(19)日本国特許庁(JP)　(12)公開特許公報(A)　(11)特許出願公開番号
特開2014-102734
(P2014-102734A)
(43)公開日　平成26年6月5日(2014.6.5)

(51)Int.Cl.		F I			テーマコード（参考）
G06F 11/28	(2006.01)	G06F	11/28	340C	5B042
G06F 9/455	(2006.01)	G06F	9/44	310D	5B376
G06F 11/36	(2006.01)	G06F	9/06	620R	
G06F 11/34	(2006.01)	G06F	11/34	S	

審査請求 未請求　請求項の数 18　ＯＬ　（全 38 頁）

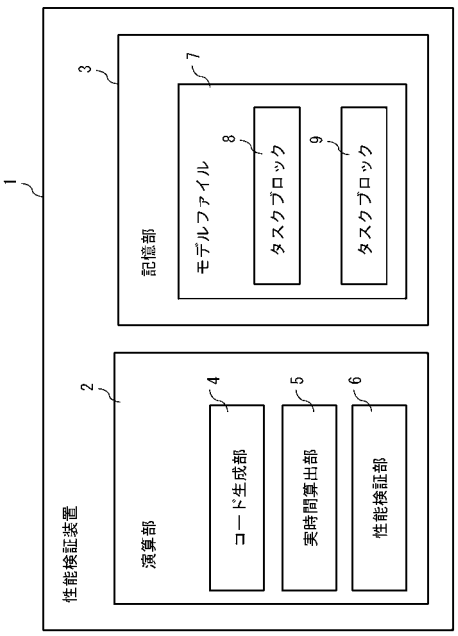
(21)出願番号　特願2012-255141 (P2012-255141)
(22)出願日　平成24年11月21日 (2012.11.21)

(71)出願人　302062931
ルネサスエレクトロニクス株式会社
神奈川県川崎市中原区下沼部１７５３番地
(74)代理人　100103894
弁理士　家入　健
(72)発明者　佐藤　光一
神奈川県川崎市中原区下沼部１７５３番地
ルネサスエレクトロニクス株式会社内
(72)発明者　松井　朋子
神奈川県川崎市中原区下沼部１７５３番地
ルネサスエレクトロニクス株式会社内

Ｆターム(参考)　5B042 GA05　HH07　HH08　MC33
5B376 BC09　BC62　BC67

(54)【発明の名称】性能検証プログラム、性能検証方法及び性能検証装置

(57)【要約】
【課題】マルチコア用途におけるモデルベース開発において、コード生成前における性能検証の精度を向上させること
【解決手段】一実施形態によれば、性能検証プログラムは、演算部において実行され、マルチコアプロセッサ上で実行される制御プログラムの動作をモデル化した複数のタスクブロックを含むモデルファイルを記憶部から読み出し、前記制御プログラムの性能検証を行う性能検証プログラムである。性能検証プログラムは、前記モデルファイルから前記タスクブロックを抽出して、前記タスクブロック毎のコードを生成するコード生成処理と、シングルコアプロセッサにおける前記コードの実行時間の実時間を前記タスクブロック毎に算出する実時間算出処理と、前記タスクブロック毎に算出された前記実時間を用いて、前記制御プログラムの実行過程を検証する性能検証処理と、を行う。
【選択図】図１



【特許請求の範囲】
【請求項１】
演算部において実行され、マルチコアプロセッサ上で実行される制御プログラムの動作をモデル化した複数のタスクブロックを含むモデルファイルを記憶部から読み出し、前記制御プログラムの性能検証を行う性能検証プログラムであって、
前記モデルファイルから前記タスクブロックを抽出して、前記タスクブロック毎のコードを生成するコード生成処理と、
シングルコアプロセッサにおける前記コードの実行時間の実時間を前記タスクブロック毎に算出する実時間算出処理と、
前記タスクブロック毎に算出された前記実時間を用いて、前記制御プログラムの実行過程を検証する性能検証処理と、
を行う性能検証プログラム。
【請求項２】
前記性能検証プログラムは、前記性能検証処理において、前記タスクブロック毎に算出された前記実時間及び前記タスクブロック毎のプロセッサコアの割り当て情報を用いて、前記制御プログラムの実行過程を検証する、
請求項１に記載の性能検証プログラム。
【請求項３】
前記性能検証プログラムは、前記性能検証処理において、マルチコアプロセッサの各プロセッサコアにおける前記タスクブロックの実行開始時間及び実行終了時間を前記実行過程として検証する、
請求項２に記載の性能検証プログラム。
【請求項４】
前記性能検証プログラムは、前記実時間算出処理において、実時間を算出する対象である計算対象タスクブロックを、シミュレータとの連携用ブロックに入れ替えて前記モデルファイルのシミュレーションを実行することにより、前記タスクブロックにおける実時間を算出する、
請求項１に記載の性能検証プログラム。
【請求項５】
前記シミュレータとの連携用ブロックは、前記コード生成処理において前記タスクブロック毎のコードに対応して生成される、
請求項４に記載の性能検証プログラム。
【請求項６】
前記モデルファイルのシミュレーションはシングルコアＰＩＬＳ環境中の統合開発環境におけるシミュレーションであり、前記シミュレータとの連携用ブロックは前記統合開発環境と連携することによりシミュレーションを実行する連携用ブロックである、
請求項４に記載の性能検証プログラム。
【請求項７】
演算部において実行され、マルチコアプロセッサ上で実行される制御プログラムの動作をモデル化した複数のタスクブロックを含むモデルファイルを記憶部から読み出し、前記制御プログラムの性能検証を行う性能検証方法であって、
前記モデルファイルから前記タスクブロックを抽出して、前記タスクブロック毎のコードを生成するコード生成処理と、
シングルコアプロセッサにおける前記コードの実行時間の実時間を前記タスクブロック毎に算出する実時間算出処理と、
前記タスクブロック毎に算出された前記実時間を用いて、前記制御プログラムの実行過程を検証する性能検証処理と、
を備える性能検証方法。
【請求項８】
前記性能検証方法は、前記性能検証処理において、前記タスクブロック毎に算出された

前記実時間及び前記タスクブロック毎のプロセッサコアの割り当て情報を用いて、前記制御プログラムの実行過程を検証する、
請求項７に記載の性能検証方法。
【請求項９】
前記性能検証方法は、前記性能検証処理において、マルチコアプロセッサの各プロセッサコアにおける前記タスクブロックの実行開始時間及び実行終了時間を前記実行過程として検証する、
請求項８に記載の性能検証方法。
【請求項１０】
前記性能検証方法は、前記実時間算出処理において、実時間を算出する対象である計算対象タスクブロックを、シミュレータとの連携用ブロックに入れ替えて前記モデルファイルのシミュレーションを実行することにより、前記タスクブロックにおける実時間を算出する、
請求項７に記載の性能検証方法。
【請求項１１】
前記シミュレータとの連携用ブロックは、前記コード生成処理において前記タスクブロック毎のコードに対応して生成される、
請求項１０に記載の性能検証方法。
【請求項１２】
前記モデルファイルのシミュレーションはシングルコアＰＩＬＳ環境中の統合開発環境におけるシミュレーションであり、前記シミュレータとの連携用ブロックは前記統合開発環境と連携することによりシミュレーションを実行する連携用ブロックである、
請求項１０に記載の性能検証方法。
【請求項１３】
マルチコアプロセッサ上で実行される制御プログラムの動作をモデル化した複数のタスクブロックを含むモデルファイルを記憶部から読み出し、前記制御プログラムの性能検証を演算部で行う性能検証装置であって、
前記モデルファイルから前記タスクブロックを抽出して、前記タスクブロック毎のコードを生成するコード生成部と、
シングルコアプロセッサにおける前記コードの実行時間の実時間を前記タスクブロック毎に算出する実時間算出部と、
前記タスクブロック毎に算出された前記実時間を用いて、前記制御プログラムの実行過程を検証する性能検証部と、
を備える性能検証装置。
【請求項１４】
前記性能検証部は、前記タスクブロック毎に算出された前記実時間及び前記タスクブロック毎のプロセッサコアの割り当て情報を用いて、前記制御プログラムの実行過程を検証する、
請求項１３に記載の性能検証装置。
【請求項１５】
前記性能検証部は、マルチコアプロセッサの各プロセッサコアにおける前記タスクブロックの実行開始時間及び実行終了時間を前記実行過程として検証する、
請求項１４に記載の性能検証装置。
【請求項１６】
前記実時間算出部は、実時間を算出する対象である計算対象タスクブロックを、シミュレータとの連携用ブロックに入れ替えて前記モデルファイルのシミュレーションを実行することにより、前記タスクブロックにおける実時間を算出する、
請求項１３に記載の性能検証装置。
【請求項１７】
前記コード生成部は、前記シミュレータとの連携用ブロックを前記タスクブロック毎の

コードに対応して生成する、
請求項１６に記載の性能検証装置。
【請求項１８】
前記モデルファイルのシミュレーションはシングルコアＰＩＬＳ環境中の統合開発環境におけるシミュレーションであり、前記シミュレータとの連携用ブロックは前記統合開発環境と連携することによりシミュレーションを実行する連携用ブロックである、
請求項１６に記載の性能検証装置。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は性能検証プログラム、性能検証方法及び性能検証装置に関し、例えばマルチコアプロセッサ上で実行される制御プログラムの性能検証プログラム、性能検証方法及び性能検証装置に関する。
【背景技術】
【０００２】
ＭＢＤ（Model Based Design）ツールを使用したＰＩＬＳ（Processor In the Loop Simulation）環境では、シングルコアプロセッサ用のモデルが作成され、それに基づくプログラムのコード生成や動作検証が行われている。
【０００３】
例えば、特許文献１や２には、モデルベース開発手法を用いてプログラムコードを生成する方法や性能予測を行うシステムが記載されている。
【先行技術文献】
【特許文献】
【０００４】
【特許文献１】特開２００３－１７３２５６号公報
【特許文献２】特開２０１１－１５４５２１号公報
【発明の概要】
【発明が解決しようとする課題】
【０００５】
近年、ＭＣＵ（Micro Control Unit）のスピード向上が限界となっているため、組み込みシステム等においてシングルコアＭＣＵからマルチコアＭＣＵへの移行が検討されている。特許文献１～２に記載されている従来技術は、シングルコアプロセッサ用途におけるモデルベース開発の技術である。従って、従来技術では、マルチコアプロセッサ用途におけるモデルベース開発において、コード生成前における性能検証の精度を向上させることができないという問題がある。
なお、その他の課題と新規な特徴は、本明細書の記述及び添付図面から明らかになるであろう。
【課題を解決するための手段】
【０００６】
一実施形態にかかる性能検証プログラムは、演算部において実行され、マルチコアプロセッサ上で実行される制御プログラムの動作をモデル化した複数のタスクブロックを含むモデルファイルを記憶部から読み出し、前記制御プログラムの性能検証を行う性能検証プログラムである。この性能検証プログラムは、シングルコアプロセッサにおけるタスクブロック毎のコードの実行時間の実時間を前記タスクブロック毎に算出する実時間算出処理と、前記タスクブロック毎に算出された前記実時間を用いて、前記制御プログラムの実行過程を検証する性能検証処理と、を特に行う。
【発明の効果】
【０００７】
前記一実施の形態によれば、マルチコアプロセッサ用途におけるモデルベース開発において、コード生成前における性能検証の精度を向上させることができる。

【図面の簡単な説明】
【０００８】
【図１】実施の形態１にかかる性能検証装置の例を示す構成図である。
【図２】実施の形態１にかかる性能検証装置の処理の一例を示したフローチャートである。
【図３】実施の形態２にかかる性能検証装置の例を示す構成図である。
【図４】実施の形態２にかかるマルチコアＰＩＬＳ環境におけるコード生成前の性能検証の一例を示したイメージ図である。
【図５】実施の形態２にかかるマルチコアＰＩＬＳ環境におけるコード生成前の性能検証の一例を示したフローチャートである。
【図６】実施の形態３にかかる性能検証装置の例を示す構成図である。
【図７】実施の形態３にかかるマルチコアＰＩＬＳ環境におけるコード生成前の性能検証の一例を示したイメージ図である。
【図８】実施の形態３にかかるマルチコアコード生成にかかる処理の一例を示したイメージ図である。
【図９】実施の形態３にかかるマルチコアモデルを対象としたオブジェクトコード生成の全体フローの一例を示したフローチャートである。
【図１０】実施の形態３にかかるステップＳ２４の詳細な処理の一例について示したフローチャートである。
【図１１】実施の形態３にかかる機能シミュレーションログの一例を示した図である。
【図１２】実施の形態３にかかる性能シミュレーションログの一例を示した図である。
【図１３】実施の形態３にかかるタスク実行時間計算部が計算した計算結果の一例を示したイメージ図である。
【図１４】実施の形態３にかかる性能シミュレーションログをタスク遷移表示部において表示したときの表示画面の一例である。
【図１５】実施の形態４にかかるマルチコアコード生成部の例を示した構成図である。
【図１６】実施の形態４にかかるマルチコアコード生成部の詳細な処理の一例について示したフローチャートである。
【図１７】実施の形態４にかかるタスクブロックが含まれるモデルの一例を示したイメージ図である。
【図１８】実施の形態４にかかるタスクブロックに関するターゲットコード及びＯＩＬ用情報の一例を示す図である。
【図１９】実施の形態４にかかるリソースブロックが含まれるモデルの一例を示したイメージ図である。
【図２０】実施の形態４にかかるリソースブロックに関するターゲットコード及びＯＩＬ用情報の一例を示す図である。
【図２１】実施の形態４にかかるＡｌａｒｍブロックが含まれるモデルの一例を示したイメージ図である。
【図２２】実施の形態４にかかるＡｌａｒｍブロックに関するターゲットコード及びＯＩＬ用情報の一例を示す図である。
【図２３】実施の形態４にかかるＩＳＲブロックが含まれるモデルの一例を示したイメージ図である。
【図２４】実施の形態４にかかるＩＳＲブロックに関するターゲットコード及びＯＩＬ用情報の一例を示す図である。
【発明を実施するための形態】
【０００９】
以下、図面を参照しつつ、実施の形態について説明する。なお、図面は簡略的なものであるから、この図面の記載を根拠として実施の形態の技術的範囲を狭く解釈してはならない。また、同一の要素には、同一の符号を付し、重複する説明は省略する。
【００１０】

様々な処理を行う機能ブロックとして以下の図面に記載される性能検証装置の各要素は、ハードウェア的には、ＣＰＵ（Central Processing Unit）、メモリ、その他の回路で構成することができ、ソフトウェア的には、メモリにロードされたプログラムなどによって実現される。したがって、これらの機能ブロックがハードウェアのみ、ソフトウェアのみ、またはそれらの組合せによっていろいろな形で実現できることは当業者には理解されるところであり、いずれかに限定されるものではない。

【００１１】

実施の形態１

図１は、実施の形態１にかかる性能検証装置の例を示す構成図である。性能検証装置１は、演算部２及び記憶部３を備える。性能検証装置１は、マルチコアプロセッサ上で実行される制御プログラムの動作をモデル化した複数のタスクブロック（タスク処理関数部分のブロック）を含むモデルファイル７を記憶部３から読み出す。そして性能検証装置１は、制御プログラムの性能検証を演算部２において実行する。まとめると、性能検証装置１は、モデルベース開発手法を用いて生成された制御プログラムの性能検証（特にマルチコアプロセッサにおいて実行される制御プログラムの性能検証）を行うことができる。

【００１２】

演算部２は、性能検証装置１において演算を実行し、記憶部３は、性能検証装置１においてプログラムやデータ等を記憶する。演算部２は、コード生成部４、実時間算出部５及び性能検証部６を備え、記憶部３はモデルファイル７を備える。

【００１３】

コード生成部４は、モデルファイル７からタスクブロック８、９を抽出して、タスクブロック毎のコードを生成する。演算部２は、例えばモデルシミュレーションプログラムを用いることにより、複数のタスクブロックにより構成されるプログラムのモデルファイルを生成することができる。モデルシミュレーションプログラムは、例えばＭＡＴＬＡＢ（登録商標）／Ｓｉｍｕｌｉｎｋ（登録商標）といったＭＢＤツールである。

【００１４】

実時間算出部５は、シングルコアプロセッサにおけるタスクブロック８、９毎のコードの実行時間の実時間を、タスクブロック８、９毎に算出する。この実時間の算出は、プログラムやエミュレータ等により実行することができる。

【００１５】

なお、実時間とは、シングルコアプロセッサでタスクブロックを実行した場合を想定して計算した実行時間のことをいい、実測値とも表現する。以下においても同様に表現する。

【００１６】

性能検証部６は、タスクブロック８、９毎に算出された実時間を用いて、制御プログラムの実行過程を検証する。実行過程とは、例えばマルチコアプロセッサにおける各プロセッサコアが、タスクブロック８、９を実行する開始時間及び実行の終了時間のことをいう。

【００１７】

モデルファイル７は、マルチコアプロセッサ上で実行される制御プログラムの動作をモデル化した複数のタスクブロック８、９を含む。なおモデルファイル７におけるタスクブロックの数は図１では２個が図示されているが、３個以上であってもよい。

【００１８】

図２は、実施の形態１にかかる性能検証装置の処理の一例を示したフローチャートである。以下、性能検証装置１の実行する処理について説明する。

【００１９】

まず、コード生成部４は、モデルファイル７からタスクブロック８、９を抽出して、タスクブロック８、９毎にコードを生成する（ステップＳ１）。次に実時間算出部５は、シングルコアプロセッサにおけるコードの実行時間の実時間をタスクブロック８、９毎に算出する（ステップＳ２）。そして性能検証部６は、タスクブロック８、９毎に算出された

実時間を用いて、制御プログラムの実行過程を検証する（ステップＳ３）。以上のようにして、性能検証装置は性能検証を実行する。

【００２０】

以上の構成より、性能検証装置１は、タスクブロック毎に算出された実時間を考慮した制御プログラムの実行過程の検証をすることができる。この実時間と、制御プログラムを実際に動作させたときのタスクブロックのコードの実行時間との差はほとんどないと考えられるため、性能検証装置１は、コード生成前におけるモデルファイルの性能検証の精度を向上させることができる。換言すれば、モデルファイルから生成したコードをマルチコアにおいて実行したときの正確な性能検証をすることができる。なお、マルチコアプロセッサでは、モデルファイルからプログラムコード（例えばオブジェクトファイル）を生成し、当該プログラムコードをプロセッサに書き込む又は読み込ませることによってモデルファイルにより定義される動作を実行する。以下の説明では、マルチコアプロセッサに与えるコードをマルチコアコードと称し、シングルコアプロセッサに与えるコードをシングルコアコードと称す。

【００２１】

従来、マルチコアコード生成前の段階でマルチコアプロセッサ用のモデルファイルの性能検証を実行する場合には、性能検証に用いるタスクブロック毎のコードの実行時間として、予測値が用いられた。しかしながら、マルチコアプロセッサにおいてプログラムが実行される場合、タスクブロックは複数のプロセッサコアに割り当てられて処理が実行される。そのため、各タスクブロックのコードの実行時間の予測値を用いたマルチコアコード生成前の段階における性能検証の結果と、各タスクブロックのコードの現実の実行時間に基づくマルチコアコード生成後の段階における性能検証の結果とが異なってしまうという課題があった。

【００２２】

実施の形態１にかかる性能検証装置１は、算出したタスクブロック毎のコードの実時間を用いて、マルチコアコード生成前の段階における性能検証を実行することができる。従って、性能検証装置１は従来と比較して、マルチコアコード生成前の段階における性能検証の結果を、マルチコアコード生成後の段階における性能検証の結果により近づけることができる。換言すれば、性能検証装置１は、コード生成前における性能検証の精度を向上させることができる。

【００２３】

なお、性能検証部６は、算出したタスクブロック毎の実時間だけでなく、タスクブロック毎のプロセッサコアの割り当て情報も用いて、制御プログラムの実行過程を検証することもできる。この場合、性能検証装置１は、タスクブロックを複数のプロセッサコアに割り当てた状態を仮定することにより、実際の制御プログラムの動作により近い性能検証をすることができる。そのため、性能検証装置１は、実際の制御プログラムの性能検証をより精度よく実現することができる。特に、性能検証装置１は、プロセッサコア毎の実行性能を検証することができる。

【００２４】

性能検証部６が検証する制御プログラムの実行過程は、例えばマルチコアプロセッサにおける各プロセッサコアが、タスクブロック８、９のコードを実行する開始時間及び実行の終了時間のことをいう。これにより、性能検証部６は、各プロセッサコアにおけるタスクブロックの実行状況を具体的に検証することができるため、制御プログラムについてより具体的な検証ができる。

【００２５】

実施の形態２

実施の形態２においては、実施の形態１において説明した性能検証装置の具体的な構成を説明するとともに、性能検証装置の実行する具体的な処理についても説明する。

【００２６】

図３は、実施の形態２にかかる性能検証装置の例を示す構成図である。性能検証装置１

０は、演算部１１及び記憶部１２を備える。性能検証装置１０は、性能検証装置１と同様、マルチコアプロセッサにおいて実行される制御プログラムの性能検証装置である。

【００２７】

演算部１１は、モデルシミュレーションプログラム２１に基づいてマルチコアプロセッサ用のモデルベース化されたモデルファイル（ソースプログラム）を生成し、その性能検証を実行する。演算部１１は、例えばＣＰＵを搭載したＩＣ（Integrated Circuit）により構成される。演算部１１は、モデル生成部１３、タスク単位コード生成部１４、モデルシミュレータ部１５及び性能検証部１７を備える。演算部１１は、図１における演算部２に対応する。

【００２８】

記憶部１２は、性能検証装置１０においてプログラム及び演算部１１が生成したファイルを格納する部分であり、例えば性能検証装置１０の記憶媒体により構成される。記憶部１２は、プログラム格納部１８及びデータ格納部１９を備える。

【００２９】

モデル生成部１３は、モデルシミュレーションプログラム２１に基づいて、マルチコアプロセッサ用のモデルファイル２２を生成する。モデルファイル２２では、性能検証する対象のマルチコアプロセッサ用の制御プログラムがモデル化されたファイルとして（モデルファイルとして）記載されており、まだコード生成はされていない。ここでモデルファイル２２には、モデル化に応じて、複数のタスクブロック（タスク処理関数部分のブロック）が含まれている。

【００３０】

タスク単位コード生成部１４は、タスク単位コード生成プログラム２３に基づいて、モデルファイル２２におけるタスクブロック毎のコード（タスク単位コード２４）を生成する。タスク単位コード生成プログラム２３はタスク単位コードを生成するのに必要なプログラムであり、例えばプログラム格納部１８に格納されている。タスク単位コード２４は、モデルファイル２２におけるタスクブロックの数だけ生成される。タスク単位コード生成部１４は、図１におけるコード生成部４に対応する。

【００３１】

モデルシミュレータ部１５は、実時間算出部１６を有し、モデルファイル２２のシミュレーションを実行する。具体的には、実時間算出部１６が、実時間計測プログラム２５に基づいて、タスク単位コード２４のシングルコアプロセッサにおける実時間を算出するシミュレーションを実行する。なお、実時間の定義は実施の形態１に記載の通りである。実時間算出部１６は、算出した全てのタスク単位コード２４における実時間を、実時間ファイル２６に記載して出力する。実時間算出部１６は、図１における実時間算出部５に対応する。

【００３２】

性能検証部１７は、実時間ファイル２６に記載された実時間のデータに基づいて、性能検証対象となるマルチコアプロセッサ用の制御プログラムをマルチコアプロセッサで実行した場合を想定した性能検証を実行する。性能検証部１７は、例えばモデルファイル２２をマルチコアプロセッサで実行した場合の総実行時間について算出することができる。以上の構成により、性能検証装置１０は目的とするマルチコアプロセッサ用の制御プログラムの性能検証を実行することができる。性能検証部１７は、図１における性能検証部６に対応する。

【００３３】

プログラム格納部１８には、モデルシミュレーションプログラム２１、タスク単位コード生成プログラム２３及び実時間計測プログラム２５が格納されている。モデルシミュレーションプログラム２１は、モデル生成部１３が生成するマルチコアプロセッサ上で実行される制御プログラムの動作をモデル化したモデルファイルを生成するためのプログラムである。タスク単位コード生成プログラム２３は、モデルファイルに含まれるタスクブロック単位のコードをタスク単位コード生成部１４で生成するのに必要なプログラムである

。実時間計測プログラム２５は、実時間算出部１６でタスクブロック単位のコードの実行時間の実時間を計測するのに必要なプログラムである。

【００３４】

データ格納部１９には、モデルファイル２２、タスク単位コード２４及び実時間ファイル２６が格納されている。モデルファイル２２は、モデル生成部１３が生成した、マルチコア上で実行される制御プログラムの動作をモデル化した複数のタスクブロックを含むモデルファイルである。タスク単位コード２４は、タスク単位コード生成部１４が生成したタスクブロック毎のコードである。実時間ファイル２６は、実時間算出部１６が算出したタスク単位コード２４の実行時間の実時間が記載されたファイルである。モデルファイル２２は、図１におけるモデルファイル７に対応する。

【００３５】

図４は、性能検証装置１０によって実行される、マルチコアＰＩＬＳ環境におけるコード生成前の性能検証の一例を示したイメージ図である。以下、図４を用いて、性能検証装置１０のタスク単位コード生成部１４、実時間算出部１６及び性能検証部１７が実行する処理を詳述する。なお、タスク単位コード生成部１４及び実時間算出部１６は、図４におけるシングルコアＰＩＬＳ環境１００において処理を実行する。

【００３６】

マルチコアＯＳ（Operating System）モデル１０１は、モデル生成部１３によって生成された、タスクブロック１０１－１、１０１－２、・・・、１０１－ｎといったｎ個（ｎは任意の数）のタスクブロックを有するモデルファイルである。マルチコアＯＳモデル１０１は、図４のモデルファイル２２に対応する。

【００３７】

マルチコアプロセッサの場合、ＰＥ（Processer Elements：ここではプロセッサコア）間の排他制御、つまりＯＳを考慮したコード生成を行う必要がある。そこで、マルチコアＯＳモデル１０１の生成においては、ＯＳを考慮したモデリングやコード生成を行うことができるよう、タスクブロックの他に、アラームブロック、ＩＳＲ（Interrupt Service Routine）ブロック及び排他制御ブロックが用意される。モデル生成部１３は、これらのブロックを用いることによりマルチコアＯＳモデル１０１を生成する。従ってマルチコアＯＳモデル１０１は、タスクブロックの他に、アラームブロック、ＩＳＲブロック及び排他制御ブロックがブロックとして含まれる。

【００３８】

タスクブロック、アラームブロック、ＩＳＲブロックには、割り当てるＰＥの情報をパラメータとしてユーザが設定できるように設定されている。コード生成のときには、そのＰＥ情報を元にマルチコアコード生成が実行される。なお、タスクブロック、アラームブロック、ＩＳＲブロック及び排他制御ブロックの詳細な説明は後述する。

【００３９】

まずタスク単位コード生成部１４は、マルチコアＯＳモデル１０１をコピーした後、コピーしたマルチコアＯＳモデル１０１においてマルチコアプロセッサ用に設定されているパラメータ設定をシングルコアプロセッサ用のパラメータ設定に変更する。このようにして、タスク単位コード生成部１４は、シングルコアモデル１０２を自動で作成する。

【００４０】

シングルコアモデル１０２は、マルチコアＯＳモデル１０１におけるタスクブロック１０１－１、１０１－２、・・・、１０１－ｎに対応するタスクブロック１０２－１、１０２－２、・・・、１０２－ｎといったｎ個（ｎは任意の数）のタスクブロックを有する。なおマルチコアＯＳモデル１０１と、シングルコアモデル１０２とは、パラメータ設定のみが異なるだけで、モデルは同一である。そのため、両者のシミュレーション結果は一致する。タスクブロック１０２－１、１０２－２、・・・、１０２－ｎは、シングルコアモデル１０２における基本機能が定義されたブロックである。

【００４１】

次にタスク単位コード生成部１４は、シングルコアモデル１０２の各タスクブロックを

自動で検出し、各タスクブロックのタスク単位コードを生成する。そして実時間算出部１６は、各タスクブロックにおけるそれぞれのタスク処理関数部分をシングルコアＰＩＬＳ環境１００に適用することによって、実測値ファイル１１０をファイルに出力する。この実測値ファイル１１０は、図４の実時間ファイル２６に対応する。

【００４２】

以下、実測値ファイル１１０をファイルに出力するまでの処理について具体的に説明する。まずタスク単位コード生成部１４は、シングルコアモデル１０２を起動する。次にタスク単位コード生成部１４は、シングルコアＰＩＬＳ環境１００において、シングルコアモデル１０２内におけるタスクブロック１０２－１をタスク単位コード生成部１４が有するシングルコアコード生成ツール１０３に適用する。図４においては、タスクブロック１０２－１はタスク単位コード生成プログラム２３に適用される。シングルコアコード生成ツール１０３は、図４におけるタスク単位コード生成プログラム２３に対応する。これにより、タスク単位コード生成部１４は、タスクブロック１０２－１に関する生成コード１０４、ＩＤＥ（Integrated Development Environment：統合開発環境）プロジェクトファイル１０５及び連携用ブロック１０６（ＰＩＬＳ連携用ブロック）を生成する。

【００４３】

生成コード１０４は、タスクブロック１０２－１のシングルコアプロセッサ用のソースコード（シングルコアコード）であり、図４におけるタスク単位コード２４に対応する。プロジェクトファイル１０５は、ＩＤＥ１０７の起動において必要なファイルであり、コードがＩＤＥにおいてどのような状態で動いているか（ＩＤＥの設定環境）が定義されている。連携用ブロック１０６は、実時間算出部１６において実時間を算出するのに用いられるブロックであり、説明については後述する。

【００４４】

タスク単位コード生成部１４が生成コード１０４等を生成処理後、実時間算出部１６は、シングルコアモデル１０２内におけるタスクブロック１０２－１を連携用ブロック１０６に置き換える。換言すれば、シングルコアモデル１０２は、正確なシミュレーションが実行されるために変更される。

【００４５】

次に実時間算出部１６内のＩＤＥ１０７は、プロジェクトファイル１０５を読み込んで起動する。ＩＤＥ１０７は、対象とするタスクブロック１０２－１（ソースコード）をビルド（コンパイラ）することにより、タスクブロック１０２－１におけるオブジェクトコードを生成する。次にＩＤＥ１０７は、生成したオブジェクトコードをＩＤＥ１０７が有するシミュレータ１０８又はエミュレータ１０９にダウンロードする。シミュレータ１０８は、例えばＣｕｂｅｓｕｉｔｅ（登録商標）シミュレータや、その他のマイクロコントローラ（マイコン）用のシミュレータである。シミュレータ１０８又はエミュレータ１０９は、図３における実時間計測プログラム２５に対応する。

【００４６】

ここでＭＢＤツールにおいてシングルコアモデル１０２のシミュレーションを開始（シングルコアモデル１０２を実行）すると、タスクブロック１０２－１がＩＤＥ１０７中のシミュレータ１０８及びエミュレータ１０９と連携する。これにより、タスクブロック１０２－１をシングルコアプロセッサで動作させたときのシミュレーションが実行される。

【００４７】

このとき、上述のように、ＭＢＤツールにおいて動作するシングルコアモデル１０２において、タスクブロック１０２－１が連携用ブロック１０６に置き換えられている。連携用ブロック１０６に置き換えてＭＢＤツールがシングルコアモデル１０２を実行することにより、連携用ブロック１０６はタスクブロック１０２－１のコードがダウンロードされたシミュレータ１０８または、エミュレータ１０９に自身の入力値を入力する。そして、シミュレータ１０８または、エミュレータ１０９はその入力値を取り込み、シミュレーションを行う。シングルコアプロセッサで実行したシミュレーションにおける出力結果は、連携用ブロック１０６から出力結果としてシングルコアモデル１０２にフィードバックさ

れる。この出力結果は、実際にタスクブロック１０２－１を実行した場合の出力結果と同じ結果である。ＭＢＤツールは、この出力結果に基づいてシングルコアモデル１０２を実行する。

【００４８】

まとめると、実時間算出部１６は、実時間を算出する対象である計算対象タスクブロックを、計算対象タスクブロックの入力値をシミュレータ１０８およびエミュレータ１０９に渡してシミュレーションを実行後に結果を出力値としてシングルコアモデル１０２に返すための連携用ブロック１０６（シミュレータとの連携用ブロック）に入れ替えて、モデルファイルのシミュレーションを実行する。これにより実時間算出部１６は、タスクブロックにおける実時間を算出する。

【００４９】

ここで、モデルファイルのシミュレーションはシングルコアＰＩＬＳ環境中のＩＤＥにおけるシミュレーションであり、連携用ブロック１０６はＩＤＥと連携することによりシミュレーションを実行する連携用ブロックである。

【００５０】

以上のようにシミュレーションを実行することによって、タスクブロック１０２－１のコードをシングルコアプロセッサで動作させたときの実測値（実時間）１１０－１が計測される。実測値１１０－１は、実測値ファイル１１０に保存される。ＩＤＥ１０７は、シングルコアモデルの実行を一旦終了する。なお実測値ファイル１１０は、実時間ファイル２６に対応する。

【００５１】

ここでシングルコアモデルにおいては、まだ実行していない（実測値が求められていない）タスクブロックが存在している。

【００５２】

次にタスク単位コード生成部１４は、シングルコアＰＩＬＳ環境１００において、タスクブロック１０２－２をタスク単位コード生成部１４が有するシングルコアコード生成ツール１０３に適用する。これにより、タスクブロック１０２－１と同様にして、タスクブロック１０２－２に関する生成コード、ＩＤＥプロジェクトファイル及び連携用ブロックが生成される。このように、連携用ブロック１０６は、タスクブロックに関するコードが生成される処理において、そのタスクブロック毎のコードに対応して生成される。

【００５３】

次に実時間算出部１６は、タスクブロック１０２－２を連携用ブロック１０６に置き換える。そしてＩＤＥ１０７は、プロジェクトファイル１０５を読み込んで起動し、対象とするタスクブロック１０２－２をビルドすることにより、タスクブロック１０２－２におけるオブジェクトコードを生成する。

【００５４】

次にＩＤＥ１０７は、生成したオブジェクトコードをＩＤＥ１０７が有するシミュレータ１０８又はエミュレータ１０９にダウンロードする。ここでＭＢＤツールがシングルコアモデル１０２のシミュレーションを開始（シングルコアモデル１０２を実行）すると、タスクブロック１０２－２がＩＤＥ１０７中のシミュレータ１０８及びエミュレータ１０９と連携することにより、タスクブロック１０２－２をシングルコアプロセッサで動作させたときのシミュレーションが実行される。

【００５５】

このとき、上述のように、ＭＢＤツールにおいて動作するシングルコアモデル１０２において、タスクブロック１０２－１は実測値の測定が完了しているため、連携用ブロック１０６には置き換えてはいない。ＭＢＤツールは、タスクブロック１０２－２を連携用ブロック１０６に置き換えてシングルコアモデル１０２を実行する。この実行処理については上述と同様である。

【００５６】

以上のようにシミュレーションを実行することによって、タスクブロック１０２－２を

シングルコアプロセッサで動作させたときの実測値１１０－２が計測される。実測値１１０－２は、実測値ファイル１１０に保存される。

【００５７】

ＩＤＥ１０７は、以上の処理をタスクブロック１個ごとにタスクブロック１０２－１～１０２－ｎまで実行することにより、タスクブロックのシミュレーションがＩＤＥ１０７においてｎ回実行される。これにより、実測値１１０－１～１１０－ｎが計測され、実測値ファイル１１０に保存される。

【００５８】

以上のように、シングルコアＰＩＬＳ環境１００で各タスクブロックの処理時間が実測値としてファイルに保存された後、性能検証部１７は、マルチコアＯＳモデル１０１のシミュレーションを実行する。ここで性能検証部１７は、シミュレーション中に性能検証部１７の有するタスク実行情報測定部１１１によりタスク実行情報測定を実行する。

【００５９】

このタスク実行情報測定部１１１において、性能検証部１７は、シングルコアＰＩＬＳ環境１００で測定した実測値１１０－１～１１０－ｎを取り込むとともに、タスクブロック１０１－１～１０１－ｎがマルチコアプロセッサにおいてどのＰＥに割り当てられるかの情報（図示せず）を取り込む。性能検証部１７は、それらの実測値及びＰＥに割り当て情報に基づくシミュレーションを実行することにより得られたシミュレーションログ１１２をファイルに出力する。性能検証部１７の有するタスク遷移表示部１１３は、シミュレーションログ１１２を読み込み、タスク実行状況をＧＵＩ（Graphical User Interface）で表示する。ユーザは、ＧＵＩによりタスク遷移を確認し、性能検証を行う。もし性能検証において問題が発生された場合には、ユーザはマルチコアＯＳモデル１０１のＰＥ割り当てに関して見直しを行う。この性能検証の詳細については実施の形態３で改めて説明する。性能検証において問題がないとユーザが判定した場合に、マルチコアＯＳモデル１０１は初めてオブジェクトコードを生成する。

【００６０】

例えば、性能検証部１７は、実測値１１０－１～１１０－ｎを取り込んでシミュレーションをするモードに切り替わり、シミュレーションログ１１２を測定用のファイルとして出力する。性能検証部１７は、このシミュレーションログ１１２を変換してツールであるタスク遷移表示ツールに読み込ませることにより、タスクがどのタイミングで起動し、実行されているか（タスク遷移）を見ることができる。

【００６１】

図５は、マルチコアＰＩＬＳ環境におけるコード生成前の性能検証の一例を示したフローチャートである。以下、図５において、演算部１１が実行する処理を改めて説明する。

【００６２】

まず、タスク単位コード生成部１４は、マルチコアＯＳモデル１０１をコピーした後、コピーしたマルチコアＯＳモデル１０１においてマルチコアプロセッサ用に設定されているパラメータ設定をシングルコアプロセッサ用のパラメータ設定に変更する。これにより、シングルコア用モデルが生成される（ステップＳ１１）。

【００６３】

以下、タスク単位コード生成部１４及び実時間算出部１６は、シングルコアモデル１０２内のタスクブロック数分、ステップＳ１２～Ｓ１９の処理を実行する。

【００６４】

タスク単位コード生成部１４は、シングルコアモデル１０２を起動する（ステップＳ１２）。次にタスク単位コード生成部１４は、対象とする１つのタスクブロックに対して、シングルコアコード生成ツール１０３を適用する（ステップＳ１３）。これによりタスク単位コード生成部１４は、対象となるタスクブロックに関する生成コード１０４（シングルコアコード）、ＩＤＥプロジェクトファイル１０５及び連携用ブロック１０６を生成する（ステップＳ１４）。ステップＳ１４において連携用ブロックは、タスクブロックに関するコードが生成されるときに、そのタスクブロック毎のコードに対応して生成される。

【００６５】
以下、実時間算出部１６が処理を実行する。まず実時間算出部１６は、対象となるタスクブロックを連携用ブロック１０６に置き換える（ステップＳ１５）。次に実時間算出部１６は、プロジェクトファイル１０５を読み込ませて、ＩＤＥ１０７を起動する（ステップＳ１６）。
【００６６】
ＭＢＤツールがステップＳ１６において変更されたシングルコアモデル１０２を実行し、ＩＤＥ１０７と対象となるタスクブロックとの連携によるシミュレーションを実行する（ステップＳ１７）。
【００６７】
ステップＳ１７が実行されるタイミングにおいて、ここで、タスクブロックの実行時間が計測され、シミュレーション終了後に計測された実測値の情報が実測値ファイル１１０に保存される（ステップＳ１８）。
【００６８】
実時間算出部１６におけるＩＤＥ１０７は、シングルコアモデルの実行を一旦終了する（ステップＳ１９）。
【００６９】
演算部１１は、シミュレーションが実行されていない（すなわち実測値が計測されていない）タスクブロックがシングルコアモデル１０２内に残っていればステップＳ１３に戻り、そのタスクブロックに関してステップＳ１２～Ｓ１９の処理を実行する。シングルコアモデル１０２において、全てのタスクブロックのシミュレーションの実行が完了している場合には、演算部１１は次のステップに移行する。
【００７０】
性能検証部１７は、マルチコアＯＳモデル１０１をシミュレーション実行する。これにより、実測値ファイル１１０が読み込まれることにより、タスク実行情報測定が実行され、シミュレーションログ１１２が生成される（ステップＳ２０）。以上のようにして、マルチコアＯＳモデル１０１のシミュレーション結果であるシミュレーションログ１１２が生成される。
【００７１】
実施の形態２において、演算部１１は、各タスクブロックにシングルコアＰＩＬＳ環境を適用することにより、シングルコアコードを生成し、生成したシングルコアコードを用いたＰＩＬＳ連携シミュレーションを実行することによって実行時間を測定した。これにより、マルチコアＯＳモデル１０１の性能検証において、実測値を用いた性能検証を行うことができる。このように性能検証において実測値を用いることにより、性能検証の精度が向上するため、マルチコアＯＳモデル１０１のマルチコアコード生成後に、コード生成前の段階に戻って修正を実行する「後戻り」を減少させることができる。これにより、マルチコアコード生成にかかる時間を削減することができる。
【００７２】
ＭＢＤツール（例えばＭＡＴＬＡＢ／Ｓｉｍｕｌｉｎｋ）を使用したＰＩＬＳ環境においては、シングルコアプロセッサ用のモデル（シングルコアモデル）が作成され、その作成されたモデルのコード生成、動作検証が実行されている。近年、ＭＣＵのスピード向上が限界となり、組み込みシステムを導入する対象をマルチコアＭＣＵへ移行することが検討されている。そのため、例えばＰＩＬＳ環境においても、マルチコアプロセッサの対応が必要となっている。
【００７３】
特許文献１、２といった従来技術は、シングルコアプロセッサにおけるモデルベース開発の技術であり、マルチコアコードの生成やその検証手法は確立されていなかった。
【００７４】
マルチコアプロセッサの場合、ＰＥ間の排他制御、つまりＯＳを考慮したコード生成を行う必要がある。そこで、ＯＳを考慮したモデリングやコード生成を行うことができるよ

う、タスクブロック、アラームブロック、ＩＳＲブロック、排他制御ブロックが用意される。ユーザは、これらのブロックを使用してマルチコアＯＳモデルを作成する。ここでタスクブロック、アラームブロック、ＩＳＲブロックには、割り当てるＰＥ（プロセッサコア）の情報をパラメータとしてユーザが設定できるようにする必要がある。マルチコアコード生成は、そのＰＥ情報に基づいて実行される。

【００７５】

また、マルチコアプロセッサの場合、どのアルゴリズム（タスクブロック）をどのＰＥに割り当てるかを決定する手段がない。タスクブロックをどのＰＥに割り当てるかは、アルゴリズムの実行時間によって決める必要がある。そこで、タスクブロック毎に、そのタスクブロックの実行時間情報が、パラメータとして設定できるようにする必要がある。従来は、この実行時間情報として、ユーザが予測した値が設定されていた。

【００７６】

以上の設定により、タスクブロックには、ＰＥ情報と実行時間情報が与えられることになる。シミュレータ等によりマルチコアＯＳモデルがシミュレーションされると、タスクブロックに与えられたＰＥ情報と実行時間情報とはシミュレーションログに出力される。シミュレーションログがユーザによってタスク遷移表示装置に読み込まれることにより、マルチコアＯＳモデルにおけるタスク実行状況をユーザがＧＵＩで確認することができる。ユーザはこのタスク実行状況に基づいて、マルチコアＯＳモデルの性能検証を行う。この性能検証において問題がない場合に、マルチコアＯＳモデルにおけるマルチコアコードが生成される。

【００７７】

以上のようなマルチコアＰＩＬＳ環境を想定した場合、タスクブロックのパラメータとして与えられる実行時間は、ユーザの予測値であった。この予測値が用いられることによって、マルチコアコード生成前の検証の精度が低くなってしまい、マルチコアコード生成後の検証で後戻りが発生するという課題があった。この後戻りにより、マルチコアコード生成にかかる時間が増加してしまっていた。

【００７８】

マルチコアプロセッサにおいては、デッドライン（制限時間）までに各プロセッサコアの処理が間に合うか否かが問題となる。特にマルチコアプロセッサにおいては、シングルコアプロセッサと異なり、全実行時間は、全タスクの累積時間とみなすことができない。そのため、何のタスクをどのプロセッサコアに割り当てるとともに、設定されたタスクの実行時間が現実とどの程度近いかが性能検証において重要となる。

【００７９】

換言すれば、各プロセッサコアの処理に要する時間の予測の精度は、設定されたタスクの実行時間の予測の良し悪しに大きく左右されることになる。このため、実測値と異なる予測値が用いられることにより、マルチコアコード生成前の検証の精度が低くなってしまっていた。

【００８０】

実施の形態２においては、シングルコアＰＩＬＳ環境を用いて、タスクブロック毎の実測値を測定している。そして、その実測値に基づくマルチコアコード生成前の検証を実行することにより、マルチコアプロセッサ向けＰＩＬＳ環境におけるマルチコアコード生成前の性能検証の精度を向上し、マルチコアコード生成後の性能検証で後戻りの発生が減少する効果が得られる。実際にタスクブロックの処理にかかる時間と同じ又はほぼ同じである実測値が性能検証において用いられるからである。

【００８１】

また、実時間算出部１６は、実時間を算出する対象である計算対象タスクブロックを連携用ブロック１０６に入れ替えて、モデルファイルのシミュレーションを実行する。これにより実時間算出部１６は、タスクブロックにおける実時間を算出する。このとき、計算対象タスクブロックはマルチコアプロセッサ上における実際の処理と同じ処理を実行するため、実時間算出部１６は、実測値をより正確に計測することができる。

【００８２】
実施の形態２において、連携用ブロック１０６を用いず、対象とするタスクブロックの出力結果として適当な値をシングルコアモデル１０２に設定することによってシングルコアモデル１０２を実行しても、実時間算出部１６は同様に実測値を求めることができる。しかし、対象とするタスクブロックの出力結果が本来の値でない場合には、実時間算出部１６がその値を用いてシングルコアモデル１０２のシミュレーションを実行すると、対象とするタスクブロックの実測値として本来とは異なる結果が得られてしまう可能性がある。従って、実測値をより正確に計測するためには、実時間算出部１６は、連携用ブロック１０６を用いてシングルコアモデルのシミュレーションを実行するのが望ましい。
【００８３】
タスク単位コード生成部１４は、連携用ブロック１０６を、タスクブロックに関するコードが生成される処理において、そのタスクブロック毎のコードに対応して生成する。そのように、タスク単位コード生成処理において連携用ブロック１０６が生成されるため、連携用ブロック１０６を別の処理で生成する必要がなくなり、実測値（実時間）算出にかかる手間を減らすことができる。
【００８４】
実施の形態２において、モデルファイルのシミュレーションはシングルコアＰＩＬＳ環境中のＩＤＥにおけるシミュレーションであり、連携用ブロック１０６はＩＤＥと連携することによりシミュレーションを実行する連携用ブロックである。このため、実施の形態２にかかる性能検証方法は、ＰＩＬＳ環境、特にＭＢＤツールを使用したＰＩＬＳ環境に適用することができる。
【００８５】
その他、性能検証装置１０によって生ずる効果は実施の形態１に記載したものと同様である。
【００８６】
実施の形態３
次に実施の形態３では、マルチコアプロセッサ用のプログラムにおいて、タスクブロックにおいて所定の性能を満たすようにＰＥ（プロセッサコア）をコード生成前に割り当てる方法を説明する。
【００８７】
上述の通り、近年ＭＣＵのスピード向上が限界となっているため、組み込みシステム等においてシングルコアＭＣＵからマルチコアＭＣＵへの移行が検討されている。そのためＰＩＬＳ環境においても、マルチコアプロセッサへの対応が必要となっている。また、ＰＩＬＳだけで性能を検証するのではなく、ＭＩＬＳ（Model In the Loop Simulation）／ＳＩＬＳ（Software In the Loop Simulation）のレベルから、いかにマルチコアプロセッサにおいて良質なソフト配分になっているかを検討できることが必要になっている。従来技術は、シングルコアプロセッサ開発向けのモデルベース環境が開示されており、マルチコアプロセッサ用のモデルベース環境には対応していない。
【００８８】
マルチコアプロセッサのモデルを扱う場合、その複雑さから一般にはＯＳモデルを使用してモデリングを実行する。ここでＯＳモデルとは、ＯＳの動作を考慮にいれたモデルであり、シングルコアプロセッサのモデルとしてはあまり利用されないが、マルチコアプロセッサのモデルとしては便利なモデルである。
【００８９】
このＯＳモデルはタスクブロック、アラームブロック及び排他制御を表現するＭＡＴＬＡＢ／ＳｉｍｕｌｉｎｋのＰｒｏｔｅｃｔｅｄ　ＲＴ（Rate Transition）ブロックと同等のブロックを備える。Ｐｒｏｔｅｃｔｅｄ　ＲＴブロックは、マルチレートのデータの整合性を保持するために利用するブロックで、本実施の形態では排他制御ブロック（リソースブロック）として活用する。
【００９０】

ここで、シングルコアプロセッサを対象にしたモデルベース環境がマルチコアプロセッサを対象としたコード生成において不十分である点（課題）は、大きくいって以下の３点である。

【００９１】

１．性能検証面

シングルコアプロセッサを対象とするモデルの性能評価は、コード生成後のＰＩＬＳ環境において行っていた。しかし、モデルの対象がマルチコアプロセッサとなった場合には、モデル中のどの動作ブロックをどのプロセッサコアに割り当てるかによって、モデルの動作性能が変わる。モデルにおけるプロセッサコアの最適な割り当てを考えるには、割り当て数分のコード生成とＰＩＬＳシミュレーションとを行う必要があるため、多くの性能検証時間を必要とする。

【００９２】

２．コード生成面

コード生成自体はシングルコアモデルにしか対応していない。つまり、各ブロックをどのプロセッサコアへ割り当てるかについては考慮されていない。そのため、コード生成においては、どの動作ブロックをどのコアに割り当てるのかを認識した後、マルチコアプロセッサとして動作するコードを生成する必要がある。

【００９３】

３．ＰＩＬＳシミュレーション面

シングルコアプロセッサを対象としたコード生成においては、ＭＡＴＬＡＢ／Ｓｉｍｕｌｉｎｋとマルチコア仮想環境（シミュレータ）との同期シミュレーションを実現する必要がある。仮想環境上でマルチコアＣＰＵのシミュレーションを実行する場合でも、シングルコアプロセッサと同様に、プラントとの接続（マルチコア仮想環境とＭＡＴＬＡＢ／Ｓｉｍｕｌｉｎｋとの接続）になるＩ／Ｆは一箇所（ポート、メモリ)になるのが一般的なため、考え方は変わらない。従って、マルチコアプロセッサを対象としたコード生成においてＭＡＴＬＡＢ／Ｓｉｍｕｌｉｎｋとマルチコア仮想環境との同期シミュレーションを実現する必要がある。しかし、マルチコアＣＰＵのＰＥ毎の複数のデバッガウィンドウとＭＡＴＬＡＢ／Ｓｉｍｕｌｉｎｋの間で、問題が起こらないデバッガ制御が必要となる。

【００９４】

実施の形態３においては、上述の課題のうち、性能検証面の課題を解決する。実施の形態３にかかる性能検証装置は、性能検証時間を削減するため、コード生成前に性能検証を行うように、性能検証処理を実行するステップの前倒しを行う。

【００９５】

具体的には以下を実施する。

１．ＯＳモデルの改造

実施の形態３におけるＯＳモデルは、シングルコアプロセッサにおけるＯＳモデルに備えられるタスクブロック、アラームブロック、Ｐｒｏｔｅｃｔｅｄ　ＲＴのほか、ＩＳＲブロックを備える。シングルコアプロセッサにおけるＯＳモデルにおいても必要なブロックのほとんどは用意されているが、更なるモデル表現の向上のため、ここではＩＳＲブロックなるものを用意している。そして、以降に行うコード生成とＰＩＬＳのため、各ブロックに以下のパラメータを用意する。

・ブロックに割り当てるＰＥの情報
・タスクブロックのアルゴリズム実行時間

【００９６】

２．性能シミュレーションの仕組みを用意

ＰＩＬＳ前のシミュレーション（ＭＩＬＳ／ＳＩＬＳ）では、遅延時間（実行時間）の概念が存在しない。そのため、一般にはモデルにおける機能が実現できるかどうかを判定する機能シミュレーションしか実施できない。実施の形態３においては、機能シミュレーション中、ブロックの動作を行う度に、ブロックに割り当てられたＰＥ情報、アルゴリズ

ム実行時間を使って、どのくらい時間が経過したかを機能シミュレーションの実行装置とは別の装置で計算する。こうして、性能シミュレーションを設計の早期段階（コード生成前）で行う。

【００９７】

以下、実施の形態３にかかる性能検証装置及び性能検証方法について具体的に説明する。

【００９８】

図６は、実施の形態３にかかる性能検証装置の例を示す構成図である。図６にかかる性能検証装置２７は、図３にかかる性能検証装置１０と比較して、新たにマルチコアコード生成部２０を備える。性能検証装置２７の他の各部が実行する処理は、性能検証装置１０が実行する処理と同一である。

【００９９】

マルチコアコード生成部２０は、性能検証部１７におけるマルチコアプロセッサ用のプログラムの検証結果が問題ない場合に、各ブロックにＰＥ及び動作時間を割り当てたマルチコアＯＳモデルをコンパイルすることにより、マルチコアコード（オブジェクトコード）を生成する。この処理の詳細については後述する。

【０１００】

図７は、性能検証部１７によって実行される、マルチコアＰＩＬＳ環境におけるコード生成前の性能検証の一例を示したイメージ図である。以下、図７を用いて、性能検証部１７が実行する処理を詳述する。

【０１０１】

図７において性能検証部１７は、タスク実行時間計算部１７１及びタスク遷移表示部１７２を備える。タスク実行時間計算部１７１は、性能検証装置２７が備えるＭＢＤツール２００（例えばＭＡＴＬＡＢ／Ｓｉｍｕｌｉｎｋ）をツールとして使用することにより、機能シミュレーションログ２０１、ＰＥ情報２０２、時間情報２０３及びリソース情報２０４を用いて、タスク実行時間を計算する。タスク実行時間計算部１７１は、実施の形態２におけるタスク実行情報測定部１１１に対応し、タスク遷移表示部１７２は、タスク遷移表示部１１３に対応する。

【０１０２】

ここで機能シミュレーションログ２０１は、タスクが処理される順番が定義されたログ（換言すればコード生成前の機能シミュレーション結果）であり、ＭＢＤツール２００が出力して、タスク実行時間計算部１７１に入力される。

【０１０３】

ＰＥ情報２０２は、モデル内のブロックをどのＰＥ（プロセッサコア）で動かすかを示す情報であり、ユーザによって任意に設定される。時間情報２０３は、各ブロックがシングルコアプロセッサでの処理においてどれだけの時間がかかるかを示す情報であり、例えば実施の形態２において計測された実測値ファイル１１０が当てはまる。リソース情報２０４は、タスクの実行において何のリソースを割り当てるかが示された情報である。タスク実行時間計算部１７１は、ＰＥ情報２０２及び時間情報２０３をモデルにおける各ブロックに割り当てて、モデルにおけるタスク実行時間を計算する。

【０１０４】

タスク実行時間計算部１７１は、性能結果である性能シミュレーションログ１７３を出力し、タスク遷移表示部１７２に出力する。性能シミュレーションログ１７３には、モデルにおけるタスクがどのタイミングでどの時間だけ実行されているか（タスク遷移の様子）が示されている。タスク遷移表示部１７２はその性能シミュレーションログ１７３に基づいて、タスク遷移の様子を表示する。ユーザは、そのタスク遷移表示部１７２に表示されたタスク遷移の様子を参照することにより、タスクの実行時間を確認する。すなわち、ユーザは、対象とするモデルの性能が満たされているか否かを検証する。

【０１０５】

なお、図７において説明した性能検証の方法は、図４において説明した性能検証の方法

に対応する。また、性能シミュレーションログ１７３は、図４におけるシミュレーションログ１１２に対応する。

【０１０６】

次に、マルチコアコード生成部２０の処理について説明する。

【０１０７】

図８は、性能検証部１７及びマルチコアコード生成部２０によって実行される、マルチコアコード生成にかかる処理の一例を示したイメージ図である。

【０１０８】

まず性能検証装置２７は、マルチコアＯＳモデル１０１のタスクブロック１０１－１～１０１－ｎにおいて、ＰＥ情報２０２及び時間情報２０３を割り当てる。割り当てたモデルは図８においてマルチコアＯＳモデル２０６として表されており、マルチコアＯＳモデル２０６内のタスクブロック２０６－１～２０６－ｎには、ＰＥの情報及び時間情報が割り当てられている。ＭＢＤツール２００には、こうしてできたマルチコアＯＳモデル２０６と制御対象であるプラントモデル２０５（物理モデル）を有する閉ループモデルが入力される。

【０１０９】

性能検証部１７は、このマルチコアＯＳモデル２０６の性能検証を実行する。この性能検証については図７において説明した通りである。

【０１１０】

性能検証において問題がない場合、マルチコアＯＳモデル２０６はマルチコアコード生成部２０に入力される。マルチコアコード生成部２０は、マルチコアＯＳモデル２０６をＣ又はＣ＋＋のソースコードに変換したマルチコアＳＷ（Software）２０７を出力する。マルチコアＳＷはＣ又はＣ＋＋のコードで記載されたソースコードであり、タスクブロック２０７－１～２０７－ｎは、それぞれＣ又はＣ＋＋のソースコードに変換されている。

【０１１１】

コンパイラ２０９には、マルチコアＳＷ２０７及びマルチコアＯＳ２０８が入力される。コンパイラ２０９は入力に基づいてコンパイラを実行し、マルチコアｏｂｊｅｃｔ２１０（オブジェクトコード）を出力する。これはターゲットとなるマルチコアＣＰＵを対象としたコードとなっている。なおマルチコアＯＳ２０８は、マルチコアプロセッサで実行されるＯＳである。

【０１１２】

性能検証装置２７は、以上により生成されたマルチコアｏｂｊｅｃｔ２１０をシミュレータ２１１に入力するとともに、プラントモデル２０５をＭＢＤツール２１２に入力する。性能検証装置２７内の連携ブロック２１３は、シミュレータ２１１とプラントモデル２０５をコシミュレーションすることにより、ＰＩＬＳを実行することができる。なおシミュレータ２１１はマルチコアプロセッサ用のシミュレータである。なおシミュレータ２１１は、マイコンで代用することもできる。

【０１１３】

なお、図８におけるＭＢＤツール２００とＭＢＤツール２１２とは同じ構成要素である。ＭＢＤツール２００にはデータとして、マルチコアＯＳモデル１０１とプラントモデル２０５が投入され、ＭＢＤツール２１２にはデータとして、プラントモデル２０５だけが投入される。

【０１１４】

図９は、マルチコアＰＩＬＳ環境におけるマルチコアモデルを対象としたオブジェクトコード生成の全体フロー（マルチコアＯＳ作成フロー）の一例を示したフローチャートである。以下、図９において、演算部１１が実行する処理を改めて説明する。

【０１１５】

まず、ユーザはマルチコアＯＳモデル１０１とプラントモデル２０５からなるモデルを作成し、用意する（ステップＳ２１）。マルチコアＯＳモデル１０１は、モデル生成部１３により生成することができる。

【０１１６】
次に、演算部１１はマルチコアＯＳモデル１０１にＰＥ情報２０２及び時間情報２０３を設定する。換言すれば、演算部１１はマルチコアＯＳモデル１０１内のタスクブロックに、対応するＰＥ情報及び時間情報を割り当てる（ステップＳ２２）。これにより、マルチコアＯＳモデル２０６ができる。
【０１１７】
ステップＳ２２において作成されたマルチコアＯＳモデル２０６をＭＢＤツール２００に入力することにより、ＭＢＤツール２００でコード生成前の動作検証（機能検証）を実行する（ステップＳ２３）。この動作検証により、機能シミュレーションログ２０１が生成される。
【０１１８】
ステップＳ２３で動作した機能シミュレーションログ２０１を性能検証部１７に入力し、性能検証部１７はコード生成前におけるモデルの性能検証を実行する（ステップＳ２４）。この性能検証の結果はタスク遷移表示部１７２に表示される。
【０１１９】
なお、ステップＳ２１～Ｓ２４の処理は、実施の形態２にて説明した性能検証の処理に対応する。
【０１２０】
ユーザはタスク遷移表示部１７２に表示された性能検証の結果を確認し、モデルの実行における動作は問題ないか（モデルが正しく実行されているか）、及び性能に問題がないか（全てのタスクが所定時間内に終了しているか）を検証する（ステップＳ２５）。
【０１２１】
ステップＳ２５において、モデルの実行における動作又は性能の少なくともいずれかに問題がある場合には（ステップＳ２５のＮｏ）、演算部１１はユーザの操作に応じて、ステップＳ２１又はステップＳ２２からの処理をやり直す。例えばモデルの実行における動作に異常がある場合には、マルチコアＯＳモデル１０１自体に問題があって正常に動作しないことが考えられる。その場合には、演算部１１は正しいマルチコアＯＳモデル１０１を再度作成することにより、ステップＳ２１からの処理をやり直す。
【０１２２】
性能に異常がある場合、すなわち所定時間内に終了していないタスクがある場合には、ＰＥの割り当てに問題があることが考えられる。又は、ブロック毎に割り当てられた時間が正しくない場合が考えられる。その場合には、ユーザの操作により、演算部１１はマルチコアＯＳモデル１０１に正しいＰＥ情報２０２及び時間情報２０３を設定する。これにより、演算部１１はステップＳ２２からの処理をやり直す。
【０１２３】
なお、モデルの実行における動作に問題がないか否かの判定については、コード生成前の動作検証を実行した後に（ステップＳ２３とステップＳ２４の間に）実行してもよい。
【０１２４】
モデルの実行において動作及び性能のいずれにも問題がない場合には（ステップＳ２５のＹｅｓ）、演算部１１はマルチコアＯＳモデル２０６をマルチコアコード生成部２０に入力することにより、マルチコアＳＷ２０７を生成する。そして演算部１１はマルチコアＳＷ２０７とマルチコアＯＳ２０８をコンパイラ２０９に入力することにより、マルチコアｏｂｊｅｃｔ２１０を生成する（ステップＳ２６）。
【０１２５】
演算部１１はマルチコアｏｂｊｅｃｔ２１０をシミュレータ２１１に入力させ、プラントモデル２０５をＭＢＤツール２１２に入力させる。そして演算部１１は、シミュレータ２１１とＭＢＤツール２１２とを連携ブロック２１３で連携させることにより、シミュレーションを実行する。このシミュレーションにより、演算部１１は動作検証（機能検証）と性能検証を行う（ステップＳ２７）。
【０１２６】

ステップＳ２７において、モデルの実行における動作又は性能の少なくともいずれかに問題がある場合には（ステップＳ２７のＮｏ）、演算部１１はユーザの操作に応じて、ステップＳ２１又はステップＳ２２からの処理をやり直す。この詳細についてはステップＳ２５にて上述した通りである。

【０１２７】

ステップＳ２７において、モデルの実行における動作及び性能のいずれにも問題がない場合には（ステップＳ２８のＹｅｓ）、ユーザはモデルにおける適切なオブジェクトコードが完成したことになる。従って演算部１１の処理は終了する。

【０１２８】

図１０は、ステップＳ２４におけるＭＢＤツール２００及び性能検証部１７の詳細な処理の一例について示したフローチャートである。以下、ステップＳ２４における性能検証フローの詳細について説明する。

【０１２９】

ステップＳ２３の後、ＭＢＤツール２００は、まずＭＢＤシミュレーションを行う。このとき、ＭＢＤツール２００は、タスクの実行順を示した機能シミュレーションログ２０１を出力する（ステップＳ２４１）。

【０１３０】

次に性能検証部１７は、機能シミュレーションログ２０１、ＰＥ情報２０２、時間情報２０３及びリソース情報２０４を元に、タスク実行時間を計算する。計算結果は、性能シミュレーションログ１７３として出力される（ステップＳ２４２）。

【０１３１】

出力された性能シミュレーションログをユーザがそのまま見るだけでは、要求を満たす性能が出ているか否か（例えばデッドラインを満たすか否か）が一目では分かりにくい。そのため、タスク遷移表示部１７２が性能シミュレーションログ１７３を元にしてモデルのタスク遷移表示をする（ステップＳ２４３）。このタスク遷移表示を見ることにより、ユーザは、ステップＳ２５において性能に問題がないか否かを検証することが容易にできる。この検証により、ユーザはステップＳ２２の処理をやり直すか（例えばＰＥの割り当てをもう一度検討しなおすか）、それともステップＳ２５以降の先のフローに進むか否かを決定する。

【０１３２】

図１１は、以上の処理における機能シミュレーションログ２０１の一例を示した図である。図１１における「ＯＳＥＫ（Offene Systeme und deren Schnittstellen fur die Elektronik im Kraftfahrzeug）」はリアルタイムＯＳであり、マルチコアプロセッサで動作できるＯＳの具体例である。

【０１３３】

図１１におけるａはタスクブロックに関する機能シミュレーションログであり、タスク名、実行時刻、タスクブロックに割り当てられたＰＥ（割り当てＰＥ）及び呼び出し先のブロックが記載されている。呼び出し先のブロックとは、タスクブロック自身が呼び出す相手先のブロックのことをいい、無い場合にはＮｕｌｌが記載される。呼び出し先が関数であり、関数を辿った先にタスクブロック、アラームブロック又はＩＳＲブロックのいずれかがあれば、呼び出し先としてそのブロックを記録する。

【０１３４】

図１１におけるｂはアラームブロックに関する機能シミュレーションログであり、タスク名、実行時刻、アラームブロックに割り当てられたＰＥ（割り当てＰＥ）及び呼び出し先のブロックが記載されている。図１１におけるｃはＩＳＲブロックに関する機能シミュレーションログであり、タスク名、実行時刻、ＩＳＲブロックに割り当てられたＰＥ（割り当てＰＥ）及び呼び出し先のブロックが記載されている。

【０１３５】

図１２は、以上の処理における性能シミュレーションログ１７３の一例を示した図である。

【０１３６】
図１２におけるａはタスク初期起動を示すシミュレーションログであり、時刻１９４７６６１において、タスク３がＩＤ１のプロセッサにおいてＲｅａｄｙ状態になったことを示している。図１２におけるｂはタスクが実行されることを示すシミュレーションログであり、時刻１９４９１８７において、タスク２がＩＤ１のプロセッサにおいてＲｕｎ状態になったことを示している。図１２におけるｃはタスク終了を示すシミュレーションログであり、時刻７１３１４０１２において、タスク２がＩＤ１のプロセッサにおいてＤｏｒｍａｎｔ状態になったことを示している。
【０１３７】
図１２におけるｄはタスク起動を示すシミュレーションログであり、時刻１９４６９２６において、タスク２がＩＤ１のプロセッサにおいてａｃｔｉｖｅ　ｔａｓｋ状態になったことを示している。図１２におけるｅは割り込みが発生したことを示すシミュレーションログであり、時刻２１３６６３７においてＩＤ１のプロセッサが割り込みハンドラ５に入ったことを示すとともに、時刻２１３６８６１においてＩＤ１のプロセッサが割り込みハンドラ５を抜けたことを示している。図１２におけるｆはアラームの設定を示すシミュレーションログであり、時刻９８９３３７において、ＩＤ１のプロセッサにおいてＩＤ１のアラームが設定されたことを示している。
【０１３８】
図１３は、以上の処理においてタスク実行時間計算部１７１が計算した計算結果の一例を示したイメージ図である。換言すれば、図１２に示したような性能シミュレーションログを分かりやすく図示したのが図１３となる。以下、これについて詳細に説明する。
【０１３９】
図１３の（ａ）には、タスクブロックが起動する起動時刻、各タスクブロックに割り当てられたプロセッサＩＤ、タスクブロック名及び実行時間のデータが記録されている。
【０１４０】
各タスクブロックの起動時間は、ＭＢＤツール２００が実行する機能シミュレーションによって判明する。従って、ＭＢＤツール２００が出力する機能シミュレーションログ２０１においては、図１３の（ａ）の「ｔｉｍｅ」に記載された各タスクブロックの起動時刻が記載されている。
【０１４１】
各タスクブロックに割り当てられたプロセッサＩＤは、ＰＥ情報２０２に記録されており、実行時間のデータは時間情報２０３に記録されている。
【０１４２】
なお、図１３の（ａ）において、タスクブロックはｔａｓｋ１＿１、ｔａｓｋ１＿２、ｔａｓｋ２＿１、ｔａｓｋ２＿２の４つが定義されている。そして、ｔａｓｋ１＿１はＰＥＩＤ＝１において、起動時刻が０において実行時間０．１で実行されることが示されている。同様に、ｔａｓｋ１＿２はＰＥＩＤ＝１において、起動時刻が０において実行時間０．２で実行されることが示されている。
【０１４３】
ｔａｓｋ２＿１はＰＥＩＤ＝２において、起動時刻が０において実行時間０．１で実行されるとともに、起動時刻が１において実行時間０．１で実行されることが示されている。ｔａｓｋ２＿２はＰＥＩＤ＝２において、起動時刻が０において実行時間０．３で実行されることが示されている。
【０１４４】
タスク実行時間計算部１７１は以上のデータに基づいて、性能シミュレーションを実行する。タスク実行時間計算部１７１は、ＰＥＩＤ毎に遅延時間を計算する。
【０１４５】
図１３の（ｂ）は、（ａ）のデータに基づいて、タスク実行時間計算部１７１が実行した性能シミュレーションの結果（性能シミュレーションログ１７３）を示した図である。（ｂ）からは、ＰＥＩＤ＝１の場合（ＩＤ１のプロセッサにおいて）、ｔａｓｋ１＿１が

０～０．１、ｔａｓｋ１＿２が０＋０．１（＝０．１）～０＋０．１＋０．２（＝０．３）の時刻に実行される（Ｒｕｎ状態になる）ことが分かる。ＰＥＩＤ＝２の場合（ＩＤ２のプロセッサにおいて）、ｔａｓｋ２＿１が０～０．１、ｔａｓｋ２＿２が０＋０．１（＝０．１）～０＋０．１＋０．３（＝０．４）、ｔａｓｋ２＿１が１．０～１．０＋０．１（＝１．１）の時刻に実行される（Ｒｕｎ状態になる）ことが分かる。

【０１４６】

なお、ｔａｓｋ１＿１は時刻０．１以降でＤｏｒｍａｎｔ状態になり、ｔａｓｋ１＿２は時刻０～０．１でＲｅａｄｙ状態、時刻０．３以降でＤｏｒｍａｎｔ状態になる。また、ｔａｓｋ２＿１は時刻０．１以降でＤｏｒｍａｎｔ状態になり、ｔａｓｋ２＿２は時刻０～０．１以降でＲｅａｄｙ状態、時刻０．４以降でＤｏｒｍａｎｔ状態になる。

【０１４７】

図１４は、図１３の（ｂ）に記載された性能シミュレーションログ１７３をタスク遷移表示部１７２において表示したときの表示画面の一例である。図１４の表示画面は、上述のタスク遷移表示遷移ツールによって表示されるタスク遷移表示を単純化して示している。なお、図１４において横軸は時間経過を示している。

【０１４８】

図１４を見ることにより、ｔａｓｋ１＿１～ｔａｓｋ２＿２がいずれのプロセッサでいつ実行されるかを、ユーザは簡単に確認することができる。図１４により、ｔａｓｋ１＿１は０～０．１の時刻に実行され、ｔａｓｋ１＿２は０．１～０．３の時刻に実行され、ｔａｓｋ２＿１が０～０．１と１．０～１．１、ｔａｓｋ２＿２が０．１～０．４の時刻に実行されることが分かる。

【０１４９】

仮にｔａｓｋ２＿２の実行時間が１．１の場合には、ｔａｓｋ２＿２は０＋０．１（＝０．１）～０＋０．１＋１．１（＝１．２）の時間に実行される。ｔａｓｋ２＿１は時刻１．０周期で起動する前提の場合、次に時刻１．０から始まるはずのｔａｓｋ２＿１の実行と重なってしまう。つまり、この場合には性能を満たした動きができないことが、タスク遷移表示部１７２への表示によりユーザが判定することができる。

【０１５０】

このようにして、性能検証装置２７は、性能検証処理において、マルチコアプロセッサの各プロセッサコアにおけるタスクブロックの実行開始時間及び実行終了時間を実行過程として検証している。

【０１５１】

従来、機能シミュレーションの段階（ＭＢＤのみのシミュレーション）では、モデルの性能検証まではできなかった。そのため、マルチコアＳＷのような、どのＰＥでタスクを動かすべきかを決定する処理について、早期段階で実現ができなかった。

【０１５２】

しかし、実施の形態３においては、機能シミュレーションの結果（ログ）とタスクのＰＥへの割り当て情報、タスクの実行時間を考慮して、コード生成前に実行時間を計算（予測）することができる。このため、早期にマルチコアＳＷのタスクのＰＥ割り当てを検討することができなかった問題を解決することができた。

【０１５３】

そして、性能検証装置２７は、性能検証処理において、タスクブロック毎に算出された実測値及びタスクブロック毎のプロセッサコアの割り当て情報を用いて、制御プログラムの実行過程を検証している。そのため、性能検証装置２７は、実際の制御プログラムの性能検証をより精度よく実現することができる。特に、性能検証装置２７は、プロセッサコア毎の実行性能を検証することができる。

【０１５４】

性能検証装置２７は、性能検証処理において、マルチコアプロセッサの各プロセッサコアにおけるタスクブロックの実行開始時間及び実行終了時間を実行過程として検証している。そのため、性能検証装置２７は、各プロセッサコアにおけるタスクブロックの実行状

況を具体的に検証することができるため、制御プログラムについてより具体的な検証ができる。

【０１５５】

なお実施の形態３にかかる性能検証方法は、実施の形態２と同様に、ＰＩＬＳ環境、特にＭＢＤツールを使用したＰＩＬＳ環境に適用することができる。

【０１５６】

実施の形態４

次に実施の形態４では、マルチコアプロセッサ用のプログラムにおいて、タスクブロックにおいて所定の性能を満たすようにＰＥをコード生成前に割り当てる方法を説明する。

【０１５７】

実施の形態４では、実施の形態３において記載した「１．性能検証面」、「２．コード生成面」、「３．ＰＩＬＳシミュレーション面」のうち、「２．コード生成面」における課題を解決する。

【０１５８】

実施の形態４では、具体的にはＯＳモデルにおける以下の４ブロックを組み合わせることにより、プログラムのアルゴリズムを表現する。このモデルではマルチコアＯＳ、特にリアルタイムＯＳであるＯＳＥＫをベースにしたコード生成がなされる。

【０１５９】

１．タスクブロック

タスクブロックは、タスクの処理を行うブロックである。タスクブロックには入力端子と出力端子が備えられており、入力端子よりコールされたタイミングで、パラメータ・ダイアログで指定された名前のタスクをＡｃｔｉｖａｔｅ　Ｔａｓｋシステムコールで呼び出すブロックである。ここで、タスクブロックに接続されたＦｕｎｃｔｉｏｎ－Ｃａｌｌ　Ｓｕｂｓｙｓｔｅｍブロックが、タスクの処理関数とみなされる。なおＦｕｎｃｔｉｏｎ－Ｃａｌｌ　Ｓｕｂｓｙｓｔｅｍブロックはタスク処理関数ブロックであり、タスクブロックの出力端子とトリガ端子とを接続することによって、タスク処理関数とみなされるブロックである。

【０１６０】

タスクブロックには、割り当てるＰＥの情報が設定される。タスクブロックにより、コード生成のときにＰＥ毎に使用するＴａｓｋコール部（コード）が生成され、ＯＩＬ（OSEK Implementation Language）上にＰＥ毎に使用するＴａｓｋ定義が生成される。また、タスクブロックとつながる関数ブロックと接続するリソースブロックによってリソース取得・解放のコードが生成される。

【０１６１】

２．Ａｌａｒｍブロック

Ａｌａｒｍブロックは、アラームの処理を行うブロックである。Ａｌａｒｍブロックには入力端子と出力端子が備えられており、入力端子よりコールされる度に、Ａｌａｒｍブロックは指定した値をカウントし、カウンタが満了する値に達したタイミングで、出力端子に接続したブロックにアラームの満了を通知する。換言すれば、Ａｌａｒｍブロックは、タスクのタイマー機能の役割を有する。なお、出力端子への通知の頻度は、ワンショット・アラームか周期アラームかによって異なる。

【０１６２】

Ａｌａｒｍブロックには、割り当てるＰＥの情報が設定される。Ａｌａｒｍブロックにより、ＯＩＬ上にＰＥ毎に使用するＡｌａｒｍが生成される。

【０１６３】

３．ＩＳＲブロック

ＩＳＲブロックは、マルチコアＳＷ内のブロックに対して、ＩＳＲ（割り込み）の通知を行うブロックである。ＩＳＲブロックには入力端子と出力端子が備えられており、入力信号から信号が入力される度に、出力端子に接続されたブロックに対し、指定された割り込み要因コードをパラメータとして関数コールを行うことによって割り込みの発生を通知

する。通知を受けたブロックでは、その割り込みに対する処理を行う。なお出力端子に接続できるのは、タスクブロックか、Ｆｕｎｃｔｉｏｎ－Ｃａｌｌ　Ｓｕｂｓｙｓｔｅｍブロックのいずれかである。

【０１６４】

ＩＳＲブロックには、割り当てるＰＥの情報が設定される。ＩＳＲブロックにより、コード生成のときにＰＥ毎に使用するＩＳＲコール部（コード）が生成され、ＯＩＬ上にＰＥ毎に使用するＩＳＲが生成される。ＩＳＲブロックは、実施の形態４にかかるマルチコアプロセッサ用のモデルにおける特徴の１つであり、排他制御の処理において必要なブロックである。

【０１６５】

４．リソースブロック（Ｐｒｏｔｅｃｔｅｄ　ＲＴブロック）

実施の形態４では、Ｐｒｏｔｅｃｔｅｄ　ＲＴブロックを、リソースの制御をするリソースブロックとして使用している。Ｐｒｏｔｅｃｔｅｄ　ＲＴブロックは、ある速度で動作しているブロックの出力データを、異なる速度で動作しているブロックの入力に伝達するブロックである。Ｐｒｏｔｅｃｔｅｄ　ＲＴブロックのパラメータにより、データの整合性や確定性を保証する転送と、より早い応答やより少ないメモリ要求とを変更するオプション指定ができる。タスク間やＰＥ間で、共通のリソースを扱う場合に、リソースブロックを関係するブロック間にＰｒｏｔｅｃｔｅｄ　ＲＴブロックを配置することによって、リソースの競合を防ぐことに応用する。このようにして、Ｐｒｏｔｅｃｔｅｄ　ＲＴブロックはリソースに類似した機能をする（リソースブロック）。

【０１６６】

リソースブロックには、割り当てるＰＥの情報が設定される。リソースブロックにより、ＯＩＬ上にＰＥ毎に使用するリソース情報が生成される。

【０１６７】

実施の形態４にかかる性能検証装置の例は、実施の形態３における図６にて説明した通りであるため、説明を省略する。また、実施の形態４におけるマルチコアコード生成の例については、実施の形態３における図８にて説明した通りであるため、説明を省略する。

【０１６８】

図１５は、実施の形態４にかかるマルチコアコード生成部２０の例を示した構成図である。以下、図１５を用いて、マルチコアコード生成部２０の詳細について説明する。

【０１６９】

マルチコアコード生成部２０はマルチコア用ブロックコード生成部２８及びマルチコアＯＳ用コード生成部２９を備える。マルチコア用ブロックコード生成部２８は、リソース考慮タスクブロックコード生成部３０、Ａｌａｒｍブロックコード生成部３１及びＩＳＲブロックコード生成部３２を有する。

【０１７０】

マルチコアＯＳモデル２０６内にあるＴａｓｋ、Ａｌａｒｍ、ＩＳＲ、リソースといったブロックには、割り当てるＰＥ情報が付属されている。マルチコアコード生成部２０は、このようなマルチコアＯＳモデル２０６を、マルチコア用ブロックコード生成部２８に入力する。

【０１７１】

リソース考慮タスクブロックコード生成部３０は、マルチコアＯＳモデル２０６内にあるタスクブロックとリソースブロック（Ｒａｔｅ　ｔｒａｎｓｉｔｉｏｎブロック）とを処理する。タスクブロックはリソース考慮タスクブロックコード生成部３０によりＴａｓｋのターゲットコード３４に変換され、リソースブロックはリソースのＯＩＬ用情報３５に変換される。なおターゲットコード３４はＣ又はＣ＋＋で記載されたコードであり、ＯＩＬ用情報３５は、設定情報（例えばリソース等の設定情報）が記載される。

【０１７２】

Ａｌａｒｍブロックコード生成部３１は、マルチコアＯＳモデル２０６内にあるＡｌａｒｍブロックを処理する。Ａｌａｒｍブロックは、Ａｌａｒｍのターゲットコード３４と

ＡｌａｒｍのＯＩＬ用情報３５とに変換される。
【０１７３】
ＩＳＲブロックコード生成部３２は、マルチコアＯＳモデル２０６内にあるＩＳＲブロックを処理する。ＩＳＲブロックは、ＩＳＲのターゲットコード３４とＩＳＲのＯＩＬ用情報３５とに変換される。
【０１７４】
次に、マルチコアコード生成部２０は、マルチコア用ブロックコード生成部２８が生成したターゲットコード（Ｔａｓｋ、Ａｌａｒｍ、ＩＳＲ）３４とＯＩＬ用情報（Ａｌａｒｍ、ＩＳＲ、リソース）３５とを、マルチコアＯＳ用コード生成部２９に入力する。マルチコアＯＳ用コード生成部２９は、ターゲットコード３４及びＯＩＬ用情報３５に基づいて、マルチコアＳＷ２０７を生成する。マルチコアＳＷ２０７には、デュアルコアプロセッサの場合は、ＰＥ１ターゲットコード２１４（ＰＥ１帰属コード）、ＰＥ２ターゲットコード２１５（ＰＥ２帰属コード）、マルチコアＯＩＬ２１６、共通コード／プロジェクト２１７が含まれる。なお、ここでは、マルチコアコード生成部２０は、２つのプロセッサコアがあるマルチプロセッサコア上で動作する制御プログラムのコードを生成する。
【０１７５】
ＰＥ１ターゲットコード２１４はＰＥ１におけるＣ又はＣ＋＋のコード、ＰＥ２ターゲットコード２１５はＰＥ２におけるＣ又はＣ＋＋のコードである。マルチコアＯＩＬ２１６は、ＰＥ１ターゲットコード２１４及びＰＥ２ターゲットコード２１５に記載されたＰＥ１及びＰＥ２の処理とＯＳをつなげるための設定ファイルである。
【０１７６】
マルチコアコード生成部２０は、マルチコアＳＷ２０７を生成するタイミングにおいて、連携ブロック２１３（シミュレータとの連携用ブロック）を生成する。連携ブロック２１３は、シミュレータ２１１と制御対象モデルを載せたＭＢＤツール２１２間で連携シミュレーション（ＰＩＬＳ）を実行するブロックである。
【０１７７】
実施の形態４におけるマルチコアＯＳ作成フローは実施の形態３における図９の通りであり、説明を省略する。
【０１７８】
図１６は、図９のステップＳ２６におけるマルチコアコード生成部２０の詳細な処理の一例について示したフローチャートである。以下、ステップＳ２４におけるマルチコアコード生成フローについて詳述する。
【０１７９】
ステップＳ２６においては、まず、リソース考慮タスクブロックコード生成部３０は、リソース考慮Ｔａｓｋコード（ターゲットコード３４）を生成する（ステップＳ２６１）。ここでリソース考慮タスクブロックコード生成部３０は、生成したリソース考慮Ｔａｓｋコードに対応するＯＩＬ用情報３５を生成する。
【０１８０】
次に、Ａｌａｒｍブロックコード生成部３１は、Ａｌａｒｍコード（ターゲットコード３４）を生成する（ステップＳ２６２）。ここでＡｌａｒｍブロックコード生成部３１は、生成したＡｌａｒｍコードに対応するＯＩＬ用情報３５を生成する。
【０１８１】
次に、ＩＳＲブロックコード生成部３２は、ＩＳＲコード（ターゲットコード３４）を生成する（ステップＳ２６３）。ここでＩＳＲブロックコード生成部３２は、生成したＩＳＲコードに対応するＯＩＬ用情報３５を生成する。
【０１８２】
マルチコアＯＳ用コード生成部２９は、以上の処理によって生成されたターゲットコード（Ｔａｓｋ、Ａｌａｒｍ、ＩＳＲ）３４とＯＩＬ用情報（Ａｌａｒｍ、ＩＳＲ、リソース）３５を使って、ＰＥターゲットコードを生成する（ステップＳ２６４）。この処理はＰＥ毎になされる。換言すれば、ステップＳ２６４の処理はＰＥ数分だけループする。

【０１８３】
次に、マルチコアＯＳ用コード生成部２９は、全てのＯＩＬ用情報３５に基づいて、図１５のマルチコアＯＩＬ２１６を生成する（ステップＳ２６５）。
【０１８４】
マルチコアＯＳ用コード生成部２９は、共通コード／プロジェクト２１７を生成する（ステップＳ２６６）。例えば共通コードは、スタートアップファイルであり、プロジェクトファイルは、出力したファイルを元にしたコンパイルを行うファイルである。
【０１８５】
マルチコアＯＳ用コード生成部２９内のシミュレータ連携ブロック生成部３３は、連携ブロック２１３を生成する（ステップＳ２６７）。以上のようにして、マルチコアコード生成部２０はステップＳ２６の処理を実行する。
【０１８６】
以下、タスクブロック、リソースブロック、Ａｌａｒｍブロック、ＩＳＲブロックのそれぞれがモデル中にある場合に生成されるターゲットコード３４とＯＩＬ用情報３５との例について説明する。
【０１８７】
図１７は、タスクブロックが含まれるモデルの一例を示したイメージ図である。図１７におけるタスクブロックは「Ｔａｓｋ１」の名前が付けられており、ＩＤ１のＰＥが割り当てられている。この割り当て情報は、マルチコアＯＳモデル２０６のブロックごとに設定されるＰＥ情報に対応する。
【０１８８】
図１７のモデルにおいて、「Ｔａｓｋ１」のタスクブロックはＦｕｎｃｔｉｏｎ－Ｃａｌｌ　Ｇｅｎｅｒａｔｏｒからの入力端子へのコールの入力により、関数ブロック（Ｆｕｎｃｔｉｏｎ　Ｃａｌｌ　Ｓｕｂｓｙｓｔｅｍ）を呼び出す。ここで関数ブロックのトリガ端子にはタスクブロックの出力端子が接続されている。関数ブロックは、関数の計算結果「１」を出力端子Ｏｕｔ１から出力する。
【０１８９】
図１８は、図１７のモデルにおけるタスクブロックに関するターゲットコード及びＯＩＬ用情報の一例を示す図である。図１８（ａ）は、図１７のモデルに関するタスクブロックのターゲットコードを示す。図１７においてタスクブロック自体は関数ブロックと接続されているため、タスクブロックに接続された関数が図１８（ａ）においてＴａｓｋの付属コードとして生成される。
【０１９０】
図１８（ｂ）は、図１７のモデルに関するタスクブロックのＯＩＬ用情報を示す。Ｔａｓｋ１がどのＰＥに割り当てられるかは、ｃｐｕ＿ＰＥ１の場所に設定される。図１８（ａ）の＜１＞に記載された通り、Ｔａｓｋ１はＰＥ１に割り当てられる。
【０１９１】
また、図１８（ｂ）の＜２＞に記載された通り、Ｔａｓｋ１はリソース１（ＲＳ１）が割り当てられている。リソース１は、例えば共有メモリ等のハードウェア資源である。
【０１９２】
図１９は、リソースブロックが含まれるモデルの一例を示したイメージ図である。図１９におけるリソースブロック（Ｐｒｏｔｅｃｔｅｄ　ＲＴ１）には、ＩＤ１のＰＥが割り当てられている。図１９における「Ｔａｓｋ１２」のタスクブロックにはＩＤ１のＰＥが割り当てられ、「Ｔａｓｋ２１」のタスクブロックにはＩＤ２のＰＥが割り当てられる。この割り当て情報は、マルチコアＯＳモデル２０６のブロックごとに設定されるＰＥ情報に対応する。
【０１９３】
図１９のモデルにおいて、「Ｔａｓｋ１２」のタスクブロックは入力端子へのコールの入力により、「Ｔａｓｋ１２」の関数ブロックを呼び出す。ここで「Ｔａｓｋ１２」の関数ブロックのトリガ端子には、「Ｔａｓｋ１２」のタスクブロックの出力端子が接続され

ている。「Ｔａｓｋ１２」の関数ブロックの入力端子Ｉｎ１には、リソースブロックの出力端子が接続されており、「Ｔａｓｋ１２」の関数ブロックはリソースブロックからの出力及び「Ｔａｓｋ１２」のタスクブロックの呼び出しに基づいて、関数の計算結果を出力端子Ｏｕｔ１から出力する。

【０１９４】

「Ｔａｓｋ２１」のタスクブロックは入力端子へのコールの入力により、「Ｔａｓｋ２１」の関数ブロックを呼び出す。ここで「Ｔａｓｋ２１」の関数ブロックのトリガ端子には、「Ｔａｓｋ２１」のタスクブロックの出力端子が接続されている。「Ｔａｓｋ２１」の関数ブロックの入力端子Ｉｎ２には、Ｆｒｏｍ［Ａ］からの入力値が入力され、「Ｔａｓｋ２１」の関数ブロックはＦｒｏｍ［Ａ］からの入力値及び「Ｔａｓｋ２１」のタスクブロックの呼び出しに基づいて、関数の計算結果を出力端子Ｏｕｔ２から出力する。

【０１９５】

リソースブロックは、「Ｔａｓｋ２１」の関数ブロックの出力を制御することにより、リソースの競合を防ぐ。「Ｔａｓｋ１２」と「Ｔａｓｋ２１」とが同じリソースＲＳ１（例えば共有メモリ）を用いてタスクを実行する場合に、「Ｔａｓｋ１２」と「Ｔａｓｋ２１」とは異なるＰＥによって実行される。そのため、「Ｔａｓｋ２１」による出力中に、「Ｔａｓｋ１２」が実行されてしまい、「Ｔａｓｋ１２」の所望の結果を取得することができなくなってしまうことが考えられる。シングルコアプロセッサで図１９に記載のモデルを実行する場合には、一つのプロセッサコア上で読み出しと書き込みを同時に行えないため、このような事態は発生しない。しかし、マルチコアプロセッサでモデルを実行する場合には、複数のプロセッサコアが独立に同じタイミングで処理を実行することが可能であるため、このような事態が発生しうる。

【０１９６】

図２０は、図１９のモデルにおけるリソースブロックに関するターゲットコード及びＯＩＬ用情報の一例を示す図である。図２０（ａ）は、図１９のモデルにおけるリソースブロックに関するターゲットコードを示す。図２０（ａ）では、Ｔａｓｋ１２及びＴａｓｋ２１のタスクブロックのコードは図８と同様に生成される。

【０１９７】

図１９に示したモデルでは、２つのＴａｓｋ間でリソースブロックを使うことにより、データ共有を行っている。このような場合、モデルにかかるターゲットコードにおいては、Ｔａｓｋから呼び出される関数内全体を囲むようにリソースの取得及び解放を行うコードを挿入する。図２０（ａ）の＜１＞の通り、ＧｅｔＲｅｓｏｕｒｃｅ（ＲＳ１）、ＲｅｌｅａｓｅＲｅｓｏｕｒｃｅ（ＲＳ１）がＲＳ１の取得及び解放を行うコードとして記載されている。なお、図２０（ａ）におけるｆｎｃ２１（）、ｆｎｃ１２（）の関数内では、リソースブロックからのデータのｒｅａｄ／ｗｒｉｔｅを行い、処理をするコードが生成される。

【０１９８】

図２０（ｂ）は、図１９のモデルに関するタスクブロックのＯＩＬ用情報を示す。Ｔａｓｋ１、Ｔａｓｋ２がどのＰＥに割り当てられるかは、それぞれｃｐｕ＿ＰＥ１、ｃｐｕ＿ＰＥ２の場所に設定される。

【０１９９】

リソースブロックに関する情報自体は、ＯＩＬ用情報にも出力される。図１９のモデルの場合リソースブロックにＰＥＩＤ＝１を設定したため、ｃｐｕ＿ＰＥ１内にリソースがＲＳ１である情報が記載される。つまり、ＲＳ１の取得及び解放を行う情報がＯＩＬ用情報にも記載される。これは図２０（ｂ）の＜２＞の通りである。

【０２００】

以上のように、リソースの取得及び解放を行う情報をターゲットコード及びＯＩＬ用情報に記載することにより、モデルの実行に際してのリソースの制御を反映させることができる。

【０２０１】

本例では示していないが、複数のリソースを共有する場合には、リソース考慮タスクブロックコード生成部３０はターゲットコードにおいて、「ＧｅｔＲｅｓｏｕｒｃｅ（ＲＳ１）；ＧｅｔＲｅｓｏｕｒｃｅ（ＲＳ２）；・・・ＲｅｌｅａｓｅＲｅｓｏｕｒｃｅ（ＲＳ２）；ＲｅｌｅａｓｅＲｅｓｏｕｒｃｅ（ＲＳ２）；」のように入れ子となるコードを生成する。

【０２０２】

なお、ＯＩＬ用情報は図１８、図２０ではＯＩＬファイルそのままを図示しているが、ステップＳ２６１の段階ではあくまで内部データ構造として保持している。ＯＩＬファイル生成（ステップＳ２６５)の段階において、マルチコアＯＳ用コード生成部２９はＯＩＬ情報に基づいてＯＩＬファイルを生成する。これは図２２、図２４にかかるＯＩＬ用情報についても同様である。

【０２０３】

図２１は、Ａｌａｒｍブロックが含まれるモデルの一例を示したイメージ図である。図２１におけるＡｌａｒｍブロックは「Ａｌａｒｍ＿ｒｅｐｅａｔ」の名前が付けられており、ＩＤ１のＰＥが割り当てられている。またタスクブロックは「Ｔａｓｋ＿ｒｅｐｅａｔ」の名前が付けられており、ＩＤ１のＰＥが割り当てられている。この割り当て情報は、マルチコアＯＳモデル２０６のブロックごとに設定されるＰＥ情報に対応する。

【０２０４】

図２１のモデルにおいて、ＡｌａｒｍブロックはＦｕｎｃｔｉｏｎ－Ｃａｌｌ Ｇｅｎｅｒａｔｏｒからの入力端子へのコールの度に指定した値をカウントし、カウンタが満了する値に達したタイミングで、出力端子に接続したタスクブロックに通知する。タスクブロックは入力端子から入力されたその通知により、関数ブロックを呼び出す。ここで関数ブロックのトリガ端子にはタスクブロックの出力端子が接続されている。関数ブロックは、関数の計算結果を出力端子Ｏｕｔ１から出力する。

【０２０５】

図２２は、図２１のモデルにおけるＡｌａｒｍブロックに関するターゲットコード及びＯＩＬ用情報の一例を示す図である。このＡｌａｒｍブロックに関するターゲットコード及びＯＩＬ用情報は、Ａｌａｒｍブロックコード生成部１９４により図１６におけるステップＳ２６２において生成される。図２２（ａ）は、図２１のモデルにおけるＡｌａｒｍブロックに関するターゲットコードを示す。

【０２０６】

図２１においてＡｌａｒｍブロックにはＰＥＩＤ＝１、タスクブロックにはＰＥＩＤ＝１が割り当てられている。Ａｌａｒｍブロックコード生成部３１はＡｌａｒｍブロックのコードを生成すると、コードにおけるＴａｓｋ（ｉｎｉｔ）という初期化Ｔａｓｋ内に「ＳｅｔＲｅｌＡｌａｒｍ（Ａｌａｒｍ＿ｒｅｐｅａｔ・・・）；」というコードを出力する。これは図２２（ａ）の＜１＞の通りである。

【０２０７】

図２２（ｂ）は、図２１のモデルに関するＡｌａｒｍブロックのＯＩＬ用情報を示す。Ａｌａｒｍブロックコード生成部３１は、Ａｌａｒｍブロックのコードを生成するのと同じタイミングで、ＯＩＬ用情報におけるｃｐｕ＿ＰＥ１の箇所に「Ａｌａｒｍ＿ｒｅｐｅａｔ」を登録する。ＯＩＬ用情報において、この「Ａｌａｒｍ＿ｒｅｐｅａｔ」の後に「Ｔａｓｋ＿ｒｅｐｅａｔ」が記載されるため、「Ａｌａｒｍ＿ｒｅｐｅａｔ」から「Ｔａｓｋ＿ｒｅｐｅａｔ」が起動することがＯＩＬ用情報において示される。これは図２２（ｂ）の＜２＞、＜３＞の通りである。

【０２０８】

図２３は、ＩＳＲブロックが含まれるモデルの一例を示したイメージ図である。図２３におけるＩＳＲブロックは「ＩＮＴＰ０」の名前が付けられており、ＩＤ１のＰＥが割り当てられている。またタスクブロックは「Ｔａｓｋ」の名前が付けられており、ＩＤ１のＰＥが割り当てられている。この割り当て情報は、マルチコアＯＳモデル２０６のブロックごとに設定されるＰＥ情報に対応する。

【０２０９】
なおＩＳＲブロックにおいては、ＩＮＴＰ０の例外コードとして「０ｘ８０」が設定されている。これは、割り込みが発生した場合に、例外処理として「０ｘ８０」に飛んで処理を実行するように制御することを意味する。
【０２１０】
図２３のモデルにおいて、ＩＳＲブロックはＦｕｎｃｔｉｏｎ－Ｃａｌｌ Ｇｅｎｅｒａｔｏｒからの入力端子へのコールの度に、出力端子に接続したタスクブロックに対し、指定された割り込み要因コードをパラメータとして関数コールを行い、割り込みの発生を通知する。タスクブロックは入力端子から入力されたその通知により、関数ブロックを呼び出す。ここで関数ブロックのトリガ端子にはタスクブロックの出力端子が接続されている。関数ブロックは、関数の計算結果を出力端子Ｏｕｔ１から出力する。
【０２１１】
図２４は、図２３のモデルにおけるＩＳＲブロックに関するターゲットコード及びＯＩＬ用情報の一例を示す図である。このＩＳＲブロックに関するターゲットコード及びＯＩＬ用情報は、ＩＳＲブロックコード生成部３２により図１７におけるステップＳ２６３において生成される。図２４（ａ）は、図２２のモデルにおけるＩＳＲブロックに関するターゲットコードを示す。
【０２１２】
図２３においてＩＳＲブロックにはＰＥＩＤ＝１、タスクブロックにはＰＥＩＤ＝１が割り当てられている。図２２のモデルではＩＳＲブロックとタスクブロックがつながっているため、ＩＳＲブロックコード生成部３２はＩＳＲブロックのコードを生成すると、ＩＳＲ（ＩＮＴＰ０）からＴａｓｋ１をＡｃｔｉｖａｔｅＴａｓｋ（起動）するコードを出力する。これは図２４（ａ）の＜１＞の通りである。
【０２１３】
図２４（ｂ）は、図２３のモデルに関するＩＳＲブロックのＯＩＬ用情報を示す。ＩＳＲブロックコード生成部３２は、ＩＳＲブロックのコードを生成するのと同じタイミングで、ＯＩＬ用情報におけるｃｐｕ＿ＰＥ１の箇所に「ＩＮＴＰ０」のＩＳＲを登録する。そして、ＩＳＲブロックにおいてＩＮＴＰ０の例外コードとして「０ｘ８０」が設定されているため、ＯＩＬ用情報には「０ｘ８０」が出力される。これは図２４（ｂ）の＜２＞の通りである。
【０２１４】
以上のようにして生成されたターゲットコード３４とＯＩＬ用情報３５に基づいて、図１６のステップＳ２６４においてマルチコアＯＳ用コード生成部２９はＰＥ帰属コードを生成する。この処理はＰＥ毎に実行される。例えば、ＰＥ１に対しては、マルチコアＯＳ用コード生成部２９は以下の処理を実行する。
・ＰＥ１用のＣ／Ｃ＋＋におけるｍａｉｎ関数の出力
・ＰＥ１に属するｔａｓｋターゲットコードを抽出して、ＰＥ１用ｔａｓｋターゲットコードとして出力
・ＩＳＲのＯＩＬ用情報３５からＰＥ１に属するＩＳＲコードを抽出して、ＰＥ１用ＩＳＲターゲットコードとして出力
・出力したＰＥ１用ｔａｓｋターゲットコードにフック関数を追記
・ＰＥ１用のメモリ初期化コードを出力
以上によって出力されたコード全てを総称して、図１５におけるＰＥ１ターゲットコード２１４としている。
【０２１５】
マルチコアコードの生成を実現するためには、ＰＥ数拡張に適合しやすくするため、ＯＳモデルをコード生成のベースにするのが望ましい。そこで、実施の形態４では、ＯＳモデルでマルチコアモデルを実現した。実施の形態４では、具体的には以下の処理を実施している。
１．シングルコアプロセッサで使用していたタスクブロック、Ａｌａｒｍブロックのほか

にＩＳＲブロックをモデル内のブロックとして用意した。
２．Ｒａｔｅ　ｔｒａｎｓｉｔｉｏｎブロックはシングルコアプロセッサのときにもモデル内に存在していたが、実施の形態４においてはリソースを扱うブロックという意味で用いられている。
３．Ｔａｓｋからコールされる関数内からＲａｔｅ　ｔｒａｎｓｉｔｉｏｎブロックを使用する場合、「ＧｅｔＲｅｓｏｕｒｃｅ（）,ＲｅｌｅａｓｅＲｅｓｏｕｒｃｅ（）」で関数全体を囲み、その中にデータのｒｅａｄ／ｗｒｉｔｅや処理がコードに記述されるようにした。
４．タスクブロック、Ａｌａｒｍブロック、ＩＳＲブロック、Ｒａｔｅ　ｔｒａｎｓｉｔｉｏｎブロックにＰＥ割り当て情報を設定できるようにした。

【０２１６】
以上の処理により、ＰＥを意識しないアルゴリズムモデルに対して、ＰＥ情報を設定するだけで、性能検証装置２７はマルチコアＯＳを使用した複雑なコードを短時間かつ正確に作成できる。また、性能検証装置２７は、マルチコアコード生成と同時に連携シミュレーションブロックも自動生成することができる。そのため、性能検証装置２７は、マルチコアシミュレータとＭＢＤツールの連携シミュレーション環境構築を短時間かつ正確に作成できる。

【０２１７】
なお、実施の形態４においてはリアルタイムＯＳのＯＳＥＫをベースにしてマルチコアコードを生成しているが、ベースにするＯＳはＯＳＥＫに限られず、例えばＩＴＲＯＮ（Industrial TRON）へ拡張することも可能である。

【０２１８】
以上、本発明者によってなされた発明を実施の形態に基づき具体的に説明したが、本発明は既に述べた実施の形態に限定されるものではなく、その要旨を逸脱しない範囲において種々の変更が可能であることはいうまでもない。

【０２１９】
実施の形態１～４のそれぞれに示した性能検証装置が実行する処理フローは、性能検証装置において適宜組み合わせて実行することができる。例えば、実施の形態２にかかる性能検証装置は、実施の形態３に示したオブジェクトコード生成の処理を実行することができる。また、実施の形態２にかかる性能検証装置は、実施の形態４に示したマルチコアコード生成の処理を実行することができる。

【０２２０】
実施の形態１～４に示した性能検証装置が実行する処理フローは、制御方法の１つとして、コンピュータに実行させることができる。例えば、処理フローを制御プログラムとしてコンピュータに実行させてもよい。

【０２２１】
プログラムは、様々なタイプの非一時的なコンピュータ可読媒体（non-transitory computer readable medium）を用いて格納され、コンピュータに供給することができる。非一時的なコンピュータ可読媒体は、様々なタイプの実体のある記録媒体（tangible storage medium）を含む。非一時的なコンピュータ可読媒体の例は、磁気記録媒体（例えばフレキシブルディスク、磁気テープ、ハードディスクドライブ）、光磁気記録媒体（例えば光磁気ディスク）、ＣＤ－ＲＯＭ、ＣＤ－Ｒ、ＣＤ－Ｒ／Ｗ、半導体メモリ（例えば、マスクＲＯＭ、ＰＲＯＭ（Programmable ROM）、ＥＰＲＯＭ（Erasable PROM）、フラッシュＲＯＭ、ＲＡＭ（Random Access Memory））を含む。また、プログラムは、様々なタイプの一時的なコンピュータ可読媒体（transitory computer readable medium）によってコンピュータに供給されてもよい。一時的なコンピュータ可読媒体の例は、電気信号、光信号、及び電磁波を含む。一時的なコンピュータ可読媒体は、電線及び光ファイバ等の有線通信路、又は無線通信路を介して、プログラムをコンピュータに供給できる。

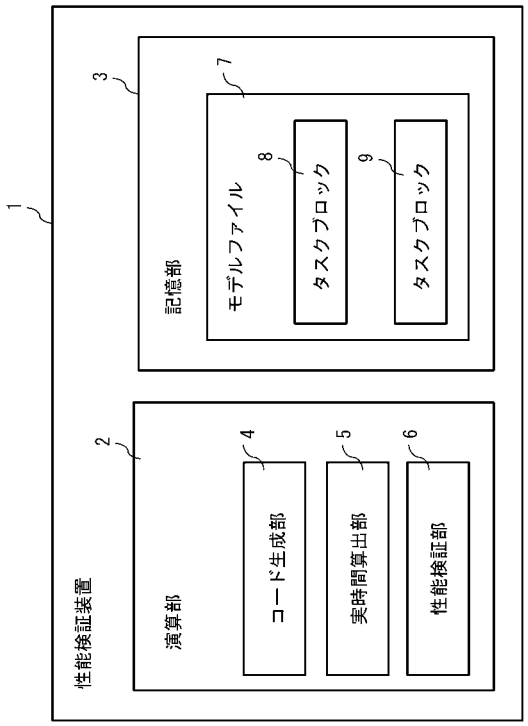
【符号の説明】

【０２２２】

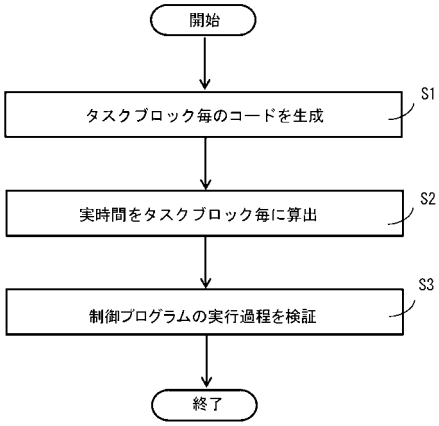
１　性能検証装置
２　演算部
３　記憶部
４　コード生成部
５　実時間算出部
６　性能検証部
７　モデルファイル
８、９　タスクブロック
１０、２７　性能検証装置
１１　演算部
１２　記憶部
１３　モデル生成部
１４　タスク単位コード生成部
１５　モデルシミュレータ部
１６　実時間算出部
１７　性能検証部
１８　プログラム格納部
１９　データ格納部
２０　マルチコアコード生成部
２１　モデルシミュレーションプログラム
２２　モデルファイル
２３　タスク単位コード生成プログラム
２４　タスク単位コード
２５　実時間計測プログラム
２６　実時間ファイル
２８　マルチコア用ブロックコード生成部
２９　マルチコアＯＳ用コード生成部
３０　リソース考慮タスクブロックコード生成部
３１　Ａｌａｒｍブロックコード生成部
３２　ＩＳＲブロックコード生成部
３３　連携ブロック生成部
３４　ターゲットコード
３５　ＯＩＬ用情報
１００　シングルコアＰＩＬＳ環境
１０１　マルチコアＯＳモデル
１０１－１～１０１－ｎ　タスクブロック
１０２　シングルコアモデル
１０２－１～１０２－ｎ　タスクブロック
１０３　シングルコアコード生成ツール
１０４　生成コード
１０５　プロジェクトファイル
１０６　連携用ブロック
１０７　ＩＤＥ
１０８　シミュレータ
１０９　エミュレータ
１１０　実測値ファイル
１１０－１～１１０－ｎ　実測値
１１１　タスク実行情報測定部
１１２　シミュレーションログ
１１３　タスク遷移表示部

１７１　タスク実行時間計算部
１７２　タスク遷移表示部
１７３　性能シミュレーションログ
２００　ＭＢＤツール
２０１　機能シミュレーションログ
２０２　ＰＥ情報
２０３　時間情報
２０４　リソース情報
２０５　プラントモデル
２０６　マルチコアＯＳモデル
２０６－１～２０６－ｎ　タスクブロック
２０７　マルチコアＳＷ
２０７－１～２０７－ｎ　タスクブロック
２０８　マルチコアＯＳ
２０９　コンパイラ
２１０　マルチコアｏｂｊｅｃｔ
２１１　シミュレータ
２１２　ＭＢＤツール
２１３　連携ブロック
２１４　ＰＥ１ターゲットコード
２１５　ＰＥ２ターゲットコード
２１６　マルチコアＯＩＬ
２１７　共通コード／プロジェクト

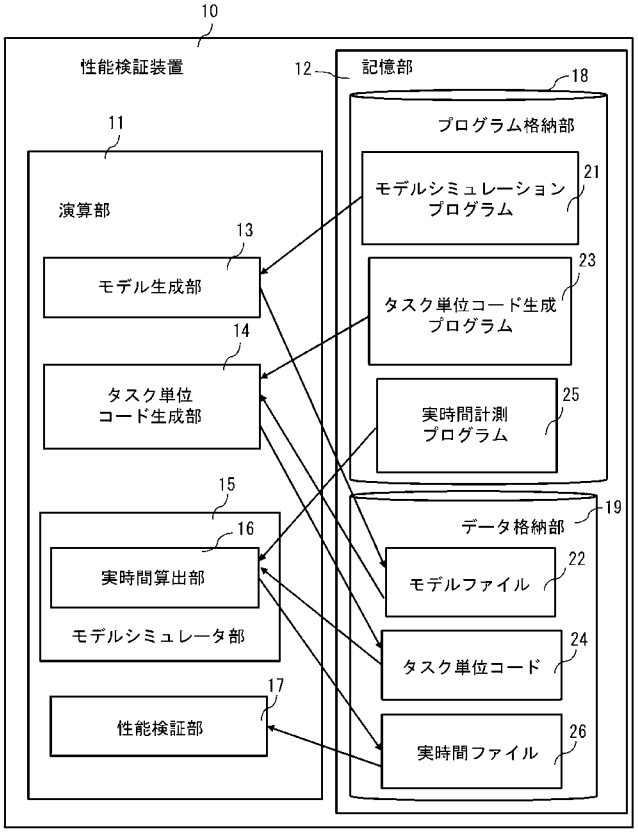
【図１】



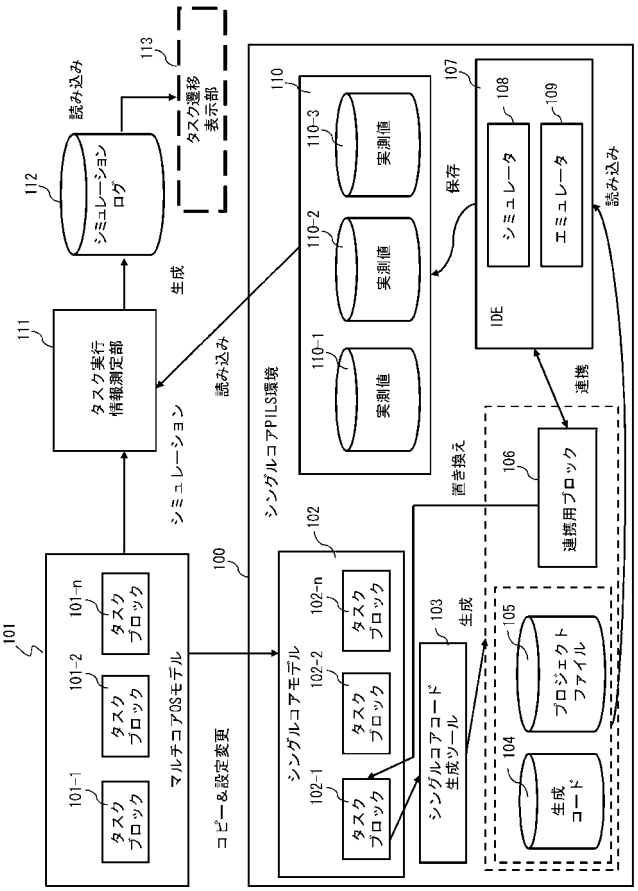
【図２】



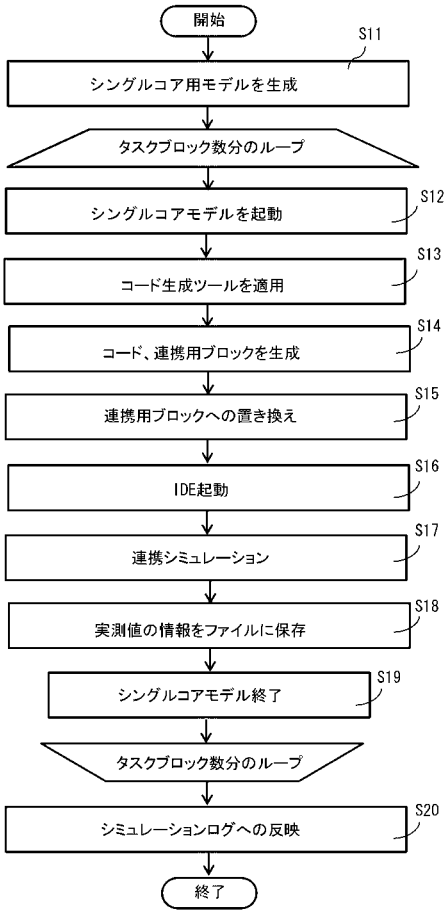
【図３】


10
性能検証装置
12
記憶部
18
プログラム格納部
11
演算部
21
モデルシミュレーション
プログラム
13
モデル生成部
23
タスク単位コード生成
プログラム
14
タスク単位
コード生成部
25
実時間計測
プログラム
15
19
データ格納部
16
実時間算出部
22
モデルファイル
モデルシミュレータ部
24
タスク単位コード
17
性能検証部
26
実時間ファイル

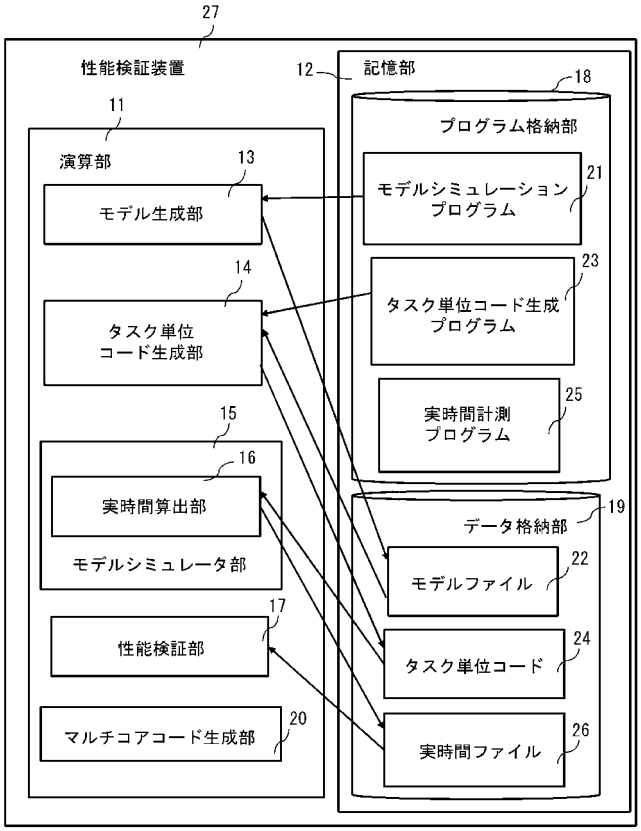
【図４】


113
タスク遷移
表示部
読み込み
112
シミュレーション
ログ
110
110-3
実測値
107
108
109
シミュレータ
エミュレータ
保存
110-2
実測値
IDE
生成
111
タスク実行
情報測定部
110-1
実測値
読み込み
シミュレーション
シングルコアPILS環境
連携
106
連携用ブロック
置き換え
102
100
101-n
タスク
ブロック
102-n
タスク
ブロック
103
生成
105
プロジェクト
ファイル
101
101-2
タスク
ブロック
シングルコアモデル
102-2
タスク
ブロック
シングルコアコード
生成ツール
104
生成
コード
マルチコアOSモデル
101-1
タスク
ブロック
コピー&設定変更
102-1
タスク
ブロック

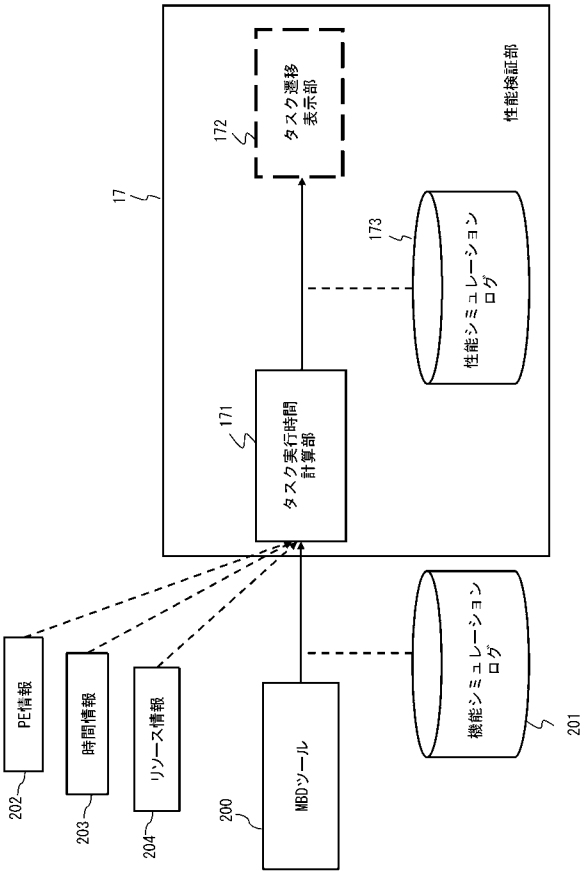
【図５】


開始
S11
シングルコア用モデルを生成
タスクブロック数分のループ
S12
シングルコアモデルを起動
S13
コード生成ツールを適用
S14
コード、連携用ブロックを生成
S15
連携用ブロックへの置き換え
S16
IDE起動
S17
連携シミュレーション
S18
実測値の情報をファイルに保存
S19
シングルコアモデル終了
タスクブロック数分のループ
S20
シミュレーションログへの反映
終了

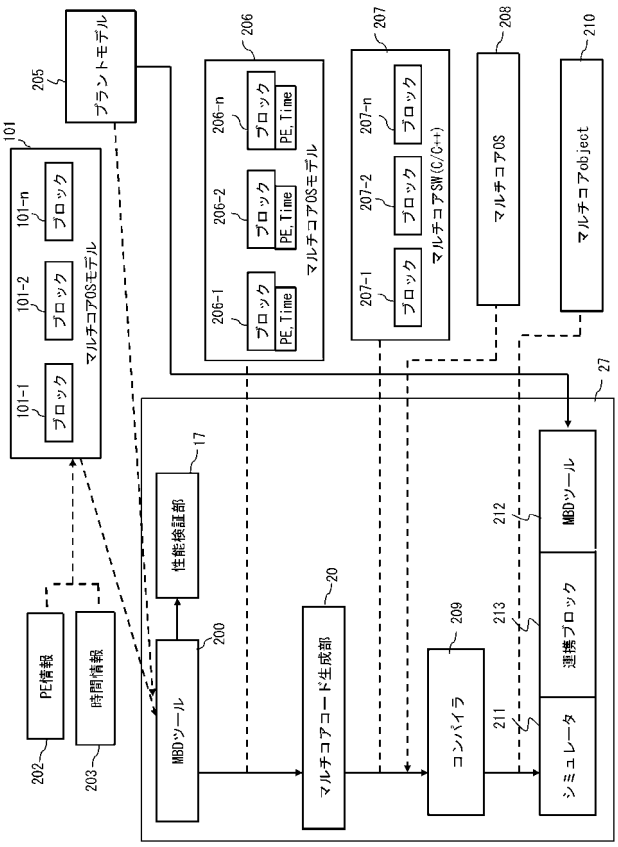
【図６】


27
性能検証装置
12
記憶部
18
プログラム格納部
11
演算部
13
モデル生成部
21
モデルシミュレーション
プログラム
14
タスク単位
コード生成部
23
タスク単位コード生成
プログラム
25
実時間計測
プログラム
15
16
実時間算出部
19
データ格納部
モデルシミュレータ部
22
モデルファイル
17
性能検証部
24
タスク単位コード
マルチコアコード生成部
20
26
実時間ファイル

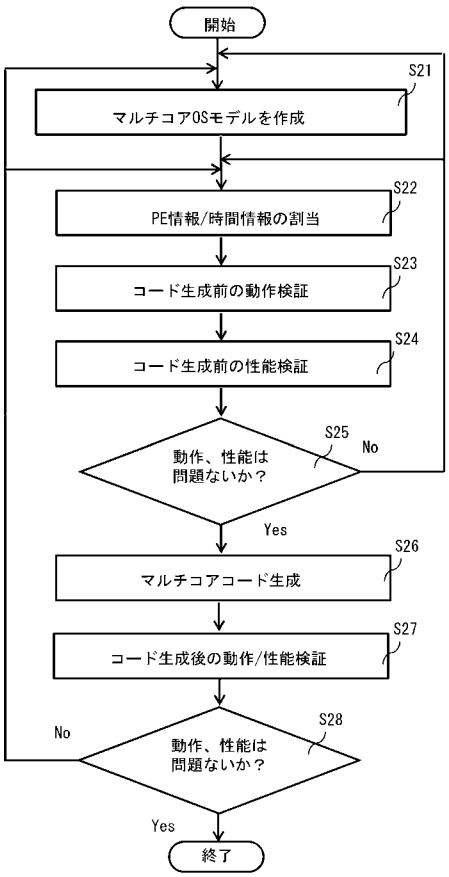
【図７】


性能検証部
17
タスク遷移
表示部
172
173
性能シミュレーション
ログ
171
タスク実行時間
計算部
PE情報
時間情報
リソース情報
MBDツール
機能シミュレーション
ログ
202
203
204
200
201

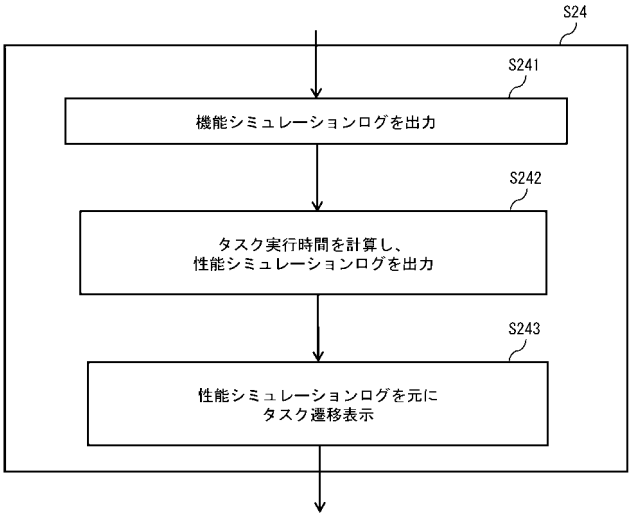
【図８】


プラントモデル
205
101
101-n
101-2
101-1
ブロック
マルチコアOSモデル
206
206-n
206-2
206-1
PE, Time
マルチコアOSモデル
207
207-n
207-2
207-1
マルチコアSW(C/C++)
208
マルチコアOS
210
マルチコアobject
27
17
性能検証部
200
MBDツール
202
203
PE情報
時間情報
20
マルチコアコード生成部
209
コンパイラ
211
212
213
MBDツール
連携ブロック
シミュレータ

【図９】


開始
S21
マルチコアOSモデルを作成
S22
PE情報/時間情報の割当
S23
コード生成前の動作検証
S24
コード生成前の性能検証
S25
動作、性能は
問題ないか？
No
Yes
S26
マルチコアコード生成
S27
コード生成後の動作/性能検証
S28
動作、性能は
問題ないか？
No
Yes
終了

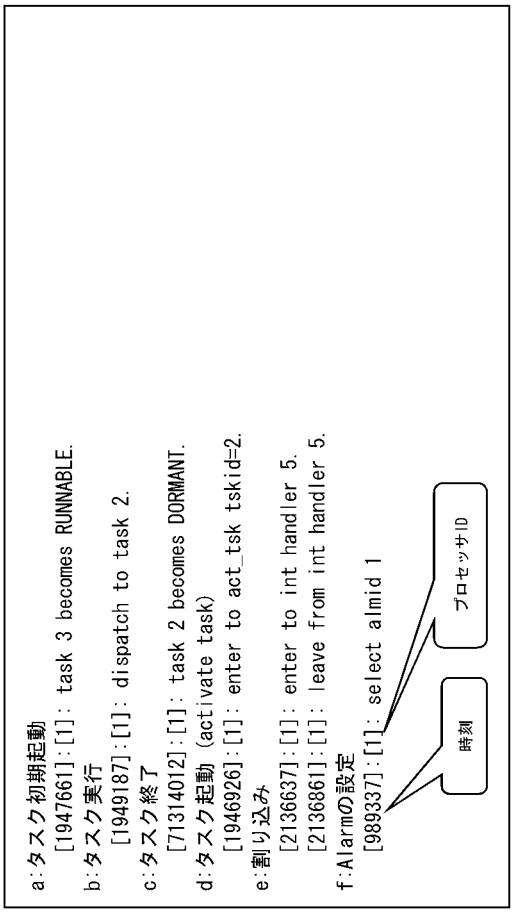
【図１０】


S24
S241
機能シミュレーションログを出力
S242
タスク実行時間を計算し、
性能シミュレーションログを出力
S243
性能シミュレーションログを元に
タスク遷移表示

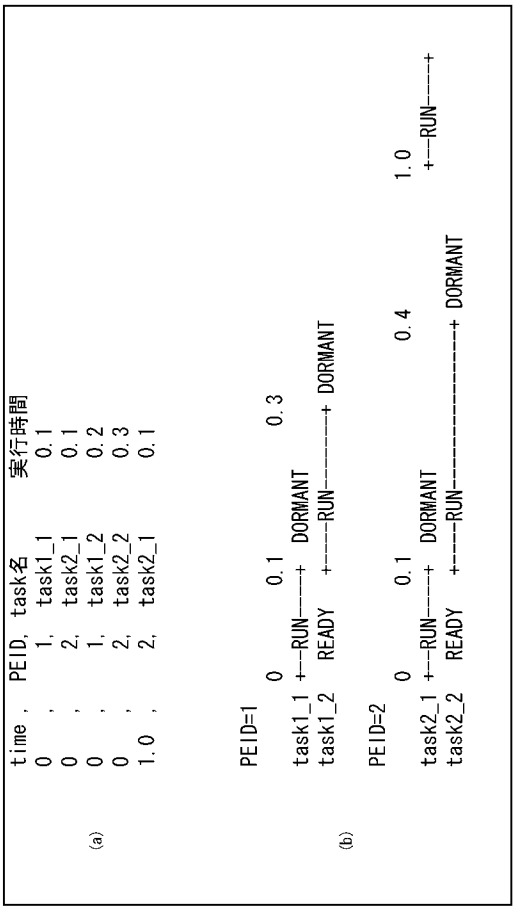
【図１１】

a:osek task,〈task名〉,〈実行時刻〉,〈割り当てPE〉,〈呼び出し先(task,alarm,ISR)〉
b:osek alarm,〈alarm名〉,〈実行時刻〉,〈割り当てPE〉,〈呼び出し先(task,alarm,ISR)〉
c:osek ISR,〈ISR名〉,〈実行時刻〉,〈割り当てPE〉,〈呼び出し先(task,alarm,ISR)〉

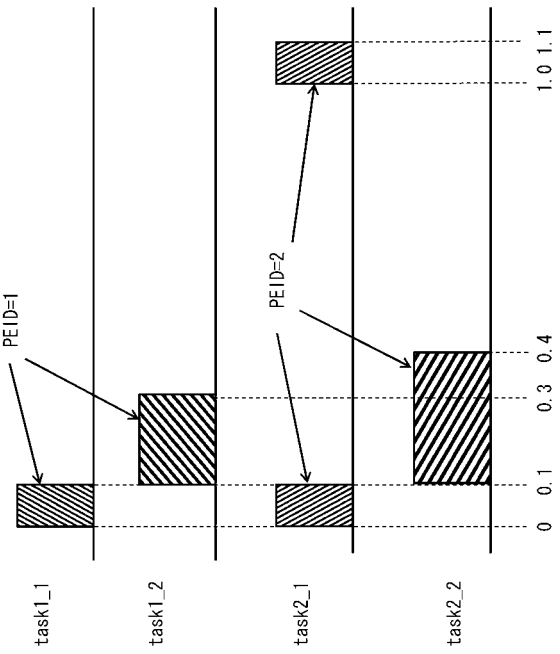
【図１２】



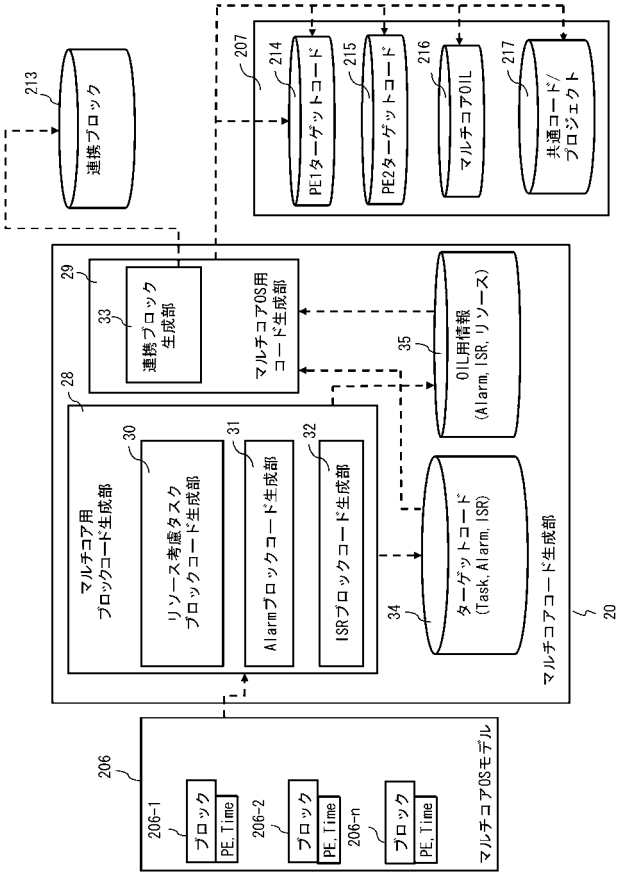
【図１３】



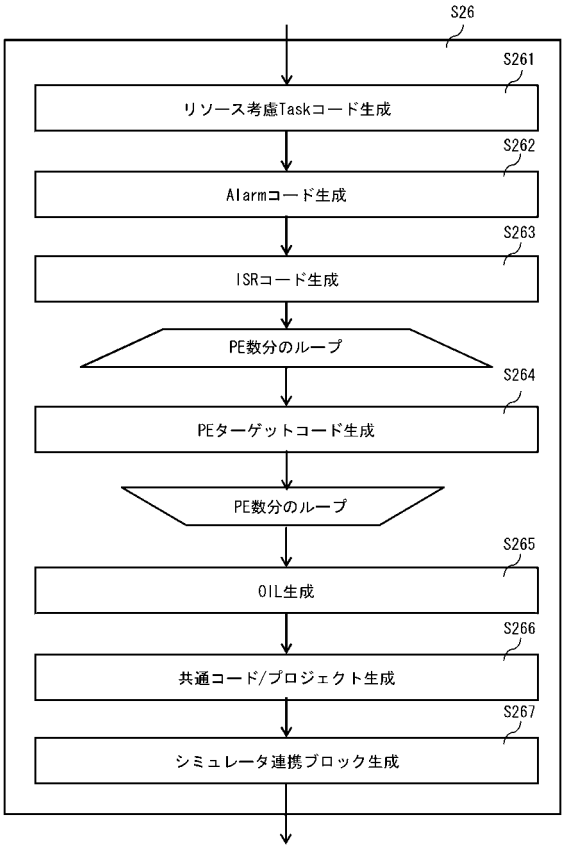
【図１４】



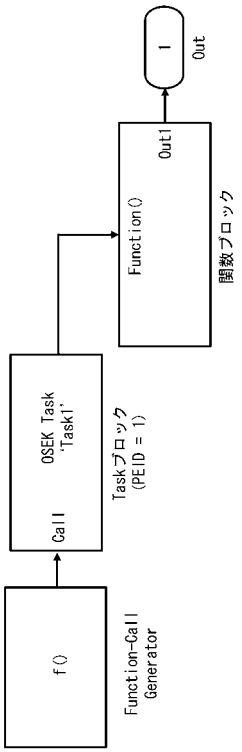
【図１５】



【図１６】



【図１７】



【図１８】

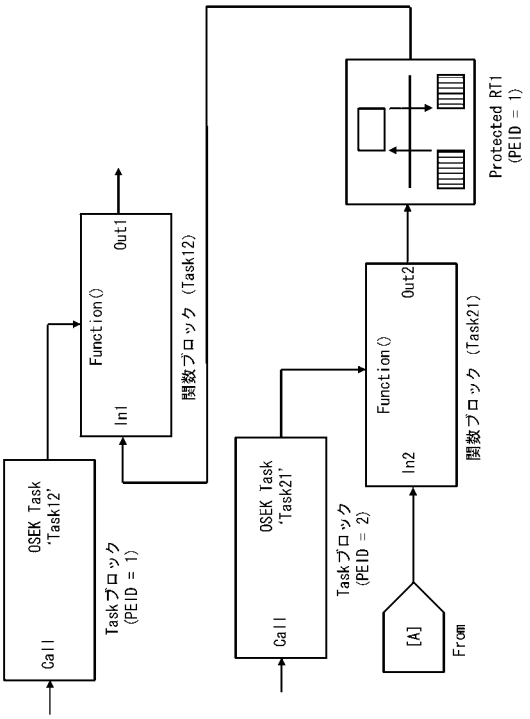
```
TASK(init) /* 初期化タスク */
{
    init_func();
    TerminateTask();
}

void init_func()
{
    ActivateTask(Task1);
}

TASK(Task1)
{
    func_task1();
    ...
    TerminateTask();
}

void func_task1()
{
    ...
}
```

(a)

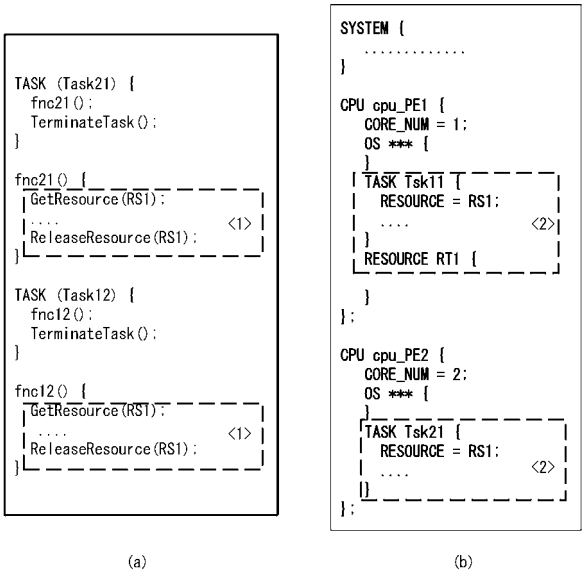
```
SYSTEM {
    ..............
}

CPU cpu_PE1 {                    <1>
    CORE_NUM = 1;
    OS *** {
    }
    TASK Tsk1 {
      RESOURCE = RS1;            <2>
      ....
    }
        RESOURCE RS1 {

    }
};

CPU cpu_PE2 {
    CORE_NUM = 2;
    OS *** {
    }
    TASK Tsk21 {
      RESOURCE = RS2;
      ....
    }
    TASK Tsk22 {
      RESOURCE = RS1;
      RESOURCE = RS2;
      ....
    }
    RESOURCE RS2 {
      ....
    }
};
```

(b)

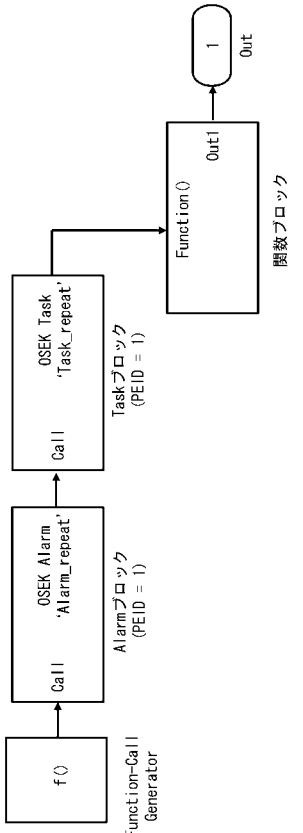
【図１９】



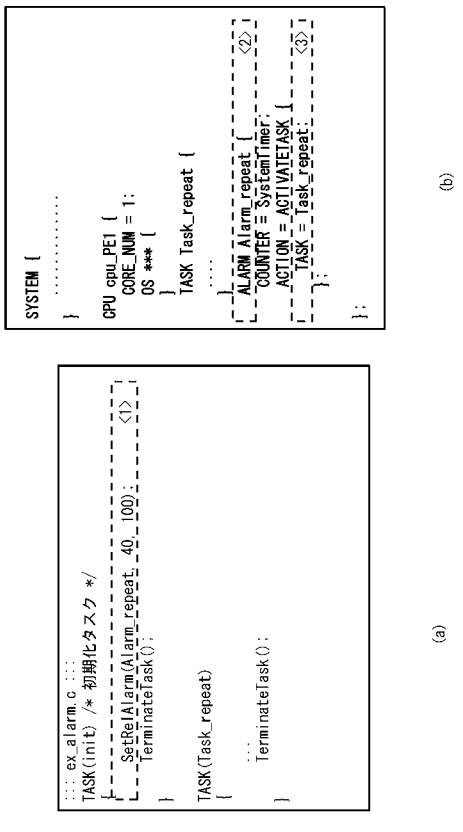
【図２０】



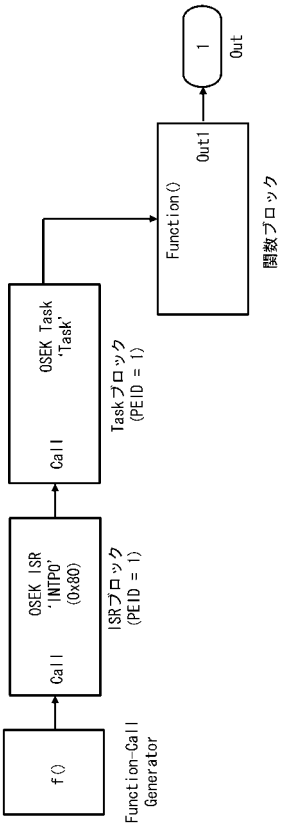
【図２１】



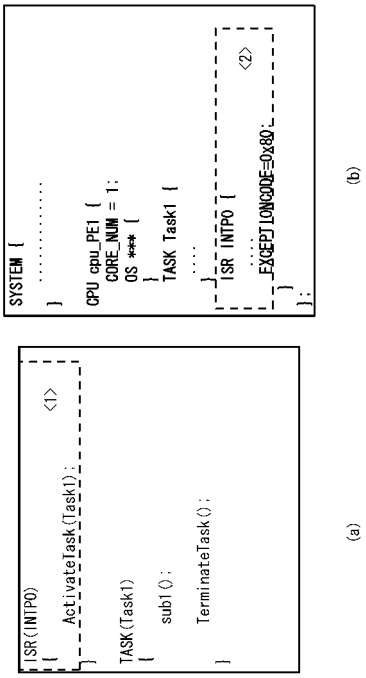
【図２２】



【図２３】


f()
Function-Call Generator
Call
OSEK ISR 'INTP0' (0x80)
ISRブロック (PEID = 1)
Call
OSEK Task 'Task'
Taskブロック (PEID = 1)
Function()
Out1
関数ブロック
1
Out

【図２４】


SYSTEM {
.....
}
CPU cpu_PE1 {
CORE_NUM = 1;
OS *** {
}
TASK Task1 {
....
}
ISR INTP0 {
....
EXCEPTION_CODE=0x80;
}
};
(b)
ISR(INTP0)
{
ActivateTask(Task1);
}
<1>
TASK(Task1)
{
sub1();
TerminateTask();
}
(a)