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

2003－ＳＬＤＭ－112　(27)
2003／11／28

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

北口　智†　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　〒560-8531 大阪府豊中市待兼山町 1-3
E-mail: †{kitaguti,tanimoto,nakata,higashino}@ist.osaka-u.ac.jp

あらまし　本稿では、バス調停方式のリファインメントを可能にする実行可能なバスシステム記述法およびシミュレーションによる性能評価手法を提案する。提案手法では、バスを共有するモジュール間の調停を階層的な記述により与える。そして、その階層バス調停記述を含めたバスシステムの構成情報を基に生成される Java スケルトンにモジュール動作仕様を記述することで、モデルリングおよびシミュレータの構築を行う。バス調停は、ID やサイズなどのメッセージの持つプロパティやタイムスライスなどを基に、階層構造を利用してトーナメントを行うものであり、自由度の高いものを記述可能である。調停方式変更によるシステム動作変更が許容可能なものであるかどうかは、シミュレーションにて確認可能である。この環境を用い、バス調停方式のリファインメントを導入することにより、開発効率の向上を計る。二次元グラフィクス描画表示システムに適用し、本手法の有用性を示す。

キーワード　性能評価、シミュレーション、実行可能仕様、バスアーキテクチャ、Java 言語、調停方式

Modeling and Simulation of Bus System with Hierarchical Bus Arbitration Policy in Java

Tomo KITAGUCHI†, Tadaaki TANIMOTO†, Akio NAKATA†, and Teruo HIGASHINO†

† Graduate School of Information Science and Technology, Osaka University
1-3, Machikaneyama, Toyonaka, Osaka 560-8531, Japan
E-mail: †{kitaguti,tanimoto,nakata,higashino}@ist.osaka-u.ac.jp

Abstract　We propose a methodology for modelling bus systems with a variety of arbitration policies by executable specifications, and for evaluating performance by simulation. In our methodology, we produce a Java skeleton automatically by a tool from a configuration information of the target bus system and an arbitration policy for each bus shared by the modules. We can construct the model of the bus system by writing the behavior of each module into the produced Java skeleton. The resulting Java program becomes both an executable specification and a simulator. The arbitration policy supported by our tool is based on any combination of factors including time slicing and the characteristics (ex. id, size, ... etc.), combined in a hierachical structure. Using our tool, we can describe many kinds of bus arbitration policies, and check whether a modification of an arbitration policy does not affect badly the system's (timed) behavior. We also apply our methodology to a two-dimensional graphics system example in order to show its effectiveness.

Key words　performance evaluation, simulation, executable specification, bus architecture, Java, arbitration policy

1. はじめに

システム LSI は、徹底した高速化や小型化、低消費電力化、低価格化が実現可能であるため、携帯電話や携帯情報端末（PDA: Personal Digital Assistant）、車載情報システム、情報家電など様々な組み込み機器の分野で注目されている。しかしながら、システム LSI は、多様な機能の複合的な実装であるというその性質上、処理要素間の通信が複雑でありエラーを持ち込む可能性が高く、設計が容易でない。さらに、近年はその応用分野の広がりから、リアルタイム制約を持つなど、要求される仕様の複雑化も合わさり、設計がより困難な状況となっている。そのような背景で、これまでに、プロセスマッピングや通信トポロ

ジの選択に焦点を当てた、通信に着目した設計と呼ばれる手法の提案が行われている [3] [4] [5]。さらに、[1] や [2] では、これまでの手法では扱われていないバス調停[注1]方式のリファインメントを行うことで、リアルタイム制約に対し、より柔軟に対応可能とした設計手法が提案されている。

システム LSI は、複数の処理要素（モジュール）がバスを介して互いに通信を行い、ひとつのシステムとして動作しており、バスを介する通信は排他的に行う。よって、通信トポロジおよびバス調停方式の選択は、システム LSI の性能に大きな影響を与える。通信トポロジにバスの本数が多いものを選択することで、バスの競合頻度は下がり、リアルタイム制約を満たすことや、高速な処理を実現することが可能である。ただし、バスの本数が多いほど回路面積は大きくなる。これに対して、バス調停方式は、バスの競合発生時にその使用権をどのモジュールに与えるかということを指定する。リアルタイム制約を持つ処理には、速くバス権が獲得できるようなスケジューリングを与えることで、システムが制約を満たす可能性を高めることが可能である。また、各モジュールがバスを効率的に使用できるようにスケジューリングすることで、システムそのものの高速化を期待することができる。

[2] では、様々な調停方式を階層的に組み合わせることにより、要求を満たす調停方式の選定を可能にしている。しかしながら、そのシミュレーションは、処理要素の動作仕様を含まないものである。そのため、[2] は、バス調停方式の変更による動作変更が、その調停方式を持つことになるシステムにとって許容可能なものであるか判定することができないという問題を持つ。また、[6] では、処理要素間の通信シナリオを入力とし、シミュレータを汎用化することで、多くのバスアーキテクチャを短期間で評価する手法が提案されている。しかし、処理要素間の通信が複雑な場合、そのシナリオを書ききることは当然困難である。

そこで、本研究は、処理要素の動作仕様、およびバス通信トポロジ、階層バス調停方式を用いたシミュレーションを可能にした設計環境を提案する。本研究では、設計するシステムごとに、その構成情報（バス通信トポロジおよび階層バス調停方式が含まれる）から生成する Java スケルトンに、システム上の各処理要素の動作仕様を記述することで、モデリングおよびシミュレータの構築を行う。本設計環境を用いることにより、機能検証とリアルタイム制約を満たすバス調停方式の選定が同時に可能であり、実行可能な仕様（Executable Specification）を得ることができる。

本研究では、バスシステムのモデリング言語として Java 言語を用いる。その理由として以下の 4 つの特徴を挙げる。

- Java 言語は、現在、最も普及している言語のひとつであり、それを採用することで、特定のハードウェア記述言語を新たに習得すること無く、多くの人が本環境を利用可能である。
- Java 言語は、バスシステムのパラダイムを崩すことなくモデリング可能であり、Java 言語のコードからでもシステムの構成を把握しやすい。加えて、Java 言語は、C/C++などと異なりマルチスレッド機能が言語仕様で定義されており、ハードウェアの平行性を容易にまた簡潔に記述可能である。
- Java 言語では、アクセス制約やアサーション機能を用いることで、変数やメソッドの意図しない操作を禁止または防止することができる。このことは、エラー検出を容易にしデバッグにかかる時間を軽減する。
- Java 言語は、プラットフォームが異なることで、その記述形式や、動作結果が異なる事が無い。従って、作業場所を選ばない設計環境となる。また、異なるプラットフォーム上で動作確認された処理要素の動作記述を再利用することも可能である。

本稿の構成は以下の通りである。第 2 節では、スケジューリング技術について述べ、第 3 節では、対象とするシステム、および、そのシステムのバススケジューリング問題について述べる。第 4 節では、設計環境の概要として、シミュレータの構成および構築方法、階層バス調停の記述形式などについて述べ、第 5 節では、二次元グラフィクス描画表示システムへ適用し、その結果を示す。最後に、第 6 節でまとめと今後の課題について述べる。

2. スケジューリング技術

ここでは、本研究に関連するスケジューリング技術について述べる。それらは、以下の 2 つに分類できる。また、これらを合わせて用いることで、多くのアプリケーションのパフォーマンス向上が確認されている [9] [10]。

2.1 発生事象起動型

発生事象起動型スケジューリング（Event Triggered Scheduling）は、イベント要求をトリガとしてスケジューリングを行う方式である。FIFO や、固定優先度、EDF（Earliest Deadline First）スケジューリングなどが、これにあたる。FIFO は、イベントの到着順に処理を行い、固定優先度スケジューリングは、あらかじめ与えられた優先度順に処理を行う。これらは、簡単に実装可能であるが、スケジューリングの良し悪しとしては、EDF スケジューリングに劣る。EDF スケジューリングは、デッドラインに近いイベントに対して、高い優先度を動的に設定する。このため、より効果的なスケジューリングが可能である。しかし、動的に優先度を与えるというその性質上、実行オーバヘッドが大きい。

また、現在広く組み込みシステムに用いらてれる、イベントトリガバス通信プロトコルとして、CAN（Controller Area Network）がある [7]。CAN は、もともと自動車の電子制御機器を統合して制御するために開発されたバス通信プロトコルであるが、現在は、FA、船舶、医療機器、産業機器など多方面で導入されている。CAN での調停は、メッセージに付加される ID を優先度として、固定優先度スケジューリングにより行われる。また、CAN は、アドレスのような情報を持たないため、ユニットを追加する場合でも、ソフトウェア、ハードウェアお

(注1)：本稿では、「調停」および「スケジューリング」を同じ意味で用いる。正確には、どのプロセスに共有資源を与えるかということが「調停」であり、そのためのプロセスへの順番の割り当てが「スケジューリング」である。

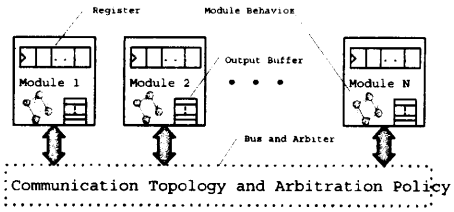

図 1　設計対象システム
Fig. 1　The Target System

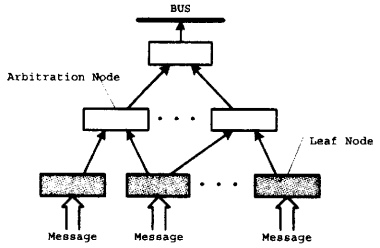

図 2　調 停 木
Fig. 2　Arbitration Tree

よびアプリケーション層に変更を加える必要がなく、システムの構成変更への高い柔軟性を持つ。

2.2　時間起動型

時間起動型スケジューリング（Time Triggered Scheduling）は、時間を用いてバスを制御するモジュールを決定する方式である。これには、TDMA（Time Division Multiple-Access）方式と FTDMA（Flexible TDMA）方式がある。TDMA 方式はシステム上のモジュールに制御可能な時間であるタイムスライスを割り振り、そのタイムスライスにあたるモジュールにバスの制御権を与えるものである。この方式は、簡単にモジュールのフェアネスを保証できるという利点がある。FTDMA 方式は、ほとんど TDMA 方式と同様であるが、タイムスライスの開始時に該当するモジュールからのメッセージが無い場合、次のモジュールにバスの制御権をまわすというものである。また、TDMA に比べ、バスの使用率は向上するが、実装が複雑になる。

また、TTP（the Time-Triggered Protocol）は、TDMA 方式を用いて調停を行うプロトコルである [8]。TTP は、CAN に比べ調停のオーバヘッドが少なく、また、レイテンシーを保証し易いが、拡張性やバス使用率は低い。主に、自動車産業や航空・宇宙産業で用いられている。

3.　設計対象システムとそのバススケジューリング問題

3.1　設計対象システム

設計の対象となるシステムは、複数個の処理要素がバスに接続されているものであり、通信トポロジは複数本のバスを用いて自由に構成可能である。図 1 に設計対象システムを示す。

処理要素は、接続されているバスごとにデータ出力のためのバッファを持ち、データ送信の際には、使用するバスの出力バッファに、送信先を指定しメッセージという形式でそれを書き込む。バッファに空きが無い場合には、待ち状態に入る。また、ある処理要素が他の処理要素にデータ送信を要求することもできる。この場合、送信要求は出力バッファと通して送られるのではなく、直接要求が要求先の処理要素に届き、要求された処理要素が出力バッファを通して、データ送信を行う。要求した処理要素は、要求したデータが届くまで待ち状態となる。

そして、出力バッファに複数個のメッセージがある場合に、調停が必要であり、どのようにスケジューリングするかということを、階層バス調停記述を用いて指定する。階層バス調停記述は、バスの数だけ必要である。

3.2　バススケジューリング問題

調停は、複数個の要素で、その数より少ない共有リソースの使用権を争う場合に必要となる。ここでは、出力バッファに複数個のメッセージが書き込まれている場合に、バスの使用権を争うことになる。そこで、その複数個のメッセージのスケジューリングを行い調停し、バスを使用できるメッセージを一つ選ぶ。本研究では、様々なスケジューリング方式を階層的に組み合わせたものを用いる。これを、調停木と呼ぶ。（ただし、実際には、葉を共有することが可能であるため、有向非巡回グラフである。）図 2 に調停木の例を示す。これを、階層バス調停記述を用いて記述する。調停木に指定できるスケジューリング方式は、第二節で挙げたスケジューリング方式、もしくは、Java 言語を用いて設計者が定義するスケジューリング方式である。ここでの目的は、デッドラインミスしないメッセージの数を最大化する調停方式を選定することである。

調停木では、葉を葉ノード、それ以外を調停ノードと呼ぶ。さらに、調停ノードに含まれるが、調停木の最上位のノードを頂点ノードと呼ぶ。そして、それぞれが、スケジューリング方式を持つ。まず、葉ノードが、その葉ノードに与えられたスケジューリング方式で最高優先度のメッセージを親の調停ノードに渡す。調停ノードでは、子ノードから渡されたメッセージを、その調停ノードが持つスケジューリング方式で最高優先度となるメッセージを、その親ノードに渡す。これを頂点ノードまで繰り返して得られるメッセージがその時間でバスを使用できるメッセージとなる。

3.2.1　定　　義

メッセージ：　メッセージは、データ値と送信元、送信先、データサイズ、到着時間、デッドライン、ID を持つ。これらの値は、スケジューリングのためにに用いられる。

葉ノード：　葉ノードは、調停木の葉にあたるものであり、親に必ず一つ以上の調停ノードを持つ。また、葉ノードには、固定優先度を除く発生事象起動型のスケジューリング方式、もしくは、Java 言語を用いて設計者が定義するスケジューリング方式を指定可能である。

調停ノード：　調停ノードは、調停木の節にあたりものであり、子ノードは葉ノードか調停ノードである。調停ノードには、全てのスケジューリング方式を指定可能である。また、頂点ノー

ドは、調停のオーバヘッドとして、朝廷にかかるクロック数を持つ。

調停木： 調停木は、必ず一つの調停ノードを持つ必要があり、かつ、頂点ノードは一つだけ持つ。

3.2.2 評価基準

調停木の評価は、シミュレーションを行い以下の項目を評価することによって行う。

- デッドラインミスしたメッセージ数
- 送信完了時間とデッドラインの差（平均値、最小値）
- 全ての処理を終えるまでにかかった時間
- 平均バススループット

4. 設計環境概要

ここでは、提案する設計環境の概要について述べる。

本環境では、バスシステム構成から生成する Java スケルトンを用いモデリングを行い、異なるシステムごとにそれ専用のシミュレータ（Java コード）を構築し、シミュレーションを行う。具体的には、まず、設計者は、システムの構成情報として、システムの処理要素数や、バス通信トポロジ、階層バス調停記述などを与え、Java スケルトンを生成する。次に、この Java スケルトンに、用意されるメソッドを用いて、システム上の各処理要素の動作仕様を記述する。この作業のみで、モデリングは完了であり、同時にシミュレータを構築したことになる。そして、シミュレーションを行い要求仕様を満たすかどうか確認する。

以下の三つの小節で、システム構成情報から出力される Java スケルトンの構成、システム構成情報、モジュールの動作記述について詳しく述べる。

4.1 シミュレータ概要

シミュレータを構成するクラスについて述べる。図 3 にクラス図を示す。

まず、大きく処理要素を表すクラス群と、バス調停回路を表すクラス群に分けられる。処理要素とバス調停回路を表すクラスは、それぞれ Thread クラスを継承しており、スレッドとして並行動作する。また、これらのクラスは、クロック同期動作のために ClockController クラスを用いる。ClockClontroller クラスは、処理要素とバス調停回路の 1 クロック間の処理の終了通知を集め（処理終了通知後の処理要素とバス調停回路は待ち状態となる）、すべての終了通知が揃ったときに、レジスタ値を更新後、クロックを次に遷移し動作を再開させる（バリア同期）。

処理要素は、Module 抽象クラスに処理内容を記述し具体化したクラスとなる。処理内容の記述については、次々小節で述べる。、処理要素は、Register クラスと OutputBuffer クラスを持つ。これは、第三節で説明した対象システムの処理要素の構成と一致する。システム上の処理要素の数だけ、Module 抽象クラスを具体化したクラスが存在する。

バス調停回路は、Arbiter クラスと調停木を構成するクラスの組み合わせを持つ。バス調停回路は、バスごとに必要であるため、バスの本数文だけ実体が存在することになる。ArbNode 抽象クラスは調停ノードを、PriorityNode 抽象クラスは葉ノードを表す。Arbiter クラスは、調停木を操作するために頂点ノードを持つため、ArbNode 抽象クラスと集約関係にある。ArbNode 抽象クラスと PriorityNode 抽象クラスのサブクラスとして、各種調停方式は具体化される。定義済みの FIFO や、固定優先度スケジューリング、TDMA 方式など以外のものを用いたい場合には、ArbNode 抽象クラスや PriorityNode 抽象クラスの具体クラスとして定義すればよい。この際、Comparator インタフェースを利用すると簡単に実装可能である。

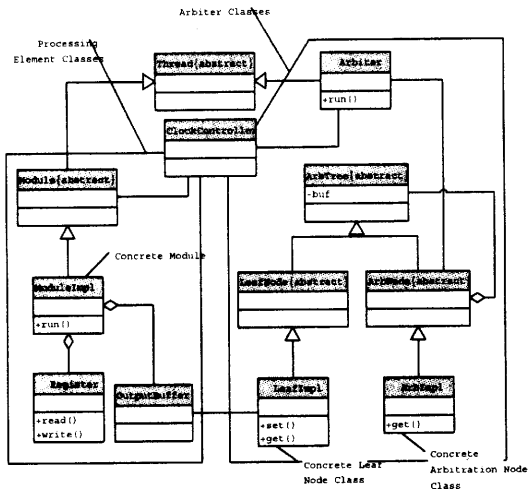

図 3 シミュレータを構成するクラス

Fig. 3 Class Diagram of Java skeleton

4.2 システム構成情報

システム構成情報は、サイクルタイムや、モジュール、レジスタの個数、バス接続情報、調停木、調停にかかるオーバヘッド（クロック数で指定）などから成る。本設計環境では、これらの情報を XML を用いて記述する。図 4 と図 5 に、記述例と出力をそれぞれ示す。これを入力として、Java スケルトンが出力される。

4.3 処理要素の動作記述

処理要素の動作は、Module 抽象クラスを具体化するクラス内の run メソッド内に記述する。処理要素は、レジスタと出力バッファを持つ（図 6 参照）。また、処理要素はシステム上の全てのレジスタへの参照を持っており、Register クラスの write メソッドや read メソッドをコールすることで、データ送信やデータ送信要求を行う。データサイズが大きく、バスのデータ幅を越え、そのデータ転送に数クロックを必要とする場合でも、そのことは意識せず、同様に記述することが可能である。メソッドが内部でそのように処理する。

簡単な記述例を示す。図 6 の (I) は、ModuleB のインデックス 3 のレジスタに、送信データを 100、経由バスを bus2、デッドラインを 0.11、ID を 1 として、送信するという例である。(II) は、ModuleC のインデックス 5 のレジスタのデータ送信を要求している。送信して欲しいデータのデッドラインを 0.43

```
<!DOCTYPE SYSTEM SYSTEM "C:\...\config.dtd">
  <SYSTEM name="ExampleSystem" cycle_time="0.11">
    <MODULE name="ModuleA">
      <REG name="RegisterA" size="10"/>
      <BUS name="bus1" id="b0"/>
      <BUS name="bus2" id="b1"/>
    </MODULE>
    <MODULE name="ModuleB">
      <REG name="RegisterA" size="7"/>
      <BUS name="bus1" id="b0"/>
      <BUS name="bus2" id="b1"/>
    </MODULE>
    <MODULE name="ModuleC">
      <REG name="RegisterA" size="4"/>
      <BUS name="bus1" id="b0"/>
    </MODULE>
  <ARBITER bus_id="b0" overhead="2">
    <A class="TDMANode" alloc_time="10">
      <P class="FIFOPriorityNode" alloc_time="20" id="b0p0"/>
      <A class="FIXEDNode" alloc_time="10">
        <P class="FIFOPriorityNode" alloc_time="20" id="b0p1"/>
        <P class="FIFOPriorityNode" alloc_time="20" id="b0p2"/>
      </A>
      <P class="FIFOPriorityNode" alloc_time="20" id="b0p2"/>
    </A>
  </ARBITER>
  <ARBITER bus_id="b1" overhead="3">
    <A class="FIFONode" alloc_time="10">
      <P class="FIFOPriorityNode" alloc_time="20" id="b1p0"/>
      <P class="FIFOPriorityNode" alloc_time="20" id="b1p1"/>
    </A>
  </ARBITER>
</SYSTEM>
```

図 4　システム構成情報の記述例

Fig. 4　Example of System Configuration File

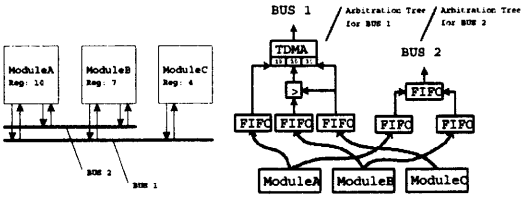

図 5　図 4 から生成されるシステム

Fig. 5　System Generated from Configuration File of Fig.4

としている。また、(III) のように指定することで、何もせずにクロックを消費することもできる。

ただし、run メソッド内でのオブジェクトのインスタンシエーションは禁止する。これは、バスシステムの動作記述制約である。インスタンシエーションは実体化を意味し、もし、これを許可する場合には、バスシステムの動作中に新たにハードウェアを構成する必要がある。例えば、run メソッド内で OutputBuffer クラスのインスタンシエーションを行った場合には、新たにもう一つ出力バッファをその処理要素に追加することを意味する。これは実現不可能であるため、その記述を禁止する。

5.　適用とその結果

本設計環境を共有メモリ方式の二次元グラフィクス描画表示システムに適用し、その結果を示す。ここでは、TDMA スケジューリング方式を基にした適用例を示す。

5.1　二次元グラフィクス描画表示システム

共有メモリ方式の二次元グラフィクス描画表示システムの概要について述べる。このシステムでは、描画コマンドに従って表示データに描画を行い、320x240 の解像度（32 ビット/ドットとする）を持つ CRT ディスプレイに、描画済み表示データを表示するものである。水平/垂直走査周期や帰線期間などの数値

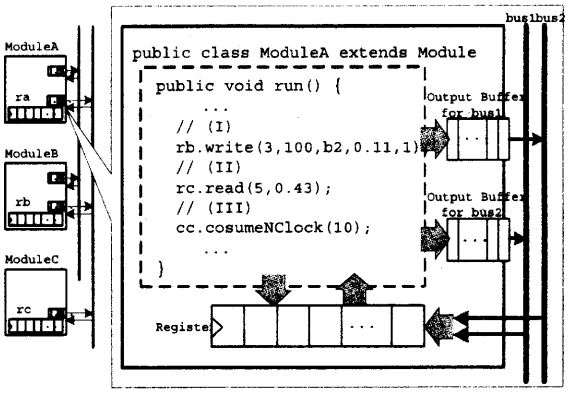

図 6　処理要素の動作記述例

Fig. 6　Example for the Behavior Description of Module

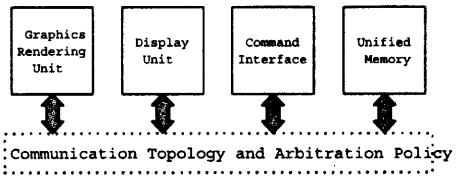

図 7　二次元グラフィクス描画表示システム

Fig. 7　Two-dimensional Graphics Rendering and Displaying System

は [11] のものを用いる。ディスプレイのリフレッシュレートは 60 回/秒、動作周波数は 7.16MHz とする。水平周期は 63.56μs であり、表示はそのうち 44.7μs の間に行う必要がある。1 コマンドの描画処理は、垂直走査周期以内に終える必要があり、処理は、水平/垂直帰線期間（9.08μs/1ms）中や水平/垂直同期パルス幅（4.47μs/190μs）など、バスが使用されていない間を用いて行う必要がある。システム構成を図 7 に示す。各処理要素の仕様は以下の通りである。

Unified Memory は、表示データと描画コマンドをまとめて格納する共有メモリである。これは、VRAM とコマンド用メモリを合わせたものである。このデバイスは、スレーブデバイスであるが、送信要求がある場合に要求されたメモリのデータを書き出す処理を記述する必要がある。

Command Interface は、外部から受け付けた描画コマンドを Unified Memory に転送する。描画コマンドには、描画データと描画を行う個所の情報が含まれている。ここでは、モデルを単純化するため、全ての描画コマンドは、同じデータサイズとする。また、Unified Memory には、描画コマンドを 2 つ格納することができ、使用状況に合わせて更新する。

Graphics Rendering Unit は、Unifyed Memory 内の描画コマンドに従い、表示データを描画する。描画コマンドを読み込み、指定されている個所に描画データを用いて描画処理を行う。

Display Unit は、Unified Memory から表示データを読み込み、それを表示する。水平帰線期間および垂直帰線期間はバスを用いる処理は行わない。

5.2 TDMA スケジューリング方式を用いた結果

各処理要素に割り当てるタイムスライスは以下の通りである。まず、Unified Memory は、Display Unit に表示データを送信するため、最低 44.7μs 以上は割り当てる必要がある。ここでは、その最低の 44.7μs（=320 クロック）を割り当てる。Command Interface と Rendering Unit は、水平帰線期間と水平パルス幅の間に、それぞれ描画コマンドや描画処理データを送信する。よって、Command Interface と Rendering Unit に、水平帰線期間と水平パルス幅を足して 2 で割った期間（48 クロック）をそれぞれ割り当てることにする。そして、残り（38 クロック）が、Display Unit への割り当てとなる（図 8 参照）。

図 8、9、10 に用いた調停木を示す。また、その結果を表 1 に示す。この結果は、リフレッシュを 10 回した直後のものである。図 8 の調停方式では、Unified Memory は、割り当てられているタイムスライスにおいて、Display Unit だけでなく、Command Interface や Rendering Unit からの送信要求にも応答していることから、表示データの送信がデッドライン以内に完了しないことがある。よって、図 9 や 10 のようにタイムスライスの共有を行った。その結果、図 9 の調停方式では、デッドラインミスが大幅に減少し、図 10 の調停方式では、デッドラインミスをすることなく、表示データの送信が完了した。

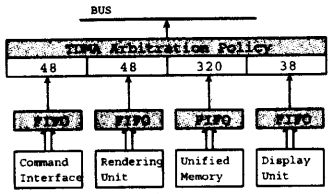

図 8 TDMA スケジューリング方式の調停木
Fig. 8 Arbitration Tree of TDMA Sheculing

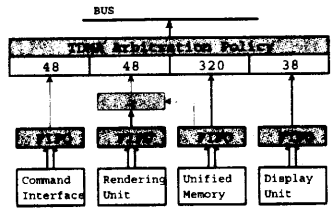

図 9 共有 TDMA スケジューリング方式の調停木 その 1
Fig. 9 Arbitration Tree of Shared TDMA Sheculing - Pattern 1

表 1 シミュレーション結果
Table 1 Simulation Results

	TDMA	Shared TDMA 1	Shared TDMA 2
# of Message	782541	771139	783799
Deadline Missed	20328	216	0
Bus Utilization	65.8%	65.6%	67.3%
Av. Deadline Slack(ms)	0.38	1.16	2.13
Min. Deadline Slack(ms)	-0.011	0.0045	0.0022

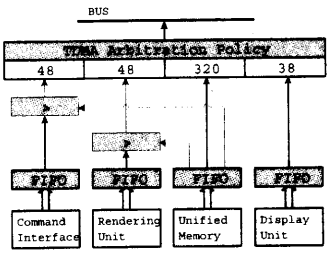

図 10 共有 TDMA スケジューリング方式の調停木 その 2
Fig. 10 Arbitration Tree of Shared TDMA Sheculing - Pattern 2

このようにして、シミュレーションにより動作確認が行われた Java コードは、実行可能な仕様として扱うことができる。

6. おわりに

本稿では、バス調停方式のリファインメントを可能にする実行可能なバスシステム記述法およびシミュレーションによる性能評価手法を提案した。階層バス調停記述およびシミュレータの概要を述べ、適用例を用いて調停方式のリファインメントを行い実行可能な仕様が得られるまでの流れを示した。

今後の課題としては、バスブリッジやプリエンプティションの実装、バス調停方式リファインメントの自動化手法の考案などが挙げられる。

文 献

[1] K. Lahiri, A. Raghunathan, G. Lakshminarayana, and S. Dey: "Communication Architecture Tuners: a Methodology for the Design of High-Performance Communication Architectures for System-On-Chips" Proceedings 2000. Design Automation Conference. p.513-18.

[2] T. Meyerowitz and A. Sangiovanni-Vincentelli: "Describing, Simulating, and Optimizing Hierarchical Bus-Scheduling Policies", Master's Report and Technical Report. UCB/ERL

[3] The Metropolis Group Web Site: "Metropolis: Design Environment for Heterogeneous Systems", http://www-cad.eecs.berkeley.edu/~polis/metro

[4] R. Ortega, G. Borriello: "Communication Synthesis for Distributed Embedded Systems", Proc. of Int. Conference on Computer Aided Design, June 1998, p. 437-444

[5] A. Pinto, L. Carloni, A. Sangiovanni-Vincentelli: "Constraint-Driven Communication Synthesis", 39th Design Automation Conference, June 2002

[6] 高橋 美和夏, 宮嶋 浩志, 福井 正博: "大規模 SoC バス・アーキテクチャ性能評価手法", 情報処理学会論文誌, vol.44 No.05 - 006

[7] RENESAS CAN/LIN MCU Web Site: "CAN 入門書 Rev. 3.00", http://www.renesas.com/jpn/products/mpumcu/specific/pdf/can_mcu/candoc/rjj99z0001_entry.pdf

[8] TTTech Web Site: "TTP の開発・設計思想", http://www.tttech.jp/ttpjpn.htm

[9] FlexRay Consortium Web Site: "FlexRay - The communication system for advanced automotive control applications", http://www.flexray-group.com

[10] L. Abeni, G. Buttazzo: "Hierarchical QoS Management for Time Sensitive Applications", Proceedings Seventh IEEE Real-Time Technology and Applications Symposium, IEEE Comput. Soc. 2001, pp.63-72

[11] RENESAS HD64413A Q2SD Literature: "HD64413A Q2SD User's Manual Rev. 2.0", http://www.eu.renesas.com/documents/automotive/hd64413a.pdf