Vol.2018-ARC-230 No.23
Vol.2018-SLDM-183 No.23
Vol.2018-EMB-47 No.23
2018/3/8

# タイムスタンプ付きメッセージを用いた時間駆動分散処理環境

市村 歩[1,a)]　兪 明連[1,b)]　横山 孝典[1,c)]

**概要：**
本論文では，通信時間が変動するネットワークを用いた分散制御システム向けの時間駆動分散処理環境について述べる．ジッタの少ないリアルタイム性のある分散処理を実現する手法として，時間駆動（Time-Triggered）ネットワークを用いた時間駆動アーキテクチャがある．しかし最近では，無線通信を用いた分散型組み込み制御システムが増えつつあり，通信時間の変動するネットワークを用いた環境でも高いリアルタイム性を実現することが求められている．我々は，分散制御システムにおいてジッタのない処理が要求されるのは実世界と直接接する入出力タスクのみであり，その他のタスクはある程度のジッタを許容できることに着目した．そして，入出力処理を行うタスクのみを物理時間に同期して実行し，それ以外のタスクはタイムスタンプ情報を利用した論理的な時間に基づいて実行する分散処理環境を提案する．そしてその実現のため，タイムスタンプ付きメッセージを扱う機能や論理時刻を管理する機能を有するミドルウェアを開発し，その有用性を確認した．

## 1. はじめに

サイバーフィジカルシステム（Cyber-Physical System）は実世界と直接相互作用するため，実時間に基づいた処理が要求される [1]．特に，自動車制御システム等の組み込み制御システムはハードリアルタイムなサイバーフィジカルシステムであり，制御処理における遅延やジッタは制御性能低下の原因となる [2] ため，デッドラインを守るとともにジッタの少ない分散処理環境が求められる．

ジッタの少ない分散処理を実現する手法として，全ノードが同期した時間に従って処理を行う時間駆動アーキテクチャ（Time-Triggered Architecture）が提案されている [3]．時間駆動アーキテクチャは，TTP（Time-Triggered Protocol）[4] や FlexRay[5] のような，時刻同期機構を持つとともに通信時間が変動しない TDMA（Time Division Multiple Access）方式に基づく時間駆動ネットワークを用いることで，全ノードで同期した処理が可能である．時間駆動アーキテクチャを実現するための基本ソフトウェアも提案されており，OSEK/VDX は OSEKtime と呼ばれる時間駆動 OS の仕様を策定している [6]．また，FlexRay を用いて全ノードで同期したシステム時刻に従ってタスク管理が可能な分散リアルタイム OS も提案されいる [7]．

ところが最近，車々間通信や路車間通信を用いた自動走行システムや車間距離制御システムのように，無線通信を利用した分散型組み込み制御システムが増えている．今後は，通信時間の変動が大きい無線通信を多用するとともに，クラウド等との連携も予測され，全ての処理を同一の時間に従って同期して実行することは現実的ではない．

そこで本研究の目的は，通信時間の変動を許容しながら，全ての処理の同期実行を必要とせずに，時間駆動アーキテクチャと同等のリアルタイム性を実現する分散処理環境を開発することである．

一般に分散型組み込み制御システムは，入力処理を行うタスク（以下，入力タスクと呼ぶ），出力処理を行うタスク（以下，出力タスクと呼ぶ），入力処理や出力処理を行わず算出処理のみを行うタスク（以下，算出タスクと呼ぶ）から構成される．我々は，直接制御対象とやりとりする入力タスクと出力タスク（以下，合わせて入出力タスクと呼ぶことがある）はジッタのない処理が要求されるが，算出タスクはある程度のジッタを許容できることに着目し，その性質を利用した分散処理環境を提案する．

本分散処理環境では，入出力タスクのみを時間駆動動作とし，算出タスクは同期を必要としないメッセージ受信イベントで起動する．また，通信時間が変動したり，必ずし

[1] 東京都市大学
Tokyo City University
[a)] g1681504@tcu.ac.jp
[b)] myoo@tcu.ac.jp
[c)] tyoko@tcu.ac.jp

Vol.2018-ARC-230 No.23
Vol.2018-SLDM-183 No.23
Vol.2018-EMB-47 No.23
2018/3/8

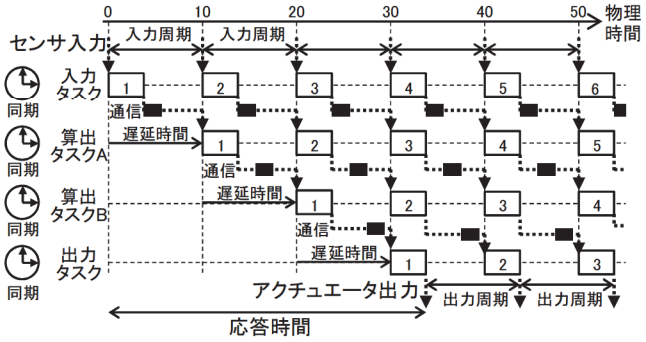


**図 1** 時間駆動分散処理のタイムチャート

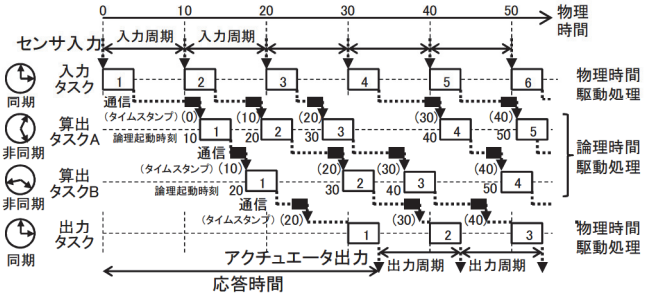


**図 2** タイムスタンプを用いた時間駆動分散処理のタイムチャート

も通信順序が保証されないネットワークに対応するため，ノード間でやりとりするメッセージにタイムスタンプを付加する．さらに，非同期動作をする算出タスクを時間駆動アーキテクチャに対応づけて管理するため，仮想的な論理時間を導入する．以上により，通信時間の変動を許容しながら，ジッタの少ない制御処理を実現する．

そして，上記分散処理環境を実現するための機能を有する分散処理ミドルウェアを開発する．本ミドルウェアは自動車制御分野の標準的なリアルタイム OS （Real-Time Operating System, RTOS）である OSEK OS 上で動作する．対象とするネットワークは有線ネットワークの CAN[9] と無線ネットワークの ZigBee[10] とする．

本論文の構成は以下の通りである．まず 2 節で，タイムスタンプ付きメッセージおよび論理時間を用いた分散処理の基本方式について説明する．次に 3 節で，分散処理ミドルウェアの構成および動作，実装について説明する．そして 4 節で開発したミドルウェアの評価を行い，5 節でまとめを述べる．

## 2. 基本方式

### 2.1 概要

時間駆動ネットワークを用いた時間駆動分散処理のタイムチャートの例を**図 1** に示す．時間同期した 4 つのノードに，それぞれ入力タスク，算出タスク A，算出タスク B，出力タスクが分散配置されている．タスクの周期は全て 10 である．入力タスクはサンプリング周期 10 でセンサからの入力処理を行い，算出タスク A にメッセージを送信する．算出タスク A は入力タスクからのメッセージを受信して処理を行い，算出タスク B にメッセージを送信する．算出タスク B は算出タスク A からのメッセージを受信して処理を行い，出力タスクにメッセージを送信する．そして出力タスクは，算出タスク B からのメッセージを受信し，出力周期 10 でアクチュエータへの出力処理を行う．いずれのタスクもジッタのない動作をしている．また，時間駆動ネットワークを使用しているため，メッセージ通信時間の変動もない．

周期タスクの 1 回の実行をジョブと呼ぶが，図 1 には，タスクの実行を表す矩形の中にジョブの番号を記載している．あるタスクのジョブの起動時刻と，他ノード上の後続のタスクのジョブの起動時刻の差を遅延時間と呼ぶことにする．この例では，タスク間の遅延時間はいずれも 10 で，周期と同じである．このため，入力タスクの 2 番目のジョブの起動時刻と，算出タスク A の 1 番目のジョブの起動時刻が一致している．それ以降の後続タスクのジョブも同様である．

本論文で提案するタイムスタンプ付きメッセージと論理時間を導入した時間駆動分散処理のタイムチャートの例を**図 2** に示す．入力タスクと出力タスクは同一の時間に従って，同期して実行される．この同期した時間を本論文では物理時間と呼ぶ．一方，2 つの算出タスクは非同期に動作し，メッセージの受信イベントで起動される．メッセージ通信時間が変動するネットワークを用いるため，2 つの算出タスクにはジッタが発生している．

本分散処理環境では，仮想的な論理時間を導入し，それによって算出タスクを管理する．論理時間によって表現した起動時刻を論理起動時刻と呼ぶ．図 2 には，2 つの算出タスクの各ジョブが開始するところに論理起動時刻を記している．図 1 と対応させるとわかるように，論理起動時刻は，時間駆動分散処理における各ジョブの起動時刻と一致するように与える．入力タスクと出力タスクには論理起動時刻を記載していないが，それらは論理時間と物理時間が常に一致しているものと見なす．

本論文では，システム全体で同期した時間（物理時間）に従ってタスクを起動する処理を物理時間駆動処理，メッセージ受信イベントでタスクを起動して論理時間に従って管理する処理を論理時間駆動処理と呼ぶことにする．すなわち本分散処理環境では，入出力タスクを物理時間駆動処理，算出タスクを論理時間駆動処理で動作させる．

なお，物理時間駆動処理を行うノードには，FlexRay を用いた時刻同期機能を持つ分散 RTOS[7] や，GNSS（Global Navigation Satellite Systems）を用いた時刻同期機能を持つ RTOS[11] を使用することを想定している．

### 2.2 タイムスタンプ付きメッセージ

本分散処理環境では，各ジョブがメッセージを送信する

Vol.2018-ARC-230 No.23
Vol.2018-SLDM-183 No.23
Vol.2018-EMB-47 No.23
2018/3/8

ときに，そのジョブの論理起動時刻をタイムスタンプとして付加する．図 2 には，各メッセージに付加されているタイムスタンプの値を，受信側にかっこ書きで記している．前述のように入力タスクの場合は論理時間と物理時間が一致するので，入力タスクが送信するメッセージには物理時間で表現した起動時刻がタイムスタンプとして付加されている．

論理時間で表現した遅延時間を論理遅延時間と呼び，時間駆動アーキテクチャの遅延時間に対応させる．論理遅延時間は定数とし，設計時にタスクの実行時間と通信時間を考慮して決定する．最悪の場合でもデッドラインを守ることを保証するには，最悪実行時間および最悪通信時間から論理遅延時間を決定する必要があるが，通信時間が変動するネットワークの場合は，平均通信時間およびデッドラインを守るための余裕時間から決定するのが実用的と考えている．論理遅延時間は定数なので，コンフィギュレーションデータとして予め与えておくことができる．

受信したメッセージに付加されているタイムスタンプと，そのメッセージを処理するジョブの論理起動時刻と，論理遅延時間の間には以下の関係がある．

論理起動時刻 = タイムスタンプ + 論理遅延時間　（式 1）

次回実行するジョブの論理起動時刻を次回論理起動時刻と呼ぶ．メッセージ受信時には，メッセージに付加されたタイムスタンプを読み出し，式 1 を用いて受信したメッセージに対応する論理起動時刻を求める．そしてそれが，次回論理起動時刻と一致した場合にジョブを起動し，そのメッセージを用いて処理を行う．一致しない場合は，メッセージの転送順序が乱れたことを意味するので，その時点ではジョブを起動せず，一致するメッセージの到着を待ち，到着した時点で，それまで受信したメッセージ数分のジョブを実行する．これにより，必ずしも通信順序が保証されないネットワークにも対応できる．

なお図 2 の例は，簡単のため算出タスクはひとつの受信メッセージしか使用していないが，実際のシステムでは複数の受信メッセージを用いる算出タスクもある．本分散処理環境でも複数のメッセージを読み出すことは可能である．ただし，どのメッセージの受信イベントでタスクを起動するかについては，現時点では，ひとつのメッセージを設計時に指定することとする．また，算出タスクの周期とタスク起動用メッセージの通信周期が異なる場合もあるが，現時点では，タスクの周期は通信周期と等しいか整数倍である場合を対象とする．

また図 2 は，ひとつのノード上にひとつのタスクしかない例であるが，実際には複数のタスクが存在するのが普通である．ひとつのノード上に入出力タスクと算出タスクの両者が存在する場合もあり，その場合は，ひとつのノード上に物理時間駆動処理と論理時間駆動処理の両者を動作させることになる．

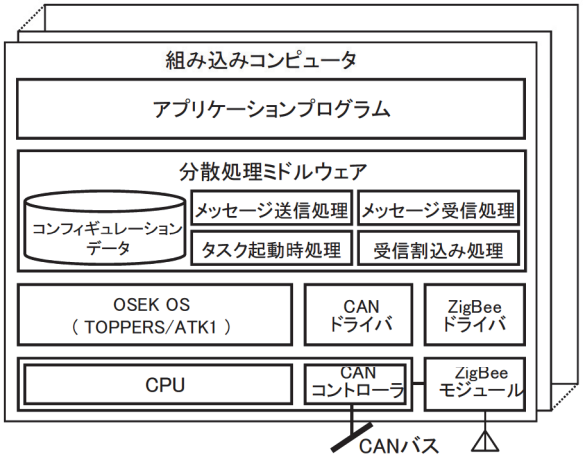


**図 3**　ソフトウェア構成

## 3. 分散処理ミドルウェア

### 3.1 ソフトウェア構成と API

本分散処理環境のソフトウェア構成を**図 3** に示す．開発する分散処理ミドルウェアは OSEK OS 上で動作する．具体的には，TOPPERS プロジェクトが開発した OSEK OS 仕様準拠の TOPPERS/ATK1[12] を用いる．また，ネットワークは CAN と ZigBee に対応する．

分散処理ミドルウェアは，受信割り込み処理，タスク起動時処理，メッセージ送信処理，メッセージ受信処理から成る．またそれらの処理が動作するのに必要とするタスクやメッセージに関する情報は，コンフィギュレーションデータとして記憶する．コンフィギュレーションデータには，タスクの種類（入力タスク，出力タスク，算出タスクのいずれか），タスク ID，タスクの周期，タスクが受信するメッセージ ID，メッセージ通信の周期，遅延時間，各情報の初期値等が記憶される．

OSEK OS のコンフィギュレーションデータは，アプリケーション開発者が記述した OIL ファイルから，システムジェネレータにより生成される [13]．ミドルウェアのコンフィギュレーションデータも，アプリケーション開発者が記述した情報に基づいて，コンフィギュレーションツールにより生成する．ただし，コンフィギュレーションツールは未開発のため，現時点ではコンフィギュレーションデータのソースコードを手作業で記述している．

受信割込み処理はメッセージ受信時に起動される割込み処理で，メッセージを受信して，タイムスタンプ情報に応じて受信バッファに記憶するほか，算出タスクを起動するための処理も行う．タスク起動時処理は，アプリケーションタスクの処理を実行する前に行う処理で，タイムスタンプや論理起動時刻の更新を行う．メッセージ送信処理はアプリケーションがメッセージを送信する時に呼び出す処理で，タイムスタンプを付加してメッセージを送信する．メッセージ受信処理はアプリケーションがメッセージを受

Vol.2018-ARC-230 No.23
Vol.2018-SLDM-183 No.23
Vol.2018-EMB-47 No.23
2018/3/8

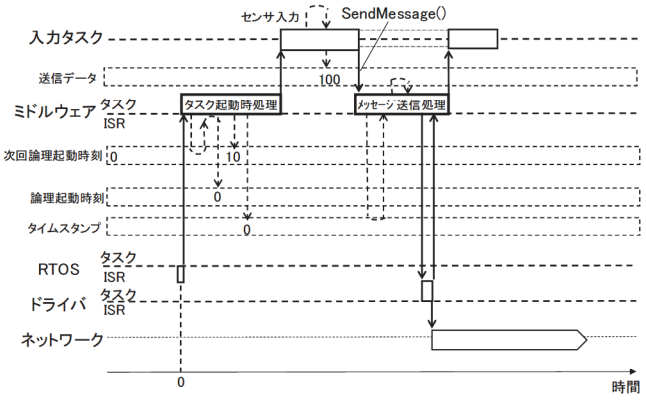


**図 4** 入力タスクの動作

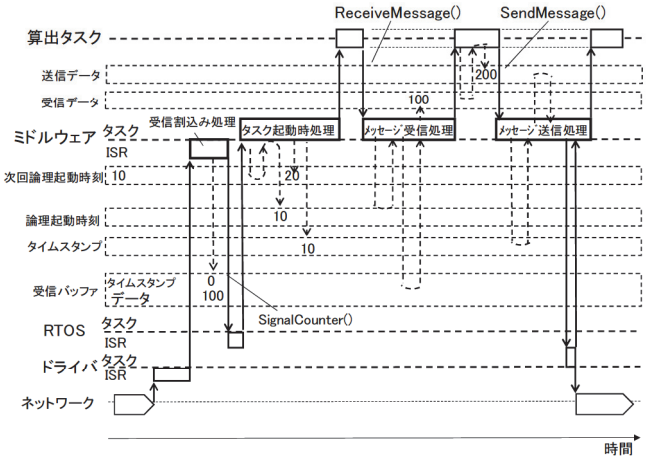


**図 5** 算出タスクの動作

信する時に呼び出す処理で，呼び出したタスクに対応したメッセージを返す．それらの処理は，入力タスク，算出タスク，出力タスクの場合で動作が異なるため，詳細は 3.2 節で述べる．

メッセージ送信処理およびメッセージ受信処理の API は，OSEK/VDX が制定した通信ソフトウェア仕様である OSEK COM[14] との互換性を考慮し，同じインタフェースとする．それらの API を以下に示す．いずれも，その第 1 引数はメッセージ ID，第 2 引数はアプリケーション中のメッセージデータ記憶領域へのポインタ，戻り値はエラー等のステータス情報である．

StatusType SendMessage(MessageID mes, DataRef pd)

StatusType ReceiveMessage(MessageID mes, DataRef pd)

### 3.2 ミドルウェアの動作

入力タスクの場合のミドルウェアの動作を**図 4** を用いて説明する．ミドルウェア内には，入力タスクがメッセージを送信する時に付加するタイムスタンプ，論理起動時刻，次回論理起動時刻を記憶する領域があり，この例では，タスクの周期は 10 で，次回論理起動時刻の初期値 0 が記憶されている．

RTOS によって入力タスクが起動されると，入力タスクの処理の前に，ミドルウェアのタスク起動時処理を実行する．タスク起動時処理はまず，記憶されている次回論理起動時刻の値（この場合は 0）を読み出して論理起動時刻として記憶する．次に，それに周期を加えて次回論理起動時刻を更新する（この例では 10 に更新）．また，タスクの起動時刻（この例では 0）を送信メッセージ用のタイムスタンプとして記憶する．その後，入力タスクの処理を実行する．

入力タスクがメッセージを送信するために SendMessage() を呼び出すと，ミドルウェアのメッセージ送信処理は，引数で指定された送信データにタイムスタンプを付加して送信メッセージを作成し，ドライバを呼び出してネットワーク上に送信する．

次に，算出タスクの場合のミドルウェアの動作を**図 5** を用いて説明する．メッセージを受信すると ISR（割込み処理）が起動され，ドライバの処理を行った後，ミドルウェアの受信割込み処理を実行する．受信割り込み処理は，受信したメッセージのタイムスタンプが表す時刻順に，タイムスタンプとデータを受信バッファに記憶する．

受信割込み処理はその後，RTOS のカウンタおよびアラーム機能を利用して算出タスクを起動するための処理を行う．具体的には，論理時間を刻むユーザ定義カウンタを更新するシステムコール SignalCounter() を呼び出す．ただし，受信したメッセージのタイムスタンプの値が式 2 を満たす場合のみ SignalCounter() を呼び出し，満たさない場合は呼び出さない．なお，カウンタとアラームを用いたタスク起動処理の詳細については 3.3 節で説明する．

タイムスタンプ = 次回論理起動時刻 - 論理遅延時間　（式 2）

RTOS によって算出タスクが起動されると，ミドルウェアのタスク起動時処理が入力タスクの場合と同様の処理を行った後，算出タスクの処理を実行する．

算出タスクがメッセージを読み出すために ReceiveMessage() を呼び出すと，ミドルウェアのメッセージ受信処理は式 3 を満たすタイムスタンプ（この例では 0）を持つメッセージを探索して，そのデータを引数で指定された領域にコピーする．

タイムスタンプ = 論理起動時刻 - 論理遅延時間　（式 3）

そして，算出タスクがメッセージを送信するために SendMessage() を呼び出すと，ミドルウェアのメッセージ送信処理は入力タスクの場合と同様の処理を行い，メッセージを送信する．

最後に，出力タスクの場合のミドルウェアの動作を**図 6** を用いて説明する．メッセージを受信すると ISR が起動され，算出タスクの場合と同様の処理を実行する．ただし，タスク起動のための処理は行わない．

出力タスクは RTOS によって周期タスクとして起動される．ミドルウェアのタスク起動時処理は入力タスクの場合と同様である．出力タスクがメッセージを読み出すために ReceiveMessage() を呼び出すと，ミドルウェアのメッセー

Vol.2018-ARC-230 No.23
Vol.2018-SLDM-183 No.23
Vol.2018-EMB-47 No.23
2018/3/8

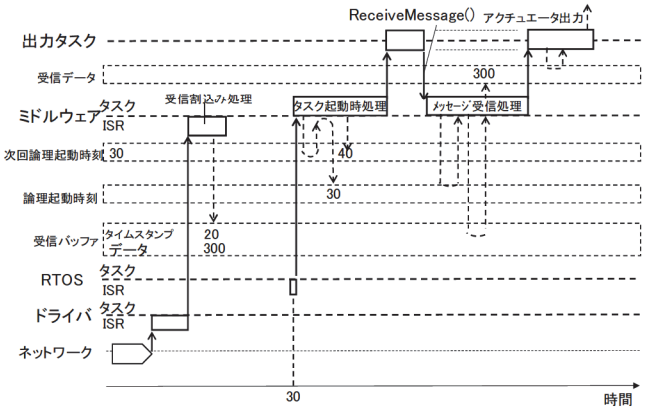


**図 6** 出力タスクの動作

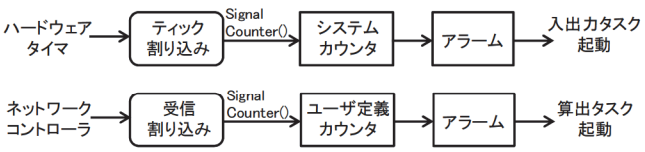


**図 7** カウンタとアラームによるタスク起動

ジ受信処理は，算出タスクの場合と同様の処理を行い，受信メッセージのデータを，引数で指定された領域にコピーする．

### 3.3 RTOS の機能を用いたタスク管理

OSEK OS で周期タスクを実現するには，システムカウンタとアラーム機能を用いる．システムカウンタはシステム時刻を刻むもので，**図 7** 上側に示すように，ハードウェアタイマによりティック毎に起動される割込み（ティック割込み）によって更新される．OSEK OS 仕様ではカウンタを更新するためのシステムコールを規定していないが，TOPPERS/ATK1 ではシステムコール SignalCounter() を呼び出すことで，カウンタを更新する．そしてシステムカウンタに接続したアラームによりタスクを起動する．これにより，タスクを周期的に起動できる．この機能を用いて，本分散処理環境における入出力タスクを起動する．

OSEK OS はシステムカウンタとは別に，ユーザ定義のカウンタも使用できる．そこで我々は，図 7 下側に示すように，ユーザ定義のカウンタを用いて論理時間を刻むこととし，そのユーザ定義カウンタに接続したアラームにより算出タスクを起動する．ユーザ定義カウンタは，ネットワークコントローラにより起動される受信割込み（ミドルウェアの受信割込み処理）が SignalCounter() を呼び出して更新する．1 回で更新するカウント数はメッセージの通信周期に対応する値とし，カウンタの OIL 記述で指定する．

ユーザ定義カウンタとアラームにより，通信周期がタスク周期と同じ場合は，受信する毎にタスクが起動される．通信周期がタスク周期の n 倍の場合は，n 回受信する毎にタスクが起動される．ただし，3.2 節で述べたように，メッセージの到着順序が乱れ，受信したメッセージのタイムスタンプを用いて式 1 により求めた論理起動時刻が次回論理起動時刻よりも新しい場合は，ユーザ定義カウンタの更新は行わない．そして，一致するメッセージを受信した時点で，それまで受信した回数分の更新を行う．この場合は，複数回続けてタスクが起動されることになる．

以上のように OSEK OS のカウンタとアラームを用いることで，物理時間駆動処理と論理時間駆動処理の両者を統一的な枠組みで実現できる．これにより，タスク起動用のアラームの OIL 記述は，物理時間駆動処理か論理時間駆動処理かによらず同じ記述になり，システムカウンタに接続するかユーザ定義カウンタに接続するかで，物理時間駆動処理と論理時間駆動処理を選択できる．

なお，現在は OSEK OS のスケジューリング機能をそのまま用いているため，全てのタスクを固定優先度でスケジューリングしている．今後は，論理時間駆動処理を各ジョブの論理デッドライン（論理起動時刻 + タスクの周期）を用いて EDF （Earliest Deadline First）[15] スケジューリングによってスケジュールすることや，時間駆動処理の対象タスク（入出力タスク）をハードリアルタイムタスク，論理時間駆動処理の対象タスク（算出タスク）をソフトリアルタイムタスクと見なし，TBS（Total Bandwidth Server）[16] 等のスケジューリングアルゴリズムを適用することを検討している．

## 4. 実装および評価

本研究では，マイクロコントローラ H8S/2638F を搭載した評価ボードを用いて実装を行った．H8S/2638F は 256kB の ROM，16kB の RAM を内蔵し，クロック周波数は 20MHz である．リアルタイム OS には TOPPERS/ATK1 Release1.0 を用いた．ネットワークに CAN を使用する場合は，H8S/2638F に内蔵されている CAN コントローラを使用し，その伝送速度は 500kbps とする．ZigBee を使用する場合は，ZigBee プロトコルをサポートした XBEE 無線モジュールと H8S/2638F を UART 9600bps のシリアル通信で結び，H8S/2638F に内蔵されているデータトランスファーコントローラ（DTC）によりバッファに転送したメッセージを，10ms 周期のポーリングで受信する．したがって ZigBee の場合は，3.2 節で述べた受信割込み処理を，割込み処理（ISR）ではなく，10ms 周期の最高優先度タスクで実行している．

分散処理ミドルウェアが実用上問題ない性能で実装できているかを評価するため，ミドルウェアの各処理の実行時間を測定した．タスク起動時処理は，入力タスクと算出タスクでは同じ動作をするが，出力タスクの場合は異なるので別に測定した．また出力タスクの場合はメッセージ数が増えると処理時間が増大するので，メッセージ数が 1 の場合とメッセージ数が 32 で最悪のケースについて測定した．

Vol.2018-ARC-230 No.23
Vol.2018-SLDM-183 No.23
Vol.2018-EMB-47 No.23
2018/3/8

**表 1** ミドルウェアの各処理の実行時間（単位:$\mu$sec）

| 測定対象 | | | 実行時間 |
|---|---|---|---|
| タスク起動時処理 | 入力タスク・算出タスク | | 11.6 |
| | 出力タスク | 1 メッセージ | 15.2 |
| | | 32 メッセージ | 31.7 |
| メッセージ送信処理 | | | 14.6 |
| メッセージ受信処理 | | 探索なし | 24.9 |
| | | 探索あり | 30.0 |
| 受信割込み処理 | 算出タスク | 正常時 | 131.6 |
| | | 逆転時 | 184.8 |
| | 出力タスク | 正常時 | 70.7 |
| | | 逆転時 | 80.5 |

メッセージ受信処理はメッセージの到着順が乱れると探索が発生するので，1 回目の参照で見つかる場合（探索なし）と 2 回目の参照で見つかる場合（探索あり）について測定した．また受信割込み処理は，算出タスクのための処理と出力タスクのための処理で異なるとともに，メッセージの到着順が正常な場合と逆転が生じた時で異なるので，それぞれ別に測定した．測定にはクロック周波数 20MHz のハードウェアタイマを用いた．従って測定の精度は 0.05$\mu$sec である．

各処理の実行時間の測定結果を**表 1** に示す．ドライバの処理時間は含んでいない．表に示した値は 100 回計測しての平均値である．最悪値は最大の場合でも 0.5$\mu$sec 程度の増加である．

評価に用いた環境の場合，周期タスク起動を行う場合のティック割込み処理の実行時間が 47$\mu$sec 程度，そのうちタスク起動にかかる時間は 33$\mu$sec 程度である．したがって，タスク起動時処理の実行時間は許容範囲と考える．また，タイムスタンプを用いない場合でもメッセージ送信に 10$\mu$sec 程度，メッセージ受信に 20$\mu$sec 程度かかるので，メッセージ送信処理およびメッセージ受信処理も許容範囲と考える．受信割込み処理については，算出タスクのための実行時間が大きいが，タイムスタンプを付加しないメッセージを受信バッファへに格納するにも 50$\mu$sec 程度かかることや，前述したティック割込み処理の実行時間を考えると妥当な値である．なお，逆転時の実行時間が大きいのはタスクを 2 回起動するためである．

自動車制御システムにおいて，ネットワーク通信を行うタスクの周期は 10〜100msec 程度なので，応答時間の観点からは，ミドルウェアの実行時間は実用上問題ない値と考える．また，今回使用した CPU の性能は高くなく，最近使用されている 1 桁高性能な CPU の場合には実行時間は大きく短縮される．したがって，開発した分散処理ミドルウェアは有用性があるものと考える．ただし，現在の実装は特に最適化等は行っておらず，性能改善の余地があるものと考えている．

## 5. おわりに

通信時間の変動するネットワークを用いた環境でも時間駆動アーキテクチャと同程度のリアルタイム性を実現することを目的に，タイムスタンプ付きメッセージと論理時間を導入した時間駆動分散処理環境を提案し，それを実現するための分散処理ミドルウェアを開発した．

今後は，本分散処理ミドルウェア向けコンフィギュレーションツールの開発，IEEE802.11p 等の無線ネットワークへの対応，機能拡張，性能改善等を行う予定である．

**謝辞**

本研究は JSPS 科研費 JP15K00084 の助成を受けたものである．

### 参考文献

[1] Lee, E. A.: Cyber Physical Systems: Design Challenges, *Proc. 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing,* pp.363–369 (2008).
[2] Cervin, A., Henriksson, D., Lincoln, B., Eker J. and Arzen, K.: How Does Control Timing Affect Performance? Analysis and Simulation of Timing Using Jitterbug and TrueTime, *IEEE Control Systems,* Vol.23, No.3, pp.16–30 (2003).
[3] Kopetz, H.: Should Responsive Systems be Event-Triggered or Time-Triggered?, *IEICE Transaction on Information & Systems*, Vol.E76-D, No.11, pp.1325–1332 (1993).
[4] Kopetz, H. and Grunsteidl, G.: TTP-A Protocol for Fault-Tolerant Real-Time Systems, *IEEE Computer,* Vol.27, No.1, pp.14–23 (1994).
[5] Makowitz, R. and Temple, C.: FlexRay - a Communication Network for Automotive Control Systems, *Proc. 2006 IEEE International Workshop on Factory Communication Systems,* pp.207–212 (2006).
[6] OSEK/VDX: *Time-Triggered Operating System, Version 1.0* (2001).
[7] 知場貴洋，齊藤政典，伊丹悠一，兪明連，横山孝典：位置透過性のあるシステムコールを有する組み込み制御システム向け分散リアルタイム OS，情報処理学会論文誌，Vol.53，No.12，pp.2702–2714 (2012).
[8] OSEK/VDX: *Operating System, Version 2.2.3* (2005).
[9] Kiencke, U.: Controller Area Network - from Concept to Reality, *Proc. 1st International CAN Conference,* pp.0-11–0-20 (1994).
[10] ZigBee Aliance, http://www.zigbee.org/
[11] 松原彩音，兪明連，横山孝典：GNSS を用いた時刻同期機能を有するリアルタイム OS，情報処理学会論文誌，Vol.57，No.8，pp.1765–1774 (2016).
[12] TOPPERS Project, http://www.toppers.jp/
[13] OSEK VDX, *System Generation OIL: OSEK Implementation Language Version 2.5* (2004).
[14] OSEK/VDX: *Communication, Version 3.0.3* (2004).
[15] Liu C. L. and Layland J. W.: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, *Journal of the ACM,* Vol.20, No.1, pp.46–61 (1973).
[16] Spuri, M. and Buttazzo, G. C.: Efficient Aperiodic Service under Earliest Deadline Scheduling, *Proc. 15th IEEE Real-Time Systems Symposium,* pp.2–11 (1994).