Vol.2021-OS-151 No.5
2021/3/1

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

渡部 聡也[1,a)]　田中 玲[2]　後藤 秀樹[2]　鄭 俊俊[1]　毛利 公一[1]

概要：1 つの機器上で複数の組込み向けシステム制御ソフトウェアを動作させるために，システム制御ソフトウェアに各コアを割り当て，ソフトウェア間で共有されるハードウェアリソースへのアクセスのみ制御する軽量なソフトウェア基盤「誉」を設計し，基本部分の実装と機能評価を行ったので報告する．現在は，複数のシステム制御ソフトウェアを動作させる手法としては仮想化手法が一般的であるが，ハードウェアリソースへのアクセスはエミュレーション処理を介して行われるため，アクセスの度にオーバヘッドとなり，組込みソフトウェアの実行時間に影響を及ぼしてしまう．また，既存の仮想化手法では，限られたリソースしか持たない組込み機器上で，ハードウェア抽象化機能，タスク管理機能が揃った OS と OS なしで動作する組込みソフトウェアが同時に複数動作することは考慮されていなかった．そこで，マルチコア技術を活用しつつ，最低限のハードウェアへのアクセス制御を行う軽量な組込み向けソフトウェア基盤「誉」の実現によりこれらを解決する．

1. はじめに

IoT や AI の普及により，家電や，車といった製品は高機能化，高性能化が要求されており、内蔵する組込み機器の数が増加している．製品が内蔵する組込み機器は，制御を行うソフトウェアを搭載しており，これらのソフトウェア間で連携することによって製品のシステムを構築している．よって，製品が内蔵する組込み機器が増えるだけでなく，組込み機器同士が連携するためのケーブル，ワイヤハーネスといった物理インタフェースの数も増加する．製品に内蔵される組込み機器や，物理インターフェースが増えることにより，以下の問題が発生する．

- 製品の内部スペースの圧迫
- 消費電力の増加
- 材料費などの製作コストの増加
- 製品全体の重量の増加
- ネットワークの複雑化

上記の問題を解決するために複数の制御ソフトウェアを 1 つの組込み機器に集約したいという要求がある．

組込み機器が搭載するソフトウェアには，リアルタイム性が重視されるモータ制御などの単一の動作を行うアプリケーションや，FreeRTOS[1]，mbedOS[2]，Toppers/ASP[3] などのリアルタイム OS，高機能な汎用 OS がある．これらは組込み機器において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，これらをまとめて「システム制御ソフトウェア」という名称を用いることとする．1 つの組込み機器に複数のシステム制御ソフトウェアの統合を行うと図 1 に示すように組込み機器の数を削減することができる．また，電源を必要とする機器が 1 つになるため，組込み機器に電力供給を行うケーブルの数も削減することができる．さらに，コア間通信や共有メモリを用いたソフトウェア間通信が可能となり，組込み機器間の通信インタフェースを削減することができる．

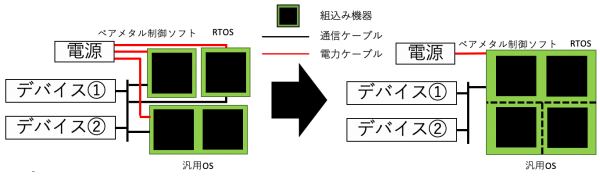

図 1　複数組込みソフトウェアの統合

複数のソフトウェアを集約するためには，互いに影響を及ぼさないためのソフトウェア構成とハードウェアの扱い方について検討する必要がある．1 つの機器で複数のソフトウェアを集約するためのソフトウェア構成として，システム制御ソフトウェア間の調整を行うための制御基盤ソフトウェアを配置する構成と，動作させるシステム制御ソフ

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

トウェアの 1 つを改変してシステム制御ソフトウェア間の調整を行う機能を持たせる構成がある．制御基盤ソフトウェアを配置する構成は，制御基盤ソフトウェアが担う処理の範囲に比例してシステム全体のオーバヘッドが大きくなるといった課題がある．一方で，システム制御ソフトウェアを改変する構成は，改変対象となるシステム制御ソフトウェアの機能に依存するため柔軟なシステムを構築できないといった課題がある．また，1 つの機器上のハードウェアを複数のシステム制御ソフトウェアから扱う方式として，VM 方式と LPAR 方式があるが，VM 方式はハードウェアの抽象化によるオーバヘッドが発生するといった課題がある．一方で，LPAR 方式は，ハードウェアを直接割り当てるためオーバヘッドは発生しないが，システム制御ソフトウェア間で干渉しないようにソフトウェアの改変が必要であるという課題や，共有するリソースの扱いが難しいといった課題がある．

以上の背景から，マルチコア上で複数のシステム制御ソフトウェアを動作させるためにシステム全体へ影響を与える命令のみをエミュレーションしつつ，プロセッサコア，メモリの割当ての管理とパーティショニング，そして共有リソース制御のためのインターフェースの提供を実現する，軽量な制御基盤ソフト「誉」を構築した．「誉」の実現により，複数のシステム制御ソフトウェアが占有リソースをオーバヘッドなしで操作することを可能にしつつ，インターフェースによる共有リソースの操作も可能となる．

以下，本論文では，2 章で既存の複数システム制御ソフトウェアの動作手法について述べ，3 章で提案手法について述べる．4 章で提案手法の実装について述べ，5 章では，機能検証について述べ．6 章で今後の課題について述べる．7 章で関連研究について述べ，8 章でまとめる．

2. 複数システム制御ソフトウェアの統合手法

本章では，1 つの機器上で複数のシステム制御ソフトウェアを制御する方法について述べ，組込み機器上で複数のシステム制御ソフトウェアを統合するための課題とそれを解決する提案手法の方針を述べる．

2.1 複数システム制御ソフトウェアの制御手法

1 つのコンピュータ，組込み機器で汎用 OS，リアルタイム OS などのシステム制御ソフトウェアを複数動作させるためには、起動するソフトウェアが互いに影響を及ぼさないためのソフトウェア構成を設計する必要がある．図 2 に複数システム制御ソフトウェア動作時のソフトウェアの構成を示す．現在，複数のシステム制御ソフトウェアを動作させるためのソフトウェア構成として，ハイパーバイザや HAL といった制御基盤ソフトウェアの上で各システム制御ソフトウェアを動作させる構成と，システム全体を制御するために改変したシステム制御ソフトウェアと他のシステム制御ソフトウェアを動作させる構成がある．制御基盤ソフトウェア上でシステム制御ソフトウェアを動作させる構成では，制御基盤ソフトウェアがシステム制御ソフトウェア間の調整を行う．そのため，制御基盤ソフトウェア上で動作するソフトウェアは，他のソフトウェアを意識することなく動作することができる．

一方で，改変したシステム制御ソフトウェアとその他のシステム制御ソフトウェアを動作させる構成では，動作させるソフトウェアの 1 つに改変を加えるため，制御基盤ソフトウェアを配置する方法に比べて使用するメモリリソースが少ない．これにより，組込み機器において限られたメモリ資源を節約することが可能である．

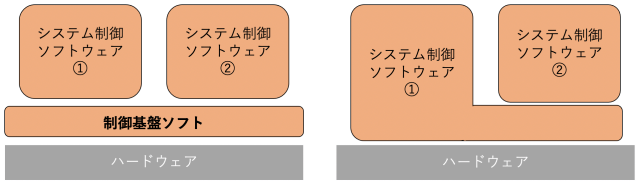

図 2 複数システム制御ソフトウェア動作時のソフトウェア構成

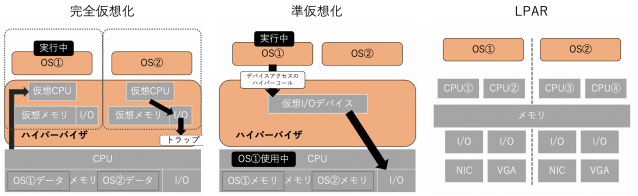

図 3 ハードウェアの制御方式

また，ソフトウェア構成の設計に加えて，機器が備えるハードウェアリソースへは，各システム制御ソフトウェアから自由にアクセスできるため，ハードウェアリソースの競合を防ぐための制御方法について検討する必要がある．図 3 に複数システム制御ソフトウェア動作時のハードウェアの制御方式を示す．複数システム制御ソフトウェア動作のためのハードウェアリソースの制御方式としては，ハードウェアを抽象化する VM 方式と，ハードウェアリソースを論理分割して割り当てる LPAR 方式がある．

VM 方式には，完全仮想化と準仮想化があり，どちらにおいても，ハードウェアリソースへのアクセスを制御するためのハイパーバイザをシステム制御ソフトウェアの下に配置して，システム制御ソフトウェアによるハードウェアアクセスを管理する．完全仮想化は，ハイパーバイザがハードウェアへのアクセスをトラップしてエミュレーション処理を行うため，ハイパーバイザ上で動作するシステム制御ソフトウェアは，自身が物理ハードウェアを制御しているかのように動作することが可能である．そのため，完全仮想化では，システム制御ソフトウェアを改変することなく動作させることができる．準仮想化は，仮想化環境で

の動作に特化するようにシステム制御ソフトウェアを改変して動作させる．そのため，システム制御ソフトウェアの改変コストはかかるが，完全仮想化に比べてハードウェアへのアクセスのオーバヘッドが小さくなる．

一方で，ハードウェアリソースを論理分割する LPAR 方式は，ハードウェアリソースを各システム制御ソフトウェアに排他的に割り当てて，複数のシステム制御ソフトウェアを動作させる．排他的にハードウェアリソースを割り当てることで，システム制御ソフトウェアが直接ハードウェアにアクセスできるため，オーバヘッドを発生させることなく動作させることが可能である．

2.2 複数組込み向けシステム制御ソフトウェア統合の課題

組込み機器において複数システム制御ソフトウェアを統合するためのソフトウェア構成として，制御基盤ソフトウェアを配置する構成では，システム制御ソフトウェアから制御基盤ソフトウェアを介した処理を行う度にオーバヘッドが発生する．そのため，制御基盤ソフトウェアが処理を担う範囲の広さに比例して，発生するオーバヘッドが増大するといった課題がある．一方で，動作させるシステム制御ソフトウェアの 1 つを改変する方法では，システム制御ソフトウェアを改変するためのコストがかかる．また，1 つのシステム制御ソフトウェアの機能に依存してしまうため柔軟なシステム設計が不可能になる．

複数のシステム制御ソフトウェア間でハードウェアリソースを扱う方式として，VM 方式は，仮想計算機の仲介処理によるコンテキストスイッチやハードウェアエミュレーションによりシステム制御ソフトウェアの処理に大きく関与してしまい，オーバヘッドが発生する．VM 方式を組込みソフトウェアに適用する場合，仮想ハードウェアを扱う処理のオーバヘッドによりリアルタイム性が損なわれるといった課題がある．特に組込み機器で問題となるのは，スケジューリングによる CPU 割り当ての変更により，システム制御ソフトウェアが意図しないタイミングで CPU の実行が停止してしまうことである．一方で，LPAR 方式はハードウェアリソースを論理分割して直接システム制御ソフトウェアにハードウェアリソースを割り当てるため，システム制御ソフトウェア間で共有する必要があるハードウェアリソースへのアクセスの競合が発生する．よって，共有するハードウェアリソースの扱いが困難であるといった課題がある．組込みシステムにおいて複数システム制御ソフトウェアを動作させるためには，これらの課題を解決する必要がある．

2.3 提案手法の方針

LPAR 方式では，ハードウェアを直接割り当てるため，システム制御ソフトウェアからのハードウェアアクセスに対してオーバヘッドは発生しない．また，CPU コアを複数搭載したマルチコア環境下では，各コアをソフトウェアに占有させることができるため CPU 切り替えによるオーバヘッドも発生しない．しかし，LPAR 方式で制御を行う場合，共有リソースの扱いが難しいといった課題が残る．そこで，処理のオーバヘッドを減らすために準仮想化方式で，共有リソースの制御を含む最小限のソフトウェア間の調整を行い，LPAR 方式でマルチコアプロセッサの各コアとシステム制御ソフトウェアが専有可能なハードウェアリソースを論理分割して直接割り当てることで，組込み機器において複数のシステム制御ソフトウェアを動作させる手法を実現する．提案手法の実現にあたり，ハードウェアの割り当て，共有するデバイスの競合制御やソフトウェア間の通信補助などの制御を行う軽量な制御基盤ソフトウェア「誉」(Hub-layer On Multicore-Architecture for Recent Embedded-system) を構築する．

3. マルチコア制御基盤「誉」

前章では，複数のシステム制御ソフトウェアを動作せるための手法について述べた．本章では，組込み機器において複数のシステム制御ソフトウェアを動作させるためにシステム制御ソフトウェア間の調整を行う制御基板ソフトウェア「誉」の構成，機能について述べる．

3.1 「誉」の構成

提案手法において，実現する「誉」の概要図を図 4 に示す．LPAR 方式をベースとして，「誉」上で動作させる各システム制御ソフトウェアに専有させる CPU コアとメモリ，専有可能なデバイスを直接割り当てる．システム制御ソフトウェアへのメモリの割当てと起動は「誉」が行い，ソフトウェア間で共有するハードウェアへのアクセスを「誉」の機能を利用するためのインターフェースへのアクセスに変更することで「誉」が共有ハードウェアへのアクセスを制御する．

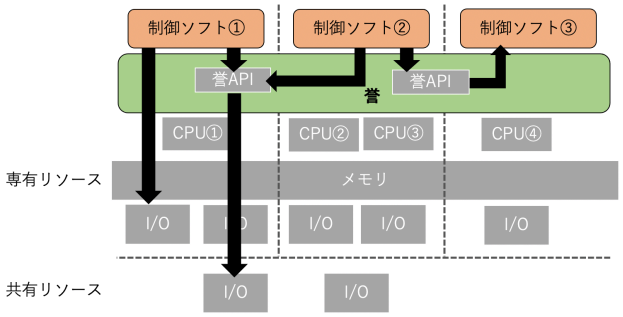

図 4 「誉」の構成

3.2 「誉」の機能

複数のシステム制御ソフトウェアを動かすためには，それぞれのソフトウェアにハードウェアリソースを割当てて

動作を開始させるための仕組みが必要である．1 つ目に起動するシステム制御ソフトウェアの機能を活用することで，複数のソフトウェアの動作を開始させる方法があるが，2 章で述べたとおり，最初に起動するシステム制御ソフトウェアの機能に依存してしまうため，「誉」上で動作させるシステム制御ソフトウェアを柔軟に選択することができない．そのため，「誉」には，前節で述べた構成を実現するための機能と，起動するソフトウェアの機能に頼らず，システム制御ソフトウェアにハードウェアリソースを与えて起動する機能が必要である．これらを踏まえて組込みシステム統合のためにシステム制御ソフトウェアを複数動作させる「誉」の実現に必要な機能を以下に示す．

(1) システム制御ソフトウェアをメモリに配置するためのローダ機能
(2) システム制御ソフトウェアを動作させるための起動エミュレーション機能
(3) システム制御ソフトウェアからの API を介した制御要求を処理する機能
(4) 「誉」が管理するシステム内のメモリ管理機能
(5) システム全体へ影響を及ぼす命令をトラップする機能
(6) トラップされた命令やアボートなどの例外を処理する機能

本節では，複数システム制御ソフトウェアを動作させる制御基盤「誉」の各機能について述べる．

ローダ機能

ローダ機能は，動作させるシステム制御ソフトウェアをメモリに配置するための機能である．2 次記憶媒体のファイルシステムのフォーマットを解析し，対象となるシステム制御ソフトウェアを任意のメモリアドレスに配置する．

起動エミュレーション機能

起動エミュレーション機能は，ローダ機能によってロードされたソフトウェアに CPU のコアを割り当てて起動するための機能である．起動されるソフトウェアは CPU，メモリ，デバイスの状態が電源を入れた直後の状態であることを想定して実行される．しかし，「誉」が各コアの初期設定を行うため，CPU のレジスタの値の変更，キャッシュの書き込みが行われており，電源投入直後の状態とは異なっている．そのため，「誉」からソフトウェアを起動する前にシステム制御ソフトウェアが想定しているハードウェアの状態に設定する．

共有リソース管理機能

共有リソース管理機能は，システム制御ソフトウェア間で共有デバイスの競合を防ぐために，共有デバイスへのアクセス処理，ソフトウェア間通信を行うためのインターフェースへのアクセスを処理する．

メモリの任意のアドレスへ配置された誉 API を「誉」上で動作するシステム制御ソフトウェアが呼び出すことで指定された API が実行される．

メモリ管理機能

メモリ管理機能は，「誉」が動作する組込みシステム内でメモリの使用状況を管理する．ローダ機能と連携して，システム制御ソフトウェアをメモリに配置する時に使用中の領域にかぶらないように，空きメモリを管理する．

命令トラップ機能

命令トラップ機能は，「誉」が動作するシステム全体への影響を与える処理を防ぐための機能である．図 5 にシステム全体へ影響を及ぼす状況を示す．1 つの制御ソフトでハードウェアリセットを必要とする障害が発生した際に，ハードウェアリセット命令が通ってしまうと，障害が発生したソフトウェアだけでなく，「誉」上で動作する他のソフトウェアへも影響を与えてしまう．これに対して，図 6 のように影響を及ぼす命令をトラップすることができれば例外処理として「誉」がハンドリングすることで，他のソフトウェアへの影響を抑えることができる．

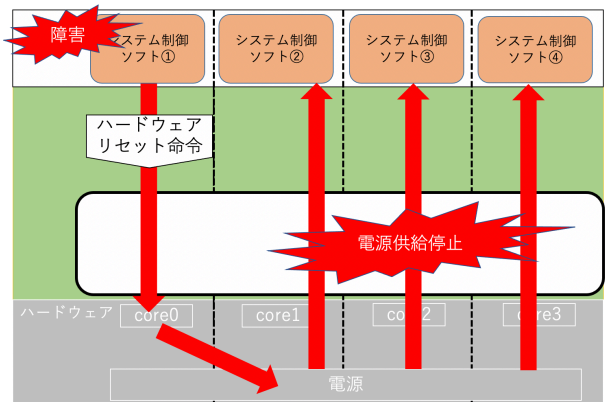

図 5　障害発生時の処理

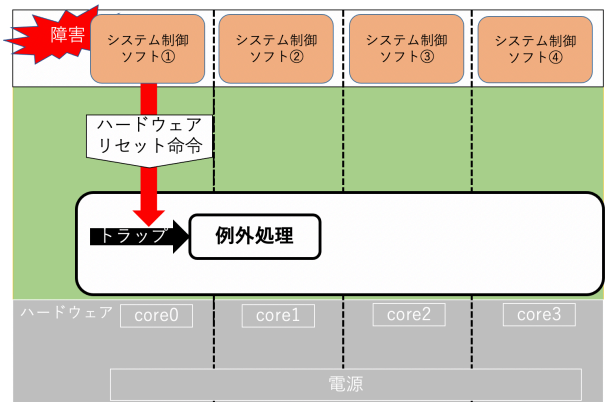

図 6　命令のトラップ

例外ハンドリング機能

例外ハンドリング機能は，トラップ機能により発生した例外と「誉」による制御中に発生したアボート例外に対処するための機能である．トラップする命令に，誉 API に置き換えることができずトラップしなければならないデバイスの処理がある場合，エミュレーション処理を行う．

4. 提案手法の実装

提案手法ソフトウェアは，組込みで広く使用される ARM プロセッサの Cortex-A シリーズを対象にして構築する．本論文では，表 1 に示す環境で実装を行なった．本章では，マルチコア制御基盤「誉」における，複数システム制御ソフトウェアを起動するための実装とソフトウェア間で共有するリソースを制御するための誉 API の実装について述べる．

表 1 動作環境

項目	内容
ボード	Raspberry Pi3 Model B
コア	Cortex-A53
アーキテクチャ	ARMv8 aarch32
メモリ	1GB

4.1 複数システム制御ソフトウェアの起動

複数のシステム制御ソフトウェアを動作させるためには，図 7 に示すように起動する各ソフトウェアを動作するメモリ領域が重ならないようにメモリに配置し，動作を開始するための CPU コアを割り当てる必要がある．また，CPU コアを割り当てる際に，CPU の状態や設定を起動するソフトウェアが想定する状態と設定に変更する必要がある．よって，マルチコア制御基盤「誉」に以下の実装を行う．

- 2 次記憶のドライバ
- ファイルシステムドライバ
- システム制御ソフトウェアにメモリを割り当てる処理
- CPU を割り当て前に初期化する処理
- CPU を起動するシステム制御ソフトウェアに割り当てる処理

2 次記憶のドライバとファイルシステムドライバは汎用的に使用できるものを構築しようとすると開発コストがかかる上に，ソフトウェアの規模が大きくなってしまう．開発環境の Raspberry Pi3 ModelB は，デフォルト設定では FAT32 でフォーマットされた SD カードから起動対象となるソフトウェアをメモリにロードして起動する．そこで，本論文では，開発環境の仕様に合わせて 2 次記憶として FAT32 でフォーマットされた SD カードの内容を読み込むために，SD カードのドライバと FAT32 ファイルシステムドライバの実装を行った．また，2 次記憶から複数のシステム制御ソフトウェアが使用するメモリ領域が重ならないように読み込むために，「誉」が指定したメモリ番地からシステム制御ソフトウェアを配置するための実装を行った．そして，ロードしたソフトウェアを電源投入直後の CPU の状態で起動するために，任意のコアの CPU のキャッシュ情報や，CPU の補助レジスタの内容を電源投入時の状態にしてから，割り当てるための実装を行い複数システム制御ソフトウェアを起動する機能を実現した．

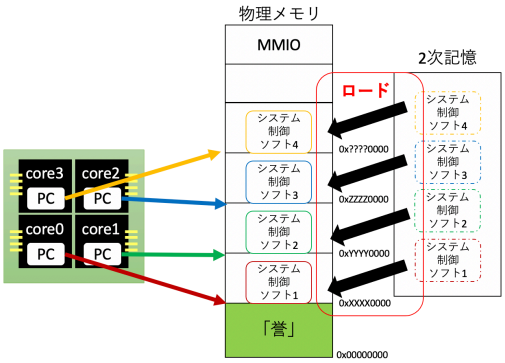

図 7 「誉」によるソフトウェアの起動

4.2 システム制御ソフトウェア間の制御を行う誉 API

提案手法では，ソフトウェア間で共有しなければならないリソースの制御とソフトウェア間で連携するための通信は，「誉」が提供するインターフェースを用いて行う．そこで，「誉」が提供する機能を組込みソフトウェアが利用するためのインターフェースとして、誉 API を実装した。システム制御ソフトウェア側からの API 呼び出しを容易にするため，引数によって実行する API を指定することができる入り口関数を実装し，図 8 に示すようにメモリのアドレス番地 0x22000000 に配置した．

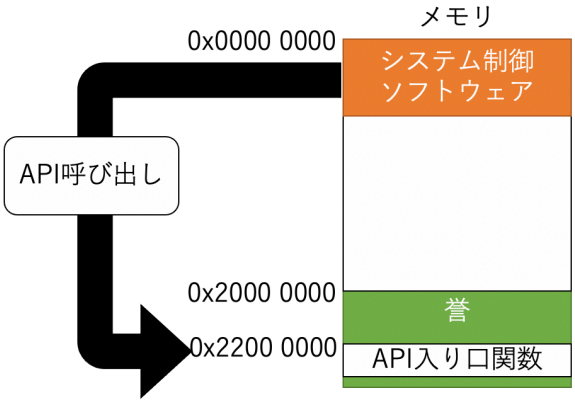

図 8 システム制御ソフトウェアからの API 呼び出し

API の入り口関数を配置する場所は，リンカスクリプトの最後に API の入り口関数を配置するためのセクションを用意し，セクション開始アドレスを固定することで指定した．図 9 にリンカスクリプトの記述内容を示す．

```
. = 0x22000000;
__homare_api_start = . ;
.homare_api : {*(homare_api)}
. = ALIGN(4);
__homare_api_end = . ;
```

図 9 リンカスクリプトの API セクション

これにより，システム制御ソフトウェアは，0x22000000

番地に配置された API の入り口関数を実行することで API を呼び出すことができる．システム制御ソフトウェアから API 呼び出しを行うために，システム制御ソフトウェア内に API を呼び出すためのアセンブリスタブを追加する．スタブから API を呼び出す際に第一引数として API の番号を設定することで呼び出す API の選択を行う．現在「誉」が提供可能な API を以下に示す．

- ソフトウェア間通信の送信用 API
- ソフトウェア間通信の受信用 API

誉 API によるソフトウェア間の通信を図 10 に示す．送信用 API は，mailbox へ通信内容の書き込みを行い通知を行う．受信用 API は，mailbox の通知内容を読み込むことで通信内容を確認する．

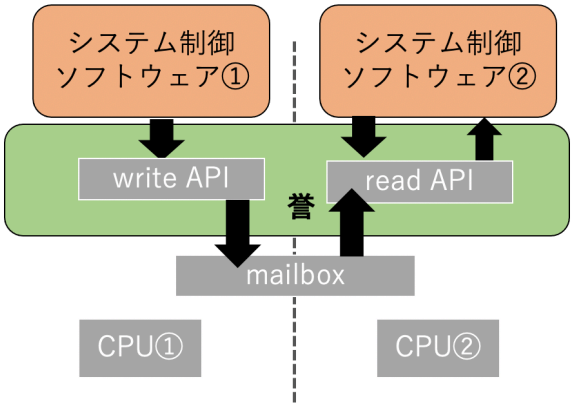

図 10　誉 API によるソフトウェア間通信

5. 動作検証

本章では，マルチコア制御基盤「誉」で行った動作検証について述べる．

5.1 検証内容

「誉」上で既存のシステム制御ソフトウェアを複数起動し，誉 API を利用したソフトウェア間の通信を行う．この検証の目的は，以下の通りである．

- 「誉」の設計に基づいて，システム制御ソフトウェアが複数同時に動作可能であることを確認する．
- 「誉」が提供する API による通信制御が可能であることを確認する．

検証用のシステム制御ソフトウェアとして，FreeRTOS と TOPPERS/ASP3 を使用し，図 11 に示す構成で市販のラジコンカーに Raspberry Pi3 Model B および Bluetooth モジュールを搭載して動作検証を行った．後輪を駆動する DC モータと前輪のステアリングを駆動するサーボモータを TOPPERS/ASP3 が GPIO を介して制御するとともに，制御内容を誉 API を用いて FreeRTOS へ送信する．一方で，FreeRTOS は制御内容を受け取り，Bluetooth を用いて観測用 PC に送信する．Bluetooth のプロファイルには SPP（Serial Port Profile）を使用し，UART を経由した無線データ通信を行う．観測用 PC には受信した制御内容をコンソールに出力するプログラムを用意した．

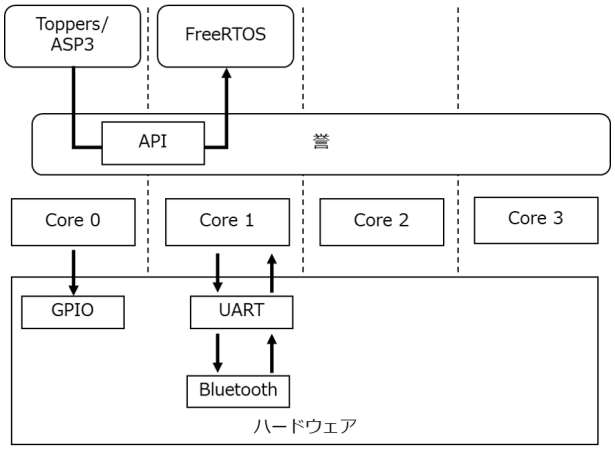

図 11　検証時の構成

5.2 動作検証結果

図 12 に，検証時のラジコンカーおよび観測用 PC の挙動を示す．右側の観測用 PC に制御内容が出力された．このことから，「誉」上では複数のシステム制御ソフトウェアが動作し，誉 API のソフトウェア間通信によりラジコンカーの制御内容を観測用 PC に送信できていることが確認できた．

図 12　検証時のラジコンカーと観測用 PC

6. 今後の課題

6.1 トラップ機能と例外処理機能の実装

提案した「誉」の機能の内トラップ機能と例外処理機能の詳細設計と実装の部分がまだであるため，これらの設計と実装を行う必要がある．トラップ機能を実現する方法と

して，ハードウェアの機能を利用する方法がある，現在開発環境で使用している ARM プロセッサの拡張機能には，セキュリティ拡張機能と仮想化拡張機能がある．どちらの拡張機能も，システム制御ソフトウェアへ送信される特定のアドレスへのアクセス，割込みアボートをトラップするための機能がある．しかし，どちらの機能も MMU による仮想アドレス変換を利用しなければならず，処理が複雑になってしまうため，ハードウェアの機能に頼らない方法についても検討する必要がある．

また，例外処理機能では，システム全体に影響を与えてしまうが，命令の発行元のシステム制御ソフトウェアでは実行する必要があるものを処理しなくてはならない．そのため，軽量な制御基盤であるという特性を失わずに，3 章で述べたハードウェアリセット以外に起こりうる問題に対しても幅広く対応できる設計にする必要がある．

6.2 リアルタイム性の確保

「誉」は，組込みシステム制御ソフトウェアを統合する制御基盤であり，リアルタイム性を考慮する必要がある．そのため，「誉」は専有可能なハードウェアリソースを直接各システム制御ソフトウェアに割り当てる一方で，共有リソースは制御 API を介して操作する．現在，誉 API には制御要求に対して実行するための優先順位をつける仕組みや終了するまでの時間のルールがない．そのため，今後動作させるシステム制御ソフトウェアとハードウェアリソースの状況によって実行する優先度を変更できる仕組みが必要となる．

7. 関連研究

複数のシステムソフトウェアを動作させる研究は，既存研究においても行われている．1 つのコンピュータで複数のシステムを動作させる研究として Xen[4]，SPUMONE[5]，DARMA[6][7][8]，SafeG[9] がある．本章では，関連研究について述べる．

7.1 Xen

Xen は，ゲスト OS に対して Xen 上で動作するための改変を施し，CPU，メモリ，デバイスといったハードウェアの仮想化を行う．電源投入後最初起動するドメイン 0 と呼ばれるゲスト OS が，ドメイン U と呼ばれる他のゲスト OS のデバイスへのアクセスを受け取りエミュレーション処理を行う．具体的には，ドメイン U がデバイスへのアクセスを行う場合，ドメイン U からドメイン 0 へハイパーコールと呼ばれる命令を発行し，デバイスへのアクセスと処理を依頼する．ハイパーコールを受け取ったドメイン 0 は，ドメイン 0 のデバイスドライバを用いてドメイン U から受け取った処理を行う．

ハードウェアへのアクセスをドメイン 0 が管理することにより，ゲスト OS 間のリソースの保護が実現される．

Xen は，メモリの仮想化，デバイスのエミュレーション処理を行う仮想化を前提にしており，動作させるゲスト OS に大きく関与する．仮想化して動作をエミュレーション範囲が大きすぎるため，オーバヘッドが生じるというデメリットがあるまた，Xen は，リソースを潤沢に確保できるデスクトップコンピュータやサーバで使用する目的で設計されているため，規模が大きくメモリリソースを大量に消費するという問題がある．

7.2 SPUMONE

SPUMONE は，RTOS と汎用 OS を 1 つのデバイスで動作させるための，組込み向けに提案された CPU のみを仮想化する軽量な VMM である．SPUMONE は，各 OS が，デバイスを直接操作できるようになっており，ゲスト OS への干渉を最小限に抑えている．2 つの OS を動作させた際に，一方の OS が障害発生時に障害から復旧するために再起動を行う場合がある．1 つの OS が他の OS に影響を与えないように再起動をする必要があることから，仮想 CPU を 1 つずつ各 OS に占有させて，1 つの OS で再起動する必要がある場合は，仮想 CPU を初期化することで他の OS に影響を与えずに再起動する．また，OS 間で通信を行うためのインタフェースを複数用意しており，仮想 CPU 間の割込みを利用する方法と共有メモリを用いる方法で実現している．CPU のみを仮想化することにで，仮想化によるオーバヘッドを最小限に抑えているが，仮想 CPU の割り当てスケジューリングによる切替え時に CPU の実行が停止してしまいオーバヘッドが発生する．

7.3 DARMA

DARMA は，ナノカーネルによって，OS を複数動作させることを目的としており，ナノカーネルが，シングルプロセッサを時間分割して複数 OS を切り替える．割込みが発生した場合は，ナノカーネルが割込みを横取りし，各 OS に振り分けて通知する．DARMA では，最初にホスト OS が起動し，ホスト OS のデバイスドライバをロードする機能によりナノカーネルがメモリ上にロードされる．その後，ゲスト OS はナノカーネルと同様にホスト OS デバイスドライバをロードする機能を用いてメモリ上にロードされる．ホスト OS とゲスト OS でのデバイスの競合を防ぐために，デバイスを排他的に割り当てる．各 OS が直接デバイスを操作することを許しており仮想化を行わないため，オーバヘッドが最小限に抑えられている．しかし，ナノカーネル上で動作させるゲスト OS は，最初に起動したホスト OS の機能により，上位のメモリ番地にロードされるためソフトウェアの改変が必要である．また，ナノカーネルとゲスト OS のメモリ上への配置は，ホスト OS の機能に頼るため，ホスト OS の機能に依存するといった課題がある．さ

らに，プロセッサを時間分割するため，プロセッサの割り当てスケジューリングによるオーバヘッドが発生する．

7.4 SafeG

SafeG は，ARM プロセッサのセキュリティ拡張機能における TrustZone を利用したデュアルシステムのための組込み向けハイパーバイザである．TrustZone により領域をセキュアワールドとノーマルワールドに分け，セキュアワールドで制御用の OS，ノーマルワールドで汎用 OS を動作させるように設計されている．セキュリティ拡張機能によりノーマルワールドからセキュアワールドへのアクセスは禁止されているため，汎用 OS が攻撃されたとしてもその影響は制御用 OS へ行かないようになっている．SafeG は，制御 OS と汎用 OS の 2 つを動作ささせることを目的として作られているため，多数のソフトウェアを集約して動作させることを目的とはしていない．

8. おわりに

本論文では，複数の組込みシステムをマルチコアで制御するための制御基板「誉」について述べた．提案した手法では，LPAR 方式と準仮想化方式の利点をそれぞれ取り入れマルチコア環境において物理コアを割り当てることにより，複数システムを動作させる．システム制御ソフトウェアから複数のシステム制御ソフトウェア間で共有するリソースを操作するための API，ソフトウェア間の通信を行うための API をそれぞれ呼び出すことでシステム制御ソフトウェア間の連携を可能とする．動作確認の結果，既存の異種の組込みソフトウェアを動作させて，誉 API によるソフトウェア間の通信が行えていることが確認できた．

今後は，「誉」に必要となる残り機能を実現するための設計についての検討と実装を行う．さらに，現在「誉」API 利用時のリアルタイム性について十分に考慮出来ていないため，誉 API 使用次のリアルタイム性の確保に向けた設計と実装を行う．

参考文献

[1] Inc. or its affiliates Amazon Web Services. FreeRTOS - Market leading RTOS (Real Time Operating System) for embedded systems with Internet of Things extensions. `https://www.freertos.org/`.
[2] Arm Limited (or its affiliates). Mbed OS — Mbed. `https://www.mbed.com/en/platform/mbed-os/`.
[3] TOPPERS Project Inc. TOPPERS プロジェクト／ ASP3. `https://www.toppers.jp/asp3-kernel.html`.
[4] Barham, P.and Dragovic, B.and Fraser, K.and Hand, S.and Harris, T.and Ho, A.and Neugebauer, R.and Pratt, I., Warfield, and A. Xen and the art of virtualization. In *In Proceedings of the nineteenth ACM symposium on Operating systems prin- ciples*, pp. 164–177, 2003.
[5] W. Kanda, Y. Yumura, Y. Kinebuchi, K. Makijima, and T. Nakajima. Spumone: Lightweight cpu virtualization layer for embedded systems. In *2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, Dec 2008.
[6] 佐藤雅英, 関口知紀, 新井利明, 井上太郎, 宮崎義弘, 中橋晃文, 梅都利和. ナノカーネル方式による異種 os 共存 技術「darma」 の実装. 全国大会講演論文集, p. 59, 1999.
[7] 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, 1999.
[8] 加藤直, 齋藤雅彦, 大野洋, 中村智明, 上脇正, 井上太郎. 組込み向けデュアル os 実行システム darma の開発. 全国大会講演論文集, 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, 情報・システム, Vol. J93-D, No. 2, pp. 75–85, feb 2010.