで、確実に修得していくことが可能になる。

(2) 領域間のインタフェースの両面からの取り組み

組込みシステムの開発には、いろいろなインタフェースが存在する。システム全体として最適化・効率化を考えるには、インタフェースの両面からの取り組みが必要である（図 4.3 参照）。

たとえば、プロダクトの視点では、ハードウェアとソフトウェアのインタフェースとして命令セットが存在する。この命令セットを理解し、ソフトウェアではアセンブラや高位言語によるプログラムによりハードウェアを動作させ、ハードウェアでは命令セットを実装するマイクロアーキテクチャを論理回路や機能モデルで実現することになる。命令セットの両面から理解することで、ハードウェアとソフトウェアの機能分担を良く理解するこ

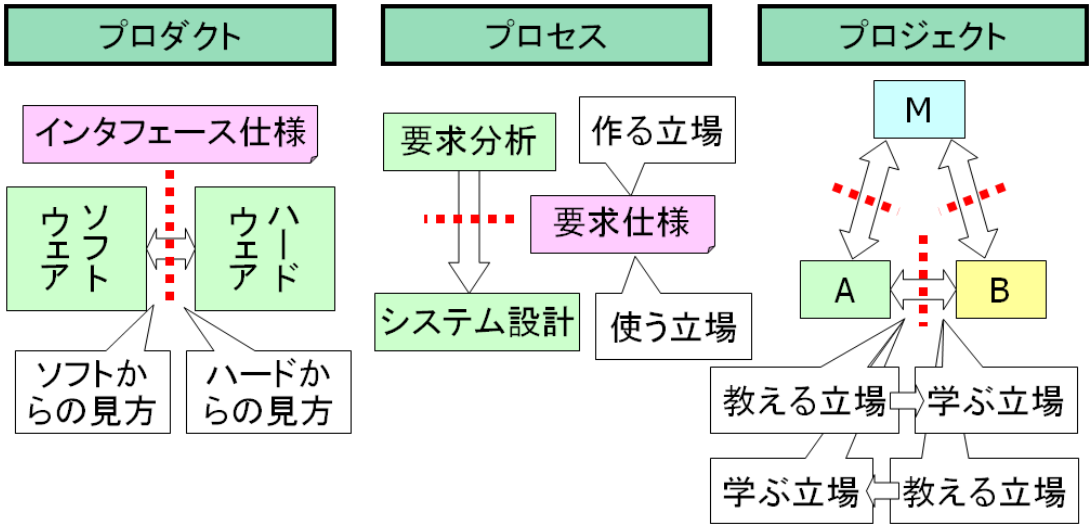


図 4.3　領域間のインタフェースの両面からの取り組みの例

とができる。

また、プロセスの視点では、仕様書をインタフェースとして、上位プロセスの作る立場と下位プロセスの使う立場が存在する。この作る立場と使う立場の両面で学ぶことにより、漏れのないわかりやすい仕様書の作成の難しさとその必要性を理解することができる。この両面の立場で学ぶというやり方は、PBL において、プロセスの各ステップで成果物を交換しながら進めていく場合に相当する[石崎 10]。

さらに、プロジェクトの視点では、構成員の間でいろいろ教え合うことが必要になり、教える立場と学ぶ立場が存在する。教えるためには、対象を十分に理解する必要があり、結果として自ら学ぶことになる。プロジェクトの構成員でソフトウェアの得意な人がソフトウェアを、ハードウェアの得意な人がハードウェアを相互に教えあうことに相当する。相手にわかりやすく教えることの必要性とその難しさを学ぶことになる。

(3) ブラックボックス化した技術の階層的なモデリングの取り組み

限られた時間の中で、効果的に技術を習得していくためには、核となる技術の基本を詳細化されたモデルとして理解し、上位の抽象化されたモデルを活用できるようにする階層的なモデルに従った理解へのアプローチが必要である。ソフトウェアでは、アセンブラ言語と高位言語（C 言語など）の 2 階層で、ハードウェアではレジスタトランスファーレベルの HDL 言語と高位言語（SystemC など）の 2 階層で、システムをモデル化して動作を理解することで対応することができると考える。

(4) ハードウェアとソフトウェアの最適な組み合わせを評価・選択する協調設計の取り組み

ハードウェアとソフトウェアによる最適な組み合わせを実現するためには、特に性能・コストの観点からのトレードオフが重要になる。機能を実現するいろいろなアルゴリズム方式、ハードウェアの実装方式（SoC 対 SiP）、半導体の実現方式（ASIC 対 FPGA）、機

能のハードウェアとソフトウェアの分担範囲、プロセッサの実現方式（シングルコア、マルチコア、メニーコアなど）などの観点から、性能・コストの見積もりを行い、目標となる組込み製品の要求を満足するような組み合わせを実現する協調設計を体系的に教えることが必要である。従来からの協調設計における教育では、例えばハードウェアとソフトウェアを分担して PBL 形式で取り組むケースが多いが、これでは両方を経験しないことから協調設計の教育としては不十分であり、必ず両方を経験させる必要があると考えている。

4.4.2　組込みシステム開発におけるキャリアパスの提案

組込みシステムの開発で重要な役割を担うシステムアーキテクトは、図 4.4 に示すような職務を経験することが望まれる。

最初は、入門として組込みシステムの仕組みと必要な技術の基本について俯瞰的に理解を進めていく。続いて、ハードウェアを制御するソフトウェアの立場と、ソフトウェアに制御されるハードウェアを実現する立場から、それぞれ基礎から応用に向けてスパイラルアップしながら経験を積んでいく。両方の経験を通して全体がある程度見通せるようになると、組込みシステムの設計の上流でハードウェアとソフトウェアによる最適分担を実現する協調設計を学ぶことになる。

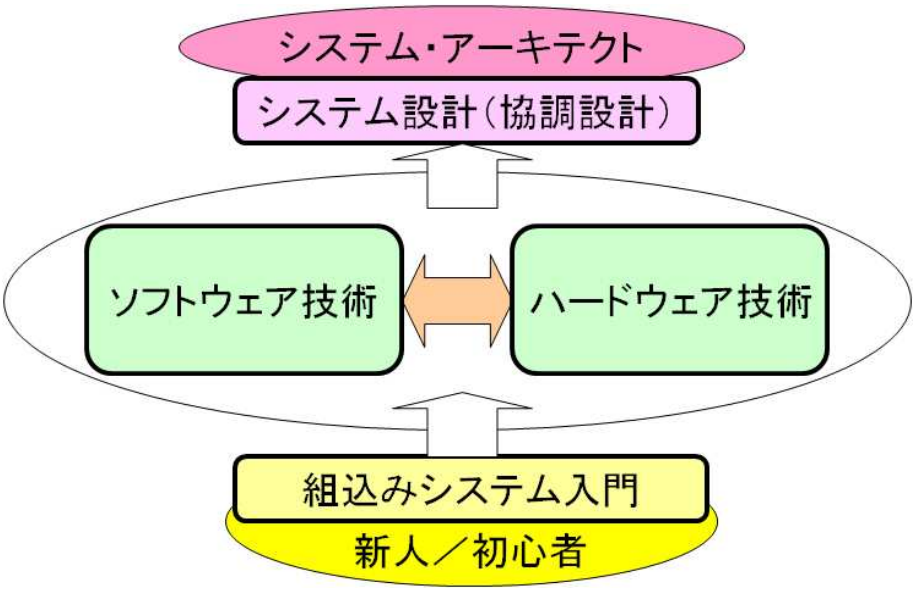


図 4.4　組込みシステム開発におけるキャリアパス

4.4.3　カリキュラムの構成

今回提案した内容を取り込んだカリキュラムの基本構成を図 4.5 に示す。4.4.2 項での組込みシステム開発におけるキャリアパスに基づいて、受講生の一人ひとりが一通り経験することを重視した構成としている。

各コースの概要を以下に示す。

(1) 入門コース：ソフトウェアとハードウェアから実現される組込みシステムの概要を学ぶ。組込みシステムの俯瞰的な全体像と構成要素間のインタフェースを理解する。

(2) 初級コース：組込みシステムを構成するソフトウェアとハードウェアについて、インタフェースである CPU の命令セットを理解した上で、簡単な小さなシステムから順次複雑で大きなシステムにスパイラルアップしながら理解を進めていく。

(3) 中級コース：組込みシステムを実現するためのハードウェアとソフトウェアによる最適な分担をトレードオフ設計により実現していく協調設計技術について学ぶ。また、全体最適化のものづくりを実現していくためのプロセスについても学ぶ。

4.4.1 項での提案をカリキュラムに以下のように反映した。

(1) 全体を俯瞰しながら基礎から応用へのスパイラルアップの取り組み

組込みシステム入門コースから、ハードウェアとソフトウェアのそれぞれの初級コースを経て、中級コースのハードウェアとソフトウェアの協調設計まで、それぞれのコース内で基本から応用に向けて一通り理解しながらレベルを上げていくカリキュラムを実現し、受講生の理解度の向上につながっていくと考える。

(2) 領域間のインタフェースの両面からの取り組み

プロダクトに関してハードウェアとソフトウェアのインタフェースとそれぞれの働きを初級コースで、ハードウェアとソフトウェアによる協調設計を中級コースで扱う。

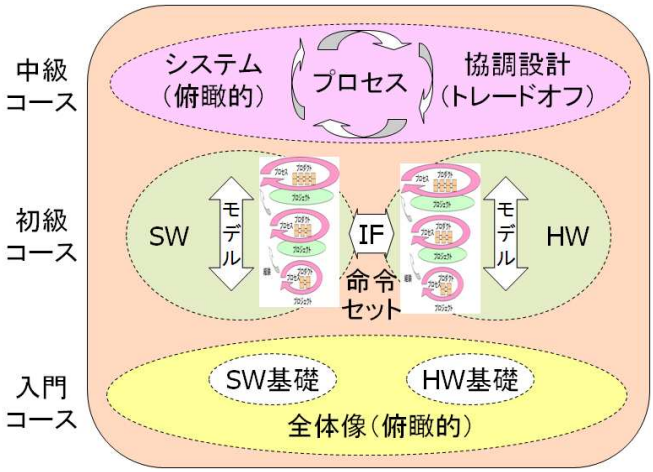


図 4.5　カリキュラムの基本構成

(3) ブラックボックス化した技術の階層的なモデリングの取り組み

ソフトウェアに関しては初級コースのマイクロカーネル制作演習（機械語と C 言語）を通じて、ハードウェアに関しては初級コースでの回路レベルと HDL レベルの演習と中級コースでのシステムレベル（SystemC）の演習を通じて取り組む。

(4) ハードウェアとソフトウェアの最適な組み合わせを評価・選択する協調設計の取り組み

中級コースでハードウェアとソフトウェアのトレードオフによる協調設計を講義形式で理解するとともに、静止画圧縮伸張の JPEG エンコーダを題材に机上演習形式で学べるようにカリキュラムを実現する。

4.5 組込みシステム教育への適用評価

4.4 節で述べて組込みシステムの教育手法の考え方に基づいて提案した高等教育機関でのカリキュラム体系を図 4.5 に示す。提案した個々のカリキュラムの概要とその実践評価について述べる。

なお、図 4.6 の実線で囲んだ組込みシステム入門、組込み HW、組込みシステム応用のコースは、サイバー大学の e ラーニングによる授業に、破線で囲んだ組込み SW のコースは岩手県立大学の夏期集中講座「組込みソフトものづくり塾」に、それぞれ適用したものである。

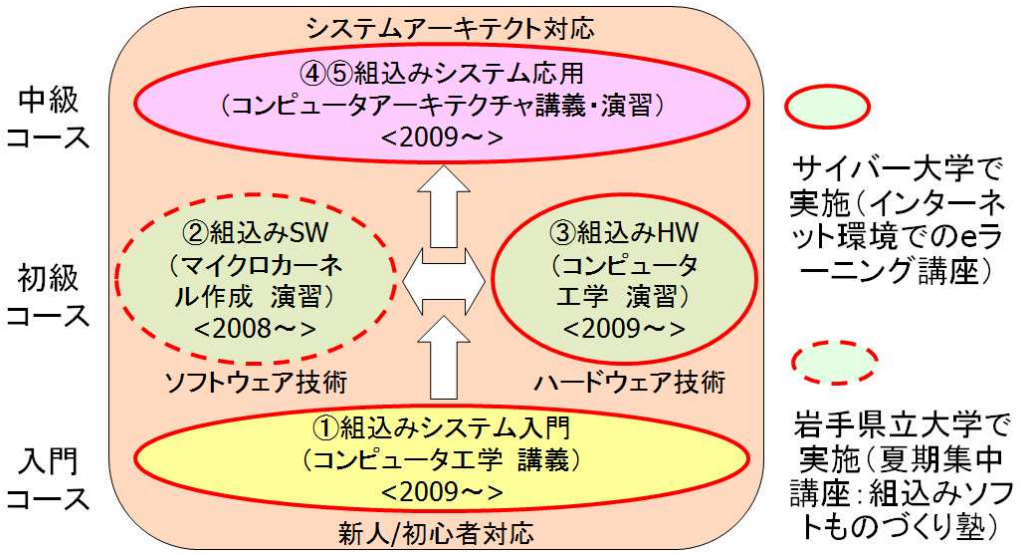


図 4.6 高等教育機関でのカリキュラム体系

4.5.1　入門コースでの取り組み

受講者が組込みシステムに興味を持ってもらうように、システム内容をわかりやすく、また挑戦しがいのある分野であることを説明する必要がある。

図 4.7 に入門コースであるコンピュータ工学講義[サイ 11]（15 単元）のカリキュラム例を示す。カリキュラムは 4 部構成とし、第一部でハードウェア技術の基本である各種の論理回路を理解し、第二部で組込みシステムの全体像とプロダクトの各構成要素を理解する。

第三部でハードウェアを制御する組込みソフトウェア技術を理解し、最後の第四部で組込みシステムの開発の流れ（プロセス）と製品例を学ぶ構成をとっている。

受講生の評価結果と主なコメント、および、講師考察を表 4.1 に示す。

組込みシステムの基礎をハードウェアとソフトウェアの両面から学べ、開発のプロセスも理解できたとの評価もあり、ある程度目的は達成できていると考えられる。なお、ソフトウェア系の受講生が多いことから、やや難しいとの評価であり、補足教材などを充実することで改善の効果は見えるが、さらに初心者に理解してもらうために映像などによるビジュアル化を進めることが改善項目である。

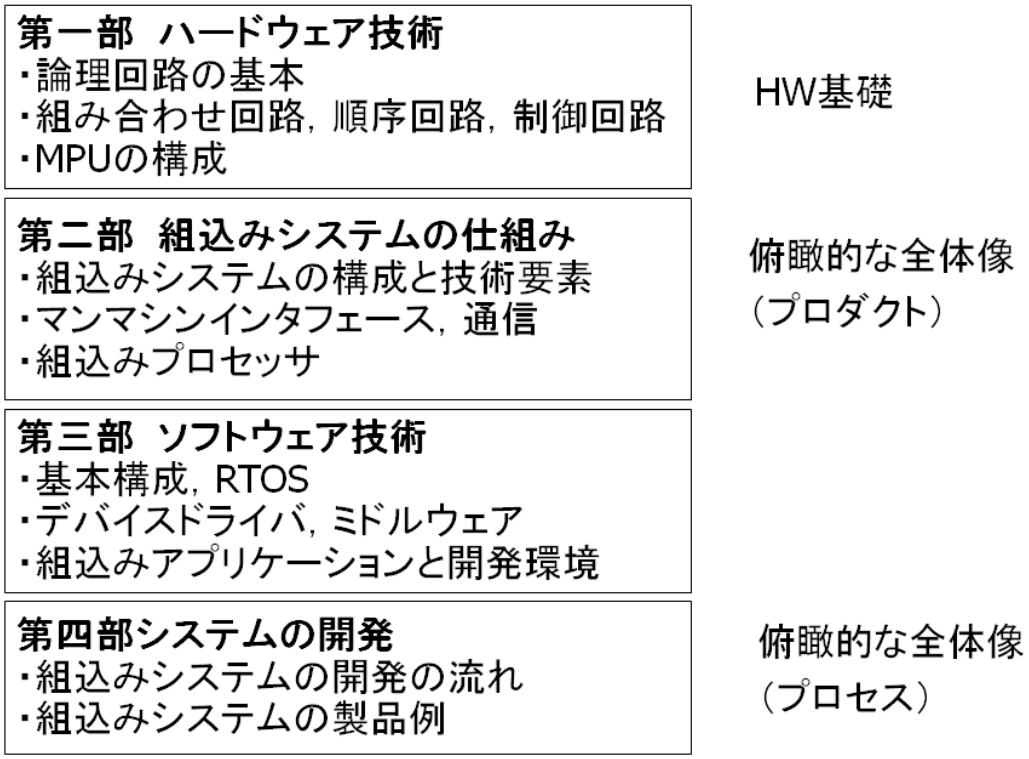


図 4.7　入門コースのカリキュラム例

4.5.2　初級コースでの取り組み

初級コースにおけるソフトウェアとハードウェアの関係を図 4.8 に示す。

(1)ソフトウェアでの取り組み

組込みシステムではシステムソフトウェア（リアルタイム OS、ドライバ、ミドルウェアなど）の上で高位言語（C、Java 等）で記述されたアプリケーションプログラムが実行される。組込みシステムが複雑になるにつれて、ハードウェアとのインタフェースである命令セットからアプリケーションまでの階層は深くなる傾向にあり、システムソフトウェアの中はブラックボックス化してきている。

組込みシステムではハードウェアを扱いリアルタイム処理も必要であることから、システムソフトウェアの中身がどのように構成されているかを理解することが必要である。

リアルタイム OS の核となるカーネル部分をアセンブラと C 言語を使って、自らコーディングして設計することにより、ブラックボックス化したシステムソフトウェアの基本構造を理解することが可能になる。このマイクロカーネル制作コース[吉田 11A]のカリキュラム例を図 4.9 に示す。

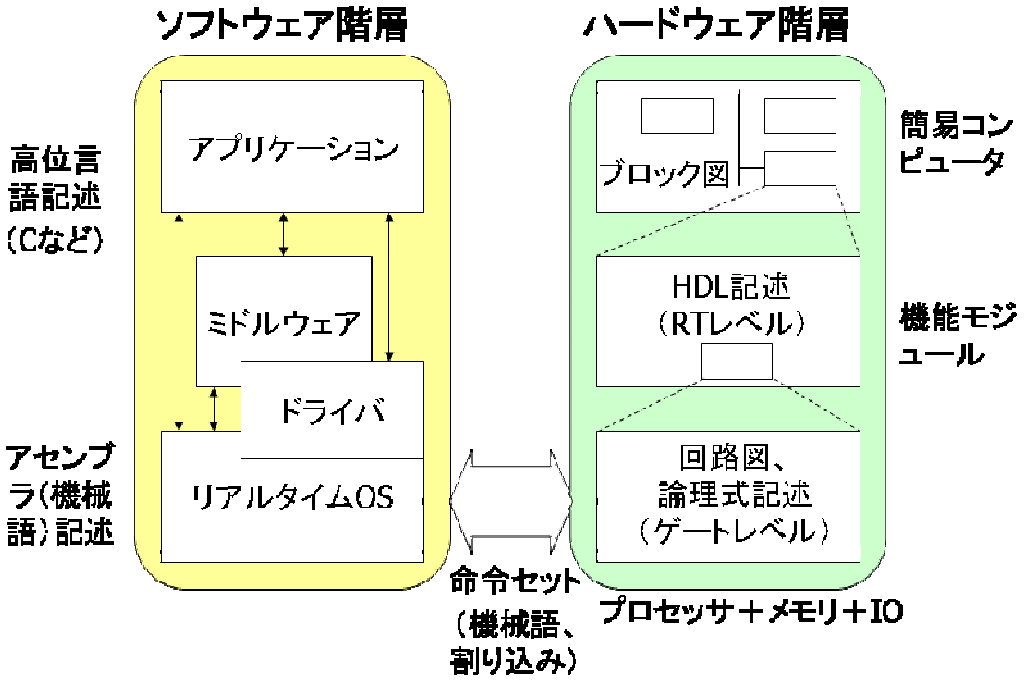


図 4.8　初級コースにおけるソフトウェアとハードウェアの関係

日程	講義内容
1日目	(1)オリエンテーション (2)リアルタイムカーネル入門 (3)ソフトウェア開発環境 (4)μK0の実装(Part1) ・ハードウェアへのアクセス方法 ・Cとアセンブラの結合方法 ・スタートアッププログラムの実装
2日目	(5)μK0の実装(Part2) ・割り込み処理(コンテキスト退避と復帰処理)
3日目	(6)μK1の実装(Part1) ・カーネルデータ定義と初期化 ・タスクスケジューリングとディスパッチ処理
4日目	(7)μK1の実装(Part2) ・システムコールの実装(start、exit)
5日目	(8)タスクプログラミング ・サンプルタスクの記述と動作確認 (9)追加課題(システムコールの拡張)<μK2>

図 4.9　マイクロカーネル制作コースのカリキュラム例

大きく 3 つのレベルに分けて、スパイラルにステップアップしていく手法をとっている。

①μK0：パワーオンリセットから割り込み処理まで。

アセンブラによりハードウェアとの接点を理解する。

また、C 言語とアセンブラの連携を理解する。

②μK1：リアルタイム OS のカーネルの基本部分（タスクスケジューリング処理、ディスパッチ処理、基本システムコール実装）まで。

③μK2：システムコールを含めて基本的な OS 機能を動作させるまで。今回は拡張オプションの扱いで、早く進行した受講生が挑戦できるように設定した。

受講生の評価結果と主なコメント、および、講師考察を表 4.1 に示す。

全体的な評価としては、受講生自身がほぼ一からプログラミングを行い、組込みソフトウェアの核になる部分を理解できたと判断できる。

(2)ハードウェアでの取り組み

ソフトウェアを実行するコンピュータ自体はすっかりブラックボックス化されており、命令セットがどのようなマイクロアーキテクチャで実現されているかをほとんど知らないのが実態である。組込みシステムでは新たなハードウェアを接続することや、低消費電力化のためにアルゴリズムをハードウェア化することも多く、マイクロアーキテクチャの実現方法を理解しておくことが必要である。

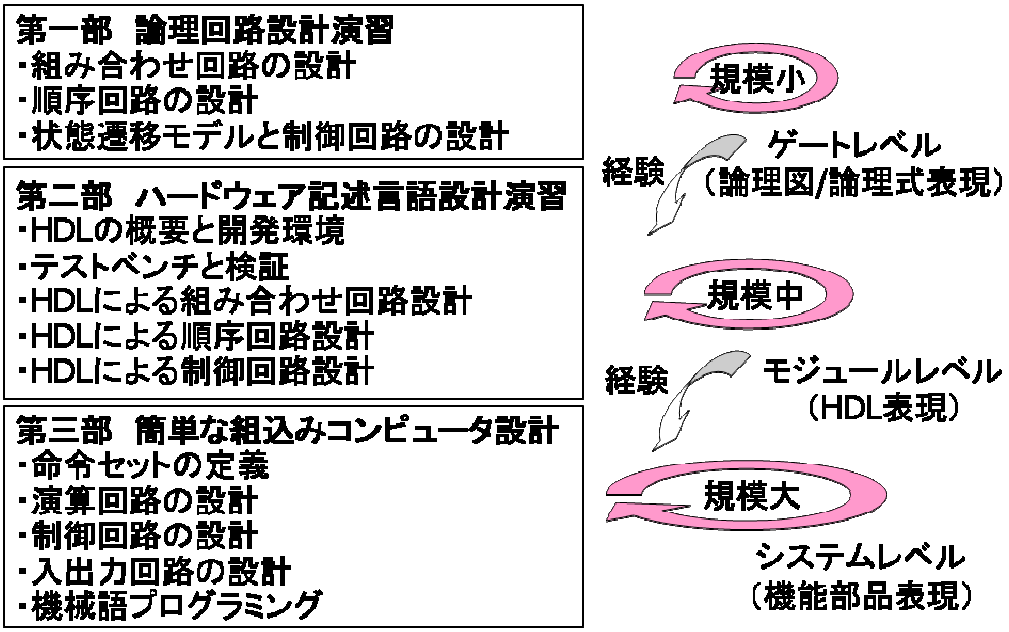


図 4.10　コンピュータ工学演習のカリキュラム例

簡単な命令セットを持つ組込みコンピュータによる簡易電卓を実現するコンピュータ工学演習[サイ 11]（15 単元）のカリキュラム例を図 4.10 に示す。

カリキュラムは 3 部構成とし、第一部で論理回路設計演習として論理式や図式表現（論理図、状態遷移図、タイミングチャートなど）を使って設計を行い、第二部でハードウェア記述言語（HDL）を使って組み合わせ回路、順序回路、制御回路の設計とシミュレーションによる動作確認を行う。第三部で簡単な組込みコンピュータ（情報処理技術者試験のアセンブラ命令セット仕様 COMET[情報 08]のサブセット）を設計し、アセンブラプログラムにより組込みシステムとしての簡易電卓システムを実現する。期間内に演習を終わらせるために、穴埋め方式に近い形で、仕組みの理解に重点を置いている。

受講生の評価結果と主なコメント、および、講師考察を表 4.1 に示す。

このコースは穴埋め方式の演習であり、すべてを自分の力で作成していないので達成感にやや欠ける面があるが、スパイラルアップでレベルを上げていくことにより、ハードウェアとしての仕組みを理解してもらえたと判断している。なお、受講生のレベル差が存在することから、標準課題（普通レベル）と自由課題（上位レベル）に分けることなどが改善項目である。

4.5.3　中級コースでの取り組み

組込みシステムを対象としたアーキテクチャ設計のコースである。組込みシステムのいろいろな実現方法を理解し製品対象のアプリケーションをハードウェアとソフトウェアで最適に実現していく協調設計について講義・演習を通じて学んでいく。

コンピュータアーキテクチャの講義[サイ 11]（15 単元）と演習[サイ 11]（15 単元）のカリキュラムの例をそれぞれ図 4.11、図 4.12 に示す。

講義のカリキュラムは 4 部構成とし、第一部でコンピュータアーキテクチャの基本を理解し、第二部で組込みシステムのものづくりの中で QCD の観点からどのようにアーキテクチャがかかわっているかを議論する。第三部で対象製品に最適になるようにトレードオフを行いながらハードウェアとソフトウェアの組み合わせで実現する協調設計を理解し、第四部でシステムレベルでの設計手法やプロセスを理解する。

演習のカリキュラムは 3 部構成とし、第一部でアーキテクチャ設計演習としてパイプライン制御やキャッシュの性能評価を通じて基本的な仕組みを理解し、第二部でシステムレベル設計演習として C レベル設計言語（SystemC）を使った簡単な設計演習とシミュレーションによる動作の確認を行う。第三部で静止画の圧縮伸張アルゴリズムである JPEG を対象に性能とコストのトレードオフを解決するためのハードウェアとソフトウェアの協調設計手法を机上演習により学ぶ。

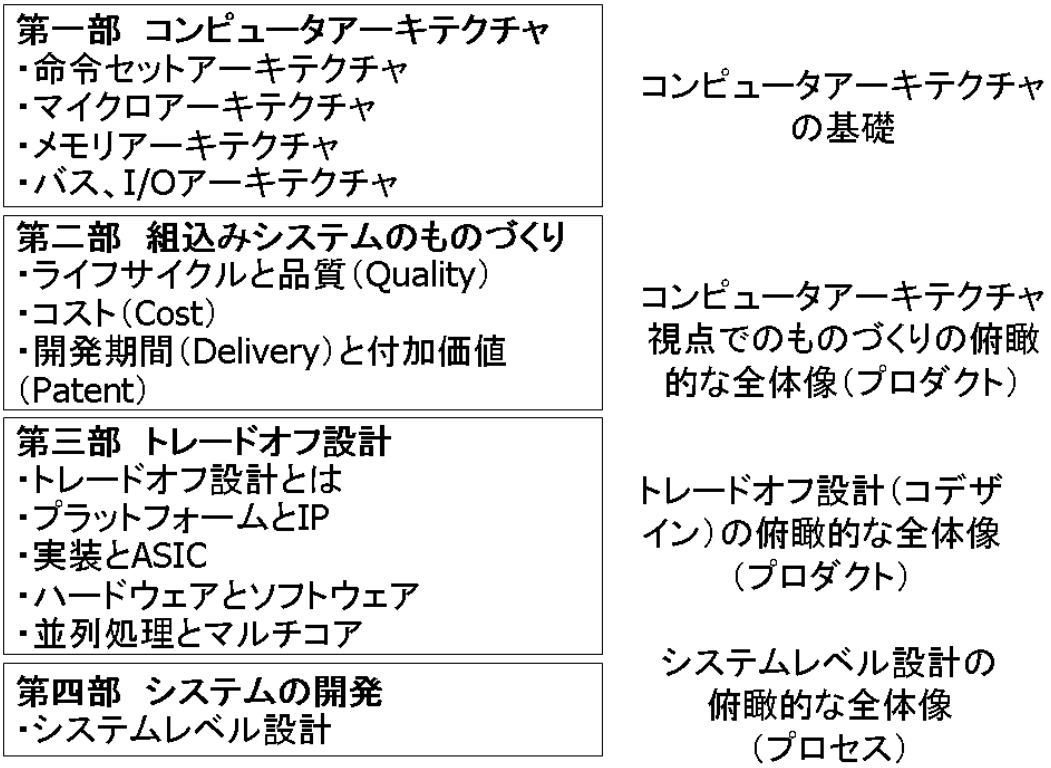


図 4.11　コンピュータアーキテクチャ講義のカリキュラム例

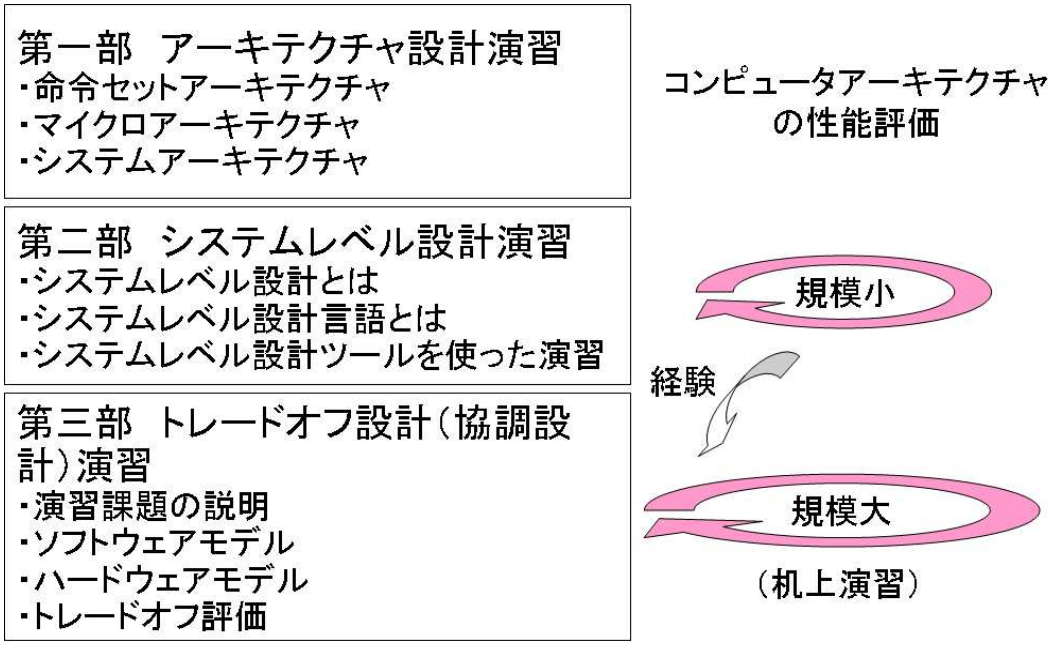


図 4.12　コンピュータアーキテクチャ演習のカリキュラム例

受講生の評価結果と主なコメント、および、講師考察を表 4.1 に示す。

講義では、理解度も満足度も高く、組込みシステムとしてのハードウェアとソフトウェアによる協調設計の重要性とその手法について理解してもらえたと判断している。演習では、演習環境の構築の負荷が大きく、また一部が机上演習にとどまっていることから実感がわきにくい面があり、実データを使って効率よく使える演習環境の構築が今後の改善項目である。

その後、2014 年前期まで改善しながら継続してきた結果、延べの合格者は、コンピュータ工学講義で 67 名、コンピュータ工学演習で 38 名、コンピュータアーキテクチャ講義で 29 名、コンピュータアーキテクチャ演習で 17 名となった。理解度と満足度については、コンピュータアーキテクチャ講義を除いて、表 4.1 の評価点を上回る結果が得られており、受講生に評価されていると判断できる。

また、受講生がどの程度理解したかの客観的な評価として、合格点以上の 4 段階（ABCD）評価の上位 2 つ（80 点以上）の占める割合を調べて見た。コンピュータ工学講義で 85%、コンピュータ工学演習で 87%、コンピュータアーキテクチャ講義で 80%、コンピュータアーキテクチャ演習で 88%であり、全体として 80%以上が理解したと判断できる。

4.6　考察

複合領域である組込みシステムにおける技術者（特にシステムアーキテクト）の育成を目指して、2.1 章の組込みシステムの現状と課題の分析と今まで取り組んできた組込み技術者育成の経験を踏まえて、以下の 4 つの視点からのアプローチに基づいたカリキュラム

表 4.1　受講生の評価結果と主なコメント

	講座名	評価項目	評価点（平均）	主なコメント
入門	コンピュータ工学講義（2009-2010）（延べ15名）	理解度	3.67↑	・組込みシステムに関わる構成や技術に関する基礎をハードウェアとソフトウェアの両面から学べたことや，開発に関する流れについても学習できた．
		満足度	3.86↑	・組込みシステムの難しさが分かりました．内容的に難しいところもあり，もっと基本を学べると良かった．
				<講師考察>ハードウェアとソフトウェアの両面から学べ、開発のプロセスも理解できたとの評価もあり、ある程度目的は達成できていると考えられる。
初級	マイクロカーネル制作演習（2008-2010）（延べ34名）	理解度	3.18→	・μカーネルの概要は知っていたのですが，詳細の動作を実際に作成できたことはよい経験になりました．特にアセンブラでプログラムを作成したことが，良い経験になりました．アセンブラを見る際に，簡易に見ることができるようになりました．
		役立ち度	4.26→	・システムのコア部分を作るためのノウハウが実践で学ぶことができました．
				<講師考察>受講生自身がほぼ一からプログラミングを行い、組込みソフトウェアの核になる部分を理解できたと判断できる。
	コンピュータ工学演習（2009-2010）（延べ11名）	理解度	3.27↑	・コンピュータ工学のレポート作成は大変難しかったですが，大変ためになりました．
		満足度	3.55↑	・ハード周りの実践授業が少ないので，私にとってとても貴重な存在です．内容的には難しい部分もありますが，今のまま続けて欲しいと思います．
				<講師考察>すべてを自分の力で作成していないので達成感にやや欠ける面があるが、スパイラルアップでレベルを上げていくことにより、ハードウェアとしての仕組みを理解してもらえたと判断している。
中級	コンピュータアーキテクチャ講義（2009-2010）（延べ8名）	理解度	4.25↑	・システム設計において，広範なトレードオフ領域から最適化がなされていることが良くわかりました．取り扱われる内容も専門的で，コストパフォーマンスの高い授業です．
		満足度	4.50↑	・システム全体を深く掘り下げられていて良い授業なのですが，反面密度が高くなります．
				<講師考察>組込みシステムとしてのハードウェアとソフトウェアによる協調設計の重要性とその手法について理解してもらえたと判断している。
	コンピュータアーキテクチャ演習（2009-2010）（延べ5名）	理解度	3.40↑	・課題の説明に時間をかけていただければと思いました．
		満足度	3.80↑	・毎回の課題が明確であり，適切な内容だったと思う．
				<講師考察>演習環境の構築の負荷が大きく、また一部が机上演習にとどまっていることから実感がわきにくい面があるが、体系的にトレードオフ設計を体感してもらえたと判断している。

を提案し、実際の授業を通じて評価を行った。

(1) 全体を俯瞰しながら基礎から応用へのスパイラルアップの取り組み

組込みシステム入門コースから、ハードウェアとソフトウェアのそれぞれの初級コースを経て、中級コースの組込みシステム対応のコンピュータアーキテクチャまで、基本から応用に向けて一通り理解しながらレベルを上げていくカリキュラムを実現できている。また、それぞれのコースにおいても、基礎から応用に向けて、スパイラルにアップしていく構成をとっており、受講生の理解度につながっていると考える。

(2) 領域間のインタフェースの両面からの取り組み

プロダクトに関してハードウェアとソフトウェアのインタフェースとそれぞれの働きを初級コースで、ハードウェアとソフトウェアによる協調設計を中級コースで扱うことで、両面からの理解を達成できたと考える。なお、プロセスとプロジェクトに関する両面からのアプローチについては今後の課題である。

(3) ブラックボックス化した技術の階層的なモデリングの取り組み

ソフトウェアに関しては初級コースのマイクロカーネル制作演習（機械語、C 言語）を通じて、ハードウェアに関しては初級コースでの回路レベルと HDL レベルの演習と中級コースでのシステムレベル（SystemC）の演習を通じて、詳細レベルから抽象レベルまで理解することができたと考える。講義系に比べて演習系は理解度と満足度ともに低いが、社会人が多い受講生にレベル差があり、それが影響していると判断している。

(4) ハードウェアとソフトウェアの最適な組み合わせを評価・選択する協調設計の取り組み

中級コースでハードウェアとソフトウェアのトレードオフによる協調設計を講義形式で理解するとともに、静止画圧縮伸張の JPEG エンコーダを題材に机上演習形式で性能・コストのトレードオフを学べるようにカリキュラムを実現し、受講生の評価も得られたと判断している。なお、演習は時間の関係で机上演習にとどまったが、今後は応用分野の拡大に対応しつつ、実験演習コースの充実も必要と考えている。

システムアーキテクトの育成は、まず入門コースでハードウェアとソフトウェアの両面から俯瞰的に基礎的な項目を学び、次の初級コースで、ハードウェアの側面とソフトウェアの側面からそれぞれの基本技術を講義と演習を通じてスパイラルに学び、その後の中級コースで、相反する条件を満足するようにトレードオフを行いながらハードウェアとソフトウェアによる協調設計・検証技術を講義と演習を通じて学ぶことにより、可能となる。最後の中級コースの受講者は少なかったが、理解度と満足度やコメントからほぼ目的を達成できたと判断できる。

4.7 IoT 時代の技術者育成についての考察

IoT 時代を迎え、アプリケーション製品開発は、組込みシステム単体でのプロダクト志向から、通信ネットワークで繋がったクラウドでの分析・評価を含めたサービス指向に移りつつある。また、製品の実現方法も、1990 年代後半から 2000 年代にかけてのシステム LSI による魅力的な製品つくりから、汎用的な CPU や GPGPU を並列に並べてスケールアウトによる性能向上を狙うアプローチや、さらにハードウェアの並列性を活かした FPGA の活用が行われるようになってきた。このように新規開発の姿が変わってきたことから、IoT 時代にふさわしいスキルセットを設定し、それに基づいたカリキュラムよる教育実践が期待されている。

筆者は、電気学会の第 2 次 M2M 技術調査専門委員会[清尾 15]の委員として「M2M/IoT システム入門」[電気 16]の出版に貢献するとともに、その活動成果をベースにサイバー大学（Web ベースの遠隔授業）において 2014 年度より「M2M を理解する」（座学形式）と「M2M を動かす」（演習形式）のゼミナールを開講し、さらに 2017 年度から「IoT 入門」（講義形式 : 受講生 346 名）をスタートさせた。IoT 分野のカリキュラム作成においては、

組込みシステムを通信ネットワークを介してクラウドにつなぎ、そこで評価・分析を行うことから、技術面では、デバイス（センサとアクチュエータを含む)、ゲートウェイ（エッジコンピューティングを含む)、サーバ（クラウドコンピューティング）、通信ネットワーク、統計分析・機械学習、および、セキュリティに、また、ビジネス面からビジネスモデルや応用事例（アプリケーション）に重点をおいている。

これらのカリキュラム作成において、本論文で提案した「スパイラルアップの取り組み」のアプローチについては、IoT システムの構築に適用し、IoT デバイスから IoT ゲートウェイ、IoT サーバへとボトムアップにシステム構築を図っている。「領域のインタフェースの両面からの取り組み」のアプローチについては、API や通信ネットワーク（プロトコル含む）によるインタフェースの取り扱いで考慮している。また、「ブラックボックス化した技術の階層的な取り組み」のアプローチでは、通信の 7 階層モデルに基づいた通信プロトコルの理解に対応させている。「協調設計の取り組み」のアプローチでは、アプリケーションの観点からハードウェア、ソフトウェア、および、通信ネットワークの最適な組み合わせを選択する必要があり、従来の「ハードウェア」と「ソフトウェア」の協調設計を拡張した扱いになる。ポスト「京」における「スーパーコンピュータ開発」と「アプリケーション開発」の協調設計でも示されるように、今後とも重要なアプローチ方法と言える。このように本論文で提案した 4 つのアプローチは、IoT 時代でも有用であると認識している。

IoT 時代では、プロダクト志向からサービス指向にビジネスモデルが大きく変わると予想されており、アイデア次第で従来とは全く違う世界が開かれる可能性があり、如何に価値創造を行うかが求められている。これに対応した「デザイン思考」のアプローチが期待されており、潜在的なユーザニーズに基づいて迅速なプロトタイピングにより価値の評価実証を繰り返す実践的な PBL 手法が重要になってくると考えられる。

4.8 まとめ

組込みシステム技術者教育の課題を解決するために、「全体を俯瞰しながら基礎から応用へのスパイラルアップの取り組み」、「領域間のインタフェースの両面からの取り組み」、「ブラックボックス化した技術の階層的なモデリングの取り組み」、および、「ハードウェアとソフトウェアの最適な組み合わせを評価・選択する協調設計の取り組み」を踏まえて、入門コース、初級コース、および中級コースのカリキュラムを体系的に構築し、高等教育機関で実践評価を行なった。

受講生の評価結果から、受講生の数が必ずしも多くはないが、全体として毎年の授業改善を通じて理解度と満足度は向上してきており、今回の複合領域である組込みシステムに対するアプローチが有効であったと推察できる。また、今回のカリキュラムでは、受講生の一人ひとりが自ら一通り経験することを重視した内容となっており、これが達成出来てはじめて、分担しながらグループで行う PBL 形式の授業が効果を発揮すると認識している。

IoT 時代を迎え、4.7 節で述べたように、IoT システムのカリキュラムを構築し実践しはじめたところであり、今回のアプローチを活かして編成することができたと考えている。

第 5 章　結論

本章では、本研究の成果についてまとめ、今後の課題や取り組みを述べる。なお、研究の対象は、1990 年代後半から 2000 年代にかけての組込みシステムを対象にしているが、IoT 時代と言われる 2017 年の今の視点から見た考察も行う。

5.1　研究の成果

半導体の進歩とデジタル化の進展により、ハードウェア（システム LSI）とソフトウェアから構成される組込みシステム製品は、1990 年代以降あらゆる分野で展開され発展している。独創的なアイデアを取り込んだ組込みシステム製品を開発するためにはハードウェアとソフトウェアで機能を分担して実現する協調設計（コデザイン）が重要になってきている。組込みシステムがますます大規模化・複雑化するに従い、品質を高めるための検証がより重要になり、技術がブラックボックス化する中で協調設計や検証に取り組めるシステムアーキテクトの育成が必要になってきている。本研究ではこれらの課題を解決するために、トップダウン協調検証システムを提案し、マルチメディア系組込みシステムなどに適用し、その効果を実証することができた。また、システムアーキテクトを育成するための体系的なカリキュラムを提案し、実際の授業の中で実践し目標を達成することができた。

本研究での取り組みと成果をまとめると、以下のようになる。

本研究の第 1 章において、研究の背景、組込みシステム開発の課題、関連研究と本研究の位置づけ、および、研究の概要を述べた。

第 2 章において、1990 年代後半から 2000 年代にかけて、応用分野の拡大・適用規模の拡大・システムの複雑化が急速に進みつつある組込みシステム開発の現状と課題について、プロダクト（製品）、プロセス、プロジェクト、ピープルの観点から分析を行った。この分析において、独創的なアイデアを取り込んだ新製品を開発する場合を対象に、設計したものがユーザを満足させる品質であるかを確認するときの検証の課題、および、複合領域にある複雑化した組込みシステムを設計し検証できる技術者の育成の課題を抽出した。また、IoT 時代の組込みシステムの現状と課題についても分析を行なった。

第 3 章において、検証に関して、1990 年代後半から 2000 年代において、設計で再利用した部分も含めてシステム全体を評価する必要があり、大規模化するシステム LSI の不具合による影響が大きいことから、多くの検証の課題が残されていることを示した。

また、競争の激しい分野で独創的な新しい組込みシステムを開発する場合には、設計の上流においてアルゴリズムの評価を行い、ハードウェアとソフトウェアで分担して正しく

動作するかを確認し、実機の近い環境で実動作による確認と人間の感性による評価までを、繰り返し効率よく実行することが必要となる。

このような検証の課題を解決するために、開発の上流でのシステムレベルシミュレーションの検証ステージから、ハードウェアとソフトウェアによる協調シミュレーションの検証ステージ、下流の実機レベルに近い協調エミュレーションの検証ステージまで、各検証ステージで目的とする不具合を取り除きながら階層的に検証を進めていくトップダウン協調検証システムを提案した。

検証ステージ間を渡って繰り返し効率よく検証ができるように、検証環境を統合し、共通テストベンチを適用すると共に、ハードウェアとソフトウェアのインタフェースとしてコンポーネント論理バスアーキテクチャを採用した。

マルチメディア系組込みシステムなど 2 件の事例を対象に、提案したトップダウン協調検証システムを適用した。、システムレベルシミュレーションの検証ステージから、ハードウェアとソフトウェアの協調シミュレーションの検証ステージ、実機に近い環境での協調エミュレーションの検証ステージまで、各検証ステージで実用レベルの性能で目的とする不具合を取り除く検証手法を実現できた。検証環境の統合、共通仕様のテストベンチの適用、コンポーネント論理バスアーキテクチャの採用により、上流から下流まで効率よく検証できることを示した。

第 4 章において、第 2 章で分析した 1990 年代後半から 2000 年代における技術者育成の課題に対応して、ハードウェアとソフトウェアによる最適な分担を実現する協調設計手法を活用して、複合領域である組込みシステム製品の開発を実践できるシステムアーキテクトを育成するための教育手法を提案した。「全体を俯瞰しながら基礎から応用へのスパイラルアップの取り組み」、「領域間のインタフェースの両面からの取り組み」、「ブラックボックス化した技術の階層的なモデリングの取り組み」、および、「ハードウェアとソフトウェアの最適な組み合わせを評価・選択する協調設計の取り組み」を踏まえて、入門コース、初級コース、および中級コースのカリキュラムを体系的に構築した。基本的に、技術者一人ひとりがすべて同じように実践することで必要なスキルを習得することを目標に取り組んだ。高等教育機関（サイバー大学他）において開発したカリキュラムによる授業運営を行い、アンケート調査による授業評価や講師考察などを通して有効であることを確認できた。

IoT 時代を迎え、第 4 次産業革命を実現するための新たな価値創造を目指した IoT システムの構築が求められている。IoT システムは、組込みシステム技術に加え、クラウドコンピューティング技術、通信ネットワーク技術、セキュリティやデータの利活用とプライバシなどの技術が必要となる複合システムである。組込みシステム開発の場合は、ハードウェアとソフトウェアによる協調設計が重要であったが、IoT システム開発の場合には、クラウ

ドコンピューティングや通信ネットワークなども加えて、目標とするアプリケーションを実現するように、構成要素を最適に組み合わせて実現するより広義の協調設計が求められている。ポスト「京」のプロジェクトにおいても「スーパーコンピュータ開発」と「アプリケーション開発」の協調設計の重要性が示されているように、協調設計のアプローチは今後とも重要であると考えられる。

IoT 時代では、組込みシステムというプロダクト価値に加えて、IoT システム全体としてサービス価値を提供するものに進化することが求められている。このような組込みシステムを包含した IoT システム開発において、潜在的なユーザニーズに基づいて迅速にプロトタイプ作成を繰り返しながら価値創造を評価・検証する「デザイン思考」のアプローチが注目されている。今回提案した「トップダウン協調検証手法」は、プロトタイプ作成による価値創造の評価検証として IoT 時代でも有効な手法と考えられる。

IoT システムは、従来の組込みシステム技術に加えて多くの技術が必要とされ、さらに、サービス指向による価値創造が求められる世界でもあることから、技術者育成がますます重要になってきている。筆者は、IoT 時代の技術者育成を目指して、今回提案した教育手法を活用して、IoT 向けのカリキュラムを検討し、2017 年度から本格的に展開をはじめたところであり、今後、受講結果に基づいて改善を図っていく予定である。

5.2 今後の課題と取り組み

独創的なアイデアを取り込んだ組込みシステム製品の開発において、トップダウン協調検証システムを適用することにより、設計の効率化や品質の向上に大きく貢献することができた。しかし、検証については、組込みシステムの大規模化・複雑化が絶え間なく進む中で、如何に不具合を減らすかが相変わらず大きな課題である。

また、技術が成熟し、標準化・水平分業化が進み、技術のブラックボックス化が拡大する中で、組込みシステム製品の付加価値を決めるシステムアーキテクトの重要性はますます重要になってきており、技術者育成についてもさらなる充実が必要である。

IoT 時代を迎え、IoT システムは、組込みシステムに比べ、ますます大規模化・複雑化しており、検証の問題は大きな課題である。価値創造を目指す IoT システムにおいては、迅速なプロトタイピングの繰り返しによる評価検証が期待されている。また、IoT システムは組込みシステム技術に加えて、多くの関連技術を必要とし、新しい価値創造を求められることから、今まで以上にシステムアーキテクトの必要性は高く、IoT システム分野の技術者育成が求められている。

IoT 時代を踏まえて、今回の研究で残された検証と技術者育成の課題と今後の取り組みについて述べる。

5.2.1 検証における今後の課題と取り組み

(1) 静的検証の適用

トップダウン協調検証システムの適用により、組込みシステム製品（特に大規模システム LSI）の開発において、不具合を減らすことができた。しかし、今回適用したシミュレーションやエミュレーションのような動的検証手法では、如何に漏れのないテストケースを準備するかに品質が依存するために、検証工数は膨大になり、しかも不具合の漏れをゼロにできる保証はない。特に、バスインタフェース部分や、複雑な競合条件が重なる部分については、不具合が残る可能性が高い。

このような部分については、正しい仕様を与え、設計したものがその仕様を満足するかを数学的に証明する静的検証（フォーマル検証）を適用することがのぞましい。既に、このような対応が行われつつあるが、より使いやすく実用的な静的検証システムの充実が必要である。

(2) モデルベース開発（MBD：Model Based Development）手法の適用

今回の研究では、設計上流のシステムレベルにおいては主に C 言語でモデル化して評価を行なったが、信号処理系や制御系のアルゴリズム評価では効率が悪い。信号処理系や制御系にふさわしいモデル記述言語を使い、下位の設計ステージに自動的に変換するモデルベース開発（MBD）手法が実用的に使えるようになってきた。設計対象（コントローラ）と制御対象（プラント）をモデル化して、全体を統合して検証を行うシステム統合化検証の充実が必要である。

5.2.2 技術者育成における今後の課題と取り組み

(1) プロジェクト型演習（PBL）への展開

今回提案した教育手法では、個人レベルでの必要な技術やスキルの習得にとどまっており、今後はこの考え方をチームで行うプロジェクト型演習に展開する必要がある。既に他の教育機関で行われているが、今回提案した 4 つの視点に準拠した取り組みが重要であると考えている。

また、遠隔教育（Web ベースの遠隔授業）をベースにした場合には、実施形態として分散 BPL になる。分散 PBL も同じ時間帯でライブで実施する同期型と、時間帯がバラバラになる非同期型がある。同期型の分散 PBL は、従来型の集合 PBL に最新の ICT（テレビ会議システム等）環境を活用することで実践されつつある。非同期型の分散 PBL については、ソフトウェア開発プロジェクトのための共有ウェブサービス（GitHub）を使ったオープンソフトウェアの開発事例に似ているが、目的意識が高く技術レベルの高い専門技術者と違い、技術レベルがバラバラで目的意識も異なる低学年次の学生の場合には、なかなかうまく機能しないのが現状である。高学年次に行なわれるゼミナールにおいては、目的意識が高く、技術レベルも揃ってくるので、非同期型の分散 PBL が可能になると考えられる。なお、非同期型の分散 PBL では、Web 上でビデオ収録による発表と掲示板機能による質疑応答の組み合わせで行なうことを想定している。

(2) 応用ドメイン対応での協調設計教育用題材の充実

応用分野で基本となる題材をベースに、カリキュラムを通して協調設計手法を習得できるような演習教材を充実する必要がある。マルチメディア系で静止画圧縮伸張方式の JPEG について、一部試行し効果的であることがわかったので、今後上流から下流まで一貫して使えるものに充実していく。計測・制御系分野では、倒立振子が適していると認識しており、順次充実していきたい。

(3) 仮想環境＋クラウドコンピューティング環境での組込みシステム教育（仮想実験室）環境の構築[清尾 11B][清尾 14A]

固定した時間や定められた場所での学習が困難な学生や技術者にとって、いつでもどこからでもアクセスできる e ラーニングによる教育は必要である。また、高価な実験機器は個人で所有することは難しく、大学の実験室などのクローズな実験環境をオープンに公開するのも難しい場合に、e ラーニング環境で、遠隔地から高価な実験機器に仮想的にアクセスし実験ができると、大変有用と考えられる。既に仮想実験室の試みは行われているが、まだ理想的な形態にはなっていないと考えられる。今後、仮想実験室環境として理想的な形態を検討し、組込みシステムや IoT システムの研究・開発が行える環境の実現を目指して取り組んでいきたい。

(4) IoT 分野の技術者育成への取り組み[清尾 13A][清尾 15][電気 16]

IoT システムは、4.7 節で述べたように、従来の組込みシステムに必要な技術に加えて、通信ネットワーク技術やクラウドコンピューティング技術やモバイルコンピューティング技術の組み合わせで実現されるまさに複雑な複合システムになってきている。このため、IoT システムに必要な技術を今までと同じやり方で実施することは授業時間の制約で無理があり、どこをブラックボックスとして使い方を教え、どこを内部に踏み込んで仕組みを教えかが重要になる。また、IoT システムの世界では、従来の組込みシステムの特徴であるプロダクト思考のビジネスモデルから、クラウド上でのサービスを含めたサービス指向のビジネスモデルへの変革がイノベーションを実現すると期待されている。その意味で、独創的なアイデアをプロトタイプとして迅速に作成し、実世界で評価を受け、フィードバックにより製品化を目指す「デザイン思考」のアプローチが注目されている。

このような背景から、本研究で提言したアプローチを踏まえて、以下の 2 種類の IoT システム教育のカリキュラムを立ち上げてきたところである。

①IoT システムに必要な基盤技術とビジネスモデルを中心とした講義形式の講座
第一部：IoT の概要、第二部：IoT の市場動向と応用事例、第三部 IoT を実現する基盤技術（デバイス/ゲートウェイ、センサ/アクチュエータ、サーバ/サービス、ネットワーク等）動

向、第四部：IoT の課題（セキュリティ、プライバシ）と展望（ウェアラブルデバイス、ロボット、ドローン/自動運転、AI・機械付学習）の 4 部構成をとり、核となる技術についてはその仕組みも紹介する形をとっている。サイバー大学において 2014 年度からゼミナールとして始め、2017 年度春学期より専門基礎科目「IoT 入門」（15 コマ）を立ち上げて現在実施しているところである。本格的な評価はこれからとなる。

②IoT システムのプロトタイプ構築を中心とした実習形式の講座
IoT システムの身近な例を対象に、各自がそれぞれプロトタイプを構築し評価する実習形式の講座である。プロトタイプの基本系を全員が作成し、発展系については各自の能力に応じて選択して作成することができるように構成している。基本系と発展系共に、第一部：IoT デバイスを動かす、第二部：IoT ゲートウェイにつなぐ、第三部：IoT サーバにつなぐ、第四部：IoT アプリケーションを創るの 4 部構成を取っている。センサやマイコンなどの部品が安価で容易に調達でき、オープン化・標準化の流れの中で開発環境やクラウド環境も安価で容易に利用できるようになってきたことから、個人をベースとした遠隔教育においても適用可能なってきた。サイバー大学において、2014 年度からゼミナール（15 コマ）として現在まで継続して実施している。また、社会人向け講習会では 2 日間コースとして実践している。

現在までの IoT システム教育への取り組みを踏まえて、今後の課題をまとめると以下の通りである。
・技術進歩が激しく、そのままにしておくとすぐ陳腐化してしまう。講義内容をタイムリーに更新していく努力が必要である。
・実際の応用システムでは、IoT システムでセンサからデータを収集するだけではなく、AI（機械学習）などによる分析により、実世界に如何にフィードバックをかけるかが重要になってきている。AI（機械学習）との連携の他に、IoT システムに関連するウェアラブル機器やドローン、自動運転、ロボットなどを含めた実践的な実習課題の充実が必要である。特に分野ごとに典型的なプロトタイプ構築手法を確立し、展開することが必要と考えている。
・現在、実習では個人ごとにすべて同じように取り組ませているので、到達レベルは限られてしまう恐れがある。独創的なアイデアを実現しようとすると、それぞれ得意分野を活かした共同作業が必要になると考えられる。これを実現させる可能性のある非同期型の分散 PBL の取り組みが今後の挑戦課題である。

謝辞

本研究の機会を与えていただき、研究と論文のご指導をいただいた安浦寛人教授に深く感謝いたします。論文をまとめるにあたりご指導いただいた福田晃教授ならびに鵜林尚靖教授に感謝いたします。また、論文作成において貴重なご意見とご指導をいただいた荒木啓二郎教授に感謝いたします。

また、毎年の研究会を通じて議論いただいた村上和彰名誉教授、松永裕介准教授、井上弘士教授、石原亨准教授（現京都大学）に感謝いたします。また、研究会の度に組込みシステム分野の技術者育成について情報交換いただいた簗添明元教授、久住憲嗣准教授に感謝いたします。

また、筆者が三菱電機株式会社在職中から、安浦研究室との共同研究にご支援いただくとともに、現在まで研究ならびに論文作成に対して、貴重なご助言をいただいた、元東京電機大学教授　工学博士小泉寿男氏（元三菱電機株式会社技術本部技師長）に深く感謝いたします。トップダウン協調設計システムの構築において、上司として指導いただいた工学博士鈴木文雄氏に感謝いたします。高位論理合成ツール Tsutsuji を開発し、トップダウン協調設計システム構築に貢献いただいた元三菱電機株式会社客員研究員参事工学博士 J. Barry Shackleford 氏（元ヒューレッド・パッカード株式会社）に感謝いたします。

また、組込みソフトウェアの専門家として、在職中、ならびに、現在まで、いろいろな面でご支援いただいた元豊橋科学技術大学特任准教授　吉田利夫氏（元三菱電機株式会社名古屋製作所 NC システム製造部長、生産システム本部品質保証推進部長）に感謝いたします。

筆者が、本研究に携われたのは、現役時代に設計システム技術センターや人材開発センターの上司や同僚のご支援・ご理解があってのことと、感謝の意を表します。

また、㈱半導体理工学研究センター（STARC）でのシステム LSI 設計教育活動、独立行政法人情報処理推進機構/ソフトウェアエンジニアリングセンター（IPA/SEC）の組込みソフトウェア開発力強化推進委員会（教育部会）での人材育成支援活動、㈱ゼネテックのシステム技術者養成センターでの社会人教育活動、および、サイバー大学のアーキテクチャコースにおける組込みシステム/協調設計の研究・教育活動での経験が大いに役立ちました。一緒に活動に携わった関係者に感謝の意を表します。

最後に、いつも忙しいと言い続けて苦労をかけた妻史子に心から感謝いたします。

2017 年 8 月

参考論文（年代順）

[Koi95] Koizumi, Hisao.; **Seo, Katsuhiko.;** Suzuki, Fumio.; Ohtsuru, Yohsuke.; Yasuura, Hiroto "A Proposal for a Co-design Method in Control Systems Using Combination of Models," IEICE Trans. Inf. & Syst. vol. E78-D, no. 3, March 1995, p. 237-247.

[Seo97] **Seo, Katsuhiko**; Koizumi, Hisao; Shackleford, Barry; Yasuda, Mitsuhiko; Mori, Masashi; Suzuki, Fumio "Top-down Co-Simulation of Hardware/Software Co-Designs for Embedded Systems Based Upon a Component Logical Bus Architecture," IEICE Trans. Vol.E80-A No.10, October 1997, p.1834-1841.

[Seo98] **Seo, Katsuhiko**; Koizumi, Hisao; Shackleford, Barry; Mori, Masashi; Kushara, Takashi: Kimura, Hirotaka "A Method for Design of Embedded Systems for Multimedia Applications," IEICE Trans. Vol.E81-C No.5, May 1998, p.725-732.

[Yas98] Yasuda, Mitsuhiro; **Seo**, **Katsuhiko**; Koizumi, Hisao; Shackleford, Barry; Suzuki, Fumio "A Top-down Hardware/Software Co-Simulation Method for Embedded Systems Based Upon a Component Bus Architecture," Asia and South Pacific Design Automation Conference 1998 (ASP-DAC ' 98). 1998, p.169-175.

[遠藤 00] 遠藤祐; 小泉寿男; **清尾克彦** "ハードウェア・ソフトウェア協調設計方式と ITS 画像処理開発への適用検証," 電学論 D, Vol.120, No.10, 2000, p.1118-1126.

[清尾 06A] **清尾克彦;** 村田裕; 山口義一; 細井真知夫 "三菱電機グループにおけるソフトウェア人材育成の取り組み," 工学教育. Vol.54, No.5, 2006, p.59-64.

[吉田 11A] 吉田利夫; 松本真英; **清尾克彦;** 茅野眞一郎; 杉野栄二; 澤本潤; 小泉寿男 "組込みソフトウェア技術者育成のためのリアルタイムカーネル実装実習プログラムとその評価," 電学論 C. 131 巻 2 号, 2011, p468-479.

[清尾 11A] **清尾克彦**,吉田利夫 "複合領域での教育手法をベースとした組込みシステム技術者教育の実践と評価," 工学教育. Vol59, No.6, 2011, p72-78

[秋山 16] 秋山康智;石原正仁;大江信宏;北上眞二;神戸英利;市村洋;**清尾克彦**;小泉寿男 "文系学生への M2M プロトタイプシステム実装教育カリキュラムの提案と評価," 工学教育. Vol.64, No.1, 2016,p.26-32

参考文献

[Ada96] Adams, J.K.; Thomas, D.E. "The design of mixed hardware/software systems," in Proc. 33rd DAC. 1996, p. 515-520.

[AMB99] AMBLER. T.; BENNETTS B. "Test and the Product Life Cycle," IEEE Design & Test of Computers, July-September 1999, p-21.

[Apt96] Aptix Corporation, "MP4 System Explorer Pro," June 1996.

[Bal03] Balarin, F.; Watanabe, Y.; Hsieh, H.; Lavagno, L. Passerone, C. Sangiovanni-Vincentelli, A. "Metropolis: an integrated electronic system design environment," Computer. April 2003, p.45 – 52.

[Bra04] Braude, Eric J.著, 羽生田栄一監訳. "ソフトウェアエンジニアリング　実践的オブジェクト指向技術体系 Software Engineering: An Object-Oriented Perspective." 翔泳社, 2004.

[Chi94] Chioda, Massimiliano; Giusto, Paolo; Jurecska, Attila; Hsieh, Harry C.; Sangiovanni-Vincentelli, Alberto; Lavagno, Luciano. "Hardware-Software Codesign of Embeddeded Systems," IEEE Micro. August 1994, p.26-36.

[Cul93] Culbertson, W. B.; Osame, T.; Otsuru, Y.; Shackleford, B.; Tanaka, M. "The HP Tsutsuji logic synthesis system," Hewlett-Packard Journal. Aug. 1993, p. 38-51.

[enP17A] enPiT1 の HP,http://www.enpit.jp/master/,参照日：2017-5-5

[enP17B] enPiT2 の HP,http://www.enpit.jp/,参照日：2017-5-5

[Ern98] Ernst, Rolf. "Codesign of Embedded Systems: Status and Trends," IEEE Design & Test of Computers. April-June 1998, p.45-54.

[Gho95] Ghosh, A.; Bershteyn, M.; Casley,R.; Chien, C.; Jain, A.; Lipsie, M.; Tarrodaychik, D.; Yamamoto, O. "A hardware-software co-simulator for embedded system design and debugging," Proceedings of the ASP-DAC '95/CHDL '95/VLSI '95. 1995, p. 155 – 164.

[Gup93] Gupta, R. K.; Micheli, G. De "Hardware-software cosynthesis for digital systems," IEEE Design and Test of Computers. vol. 10, no. 3, Sept. 1993, p. 29-41.

[Hen03] Henkel, Jorg; Hu, Xiaobo Sharon; Bhattacharyya, Shuvra S. "Taking on the Embedded System Design Challenge," IEEE Computer. vol.36, no.4, 2003, p.35-37.

[Hen07] Henzinger, Thomas A.; Sifakis, Joseph "The Discipline of Embedded Systems Design," Computer. October 2007, p.32-40.

[Hol96] Holmann, E.; Yamada, A.; Yoshida, T.; Uramoto, S. "Real-time MPEG-2 software decoding with a dual-issue RISC processor," VLSI Signal Processing IX. Oct. 1996.

[Hua95] Huang I. J.; Despain, A. M. "Synthesis of application specific instruction sets," IEEE Trans. Comput. Aided Des. Integrated Circuits & Syst. vol. 14, no. 6, June

1995, p. 663-675.
[Jac05] Jackson, David Jeff; Caspi, Paul “Embedded Systems Education: Future Directions, Initiatives, and Cooperation,” ACM SIGBED Review - Special issue: The first workshop on embedded system education (WESE). Volume 2 Issue 4, October 2005, p.1-4.
[Joi04] Joint Task Force on Computer Engineering Curricula. “Computer Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering (CE2004),” IEEE Computer Society and ACM. 2004.
http://www.eng.auburn.edu/ece/CCCE/CCCE-FinalReport-2004Dec12.pdf, 参照日：2011-9-23
[Koi95] Koizumi, H.; **Seo, K.;** Suzuki, F.; Ohtsuru, Y.; Yasuura, H. "A Proposal for a Co-Design Method in Control Systems Using Combination of Models," IEICE Trans. Inf. & Syst. vol. E78-D, no. 3, March 1995, p. 237-247.
[Koo05] Koopman, Philip et al. “Undergraduate Embedded System Education at Carnegie Mellon,” ACM Transactions on Embedded Computing Systems. Vol.4, No.3, August 2005, p.500-528.
[Lee03] Lee, Edward A. "Overview of the Ptolemy Project," Technical Memorandum No. UCB/ERL M03/25. University of California, Berkeley, CA, 94720, USA, July 2, 2003.
[Lin96] Lin, B.; Vercauteren, S.; Man, H.D. "Embedded architecture co-synthesis and system integration," Codes/CASHE'96, March 1996, p. 2-9.
[Mic94] Micheli, G. De “Computer-aided hardware-software codesign,” IEEE Micro. vol.14, no.4, 1994.
[Mic95] Micheli, G. De（松永裕介訳）. “ハードウェア／ソフトウェア協調設計のコンピュータ支援における問題および方法について,” 情報処理. vol.36, no.7, 1995
[Mic96] Micheli, G. De; Sami, M. (Ed.) “Hardware/Software Co-Design,” Kluwer Academic Publishers. 1996.
[Mic97] Micheli, Giovanni De; Gupta, Rajesh k. “ Hardware/Software Co-Design,” Proceedings of The IEEE. Vol.85, No.3, March 1997, p.349-365.
[Mic01] Micheli, Giovanni De; Ernst, Rolf; Wolf, Wayne (Editor). “Readings in Hardware/Software Co-Design,” Morgan Kaufmann. 1st edition (June 15, 2001).
[Mit09] Mitsui, Hiroyasu; Kambe, Hidetoshi; Koizumi, Hisao “Use of Student Experiments for Teaching Embedded Software Development Including HW/SW Co-Design,” IEEE TRANSACTIONS ON EDUCATION. VOL.52, No.3, August 2009, p.436-443.
[Pas97] Passerone, C.; Lavagno, L.; Sansoe, C.; Chiodo, M.; Sangiovanni-Vincentelli, A. “Trade-off evaluation in embedded system design via co-simulation,” Proceedings of

ASP-DAC'97. January 1997, p. 291-297.
[Rhi11] Rhines, Walley "Keynote Speech:From Volume to Velocity," DVCon 2011. March 1, 2011.
[Row94] Rowson, J. "Hardware/software co-simulation," Proceedings of the 32nd DAC. June 1994, p. 439-440.
[San05] Sangiovanni-Vincentelli, Alberto Luigi; Pinto, Alessandro "Embedded system education: a new paradigm for engineering schools?," ACM SIGBED Review - Special issue: The first workshop on embedded system education (WESE). Volume 2 Issue 4, October 2005, p.5-14.
[Sat94] Sato, J.; Alomary, A. Y.; Honma, Y.; Nakata, T.; Shiomi, A.; Hikichi, N.; Imai, M. "PEAS-I: A hardware/software codesign system for ASIP development," IEICE Transactions on Fundamentals. vol. E77-A, no. 3, March 1994, p. 483-491.
[Sch96] Schnaider, B.; Yogev, E. "Software development in a hardware simulation environment," in Proc. 33rd DAC. 1996, p. 684-689.
[Sch10] Schaumont, Patrick R. "A Practical Introduction to Hardware/Software Codesign," Springer, 2010.
[Seo97] **Seo, Katsuhiko**; Koizumi, Hisao; Shackleford, Barry; Yasuda, Mitsuhiko; Mori, Masashi; Suzuki, Fumio. "Top-down Co-Simulation of Hardware/Software Co-Designs for Embedded Systems Based Upon a Component Logical Bus Architecture," IEICE Trans. Vol.E80-A No.10, October 1997, p.1834-1841.
[Seo98] **Seo, Katsuhiko**; Koizumi, Hisao; Shackleford, Barry; Mori, Masashi; Kushara, Takashi: Kimura, Hirotaka "A Method for Design of Embedded Systems for Multimedia Applications," IEICE Trans. Vol.E81-C No.5, May 1998, p.725-732
[Sha96] Shackleford, B.; Yasuda, M.; Okushi, E.; Koizumi, H.; Tomiyama, H.; Yasuura, H. "Satsuki: an integrated processor synthesis and compiler generation system," IEICE Trans. Inf. & Syst. vol. E79-D, no. 10, Oct. 1996, p. 1373-1381.
[Sta97] Staunstrup, Jorgen; Wolf, Wayne. Coeditor "Hardware/Software Co-Design: Principles and Practice," Kluwer Academic Publishers. 1997.
[Sys02] SystemC の HP, http://www.accellera.org/downloads/standards/systemc,参照日：2017-7-26
[Ten00] Tennenhouse, David. "Proactive Computing," Communications of the ACM, Vol.43, No.5, 2000.
[Tho93] Thomas, D. E.; Adams, J. K.; Schmit H. "A model and methodology for hardware-software codesign," IEEE Design & Test of Computers. vol. 10, no. 3, 1993. p. 6-15.
[Tsu06] Tsukizoe, Akira; Yasuura, Hiroto; Hisazumi, Kenji; Fukuda, Akira; Hayashida,

Takanori; Nakanishi, Tsuneo “QUBE: A Practical Education Program,” Proceedings of the 2006 Workshop on Embedded Systems Education, October 2006, p.4-9.
[Vah03] Vahid, Frank “The Softening of Hardware, ” Computer. April 2003, p.27-34.
[Wol94] Wolf, Wayne H. “Hardware-Software Co-Design of Embedded Systems,”, Proceedings of The IEEE. Vol.82, No.7, July 1994, p.967-986.
[Wol00] Wolf, Wayne; Madsen, Jan “Embedded Systems Education for the Future,” Proceedings of the IEEE. Vol.88, No.1, January 2000, p23-30.
[Wol03] Wolf, Wayne “A decade of hardware/software codesign,” IEEE Computer. vol.36, no.4, 2003, p.38-43.
[Wol07] Wolf, Wayne “The Embedded Systems Landscape,” Computer. October 2007, p.29-31.
[Wol09] Wolf, Wayne 著, 安浦寛人監訳. “組込みシステム設計の基礎 Computers as Components-Principles of Embedded Computer System Design,” 日経 BP 社. 2009.
[Yam05] Yamamoto, M.; Tomiyama, H.; Takada, H.; Agusa, K.; Mase K.,; Kawaguchi, N.; Honda, S.; Kaneko, N. “NEXCESS: Nagoya University Extension Courses for Embedded Software Specialists,” Proceedings of the 2005 Workshop on Embedded Systems Education, September 2005. p. 16-20.
[YaM98] Yasuda, Mitsuhiro; **Seo**, **Katsuhiko**; Koizumi, Hisao; Shackleford, Barry; Suzuki, Fumio “A Top-down Hardware/Software Co-Simulation Method for Embedded Systems Based Upon a Component Bus Architecture,” Asia and South Pacific Design Automation Conference 1998 (ASP-DAC ’98). 1998.
[Yas95] Yasuura, H.; Nakamura, S.; Tomiyama, H.; Akaboshi, H. "Hardware-software codesign with soft-core processor," SASIMI '95. Aug. 1995, p. 79-84.
[Yas98] Yasuura, H; Tomiyama, H; Inoue, A; Fajar, Eko. “Embedded System Design Using Soft-Core Processor and Valen-C," Journal of Information Science and Engineering, No.14, September 1998, p.587-603.
[Yas06] Yasuura, Hiroto “Embedded System Education in the e-Living and SoC Era,” 6th ACM & IEEE Conference on Embedded Software (EMSOFT'06). October 2006.
[ZUK96] ZUKEN Incorporated, Tsutsuji Reference Manual, 1996.
[秋山 16] 秋山康智;石原正仁;大江信宏;北上眞二;神戸英利;市村洋;**清尾克彦**;小泉寿男 “文系学生への M2M プロトタイプシステム実装教育カリキュラムの提案と評価,” 工学教育. Vol.64, No.1, 2016,p.26-32.
[石崎 10] 石崎繁利; 尾崎純一; 齋藤茂; 中辻武; 英崇夫 “分担方式によるものづくり教育の試み,”工学教育. Vol.58, No.2, 2010, p.64-69.
[石田 12] 石田利永子;山本雅基;海上智昭;森孝夫;本田晋也;一場利幸;高瀬英希;高田広章 “共同研究と公開講座による組込みソフトウェア技術者育成の取り組み,” 工学教育. Vol.60,

No.3, 2012,p.75-81.
[今井 95] 今井正治 “ハードウェアの見積りと生成, ” 情報処理. vol.36, no.7, 1995.
[今井 98] 今井正治; 武内良典 “ハードウェア／ソフトウェア・コデザイン手法,” 電子情報通信学会誌. vol.81, no.11, 1998.
[今井 02] 今井 正治; 武内 良典; 塩見 彰睦; 佐藤 淳; 北嶋 暁 “特定用途向きプロセッサ開発システム ASIP Meister,” 電子情報通信学会技術研究報告, DSP2002-125, 2002, p. 39-44.
[今井 04] 今井正治 “システムレベルデザインに向けて,” 情報処理. vol.45, no.5, 2004, p.451-455.
[大原 08] 大原茂之 “コンピュータエンジニアリング領域（J07-CE）,” 情報処理. Vol.49, No.7, July 2008, p.750-758.
[筧 08] 筧捷彦 “情報専門学科カリキュラム標準 J07 について,” 情報処理. Vol.49, No.7, July 2008, p.721-727.
[経産 07] 経産省, 独立行政法人 情報処理推進機構 “2007 年版 組込みソフトウェア産業実態調査,” https://sec.ipa.go.jp/download/200706es.php, 参照日：2011-10-31.
[黒川 12] 黒川利明 “大学・大学院におけるデザイン思考（Design Thinking）教育,” 科学技術動向, 2012 年 9・10 月号, pp.11-23.
[黒坂 04A] 黒坂均; 荒木大 編集 “特集 システムレベルデザイン,” 情報処理. vol.45, no.5, 2004, pp.449-449.
[黒坂 04B] 黒坂均; 竹村和祥; 橘昌良 “システムレベル設計フローと設計言語,” 情報処理. vol.45, no.5, 2004, pp.456-463.
[経済 07] 経済産業省 組込みソフトウェア開発力強化推進委員会 組込みスキル標準領域教育部会編：「平成 18 年度活動報告書」,2007,
http://sec.ipa.go.jp/reports/20070709/ETSS2007_report1_20070709.pdf, 参照日：2011-6-5
[小池 11] 小池豊 “システム LSI 技術者向け教育活動,” EDSFair2011 ブース内セミナー資料, Nov. 2011.
http://www.starc.jp/download/edsf2011nov/booth-koike.pdf,参照日：2011-12-25
[小池 12] 小池豊;鴨野豊;土屋主税 “システム LSI 設計技術者向け教育活動,” 工学教育. Vol.60, N0.3, 2012,p.63-69.
[小澤 03] 小澤時典; 橘詰恒雄 “STARC における SoC 設計教育支援の現状,” 工学教育. 51-3, 2003, p.28-33.
[サイ 11] サイバー大学 HP 担当科目,
http://www.cyber-u.ac.jp/faculty/it/teacher/seo_katsuhiko.html,参照日：2011-6-5
[産業 07] 産業構造審議会情報経済分科会, 第１１回情報サービス・ソフトウェア小委員会 “イノベーションの促進について,” 平成 19 年 4 月 13 日,

http://www.meti.go.jp/committee/materials/downloadfiles/g70419b03j.pdf, 参照日：2011-12-25
[情報 07A] 情報処理推進機構, 独立行政法人 ソフトウェアエンジニアリングセンター監修:「組込み技術者教育ベストプラクティス集」,2007, http://sec.ipa.go.jp/reports/20070607/ebtrainbp20070612.pdf,参照日：2011-6-5
[情報 07B] 情報処理推進機構, 独立行政法人 ソフトウェア・エンジニアリング・センター編 “組込みソフトウェア向け開発プロセスガイド,” 翔泳社, 2007.
[情報 08] 情報処理推進機構編,独立行政法人:「情報処理技術者試験 Ver1.0 別紙 1 アセンブラ言語仕様」, 平成 20 年 10 月, http://www.jitec.jp/1_00topic/topic_20081027_hani_yougo.pdf, 参照日：2011-6-5
[情報 09A] 情報処理推進機構, 独立行政法人 ソフトウェア・エンジニアリング・センター編 “新版 組込みスキル標準 ETSS 概説書,” 翔泳社, 2009.
[情報 09B] 情報処理推進機構, 独立行政法人 ソフトウェア・エンジニアリング・センター編 “組込みスキル標準 ETSS 教育プログラムデザインガイド,” 翔泳社, 2009.
[清尾 99A] **清尾克彦** “巻頭論文 設計プロセスの革新,” 三菱電機技報.VOL.73, No.9, 1999. http://www.mitsubishielectric.co.jp/giho/9909/9909102.pdf,参照日：2017-7-26
[清尾 06A] **清尾克彦;** 村田裕; 山口義一; 細井真知夫 “三菱電機グループにおけるソフトウェア人材育成の取り組み,” 工学教育. Vol.54, No.5, 2006, p.59-64.
[清尾 11A] **清尾克彦**,吉田利夫“複合領域での教育手法をベースとした組込みシステム技術者教育の実践と評価,” 工学教育. Vol59, No.6, 2011, p72-78.
[清尾 11B] **清尾克彦** “組込みシステム教育における仮想実験環境の検討,” 平成 23 年度電気学会 C 部門富山大会, p.760-761, 2011.
[清尾 13A] **清尾克彦** “M2M（Machine to Machine）技術の動向と応用事例,” サイバー大学紀要第 5 号, p.1-22. , 2013.
[清尾 14A] **清尾克彦** “仮想実験環境による組込みシステム教育演習の取り組み,”サイバー大学 e ラーニング研究第 3 号,2014.
[清尾 15] **清尾克彦**, 辻秀一,三井浩康,中西美一 “スマート社会を支える M2M システム技術の最新動向-M2M システム構築技術-,” 平成 27 年電気学会全国大会,Vol.3,3-S14(11)-(14),2015.
[ゼネ 07] ゼネテック 株式会社：「ゼネテック システム技術者養成センター 組込みエンジニアコース（入門,初級,実践,プロジェクト）」,組込み技術者教育ベストプラクティス集,IPA,pp.67-68,2007,http://sec.ipa.go.jp/reports/20070607/ebtrainbp20070612.pdf, 参照日：2011-6-5
[田中 11] 田中基夫, WishBank Project, http://www.methodologylab.com/Design_Methodology_Lab/WishBank.html（2011.10.29 参照）, 現在、HP 社より開発・再配布ライセンスを受けて公開.

[簗添 06] 簗添明; 林田隆則; 安浦寛人; 平川和之; 伊藤文章; 村上貴志; 久住憲嗣; 中西恒夫; 福田晃 “シリコンシーベルト福岡のシステム LSI 設計人材育成ー社会人教育,” 工学教育.Vol.54, No.5, 2006, p.38-42.
[簗添 10] 簗添明 “人材養成：QUBE 実践成果とさらなる発展方向,” システム LSI 設計人材養成シンポジウムーQUBE(2005～2009 年度)成果報告,（2010.3.5）.
[電気 16] 電気学会第 2 次 M2M 技術調査専門委員会編 “M2M/IoT システム入門,” 森北出版株式会社,2016/3/31.
[名古 09] 名古屋大学大学院 情報科学研究科附属組込みシステム研究センター “組込みソフトウェア技術者人材養成プログラム,” 研究成果報告書,第 1 巻, 2009, p.217-225.
[野々04] 野々垣直浩 “ハードウェア／ソフトウェア協調シミュレーション技術,” 情報処理. vol.45, no.5, 2004, pp.484-491.
[橋詰 06] 橋詰恒雄; 有賀正憲; 加沼安喜良; 今村健 “大学での SoC 設計推進へ向けた STARC の取り組み,” 工学教育. Vol.54, No.5, 2006, p.31-37.
[林田 08] 林田隆則; 久住 憲嗣; 簗添 明; 中西恒夫; 福田 晃; 安浦寛人 “設計技術教育カリキュラムの体系化および評価のためのシステム LSI 設計技術スキルマップの策定,” DA シンポジウム 2008. 2008.
[平川 12] 平川和之;笹尾勤;福田晃;伊藤文章 “福岡システム LSI カレッジにおける企業技術者教育,” 工学教育. Vol.60, No.3, 2012,p.70-74.
[ポスト 17] ポスト「京」・フラッグシップ 2020 プロジェクトの HP, http://www.aics.riken.jp/jp/post-k/project,参照日：2017-5-6
[マイ 15] マイケル.E.ポータ他 “IoT 時代の競争戦略,” Harvard Business Review,2015 年 4 月号,ダイヤモンド社.
[宮崎 95] 宮崎敏明 “信号処理分野におけるコデザイン,” 情報処理. vol.36, no.7, 1995.
[安浦 95] 安浦寛人 “基本ソフトウェアとコデザイン,” 情報処理. vol.36, no.7, 1995.
[安田 98] 安田光宏; **清尾克彦** “コンポーネント論理バスアーキテクチャーに基づくトップダウン コ・シミュレーション手法の提案,” DA シンポジウム '98. 1998.
[安田 99] 安田光宏;鍋田芳則;楠原崇史 “システムレベル協調設計・検証技術ー組み込みソフトウェアとハードウェアの協調設計検証ー,” 三菱電機技報.VOL.73, No.9, 1999.
[山本 06] 山本雅基; 阿草清滋; 間瀬健二; 高田広章; 河口信夫; 冨山宏之; 本田晋也; 金子伸幸 “NEXCESS：社会人組込みソフトウェア技術者教育におけるスキル育成,” 工学教育.Vol.54, No.5, 2006, p.49-54.
[山本 12] 山本雅基;海上智昭;塩谷敦子;森孝夫;高田広章 “組込みソフトウェア開発管理者の育成スキルを高める教育の取り組み,” 工学教育. Vol.60, No.3, 2012,p.82-85.
[吉田 11A] 吉田利夫; 松本真英; **清尾克彦;** 茅野眞一郎; 杉野栄二; 澤本潤; 小泉寿男 “組込みソフトウェア技術者育成のためのリアルタイムカーネル実装実習プログラムとその評価,” 電学論 C. 131 巻 2 号, 2011, p468-479.

[渡辺 06] 渡辺登“組込みスキル標準（ETSS）の概要,”工学教育.Vol.54, No.5,2006, p.97-102.

用語集

用語	説明
A/D 変換	アナログ信号をデジタル信号に変換すること。
API（Application Program Interface）	あるコンピュータプログラム(ソフトウェア)の機能や管理するデータなどを、外部の他のプログラムから呼び出して利用するための手順やデータ形式などを定めた規約のこと。
Aptix 社	システム LSI などの電子システムを検証するエミュレータを開発・販売する CAD ベンダ（米国、1989 年設立)。現在は存在しない。
ASAP（Application Specific Adaptable Processor）	レジスタ数やビット長を可変にできるアプリケーション適応型のプロセッサ（九州大学安浦研究室と三菱電機株式会社との共同研究の成果物)。
ASIC（Application Specific Integrated Circuit）	半導体集積回路(IC：Integrated Circuit)の分類の一つで、ある特定の機器や用途のために、必要な機能を組み合わせて設計、製造されるもの。
COMET	基本情報技術者試験のために定義されたアセンブラ言語を実行する最低限の機能のみを実装した 16 ビット固定長の架空の電子計算機。
CPLD（Complex Programmable Logic Device）	内部の論理回路の構造を何度も繰り返し再構成できる半導体チップ(PLD：Programmable Logic Device)の一種で、特にゲートの規模が大きく、複数の論理ブロックを複合した構造を持ったデバイス。
CPU（Central Processing Unit）	中央処理装置。
enPiT1	分野・地域を超えた実践的情報教育協働ネットワーク（大学院生向け）プロジェクト（2012 年度～2016 年度)。
enPiT2	成長分野を支える情報技術人材の育成拠点の形成（学部生向け）プロジェクト（2016 年度～2020 年度)。
ETSS（Embedded Technology Skill Standard）	IPA の SEC において 2005 年に制定された組込み技術スキル標準。ETSS は、スキル標準、キャリア標準、教育研修標準から構成されている。
e ラーニング	情報技術を用いて行なう学習。
FPGA（Field Programmable Gate	内部の論理回路の構造を何度も繰り返し再構成できる半導体チップ(PLD：Programmable Logic Device)の一種で、回路規

Array)	模が数万ゲート以上に及ぶ大規模で複雑なデバイス。
FPIC (Field Programmable Interconnect Component)	内部の相互接続を何度も繰り返し再構成できる半導体デバイス。Aptix 社のエミュレータで使用されていた。
GPGPU (General Purpose Graphic Processing Unit)	画像処理を高速に実行する GPU (Graphic Processing Unit) の機能を、汎用的な並列計算用途に転用したもの。画像認識や機械学習などの高速並列処理に使われている。
HDL (Hardware Description Language)	ハードウェア記述言語。集積回路を設計するためのコンピュータ言語である。
IoT (Internet of Things)	「モノのインターネット」と言われ、センサやデバイスといった「モノ」がインターネットを通じてクラウド（サーバ）に接続され、収集された情報を分析した結果に基づいて、実世界を制御する仕組み。
IP (Intellectual Property)	再利用可能な設計資産を意味する半導体分野の用語。ソフトバンクが買収した ARM 社の ARM コアは IP の代表格である。
IPA (Information-technology Promotion Agency)	独立行政法人　情報処理推進機構。
JPEG	静止画圧縮伸張アルゴリズムの国際標準規格の１つ。
LDF	Tsutsuji のハードウェア記述言語。
LPWA (Low Power Wide Area)	低消費電力、低ビットレート、広域カバレッジを特徴とする無線ネットワークの総称。
LSI (Large Scale Integrationt)	多数のトランジスタやダイオード、抵抗、コンデンサなどの電子部品（素子）を、一つの半導体チップに組み込んだ集積回路。
M2M (Machine-to-Machine)	機器同士がネットワークで接続され、相互に情報のやりとりを行うことで、人手を介さずに情報収集や管理・制御を実現する仕組み。現在は、IoT の一部として扱われることが多い。
M32R/D	三菱電機株式会社製の DRAM 内蔵 RISC プロセッサ。
Matlab	米国 Mathwork 社が提供する数値解析ソフトウェア。
NEXCESS	名古屋大学において 2004 年度から 2008 年度の 5 年間にわたり開設された「組込みソフトウェア技術者人材養成プログラム」。
PBL (Project Based Learning、	Project Based Learning：具体的な学習課題を立てて少人数グループでプロジェクトを完遂させる「プ ロジェクトにもと

Problem Based Learning)	づく学習」のこと。Problem Based Learning は、学習者自身が中心となり、反省的反復の作業をともないながら、実践される少人数グループの 教育手法こと。
PDCA （Plan, Do, Check, Act）	品質管理で使われる Plan（計画）、Do（実行）、Check（評価）、Act（改善）の 4 段階のサイクルの頭文字をとったもの。この 4 段階を順次行って 1 周したら、最後の Act を次の PDCA サイクルにつなげ、螺旋を描くように 1 周ごとに各段階のレベルを向上させて、継続的に業務を改善していくこと。
PMBOK （Project Mangement Body Of Knowledge）	プロジェクトマネジメントを行なう時に必要な知識を体系的にまとめたもの。
QCD （ Quality, Cost, Delivery）	モノづくりで重要となる品質（Quality）とコスト（Cost）と納期（Delivery）の頭文字をとったもの。
QUBE	九州大学において 2005 年度から 2009 年度の 5 年間にわたり開設された「システム LSI 設計人材育成実践プログラム」
RISC （Reduced Instruction Set Computer）	コンピュータの命令セットアーキテクチャ(ISA：Instruction Set Architecture)の設計手法の一つで、縮小命令セットコンピュータと呼ばれる。命令の種類を減らし、回路を単純化して演算速度の向上を図っている。
RTL （ Register Transfer Level）	論理回路の記述において、レジスタからレジスタへの転送と、その間の論理演算の組み合わせとして記述した抽象度のレベル。
RTL モデル	RT レベルの抽象度で記述されたモデル
SEC （ Software Engineering Center）	情報処理推進機構（IPA）の組織の一部として、ソフトウェアエンジニアリングセンターと呼ばれていたが、現在はソフトウェア高信頼化センター（Software Reliability Enhancement Center）に名称が変更されている。
SiP （System in Package）	複数の半導体チップを 1 つのパッケージに集積したもの。
SoC （System on Chip）	システムを構成する機能を 1 チップの LSI で実現したもの。本研究ではシステム LSI と同義語として使っている。
SoS （System of Systems）	複数のシステムから構成され機能するシステムのこと。構成要素である各システムは独立した異なるシステムであり、個別に管理・運用される。
STARC	半導体業界各社が協力して 1995 年度に設立した「㈱半導体

（Semiconductor Technology Academic Research Center）	理工学研究センター」で、大学の研究活動を支援するとともに、システム LSI 設計技術者の教育を推進してきた。2015 年度末で解散した。
Statemate	米国 IBM 社（元 Telelogic 社）が提供する構造化分析ベースモデル駆動型開発ツール。
SystemC	電子回路機器の機能設計への使用を目的としたハードウェア記述言語（HDL）の一種で、プログラム言語である C++のクラスライブラリとして提供されている。2011 年に IEEE 1666 規格に登録された。
Tsutsuji	1990 年代に米国 HP 社が開発した設計・検証ツールで、日本の㈱図研が Vps として販売。ハードウェア記述言語（LDF）、ブロック図、C 言語動作モデルを組み合わせて高速にシミュレーションを実行できる。また、仮想計測器環境によりビジブルなテストベンチを提供することができる。
UML（Unified Modeling Language）	統一モデリング言語と呼ばれ、主にオブジェクト指向分析や設計のための、記法の統一がはかられたモデリング言語である。
Verilog	1984 年に登場したデジタル回路の設計用の論理シミュレータであり、そこで使用するハードウェア記述言語でもある。1995 年に IEEE で規格化された。
VHDL	1981 年に登場したデジタル回路の設計用のハードウェア記述言語の一種である。もともとは米国国防総省が納品する機器に含まれる ASIC の動作記述のために開発したものである。1987 年に IEEE で規格化された。
Vps（Virtual Prototyping System）	㈱図研から販売された Tsutsuji の商品名。
アーキテクチャ設計	システムに要求されている機能・性能を、システムを構成する要素に配分して構成要素の仕様を明確にするとともに、構成要素間のインタフェースを明確化すること。この段階でハードウェアとソフトウェアの分割が行なわれる。
アクセスネットワーク	ユーザの通信機器をインターネットプロバイダに接続（アクセス）するためのネットワーク。無線系の携帯電話サービスや有線系の光通信網などの広域ネットワークが該当する。
アクチュエータ	入力されたエネルギーを物理運動に変換する装置。
インタフェース合成	ハードウェアとソフトウェアのインタフェースは、ハードウ

（Interface Synthesis）	ェア側のインタフェーイス回路部分とソフトウェアのドライバプログラム部分から構成されており、これらの部分を生成することを指す。
エッジ	IoT システムにおいて、エッジとは、クラウドに対して、ユーザに近い側を指す。
エッジコンピューティング	クラウドに対して、ユーザ側に近い閉じられた世界で行なわれるコンピュータ処理。通信遅延を短縮し、リアルタイムでセキュアなサービスを提供する。
エミュレーション	ある装置やソフトウェア、システムの挙動を別の装置やソフトウェアなどによって模倣し、代替として動作させること。
エミュレータ	エミュレーションを行なう装置またはソフトウェア。
エリアネットワーク	限られた領域内で相互接続されるネットワークで、無線系の Wi-Fi、Bluetooth、ZigBee などや、有線系の Ethernet、PLC などの狭域ネットワークが該当する。
カーネル	オペレーティングシステム（OS）の中核となる部分で、動作中のプログラムの実行状態を管理したり、ハードウェア資源を管理してアプリケーションソフトがハードウェアの機能を利用する手段を提供したりする。
クラウド	インターネットなどのネットワークに接続されたコンピュータ（サーバ）が提供するサービスを、利用者がネットワーク経由で利用するコンピューターの利用形態の一つ。
クラウドコンピューティング	インターネットなどのネットワークに接続されたコンピュータ上で行なわれる計算処理。
ゲートウェイ	センサやアクチュエータが接続されたデバイス（機器）とクラウド（インターネット上のサーバ）との間に存在し、デバイスとはエリアネットワークで、クラウドとはアクセスネットワークで接続され、、データの集約・中継やプロトコル変換などを行なう装置。
ゲートモデル	AND や OR などのゲートレベルの抽象度で記述された動作記述モデル。
コントローラモデル	制御システムにおける設計対象の機能を Matlab などのモデル記述言語で記述したもの。
コンポーネント論理バス（Component Logical Bus）	ハードウェアとソフトウェアのインタフェースとして、メモリマップド IO 方式のアーキテクチャを適用したバスにつけた名称。本研究において、設計の上流の抽象度の高い検証ステージから、実機に近い環境での検証ステージまで統一したイ

	ンタフェースとして適用した。
サービス指向	ソフトウェアの機能をサービスと見立て、そのサービスをネットワーク上で連携させてシステムの全体を構築していこうとするアプローチ。
システム	複数の機能が集まって相互に関係しながら全体でまとまった機能を実現している存在。
システム LSI （System LSI）	CPU、メモリ、ロジック、ソフトウェアを構成要素として複数の機能をワンチップ上に集積し、応用システムの主要な機能を実現し制御する LSI。本研究では、SoC と同義語で扱っている。
システム LSI 設計人材育成実践プログラム	QUBE を参照。
システムアーキテクト	本研究では、組込みシステムの開発に必要となる要件を定義し、それを実現するためのアーキテクチャを設計する者を指す。
システムエミュレータ	組込みシステム全体を対象にできるエミュレータで、Aptix 社の MP4 の商品名に使われている。
システムコール	オペレーティングシステム（OS）の機能を呼び出すために使用される機構のこと。
システム設計	システムに対する要求仕様を、システムとして実現するために，システムが満たしていなければならない条件などをシステム仕様にまとめること。この段階ではハードウェアとソフトウェアの分割はまだ行なわれていない。
システムモデル	システムの機能を抽象度の高いレベルで表現した動作記述モデル。
システムレベル	システム全体の機能を対象に扱う段階を指す。ハードウェアとソフトウェアとを区別なく、ソフトウェアの記述と同等の抽象度で論理システム全体を記述している。
システムレベルシミュレーション	システム設計の段階で、システム全体の機能を評価するシミュレーション。
システムレベル設計	開発プロセスの上流に位置づけられるシステム設計とアーキテクチャ設計を包含したものとして使われている。
シミュレーション	実物の代わりに、コンピュータ上に対象物の動作を記述したモデルを作って実験を行なうこと。
シングルプロセス	１つのアドレス空間でプログラムを実行する仕組み。
ステークホルダ	プロジェクトにおいて直接・間接的な利害関係を有する者。

センサ	物理的な変化、あるいは、化学的な変化などを検出して電気信号に変換する素子。
ソフトウェアエンジニアリングセンター	SEC を参照。
ターゲットプロセッサ	実行時に使われるプロセッサ。
タスクスケジューリング	実行単位であるタスクの実行順序を制御すること。
ディスパッチ	複数のプログラムを実行中のマルチタスクオペレーティングシステムにおいて、プログラムに実行権を渡すこと。
デザイン思考	米国のデザインコンサルティングファーム IDEO（アイディオ）社が提唱し、スタンフォード大学 d.school（2005 年創設）を中心に実践されている新たな市場創造/顧客創造のアプローチの一つ。共感（Empathize）→ 問題定義（Define）→ アイデア創出（Ideate）→ プロトタイピング（Prototyping）→ 検証（Test）の 5 つのプロセスを高速に何度も回すことが特長である。
ドライバ	コンピュータに接続した機器を制御・操作するための専用のソフトウェア。
トレードオフ	一方を追求すれば他方を犠牲にせざるを得ないという二律背反の状態・関係のこと。
ビジネスモデル	事業で収益を上げるための仕組み。事業として何を行ない、ターゲットは誰で、どのようにして利益を上げるのか、という「儲け」を生み出すための具体的なシステムのこと。
ビッグデータ	巨大で複雑なデータ集合の集積物。
ブラックボックス	内部の動作原理や構造を理解していなくても、外部から見た機能や使い方のみを知っていれば利用できる装置や機構の概念。
プラットフォーム（Platform）	基本となるプロセッサやバスなどのハードウェア構成を定め、その上で動作する組込み OS などの基本的なソフトウェア構成を定めたもの。
プラットフォームベース設計	応用分野に適したプラットフォームをベースに類似システムを効率良く設計する手法。
プラントモデル	制御システムにおける制御対象の動作を Matlab などのモデル記述言語で記述したもの。
プロトタイピング	プロトタイプを作ること。
プロトタイプ	システム開発の初期の段階で、動作確認用として作成する試

	作品。
プロファイリング	システム機能としてどの程度のデータ演算量やトラフィック量が必要かを、実際のアプリケーションでの使用状況に基づいて解析すること。
ポーリング方式	通信やソフトウェアにおいて、複数の機器やプログラムに対して順番に定期的に問い合わせを行ない、一定の条件を満たした時に処理を行なう方式。
ポスト「京」	2014 年度から開始されたスーパーコンピュータ「京」の後継プロジェクト。「スーパーコンピュータ開発」と「アプリケーション開発」の協調設計が行なわれている。
マイクロアーキテクチャ	用語の生まれた当初はマイクロプログラム方式におけるコンピュータ・アーキテクチャという意味であったが、現在では、命令セットアーキテクチャより下位の、実装におけるアーキテクチャを指す。
マルチプロセス	複数のアドレス空間でそれぞれのプログラムを並列に実行させる仕組み。
ミドルウェア	コンピュータの基本的な制御を行うオペレーティングシステム(OS)と、各業務処理を行うアプリケーションソフトウェアとの中間に入るソフトウェアのこと。
ムーアの法則	インテル創業者の一人であるゴードン・ムーアが、1965 年に自らの論文上で唱えた「半導体の集積率は 18 か月で 2 倍になる」という半導体業界の経験則。
モデルベース開発（ MBD : Model Based Development）	モデルベース設計により、V 字型開発プロセスで開発を進めていく手法。
モデルベース設計	設計の上流から、設計対象（コントローラ）の機能や制御対象（プラント）の動作をMatlab などのモデル記述言語で記述して評価検証を進めていく設計手法。
ライフサイクル	製品が市場に登場してから退場するまでの間を指す。
リアルタイム	システムが定められた時間制約に従って動作すること。
リアルタイム OS	リアルタイム機能をサポートするオペレーティングシステム（OS）。
ロードマップ	将来どのような製品をリリースしていくかという計画を時系列でまとめた図。

遠隔教育	空間的に離れた状態で行なわれる教育。印刷教材や、テレビやラジオなどの放送系教材，インターネットなどの通信系教材などを用いたものがある。
開発プロセス	開発を成功に導くための手順や工程，要員，成果物，進め方に関する基本的な考え方を定義したもの。
機械学習	人工知能における研究課題の一つで、対象とする過去の大量データから特徴量を抽出（学習）し、現在データの分類や、未知データの予測を行なう方法。
技術成熟度	製品に使われている技術の完成度。
協調エミュレーション（Co-Emulation、コエミュレーション）	ハードウェア部分とソフトウェア部分をエミュレーションにより同時に動作させること。
協調シミュレーション（Co-Simulation、コシミュレーション）	ハードウェア部分とソフトウェア部分をシミュレーションにより同時に動作させること。
協調検証（Co-Verification、コベリフィケーション）	ハードウェア部分とソフトウェア部分を同時に動かして検証すること。
協調合成（Co-Synthesis、コシンセシス）	協調設計で行なわれるハードウェア部分の動作合成、ソフトウェア部分のソフトウェア合成とインタフェース合成の総称。
協調設計（Co-Design、コデザイン）	アーキテクチャ設計の段階で、製品にとって性能・コスト・納期・消費電力などが最適なバランスになるようにハードウェアとソフトウェアの分担を決めていく設計。
組込みシステム（Embedded System）	各種の機器に組み込まれてその制御を行うコンピュータシステム。
組込みソフトウェア（Embedded Software）	組込みシステム上のソフトウェアのこと。
組込み技術スキル標準	ETSS 参照。
高位言語	システムの機能を抽象度の高いレベルで記述できる言語で、C 言語や C++をベースとした SystemC などが該当する。システムレベル記述言語とか高位記述言語とも呼ばれる。
高位言語モデル	高位言語で記述したモデル。

高位合成	抽象度の高い高位言語で記述された動作記述から，クロックに同期して動作する RT レベルの記述を合成すること。動作合成とも呼ばれる。
再利用可能な設計資産	IP を参照。
実チップモデル	プロセッサの機能を実際のチップを使って動作させるモデル。
情報専門学科カリキュラム標準 J07	2007 年度に、世界標準である米国 IEEE/ACM の CC2001-CC2005 を土台として、日本の情報専門教育の状況に対応した見直しを行い、コンピュータ科学（J07-CS） 情報システム（J07-IS） ソフトウェアエンジニアリング（J07-SE） コンピュータエンジニアリング（J07-CE） インフォメーションテクノロジ（J07-IT） の５つの領域と、広く情報について学ぶ内容を定めた一般情報処理教育 GE についてまとめたカリキュラム標準である。
垂直統合型	技術開発、生産、販売、サービス提供などの異なった業務を単一の企業（グループ）がすべて担うビジネスモデル。
水平分業型	技術開発、部品生産、組み立て、販売、アフターサービスなどの業務ごとに、別々の企業（グループ）が得意分野をそれぞれ受け持つビジネスモデル。
製造品質	製品が不良なく作られているかの度合い。
静的検証	検証対象が仕様を満足しているかどうか数学的に検証する手法。原理的にはすべてのケースでの正しさを検証できるが、対象規模が大きいと計算量が膨大となり制約がある。フォーマル検証とも呼ばれる。
設計モデル	コンピュータで実行できるように表現した設計対象
設計探索	アーキテクチャ設計において、最適なハードウェアとソフトウェアの分担を見つける（探索する）設計作業を指す。
設計品質	設計が仕様どおりに作られているかどうかの度合い。
第 3 の IT 化の波	IoT 化時代を迎え、マイケル・ポーターが提唱したもの。IoT により常にモノがネットワークにつながることで、クラウドにデータを蓄積し、そのデータの分析結果を製品とやり取りすることで、製品の機能を飛躍的に向上させると予測している。
第 4 次産業革命	ドイツ政府が民間と一体となって推進しているインダストリ 4.0 の戦略的プロジェクト（2011 年）が狙いとしているもの。

	製造業の生産工程のデジタル化・自動化・バーチャル化する事でマスカスタマイゼーションを可能とし、製造コストの大幅な削減を狙っている。
通信プロトコル	ネットワーク上での通信に関する規約を定めたもの。
統計解析	集められた多くのデータから数学的な手法で事柄の関係性を明らかにする方法。
動的検証	検証対象をシミュレーションやエミュレーションなどにより実際に動作させて検証する手法。与えたテストパターンに対して正しいかどうか判断できるが、それ以外のときは正しいかどうか判断できず、テスト漏れが生じる恐れがある。
匿名化処理	個人を特定できないように個人情報を加工すること。
独立行政法人　情報処理推進機構	IPA を参照。
分散 PBL	遠隔地に分散したメンバーによる PBL。
命令セットモデル	プロセッサの命令セットの動作を模擬するように高位言語で記述されたモデル。
要求品質	顧客のニーズを満足しているかどうかの度合い。