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

2004－SLDM－117（1）
2004／12／1

実時間制約検証に特化したCANバスモデルの提案とシミュレータの試作

山口　聖二†　伊地知孝仁†　谷本　匡亮†　中田　明夫†　東野　輝夫†

† 大阪大学 大学院情報科学研究科
〒 565-0871 大阪府吹田市山田丘 1-5
E-mail: †{s-yamagt,ijichi.takahito,tanimoto,nakata,higashino}@ist.osaka-u.ac.jp

あらまし　本稿では，主に車内通信などに利用されているシリアルバスプロトコルである CAN（Control Area Network）プロトコルについて，実時間制約検証を効率よく行うことができるよう仕様を抽象化したモデルを提案する．提案モデルでは，メッセージ送信時のバス使用権の取得や通信におけるエラー発生時の処理，クロックサイクル消費の扱いなどの点で抽象化を行う．その結果，実際のシステムでバス上を流れる波形を考慮する必要がなくなるため，検証の高速化を実現できる．また，静的解析と異なり，具体的なバス通信動作および通信エラー処理をシミュレート可能である．提案する抽象化モデルに基づいて，ユニット毎の動作記述，バスクロック単位での動作確認が可能なシミュレータを試作する．試作したシミュレータを文献 [5] の例題に適用し，シミュレーション結果を比較することにより，提案モデルの有効性を示す．

キーワード　CAN，抽象化，リアルタイムシステム，バスアーキテクチャ，実時間制約，シミュレーション

Proposal of a CAN Bus Model Aimed at Real-Time Constraint Verification and Implementation of the Simulator

Seiji YAMAGUCHI†, Takahito IJICHI†, Tadaaki TANIMOTO†,
Akio NAKATA†, and Teruo HIGASHINO†

† Graduate School of Information Science and Technology, Osaka University
1-5, Yamadaoka, Suita, Osaka 565-0871, Japan
E-mail: †{s-yamagt,ijichi.takahito,tanimoto,nakata,higashino}@ist.osaka-u.ac.jp

Abstract　In this paper, we propose a model for CAN (Control Area Network) protocol, which is a serial bus protocol mainly used for the communications in car system. Our model is an abstraction of the specification of CAN protocol, aimed at improving performance of dynamic verification (simulation) of real-time constraints for bus communications. In our proposed model, we abstract bus arbitration, error handling, and clock cycle handling. By the proposed abstraction, we need not perform a costly waveform simulation. Moreover, unlike static analyses, we can also check correctness of the system behavior in the presence of bus communication errors by simulation. We have implemented a simulator based on our proposed abstracted model. In the simulator, we can specify a concrete I/O behavior for each unit of a bus system, enabling us to check whether real-time constraints are satisfied in the specified behavior setting, as well as influence of bus communication errors. We present some experimental results to show that our abstraction is appropriate and effective for the real-time constraint verification.

Key words　CAN, abstraction, real-time system, bus architecture, real-time constraint, simulation

1. はじめに

バスシステムは，一般で利用されている機械や自動車通信システムなどの幅広い分野で使われている．例えば，自動車産業界では安全性，快適性，低公害，低コストを求め，様々な電子制御システムが開発されてきた．その中で，低コスト化，軽量化の実現を目標としてのワイヤーハーネスの削減や，人命に関わる車載機器の通信のための高信頼性（ノイズ対策や対故障性）の確保，複数の LAN を介した大容量データの高速通信の実現などのニーズに応えるために，シリアル通信プロトコルである

CAN（Control Area Network）が開発された．CAN プロトコルは主に車内通信で使われてきたが，現在では高性能，高信頼性から，自動車だけでなく医療機器や産業用フィールドバスなど幅広く利用されている．

このようにバスシステムは様々な分野で利用されているが，一つの回路に多様な機能を実装することで，ユニット間の通信が複雑化している．それにより，機能が少ない場合と比べ設計中にエラーを含んでしまう危険性が増大するため，設計が容易ではなくなる．さらに，ユニット間の通信に実時間制約を持つ場合，ユニット間の通信にかかる時間も考慮する必要が出てくるため，設計はさらに複雑なものとなる．そのような背景から，システムを計算機で設計する様々な研究が行われている．[1] [2]

CAN バスシステムの動作を正確に検証するには，バスに流れる波形と各ユニットの動作が正しく連動していること，各ユニットが正しく動作していることなどを検証する必要がある．しかし，CAN バスシステムの動作における実時間制約の検証に限定すれば，詳細なバスの波形のシミュレーションを行うことは必ずしも必要でない．文献 [5] では，波形シミュレーションを行わず，ワーストケース解析に基づく静的手法により実時間制約検証を行う手法が提案されている．しかし，文献 [5] の手法は静的解析であるため，CAN プロトコルのエラー処理を扱うことができない．

従って，本研究では，実時間制約を持つ CAN プロトコルでの通信に対して，ユニット間の通信が実時間制約を満たしているかを効率よく検証できるよう，CAN プロトコルの仕様をエラー処理も含め抽象化した CAN バスモデルを提案する．

本稿の構成は以下の通りである．第 3 節では，提案手法において CAN プロトコルの抽象化を行う箇所について述べる．第 4 節では，第 3 節で述べた提案モデルを用いた実時間制約検証を行うためのシミュレータの設計法，動作内容について述べる．第 5 節では，第 4 節で述べたシミュレータを用いて，具体的な CAN バスシステムの例による CAN バスモデルを動作させることにより，実時間制約検証を行うことが可能であることを示す．最後に第 6 節で，本稿のまとめと今後の課題について述べる．

2. CAN プロトコルの概要と特徴

この節では，モデル化の対象となる CAN プロトコルについて，概要と機能上の特徴について述べる．なお，本節では，後に行う抽象化で対象となる箇所を主に述べており，それ以外の部分については文献 [4] を参照のこととする．

2.1 CAN プロトコルの概要

CAN プロトコルは国際的に標準化されたシリアル通信プロトコルであり，欧州では自動車 LAN の標準プロトコルとして採用されている．CAN バスシステムを構成するバスは 2 本のワイヤで構成され，それらの電位差によりバスのレベルが決定される．2 本のワイヤ間の電位差が一定の値より大きければドミナントレベル，それ以下であればレセシブレベルとなる．バスに接続されているユニットのいずれかからドミナントレベルの出力が行われると，バスのレベルはドミナントレベルとなる．

2.2 CAN プロトコルの特徴

CAN プロトコルの主な特徴として，以下の点が挙げられる．

マルチマスタ，マルチキャスト：バスが空いている状態では，全てのユニットがメッセージを送信可能である．また，バスに接続している全てのユニットがバスに送信されたメッセージを同時に受け取ることが可能である．

バスの使用権：CAN はシリアルバスプロトコルであり，同時に 1 つのメッセージしかバス上に流すことができない．よって，バスが空いている状態から同時に複数のユニットがメッセージを送信した場合，調停を行い，どのメッセージをバス上に流すかを決定する必要がある．調停は，バスに流れるメッセージが持つ ID を上位の桁から比較し，最も値が小さい ID を持つメッセージにバスの使用権が与えられる．バスの使用権が与えられたメッセージはそのままバスを使用してメッセージを送信することができ，使用権を与えられなかったメッセージを送信していたユニットは，そのことが判明次第，メッセージの送信を中断し，受信動作に移らなければならない．

システムの柔軟性：バス上を流れるメッセージの送信先や目的などの情報は，メッセージの持つ ID により判別されるため，バスに接続するユニットにはアドレスが割り振られていない．そのため，ユニットの追加，変更に対する柔軟性を持つ．同時に，バスの遅延時間と電気的付加による制限は加わるものの，接続可能なユニット数に理論的な制限は存在しない．

エラーの検出，通知，リカバリ機能：全てのユニットが受信メッセージに含まれるエラーを検出することができ，検出した場合，即座に他の全てのユニットにエラーを通知する．メッセージ送信ユニットがエラーを検出した場合，メッセージの送信を強制的に中断し，エラー処理後，同じメッセージの再送信を行う．この再送信は，メッセージが正常に送信完了するまで繰り返される．

2.3 送受信されるメッセージの種類

バス上を流れるメッセージはフレームという単位で送信される．フレームはいくつかの種類に分類することができ，それぞれの役割が表 1 のように決まっている．

この中で，通常のデータ転送で利用されるフレームはデータフレームである．データフレームの構成を図 1 に示す．

表 1　データフレームの種類と役割

フレームの名称	フレームの役割	フレームの長さ（単位はbit 括弧内は拡張フォーマット）[4]
データフレーム	送信ユニットが受信ユニットにデータを送信するためのフレーム	44～108 (64～128)
リモートフレーム	受信ユニットが同じIDを持つ送信ユニットへデータを要求するためのフレーム	44 (64)
エラーフレーム	エラーを検出したときに，他のユニットへエラーを通知するためのフレーム	14～20
オーバーロードフレーム	受信ユニットが受信準備未完了であることを通知するためのフレーム	14～
インタースペースフレーム	データフレーム，リモートフレームを前のフレームと分離するためのフレーム	3～11

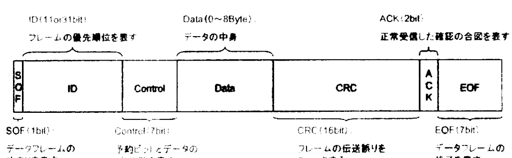

図 1　データフレームの構成

また，各フレームの内容について，以下のような特徴がある．

- データフレーム

データフレーム内には前述の ID があり，メッセージの優先順位の決定に利用される．

- エラーフレーム

エラーフレームは 6 ビットのエラーフラグと 0〜6 ビットのエラーフラグの重ね合わせ，8 ビットのエラーデリミタから構成される．エラーフラグは後述するフレーム送信ユニットのエラー状態により内容が変化し，エラーアクティブ状態であれば，6 ビットの連続したドミナントレベル（アクティブエラーフラグ），エラーパッシブ状態であれば，6 ビットの連続したレセシブレベル（パッシブエラーフラグ）となる．

- オーバーロードフレーム

オーバーロードフレームは 6 ビットのオーバーロードフラグ，オーバーロードフラグの重ね合わせ，8 ビットのオーバーロードデリミタから構成される．オーバーロードフラグは 6 ビットの連続したドミナントレベルである．

2.4 エラー状態の扱い

バスに接続している全てのユニットは，表 2 に示す 3 つの状態のうち，いずれか 1 つの状態にある．

表 2 ユニットのエラー状態

エラー状態の名称	エラー状態の特徴
エラーアクティブ状態	バスの通信に正常に参加できる状態である．エラーを検出した場合，アクティブエラーフラグを送出し，他のデータフレームを強制的に終了させる．
エラーパッシブ状態	バスの通信に正常に参加できるが，エラーを起こしやすい状態である．エラーを検出した場合，パッシブエラーフラグを送出する．パッシブエラーフラグは他のデータフレームを強制的に終了することはできない．
バスオフ状態	バスの通信に参加できない状態である．メッセージの送受信について，全ての動作が禁止される．

状態間の遷移はメッセージの送信の成功，失敗により値の変化するエラーカウンタの値で決定される．エラーカウンタは送信エラーカウンタと受信エラーカウンタという 2 種類が存在する．エラーカウンタの値の変化，およびエラーカウンタの値による状態の遷移を表 3，図 2 に示す．

表 3 エラーカウンタの値の変化

	エラーの種類，エラーカウンタ変動条件	送信エラーカウンタ	受信エラーカウンタ
1	受信ユニットがエラーを検出したとき（ただしエラーフラグ，オーバーロードフラグ送信中にビットエラーを検出したときは無効）	-	+1
2	受信ユニットがエラーフラグを送信した後のビットにドミナントビットを検出したとき	-	+8
3	送信ユニットがエラーフラグを出力したとき	+8	-
4	送信ユニットがアクティブエラーフラグまたはオーバーロードフラグを送信中にビットエラーを検出したとき	+8	-
5	受信ユニットがアクティブエラーフラグまたはオーバーロードフラグを送信中にビットエラーを検出したとき	-	+8
6	各ユニットがアクティブエラーフラグ，オーバーロードフラグの最初から14ビットの連続したドミナントを検出したとき．その後，8ビット連続のドミナントを検出するたび	送信したとき +8	受信したとき +8
7	パッシブエラーフラグの後に追加の8ビット連続のドミナントを検出したとき	送信したとき +8	受信したとき +8
8	送信ユニットがデータを正常に受信したとき（ACKが返りEOFの完了までエラーを検出しなかったとき）	-1（送信エラーカウンタ>0） ±0（送信エラーカウンタ=0）	-
9	受信ユニットがデータを正常に受信したとき（CRCフィールドまでエラーを検出せず，ACKを正常に返すことができたとき）	-	-1（1≦受信エラーカウンタ≦127） ±0（受信エラーカウンタ=0） +27（受信エラーカウンタ>127）
10	バスオフユニットが連続した11ビットのレセシブビットを128回検出したとき	0	0

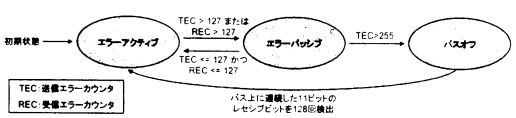

図 2 エラー状態の状態遷移図

2.5 通信の同期

CAN では，各ビットの始まりや終わりに同期を取るための信号を付加しない．同期はメッセージ受信開始時（ハードウェア同期）や，データ通信に発生するバスのレベルの変化を利用した手法（再同期）を採用している．

3. 提案する CAN バスモデル

この節では，前節で述べた特徴を抽象化することによる，実時間制約検証に適したモデルの設計手法について述べる．

本研究では以下の点で抽象化を行った．

- 正常転送時の転送サイクル数を保存した波形の抽象化
- バス権調停制御の抽象化
- エラー制御の抽象化
- エラー発生時における波形の抽象化

3.1 正常転送時の転送サイクル数を保存した波形の抽象化

CAN プロトコルはシリアルバスプロトコルであり，1 ビットのデータ転送にかかる時間はバスにおける 1 クロック分となる．よって，メッセージ送信にかかる時間は送信メッセージ全体のビット長と等しいクロック数の経過にかかる時間となる．

提案モデルは CAN バスシステムの実時間制約検証を効率よく行うことを目的としており，バス上を流れる波形には着目していない．よって，ユニットのメッセージ送信を以下の手順により，シリアル転送と同様になるよう抽象化する．

まず，メッセージ送信前に，送信予定であるメッセージ全体のビット長を計算する．これは，予定しているメッセージの送信にかかるクロック数と同じ値となる．次に，メッセージを送信する際，そのクロック数の間，他のユニットがメッセージを送信できないよう，バスのロックを開始する．この段階で，送信メッセージの持つ ID に関係のあるユニットへ一度にメッセージの内容を転送する．そして，バスのロック開始からメッセージ送信にかかるクロック数の経過後，バスのロックを解除することで，シリアル転送を用いたメッセージ送信を擬似的に再現する．メッセージをシリアル転送した場合と提案手法による場合のバスの扱いを図 3 に示す．

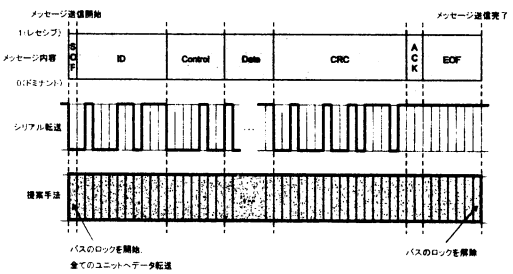

図 3 正常転送時におけるメッセージ送信の抽象化

3.2 バス権調停制御の抽象化

バスが使用されていない状態から，複数のユニットが同時にメッセージを送信しようとした場合，メッセージに含まれる ID の値により，どのメッセージにバス使用権が与えられ，バス上に流されるかが決定する．この処理は本来，バスに接続する全てのユニットの内，1 つでもドミナントレベルのデータを送信

すると，バスに流れるデータはドミナントレベルになるという性質を用いて，送信したデータと受信したデータが一致する限り送信を続けるという手法を採用している．つまり，メッセージをバスに流す前に送信予定のメッセージが持つ ID の値が分かれば，バス使用権を取得するメッセージとそれを送信するユニットをメッセージ送信前に判別することができる．メッセージが持つ ID を用いた調停の結果，バス使用権を得られなかったユニットはメッセージを送信せずにそのまま受信動作に入る．以上により，バス上に流れるメッセージを 1 つに決定し，送信動作の中断や実時間での調停などの処理を無くすことができる．

3.3 エラー制御の抽象化

メッセージ送信における各ユニットのエラー状態による影響は，メッセージ送信可能となるタイミングが異なる点にある．エラーパッシブ状態にあるユニットはインターフレームスペースの増加により，エラーアクティブ状態にあるユニットよりも先んじてメッセージの送信ができない．また，バスオフ状態にあるユニットはメッセージの送受信を行うことができない．

提案モデルでは CAN における通信の実時間制約検証を目的としており，エラー状態を正確に追跡する必要はない．また，頻繁にエラーが発生する状態にある回路では実時間制約が満たされない可能性が高いといえる．よって，エラー状態遷移の簡略化，及び各エラー状態におけるメッセージ送信の優先度の指定を導入し，検証を効率良く行えるように抽象化を行う．

まず，エラー状態を決定する送信エラーカウンタと受信エラーカウンタをまとめて 1 つのエラーカウンタとする．エラーカウンタの値の変化はメッセージの送信が正常に完了したか，エラーが発生したかという基準で行うものとする（表 4）．

表 4　エラーカウンタ変化の抽象化

	エラーの種類，エラーカウンタの変動条件	エラーカウンタ
0	メッセージの送受信が正常に行われた	-1(エラーカウンタ>0) ±0(エラーカウンタ=0)
1〜9	エラーを検出，または出力したとき	+8

また，エラーカウンタの値を用いてエラー状態の決定と遷移を行う．エラーカウンタの値によるエラー状態の遷移を図 4 に示す．エラー状態の遷移において，本来，バスオフ状態からエラーアクティブ状態への遷移が存在するが，それも無いものとして抽象化を行う．

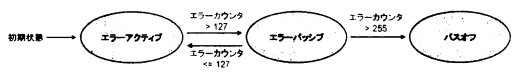

図 4　エラー状態遷移の抽象化

また，メッセージ送信におけるエラー状態と優先度について述べる．通常の CAN プロトコルではメッセージを送信する前に，以前に受信したメッセージに対するインターフレームスペースを送信し終えている必要がある．インターフレームスペースの大きさは，エラーアクティブ状態で 3，エラーパッシブ状態で 11 となっている．メッセージを送信しようとした際，エラーアクティブ状態のユニットがインターフレームスペースの送信を終えて，エラーパッシブ状態のユニットよりも早く送信準備が完了し，メッセージを送信することができる．これによりエラーアクティブ状態にあるユニットはエラーパッシブ状態にあるユニットよりもメッセージ送信の優先度が高いといえる．

この部分における処理を簡略化するために，メッセージの持つ ID を上回る優先度をユニット毎にエラー状態により決定する手法を導入し，複数のユニットが同時にメッセージ送信を行おうとした場合，エラーアクティブ状態にユニットを優先するということにする．また，メッセージ送信後のインターフレームスペースをエラー状態によらず一律の 3 という値に決定する．

エラーパッシブ状態にあるユニットが，インターフレームスペースが 3 という短い間隔でメッセージを送信できるという誤差が生じるが，メッセージのデッドラインと比較するとごく僅かなクロック数であるため，無視できるものとする．また，エラーアクティブ状態のユニットが ID による優先度の低いメッセージを，エラーパッシブ状態のユニットが ID による優先度の高いメッセージを同時に送信しようとした場合，ID による優先度の低いメッセージ（すなわちエラーアクティブ状態にあるユニットからのメッセージ）が優先されてしまう．本来，ID による優先度の高いメッセージ（すなわちエラーパッシブ状態にあるユニットからのメッセージ）を優先すべきであるが，実際にはエラーパッシブ状態ではインターフレームスペース直後の場合，エラーアクティブ状態のユニットに先に送信されてしまうため，問題でない．また，インターフレームスペースとの間隔が開いた送信では，実際の動作とは誤差が生じるが，ここではエラーパッシブ状態になることがそれほど多くないという観点から無視するものとする．

3.4 エラー発生時の波形の抽象化

CAN プロトコルの特徴としてエラーの検出，通知，リカバリの各機能があることは既に述べたが，それらも波形を考慮しない形に抽象化する．

エラーの検出は送信ユニットが送信したメッセージを各ユニットが受信し，エラーが含まれていることを調べることで検出される．しかし，抽象化により波形を検知することができないため，メッセージ送信中にエラーの発見を行うことは不可能である．よって，メッセージ送信時にエラーの発生を，乱数や経過クロック数，ユニットの状態などから予め指定することにより，エラー発生時のシステム全体の振る舞いを再現できるようにする．エラーが発生するとした場合，通常と同様にメッセージの送信を行うが，メッセージ長をエラーが検出されるまでに送受信される長さに変更し，エラー発生によるメッセージ送信の中断を実現する．

エラー検出後，それを他のユニットに通知するために，エラーを検出したユニットからエラーフレームがバスに流される．このエラーフレームの長さもバスに流れるデータや各ユニットの受信メッセージにより変化するため，波形を考えない抽象化では正確な再現はできない．よって，エラーフレームの長さを，実際のシステムで送信される可能性のある長さの範囲内で，ユニットの状態や乱数，関数などから決定し，エラーフレームの長さだけバスクロックを空転させる．

また，エラー発生後のリカバリについては，通常の CAN プロトコルと同様に，エラーがあったメッセージを送信したユ

ニットがメッセージを再送信することにより実現可能である．

4. シミュレータの試作

本節では，前節で述べた抽象化を適用し作成されたモデルについて，そのモデルが実時間制約を満たすかどうかの確認を行うために試作したシミュレータついて述べる．なお，シミュレーションで対象としたシステムは文献[5]と同じ電気自動車の車内通信バスシステムである．

対象システムには7個のユニットが存在し，ユニット間でメッセージを送信しあうことでシステム全体の制御を行っている．図5に対象システムのユニット構成を示し，表5にバスに送信されるメッセージセットの一部を示す．表5にある Signal Number はメッセージの ID とは異なり，メッセージの種類を表す通し番号である．メッセージの ID は，メッセージの送信周期やデッドラインから別途決定される数値である．

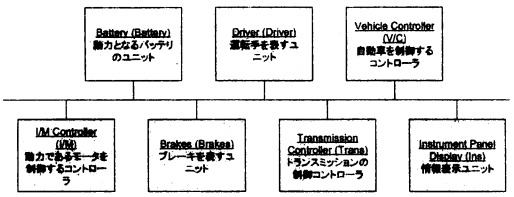

図5　対象システムのユニット構成

表5　対象システムで送信されるメッセージ例

Signal Number	Signal Description	Size [bits]	Jitter [ms]	Time Cycle [ms]	Periodic / Sporadic	Deadline [ms]	From	To
1	Traction Battery Voltage	8	0.6	100.0	Periodic	100.0	Battery	V/C
2	Traction Battery Current	8	0.7	100.0	Periodic	100.0	Battery	V/C
3	Traction Battery Temp. Average	8	1.0	1000.0	Periodic	1000.0	Battery	V/C
	...							
21	Motor/Trans Over Temperature	2	0.3	1000.0	Periodic	1000.0	Trans	V/C
22	Speed Control	3	0.7	50.0	Sporadic	20.0	Driver	V/C
23	12V Power Ack Vehicle Control	1	0.2	50.0	Sporadic	20.0	Battery	V/C
	...							
51	Shutdown	1	0.7	50.0	Sporadic	20.0	I/M C	V/C
52	Status/Malfunction (TBD)	8	0.8	50.0	Sporadic	20.0	I/M C	V/C
53	Main Contactor Acknowledge	1	1.5	50.0	Sporadic	20.0	V/C	I/M C

シミュレータの設計手法は文献[6]にある手法を採用する．

文献[6]にある手法では Java 言語によるスレッドプログラミングを用いて，システム全体を構築する．システム全体は，処理要素を表すクラス群，バス調停回路を表すクラス群，バスクロックを管理するクラスの3つに大きく分けられる．シミュレータでは，処理要素はバスに接続している各ユニットに対応しており，ユニットの個数分だけ用意される．また，バス調停回路は，CAN バスモデルの対象システムにおいて，全てのユニットが1つのバスに接続しているため，そのバスについての調停を表す1つのクラスが用意される．そして，バスクロックを管理するクラスは全ての処理要素やバス調停回路が1クロック分の動作を終了するまで待機し，全てのクラスの動作の終了を確認すると，バスの状態や処理要素に含まれるレジスタの状態を更新し，全てのクラスに次のクロックの動作を行うように指示を出す．このようにシミュレーションの過程で，1クロックごとに同期を取る手法をバリア同期（文献[3]）という．

各クラスが Thread クラスを継承しており，スレッドとして並行動作する．図6にシミュレータの構成を示す．

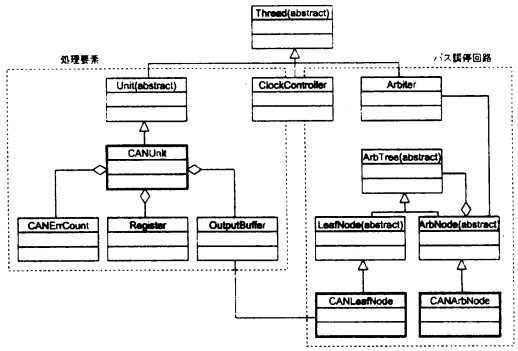

図6　シミュレータの構成

4.1 処理要素を構成するクラス

処理要素は表6に示されるクラスで構成される．

表6　処理要素を構成するクラス

処理要素を構成するクラスの名称	各クラスの役割
CANUnitクラス	処理要素全体を表すクラス．バスに接続するユニットに対応し，以下のクラスを含む
CANErrCountクラス	処理要素のエラー状態を管理するクラス．エラーカウンタを保持しており，メッセージ送信の成功，失敗に応じてエラーカウンタ値の変更，エラー状態の更新を行う
Registerクラス	処理要素におけるレジスタを表すクラス．処理要素内で内部データや受信データを保持する
OutputBufferクラス	処理要素における出力バッファを表すクラス．処理要素からバスに送信するフレームを一時的に保持する

シミュレータでは CANUnit クラスがバスに接続するユニットを表しており，CANErrCount，Register，OutputBuffer の各クラスのオブジェクトを含んでいる．各クラスにより，処理要素のエラー状態，バスとのインターフェースを実現している．

4.1.1 メッセージ送信処理

処理要素からバスへメッセージを送信する場合は，処理要素に含まれる OutputBuffer をインターフェースとして用いる．メッセージの作成は処理要素の動作として，含まれるデータや ID，デッドラインを含めて記述される．処理要素でメッセージ ID 毎に保存されている送信タイミングを表すクロック数が，シミュレーション開始からの経過クロック数を超えた時に，作成したメッセージを OutputBuffer に書き込む．OutputBuffer に書き込まれたメッセージは，後述のバス調停回路による調停の結果，送信可能となるまで保持され，送信できる状態になると，OutputBuffer からバスに流される．メッセージの送信が完了すると，CANErrCount で保存されているエラーカウンタ，エラー状態を更新して，送信処理を完了とする．

4.1.2 メッセージ受信処理

バスに流されたメッセージを受信する際は，まず，メッセージに含まれる ID を調べることで，そのメッセージが処理要素に関係があるかどうかを調べる．関係のある処理要素では，メッセージの受信処理を行う．バスからメッセージを受信する場合は，バスから Regiter クラス内の配列に直接書き込まれる．Register クラス内の配列は，メッセージ ID 毎に，メッセージに含まれるデータを保存できるだけの容量が確保されている．

4.2 バス調停回路を構成するクラス

バス調停回路は表7に示されるクラスで構成される．

バス調停回路では，各処理要素の出力バッファに入っているメッセージの中から，バスに流すメッセージを固定優先度による調停で決定する．Arbiter クラスのオブジェクトでは，各処理

要素が持つ優先度（エラー状態により決まる優先度）と，メッセージが持つ ID を考慮して，調停を行う．

4.3 クロックを管理するクラス

クロックを管理するクラス（ClockController クラス）では，各処理要素とバス調停回路の動作を 1 クロックずつ区切り，全ての構成要素の動作が完了したことを確認してから，データや状態の更新をし，次のクロックの動作をするよう指示を出す．

5. シミュレータを用いた提案モデルの実時間制約検証

本節では，前節でモデル化の対象としたシステムのモデルを前節で述べたシミュレータを用いて動作させ，文献 [5] におけるシミュレーション結果と比較することにより提案モデルの有効性を示す．文献 [5] では，

- 通常のメッセージセット
- piggyback(注1)を適用したメッセージセット

における実時間制約検証を行っている．

上記の場合において，文献 [5] とシミュレーション結果を比較し，実時間制約検証が正しく行えることを示す．

5.1 通常のメッセージセットにおける実時間制約検証

文献 [5] にある通常のメッセージセットを用いてシミュレータを動作させた結果を表 8 に示す．

通常のメッセージセットにおけるシミュレーション結果は，125kHz ではエラー発生確率によらずデッドライン超過となり，文献 [5] と一致した．また，250kHz において，エラー発生確率を 5%以上にすることでデッドライン超過となる場合があることを確認できた．これは，エラーの発生および発生時に行われる処理とデッドラインの短いメッセージの送信処理が重なったことなどが原因であると考えられる．

また，バス利用率について，文献 [5] よりも低い値となったが，これはパッシブエラーフラグの縮小や非周期メッセージの送信頻度に依存するものであり，実時間制約検証においては問題はないといえる．

5.2 piggyback を適用したメッセージセットにおける実時間制約検証

次に，piggyback を適用したメッセージセットを用いてシミュレータを動作させた結果を表 9 に示す．

表 7 バス調停回路を構成するクラス

バス調停回路を構成するクラスの名称	各クラスの役割
Arbiterクラス	調停回路を表すクラス．調停木を用いて出力バッファ内にあるメッセージのスケジューリングを行う
ArbNodeクラス	調停ノードを表すクラス．調停木における葉以外の部分を構成する
LeafNodeクラス	葉ノードを表すクラス．調停木における葉の部分を構成する

表 8 通常のメッセージセットにおけるバス利用率

Bus Speed	Bus Utilization / デッドライン超過回数				
	文献[5]	シミュレータ1 (エラー確率1%)	シミュレータ2 (エラー確率3%)	シミュレータ3 (エラー確率5%)	シミュレータ4 (エラー確率10%)
125kHz	125.29% あり	100% /5189	100% /7767	100% / 8039	100% / 6418
250kHz	62.64%, なし	53.23% / 0	54.06% / 0	54.93% / 1	57.46% / 1
500kHz	31.32% なし	26.58% / 0	26.97% / 0	27.66% / 0	28.77% / 0
750kHz	-	17.74% / 0	18.05% / 0	18.32% / 0	19.22% / 0
1Mhz	15.66% なし	13.31% / 0	13.62% / 0	13.78% / 0	14.40% / 0

(注1)：通常のメッセージセットに対して，送信ユニット，メッセージ送信の頻度や周期時間，メッセージのデッドラインを考慮し，複数のメッセージを 1 つにまとめる手法．この手法の適用により，送信するメッセージのヘッダを圧縮できるため，バスの利用率の向上を図ることできる．

表 9 piggyback を適用したメッセージセットにおけるバス利用率

Bus Speed	Bus Utilization / デッドライン超過回数				
	文献[5]	シミュレータ1 (エラー確率1%)	シミュレータ2 (エラー確率3%)	シミュレータ3 (エラー確率5%)	シミュレータ4 (エラー確率10%)
125kHz	84.44% / なし	80.39% / 0	81.60% / 0	83.24% / 0	86.48% / 15
250kHz	42.22% / なし	40.22% / 0	40.88% / 0	41.57% / 0	43.43% / 0
500kHz	21.11% / なし	20.20% / 0	20.47% / 0	20.85% / 0	21.62% / 0
750kHz	-	13.44% / 0	13.61% / 0	13.85% / 0	14.44% / 0
1Mhz	10.55% / なし	10.08% / 0	10.22% / 0	10.40% / 0	10.81% / 0

piggyback を適用することでメッセージの総数を減らしたシミュレーションの結果は，125kHz におけるデッドライン超過がなくなったという点で文献 [5] と一致した．また，メッセージの総数が減少したことにより，メッセージ送信におけるオーバーヘッドなどに起因する誤差が減少したため，バス利用率の値も文献 [5] に近いものとなった．

6. おわりに

本稿では，CAN プロトコルについて，効率よく実時間制約検証を行うための仕様を抽象化したモデルと，そのモデルが実時間制約を満たすかどうか検証するためのシミュレータの設計手法を提案した．モデルで抽象化した部分とシミュレータの設計方針を述べ，具体的な適用例を用いて既存のシミュレーション結果と比較することにより，妥当な実時間制約検証が行えることを確認した．

今後の実装の課題として，以下が挙げられる．

- 実時間制約検証の妥当性の指標となるカバレッジの定義
- 複数バスクロック周波数を持つシステムへの拡張
- より大きなシステムへの提案手法の適用

謝　辞

本研究を遂行するにあたり，さまざまな御助言，御協力を頂きました，大阪大学東野研究室卒業生の北口 智 氏に心から感謝します．

文　献

[1] T. Meyerowitz, C. Pinello, and A.Saniovanni-Vincentelli, "A Tool for Describing and Evaluating Hierarchical Real-Time Bus Scheduling Policies", *Proc. of 40th Design Automation Conference*, 2003

[2] K. Lahiri, A.Raghunathan, G. Lakshminarayana, and S. Dey, "Communication Architecture Tuners: a Methodology for the Design of High-Performance Communication Architecture for System-On-Chips", *Proc. of 37th Design Automation Conference*, pp. 513-518, 2000

[3] Doug Lea, "Java スレッドプログラミング　並列オブジェクト指向プログラミングの設計原理", *Object Oriented Selection*, 株式会社 翔泳社 *ISBN4-88135-918-5*, pp. 408-413, 2000

[4] RENESAS CAN/LIN MCU Web Site: "RENESAS Technology CAN 入門書 Rev. 4.00", *http://www.renesas.com/jpn/products/mpumcu/specific/can_lin_mcu/candoc/rjj99z0001_entry_0400.pdf*

[5] T. Kindell and A. Burns, "Guaranteeing Message Latencies on Control Area Network (CAN)", *Proc. of 1st International CAN Conference*, 1994

[6] 北口 智, 谷本 匡亮, 中田 明夫, 東野 輝夫, "Java 言語による階層バス調停ポリシーを持つバスシステムのモデリング及びシミュレーション", **情報処理学会システム設計技術研究会「デザインガイア** *2003*」, **情処研報** *2003-SLDM-112(27)*, pp.169-174, 2003.