Vol.2022-OS-154 No.9
2022/3/15

組込みシステム集約のための マルチコア制御基盤「誉」におけるコアの抽象化

林 海豊[1,a)]　井内 晴菜[2]　毛利 公一[1]

概要：既存の複数の組込みシステムソフトウェアを 1 つの計算機に容易に集約するためのマルチコア制御基盤「誉」において，集約するシステムソフトウェアの改変コストを軽減するため，コア抽象化機能の実装と評価を行ったので報告する．ハードウェアの中には，特定の CPU コアに依存した操作が必要となるものがあるため，特に CPU コアの番号を意識したソフトウェアを実装する必要がある．しかし，誉によるシステム集約や柔軟なコア割当ての際には実装時の想定通りの CPU コアが割り当てられるとは限らないため，特定の CPU コアに依存したシステムソフトウェアを，想定外の CPU コアでも動作するよう改変する必要があった．コア抽象化機能は，システムソフトウェアに抽象化したコア番号を与え，物理コア番号に依存する処理を代行する．これにより，システムソフトウェアを容易にコア非依存化することを可能とする．

1. はじめに

近年，家電や自動車といった組込み機器の高機能化により，機器を構成する計算機やデバイスの数が増加している．これに伴い，組込み機器の重量，製造コストなどの増加や内部の配線量増加によるスペースの圧迫といった問題が生じている．一方，プロセッサは高性能化しており，特にマルチコア化は組込み機器の計算機においても進んできている．そのため，組込み機器の計算機でタスクを並列実行可能となりつつある．しかし，組込み機器で動作する RTOS や OS のないソフトウェアは，シングルコアで動作するものも多く，マルチコアを活用できていない現状がある．汎用 OS，RTOS，OS のないソフトウェアといったこれらのソフトウェアは，組込みシステムの制御を行うソフトウェアであるため，本論文では一括して「システム制御ソフト」と呼称する．上記の問題を解決するために，これまで組込み機器において個別の計算機で実行されていた各システム制御ソフトを，1 つの計算機に集約したいという要求がある．集約することで**図 1** のように機器を構成する計算機を削減することが可能となり，計算機やデバイスの増加で生じていた問題を解消することができる．

以上の背景から，我々は，組込みシステムの容易な集約を実現するため，マルチコアプロセッサにおいて各コアを

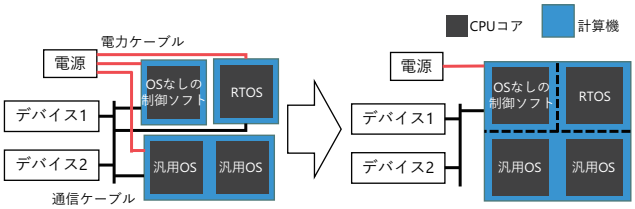

図 1　システム制御ソフトの集約

システム制御ソフトに柔軟に割り当て可能な基盤ソフトウェア「誉」を開発してきた [1]．誉でシステムを集約する際，集約対象となるシステム制御ソフトの改変が必要な場合があり，必要な改変を行って誉で動作可能としたシステム制御ソフトをゲストと呼ぶ．これまでの取り組みでは，ゲストのバイナリをメモリにロードする機能や，ゲストに CPU コアを割り当てて起動させる機能などを誉に実装してきた．この実装により，予めリソースの重複を避けた 2 つのゲストに誉から個別の CPU コアを割り当て，同時に実行できることが確認されている．

しかし，誉でシステムを集約する際，特定の CPU コアで動作することを想定しているシステム制御ソフトが問題となる．特に，CPU コアの番号を意識した実装がなされており，誉から想定外の番号の CPU コアが割り当てられるとうまく動作できない．また，ハードウェアにおいても，物理コア番号によって操作を変更する必要があるリソースの存在が問題となっている．したがって，誉によるシステム集約を実現するためには，特定コアに依存したシステム制御ソフトをどのコアでも動作するよう改変する必要が

[1] 立命館大学
Ritsumeikan University
[2] アドソル日進株式会社
Ad-Sol Nissin Corp.
[a)] krin@asl.cs.ritsumei.ac.jp

あった．そこで，本論文では，システム制御ソフトを容易にコア非依存化できるよう，システム制御ソフトに対して物理コア番号に依存しないプログラミングモデルとしてコア抽象化機能を提供する．コア抽象化機能では，コアディスクリプタと呼称する抽象化したコア番号を導入する．コアディスクリプタはゲストに割り当てたコア数だけ生成され，ゲスト毎に 0 番から昇順にシステム制御ソフトに提供する．システム制御ソフトは，与えられたコアディスクリプタを物理コア番号の代わりに使用することで，常に同じコア番号で動作することができる．ただし，物理コア番号によって挙動を変更する必要がある処理は，コアディスクリプタで動作しているシステム制御ソフトでは適切に行うことができない．そのようなコア依存の処理に関しては，誉に代行させることが可能な API を用意した．システム制御ソフトは，コア抽象化機能のコアディスクリプタおよびコア依存処理の代行 API を利用することで，物理コア番号に依らず常に同じコア番号で動作することが可能となる．

コア抽象化機能を実現するためには，CPU コア管理機能，システム構成パラメータの管理・提供機能，コアディスクリプタ提供機能，コア依存処理の代行機能の 4 つの機能を実現する必要がある．本論文では，これら 4 つの機能を実現し，コア抽象化機能を実現した．さらに，システム制御ソフトがコア抽象化機能に対応するための改変コストを抑えるため，API のスタブをまとめたライブラリを用意した．

また，コア抽象化機能の評価を行ったところ，システム制御ソフトを 30 行改変することで，コア非依存な誉用ゲストを作成できた．さらに，システム制御ソフトがコア抽象化機能を利用する際に呼び出す API のオーバヘッドは，0.6〜83µs 程度であることが確認できた．

以下，本論文では，2 章でマルチコア制御基盤「誉」について述べ，3 章で誉のコア抽象化機能について述べる．4 章でシステム制御ソフトの改変について述べ，5 章でコア抽象化機能の機能評価について述べる．6 章でコア抽象化機能の性能評価について述べ，7 章で関連研究について述べる．また，8 章で今後の課題について述べ，9 章でまとめる．

2. マルチコア制御基盤「誉」

2.1 基本コンセプト

我々は，組込みシステムの容易な集約を実現するため，マルチコアプロセッサにおいて各 CPU コアをシステム制御ソフトに柔軟に割り当て可能な基盤ソフトウェア「誉」を開発してきた．誉は，組込みシステムでの利用を想定するため，オーバヘッドは可能な限り小さいことが望ましい．また，既存のシステム制御ソフトを活用するため，システム制御ソフトを集約するための改変コストは，可能な限り小さいことが望ましい．したがって，誉では，リアルタイム性とシステム制御ソフトの移植性を確保しながら，CPU コアの柔軟な割当てを実現する必要がある．

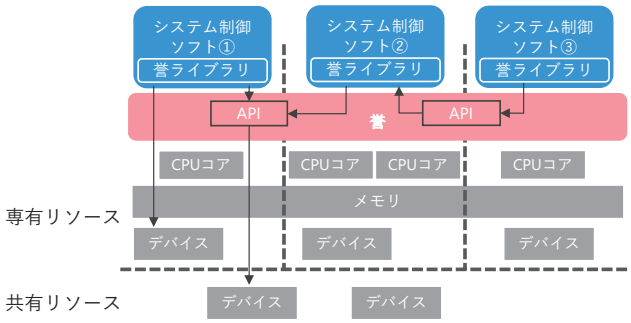

図 2 誉で集約したシステムのアーキテクチャ

誉のように単一計算機上で複数のシステムソフトウェアを実行する既存手法として，ハイパーバイザと LPAR がある．ハイパーバイザ方式では，ハードウェアリソースを管理するハイパーバイザをシステムソフトウェアの下に配置する．ゲスト間でのデバイス共有などが可能であり，柔軟なシステム構成がとれる一方，ゲストのリソースアクセスがハイパーバイザによって代行される際にオーバヘッドが生じるため，リアルタイム性が低い．また，ハイパーバイザは，完全仮想化と準仮想化に細分されるが，両方式はゲストの移植性とリアルタイム性のトレードオフの関係にある．LPAR は，ハードウェアリソースを論理的に分割し，各ゲストに割り当てる方式である．ゲストは割り当てられたリソースに直接アクセス可能であるため，リアルタイム性が高い．一方で，デバイスの共有などの実現が難しく，集約可能なシステムに制限がある．

これらの既存手法では，誉に必要な特性を満たしながら，幅広い構成の組込みシステムを集約することは難しい．そこで，誉は，準仮想化のハイパーバイザと LPAR を組み合わせた方式で実現する．誉によって組込みシステムを集約した際のアーキテクチャを**図 2** に示す．誉は LPAR をベースとし，各ゲストに専有させる CPU コア，メモリ，デバイスを直接割り当てる．LPAR で割り当てたリソースは，ゲストがアクセスする際のオーバヘッドがないため，リアルタイム性を確保できる．一方，デバイスの共有などといった LPAR で実現困難な処理は，API を用意して準仮想化で実現し，幅広い構成のシステムを集約可能とする．また，システム制御ソフトの移植性を確保するため，API のスタブを誉ライブラリとして提供する．このような方式を取ることで，誉に必要なリアルタイム性とシステム制御ソフトの移植性を確保しながら，柔軟な CPU コアの割当てによる組込みシステムの集約が実現できる．

2.2 機能

2.1 節で述べた構成を実現するため，誉では以下に示す機能を実現する必要がある．

CPU 管理
CPU 管理機能は，CPU コアの柔軟な割当てを実現するため，ゲストへの CPU コアの割当てや割り当てた CPU コアの抽象化を行う．

メモリ管理
メモリ管理機能は，ゲスト間で使用するメモリが重複しないように調整する．メモリ管理機能によって，重複のないメモリ領域の割当てやメモリ領域間の保護を実現する．

デバイス管理
デバイス管理機能は，ゲストによるデバイス利用を補助したり，複数のゲストで同じデバイスを操作することを防いだりする．デバイス管理機能によって，デバイスの専有や共有を実現する．

ゲスト起動補助
ゲスト起動補助機能は，CPU コアを割り当てたゲストの起動を補助する．ゲストのバイナリの管理・ロードや，ゲストの動作環境情報であるシステム構成パラメータの管理・提供を行う．また，ゲストを起動する際に起動環境の調整を行う．

各種サービス
各種サービスは，組込みシステムを 1 つの計算機に集約することで実現できなくなった操作を代替する．たとえば，ネットワークを用いた計算機間通信の代替であるコア間通信サービスや，シリアルケーブルを用いたログ出力の代替であるログ出力サービスがある．

これら機能のうち，CPU 管理，メモリ管理，ゲスト起動補助，各種サービスについては先行研究 [1] で一部が実装されている．この実装により，使用する CPU コア，メモリ，デバイスが重複しないよう改変した 2 つのゲストに対して，誉で個別の CPU コアを割り当てて同時に実行可能なことが確認されている．

2.3 動作概要

誉システムの動作概要は以下の通りである．なお，ゲストの起動処理は，誉のシェルで対話的に行うか，予め処理を設定して自動的に行う．

(1) 誉を起動する
  (a) 誉が全ての CPU コアを初期化する
  (b) 1 つの CPU コアで誉シェルを動作させ，それ以外のコアは休止させる
(2) ゲストを起動する
  (a) 誉が SD カードをマウントし，ゲストのバイナリをロードする
  (b) ゲストに CPU コアを割り当て，割り当てられた CPU コアの処理を再開させる
  (c) 再開した CPU コアで環境の調整を行った後，ゲストの初期化処理にジャンプする

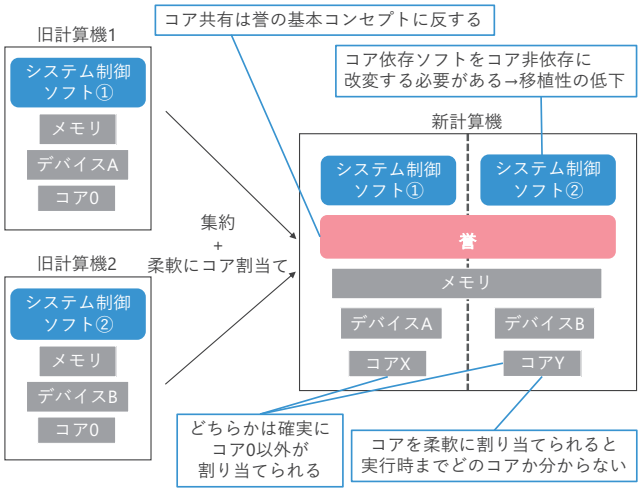

図 3 コア依存ソフト集約時の問題

(3) ゲストからの API 呼び出し
  (a) ゲストの起動後，ゲストは，必要に応じて API を呼び出す
  (b) 誉ライブラリの API スタブが，呼出し規約通りに引数を格納して誉の API ハンドラにジャンプする
  (c) 誉の API ハンドラは，ゲストに指定された処理を行った後，呼び出し元ゲストにリターンする

3. コア抽象化機能

3.1 問題定義

誉で組込みシステムを集約する際，特定の CPU コアで動作することを想定しているシステム制御ソフトが問題となる．特に，動作する CPU コアの番号を意識した実装がなされており，誉から想定外の番号のコアが割り当てられるとうまく動作できない．

そのような特定の CPU コアに依存しているシステム制御ソフトを集約する際に生じる問題を**図 3** に示す．コア 0 に依存しているシステム制御ソフト①と②を集約した場合，誉は CPU コアを LPAR で割り当てるため，どちらかのシステム制御ソフトにはコア 0 以外の CPU コアが割り当てられることになる．そのため，コア 0 以外の CPU コアが割り当てられるシステム制御ソフトは，割り当てられる CPU コアで動作できるよう改変する必要がある．

また，誉は，マルチコアを活用するため，その時々で空いている CPU コアをシステム制御ソフトに柔軟に割り当てたい．しかし，システム制御ソフトとしては，実行開始までどの CPU コアが割り当てられるか不明なため，どの CPU コアでも動作できるよう改変する必要がある．

このように，誉によってシステム制御ソフトを集約したり柔軟なコア割当てを行ったりする場合，特定の CPU コアに依存したシステム制御ソフトの改変が必要となる．これにより，誉への対応コストが増加し，システム制御ソフトの移植性が低下することが予測される．

上記の問題に対して，誉がシステム制御ソフト間で CPU コアを共有させることで対応することも考えられる．しかし，コア共有を行うと，CPU コアのスケジューリングを行う必要があり，リアルタイム性が低下してしまう．また，マルチコアを活用できないため，コア共有は誉の基本コンセプトに反する．したがって，誉によるシステム集約や柔軟なコア割当てを実現するためには，システム制御ソフトを容易にコア非依存化できる必要がある．

3.2 解決手法の概要

3.1 節で述べた問題を解決するため，システム制御ソフトが物理コア番号とそれに依存した処理を抽象化して扱えるようにするコア抽象化機能を誉で実現した．コア抽象化機能の概要を**図 4** に示す．問題の解決策として，システム制御ソフトが常に同じコア番号で動作できるよう，抽象化したコア番号（以下，コアディスクリプタ）を与えることが考えられる．コアディスクリプタは，システム制御ソフトに割り当てる CPU コアの番号に依らず，常に同じ番号を与えることが可能である．そのため，システム制御ソフトは，物理コア番号をコアディスクリプタで代用することで，常に同じコア番号で動作することが可能となる．

ただし，システム制御ソフトが行う処理の中には，物理コア番号に依存して挙動を変える必要があるものが存在する．そのようなコア依存処理の一例として，割込みコントローラにおけるルーティング操作が挙げられる．割込みコントローラでは，発生した割込みをどの CPU コアにルーティングするかを設定することが可能である．そのため，システム制御ソフトが自身の CPU コアに割込みをルーティングするためには，自身が動作する CPU コアの番号を元に設定する必要がある．物理コア番号とコアディスクリプタは必ずしも同じ値ではないため，システム制御ソフトがコアディスクリプタを元にコア依存処理を行うと，意図しない処理を行ってしまう可能性がある．そこで，コア依存処理に関しては，誉に代行させることが可能な API を用意する（以下，コア依存処理代行 API）．システム制御ソフトは，コア依存処理を行う代わりに，コア依存処理代行 API を呼び出すようにする．このとき，システム制御ソフトに複数の CPU コアが割り当てられていると，誉は処理がどの CPU コアに依存しているかが判別できない．そのため，システム制御ソフトは，コアディスクリプタによって依存先の CPU コアを指定する．API で呼び出されたコア抽象化機能は，与えられたコアディスクリプタを対応する物理コア番号に変換し，コア依存処理を代行する．したがって，システム制御ソフトウェアは，コアディスクリプタのみを意識したままコア依存処理を行うことが可能となる．

3.3 ソフトウェア構成

コア抽象化機能のソフトウェア構成を**図 5** に示す．コア

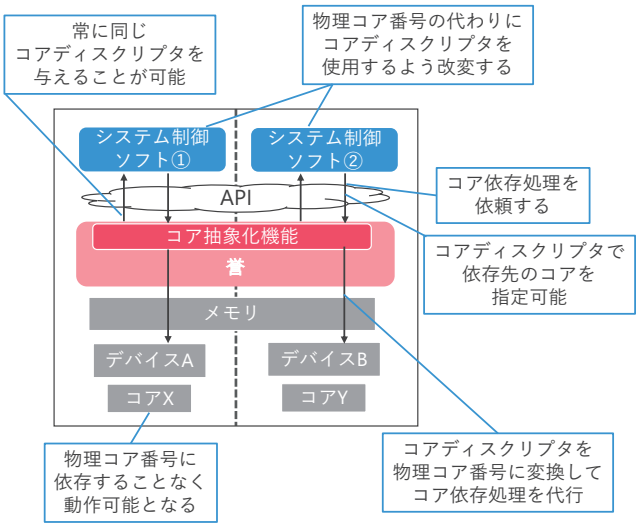

図 4 コア抽象化機能の概要

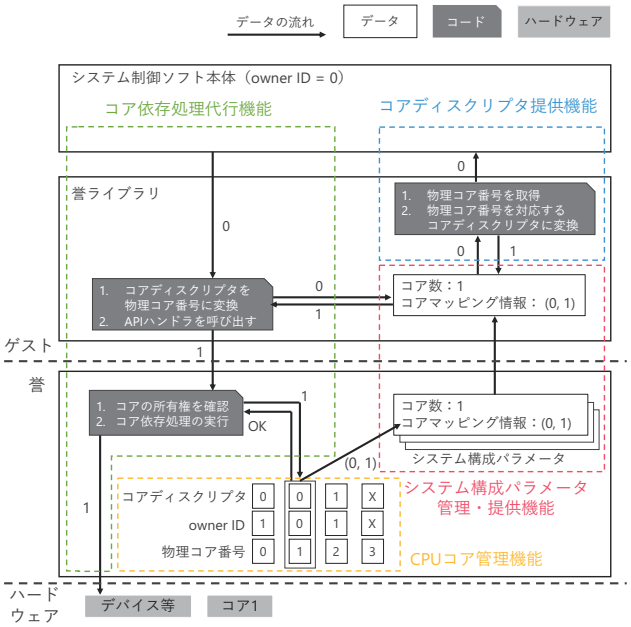

図 5 コア抽象化機能のソフトウェア構成

抽象化機能は，誉本体とゲストに組み込まれた誉ライブラリによって実現し，以下に示す 4 つの機能で構成される．

- CPU コア管理機能
- システム構成パラメータの管理・提供機能
- コアディスクリプタの提供機能
- コア依存処理の代行機能

これらの機能を**表 1** に示す誉の動作環境に実装し，誉のコア抽象化機能を実現した．誉は，組込みシステムで広く使用されている Arm プロセッサの Cortex-A シリーズを対象としている．以降の節では，コア抽象化機能を構成するこれらの機能および図 5 における動作例について述べる．

3.3.1 CPU コア管理機能

CPU コア管理機能は，物理コア番号，コアディスクリプタ，owner ID の管理を行う．owner ID は，その CPU コ

表 1　誉の動作環境

ボード	Raspberry Pi 3 Model B
CPU コア	Cortex-A53 * 4 コア
アーキテクチャ	Armv8-A aarch32
メモリ	1GB

アを所有するゲストを示す ID であり，コア依存処理の代行時に CPU コアの所有権を確認するために用いられる．

図 5 の場合，CPU コア管理機能は，owner ID 0 および owner ID 1 の 2 つのシステム制御ソフトに割り当てた CPU コアについて管理している．図に示されているシステム制御ソフトの owner ID は 0 であり，物理コア番号 1 番の CPU コアが割り当てられている．また，対応するコアディスクリプタは 0 番となっている．なお，図には示されていない owner ID 1 のシステム制御ソフトには，物理コア番号 0 番と 2 番の CPU コアが割り当てられており，対応するコアディスクリプタは 0 番と 1 番である．

3.3.2　システム構成パラメータの管理・提供機能

コア抽象化機能では，システム制御ソフトにコアディスクリプタを提供する必要がある．そこで，システム制御ソフトの動作環境情報をシステム構成パラメータとして提供する．このシステム構成パラメータに，コアディスクリプタと物理コア番号の対応関係をコアマッピング情報として格納する．そのため，システム制御ソフトは，提供されたシステム構成パラメータから自身のコアディスクリプタを把握し，利用することができる．システム構成パラメータの管理・提供機能では，システム構成パラメータの作成，削除，システム制御ソフトへのパラメータ提供などを行う．

システム制御ソフトは，homare_get_system_params API を呼び出すか，特定のメモリ番地に配置されたパラメータに直接アクセスすることで，自身のシステム構成パラメータを取得することができる．homare_get_system_params のインタフェースは，以下の通りである．

```
void homare_get_system_params()
```

図 5 では，owner ID 0 のシステム制御ソフトについてシステム構成パラメータの作成と提供を行っている．CPU コア管理機能から，owner ID 0 のシステム制御ソフトに割り当てた CPU コアについて，コアディスクリプタと物理コア番号の対応関係 (0, 1) を取得し，システム構成パラメータに格納している．このようにして作成されたシステム構成パラメータは，owner ID 0 のシステム制御ソフトの誉ライブラリに提供される．

3.3.3　コアディスクリプタ提供機能

コアディスクリプタ提供機能は，システム構成パラメータのコアマッピング情報を元に，システム制御ソフトにコアディスクリプタを提供する homare_get_cpuid API である．homare_get_cpuid のインタフェースを以下に示す．

```
uint32_t homare_get_cpuid()
```

システム制御ソフトは，この API を呼び出すことで，現在動作中の物理コア番号に対応したコアディスクリプタを取得することができる．取得したコアディスクリプタを物理コア番号の代わりに使用することで，システム制御ソフトは常に同じコア番号で動作することが可能となる．

図 5 の場合，owner ID 0 のシステム制御ソフトには，コア 1 が割り当てられているため，コアディスクリプタ提供機能は物理コア番号 1 番を取得する．システム構成パラメータのコアマッピング情報 (0, 1) を参照し，物理コア番号 1 番に対応するコアディスクリプタ 0 番をシステム制御ソフトに提供する．

3.3.4　コア依存処理の代行機能

コア依存処理の代行機能は，システム構成パラメータのコアマッピング情報を元に，誉にコア依存処理を代行させる API を提供する．システム制御ソフトが処理依存先のコアディスクリプタを指定して本 API を呼び出すと，API スタブが，コアディスクリプタをコア依存処理に用いる物理コア番号に変換して誉の API ハンドラを呼び出す．API ハンドラでは，依存先の CPU コアが呼び出し元のゲストのものであるかを確認した後，コア依存処理を代行する．コア依存処理の代行機能により，システム制御ソフトは，コアディスクリプタのみを意識したまま物理コア番号に依存した処理を行うことができる．

コア依存処理は，Mailbox によるコア間通信処理や割込みコントローラの操作などがあるが，一例として割込みコントローラの操作について代行する API を実装した．なお，システム制御ソフトの移植性とリアルタイム性を考慮し，1 つの割込みに対しての割込みコントローラ操作を容易に呼び出せる homare_irq_config API と，一度に複数の割込みを操作可能な homare_irq_config_multi API の 2 つを用意した．それぞれのインタフェースは，以下の通りである．

```
int homare_irq_config(uint32_t cd,
            uint32_t irqID, uint32_t opID)
```

```
int homare_irq_config_multi(uint32_t cd,
        uint32_t irqregID, uint32_t irqID,
                           uint32_t opID)
```

homare_irq_config API は，引数 cd（コアディスクリプタ）で指定した CPU コアと引数 irqID で指定した 1 つの割込みに対して，引数 opID で指定した割込みコントローラ操作を行う．また，homare_irq_config_multi API は，irqregID と irqID の 2 つの引数によって，一度に複数の割込みを指

定可能である．

図 5 では，owner ID 0 のシステム制御ソフトが，コアディスクリプタ 0 番に依存した処理を誉に代行させている．誉ライブラリでは，システム構成パラメータのコアマッピング情報 (0, 1) を参照してコアディスクリプタ 0 番を物理コア番号 1 番に変換し，誉の API ハンドラを呼び出す．API ハンドラでは，コア依存処理を行う前に，物理コア番号 1 番が owner ID 0 のものであるか CPU コア管理機能に確認する．確かにコア 1 は owner ID 0 のものであるため，API ハンドラは物理コア番号 1 番に依存したコア依存処理を実行する．

4. システム制御ソフトの改変

システム制御ソフトが誉のコア抽象化機能を利用する際，システム制御ソフト内のコア依存処理をコア依存処理代行 API に置き換えるなどの改変が必要である．ただし，誉ライブラリで用意している API スタブを利用することで，システム制御ソフト改変量を軽減することが可能である．また，誉で複数のシステム制御ソフトを集約する場合，共有するリソースを除いて互いに干渉しないよう改変する必要がある．

本章では，システム制御ソフトの一つである TOPPERS/ASP3 カーネル（以下，ASP3）[2][3] を，誉で 2 つ同時に実行するために行った改変について述べる．以降，同時に実行する ASP3 の 1 つを ASP3-A，もう 1 つを ASP3-B と呼称する．

システム制御ソフトに施す改変は，目的によって以下の 2 つに大別できる．オリジナル ASP3 に改変 (1) を施すことで，コア抽象化機能を利用する ASP3-A を作成し，さらに ASP3-A に改変 (2) を施すことで，コア抽象化機能を利用しつつ ASP3-A と同時実行可能な ASP3-B を作成した．

(1) コア抽象化機能を適用させるための改変
(2) 複数システム制御ソフトを同時実行するための改変

4.1 改変 (1)

システム制御ソフトが誉のコア抽象化機能を利用するためには，以下に示す改変を行う必要がある．実際に ASP3 に改変 (1) を施したところ，合計 30 行で改変を行えた．

- 誉ライブラリをシステム制御ソフトのイメージに取り込む．なお，誉ライブラリ自体の規模は，234 行である．
- 初期化時に homare_get_system_params API を呼び出し，システム構成パラメータを取得する
- 物理コア番号を取得する処理を，homare_get_cpuid API の呼出しに置き換え，コアディスクリプタを取得するようにする
- コア依存処理を，対応するコア依存処理代行 API の呼出しに置き換える

4.2 改変 (2)

現状，誉で複数のシステム制御ソフトを同時実行する場合，各システム制御ソフトが使用するメモリと専有するデバイスに関して，重複を避けるようシステム制御ソフト間で調整する必要がある．ASP3-A と ASP3-B は，CPU コアの他に，以下に示すリソースを使用する．

- メモリ
- Generic Timer
- LED

したがって，上記のリソースに関しては，ASP3 間で競合を避ける必要がある．ただし，Generic Timer は CPU コア毎に存在しており，システム制御ソフトが Generic Timer を操作する際，自身が動作する CPU コアの Generic Timer が自動的に操作対象となる．そのため，Generic Timer に関しては，重複を避ける必要はない．実際に ASP3-A に改変 (2) を施したところ，合計 18 行の改変を行うことで，ASP3-A と同時実行可能な ASP3-B が作成できた

5. 機能評価

5.1 評価方法

誉で動作させるゲストとして，4 章で改変した ASP3-A および ASP3-B を用いる．コア抽象化機能によって ASP3-A と B がどの番号のコアでも動作可能なことを確認するため，両 ASP3 に割り当てる CPU コアが異なるような 2 つのシステム構成で評価を行った．機能評価に用いた 2 つのシステム構成を**図 6** に示す．システム構成 A では ASP3-A にコア 0，ASP3-B にコア 1 を割り当て，システム構成 B では ASP3-A にコア 1，ASP3-B にコア 0 を割り当てる．ASP3 はコア番号 0 番でのみ動作を開始するため，コア抽象化機能からコアディスクリプタ 0 番を提供する必要がある．また，両 ASP3 は，Generic Timer によるタイマ割込みが生じる度に LED を点滅させる．タイマ割込みを扱うためには，コア依存処理である割込みコントローラの操作が必要であるため，ASP3 が LED を点滅させることができれば，コア依存操作の代行が機能しているといえる．

5.2 評価結果

システム構成 A とシステム構成 B において ASP3-A と ASP3-B を実行したところ，**図 7** のような赤色 LED と青色 LED の点滅が確認できた．したがって，誉のコア抽象化は機能しており，ASP3-A と ASP3-B は，コア抽象化機能を利用してコア非依存化可能なことが確認できた．

6. 性能評価

6.1 評価項目

誉において，CPU コアは LPAR 的に割り当て，各ゲストに個別の CPU コアを専有させている．そのため，ゲストが CPU コアを利用する際のオーバヘッドはない．コア

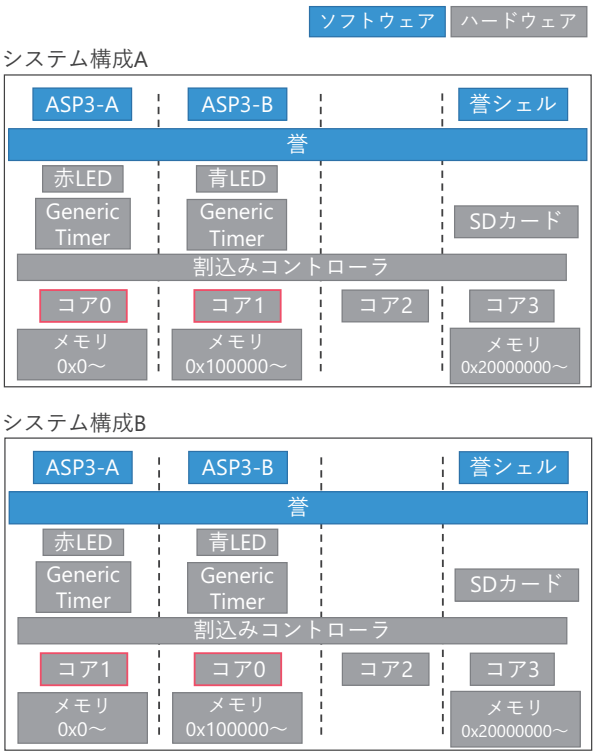

図 6 機能評価で用いたシステム構成

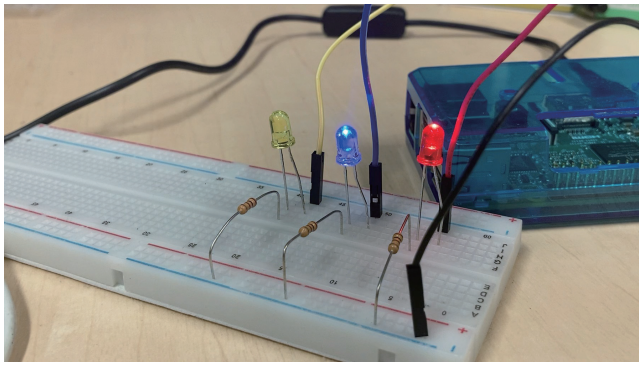

図 7 ASP3-A，ASP3-B による LED の点滅

抽象化機能のオーバヘッドは，ゲストがコア抽象化機能を利用する際に呼び出す以下の API の処理時間である．

- システム構成パラメータ取得 API
- コアディスクリプタ提供 API
- コア依存処理代行 API

ただし，システム構成パラメータ取得 API は，ゲストの初期化時に 1 度だけ呼び出せばよいため，性能への影響は小さい．一方コアディスクリプタ提供 API と，コア依存処理代行 API の一例として実装した割込みコントローラ操作代行 API は，ゲストの実行中に何度も呼び出される可能性があるため，これらの API の所用時間を計測した．

6.2 評価方法

誉の上で ASP3 を動作させ，ASP3 から各 API を呼び出し，API から戻ってくるまでの所用時間を 100 回ずつ計測した．なお，割込みコントローラ操作代行 API の所用時間は，以下に示す各条件に影響されると考えられるため，計測はこれら条件を変更した 29 パターンで行った．

- 使用する API
- 操作対象の割込みが属する割込みコントローラ
- 行う割込みコントローラ操作
- 操作する割込みの数

6.3 計測結果と考察

コアディスクリプタ提供 API の所用時間は 0.673〜1.260µs であった．また，割込みコントローラ操作代行 API の所用時間は，9〜83µs であった．

これらコア抽象化機能の API のオーバヘッドが問題となるかは，誉を適用する組込みシステムに依存する．たとえば，一般的な車両制御において，制御周期は 1〜10ms と想定されている [4][5][6]．このような車両制御周期と比較した場合，コア抽象化機能の API のオーバヘッドは十分に小さいといえる．

7. 関連研究

まず，SPUMONE[7]，DARMA[8]，SafeG[9] などの，シングルコア環境で複数のシステムソフトウェアを動作させる研究がある．これらは，1 つの CPU コアを複数のシステムソフトウェアで共有する必要があるため，CPU コアのスケジューリングが必要であり，そのオーバヘッドが存在する．一方，誉は，マルチコア環境を想定し，各システムソフトウェアに CPU コアを専有させる．そのため，CPU コアのスケジューリングを行う必要はなく，システムソフトウェアが CPU コアを利用する際のオーバヘッドは存在しない．

また，Xen[10][11]，eMCOS Hypervisor[12]，QNX Hypervisor[13] などの，マルチコア環境に対応した組込みシステム向けハイパーバイザがある．これらは，CPU コアやデバイスをゲスト間で共有できるほか，1 つのゲストに専有させることでアクセス時のオーバヘッドを削減することも可能である．また，コードサイズが軽量であるため，安全性の検証コストやメモリ使用量が小さいという特徴がある．誉も同様に CPU コアやデバイスの専有割当てが可能であるため，誉は組込みシステムを集約するアプローチとして適切であることが分かる．ただし，誉は，組込みシステムの集約に必要な機能のみを実装している軽量なマルチコア制御基盤である．基本的に LPAR を用い，可能な限り仮想化を行っていない．また，仮想化する場合も，エミュレーションが必要な完全仮想化で実現するのではなく，よりオーバヘッドの小さい準仮想化で実現している．したがって，誉は，これら組込みシステム向けハイパーバイザと比較してコードサイズやオーバヘッドがより小さいといえる．

8. 今後の課題

8.1 メモリ管理機能とデバイス管理機能の実装

2 章で提案した誉の機能の内，メモリ管理機能とデバイス管理機能の一部が未実装である．具体的には，ゲストが使用しているメモリ領域間の保護と，デバイスの専有・共有機能である．誉の動作環境である Arm アーキテクチャにおいて，デバイスの操作は MMIO で実現されている．そのため，メモリとデバイスのアクセス制御は，MMU によって実現できる．Arm アーキテクチャの MMU は，2 段構造になっており，ゲストが管理可能な Stage-1 MMU と，ハイパーバイザが管理可能な Stage-2 MMU がある．Stage-2 MMU のアドレス変換を誉が管理することで，ゲストによるメモリやデバイスへのアクセスを制御できる．

8.2 システム構成パラメータの提供方法

現状，システム構成パラメータの提供は，システム制御ソフトから API を呼び出すか，システム制御ソフト自ら特定のメモリアドレスにアクセスすることによって実現される．しかし，この 2 つのパラメータ提供方式では，既存の方法でシステム構成を把握するようなシステム制御ソフトに対応できていない．システム構成を把握するための既存手法として，Device Tree や BIOS/UEFI がある．これら既存手法でシステム構成を把握するシステム制御ソフトを誉で集約する際には，誉に Device tree や BIOS/UEFI と同等のシステム構成パラメータ提供方法を用意するか，システム制御ソフトを改変する必要がある．

8.3 64bit モードや他のアーキテクチャへの対応

現状の誉は，Arm アーキテクチャの 32bit モードに対応している．近年では組込みシステムにおいても 64bit 化が進められており，RISC-V など他の組込み対応アーキテクチャも登場している．誉がこれらに対応することで，誉の適用範囲を拡大させることが可能である．

9. おわりに

本論文では，組込みシステムを 1 つの計算機に容易に集約するためのマルチコア制御基盤「誉」におけるコアの抽象化機能について述べた．誉で組込みシステムを集約する際，特定の CPU コアに依存しているシステム制御ソフトが問題であった．特に，CPU コアの番号に依存したプログラミングがなされていることで，誉によるシステム集約や柔軟なコア割当てが困難であった．また，この問題をシステム制御ソフト側のみで対応しようとすると，システム制御ソフトの改変コストが増加し，移植性が低下してしまう．そこで，コアディスクリプタという抽象化されたコア番号をシステム制御ソフトに提供することで，システム制御ソフトが常に特定のコア番号で動作することを可能とした．また，物理コア番号に依存した処理に関しては，誉が処理を代行する API を用意した．コア抽象化機能により，システム制御ソフトを容易にコア非依存化可能とし，誉によるシステム集約や柔軟なコア割当ての実現が容易になった．コア抽象化機能の評価を行ったところ，システム制御ソフトを 30 行改変することで，コア非依存な誉用ゲストを作成できた．性能評価では，ゲストがコア抽象化機能を利用する際に呼び出す API のオーバヘッドを計測し，0.6〜83µs 程度であることが確認できた．

参考文献

[1] 渡部聡也，田中　玲，後藤秀樹，鄭　俊俊，毛利公一：組込みシステム統合のためのマルチコア制御基盤「誉」の設計，情報処理学会研究報告，Vol. 2021-OS-151, No. 5, pp. 1–8 (2021).
[2] TOPPERS Project Inc.: TOPPERS プロジェクト/ASP3，TOPPERS プロジェクト（オンライン），入手先〈https://www.toppers.jp/asp3-kernel.html〉(参照 2022-1-18).
[3] AZO234: Raspberry Pi ターゲット依存部 for TOPPERS/ASP3, GitHub(オンライン), 入手先〈https://github.com/AZO234/RaspberryPi_TOPPERS_ASP3〉(参照 2022-1-18).
[4] 小河原徹，山本泰生，曾　珍，廣部直樹：組込機器でリアルタイム実行可能なセキュリティ攻撃検知・分類アルゴリズムの実装，Omron technics: オムロングループ技術論文誌，Vol. 52, No. 1, pp. 112–118 (2020).
[5] 徳永雄一，西山博仁，伊藤益夫，千田修一郎：CAN を用いた階層統合型車載ネットワークの提案，情報処理学会論文誌，Vol. 59, No. 11, pp. 2074–2084 (2018).
[6] 石郷岡祐，横山孝典：組み込み制御システム向け時間駆動分散オブジェクト環境，情報処理学会論文誌，Vol. 48, No. 9, pp. 2936–2945 (2007).
[7] Kanda, W., Yumura, Y., Kinebuchi, Y., Makijima, K. and Nakajima, T.: SPUMONE: Lightweight CPU Virtualization Layer for Embedded Systems, *Proc. 2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, pp. 144–151 (2008).
[8] 新井利明，関口知紀，佐藤雅英，井上太郎，中村智明，岩尾秀樹：ナノカーネル方式による異種 OS 共存技術「DARMA」の提案，全国大会講演論文集，第 59 回，アーキテクチャ，pp. 159–160 (1999).
[9] 中嶋健一郎，本田晋也，手嶋茂晴，高田広章：セキュリティ支援ハードウェアによるハイブリッド OS システムの高信頼化，電子情報通信学会論文誌 D，Vol. J93-D, No. 2, pp. 75–85 (2010).
[10] Barham, P., Dragovic, B., Fraser, K. et al.: Xen and the Art of Virtualization, *SIGOPS Oper. Syst. Rev.*, Vol. 37, No. 5, pp. 164–177 (2003).
[11] Xen Project: Xen ARM with Virtualization Extensions whitepaper, Xen Project Wiki (online), available from 〈https://wiki.xenproject.org/wiki/Xen_ARM_with_Virtualization_Extensions_whitepaper/〉 (accessed 2021-01-26).
[12] イーソル株式会社：eMCOS Hypervisor, イーソル株式会社（オンライン），入手先〈https://www.esol.co.jp/embedded/emcos-hypervisor.html〉(参照 2022-1-18).
[13] BlackBerry QNX: QNX Hypervisor, BlackBerry QNX (online), available from 〈https://blackberry.qnx.com/en/products/foundation-software/qnx-hypervisor〉 (accessed 2022-1-18).