共有資源モデル記述を用いた実時間制御システムのモデリング手法

田中 輝明[1,2,a)]　Salita Sombatsiri[2,b)]　武内 良典[2,c)]　今井 正治[2,d)]

概要：近年の実時間制御システムにおいては，その基幹となる制御機能と共にネットワーク通信等の各種サービス機能を搭載した実時間制御用コントローラの実現が必要とされている．その実現検討では，ハードウェア資源の共有を積極的に行い，確度の高い性能見積もりが可能なアーキテクチャ設計手法が必要である．本稿では，実時間制御システムを制御するためのコントローラ・アーキテクチャの検討を効率的に実施可能な共有資源モデル記述を用いたシステム・モデリング手法を提案する．本手法は，既存のモデル化手法では扱いが困難であったアーキテクチャ上の資源制約を設計者が理解しやすい記述によりモデル化し，コントローラを含む対象システムのシミュレーションによる動作確認と性能評価を可能とするものである．本手法により並行処理動作，時間指定動作，および資源共有に伴う並行プロセスの逐次的実行をシステムアーキテクチャ検討の初期段階で実施し，確度の高いシステムの動作確認および性能評価が可能となる．

キーワード：システムレベル設計，実時間制御用コントローラ，組込みシステム，共有資源，システムシミュレーション，SystemC

A Modeling Method for Real time Control System using Shared Resource Desctiption

Teruaki Tanaka[1,2,a)]　Salita Sombatsiri[2,b)]　Yoshinori Takeuchi[2,c)]　Masaharu Imai[2,d)]

Abstract: This paper studies a modeling method for real time cotrol system using shared resource models. Functions of mordern real time control systems require not only conventional real time controls but also other services such as network communication functions. In order to implement these real time control sytems, hardware sharing is indipensable technique and accurate estimation in early design stage becomes more importatnt. This paper proposes a modeling method for real time control system using shared resource models. Proposed modeling enables compact design description to designer, and enalbes to estimate functional validaton and performance evaluation. Experimental result shows that prposed model can model target real time control system compactly in early design staget, and estmiate accurate performace evaluation.

Keywords: System-level Design, Realtime Controller, Embedded System, Shared Resource, System Simulation, SystemC

1. はじめに

近年，M2M (Machine to Machine), IoT (Internet of Things) や CPS (Cyber-Physical Systems) などで用いられる装置はネットワーク通信機能を持ち，相互のデータ交換やクラウド環境のような上位の大規模計算機との間での通信機能が求められている．これらのシステムでは，高速・大容量のネットワーク通信を使用し，様々な計算リソースと有機的に接続することで，より知的な制御を行い得る産業用機器の実現が期待されている．

産業用途において，物理世界からのデータセンシングや物理世界への制御を行う代表的な基幹装置としては，PLC (Programmable Logic Controller）のような実時間制御を行う産業用コントローラがある．一般的にプログラマブルな産業用コントローラは，ある時刻でのスナップ情報とし

1　三菱電機株式会社 先端技術総合研究所
8-1-1 Tsukaguchi-Honmachi, Amagasaki, Hyogo 661-8661, Japan
2　大阪大学 大学院 情報科学研究科
1-5 Yamadaoka, Suita, Osaka 565-0871, Japan
a)　tanaka.teruaki@ak.mitsubishielectric.co.jp
b)　s-salita@ist.osaka-u.ac.jp
c)　takeuchi@ist.osaka-u.ac.jp
d)　imai@ist.osaka-u.ac.jp

て全入力データを取り込み，それら入力データと現内部状態に基づきプログラム記述された制御処理を実行し，その結果を規定時刻で一斉に外部出力することでリアルタイム性を有する制御機能を実現している．

また近年の実時間制御用コントローラが搭載するネットワーク通信機能においては，産業用途としても Ethernet の使用要求が増加し，一般的な IP ベースの TCP / UDP に基づくアプリケーション・プロトコルの使用が増加している．その実現に必要なプロトコルスタックは，処理内容の複雑さや継続的な追加プロトコル対応への必要性から，CPU を用いた SW 処理として実現されることが多い．

このような実時間制御用コントローラに必要とされる制御機能と通信機能とを同一の電子システム・プラットフォームで実現する場合，使用する HW コンポーネント間で様々な競合および干渉が発生する．例えば，制御と通信の両処理が共に CPU を使用する場合，機能によっては競合が生じる．またネットワーク通信で使用するパケットデータを保持するメモリ領域では，アプリケーション処理から送信データを書き込むのと同じタイミングでプロトコルスタックが受信データを書込む場合，バッファメモリへのアクセス競合が発生する．プロトコルスタックが使用する送受バッファメモリは兼用するケースが多く，通信内容で変動するバッファサイズと非同期で発生する送受信要求タイミングにより，その競合・干渉状況は変動する結果となる．このような競合や干渉の発生，またその状況の複雑さにより，実時間制御システムを実現するコントローラ開発においては，その設計検討時に制御性能と通信性能を確定的に規定することは大変困難になってきている．

本問題の解決のため本稿では，共有資源モデル記述を用いたシステムモデリング手法を提案する．提案する手法はシステムレベル記述言語である SystemC に対して，共有資源のモデル化要素を，その関連する時間指定動作や資源獲得方法などの実現と共に追加したものである．本稿にて説明する本共有資源モデルは，処理動作を実現する前提となるプロセッシングエレメント (PE) への処理割り当てにより発生する処理間の PE 排他に着目したものである．共有資源モデルは処理の主体となる SystemC プロセス間で共有され，処理に対して既定した実行時間を，共有資源を獲得している間は消費し，獲得できない間は消費を停止する機能を有している．この共有資源モデルにより，実時間制御システムを実現するコントローラ・アーキテクチャ上に発生する処理実行時の競合・排他要素を簡潔かつ明確に表現可能となり，コントローラ設計の初期段階で適切なアーキテクチャ構成を検討・評価および確定することが可能となる．

本稿の構成は以下の通りである．続く 2 節で関連する先行研究について紹介し，3 節で提案する共有資源モデルによるシステムモデリング手法の説明を行う．4 節では提案するシステムモデリング手法を用いたコントローラ・アーキテクチャモデルとその評価手法および結果について説明する．5 節では提案するシステムモデリング手法に関する考察を記載する．最後に 6 節にて本稿内容に関するまとめを記載する．

2. 先行研究

IoT，M2M，CPS の領域においてはネットワーク通信を用いた産業システムの変革が検討されている．文献 [1] では製造産業に特化した CPS において，SoS (System of Systems)，IoT，Cloud Technology，Big Data や Industry 4.0 等により，新しい Manufacturing System が出現可能であることが示されている．

産業システムの実現形態は “Field level”, “Control level”, “Operations management level” で階層化される．システムの基幹となる制御装置は Control level にある PLC 等の実時間制御用コントローラであるため (文献 [2],[3])，ネットワーク親和性の高い実時間制御用コントローラの実現が必要とされている．文献 [4] ではネットワーク通信機能を持ちマルチコア/メニーコア CPU を用いた PLC の実現検討がなされている．同文献ではその実現において “SCT (Scan Cycle Time) “ と “HTC (high-throughput communication) “ が，PLC アーキテクチャの主要な KPI (Key Performance Indicator) であり，アーキテクチャ上で PLC 特有であるスキャン処理の時間を評価可能であることが重要であることが示されている．また文献 [5] では主要なネットワークと想定される Ethernet に着目し，Ethernet ベースのネットワークに対応した PLC のハードウェア設計について検討されている．また文献 [6] では，ネットワークアプリケーション例として様々なリモート環境から PLC が保有するデータ読出しアクセスを行うための Web サーバ機能を搭載した PLC の実現検討がなされている．しかしながら，これらの設計検討は，特定のハードウェアデバイスの使用を前提とし，その特定のハードウェアを元に詳細な性能見積もりを実施する手法であり，実時間制御用コントローラ設計の初期段階であるハードウェア部品選定を含む設計プロセスで使用するには抽象度が低い手法となっている．また，製品価格や実行性能等の様々な要因に対して最適なハードウェア設計を行うためには，想定するハードウェアデバイスの組合せごとに抽象度の低い手法での見積もりを実施する必要があり，設計工数が大きくなるという問題があった．

そのような設計作業の効率化のため，モデルベース設計手法の導入も検討されている．例えば文献 [7] では，IEEE 1666 として規定されたシステムレベル記述言語である SystemC を用いた Industrial Automation System のシステム設計が検討されている．また文献 [8] では，FMI (Functional Mock-up Interface) と SCNSL (SystemC Network

Simulation Library) を用いた分散制御システムの協調シミュレーション手法が検討されており，文献 [9] では，CPS に対して SystemC を用いた Ethernet 通信のモデル化によるシステム実現が検討されている．同様に文献 [10] では，SystemC を用いたデジタル信号処理のシステム設計を行うための方法論と，その中で共有資源を扱い際に発生する競合状態やプリエンプションに関する必要性が提示されている．SystemC も言語規定で標準的なセマフォやミューテックスは定義されているが，その機能は限定されており，時間待ちと連動した排他制御の実現や排他資源獲得時のポリシィ（FIFO やプリエンプション等）を実現する機能はなく，アーキテクチャ検討時における処理実現で必要となる排他モデルを直接的に表現し，また実現する手段を有していない．

3. 共有資源モデルを用いたシステムモデリング手法

実時間制御用コントローラのアーキテクチャ評価において，その主要な KPI は SCT (Scan Cycle Time) と HTC (high-throughput communication) であり，それら性能を定量的に評価するための “時間” を陽に扱うモデル化要素が必要である．またそのアーキテクチャ実現において Cost-Effectiveness を考慮する場合，CPU/メモリ/バス等の様々な HW リソースはアーキテクチャ上で最大限共用する必要がある．特に近年は実現コストを強く考慮する必要があり，独自かつ専用 IC は出来るだけ使用せず，COTS (Commercial Off-The-Shelf) LSI である MCU (Micro Control Unit) を使用し，MCU を核としたアーキテクチャが多くの実現検討において必要となっている．

近年の MCU では，Ethernet 通信機能を搭載した製品が多く，また各ベンダにおける MCU のバリエーション数も幅広く，その選択肢は広がっている．各 MCU の内部アーキテクチャや回路機能，例えばバスのデータ幅や周波数，内蔵メモリ容量やアクセスポート数，ネットワークコントローラ機能や関連 DMA 機能，外部バス IF や関連 DMA 機能等は少しずつ異なっており，その差がコントローラにおける性能差やコスト差を与える結果となる．そのため，MCU 差異を簡潔にモデル化し，上記 KPI を簡潔に評価し，最適な選択肢を得ることがコントローラ・アーキテクチャ設計のポイントとなる．

電子システムアーキテクチャを検討する場合，SystemC を用いたシステムレベル設計手法が多く使用されている．SystemC を用いることで対象システムの

- 並列動作
- 明確な時間動作や時間待ち動作
- データ通信
- イベント
- アルゴリズム処理

を簡潔にモデル化し，シミュレーション実行することが可能となる．しかしながら SystemC では，共有資源利用に対し発生する排他処理と排他による非実行時の残時間待ち動作を簡潔に表現かつ実現するモデル要素は提供されていない．そのため資源制約の多いシステムのアーキテクチャの検討では，多くのモデル要素を開発する必要があり，その使用は容易ではなかった．

これらの課題を解決すべく本稿では共有資源を持つシステムモデリング手法を提案する．以下に本手法の詳細内容と，開発したモデルライブラリの実現方式およびその動作について説明する．

3.1 システムレベル共有資源モデル

共有資源を持つシステムモデリングに必要となる要素としては，
(1) 構造的なシステム要素 (CPU, メモリ等)
(2) 処理時間や開始時間等の時間制御
(3) データ通信量等のデータサイズ
(4) 複数処理の並行実行動作
(5) 複数処理間での資源排他
(6) 資源排他での待ちに関する詳細動作
が考えられる．ここで (1) から (4) までは SystemC 言語がサポートする基本モデルをそのまま使用可能である．そのためサポートされていない (5) および (6) を実現する必要がある．また共有資源モデルの設計方針としては，各動作モデルに対して簡潔に資源の割当と割当解除を行えるモデル特性を持つことが設計効率化のために必要である．そのため，様々なアーキテクチャ上の制約を容易に表現可能かつ構成可能であることを目的とし，図 1 に示すように，共有資源モデルを表現できるように SystemC を拡張した．

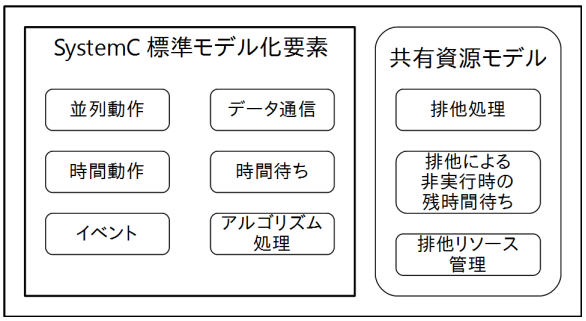

図 1 SystemC 標準モデル化要素の本提案での拡張モデル

提案する共有資源モデリング手法の概要を図 2 に示す．ここで slm_resource は例えば CPU の様な処理の実現を行うための共有資源モデルである．共有資源を使用する処理等を表現する動作モデルは slm_shared_module である．slm_shared_module は共有資源 slm_resource を参照しアクセス可能である．各動作モデルは，共有資源モデルを使用する処理を実行する際に共有資源の獲得処理となる `lock`

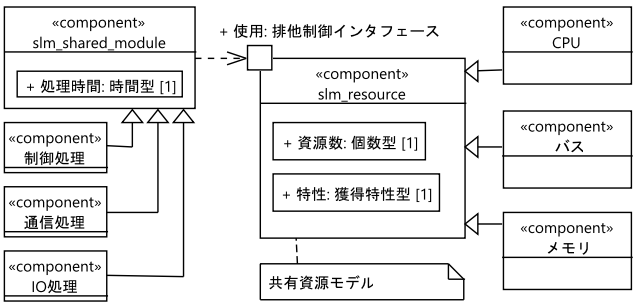

図 2 共有資源モデリング手法

を発行し，動作モデルに既定された処理の実行時間に対応する時間待ちを実施し，処理の実行後には `unlock` を発行し，共有資源の解放を通知する．

共有資源獲得の際，その共有資源が空いていれば動作モデルは処理を継続するが，空きが無い場合には，動作モデルは共有資源の獲得待ちとなる．また複数の動作モデルから共有資源数以上の獲得要求が発行された場合，共有資源数を超える数の動作モデルは資源の獲得待ちとなる．獲得待ちとなっている状態で，他の動作モデルから共有資源が解放され使用可能となった場合には，待ちを行っている動作モデルは資源を獲得可能となる．ただし，どの動作モデルが次に共有資源を獲得可能であるかを規定する必要がある．その資源獲得特性として一般的に想定される FCFS (First-Come-First-Serve) と PRIORITY（優先順位順）とを指定可能としている．FCFS では資源獲得を要求した動作モデルの要求順に沿って資源の割り当てがおこなわれる特性である．また PRIORITY では，資源獲得に対する優先順位を各動作モデルは保有しており，資源獲得を発行した際に既に資源を使用中である動作モデルよりも優先度が高かった場合には，その中で最も低い優先度を持つ動作モデルから共有資源を横取りする．横取りされた動作モデルは，その時点までの時間待ちを消費したとし，処理終了までの残時間を計算し，残時間が存在する場合には自動的に再度資源獲得待ちを発行する．この共有資源の動作に関する意味的な側面を規定するため，slm_resource は同じく本モデル環境で定義している排他制御オブジェクト slm_mutex をその基本特性とする構造としている．slm_mutex は，資源獲得/解放操作の実体となる根本的な `lock` と `unlock` を提供している．

3.2 共有資源モデルの実現

上述したモデルの実現においては，OSCI (Open SystemC Initiative) が提供している SystemC ライブラリをシミュレーションフレームワークとして使用し，共有資源モデルや同資源を使用する動作モデルは，同ライブラリで提供される C++言語の各種クラスを用いて構築している．

そのクラス構造を図 3 に示す．なお本図は UML(Unified Modeling Language) を用いオブジェクト指向モデルとして，そのクラス構造を表現したものである．図 3 において

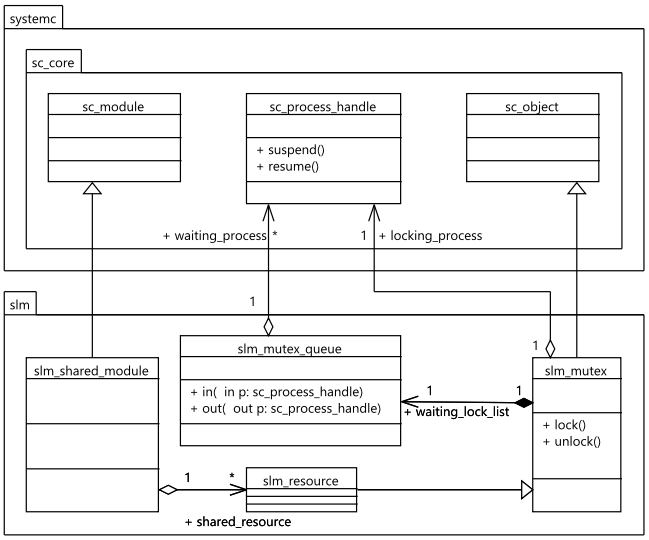

図 3 共有資源に関する基本的なクラス構造

`systemc` パッケージ表現が OSCI が提供する SystemC ライブラリを示しており，`slm` パッケージ表現が今回開発した共有資源モデル実現クラスを示している．

開発した slm パッケージにおいて，動作モデルとなる slm_shared_module クラスは SystemC における基幹動作モデルである sc_module クラスを継承した構造としている．slm_shared_module クラスは，共有資源である slm_resource クラスを共有することが可能となっており，その排他処理としての特性を実現するため，排他制御クラスとして作成した slm_mutex を継承する構造としている．共有資源クラス slm_resource は，その資源獲得を行うための `lock` 操作と，資源解放を行うための `unlock` 操作とを保有しているが，それらは slm_resource の親クラスである slm_mutex が提供する同操作にて規定されている．

本共有資源モデルの実現では，この slm_mutex における `lock`, `unlock` 操作，およびロック中のプロセスに対するロック強制解除に伴う時間待ちの制御が重要な処理となるのため，その詳細について以下で説明する．

3.3 lock,unlock 操作と強制待ち解除の実現

slm_mutex クラスは共有資源を保有している動作モデルを識別すると共に，共有資源の獲得待ちを行っている複数の動作モデルを管理するためのキュー slm_mutex_queue を保有している．また動作モデルの識別には OSCI SystemC ライブラリの sc_core パッケージに含まれるプロセスハンドル sc_process_handle を使用している．sc_process_handle が提供する操作を使用することで slm_mutex での lock 操作および unlock 操作は Algorithm 1 および Algorithm 2 に示す動作として実現している．

lock 操作では，呼び出しプロセスのハンドルを取得し，共有資源が空いていればロックし，呼び出しプロセスをロックプロセスとして登録する．空いていない場合，資源をロックしているプロセスの優先度と，呼び出しプロセスの優先度とを取得する．両者を比較し，呼び出しプロセスの優先度が高い場合には，呼び出しプロセスをロックプロセスとして登録すると共に，資源をロックしているプロセスに対し，sc_process_handle の throw_it メソッドを用いて例外をスローする．その詳細は後述するが，資源をロックしているプロセスは本例外のスローにより，発行中の wait を抜けることが出来る．また，上述の優先度比較において，呼び出しプロセスの優先度が低い場合には，呼び出しプロセスを取得待ちリストに登録し，呼び出しプロセスを Suspend 状態に移行させる．これにより処理の途中でプロセス動作は一時停止し待ち状態に入る．プロセスが再開された場合，その動作として資源をロックし，自身を呼び出しプロセスとして登録する．

Algorithm 1 排他資源の獲得 [lock()]

```
proc_hndl ← sc_get_current_process_handle() { 呼び出しているプロセスのハンドル取得 }
proc_kind ← proc_hndl.proc_kind() { 呼び出しているプロセス種別の取得 }
if SC_NO_PROC_ ≠ proc_kind then
  if !locked then
    locked ← true { 資源使用中を設定 }
    locking_process ← proc_hndl { 使用中のプロセス情報 (プロセスハンドル) を保持 }
  else
    self_priority ← get_priority(proc_hndl) { 呼び出しプロセスの優先度を取得 }
    locked_priority ← get_priority(locking_process) { ロック中のプロセスの優先度を取得 }
    if self_priority > locked_priority then
      cur_proc_handle ← locking_process { 現在ロック中のプロセスハンドルを退避 }
      locking_process ← proc_hndl { 呼び出したプロセスをロック中のプロセスに設定 }
      cur_proc_handle.throw_it(std :: exception) { 現在ロック中のプロセスに例外を発行 }
    else
      waiting_lock_list.in(proc_hndl) { 取得待ちリスト登録 }
      proc_hndl.suspend() { 呼び出したプロセスをサスペンド (待ち状態に遷移)}
      locked ← true { 待ち解除後, 資源獲得プロセスとして自身を設定 }
      locking_process ← proc_hndl
    end if
  end if
  wait(SC_ZERO_TIME) {SystemC カーネル呼び出し }
end if
```

unlock 操作では，同じく呼び出しているプロセスのハンドルの取得後，自身が資源を獲得していた場合，資源獲得を待つプロセスが存在するかどうかを確認し，獲得待ちプロセスある場合には，そのプロセスを Resume 状態とする．

Algorithm 2 排他資源の解放 [unlock()]

```
proc_hndl ← sc_get_current_process_handle() { 呼び出しているプロセスのハンドル取得 }
proc_kind ← proc_hndl.proc_kind() { 呼び出しているプロセスの種別 }
if (SC_NO_PROC_ ≠ proc_kind)&(locked == true)&(locking_process == proc_hndl) then
  rest_proc ← waiting_lock_list.out(nxt_locking_proc) { 資源獲得を待つプロセスを取得 }
  if rest_proc then
    nxt_locking_proc.resume() { プロセスをレジューム }
    wait(SC_ZERO_TIME) {SystemC カーネル呼び出し }
  else
    locked ← false
    locking_process ← nullhndl { 資源を解放 }
  end if
end if
```

なお lock, unlock の両操作共に，その実行は SystemC カーネル上においてアトミックに処理される．また SystemC が管理するプロセス群におけるプロセス切り替えを陽に行うため，SystemC が提供する wait(SC_ZERO_TIME) 操作を両操作の内部で発行している．

共有資源を使用する動作モデルの定義となる slm_shared_module は，使用する共有資源 slm_resource の参照 resource を保有する sc_module のサブクラスである．動作モデルの共有資源の獲得が PRIORITY に基づき実施される場合，共有資源をロックする主体となる動作モデルをその優先度に応じて動的に切り替える必要がある．その実現を行うのが slm_shared_module の waste_processing_time() 操作である．その処理内容を Algorithm 3 に示す．

Algorithm 3 共有資源使用モジュール処理時間発行 [waste_processing_time()]

```
repeat
  resource.lock() { 共有資源の獲得（待ち）}
  begin_time ← sc_time_stamp() { 開始時刻の取得 }
  try{ { 処理の時間待ち発行 }
  wait(proc_time_rest) { 時間待ち }
  end_time ← sc_time_stamp() { 完了時刻の取得 }
  }
  catch(std :: exception& e){ { 待ちが解除された場合 }
  end_time ← sc_time_stamp() { 待ち解除時刻の取得 }
  }
  proc_time_rest − = (end_time − begin_time) { 残処理時間計算 }
until proc_time_rest > sc_time(0, SC_NS)
resource.unlock() { 共有資源の開放 }
```

waste_processing_time() 操作では，まず共有資源の獲得要求を *resource.lock*() にて発行する．共有資源を獲得できなかった場合，獲得まで時間待ちを行う．獲得後，その

時点からの処理時間待ちを行うため sc_time_stamp() にて開始時刻 begin_time を取得する．その後，既定された処理時間分の待ちとして SystemC の `wait()` を発行することで時間待ちに入る．そのまま処理時間待ちを完了した場合は，sc_time_stamp() にて終了時刻 end_time を取得し，残処理時間 proc_time_rest を計算する．その値が 0 となっている場合は，そのままループを抜け，resource.unlock() にて資源を解放し，本処理を抜ける．もし `wait()` を発行している途中に他の動作モデルから `lock()` により資源獲得要求があり，優先度を比較した結果として他の動作モデルに資源が横取りされた場合，`wait()` をスローされる例外で抜け，catch に記述した例外処理に遷移する．例外処理では，それまでの経過時間を計算するため，現時刻を end_time として取得し，開始時刻 begin_time からの差分で残待ち時間を計算し，待ち時間が残っている場合はループ先頭に戻り，再度，共有資源の獲得待ち，および時間待ちに入る．

4. 実時間制御システムのモデル化

前節に説明した内容にて構築した共有資源 (slm_resource) および動作モデル (slm_shared_module) を用いて，対象となる実時間制御システムに対しモデル化を実施した．その概略を図 4 に示す．本システムにおいて実時間制御用コントローラでは 5 つの主要な処理機能（動作モデル）が存在している．それら機能名と処理内容を以下に列挙する:

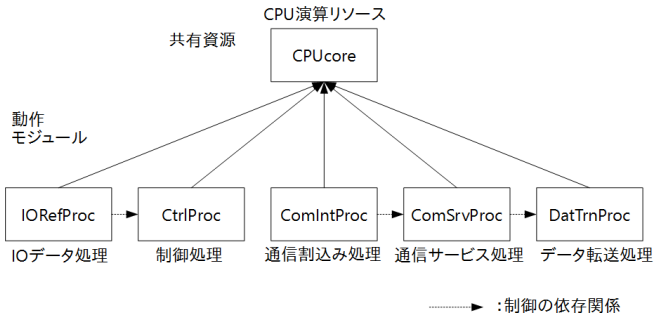

図 4　共有資源モデルを使用したシステムモデル

IO データ処理　スキャンにおける入出力データの更新
制御処理　実時間制御処理
通信割込み処理　外部からの通信要求割込み
通信サービス処理　通信要求に対する応答処理
データ転送処理　通信応答データの送信処理

また，これらの処理間には制御の依存関係が存在しており，

- IO データ処理→制御処理
- 通信割込み処理→通信サービス処理→データ転送処理

の順で処理の実行制御が行われる．またこれら処理は 2 つの処理系統「制御処理系」「通信処理系」にて構成されており，基本的には制御処理系の実行優先度の方が，通信処理系の実行優先度よりも高いとしている．

また，これらの処理は共有資源である”CPU” を使用して動作することとし，その処理時間は各々以下であるとする:

- IO データ処理 (IoRefProc) = 1ms
- 制御処理 (CtrlProc) = 8ms
- 通信割込み処理 (ComIntProc) = 1ms
- 通信サービス処理 (ComSrvProc) = 4ms
- データ転送処理 (DatTrnProc) = 1ms

定周期となる IO データ処理は 15ms 毎に起動され，通信割込み処理は平均的に 13ms 毎に通信要求が発生するとする．

このシステムにおいては，

- CPU を 1 つとし，制御処理系が通信処理系よりも優先度が高い場合，どのようなシステム動作となるか
- CPU が 1 つの場合，制御処理と通信割込み処理との優先度の違いで，どのような動作差異が発生するか
- CPU を 2 つとし，制御処理系と通信処理系を独立して実行させる場合，どのような動作となるか

等の検討を設計初期段階で数多く実施する必要がある．

その検討実施例として，作成したシミュレーションモデルのプログラム記述を参考までにリスト 5 に記載する．リストにおいて，slm_shared_module のインスタンス化における最後の 2 つの引数が，共有資源の指定と，共有資源を使用する場合の優先度を示している．

各種評価は，このプログラムコードの宣言や数値を簡潔に修正し実施した．なお，その動作結果となる動作波形の説明において，各処理の動作状態を数値で出力しているため，その数値が示す状態名とその意味を以下に列挙しておく:

0　RUN：実行中
1　STOP：停止中
2　WAIT_CIN：制御入力待ち
3　WAIT_COUT：制御出力待ち
4　WAIT_RES：リソース空き待ち
5　WAIT_PREE：プリエンプション発生
6　WAIT_PERIOD：周期待ち

4.1 CPU1 つでの通常動作

CPU が 1 つの場合において，制御処理が通信割込み処理よりも優先度が高いと規定して動作させた結果が図 6 である．IO データ処理 (IoRefProc) を 1ms で終了後（0 が RUN 状態），制御処理 (CtrlProc) がすぐに実行され，それが 15ms 周期で繰り返されていることが分かる．また通信割込み処理 (ComIntProc) は時刻 0 で通信割込みが入ったが CPU が使用できないため，制御処理 (CtrlProc) の終了まで処理が待たされていることが見て取れる (4 が WAIT_RES 状態)．その後，動作を開始・終了し，通信サービス処理 (ComSrvProc) を開始するが，その実行の途中で再度通信割込み処理 (ComIntProc) が入ってしまい，通信サービス処理 (ComSrvProc) が待ち状態 (4 の WAIT_RES

```
1  int sc_main(int argc, char* argv[])
2  {
3    // 共有資源(CPU および CPU-EX)
4    slm_resource CPUcore("CPU", PRIORITY);
5    slm_resource CPUcoreEx("CPUcoreEx", PRIORITY);
6
7    // 5つの動作モデルをインスタンス化
8    slm_shared_module<0,1, PERIODIC_OUT, 15,SC_MS>
9      slm_IoRefProc("IoRefProc", CPUcore, 10);
10   slm_shared_module<1,0, FREE_RUN>
11     slm_CtrlProc("CtrlProc", CPUcore,  9);
12   slm_shared_module<0,1, PERIODIC_OUT, 13,SC_MS>
13     slm_ComIntProc("ComIntProc", CPUcoreEx, 8);
14   slm_shared_module<1,1, FREE_RUN>
15     slm_ComSrvProc("ComSrvProc", CPUcoreEx, 7);
16   slm_shared_module<1,0, FREE_RUN>
17     slm_DatTrnProc("DatTrnProc", CPUcoreEx, 6);
18
19   // 処理時間の設定
20   slm_IoRefProc.proc_time    = sc_time( 1,SC_MS);
21   slm_CtrlProc.proc_time     = sc_time( 8,SC_MS);
22   slm_ComIntProc.proc_time   = sc_time( 1,SC_MS);
23   slm_ComSrvProc.proc_time   = sc_time( 4,SC_MS);
24   slm_DatTrnProc.proc_time   = sc_time( 1,SC_MS);
25
26   // 制御依存の既定
27   slm_channel ch0("ch1", 1);   // 制御依存用チャネル
28   slm_channel ch1("ch2", 1);   // 制御依存用チャネル
29   slm_channel ch2("ch3", 1);   // 制御依存用チャネル
30   slm_IoRefProc.outp(0, ch0); // データ処理→IO
31   slm_CtrlProc.inp(0, ch0); // →制御処理
32   slm_ComIntProc.outp(0, ch1);// 通信割込み処理→
33   slm_ComSrvProc.inp(0, ch1); // →通信サービス処理
34   slm_ComSrvProc.outp(0, ch2);// 通信サービス処理→
35   slm_DatTrnProc.inp(0, ch2); // →データ転送処理
```

図 5　共有資源を使用するシステムモデルのプログラム記述

状態) となっていることが分かる．その後，通信サービス処理 (ComSrvProc) を再開・終了し，データ転送処理 (DatTrnProc) を開始しようとするが，そのタイミングで IO データ処理 (IoRefProc) の実行タイミングとなってしまい，データ転送処理 (DatTrnProc) の実行自体が，制御周期を超え，大きく待たされていることが分かる．

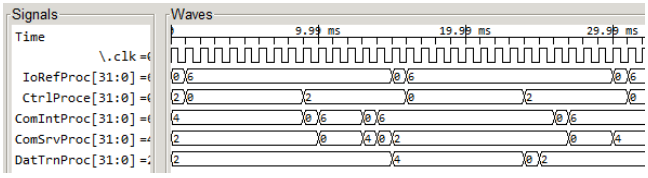

図 6　動作結果波形 1

4.2　CPU1 つで通信割込みが制御処理よりも優先される場合

CPU が 1 つの場合において，通信割込み処理が制御処理よりも優先度が高いと規定して動作させた結果が図 7 である．この設定は，先のプログラムコードにおいて，制御処理 (CtrlProc) と通信割込み処理 (ComIntProc) との共有資源使用に関する優先度値を逆に設定しシミュレーション動作させた結果である．具体的にはリスト 5 の 11 行目と 13 行目における共有資源獲得の優先度を示す最終引数値を双方を逆に設定している．

通信割込みが入った後の時刻 1ms の所で，両処理の実行順番が変わっていることが分かるが，全体の動作として大きな差が発生していないことが本シミュレーション結果からわかる．ただし，制御処理系では通信割込み処理 (ComIntProc) により実行が待たされるケースが発生するため，処理完了までのジッタが発生する結果となる．

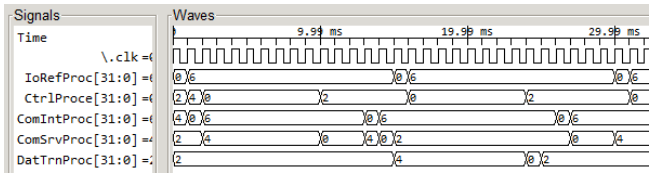

図 7　動作結果波形 2

4.3　CPU2 つでの制御処理と通信処理の実行

本評価は CPU 資源を 2 つ使用可能であるとし，制御処理系と通信処理系とが別の CPU 資源を使用するとした場合の検討である．その動作結果は図 8 に示す通りである．

この設定は，先のリスト 5 において，2 つの独立した CPU 資源である CPUcore/CPUcoreEx を宣言し，制御処理系 (IoRefProc, CtrlProc) のモジュールと通信処理系 (ComIntProc, ComSrvProc, DatTrnProc) のモジュールとのインスタンス化を行う際に，系統ごとに別々の資源を割り当てたものである．なおリスト 5 は，その際のコード記述となっている．

図 8 の動作結果から，所望の通り，制御処理系および通信処理系が，各々共有資源による干渉無く，独立して動作していることがわかる．

しかしながら，本構成は制御性能や通信性能の実現においては最適となるアーキテクチャではあるが，2 つの CPU を使用しての構成であるため，その実現コストは非常に高いアーキテクチャとなっている．

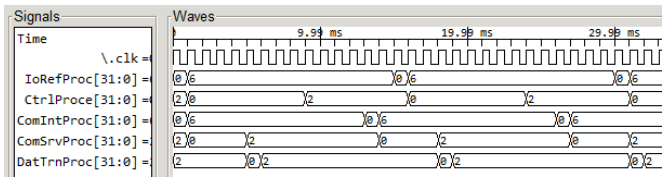

図 8　動作結果波形 3

5. 提案手法に関する考察

このように開発したシステムモデリング手法では，共有資源という HW リソースに関するモデル，およびその資源獲得という動作モデルを導入し，その使用となる割り当ての宣言を簡易に記述することで，共有資源を使用する際のシステム動作を簡潔に記述できることを確認できた．本手法で使用するモデル記述およびシミュレーション実行基盤は SystemC 言語を使用しており，同言語で提供される並行動作，時間指定動作のシミュレーション機能に合わせ，今回提案の共有資源モデルとを組み合わせることにより，必要なシステムの挙動を，設計の初期段階で簡潔に模擬動作させることが可能である．特に，システムにおける必要となる処理機能と処理機能間の制御依存関係を最初に定義し，その後，使用する HW リソースの選定や割り当てに対する数多くの選択肢を評価する場合には，本提案手法による簡潔な操作は設計工数を削減し，結果，より適切なアーキテクチャ選定を実施可能な結果となる．

また共有資源モデルとしては，一つのインスタンスで複数の資源を宣言し，その獲得と解放を同じように扱うようにすることも可能である．これにより，マルチコア/メニーコア CPU における CPU コアの模擬を行うことも可能であり，そのシステムの振る舞いをシミュレーション動作として確認することが可能である．

6. まとめ

本稿では，共有資源モデル記述を用いて実時間制御用コントローラを設計するためのシステムモデリング手法に関する提案を行った．提案したモデリング手法は，CPU のような処理の実現を行うための共有資源を完結に表現できると共に，時間軸上でのシミュレーション動作を実行・評価可能である方式である．

実際に，CPU を共有資源として想定し，SW として実現される複数の処理間で CPU 資源の使用要求を切り替えながら動作するシステム構成を簡潔に記述し，所望の動作を実現可能であることを確認できた．

また複数の共有資源を作成し，複数資源を使用するようなアーキテクチャ上の割り当てを簡潔に指定し，同じくそのシミュレーション動作を簡潔に実行・評価可能であることを確認することができた．

また提案手法では電子システムレベルの設計言語である SystemC のシミュレーションライブラリを基盤として構築しており，通常の SystemC の機能動作と共に提案手法のモデルも動作可能である．そのため本手法を使用するために必要となる学習コストの低減や過去の設計検討用ライブラリの再利用性による設計作業の効率化にも効果があることを確認できた．

参考文献

[1] A Wang, L., Trngren, M., and Onori, M.: *Current Status and Advancement of Cyber-Physical Systems in Manufacturing.* Journal of Manufacturing Systems, 37(Part 2), 517-527. (2015).

[2] Alphonsus, E. R., Abdullah, M. O.: *A review on the applications of programmable logic controllers (PLCs).* Renewable and Sustainable Energy Reviews, 60, 1185-1205.(2016).

[3] Vyatkin, V.: *Software engineering in industrial automation: State-of-the-art review.* Industrial Informatics, IEEE Transactions on, 9(3), 1234-1249.(2013).

[4] Canedo, A., Ludwig, H., Faruque, A., and Abdullah, M.: *High communication throughput and low scan cycle time with multi/many-core programmable logic controllers.* Embedded Systems Letters, IEEE, 6(2), 21-24. (2014).

[5] Wang, B., and Wang, G.: *A new intelligent programmable logic controller based on switched networks.* In Natural Computation (ICNC), 2013 Ninth International Conference on (pp. 1712-1717). IEEE. (2013).

[6] Mahato, B., Maity, T., and Antony, J.: *Embedded Web PLC: A New Advances in Industrial Control and Automation.* In Advances in Computing and Communication Engineering (ICACCE), 2015 Second International Conference on (pp. 156-160). IEEE.(2015).

[7] Fennibay, D., Yurdakul, A., and Sen, A.: *A Heterogeneous Simulation and Modeling Framework for Automation Systems.* Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 31(11), 1642-1655. (2012).

[8] Pedersen, N., Madsen, J., and Vejlgaard-Laursen, M.: *Co-Simulation of Distributed Engine Control System and Network Model using FMI & SCNSL.* IFAC-PapersOnLine, 48(16), 261-266. (2015).

[9] Zhang, Z., and Koutsoukos, X.: *Modeling Time-Triggered Ethernet in SystemC/TLM for Virtual Prototyping of Cyber-Physical Systems.* In Embedded Systems: Design, Analysis and Verification (pp. 318-330). Springer Berlin Heidelberg. (2013).

[10] Christian Haubelt, Joachim Falk, Joachim Keinert, Thomas Schlichter, Martin Streubhr, Andreas Deyhle, Andreas Hadert, and Jrgen Teich.: *A SystemC-based design methodology for digital signal processing systems.* EURASIP J. Embedded Syst. 2007, 1 (January 2007), 15-15.