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

2008－EMB－10 (11)
2008／11／28

動的再構成可能組込みシステムのモデル化と仕様記述

中居 佑輝 †, 山根 智 †,

† 金沢大学大学院自然科学研究科電子情報工学専攻

近年，我々の身の回りには組込み機器があふれている．その中で，今後普及が予想される動的再構成可能組込みシステムに焦点をあて，情報と制御の融合的アプローチにより，階層化されたハイブリッドモデリング言語を提案する．また，提案言語をCPUと動的再構成可能プロセッサの協調動作に適用する．

Modeling and specification of dynamically reconfigurable embedded system

Yuki NAKAI † Satoshi YAMANE †

† Kanazawa University

We focus dynamically reconfigurable embedded systems, and propose the hierarchized hybrid modeling language. We apply our proposed language into co-design of CPU and dynamically reconfigurable processor.

1 まえがき

本稿では，動的に構造が変化していく組込みシステム全体の枠組みを記述できるモデリング言語を提案する．組込みシステムは高い信頼性を要求されながら，低消費電力など厳しいリソース制限が課されている．これらシステムの信頼性・安全性を検証することは非常に重要である．一般に行われている検証は，コーディングしたシステムをシミュレーションツールを用いて検証する方法と，実装後の実機を用いた検証である．しかし，この方法では設計仕様そのものにミスがあった場合にそれを発見することは難しく，またバグを発見できたとしても上流工程から修正を加える必要がある．ここに，設計仕様から検証を行う必要性がある．また設計仕様での検証は，ソフトウェア(SW)とハードウェア(HW)の協調動作をモデル上で考える必要がある．そこで本稿では，動的に構造が変化し，また，SW／HWの協調動作を表現できるモデリング言語を提案する．またその言語を用いて，動的再構成可能プロセッサとCPUの協調動作を仕様記述する．このモデルにより，CPUでの処理(ソフトウェア)と動的再構成可能プロセッサでの処理(ハードウェア)の協調動作を表すことができる．

2 動的再構成可能システム

構成が動的に変化していくシステムを動的再構成可能なシステムという．動的再構成可能システムには，ネットワークからLSIまであり，構造の生成・消滅により統一的にモデル化する．

2.1 ネットワーク

ネットワークの分野における動的再構成とは，ハイブリッドネットワークの構成要素間の通信リンクを動的に生成・消滅させて，ネットワークの構造を動的に変化させるものであり，バイオケミカルプロセス，輸送システムやモジュラ再構成可能ロボットなどがターゲットである．これらの仕様記述手法としては，カリフォルニア大学バークレー校A.Deshpandeらの SHIFT[1] とオランダのアイントホーヘン大学F.Kratzらの R-Charon[2) 3) 4)] がある．しかし，イベント駆動の処理が記述できない．

2.2 動的再構成可能プロセッサ

LSIにおける動的再構成とは，タスクの処理中にプロセッサの構造を処理に最適な構成に変化させていくことで，効率の良い処理を可能とするものである．また，通常ならば処理の種類毎に専用のプロセッサを用意しなくてはならないものを同一プロセッサ上で実現することができる．LSI設計

手法に関する研究は盛んに行われており，慶應義塾大学天野英晴教授らのものが代表的である[5) 6) 7)]．一方，モデル化に関する研究はきわめて少なく，ドイツのパダーボーン大学の J. Teich[8)] や大阪大学潮俊光教授ら[9)] などの LSI をオートマトンでモデル化した離散モデル記述はあるが，SW と HW の協調動作を表すために必要な，様々なプロセッサの速度を表すためのハイブリッドモデル化は行われていない．

3 モデリング言語

本稿では，動的に構造が変化していくシステムの SW／HW 協調動作を記述できるモデリング言語を提案する．提案する言語は，R-Charon[2) 3) 4)] にイベント駆動の表記・動作を拡張した言語であり，同期遷移を可能としたものである．これにより，ネットワーク全体を表すような大きなモデルから，プロセッサレベルの小さなモデルまで，多層に渡って表現できるようになる．提案モデリング言語は以下の 4 つの概念から成る．

- System はシステム全体を表す．
- Structure はシステムを構成する Agent の構造を表す．
- Agent はシステムを構成する Agent を表す．
- Mode は Agent の動作を表す．

以下，言語の定義を説明する．

3.1 System

動的に構成を変化させるシステムを System として定義する．

定義 1 (System の定義)

System $= (S, A)$ である．ここで，

- S は Agent の構造の集合
- A は Agent の集合 □

3.2 Structure

システムの構成要素である Agent の構造 S$\in S$ を Structure として定義する．

定義 2 (Structure の定義)

S$= (TM_{str}, V_{str}, Event_{str})$ である．ここで，

- TM_{str} は最上位層 Mode の集合
- V_{str} は変数の集合

  変数は以下のように分類する．
  - グローバル変数の集合　$V_{gstr} \subseteq V_{str}$
  - ローカル変数の集合　$V_{lstr} = V_{str} \backslash V_{gstr}$
  - アナログ変数の集合　$V_{astr} \subseteq V_{str}$
  - 離散変数の集合　$V_{dstr} = V_{str} \setminus V_{astr}$
  - 参照変数の集合　$V_{refstr} \subseteq V_{dstr}$
- $Event_{str}$ はイベントの集合

  イベントは以下のように分類する．
  - グローバルイベントの集合 $Event_{gstr} \subseteq Event_{str}$
  - ローカルイベントの集合 $Event_{lstr} = Event_{str} \setminus Event_{gstr}$
  - 入力イベントの集合　$Event_{istr} \subseteq Event_{str}$
  - 出力イベントの集合　$Event_{ostr} = Event_{str} \setminus Event_{istr}$ □

3.3 Agent

S$\in S$ の構造を持つ Agent A$\in A$ を Agent として定義する．

定義 3 (Agent の定義)

A$= (TM_{agent}, V_{agent}, Event_{agent}, I)$ である．ここで，

- TM_{agent} は最上位層 Mode の集合

  構造 S の TM_{str} と同等の Mode の集合である．
- V_{agent} は変数の集合

  構造 S の V_{str} と同等の変数の集合である．変数は以下のように分類する．
  - グローバル変数の集合　$V_{gagent} \subseteq V_{agent}$
  - ローカル変数の集合　$V_{lagent} = V_{agent} \setminus V_{gagent}$
  - アナログ変数の集合　$V_{aagent} \subseteq V_{agent}$
  - 離散変数の集合　$V_{dagent} = V_{agent} \setminus V_{aagent}$

- 参照変数の集合　$V_{refagent} \subseteq V_{dagent}$

- $Event_{agent}$ はイベントの集合

  構造Sの $Event_{str}$ と同等のイベントの集合である．イベントは以下のように分類する．

  - グローバルイベントの集合 $Event_{gagent} \subseteq Event_{agent}$
  - ローカルイベントの集合 $Event_{lagent} = Event_{agent} \setminus Event_{gagent}$
  - 入力イベントの集合 $Event_{iagent} \subseteq Event_{agent}$
  - 出力イベントの集合 $Event_{oagent} = Event_{agent} \setminus Event_{iagent}$

- I は各変数の初期値の集合

  構造SからAgent Aを生成する際に与えられる初期値の集合である．□

ここで、参照変数について説明する．参照変数は，あるAgent内から他のAgentのグローバル変数またはグローバルイベントを参照するときに使用する．参照したい変数(イベント)の属するAgentが $ref \in V_{refagent}$ で示されていて，参照したいグローバル変数(イベント)が $v(event)$ であるとき，この変数(イベント)は $ref.v(ref.event)$ として参照できる．Fig.1に例を示す．$\mathrm{ref_1}$ が Agent $\mathrm{A_2}$ を指しており，$\mathrm{A_1}$ から $\mathrm{A_2}$ のグローバル変数 $\mathrm{v_1}$ を $\mathrm{ref_1.v_1}$ として参照できる．

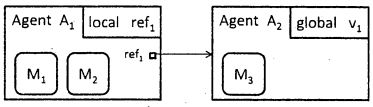

Fig. 1　ref

3.4　Mode

S∈ S の構造を持つ Agent A∈ A の動作を Mode M∈ TM_{agent} として定義する．Agent Aの最上位層 Mode TM_{agent} の要素が複数ある場合，それらのModeは並列に動作する．

定義 4 (Modeの定義)

M= $(E, X, V_{mode}, SM, Cons, Event_{mode}, Reset, ACD, T)$ である．ここで，

- E は entry point の集合

  entry pointは遷移の入り口となるポイントである．

- X は exit point の集合

  exit pointは遷移の出口となるポイントである．

- V_{mode} は変数の集合

  Modeは変数を持っており，以下の種類に分類する．

  - グローバル変数　$V_{gmode} \subseteq V_{mode}$
    グローバル変数はその変数の属するModeと同層のMode，あるいはそのModeの下位層のModeから参照できる．Modeの属するAgentがAであり，Agent Aの変数集合が V_{agent} であるとき，AgentとModeが保有する変数の関係は，$V_{mode} \subseteq V_{agent}$，$V_{gagent} \subseteq V_{gmode}$ である．
  - ローカル変数　$V_{lmode} = V_{mode} \setminus V_{gmode}$
    ローカル変数はその変数の属するMode内からのみ参照できる．
  - アナログ変数の集合　$V_{amode} \subseteq V_{mode}$
  - 離散変数の集合　$V_{dmode} = V_{mode} \setminus V_{amode}$
  - 参照変数の集合　$V_{refmode} \subseteq V_{dmode}$

- SM は下位層Modeの集合

  定義はModeと同じであり，Modeは階層化される．

- $Cons$ は制約の集合

  $Cons$ は不変条件，代数制約，傾き制約の集合から成る．

  - 不変条件
    Modeに留まることのできる条件である．$inv\{g(x_1, x_2, ..., x_n) \sim h(y_1, y_2, ..., y_n)\}$ と記述する．ただし，g, h は多項式であり，$\sim \in \{\leq, <, >, \geq, ==\}$ である．また，読み取り可能な変数の集合を $V_{+mode} = V_{mode} \bigcup V_{refmode}.V_{gagent}$ とし，$x_1, x_2, ..., x_n, y_1, y_2, ..., y_n \in V_{+mode}$ とする．

- 代数制約
  変数がとることのできる値の制約である．$alg\{x \bowtie g(x_1, x_2, ..., x_n)\}$ と記述する．ただし，$\bowtie \in \{\leq, <, >, \geq, =\}, x \in V_{amode}$ である．
- 傾き制約
  変数 x に勾配を割り付ける関数を $d(x)$ とする，この勾配を $diff\{d(x) \bowtie g(x_1, x_2, ..., x_n)\}$ と記述することで制限する．

- $Reset$ は変数値割り当て関数の集合
  エッジに関数 $x := g(x_1, x_2, ..., x_n)$ と記述することで，遷移時に値が割り当てられる．

- ACD は Agent 生成・破棄関数の集合
  ACD は Agent 生成関数の集合 $ACREATE$ と Agent 破棄関数の集合 $ADESTROY$ から成る．

- $Event_{mode}$ はイベントの集合
  Mode はイベントを持っており，以下のように分類する．
  - グローバルイベントの集合 $Event_{gmode} = Event_{mode}$
    グローバルイベントはそのイベントの属する Mode と同層の Mode，あるいはその Mode の下位層の Mode から参照できる．Mode の属する Agent が A であり，Agent A のイベント集合が $Event_{agent}$ であるとき，Agent と Mode が保有するイベントの関係は，$Event_{mode} \subseteq Event_{agent}$ である．
  - 入力イベントの集合 $Event_{imode} \subseteq Event_{mode}$
  - 出力イベントの集合 $Event_{omode} = Event_{mode} \setminus Event_{imode}$
    $a?, b? \in Event_{imode}$，$a!, b! \in Event_{omode}$ であるとき，$a!$ と $a?$，$b!$ と $b?$ がそれぞれ同期して遷移を起こす．

- T はエッジの集合
  $T \subseteq E \times Event_{mode} \times Guard \times ACD \times Reset \times Event_{mode} \times X$
  $e \in E \bigcup SM.X$ から $x \in X \bigcup SM.E$ へのエッジ $t \in T$ は $(e, event_i, guard, \alpha, event_o, x)$ の組から成る．
  - $e \in E \bigcup SM.X$ は遷移の起点となるポイント
  - $event_i \in Event_{imode} \bigcup \{\emptyset\}$ は入力イベント
    ただし，$\emptyset$ の記述は省略できる．
  - $guard \subseteq Guard$ は遷移のガードの集合
    ただし，$\emptyset$ の記述は省略できる．
  - α は $(reset, acd)$ の組から成る．
    * $reset \subseteq Reset$ は変数値割り当て関数の集合
      ただし，$\emptyset$ の記述は省略できる．
    * $acd \subseteq ACD$ は Agent を生成・破棄する関数の集合
      ただし，$\emptyset$ の記述は省略できる．
      集合 α には順序関係があり，例えば，$y := g(x), z := h(y)$ と割り当てた場合，その評価は $y := g(x) \bigwedge z := h(g(x))$ となる．
  - $event_o \in Event_{omode} \bigcup \{\emptyset\}$ は出力イベント
    ただし，$\emptyset$ の記述は省略できる．
  - $x \in X \bigcup SM.E$ は遷移の終点となるポイント □

まず，Agent の生成・破棄について説明する．Agent の生成・破棄オペレータはエッジに割り当てる．

- Agent の生成 $create(v_r, Init, \mathrm{S}) \in ACREATE$
  エッジに $create(v_r, Init, \mathrm{S})$ を割り当てることで，遷移時に S の構造を持った Agent が生成される．ただし，v_r は生成した Agent の参照先を格納する変数．$Init$ は変数への初期値割り当てである．$Init$ を省略した場合は Agent の I が割り当てられる．$V_{str} = \{v_1, v_2, ..., v_n\}, Init = \{Val_{i_1}, Val_{i_2}, ..., Val_{i_n}\}$ であるとき，初期値は $(v_1 := Val_{i_1}, v_2 := Val_{i_2}, ..., v_n := Val_{i_n})$ となる．また，生成時に Agent の TM が活動を始める．

- Agent の破棄 $destroy(v_r) \in ADESTROY$

  エッジに $destroy(v_r)$ を割り当てることで，遷移時に参照変数の指す Agent が破棄される．

次に，イベントについて説明する．

- イベントの優先度

  ある Mode において，異なる入力イベントのエッジがある場合，それらと同期するイベントが同時に出力された時には，あらかじめ決められているイベントの優先度に基づき遷移を行うとする．(Fig.2)

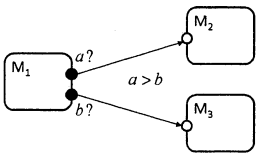

Fig. 2　event priority

- 同種イベントの同時出力

  同じイベントが同時に出力された場合，それらイベントは 1 つのイベント出力として扱う．(Fig.3)

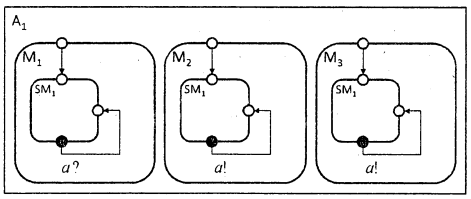

Fig. 3　concurrent

最後に，履歴について説明する．各ポイントは default point($de \in E, dx \in X$) とそれ以外の point に分類される．例外として最上位層 Mode は default entry point の代わりに init point を持ち，default exit point を持たない．Fig.4 を用いて説明する．まず，init から遷移が起きる．そして，M_1 の de から SM_1 の de へ遷移が起きる．ここで，不変条件により M_1 に留まれなくなった時，dx 以外の exit point(則ち x_1) から外の Mode へ遷移をする．その際，遷移前の Mode SM_1 を履歴として記録しておく．その後，de 以外の entry point を介して戻って来る際に，前回の Mode SM_1 へ遷移する．このように，default point 以外の exit point を介した遷移は，遷移前の状態を履歴として記録しておき，default point 以外の entry point を介して戻って来たときに履歴の状態へ遷移する．default point を介した遷移は履歴をリセットして遷移する．

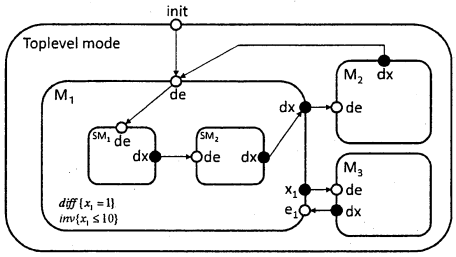

Fig. 4　history of entry and exit point

4　動的再構成可能プロセッサへの適用

本章では，提案言語を動的再構成可能プロセッサに適用する．

4.1　事前準備

本モデルで扱う定数，関数を定義する．モデルでは，タスクの状態を表す文字を定数として扱う．NULL は-1，READY は 1，EXEC は 2，WAIT は 3 である．また，変数値割り当てとして以下の関数を使用する．

- Add($refset_1$,ref_1)

  ref_1 を $refset_1$ に後ろから加えた配列を返す関数

- Del($refset_1$,ref_1)

  $refset_1$ から ref_1 を取り除いた配列を返す関数

- Sel($refset_1$,struct)

  $refset_1$ から構造 struct の Agent への参照変数を取り出す関数

- Max($refset_1$,v_1)

  $ref_i \in refset_1$ の全ての $ref_i.v_1$ の値を比較し，それが最大だったものの ref_i を返す関数

- Min($refset_1$,v_1)

$ref_i \in refset_1$ の全ての $ref_i.v_1$ の値を比較し，それが最小だったものの ref_i を返す関数

- Head($refset_1$)

  $refset_1$ の先頭の参照変数を返す関数

4.2 全体構造

本モデルの全体構造は Fig.5 である．SW，HW の処理はタスクとして扱う．HW は動的再構成可能プロセッサとし，Co-Processor として扱う．SW のタスクは周期的，非周期的に実行されるものとし，その起動タイミングは設計工程で決められる．SW のスケジューリングポリシィは，プリエンプション有の優先度順スケジューリングである．また，HW のタスクは SW のタスクに呼び出され，必要に応じた処理を行い，その結果を呼び出した SW のタスクに返す．その間呼び出し元の SW のタスクは I/O 待ち状態となる．動的再構成可能プロセッサは FPGA を改良したようなもので，1 クロックで構成を変化できるものである．基盤面積さえ足りていれば，同時に複数のタスクを処理することも可能である．その際，プロセッサの周波数は最大遅延に合わせたクロック周波数に適宜変更される．

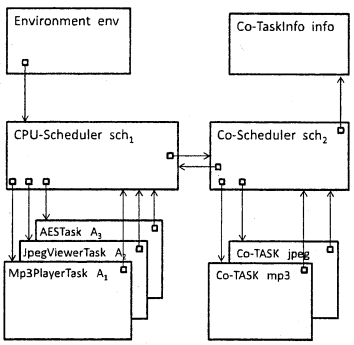

Fig. 5 System

4.3 入力

本節ではモデルの入力となる 3 つの要素を説明する．Agent は四角，Mode は角の丸い四角で記述する．

1. CPU タスク

   タスク設計者が CPU の処理内容を記述する．必要であれば，動的再構成可能プロセッサ (Co-Processor) へタスクを割り当てる動作をここで記述する．(Fig.6 内 Execute と Fig.7)

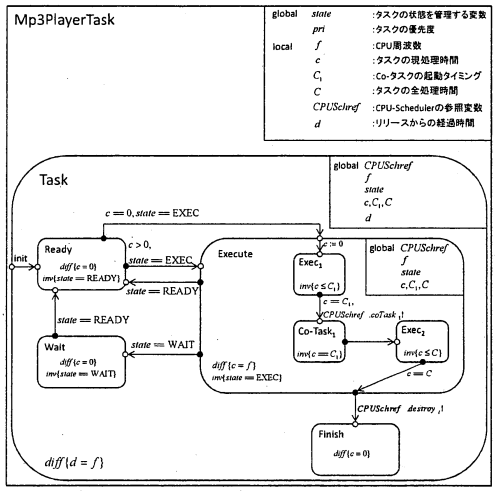

Fig. 6 CPU Task1

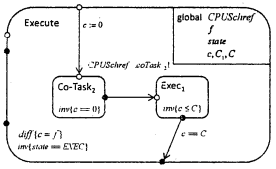

Fig. 7 Execute of CPU Task2

2. 環境（CPU タスク起動タイミング）

   タスク設計者が，各タスクの起動タイミングを記述する．通常は周期タスクとなることが多いが，必要に応じて非周期タスクも記述できる．(Fig.8)

3. 動的再構成可能プロセッサ (Co-Processor) タスク

   LSI 設計者が，動的再構成可能プロセッサで処理される各タスクのパラメータを入力する．パラメータとは，処理に必要なクロック数，クロック周波数 (1 ／最大遅延時間)，必要基盤面積である．(Fig.9)

4.4 動作

ここでは，実際にモデル上でどのような動作をするのか順を追って説明する．

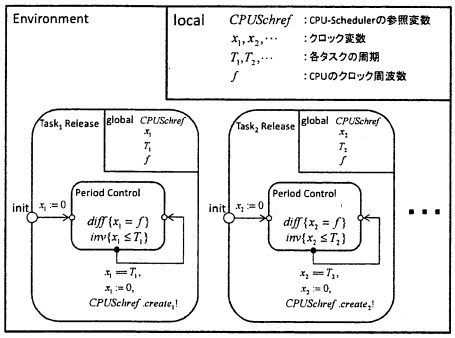

Fig. 8　Environment

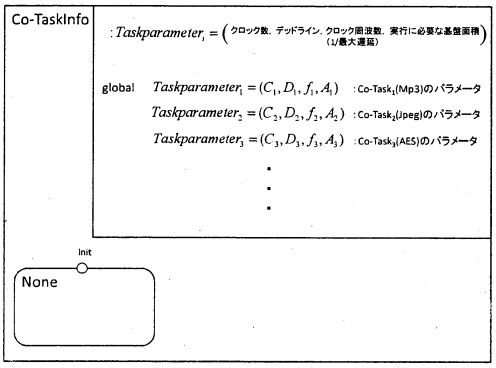

Fig. 9　Co-Task Info

4.4.1　環境

環境はCPUタスクの起動タイミングが入力される.(Fig.8). 本稿では, 周期T_1, T_2で起動するタスクを扱う. タスクの起動はCPUスケジューラにイベントとして伝えられる.

4.4.2　CPUスケジューラ

CPUスケジューラは3つのModeから構成される.(Fig.10) 1つはCPUタスクの生存を管理するModeである. 環境から起動イベントが出力されるとそれに対応したタスクを生成する. また, タスクの処理が終わるとそのタスクを消滅させる. 2つめはディスパッチャのModeである. ディスパッチャは, タスクへのCPU割り当てを管理するものである. 本モデルではプリエンプション有の優先度順スケジューリングを適用した. 3つめは, タスクのI/O処理を管理するModeである. CPUタスクがCo-Processorへ処理を渡したとき, その応答が返ってくるまでそのタスクは待ち状態になる.

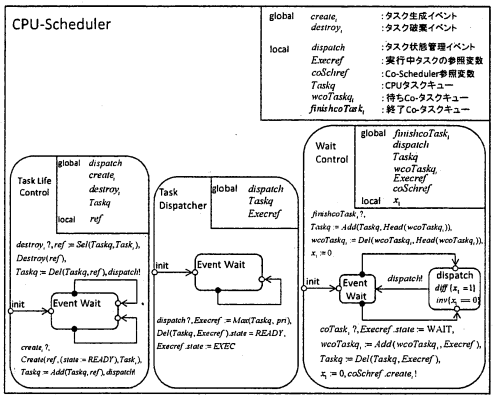

Fig. 10　CPU Scheduler

4.4.3　CPUタスク

CPUタスクは, 実行可能状態, 待ち状態, 実行状態, 終了状態からなる. 特に実行状態はタスク設計者の入力となり, 詳細, 或いは抽象的な内部処理を記述することができる.(Fig.6,7)

4.4.4　動的再構成可能プロセッサスケジューラ

動的再構成可能プロセッサ (Co-Processor) スケジューラは2つのModeからなる.(Fig.11) 1つは動的再構成可能プロセッサタスクの生存管理をするModeである. CPUスケジューラからタスクを割り当てられると, それに対応したタスクを生成し, タスクの処理が終了するとそのタスクを破棄しCPUスケジューラへタスク終了のイベントを出力する. タスク生成時には, タスクの各パラメータを与えてAgentを生成している.(Fig.9,12) 2つめは, タスクの状態を管理するディスパッチャのModeである. 特に動的再構成可能プロセッサは, タスクの処理に最適な構成に変化していくため, 同時に複数のタスクを処理することも可能である. 本モデルでは, 基盤面積の許す限り待ち状態のタスクを処理する方針をとった. また, プロセッサの周波数を処理の内容に最適な周波数へと, 動的に変化させている.

4.5　拡張性

本モデルでは, プリエンプション有の優先度順スケジューリングを採用した. しかし, 到着順ス

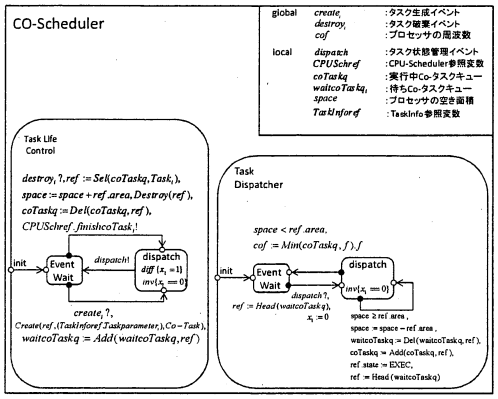

Fig. 11 Co-Scheduler

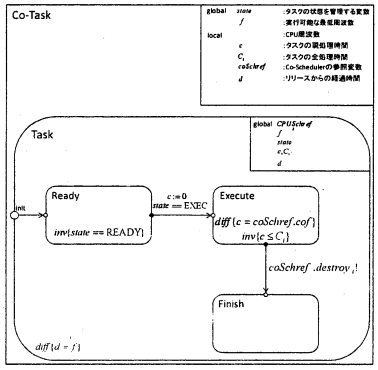

Fig. 12 Co-Task

ケジューリングなど，スケジューリングポリシィを変えたい場合は，構造 CPU-Scheduler の Mode Task dispatcher を変更するだけで適応できる．また，同様に Co-Processor のスケジューリングポリシィも変更できる．

5 おわりに

本稿では，R-Charon の手法を基に，動的再構成可能なシステムのモデリング言語を SW と HW の両面からアプローチできるような言語に拡張した．今後，この言語から自動で検証を行えるような理論を開発することにより，より円滑で無駄のない開発を可能にし，動的再構成システムの普及に貢献できることを望んでいる．

参考文献

1) A. Deshpande, et al. The SHIFT programming language and run-time system for dynamic networks of hybrid systems. IEEE Transactions on Automatic Control 43(4), 1998, pp. 584-587.

2) F. Kratz. A modeling language for reconfigurable distributed hybrid systems. MS thesis, Technische Universiteit Eindhoven, 2005.

3) F. Kratz, et al. R-Charon, a Modeling Language for Reconfigurable Hybrid Systems. LNCS 3927, 2006, pp. 392-406.

4) R. Alur, et al. Compositional modeling and refinement for hierarchical hybrid systems. The Journal of Logic and Algebraic Programming, Vol. 68, Issue 1-2, 2006, pp. 105-128.

5) 桂, 他. ストリームアプリケーションを用いたマルチコア DRP の性能評価. IEICE technical report Vol.106, No.394, 2006, pp. 49-54.

6) V. M. Tuan, et al. Performance Evaluation of Hardware Multi-process Execution on the Dynamically Reconfigurable Processor. IEICE technical report Vol.106, No.247, 2006, pp. 25-30.

7) 天野, 他. 動的リコンフィギャラブルプロセッサにおける可変クロック機構の導入. 電子情報通信学会技術研究報告 CPSY2004-65, Vol.104, No.589, 2005, pp.13-16.

8) J. Teich, M. Koster. (Self-)reconfigurable finite state machines: Theory and implementation. Proceedings of the conference on Design, automation and test in Europe, 2002, pp. 559-568.

9) K. ONOGI, and T. USHIO. Scheduling of Periodic Tasks on a Dynamically Reconfigurable Device Using Timed Discrete Event Systems. IEICE Transactions E89-A(11), 2006, pp. 3227-3234.