社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2004－ＳＬＤＭ－114　（６）
2004／3／18

UMLを用いた実時間・高信頼性組込みシステムの上位設計についての検討

松井　健† 小松　聡†† 藤田　昌宏†††

† 東京大学工学部電子情報工学科　〒113-8656 東京都文京区本郷 7-3-1
†† 東京大学大規模集積システム設計教育研究センター　〒113-0032 東京都文京区弥生 2-11-16
††† 東京大学大学院工学系研究科電子工学専攻　〒113-8656 東京都文京区本郷 7-3-1
E-mail: †{matsui,komatsu}@cad.t.u-tokyo.ac.jp, ††fujita@ee.t.u-tokyo.ac.jp

あらまし　本研究では、UML（統一モデリング言語）を利用して実時間・高信頼性組込みシステムの上位設計を行う手法について検討する。本研究で提案する手法では、システム動作のスナップショットを多数用意したものから状態マシンを得るという立場に立ち、その過程において処理時間の見積もりやシステム動作の検証を行うことで設計手戻りの抑制を図っている。また、ハードウェア/ソフトウェア協調設計への対応、さらに下位設計ツールとしてSCE (System-On-Chip Environment) との連携を視野に入れる。本手法により、組込みシステムの上位設計において設計や検証の省力化を実現させることができ、ひいては設計を効率よく進めることができる。

キーワード　UML（統一モデリング言語）、組み込みシステム、上位設計、ハードウェア/ソフトウェア協調設計、SCE (System-On-Chip Environment)

Ken MATSUI†, Satoshi KOMATSU††, and Masahiro FUJITA†††

† Department of Electronics Engineering, Undergraduate School of Engineering, University of Tokyo
Yayoi 2-11-16, Bunkyo-ku, Tokyo, 113-0032 Japan
†† VLSI Design and Education Center, University of Tokyo
Yayoi 2-11-16, Bunkyo-ku, Tokyo, 113-0032 Japan
††† Department of Electronics Engineering, Graduate School of Engineering, University of Tokyo
Hongo 7-3-1, Bunkyo-ku, Tokyo, 113-8656 Japan
E-mail: †{matsui,komatsu}@cad.t.u-tokyo.ac.jp, ††fujita@ee.t.u-tokyo.ac.jp

Abstract　In this paper, we examine and propose the process of the high-level system design of a real-time embedded system using UML (Unified Modeling Language). In this process, we plan to verify the design at high-level, and reduce the re-design that arises from the failure in the previous design. In addition, we consider performing hardware/software co-design and using "SCE (System-on-Chip Environment)" as the middle-level system design tool for that purpose. By using this process in the high-level system design of an embedded system, we can design and verify a system efficiently.

Key words　UML, Embedded System, High-level system design, HW/SW co-design, SCE (System-On-Chip Environment)

1. はじめに

1.1 研究の背景

LSIは年々高集積化・大規模化を続け、近年ではシステムLSIと呼ばれる「ひとつのチップに電子機器の機能をすべて収める」技術が確立されている。システムLSIの内部はいくつかの機能別の回路で構成されているが、設計や検証の省力化による開発期間の短縮といった観点から、それらをすべてゼロから開発するのではなくIP（Intellectual Property：機能別に用意された設計済み回路）を組み合わせることで効率良い開発を実現するのが一般的である。

しかし、LSIが多種多様な用途に用いられ、開発分野が多岐に渡るようになった現在、システム構築の際にIPを有効に活用することが困難になっており、これを解決する為の手法が求められるようになってきた。特に、このようなシステムLSIを用い、さらにシステムの高性能化・低価格化が求められる組み

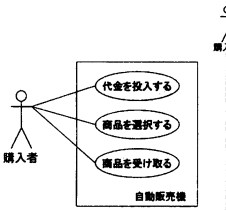

図 1　ユースケース図の例：
自動販売機の機能分析

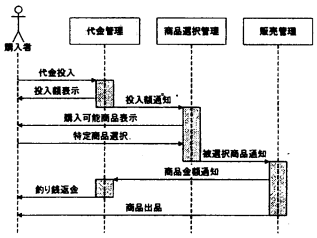

図 2　シーケンス図の例：
自動販売機の内部動作の
スナップショット

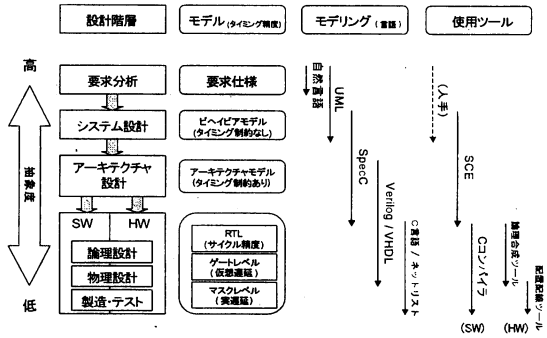

図 3　組み込みシステムの設計フロー概要

込みシステムについて、ハードウェア/ソフトウェア協調設計を踏まえたオブジェクト指向システム記述・設計がどのように行われるべきか、有効な手法が模索されている。

1.2　本研究の目的

以上のような問題に応えるため、組み込みシステムのモデリングに UML（Unified Modeling Language：統一モデリング言語）[1] を応用することが提案されている。そこで本研究では、一般的なハードウェア/ソフトウェア混在組み込みシステムのひとつであるコンパクトフラッシュ（CF）メモリインタフェイスを例題として取り上げ、実際に UML を用いて分析設計を行い、これを通して組み込みシステム一般に適した UML 記述のあり方を考察した。このとき、特に「ハードウェア/ソフトウェア協調設計によく対応する」「SpecC [3] などのシステム記述言語を用いた仕様の表記や設計が容易となる」「IP の再利用性に優れる」「抽象度が高い段階での検証を実現しうる」といった性格を持たせることを目標とした。

2.　関 連 技 術

2.1　UML

UML とは

UML とは既存のオブジェクト指向方法論で提案されたオブジェクト指向モデリング概念を統合したもので、分析・設計モデルのための記法と図（ダイアグラム）を標準化したものである。複数の図を用いることで、複雑なシステムを多面的にとらえ、分析設計に活用することができる。一例を図 1・図 2 に示す。図 1 はユースケース図と称し、システムが外部に対して提供する機能を分析するために用いる。また図 2 はシーケンス図と称し、ある時点でのシステム内部処理のスナップショットを表現するために用いる。UML は開発プロセスについては規定をしていないため、現在はソフトウェアの成果物を仕様化・図式化するといった用途によく用いられ、システム設計、特にハードウェア設計に積極的に適用する手法はまだ確立されていない。

なお、現行の規定では厳密性が低い・ダイアグラム間の相互運用ができないといった問題点が指摘されているが、ダイアグラムの拡張や形式的記述法を導入することでこれを改善し、システムモデリング能力を向上させる事を目的とした新しい規定が UML2.0 [4] としてまとめられつつあり、2004 年 4 月に正式公開の予定である。

eUML

組み込みシステムの開発に UML を利用するには、UML とは別に開発プロセスを定めなくてはならない。そこで、UML を設計に積極的に活用するための一つの解として、組み込みシステムのソフトウェア分析・設計についてのベスト・プラクティスをまとめたものが、渡辺博之らによりガイドライン化された。これが eUML [2] である。

しかし、eUML ではシステムアーキテクチャとして ROM/RAM へのマッピング方法、C 言語による実装、RTOS の利用などを定めることはできるが、LSI の入出力ピンなど、ハードウェア的なことについては定められていない。これは、eUML が特定ハードウェア上でのソフトウェア開発に重点を置いているからである。この点において、ハードウェアとソフトウェアの協調設計を視野に入れた組み込みシステム開発には eUML は適さないことが予想され、実際に後述の例題について適用を試みた結果、以下のような問題点があらわれている。

- LSI のピンの種類や役目など、ハードウェアアーキテクチャに関する記述ができない
- たとえソフトウェアだけを扱うにせよ CPU の数が複数になった場合を表現できない。つまり 1CPU 上で 1 プロセスが走るモデルしか扱えない
- ハードウェアの交換があった場合、再利用性が相当に損なわれる可能性がある。つまり特定のハードウェアがあることを前提として分析を進める側面がある為、ハードが変わると再設計部分がきわめて多くなる

2.2　組み込みシステムの設計フロー概要

図 3 に組み込みシステムの設計フロー概要を示す。アーキテクチャ設計以上の上位設計において要求仕様と制約条件を満足するよう仕様を定め、論理設計以下の下位設計において仕様を満足するよう詳細設計を進めていくのが基本的な流れである。

なお、本研究ではこのうち要求分析とシステム設計を扱っており、またアーキテクチャ設計でのアーキテクチャ選択に、処理時間の見積もりという形で、一定の判断材料を与えている。

2.3　ハードウェア/ソフトウェア協調設計

特定の機能を専用ハードウェアで実装する場合、処理速度に優れるが、設計期間が長くなり再利用性も損なう。ソフトウェ

アで実現する場合はその逆となる。そこで、システム設計の際には双方をそれぞれバランスよく利用することで最適な実装を実現し得る。これがハードウェア/ソフトウェア協調設計であり、組み込みシステム開発で近年用いられるようになってきた手法である。

ハードウェアとソフトウェアの切り分けの判断は、最終的にアーキテクチャ設計段階で行われる。一般に、ハードウェアとソフトウェアの切り分けの判断は難しい。これを少しでも容易にするために、システム設計の段階から切り分けを考慮に入れたモデリングを行い、一定の評価を下しておけるようにすることが望まれる。

2.4 SCE とは

図3に示したように、一般にシステム設計からアーキテクチャ設計までの設計段階ではSpecCと呼ばれるCベースシステム記述言語で仕様が表され、アーキテクチャ選定などの処理が行われる。従来はこの処理を人手で行ってきたが、近年UC Irvine のD. Gajski らによってGUIでの対話的作業によりこれを大幅に自動化するツールが開発された。これがSCE(System-On-Chip Environment) [5] である。SCEはまた、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る検証を行うこともでき、組み込みシステムの設計フローにおいて今後重要な役割を持つことになると考えられている。

3. UMLによる組み込みシステムの記述

3.1 設計上のメリット・デメリット

一般に、システム記述にUMLを用いることのメリットは「開発関係者間で意思疎通が容易になる」「自然言語による曖昧な記述を減らし、再利用性を高められる」ことにあり、さらに組み込みシステムの設計にUMLを用いる場合については「ソフトウェア・ハードウェア双方に対し一貫した表記ができるため、成果物の追跡を行いやすく、設計をIPとして後の設計に利用しやすい」「開発初期から既存IPの利用を考慮できる」「記述の抽象度が高く、より大きなシステムに対しても検証を行う土台を作ることができる」ことが挙げられる。最後の二点に関しては、組み込みシステムの設計期間短縮に直接寄与するという点で特に重要である。すなわち、特定の既存IPの利用を仮定して仕様を決定できたり、詳細設計前に不具合を発見・修正できたりすることで、設計の手戻りによる設計期間やコストの増大を抑えうると考えられる。

一方、UMLには「重複する情報を持った図が複数存在し、また各図の間の関係を表現する方法が規定されていない」「UMLだけではシステムの詳細な性能評価を行いづらい」というデメリットもある。これらは設計成果物の保守性を低下させ、また一方で形式的検証手法を適用する際に障害となる可能性があるため、注意を要する。

3.2 システム設計にUMLを用いる際の方針

本研究では、ハードウェア/ソフトウェア混在の組み込みシステムを設計するため、要求分析からはじまって、最終的にSCEの入力となりうるSpecC記述を容易に導けるレベルのUML記述を得ることを目標とし、そのための指針を考察した。その

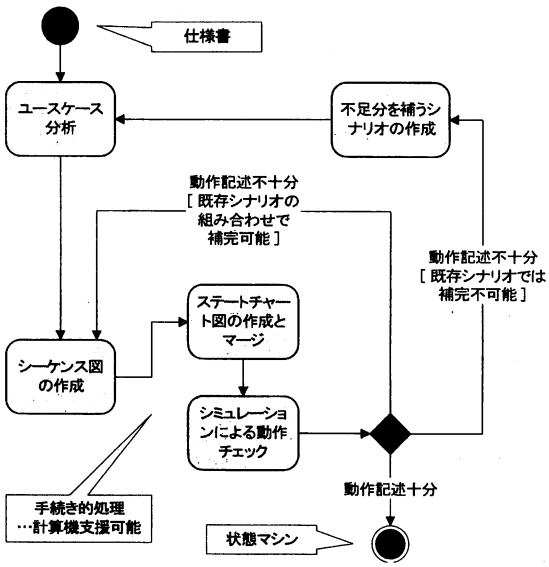

図4 組み込みシステム設計にUMLを用いる際の基本的方針

基本的方針を以下に示し、また図4にまとめた。

(1) まずユースケース分析によりシステムに要求される機能を把握する。

(2) 次に、ユースケース分析の結果からシステムの動作のスナップショットであるシーケンス図を、システムの動作に必要と考えられるだけ複数作成する(あるいは、ひとつ導いた後、順次類似シーケンスを導く)。このとき、処理に必要な時間を粗く見積もることができるので、その結果望ましくないと考えられるシーケンスはこの時点で捨て、改善する。

(3) シーケンス図を得たら、メッセージのやりとりに注目して機械的にステートチャート図を求める。

(4) 得られたステートチャート図についてシミュレーションを行うことで不足している動作記述を考え、必要と思われる動作をシナリオとしてユースケース分析に追加し、そのシーケンス図を描き、そこから導かれるステートチャート図と元のステートチャート図をマージする。

(5) そして最終的にはシステムに要求された動作を全て実現するステートチャート図が得られる。

ステートチャート図から実行可能なSpecC記述を得ることは可能であるため、以上の分析を行うことでUMLからSCEへと設計をつなげていけるものと考えた。

なお、ハードウェアとソフトウェアの切り分けをUML記述の段階で行うか、あるいはSCEで行うかの選択がある。これは、SCEに入力するSpecC記述が仕様モデルかアーキテクチャモデルかの違いになってあらわれる。本研究では、ひとまず後者の立場に立って、システムの機能に注目して分析を行うことでハードウェアとソフトウェア双方を視野に入れる設計を行う、という方針をとったが、将来的には前者の立場に立ち、UMLレベルでもシステムの性能評価と実装への指針についてある程度分析しうる手法について考察する必要がある。

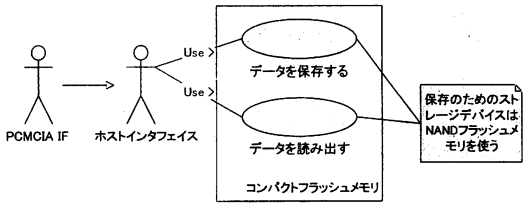

図 5　最も抽象度の高いユースケース図

4. 例題：CF メモリインタフェイスの設計

組み込みシステム設計に適した UML によるシステム記述方法を探るため、デジタルカメラに使われるリムーバブル・ストレージメディアとして良く知られているコンパクトフラッシュ（CF）メモリのインタフェイスを例題として考え、前項で示した手法が適用できるか検討する。なお、この例題は過去に同様のシステムを設計した例から、ハードウェア/ソフトウェアが混在し、規模がさほど大きくならないことがわかっている。

4.1 ユースケース分析

ユースケース分析とは、システムが外部に提供する機能を抽出する作業である。図 3 中の要求分析に相当する。

まず、CF メモリを設計するためにどのような機能が必要とされるか、というところから始める。ここでは、パソコンにある PCMCIA インタフェイスに何らかのストレージを接続して、それに対してデータの保存と読み出しを行うということが第一の機能となる。そこで、図 5 のように PCMCIA インタフェイスが「データを保存する」「データを読み出す」といった処理をシステムに求めている、というユースケース図を得る。ここで、ユースケース図中の各ユースケースについて、自然言語によりそのシナリオを記述する。例外処理については、はじめから全てを列挙することは難しいので、くり返し詳細化の過程でしばしば追加されていくことになる。

次に、詳細なユースケース分析を行うにあたって、高抽象度ユースケース分析で制約として与えられた NAND フラッシュメモリについてその機能を洗い出す必要があるので、図 5 を得たときと同様、これを行う。NAND フラッシュメモリの機能については、すでに仕様書としてまとめられているので、これからユースケース図を描くことになる。さらに、ユースケース図中の各ユースケースについてシナリオを記述する。

さて、以上で得られた NAND フラッシュメモリのユースケース分析結果を、高抽象度ユースケース分析で得られた結果とあわせると、図 6 を得る。ここで NAND メモリへのデータ書き込み処理は、ホストインタフェイス側でのデータの保存処理の一部であるとして両者を include 関係で結んだ。NAND メモリからのデータ読み出しについても同様である。また、ホストインタフェイス側に必要な処理を考えた結果、システムを初期化する処理が必要と考えられたので、この処理をユースケースとして書き加えた。その上で、データの保存と読み出しについて、論理ー物理アドレス変換・NAND シリアルデータ作成/展開といった機能を依存関係を用いて表した。なお、ECC 付加/適用

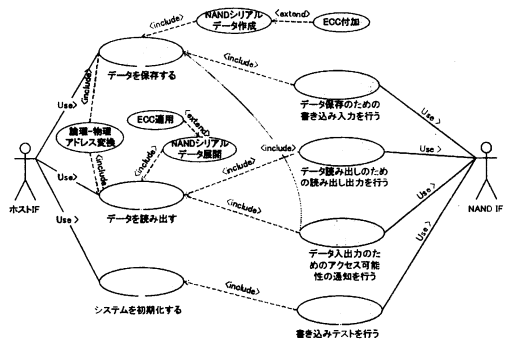

図 6　CF メモリインタフェイスの詳細化したユースケース図

については NAND シリアルデータ作成/展開に対するオプショナルな機能であるので、extend 関係でそれらを結んだ。

4.2 シーケンス図を用いたシステムの動作分析

ここではユースケース分析で洗い出された各機能について、その動作の実現を考える。まずユースケース分析から、PCMCIA インタフェイス・NAND メモリインタフェイスとの入出力と、それらの間でアドレス空間の変換が必要であることがわかる。よって、一番単純なシステムの動作としては、単純にアドレス空間の対応を取ってストレージへ読み書きするというものが考えられる。

しかしこのとき、システムのとるシーケンス（図 7）を考えると、このままではストレージの記憶容量の活用や処理時間の短縮が実現できないことがわかる。具体的にはシーケンス図をもとにして次のような処理時間見積もりが得られ、その結果 NAND メモリへの書き込み速度がネックになることがわかる。また、「論理ー物理アドレス変換（X）」「シリアルデータ作成（Y）」の値が数百 μs オーダになるとすれば、これも全体の処理時間に大きく影響することがわかる。

・ 1B あたりの書き込み時間

(400ns+200μs)+(300ns+Xns+Yns)

…X>100 のとき。前項は外部との入出力に要する時間、後項は内部処理に要する時間。内部処理に要する時間の最小単位は、10MHz 動作を仮定して 100ns とした

・ 1 メモリチップ (64MB) あたりの保存容量

128kB:0.2%…1 ページ 1B しか書き込んでいないため

これに対し、同一ページ (512B) への保存要求が続くならば書き込むべきデータをデータバッファにためておき、一度に NAND フラッシュメモリに書き込むことを考え、図 8 を得る。シーケンス図上から同様に処理時間の粗い見積もりを行うと、以下のように確かに図 8 のシーケンスの方が優れていることが確認できる。さらにこの結果から、「論理ー物理アドレス変換（X）」「シリアルデータ作成（Y）」「シリアルデータ展開（Z）」「バッファへの書き込み・読み出し（BUFIN, BUFOUT）」がいずれも数十～百 μs のオーダなら、「論理ー物理アドレス変換」の高速化が全体の処理を高速化させる上できわめて重要であることがわかる。したがって、この部分をハードウェア化するなどし、高速化を図るべきことが指針として得られる。

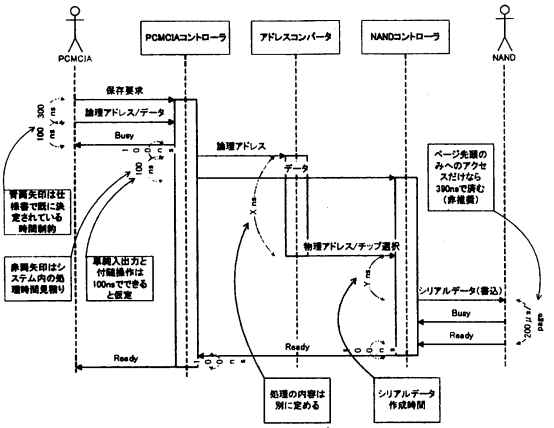

図 7　最も単純なデータ保存シーケンス図

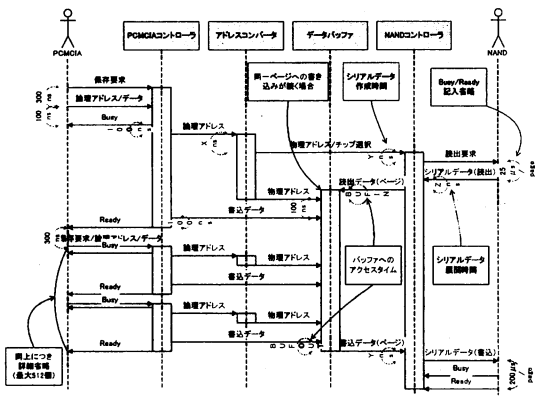

図 8　バッファを使い、同一ページをまとめて NAND メモリへアクセスするシーケンス図

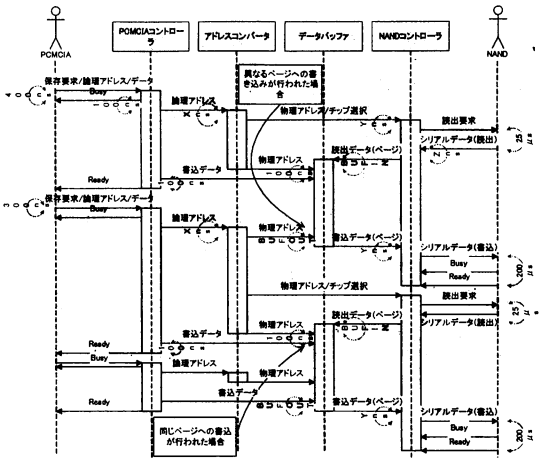

図 9　バッファを使い、ページ単位で NAND メモリへアクセスするシーケンス図

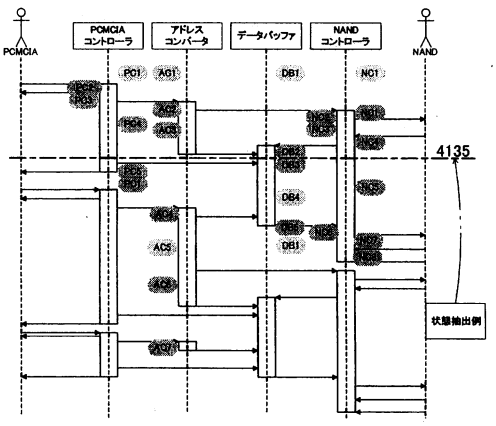

図 10　番号を付与した書き込みシーケンス図

・ 1B あたりの書き込み時間

$\{(400ns+25\mu s+300ns+X+Y+Z+BUFIN)+(400ns+200ns+X)*510+(400ns+X+BUFOUT+Y+200\mu s)\} / 512$

$=(532\mu s+512X+2Y+Z+BUFIN+BUFOUT) / 512$

(X,Y,Z,BUFIN,BUFOUT の時間単位は ns)

・ 1 メモリチップ (64MB) あたりの保存容量

64MB:100%…1 ページを完全に使用

さて、ここでは同一ページへの書き込みが連続することを前提とした。一般にまとまったデータを書き込む際はそれらのアドレスが隣接することが多いので、その点においては主たるシーケンスとして正しいと言える。しかし、実際には異なるページへの書き込みが行われることも当然起こり得るので、それを考慮しなければシーケンス図からステートチャート図を作成するときに望ましいステートチャート図を得ることができなくなってしまう。そこで、異なるページへの書き込みと、同一ページへの書き込みが混在するシーケンスを考えた。その結果を図 9 に示す。

以上で PCMCIA からの保存要求に対し、十分な性能を発揮するシステムの主たるシーケンスを得ることができた。これと同様の手順を踏むことで、PCMCIA からの読み出し要求に対するシーケンス、初期化シーケンスについても同様に得ることができる。

4.3　ステートチャート図の作成と改善

前項で得られた書き込みシーケンス図において、メッセージのやりとりが行われる前後で状態が変わると考えれば、その関係を列挙することで機械的にステートチャート図が導ける。

具体的には、まず図 10 に示すように、メッセージの前後で異なる番号を付与する。その上で、番号の組み合わせを一つの状態と見なしてこれらの間の関係を結ぶと、ステートチャート図が得られる。ここで、状態に付けた番号の組み合わせが同じものがある。これらはそれぞれ、あるシーケンスの中にある同一の状態と考えることができる。したがって、ステートチャート図を記述する際はこれらは一つの状態にまとめることになる。

このようにして得られた書き込みシーケンスに対するステートチャート図を図 11 に示す。また、同様にして読み出しシーケンスや初期化シーケンスに対するステートチャート図も得ることができる。

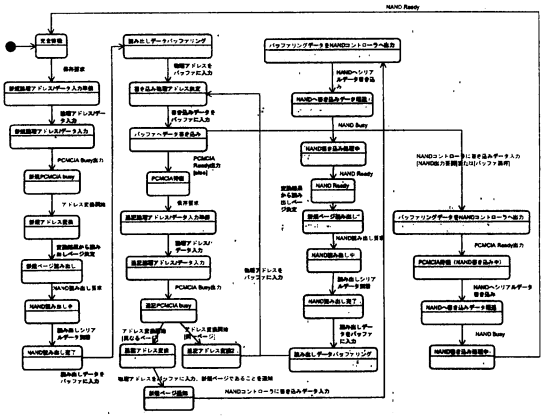

図 11　書き込みシーケンスに対するステートチャート図

以上、ユースケース分析で洗い出された基本的な機能について、ステートチャート図を得ることができた。この後はこれらのステートチャート図に対してユースケース分析から考え出される入力パターンを用意して、ステートチャート図ベースのシミュレーションを行うことで、今まで考慮しなかった設計の漏れ・抜けなどの誤りを発見・補完していくことになる。

一例として、保存要求・読み出し要求・保存要求という順序で PCMCIA インタフェイスから入力がある場合を考える。一旦読み出しがはじまった後ははじめの分析で得られた読み出しシーケンスにおいて PCMCIA が Ready かつ機能実体全てが非活性という「完全待機」状態にならないため、「完全待機」状態からはじまるべき保存シーケンスが読み出しシーケンスの後呼ばれることができず、読み出し要求の次に保存要求が来たときに破綻することがわかる。

そこで、補完シーケンスとして保存要求と読み出し要求が交互にあらわれるシナリオとシーケンスを考え、これから補完ステートチャート図を導く。そして、これと今まで得たステートチャート図とをマージすることで、図 12 に示すような、先ほどの入力にも正しく動作するステートチャート図を得ることができる。

5. 結　　論

本研究のまとめ

本研究では組み込みシステムの上位設計に UML を用いて手続き的に設計を進める手法について一定の方針を立て、さらに CF メモリインタフェイスの例題を通してそれが実践できることを確認した。特に、SCE との連携を考えることとシステムの実現する機能に着目することとでハードウェア/ソフトウェア協調設計に適した手法とすることができた。

一方で、得られたステートチャート図に対して要求分析から考え出されるテストパターンを入力することで、考慮するべきシステム動作のスナップショットの漏れ・抜けを発見できる、すなわち要求分析～システム設計という異なる設計段階間に対する検証ができることも確認した。

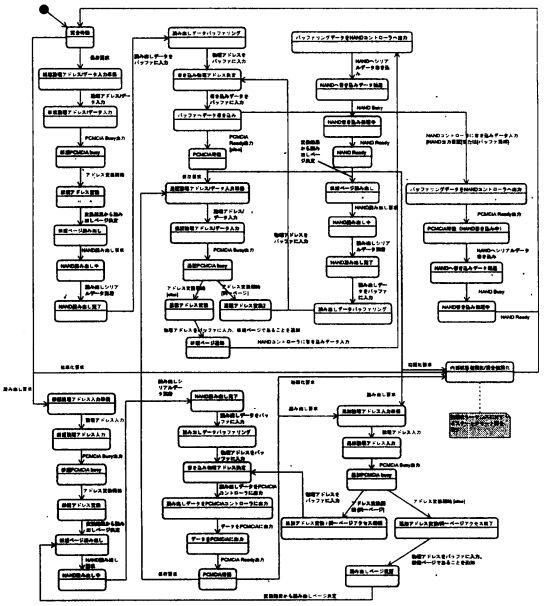

図 12　システム全体のステートチャート図

今後の課題

より上位の設計段階から実行可能な記述を得、計算機を用いて十分な検証を行うという立場に立つと、現行の UML では表記法が意味論的に厳密ではないため、その実現に困難が伴う。そこで、厳密で実行可能な表記法を導入する必要がある。形式的記述を UML ダイアグラムと併用することで網羅的な動作検証が可能になり、上位設計における十分な検証が実現できるであろう。また、ステートチャート図の生成とそのマージなど、手続き的な操作を求められる箇所の分析作業を機械化し、誤り混入の排除・省力化を図ることもできよう。

一方で、設計初期から特定の機能を特定のハードウェアまたはソフトウェアに割り当てることが決まっている、ということが実際の設計においてはしばしばあり得る。そこで、機能のみを UML で記述するのではなく、機能の実装手段についても UML で記述し、設計に取り入れる手法を考察すべきであろう。

文　　献

[1] グラディ・ブーチ「UML ユーザーズガイド」ピアソン・エデュケーション, 2002.

[2] 渡辺博之 他「組み込み UML」翔泳社, 2002.

[3] D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, S. Zhao, "SpecC: Specification Language and Methodology," Kluwer Academic Publishers, Boston, March 2000.

[4] "UML 2.0 Specifications nearing completion," http://www.omg.org/uml/

[5] S. Abdi, J. Peng, R. Dömer, D. Shin, A. Gerstlauer, A. Gluhak, L.Cai, Q. Xie, H. Yu, P. Zhang, D. Gajski, "System-On-Chip Environment (SCE): Tutorial," CECS, UC Irvine, Technical Report CECS-TR-02-28, September 2002.

[6] CompactFlash Association, http://www.compactflash.org/

[7] NAND Flash Applications Design Guide, http://www.semicon.toshiba.co.jp/eng/prd/memory/doc/pdf/nand_applicationguide_e.pdf