Vol.2017-ARC-225 No.4
Vol.2017-SLDM-179 No.4
Vol.2017-EMB-44 No.4
2017/3/9

モデルベース並列化（MBP）におけるマルチレートモデルの車載RTOS向けランタイムとコード生成

中野 友貴[1]　本田 晋也[1]　枝廣 正人[1]　鈴木 均[2]

概要：近年の自動車開発においては MATLAB/Simulink を用いてのモデルベース開発が盛んに行なわれている．一方，自動車に搭載される機能は，安全性，機能性などの社会のニーズに応えるために，年々増加傾向にある．それに伴い，個々の ECU の高性能化が求められ，結果として ECU のマルチコア化が進んでいる．また，ECU の数を減らすために機能の統合も進んでいる．モデルからマルチコア向けの並列化コードを生成するツールとしてモデルベース並列化（MBP）ツールが存在するが，複数のモデルの処理を通常の RTOS を用いて一つの ECU 上で実現すると異なるモデルの処理が互いに阻害し合う可能性がある．この問題を解決するために，時間パーティショニング（TP）機構を持つ RTOS を使うことで，それぞれのモデルの処理を時間単位で分割する．本論文では TP 機構を持ったマルチコア RTOS 向け並列化コードを生成する MBP ツールを提案する．

Runtime and code generation for Automotive RTOS of multirate model in model base parallelization

Yuuki Nakano[1]　Shinya Honda[1]　Masato Edahiro[1]　Suzuki Hitoshi[2]

Abstract: In recent years, model-based development using MATLAB/Simulink has been performed actively. Meanwhile, the functions installed in automobiles are on an increasing trend year by year to meet the needs of society such as safety and functionality. Accordingly, it is required to improve the performance of individual ECUs, and as a result, the ECUs are becoming multi-core. Integration of functions is also proceeding to reduce the number of ECUs. There is a model-based parallelization (MBP) tool as a tool to generate parallel code for multi-core from the model, but there is a possibility that processing of multiple models inhibit each other when it is realized on one ECU using normal RTOS. In order to solve this problem, we use RTOS with time partitioning (TP) mechanism to divide processing of each model by time unit. In this paper, we propose an MBP tool for generating parallel code for multicore RTOS with TP mechanism.

1. はじめに

近年の車載アプリケーション開発では，開発速度や品質の向上のために図 1 のようなモデルを作成し開発を行うモデルベース開発が盛んに行なわれるようになってきており，そのツールとして Simulink が積極的に活用されている．

一方，車載アプリケーションの高機能化に伴い，自動車の機能を実現するための制御用コンピュータである ECU は高い性能が求められているため，マルチコア化による性能向上が行なわれている．そこで，マルチコアに対応したモデルベース開発を実現する必要がある．そのため，我々はこれまで，モデルベース並列化ツール（MBP ツール）を開発してきた．本ツールは，入力となるモデルを並列化し，メニーコア向けの実装を自動生成する．

一方，搭載 ECU の増加による自動車のコスト増加および車両全体の重量が増加するといった問題も発生している．この問題を解決するために，従来異なる ECU で実現していたアプリケーションを一つの ECU で実現する動きがある．しかし複数のアプリケーションを一つの ECU で

[1] 名古屋大学 大学院情報科学研究科
Graduate School of Information Science, Nagoya University
[2] ルネサス エレクトロニクス株式会社
Renesas Electronics Corporation

Vol.2017-ARC-225 No.4
Vol.2017-SLDM-179 No.4
Vol.2017-EMB-44 No.4
2017/3/9

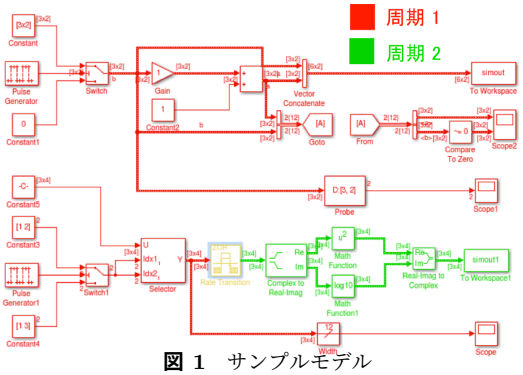

図 1 サンプルモデル

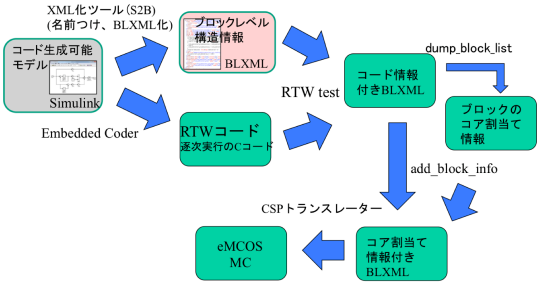

図 2 MBP ツールの設計フロー

動作させると，それぞれのアプリケーションは他のアプリケーションの動作を考慮していないため，他のアプリケーションのタスク実行を阻害したり，バス競合が発生したりしてアプリケーションのリアルタイム性を損なう可能性があるという新たな問題が発生する．

そこで本研究では，前述の MBP ツールをベースに，車載マルチコアシステムを対象に，複数のモデル（アプリケーション）を少ない干渉で実行可能な実装を生成する環境を実現した．入力とするモデルは，複数の周期で構成されているマルチレートモデルを前提とする．複数のモデルを干渉なく実行する手法として，時間パーティション（TP）機能を用いる．TP 機能とは一つの ECU 上で複数のアプリケーションを実行する際，アプリケーションの実行タイミングを時間的に分割することで他のアプリケーションを阻害しないようにする機能である．自動車業界で広く利用されている AUTOSAR 仕様の OS（A-OS）を拡張して TP 機能に対応した RTOS が開発されている [1]．この RTOS をマルチコアに拡張して，実装を生成する．

2. メニーコア向けモデルベース開発

2.1 モデルベース開発

モデルベース開発とは開発対象や制御装置をコンピュータ上での表現方法の一つであるモデルを使って設計・検証を行う手法である [2]．モデルはツール上でシミュレーションを行うことができ，設計段階で動作の確認を行うことができる．モデルベース開発のためのツールを CAE ツールと呼び，その一つである Simulink が広く用いられている．

Simulink の基本的な機能・要素として “ブロックとブロック線”，“モデルの周期”，“RateTransition ブロック” の 3 つがある．Simulink モデルの例を図 1 に示す．ブロックは特定の処理を実行する要素で，図 1 において四角や三角，丸のオブジェクトである．そしてブロック同士を接続しているのがブロック線で矢印の向きがデータのフローを表す．それぞれのブロックには周期（レート）を設定することができ，ブロックは設定された周期ごとにデータフローに沿って処理される．モデル内のブロック全ての周期が同じモデルをシングルレートモデル，異なる周期のブロックが混在するモデルをマルチレートモデルと言う．モデルにおいて周期は色で表し，図 1 は赤色と緑色の 2 種類の周期を持つマルチレートモデルである．

マルチレートモデルでは異なる周期を持つブロック間でデータの受け渡しを行う場合があり，その際データの整合性を取るために黄色いブロックの RateTransition ブロックが用意されている．具体的には，RateTransition ブロックを挿入すると，ブロック間の通信がゼロ次ホールドまたは単位周期遅れ通信となる．

車載システムは低コストや消費電力・発熱の観点からプロセッサリソースの制約が厳しい．そこでシングルレートモデルではプロセッサ性能が間に合わない場合，制御品質が許容できるレベルで特定のブロックの周期を落とすことで負荷を軽減する．実機での実行時には短周期の処理と長周期の処理を異なるタスクで実行し，短周期のタスクの優先度を高くすることにより長周期のタスクをプリエンプトすることで，短周期処理の制御周期を満たす [5]．

2.2 モデルベース並列化

モデルベース並列化とはコードベースで並列化を行うのではなく，モデルベース開発においてモデルの段階で並列化を行い，その並列化したモデル情報をもとに並列化コードを生成する手法である．

我々は，これまでメニーコアを対象に図 2 に示す設計フローの MBP ツールを実現した．実現したフローではモデルをブロック単位に分割し，ブロックの情報を BLXML（Block-Level XML）と呼ぶファイルに格納した後，ブロック毎のコード情報やコア情報を BLXML ファイルに追加する．そして CSP トランスレーターと呼ぶコード生成ツールによって BLXML からメニーコア向けの OS である eMCOS 向けの並列化コードが生成される．

3. AUTOSAR OS 仕様

AUTOSAR OS 仕様（A-OS）とは，車載システム向けのディファクトスタンダートとなっているソフトウェアプラットフォーム仕様である AUTOSAR 仕様 [4] で策定されている RTOS 仕様である．本章では本研究に関連する A-OS 仕様について説明する．

A-OS では優先度ベースのマルチタスク環境を提供する．

また，時間同期の処理を実現するためにアラームと呼ばれる機能を提供する．アラームの動作を開始することをアラームのセット，アラームの動作を停止することをアラームの停止，アラームが処理するタイミングになったことをアラームの満了と呼ぶ．アラームの満了時にイベント通知などの特定の処理を行うことができる．アラームは，カウンタと呼ばれるオブジェクトに同期して動作する．

タスク間やアラームとタスク間の同期機構としてイベントフラグがあり，タスクを待ち状態とするイベント待ちやタスクを待ち状態から解除するイベント通知によって同期を実現する．A-OS および OS オブジェクトのコンフィギュレーションは arxml ファイルという XML 形式のファイルで指定する．

マルチコア仕様も規定されており，マルチコア仕様においてはコアごとに独立して優先度ベースのスケジューリングを行う AMP 型の仕様となっている．さらに，コアをまたいで OS の API を発行することが可能である．

4. 車載マルチコア向けモデルベース開発

本研究では，前述の MBP ツールをマルチコアを用いた車載システム向けに拡張する．

4.1 対象システムと前提

対象とする車載システムは最大 4 コアのマルチコアを前提とする．OS としては車載システム向けのソフトウェアプラットフォーム仕様である AUTOSAR 仕様 [4] で策定されている RTOS 仕様（A-OS）を対象とする．さらに，車載システムの機能統合（ECU 統合）が進んでるという状況から，単一の ECU で複数のアプリケーションが動作させるケースを対象とする．

4.2 課題

既存の MBP ツールを前述の車載システムに適用させるには，以下の課題がある．

- 課題 1：A-OS 向けコード生成ができない
- 課題 2：周期をコードに反映できていない
- 課題 3：マルチタスクをサポートしていない
- 課題 4：同期通信の効率が悪い
- 課題 5：ECU 統合を考慮していない

課題 2 は，既存のコード生成では，周期の比（相対時間周期）のみが考慮され，モデル内のブロックの処理がすべて終わると，設定された周期を待たずに即座に次の処理が開始されるという問題である．

課題 3 は，既存のコード生成では，2.1 節で述べた，マルチレートモデルのマルチタスク化をサポートしていないという問題である．

課題 4 の問題は，既存のコード生成が出力するコードでは，異なるコアに割り当てられたタスク間でのデータ送受信の際は，ポーリングにより同期を実現しておりオーバーヘッドが高いという問題である．

課題 5 の問題については，ECU 統合の際，各アプリケーションは別のサプライヤーによって開発されることが多く，その際，各サプライヤーは各自が開発したアプリケーションのみを実行して検証を行う．そして統合後はすべてのアプリケーションを一つの ECU に統合する．この際，個別のコアに着目すると，OS はスケジューリングの際にアプリケーション単位ではなく，タスク単位で優先度ベースのスケジューリングを行う．そのため，マルチコア環境においては，個々のアプリケーションを単独で動作させた場合と，統合して動作させた場合とでは，各コアで動作する処理が異なる可能性がある．マルチコアにおいては，コア間でバスやメモリといったハードウェアリソースを共有するため，同時に動くプログラムにより実行時間やタイミングが影響を受ける．そのため，統合後にあたらめて動作検証が必要になり検証工数が膨大となる．

4.3 時間パーティショニングの利用

課題 5 を解決するため，時間パーティショニング（TP）機能を用いることにする．

4.3.1 時間パーティショニング

TP 機能は，ある OS アプリケーション（OSAP）で生じた時間の違反が，他の OSAP に波及することを防ぐことを目的とした機能である．A-OS 仕様では規定されておらず，TOPPERS プロジェクトからシングルプロセッサ向けに A-OS 仕様を拡張した実装が公開されている．航空機向けの RTOS である ARINC653 と同様に，OSAP ごとにプロセッサコアを利用できる時間を静的に指定することで，プロセッサ時間を OSAP ごとに分割し，OSAP のタイミングフォールトの波及を防ぐ．TP 機能に対応した OS では，静的に指定された固定周期（システム周期）の中で，OSAP ごとに静的に指定された時間（タイムウィンドウ）を割り当てる．タイムウィンドウが割り当てられた OSAP は割り当てられた時間の間で実行され，そのタイムウィンドウにおいて割り当てられた OSAP に所属するタスクを実行状態とする．タイムウィンドウの時間が終わると OSAP は処理を中断する．

4.3.2 マルチコア拡張

本研究ではマルチコアシステムを対象としているため，シングルプロセッサ向け TP 機能を持つ A-OS 仕様をマルチプロセッサ向けに拡張し，その実装を開発した（TP-OS）．TP-OS で複数のモデルを動作させた例を図 3 に示す．各モデルにはタイムウィンドウが割り当てられ，そのタイムウィンドウ内で動作する．タイムウィンドウの数やサイズ及び割り当ては全てのコアで同一とする．TP-OS は，各コアのタイムウィンドウ切り替えタイミングを同期する機能を持つ．このように実行することによって，各モデルに

Vol.2017-ARC-225 No.4
Vol.2017-SLDM-179 No.4
Vol.2017-EMB-44 No.4
2017/3/9

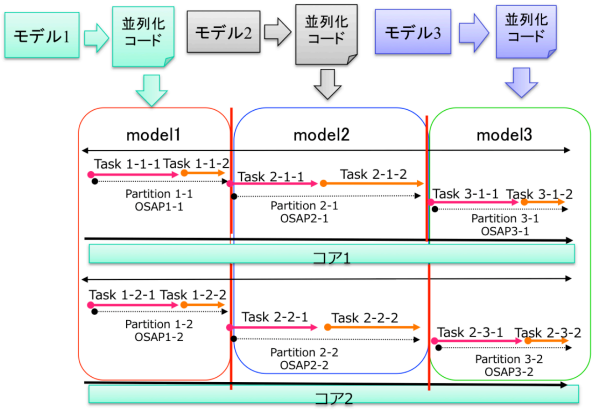

図 3　TP を使った複数モデルの動作例

タイムウィンドウを割り当て，個別に TP-OS を用いて検証した場合と，それらを統合した場合とで実行タイミングが変わらないため，課題 5 が解決される．

また，TP-OS は OSAP 単位では，マルチプロセッサ版の A-OS とコードの互換性があるため，コード生成の際には TP 機能を考慮する必要はない．

4.4　ブロック（タスク）間のデータ通信方式

課題 4 を解決するために，ポーリング以外の同期機構によるコア間の通信方式を定める．本研究では，後述するようにモデルのブロックをタスクで実行するため，以下はタスク間の通信として説明する．

TP-OS においては，タイムウィンドウ内はコア毎に優先度ベースのスケジューリングが行われており，コアが異なるタスク間の実行順序は静的に定まらないため，通信のためには，何らかの同期機構が必要となる．

まずブロック間の通信パターンを整理する．マルチレートモデルではブロック間の周期の関係は以下の 3 種類であり，それぞれのパターンで用いられる通信パターンを示す．

(1) 同じ周期のブロック間
　このパターンでは RateTransition を用いないゼロ次ホールド通信となる．シングルレートモデルにおけるモデル間の通信と同様で，周期毎に受信ブロックは，その周期で送信ブロックが出力したデータを受け取り，処理を行う．

(2) 短周期のブロックから長周期のブロック
　RateTransition を用いたゼロ次ホールド通信となり，図 4 の A に示すようにパターン 1 と同様の振る舞いとなる．

(3) 長周期のブロックから短周期のブロック
　RateTransition を用いることで，図 4 の B に示す送信ブロックの前の周期のデータを受信ブロックが受け取る単位周期遅れ通信となる．単位周期遅れとするのは，通信タイミングをタスクの最初に固定することで通信の値の一貫性を保つためである．

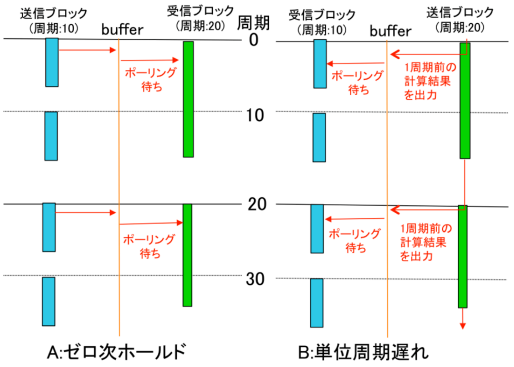

図 4　RateTransition 通信方式

これらの通信パターンの実装方式は次の通りである．

- イベントフラグ方式（ゼロ次ホールド）
- ダブルバッファ方式（単位周期遅れ）

4.4.1　イベントフラグ方式

イベントフラグ方式は，通信パターン 1,2 を対象とした方式である．このパターンでは，受信タスクがデータ更新を待つ必要があり，バス負荷を上げないために，更新を待ち状態で待ち，送信タスクは更新をイベントフラグにより通知する．受信タスクは待ち状態の解除後，値を読み込む．

短周期のタスクから長周期のタスクへデータを送る場合について説明する．イベントフラグを用いて短周期のタスクから長周期のタスクへデータを送る場合の動作図および送信タイミングの計算例を載せたものが図 5 で，これは周期が 10 のタスク（10Task）から周期が 20 のタスク（20Task）へデータが送信されている場合の例である．10Task は 20Task が起動する周期のタイミングのみデータを共有バッファに書き込む．書き込むタイミングの計算は，送信タスクが専用の変数を 2 つ用いて行う．専用変数のうちの 1 つは自タスクの現在の周期を保持する変数（v1）で，もう一つは送信先のタスクの周期を保持する変数（v2）である．専用変数の初期値は共に 0 で，v1 はタスク内の処理がすべて終わった後に自タスクの周期が加算され，v2 はデータの書き込みが行われた場合のみ受信タスクの周期が加算される．そしてデータの送信は，**（v1 ＋ 自タスク周期 ≤v2 ）**の条件判定で偽となった場合に行われる．受信側は起動した際，毎回受信処理を行う．

4.4.2　ダブルバッファ方式

ダブルバッファ方式は，送信側の周期が長いパターン 3 を対象とした方式で，このパターンでは受信タスクはデータの更新を待つ必要はないが，送信側の周期に応じて読み込むバッファを切り替えるようにし，RateTransition の単位周期遅れを実現するための方式である．ダブルバッファを用いた動作図およびバッファ切り替えの計算例を載せた図が図 6 である．

タスクのバッファ選択の方法について説明する．送信タスクは起動した周期で計算した結果をタスクの処理の最後

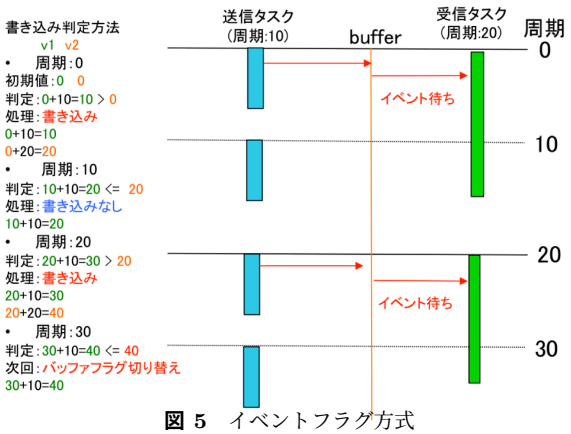

図 5 イベントフラグ方式

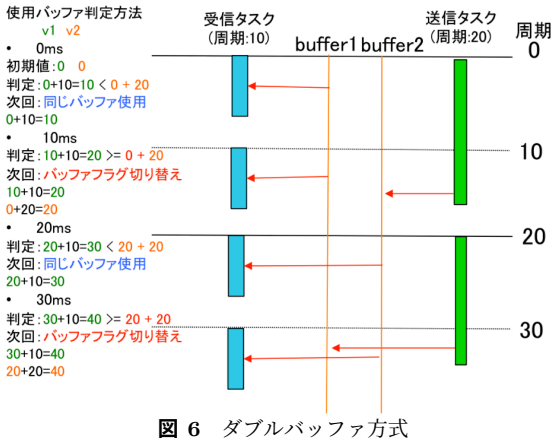

図 6 ダブルバッファ方式

に，前回書き込んだバッファとは異なるバッファに書き込む．書き込む先のバッファの切り替えは，バッファに書き込んだ後にバッファ判定フラグの値を 0 と 1 で切り替えることで行う．受信タスクは送信タスクが前回の周期で計算結果を書き込んだバッファから読み込む必要がある．その方法として，まず最初の周期は受信側が読み込むバッファを送信側と逆側のバッファから読み込むように静的に決めておく．1 周期目以降は毎回の読み込み後に，次回どちらのバッファから読み込むかを判断する．そのバッファ判定フラグの切り替えは，専用の変数を 2 つ用いて行う．専用変数のうちの 1 つは自タスクの現在の周期を保持する変数（v1）で，もう一つは送信タスクの周期を保持する変数（v2）である．専用変数の初期値は共に 0 で，v1 はタスク内の処理がすべて終わった後に自タスクの周期が加算され，v2 はフラグの切り替えが行われた場合のみ送信タスクの周期が加算される．そしてフラグの切り替えバッファからデータを読み込んだ後，**（v1 ＋ 自タスク周期 <v2 ＋ 送信側のタスクの周期）**の条件判定で偽となった場合にバッファ判定フラグの切り替えが行われる．受信側のバッファの切り替えタイミングは，受信タスクが次回起動するまでにまたは次回の起動と同時に送信タスクが起動する場合に行われるため，それぞれのタスクが同時に同じバッファを扱うことはない．また，このダブルバッファ方式では送信タスクが書き込んだ値は再び送信タスクが起動した時以降に読み込まれるため，単位周期遅れも実現することができる．

4.5 コード生成

本節では，車載マルチコア向けのコード生成について説明する．定めたコード生成は，2.2 章で述べた CSP トランスレーターをベースに実装した．

まず，課題 3 を解決するため，コア毎に，そのコアに割り付けられたブロックの周期毎にタスクを生成する．2.1 節で説明したマルチレートモデルの振る舞いを実現するため，生成したタスクには，周期の短いタスクから高い優先度を順に割り当てる．

次に，問題 2 を解決するために，タスク毎に割り当てられた一連のブロックの処理を終了した後にタスクを待ち状態とする．そしてそのタスクの動作周期になるとそのタスクの待ち状態を解除させる．この周期的な起床は，アラーム機能とイベント機能を組み合わせることによって実現する．アラームは周期毎かつコア毎に生成される．カウンタはマスタプロセッサ（コア 1）で実行されるため，各コアのアラームは同期して実行される．

最後に，課題 1 を解決するため，前述のタスク生成やタスク間の通信機構を，A-OS を対象としてコード生成する機構を実装した．さらにタスクの記述やタスクのコンフィギュレーションのための arxml ファイルを生成する．

5. 評価

A-OS を用いたタスク実行タイミング検証と，シミュレーションと実機での値の出力結果比較を行う．実機での動作ではルネサスエレクトロニクス社製の車載マイコンである RH850/F1H を使い，A-OS として AUTOSAR 仕様準拠の RTOS である ATK2-SC1-MC（MC-OS）およびそれを拡張した ATK2-SC1-MC-TP（TP-OS）を用いた．

5.1 タスク実行タイミング検証

MC-OS と TP-OS でそれぞれ同じモデルを動作させた場合のタスクの実行タイミングの検証を行う．my_sldemo_engine と my_sldemo_metro という二つのモデルを同時に動作させ，その動作を TLV というタスクの実行タイミングを可視化するツールにより確認した．my_sldemo_engine はサンプルモデルである sldemo_engine モデルの周期設定を連続から離散に変更し，さらにマルチレートモデル化したモデルで，my_sldemo_metro はサンプルモデルである sldemo_metro モデルの周期設定を連続から離散に変更したシングルレートモデルである．

5.1.1 タスクおよびシステム周期・タイムウィンドウ情報

前述の 2 つのモデルでは計 4 つのタスクが動作する．タスク名は 3 つの要素で決定され，CoreN の N が割り当てコア，MTask の M が周期（単位は ms），CoreN より前の部分がモデル名となっている．したがって，

my_sldemo_engine_Core1_1000Task は my_sldemo_engine モデルから生成されたタスクで，コア 1 に割り当てられ，その周期は 1000ms である．タスクの優先度は my_sldemo_engine のタスクのほうが my_sldemo_metro のタスクより低い．TP-OS で動作させる場合のシステム周期は各コアとも 1000ms で，400ms のタイムウィンドウをそれぞれのコアで 2 つずつ設定した．そして一つ目のタイムウィンドウに my_sldemo_engine のタスクを二つ目のタイムウィンドウに my_sldemo_metro のタスクを割り当てた．

5.1.2 結果

二つのモデルを動作させた結果を図 7 および図 8 に示す．図 7 は MC-OS 上で動作させた場合の結果を拡大したものである．また図 8 は TP-OS 上で動作させた場合の結果で，その全体図と拡大図である．図 7 と図 8 ではその拡大の比率は異なっている．図の緑色の四角の部分がタスクが実行されている時間で，赤い線の時間が次回の周期までの待ち状態，橙色の線の時間が実行可能状態またはデータ同期待ち状態である．図の背景の色がコアを表し，白がコア 1，赤がコア 2 である．図 7 より MC-OS 上で動作させた場合，異なるモデルのタスクが同時に実行されているタイミングがあることがわかる．一方 TP-OS 上で動作させた場合，図 8 からわかるように，モデルごとに動作タイミングが完全に分離している．また，図 8 より処理が一定周期ごとに行われていることも確認出来る．

5.2 シミュレーションと実機での値の出力結果比較

Simulink のシミュレーション機能が出力する結果と並列化コード生成ツールが生成した並列化コードを実機上で MC-OS および TP-OS を使って動作させ，値を出力させた結果の比較を行う．使用したモデルは my_sldemo_engine であり，結果を表 1 に示す．すべての場合で出力結果が同じとなっていることがわかる．

5.3 考察

実行タイミング検証では，TP-OS において異なるモデルのタスクは同時には実行されていないことが確認でき，出力コードが TP 機能を活用できていることがわかる．さらにタスクが 1 周期分の処理が終わった後に終了し，次の周期で起動していることから実時間周期が正常に実行されていることが確認出来る．出力結果比較からは並列化コード生成を経て，モデルの処理を並列コードとして正しく実現できていることが確認できた．

6. おわりに

本研究では，車載マルチコア向けモデルベース開発のための並列化コード生成ツールの作成を行った．作成したツールを用いてコード生成を行い，TP-OS を用いて動作させた結果，シミュレーション結果と一致した．さらに，

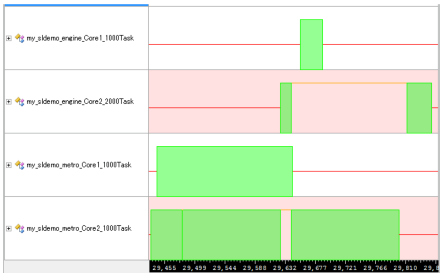

図 7 MC-OS 上でのサンプルモデル実行結果

図 8 TP-OS 上でのサンプルモデル実行結果

表 1 my_sldemo_engine 出力結果（m rad/sec）

周期	シミュレーション	MC-OS	TP-OS
1	0	0	0
2	0	0	0
3	2.0004	2.00038	2.00038
4	2.0004	2.00038	2.00038
5	1.6507	1.6507	1.6507
6	1.6507	1.6507	1.6507
7	2.3199	2.3199	2.3199
8	2.3199	2.3199	2.3199

実時間周期が実現できていること，モデルごとに処理タイミングが分離できていることが確認できた．

参考文献

[1] 次世代車載システム向け RTOS 外部仕様書,https://www.toppers.jp/docs/tech/ngka_spec-01-320.pdf（2016 年 1 月 24 日参照）
[2] モデルベースデザイン,https://jp.mathworks.com/help/simulink/gs/model-based-design.html（2016 年 1 月 24 日参照）
[3] RH850,https://www.renesas.com/ja-jp/products/microcontrollers-microprocessors/rh850/rh850f1x/rh850f1h.html（2016 年 1 月 24 日参照）
[4] AUTOSAR,http://www.autosar.org/（2016 年 1 月 24 日参照）
[5] Japan MATLAB Automotive Advisory Board, ”CONTROL ALGORITHM MODELING GUIDELINES USING MATLAB, Simulink, and Stateflow Version 4.01（日本語版）”, 2015.