3 System Level Design

システムのモデル化と計算モデル

富士通（株）
大塚 正人
otsuka.masato@jp.fujitsu.com

（株）インターデザイン・テクノロジー
荒木　大
araki.dai@interdesigntech.co.jp

埼玉大学
吉田 紀彦
yoshida@ics.saitama-u.ac.jp

計算モデルとは

年々大規模・複雑化するシステムオンチップ（以下，SoCと呼ぶ）の開発の設計生産性を高めるため，システムレベルデザインの重要性がここ数年ますます高まっている．この問題を解決する技術として，システムレベル設計言語やシステムレベル設計ツールの普及が進みつつある．しかし言語と設計ツールが揃えばすべてが解決されるわけではない．

システムレベル設計言語を有効活用するためには，言語を使ってシステムの仕様をどのようにモデル化して記述するのかという方法論や技術が必要になる．また，システムレベル設計ツールを有効に活用するためには，ツールがどのようなモデルに対応しているのか，どのような動作をするのかを形式的に理解することが重要になる．このためのベースとなる概念として計算モデルがある．

「計算モデル（Model of Computation）」とは，システムの機能や振舞い，あるいはその性能を形式的に定義し，詳細化するためのフレームワークである．計算モデルを使えば，実装方法に関する詳細を抜きにして，システムの振舞いをモデル化して分析することができる．

計算モデルの特徴や性質を正しく理解し，計算モデルが得意とする分野のデザインやアプリケーションに対して，計算モデルの特徴を活かしたモデル化を行うことにより，設計品質と設計生産性の高い効率的な設計が可能となると考えられる．また，本特集の「システムレベル設計フローと設計言語」で解説されるシステムレベル設計言語を用いた設計を行う際にも，設計言語と計算モデルとの関係を把握していることが望ましい．設計言語自身は計算モデルを直接表現するわけではなく，コーディング・ルールやライブラリなどを整備して，特定の計算モデルの枠組みに合うように仕様を記述することにより，何らかの検証や性能見積りを行うことができ，下流の抽象度のモデルに自動的に合成することができるようになる．

一般にプログラミング言語の世界における「計算モデル」と言えば，関数型計算モデルや項書き換え型計算モデルまたは述語論理等が代表例として想起されるが，本稿では，システムレベルデザインが対象とするSoC，ハードウェア／ソフトウェアシステム，組み込み系機器における機能や演算の振舞いを表現するのに利用されるモデルに対象を絞って解説する．また，これ以外に，対象システムの物理的な構造を表現するためのモデルや，Entity-Relationship Diagramのようにデータの構造に着目したモデルもあるが，これらも本稿の解説の対象からは外す．

システムの機能や振舞いを表現するための計算モデルは大きく次の4つのカテゴリに分類することができる．

- プロセス系：互いに通信しあうプロセスの集まりによって対象を表現する計算モデル
- ペトリネット系：ペトリネットをベースとして拡張された計算モデル
- FSM系：FSM（Finite State Machines）をベースとして拡張された計算モデル
- データフロー系：データフローグラフをベースとして拡張された計算モデル

後で紹介する，CSP（Communicating Sequential Processes）やKPN（Kahn Process Networks）は，プロセス系の計算モデルの代表例である．CSPではプロセス間通信はランデブーで表現する，KPNではプロセス間通信をサイズ無制限のFIFOで表現する．また，このような，さまざまなかたちのプロセス間通信モデルを形式的に定義するための枠組みとしてTSM（Tagged Signal Model）がある[1]．TSMはプロセス間通信を定義するためのメタモデルである．

ペトリネット系の計算モデルの例としては後で紹介するMTG（Multi-Thread Graph）などがある．

FSM系の計算モデルには，UMLの中でオブジェクト／クラスの動的な振舞いを表記する記法として採用されているStateChart[2]がある．本稿ではシステムレベルデザインにおけるFSM系の計算モデルの代表例としてCFSM（Co-design Finite State Machines）を後で紹介する．CFSMは，FSMとプロセス間通信モデルを融合した計算モデルである．これ以外に，RTL設計をフォーマルに表現する計算モデルとしてFSMD（Finite State Machines with Datapath）[3]などがある．

データフロー系の計算モデルには，後で紹介するSDF（Synchronous Dataflow）などがある．

システムレベルデザインにおける計算モデルの要件と特徴

システムレベルデザインにおいて計算モデルに求められる要件や，計算モデルを比較する上での分類軸を説明する．

(A) 時間のモデル化

演算や通信の実行に要する時間の概念を表現できるかどうかにかかわる分類である．一般に「Un-timedモデル」と「Timedモデル」の2種類がある．プロセス間の同期関係をイベントの生成・イベントの受け渡し・イベントの消費の関係で表現すると，次のような分類ができる．

「Un-timedモデル」では演算や通信に要する時間を直接表現する概念はないが，イベント間の半順序関係を規定できる．

「Timedモデル」では，時間の概念があり，イベント間の全順序関係まで規定できる．また，「Timedモデル」で扱う時間の概念には次のようなものが挙げられる．

- 通信時間：イベントの受け渡しに要する時間，すなわち，プロセスがイベントを生成してから別のプロセスがそれを消費可能となるまでの時間．
- プロセス実行開始時間：プロセスが入力イベントを消費可能となってから実際に消費するまでの時間．
- プロセス実行時間：プロセスが入力イベントを消費してから出力イベントを生成するまでの演算に要する時間．

(B) プロセス間通信

プロセス間ネットワークの通信路の性質をどのように表現できるかに関する分類である．

- FIFOの有無：通信路をサイズ1のバッファで表現するのか，FIFO待ち行列で表現するのか，などでの区別がある．また通信路をFIFOで表現できる場合は，待ち行列のサイズが有限であるか無限であるかによっても計算モデルで扱える対象に違いが生じる．
- 読み取り制約：「ブロッキング・リード」の場合は，通信路が読み取り可能状態にない場合は読み取り可能になるまでプロセスが待ち続けるモデルになる．「ノン・ブロッキング・リード」の場合は，通信路が読み取り可能状態にない場合はプロセスが他の処理を行うことができるモデルになる．
- 書き込み制約：「ブロッキング・ライト」の場合は，通信路が書き込み可能状態にない場合は書き込み可能になるまでプロセスが待ち続けるモデルになる．「ノン・ブロッキング・ライト」の場合は，通信路が書き込み可能状態にない場合は，書き込もうとしたデータを捨てるか強制的に書き込むモデルになる．

(C) 同期モデル／非同期モデル

プロセスの動作やプロセス間の通信が同期か非同期かの違いによる分類である．これは，前項のプロセス間通信の性質をどのように表現できるかによって間接的に決まる特徴である．たとえば，通信路をFIFO待ち行列で表現できるならば，プロセス間の通信を非同期に行うことができる．一方，通信路にバッファが存在せずブロッキングでしか読み書きができない通信路の場合は，プロセス間の通信はランデブー型の同期通信となる．

(D) 割り込み

プロセスに対して，外部から割り込みを行うことが可能な表現方法の有無による分類である．後述のCSPの場合は，プロセス内で割り込みを表現できる．また，MTGの場合はイベントノードによって表現することができる．

(E) 活性化ルール

プロセスを実行可能にするための手段あるいはルール

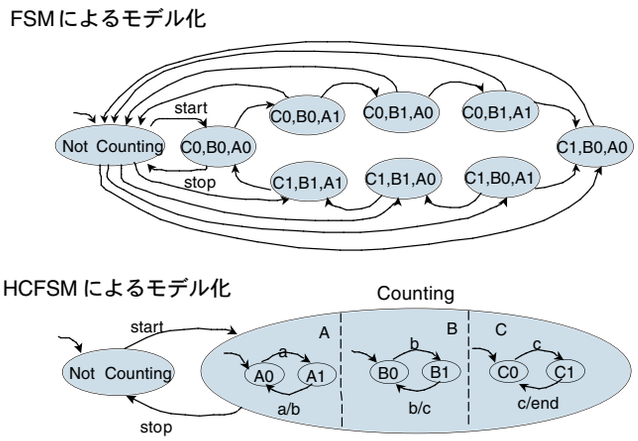

図-1　階層構造の利用（FSMとHCFSM）（文献5より）

の指定方法による分類である．プロセス間通信モデルの場合は，イベントを通信することによってプロセスの活性化を表現する．ペトリネット系のモデルの場合は，入力トークンが揃うとプロセスが実行可能になる．FSM系のモデルの場合は，状態遷移のトリガーとなるイベントを外部から受け取るかたちで表現される．

(F) 階層構造

階層的にモデルを定義する表現手段が用意されているかどうかによる分類である．図-1は3ビットカウンタの動作を単純なFSMで表現した場合と，階層的なFSM表現を持つHCFSM（Hierarchical Concurrent Finite State Machines）[4), 5)]で表現した場合の違いを示している．

FSMでは組合せの数だけ状態数が必要になるが，HCFSMではビットごとにカウンタの動作を表現できる点でFSMよりも理解しやすい計算モデルになっていることが分かる．

(G) プロセスの状態

CSPやKPNの場合はプロセスの状態表現はチューリングマシンに相当する．ペトリネット系の計算モデルの場合は，トークンの数や色で表現される．FSM系の計算モデルの場合は明示的な有限状態表現となる．

(H) 決定性

スケジューリングアルゴリズム等に依存せず，等価な入力イベント系列に対して常に等価な出力イベント系列が得られる保証があるかどうかによる分類である．決定性の有無は，計算モデルの構造によって左右される．フォーマルな検証や，モデルの合成を行うことを考えた場合には，一般には，決定性のある計算モデルを利用したほうがよい．

これに関連して，入力イベント系列に依存しない静的スケジューリングが可能かどうか，という観点での分類もできる．

(I) デッドロック

デッドロックが起こり得るモデルを表現可能かどうかによる分類である．デッドロックが起こり得るモデルの場合は，デッドロックの有無を検証することが可能かどうかでも分類できる．デッドロックの有無も，計算モデルの構造によって左右される．記述したモデルに対するフォーマルな検証を行いたい場合にはデッドロックが検出可能であるかは重要な要因となる．

(J) リソースシェアリング

リソースの排他的利用の制御やリソースの利用に関する制約を表現可能かどうかに関する分類軸である．KPNやCSPは，そのままでは表現が不可能である．ペトリネット系のモデルでは表現が可能であり，FSM系のモデルの場合は表現が可能であり，さらにこれに関する検証も可能である．

代表的な計算モデル

ここではシステムレベルデザインにおける代表的な計算モデルをいくつか取り上げて解説する．

Communicating Sequential Processes（CSP）[6)]

CSPでは，対象システムを複数のプロセスが通信チャネルを介したプロセス間通信により並列に動作するモデルとして表現する．

CSPのプロセス間通信ではプロセスが両方のパーティによって合意された時のみコミュニケートされるランデブー型モデルとして表現される．具体的には，プロセス間通信はブロッキング型での通信チャネルを使って行う．1つのプロセス内ではタイミング問題を扱わないUn-timedな逐次処理で表現する．このようにプロセス間通信を定式化することによって，プロセス代数で通信と並列システムを数学的に記述することができる．これによってシステムが安全に動作するかを検証することが可能になる．

Kahn Process Networks（KPN）[7)]

KPNモデルは図-2に示すように並列に動作可能なプロセスと，プロセス間で通信を行うための唯一の手段であるFIFOでシステムを表現する．FIFOのバッファサイズは無限であり，データの送り側プロセスは受け側プ

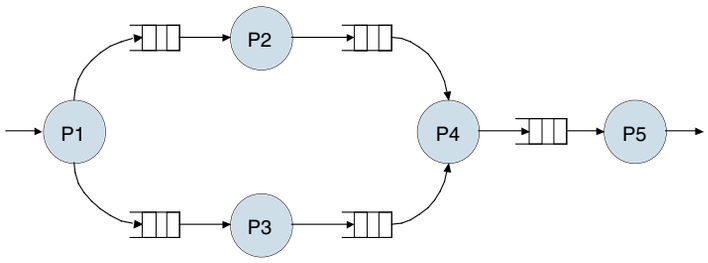

図-2　Kahn Process Networks

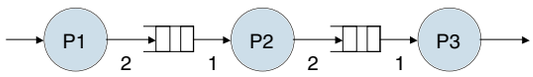

図-3　Synchronous Dataflow の例

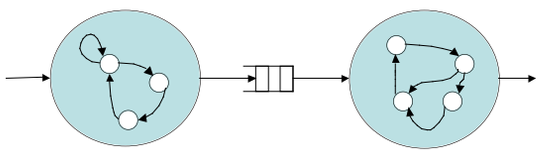

図-4　Co-design Finite State Machines

ロセスの状態に依存せずいつでも確実に送信を行うことができる（ノン・ブロッキング・ライト）．一方受け側プロセスはFIFOにデータが存在しなければ送り側プロセスがデータを送るのを待ち続けなければならない（ブロッキング・リード）．

通信路におけるデータの損失を考慮する必要がないことと，データの送受信の仕組みが単純であることから，通信プロトコルやタイミング等の詳細を気にすることなくプロセス内部の処理にのみ集中して対象システムをモデル化することが可能なのが特徴である．さらにKPNモデルは，プロセスの実行順序をどのように変えようとも，同じ入力系列に対しては同じ出力系列を返すという性質（決定性）を備える．これらの特徴からKPNはシステムレベル設計の初期段階での機能設計に用いると特に有効である．

一方で，すべてのプロセス間通信がFIFOによる非同期通信によって行われるため，プロセス間で頻繁に制御信号のやりとりを行う必要がある制御システムや，リアルタイムシステムのモデル化には適していない．

Synchronous Dataflow（SDF）[8]

KPNではサイズ無制限のFIFOを用いたが，これは計算モデルで表現したシステムを実装する際には大きな障壁となる．また，プロセスの実行順序が任意であるため，動的なスケジューリングによるオーバーヘッドも無視できない．

SDFではFIFOに制約を与えることにより，これらの問題が解決される．SDFモデルでは1つのプロセスをより細かい実行単位であるアクターの逐次実行で定義し，アクターが実行されるごとに書き込み・読み出しするデータの数をFIFOごとにあらかじめ決定しておく．

図-3にSDFの例を示す．図でプロセスとFIFOの間の数字は，実行ごとに書き込み・読み出しをするデータの数を意味する．図の例ではP1の1回の実行に対してP2が2回でP3が4回実行されるとすれば，FIFOのサイズはそれぞれ最大2および4でよいことになる．

このように必要なFIFOのバッファサイズが決まり，スケジューリングが静的に決定可能となる．また，SDFモデルはKPNの場合と同様に決定性を有する．

これらの特徴によりSDFはデータフロー系システムの仕様記述に特に向いており，SDFを利用したさまざまな設計ツールがある．これらには，Synopsys社のCoCentric System Studio，CoWare社のSPW，カリフォルニア大学バークレイ校のPtolemy IIなどがある．

Co-design Finite State Machines（CFSM）[9]

KPNは自由度の高いモデル化が可能で汎用性に優れているが，記述したモデルをハードウェアおよびソフトウェアからなるシステムとして実装するためには大きなギャップが存在する．CFSMは，名前にCo-designとあるように，モデルからハードウェア・ソフトウェア協調設計によりシステムの実装を実現すべく考案された計算モデルである．

CFSMモデルではプロセスをFSMで定義し，プロセス間の通信にはサイズ1のバッファを用いる．しかしながら，サイズ1のバッファによる通信では制約が大きいため，図-4のようにプロセス間通信にFIFOを使えるようにCFSMを拡張したAbstract CFSM（ACFSM）およびExtended CFSM（ECFSM）が提案されている[9]．

ACFSMではサイズ無制限だが読み書きがノン・ブロッキングなFIFOを用いて通信を行う．さらにエラーや例外処理のモデル化のためにFIFO上のデータをすべて消去する操作を許す．FSMは遷移可能な状態が複数存在する場合にはどの状態に遷移してもよく，このためKPNとは異なりモデルが決定性を有することはない．

ACFSMをより実装寄りにしたECFSMではFIFOのバッファサイズは有限となる．また，FIFOごとにブロッ

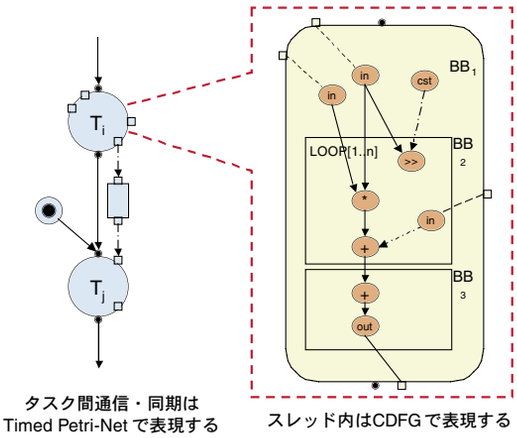

図-5　MTGの基本構造（文献11より）

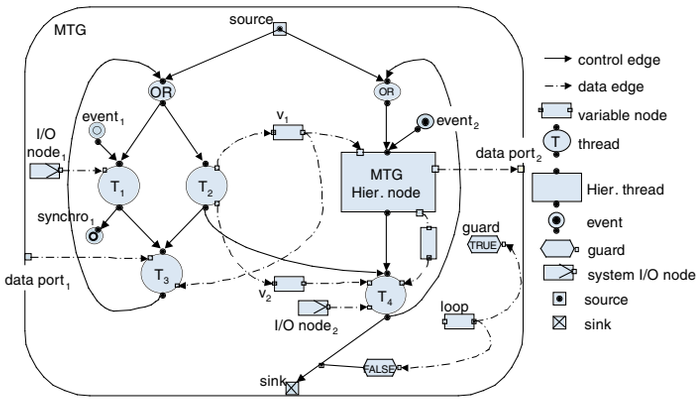

図-6　MTGの構成要素（文献11より）

キングまたはノン・ブロッキング・ライトを選択することができる．有限サイズのFIFOにノンブロッキング書き込みを行った場合，当然データの損失が考えられる．このため，モデル作成者はFIFOのサイズを十分大きく確保するか，データの損失を考慮した通信プロトコルをモデル化するか，あるいはスケジューリングにタイミング制約を与えてデータの損失が起こらないことを保障しなければならない．

ACFSMとECFSMの使い分けとしては，設計の初期段階では通信の詳細を検討する必要がないACFSMを用いて各プロセスの機能を決定し，ECFSMで通信の詳細化を行うというような使い方をする．対応ツールとしてはカリフォルニア大学バークレイ校のPolisシステム[10]や，Polisをベースとした Cadence Design Systems社のVCC（Virtual Component Co-design）Environmentがある．

Multi-Thread Graph（MTG）[11]

MTGは，ベルギーのIMECで開発された主にリアルタイム組み込みオペレーティングシステムの仕様を記述するための計算モデルである．

MTGの基本的な構造を図-5に示す．MTGではTimedペトリネットをベースとしたグラフ表現でタスク間の通信を表現する．タスクがMTG上でのノードに対応する．タスク内の計算処理はCDFG（Control Data Flow Graph）で表現する．

MTGのグラフは，リソースシェアリングのための表記法（semaphore node）や外部環境との同期のための表記法（event nodeやsynchronization node）等の表現要素を追加して，システム仕様の記述・表現能力を高めている（図-6参照）．

制御フローとデータフローの両方を表現できるグラフで，タイミングの制約も記述することができる．同期通信と非同期通信の両方を表現することができ，リソースの排他制御も容易に表現できる．また，モデルを階層的に表現するための記法がサポートされており，システム全体を機能ごとに分割して表現することができる．

システム仕様のモデル化，検証，実装に使用することができ，タイミングの解析やシステム合成も考慮されている．これは，動作セマンティクスが形式的に定義されていることによる．これによって，デッドロック，デッドコード，リソース競合も解析することができる．

リアルタイム組み込みシステム，特に組み込みソフトウェアの設計に対して，並行プロセス仕様記述から自動的にプロセッサ向けのマシンコードを生成するシステム合成という設計方法論が提案されている．シングルプロセッサ，あるいは，マルチプロセッサのプラットフォーム上でタスクレベル，命令レベルの各々のレベルで並列性の解析とスケジューリングを行い，ターゲットプロセッサのマシンコードを生成する．

MTGは，リアルタイム組み込みシステムの組み込みソフトウェアの設計に着目して開発された計算モデルであるが，ハードウェア設計に適用できる要素が多く，システムレベルデザインにも活用できる可能性が高いと考えられる．

計算モデルを利用したシステムレベル設計

計算モデルを利用したシステム設計の事例としてSPADE（System level Performance Analysis and Design space Exploration）を紹介する．文献12）では，MPEG-2デコーダの設計を例に，SPADE手法でシステム設計を行う際にどのように計算モデルを用いるかが解説されている．

すでに手元にC言語で記述されたMPEG-2デコード

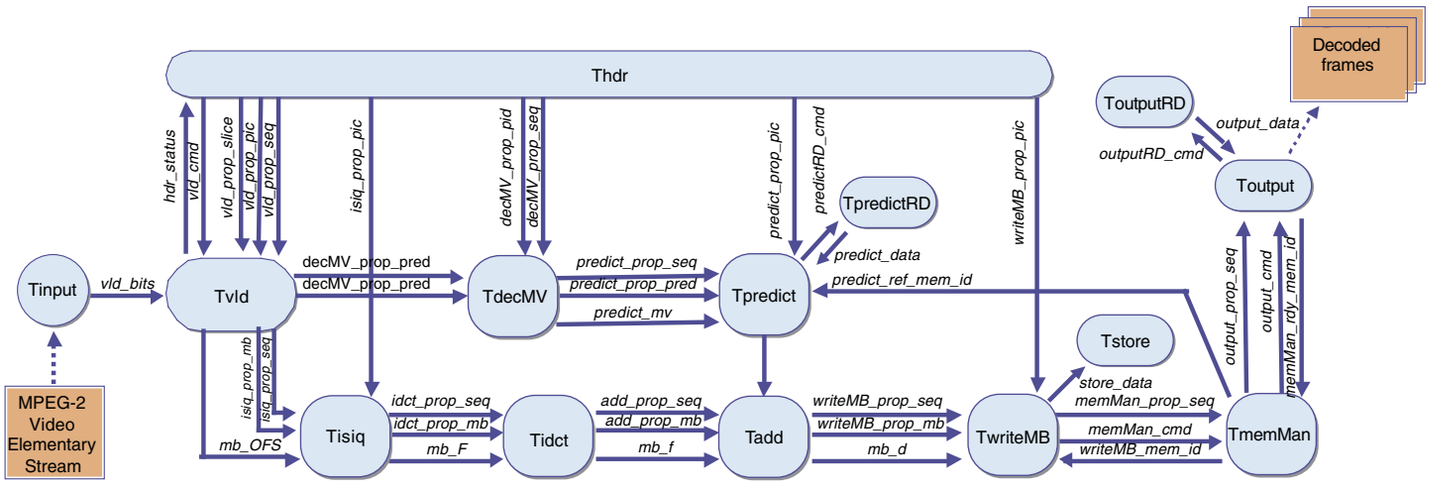

図-7　MPEG-2デコーダのKPNモデル（文献12より）

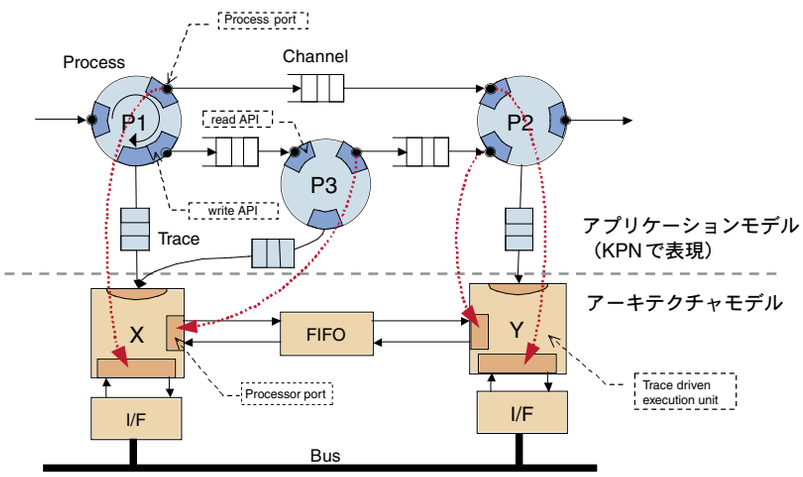

図-8　KPNモデルとアーキテクチャモデルのマッピング（文献12より）

プログラムが存在すると仮定する．SPADEではまずMPEG-2デコードプログラムをKPNで表現する（図-7参照）．すなわちバッファサイズ無限のFIFOで通信を行う複数のプロセスと，FIFO,およびFIFOでやりとりされるデータタイプをC++言語で記述する．各プロセスは，(1) 入力用FIFOにデータがくるまで待ち，(2) 入力用FIFOからデータを読み込み，(3) データに対するプロセス固有の処理を行い，(4) 出力用FIFOにデータを書き込む，という処理を繰り返す．

正常に動作するKPNのアプリケーションモデルができたら，次にプロセスとFIFOにプロファイリング用のライブラリメソッド呼び出しを追加してシミュレーションを行う．これにより各プロセスのどの処理がどれくらいの頻度で実行されるか，また各FIFOがどれくらいのデータ量を扱うかを計測できる．これらのプロファイルを元にプロセスの演算の粒度やFIFOでやりとりされるデータの粒度を決定する．

KPNのアプリケーションモデルが完成すると，次にアーキテクチャモデルとのマッピングを行う．アーキテクチャモデルは，アプリケーションモデルとは独立に作られるもので，プロセッサとメモリをどのようにバスで接続するかといったアーキテクチャ構成を純粋にモデル化したもので，機能に関するモデルを一切含んでいない．

図-8を使ってアプリケーションモデルと，アーキテクチャモデルのマッピングについて説明する．

図-8の上側がアプリケーションモデルで，P1～P3の3つのプロセスがFIFOを介して通信しあう．仕様記述においては，FIFOはチャネルとして表現されており，プロセスが持つポートを介してread/write APIを使ってデータを送受信するかたちで記述される．また，プロ

セスの内部における演算やread/writeのデータ送受信の振舞いがトレースのかたちで取り出せるようになっている．

図-8の下側のアーキテクチャモデルは，トレースを実行することができる仮想CPUであるTDEU（trace driven execution unit）と，複数のTDEUの間での通信インタフェースが定義される．図では，汎用バスを使ったインタフェースとFIFOが書かれているが，FIFOを実装する手段としてはTDEU間で共有メモリを介して行うなどのさらに詳細な実装手段の選択が可能である．

図-8では，プロセスP1とP3はTDEUのXで実行し，P2はYで実行するようにマッピングしている．また，P1からP2への通信はバスを経由し，P3からP2への通信はFIFOを経由するようにマッピングしている．この状態でシミュレーションを行うことにより，CPUやバスの負荷の観点でシステムの性能を計測することができる．また，アプリケーションモデルとアーキテクチャモデルを独立に設計できるようになっていることで，マッピングを種々変更しても，仕様記述の書き換えはローカルにしか発生せず，アプリケーションを実装するための最適なアーキテクチャの探索が容易になる．

まとめ

本稿では，システムレベルデザインに有効な計算モデルを紹介して，その活用方法について解説を行った．

本稿で解説したKPNやSDFは，マルチメディア系やデータ処理系のSoCの機能仕様を抽象度が高いレベルで記述するのに向いている点で近年注目されており，本特集の「システムレベル設計フローと設計言語」で解説されている機能検証フェーズで特に有効である．また，SPADEのように仕様設計のフェーズとアーキテクチャ設計のフェーズとのインタオペラビリティを高める観点でも，仕様設計をフォーマルな計算モデルを使って記述しておくことのメリットは高い．また，KPNやSDFについては，これらをサポートする設計ツールが現れつつある．

CFSMやFSMDなどは，仕様設計とアーキテクチャ設計の双方をカバーする計算モデルであり，従来からハードウェア・ソフトウェア協調設計ツールや動作合成ツールのフレームワークとして活用されてきている点でハードウェア設計者にはなじみが深いモデルであるが，より上流のシステムレベルでの仕様を表現するために用いる計算モデルとして相応しいかは疑問である．

MTGのようなモデルは，豊かな表現力によりさまざまなシステムをモデル化可能である一方で，計算モデルが複雑すぎる嫌いもあり，記述の目的に応じたガイドライン等の整備も必要であろう．またMTGを利用した消費電力見積り等の幅広い研究がなされているが，実用ツール開発はまだこれからといえる．

このように，システムレベルでの設計技術の進歩と，設計生産性の向上を達成する上でモデル化のための諸技術は重要かつ不可欠な要因と言えるが，特に実設計者のレベルにおいて計算モデルに対する知識と理解およびその活用がどこまで進んでいるかに関しては疑問視される．本稿が，計算モデルの重要性を再認識する一助となれば幸いである．

謝辞　本稿の執筆にあたって，JEITA EDA技術専門委員会SLD研究会の入江將裕，菰田浩，牧野真，山口雅之，山元浩幸，李建道の各氏にご協力をいただきました．

参考文献

1）Lee, E. A. and Sangiovanni-Vincentelli, A. : A Framework for Comparing Models of Computation, IEEE Transaction on CAD, Vol. 17, No.12（1998）.
2）Harel, D.: Statecharts: A Visual Formalism for Complex Systems, Science of Computer Programming, Vol.8, No.3（1987）.
3）Gajski, D. D., Dutt, N. D., Wu, A. C. and Lin, S. Y. : High-Level Synthesis: Introduction to Chip and System Design, Kluwer Academic Publishers（1992）.
4）Girault, A., Lee, B. and Lee, E. A. : Hierarchical Finite State Machines with Multiple Concurrency Models, IEEE Trans. on CAD, Vol.18, No.6（June 1999）.
5）Lee, B. : Specification and Design of Reactive Systems, Ph.D. Thesis, Electronics Research Laboratory, Univ. of California, Berkeley（May 2000）.
6）R Hoare, C. A., 吉田信博訳: ホーアCSPモデルの理論（C. A. R Hoare : Communicating Sequential Processes），丸善（株）（Feb. 1992）.
7）Kahn, G. : The Semantics of a Simple Language for Parallel Programming, Proc. of the IFIP Congress 74（1974）.
8）Lee, E. A. and Messerschmitt, D. G. : Synchronous Data Flow, Proc. of the IEEE, Vol.75, No.9（Sep. 1987）.
9）Sgroi, M., Lavagno, L. and Sangiovanni-Vincentelli, A.: Formal Models for Embedded System Design, IEEE Design & Test of Computers（April-June 2000）.
10）Balarin, F., Chiodo M. et al.: Hardware-Software Co-Design of Embedded Systems: The Polis Approach, Kluwer Academic Press（June 1997）.
11）Thoen, F. and Catthoor, F.: Modeling, Verification and Exploration of Task-Level Concurrency in Real-Time Embedded Systems, Kluwer Academic Publishers（2000）.
12）Lieverse, P. et al.:A Methodology for Architecture Exploration of Heterogeneous Signal Prossessing Systems, J. of VLSI Signal Processing, Vol.29（2001）.

（平成16年4月8日受付）