Bluetoothを用いたマルチVM対応 mrubyバイトコードローダ

山本 拓朗[1]　大山 博司[2]　安積 卓也[3]

概要：近年，組込みシステムは複雑化・大規模化しているため，組込みソフトウェアの生産性が問題になっている．組込みソフトウェア開発の生産性の向上を目的として，mruby（軽量 Ruby）を適用させたコンポーネントベース開発が可能なフレームワークである mruby on TECS を提案してきた．現状の mruby on TECS では，プラットフォームに mruby バイトコードを組み込んでいるため，mruby プログラムを修正する度にコンパイル・リンクし直す必要がある．さらに，マルチ VM を提供しているが，複数の mruby プログラムを効率よく並行動作させるには開発者がリアルタイム OS の機能を熟知している必要がある．本研究では，mruby on TECS の拡張として，mruby アプリケーションのバイトコードを Bluetooth で転送することで開発効率を向上させる．さらに，複数の mruby プログラムを協調動作できるフレームワークを提案する．

mruby Bytecode Loader Using Bluetooth in Multi-VM Environment

Takuro Yamamoto[1]　Hiroshi Oyama[2]　Takuya Azumi[3]

Abstract: In recent years, the productivity of embedded systems has become a problem due to their complexity and large-scale. For the purpose of improving the productivity for embedded software development, the mruby on TECS framework has been proposed that is applied mruby (Lightweight Ruby) and supports component-based development. In the current mruby on TECS, the mruby programs have to be compiled and linked every time the programs are modified because the mruby bytecodes are incorporated in the platform. Moreover, while the framework supports multi-VM, developers need to be familiar with the functions of RTOSs to effectively execute multiple mruby programs in concurrent or/and parallel. To improve the development efficiency, this paper proposes an mruby bytecode loader using Bluetooth as an extension of mruby on TECS. In addition, multiple mruby programs cooperatively run in the proposed framework. A RiteVM sceduler makes multitasking processing more easy-to-use than that of mruby on TECS.

1. はじめに

近年，組込みシステムは高品質・高性能化に伴い，複雑化・大規模化している上に，製品の低コスト化や短期間での開発も要求されている．

ソフトウェアを効率的に開発するアプローチのひとつに，コンポーネントベース開発がある．コンポーネント化されたソフトウェアは，再利用性が高く，検証が容易になるため，コンポーネントベース開発を適用することで，複雑かつ大規模なソフトウェアを効率的に開発することができる．システムの拡張や仕様変更にも柔軟に対応できる．組込みシステムのコンポーネント技術として，TECS [1] や AUTOSAR [2] などがある．

効率的なソフトウェア開発のもう一つのアプローチとして，スクリプト言語がある．現在のソフトウェアのほとんどは C 言語で開発されているが，C 言語による開発はコストが高く，開発期間も長くなる．スクリプト言語は，その使いやすさから高い生産性をもっているため，効率的かつ短期間での開発ができる．有名なスクリプト言語として，

[1] 大阪大学基礎工学部
School of Engineering Science, Osaka University
[2] オークマ株式会社
OKUMA Corporation
[3] 大阪大学基礎工学研究科
Graduate School of Engineering Science, Osaka University

Ruby, Perl，Python，Lua などがある．

しかし，スクリプト言語は可読性や使いやすさが高い反面，C 言語よりも実行時間が遅い．組込みシステムにとって，最悪実行時間や応答時間といったリアルタイム性は非常に重要であるため，スクリプト言語を組込みシステムに利用することは難しい．

mruby on TECS は，mruby(軽量 Ruby) [13] と，組込みシステムに適したコンポーネントシステムである TECS (TOPPERS Embedded Component System) を組み合わせたフレームワークである [3]．mruby on TECS では，mruby プログラムから C 言語の関数を呼ぶ機能を提供しており，mruby に比べて，アプリケーションを約 100 倍速く実行できる．

現状の mruby on TECS では，いくつかの問題がある．mruby on TECS は，mruby プログラムをロードするための方法として，記憶装置しか対応していないため，作業効率が悪い．(LEGO MINDSTORMS EV3 のプラットフォーム [4]) mruby プログラムを修正する度に，SD カードの抜き差しや ROM への書き込みを行う必要がある．さらに，mruby on TECS はマルチ VM に対応してるが，複数の mruby アプリケーションを並行実行させる場合，開発者がタスクを待ち状態へ遷移させる OS の機能を呼び出さなければならない．

提案フレームワークは，mruby on TECS を拡張して，Bluetooth を用いた mruby バイトコードローダと，実用的なマルチタスク処理の実装を行った．提案フレームワークでは，プラットフォーム部分をコンパイル・リンクし，ターゲットデバイス上で起動する．ホスト PC 側では，mruby アプリケーション (.rb) をバイトコード (.mrb) にコンパイルし，Bluetooth を通してターゲットデバイスにバイトコードを転送する．ターゲットデバイス側では，RiteVM に実装されたローダが転送されたバイトコードを受信し，すでにプラットフォームに含まれている mruby ライブラリと合わせてアプリケーションを実行する．これによって，繰り返し SD カードの抜き差しや ROM への書き込みをする手間や OS を再起動する時間が省けるため，作業効率を上げることができる．さらに，各 VM の処理を平等に実行する RiteVM スケジューラを提供することで，マルチタスク処理を効率的に利用できる．

本論文における貢献は次のとおりである．

- mruby バイトコードローダの実装により，mruby on TECS を使ったソフトウェア開発における作業効率を向上させる．
- RiteVM スケジューラの実装により，複数の mruby プログラムを効率的に並行・並列動作させる．
- コンポーネントベース開発の利点を明らかにする．

本論文の構成を次に述べる．まず 2 章で，提案フレームワークのベースである mruby on TECS について述べる．

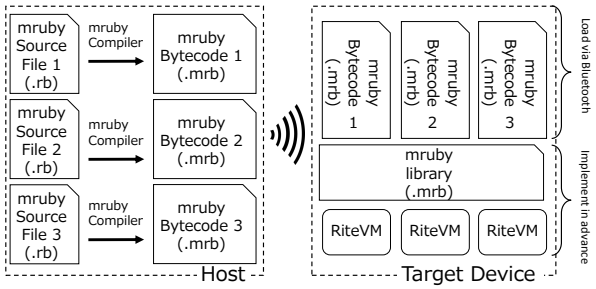

図 1 システムモデル

3 章では，提案フレームワークの設計と実装について述べる．4 章では，ローダを使用した場合の作業効率の比較，RiteVM を使用した場合の実行時間のオーバヘッドを評価する．5 章では，関連研究について述べる．最後に，6 章で本論文をまとめる．

2. 既存フレームワーク

図 1 に提案フレームワークのシステムモデルを示す．RiteVM と mruby ライブラリを含むプラットフォーム部分は，はじめにターゲットデバイス上で起動されている．mruby アプリケーションのバイトコードは，ホストからターゲットデバイスへと転送される．それぞれバイトコードは，RiteVM に割り当てられ，並行して起動される．

この章では，提案フレームワークのベースになっている mruby on TECS について述べる．mruby on TECS で利用されている mruby と TECS についても述べる．

2.1 mruby

mruby は，Ruby の可読性や使いやすさはそのままに，リソースが少なく起動が速いため，組込みシステムに適している．mruby には，RiteVM と呼ばれる VM が実装されているため，どの OS 上でもアプリケーションを起動させることができる．mruby コンパイラは，mruby プログラム (.rb) を，中間言語であるバイトコード (.mrb) にコンパイルする．RiteVM が実装されていれば，どのターゲットデバイス上でも同じ mruby プログラムを実行できる．

2.2 TECS

TECS (TOPPERS Embedded Component System) は，組込みシステム向けのコンポーネントシステムである．TECS によるコンポーネントベース開発は，システム全体の構造を可視化するため，システムの複雑さや難しさを減らすことができる．さらに，ソフトウェアの共通部分はコンポーネントとして扱われるため，開発の重複を減らし，生産性を向上させる．

TECS でのコンポーネントの生成と結合は静的に行われるため，最適化され，実行時間や消費メモリのオーバヘッ

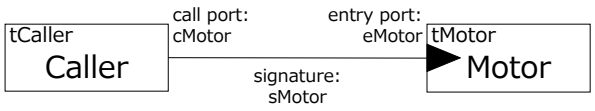

図 2　TECS コンポーネント図の例

```
1  signature sMotor{
2      int32_t getCounts( void );
3      ER resetCounts( void );
4      ER setPower( [in]int power );
5      ER stop( [in] bool_t brake );
6      ER rotate( [in] int degrees, [in] uint32_t speed_abs,
7                 [in] bool_t blocking );
8      void initializePort( [in]int32_t type );
9  };
```

図 3　シグニチャ記述

ドは少ない．他の特徴として，C 言語での実装，ソースレベルでの移植性，コンポーネントの粒度が小さいことなどがある．

2.2.1　コンポーネントモデル

図 2 にコンポーネント図の例を示す．TECS では，インスタンス化されたコンポーネントはセル (*cell*) と呼ばれ，受け口 (*entry*)，呼び口 (*call*)，属性，内部変数を持つ．受け口は自身の機能を提供するインタフェースで，呼び口は他のセルの機能を利用するためのインタフェースである．セルは複数の受け口や呼び口を持つことができる．セルの提供する関数は，C 言語で実装される．

受け口と呼び口の型は，セルの機能を使うためのインタフェースであるシグニチャによって定義される．セルの呼び口は，同じシグニチャを持つ他のセルの受け口と結合できる．

セルの型は，セルタイプと呼ばれ，受け口，呼び口，属性，内部変数の組を定義している．

2.2.2　コンポーネント記述

TECS のコンポーネント記述は，シグニチャ記述，セルタイプ記述，組上げ記述の 3 つに分類され，.cdl ファイルに記述する．図 2 に示されるコンポーネント記述について次に述べる．シグニチャ記述は，セルのインタフェースを定義する．*signature* キーワードに続けて，シグニチャ名 (接頭辞 “s”) を記述する．図 3 のようにシグニチャ sMotor を定義することができる．TECS では，インタフェースの定義を明確にするために，入力と出力にはそれぞれ，[in] と [out] という指定子がつけられる．

セルタイプ記述は，受け口，呼び口，属性，内部変数を用いてセルタイプを定義する．*celltype* キーワードに続けて，セルタイプ名 (接頭辞 “t”) を記述する．図 4 に示すように，受け口は，*entry* キーワードに続けて，シグニチャ名, 受け口名を記述する．同様にして，呼び口も定義できる．属性と変数は，それぞれ *attr*，*var* キーワードを用いて列挙する．

```
1  celltype tCaller{
2      call sMotor cMotor;
3  };
4  celltype tMotor{
5      entry sMotor eMotor;
6      attr{ int32_t attr = 100; };
7      var{ int32_t var; };
8  };
```

図 4　セルタイプ記述

```
1  cell tMotor Motor{
2  };
3  cell tCaller Caller{
4      cMotor = Motor.eMotor;
5  };
```

図 5　組上げ記述

組上げ記述は，セルをインスタンス化し，セル同士を結合する．*cell* キーワードに続けて，セルタイプ名，セル名を記述する．呼び口名，“=”，結合先の受け口名の順に記述し，セル同士を結合する．図 5 の例では，セル Caller の呼び口 cMotor と，セル Motor の受け口 eMotor が接続されている．

2.3　mruby on TECS

mruby on TECS は，mruby(軽量 Ruby) を用いた組込みソフトウェアのコンポーネントベース開発が可能なフレームワークである．

2.3.1　システムモデル

mruby バイトコードはそれぞれ，コンポーネント化された RTOS のタスクとして，RiteVM 上で実行される．TECS のコンポーネントは，モータドライバやセンサドライバのような組込みドライバをサポートしている．

mruby-TECS ブリッジによって，mruby プログラムから C 言語の関数を呼び出すことができる．mruby-TECS ブリッジについては，以下で詳しく説明する．

本研究では，RTOS として，TOPPERS/HRP2 [5] を使用した．TOPPERS/HRP2 は，μ ITRON [6] をベースにしたメモリ保護機能を持つ RTOS である．しかし，TECS は, TOPPERS/HRP2 だけでなく, OSEK [7] や TOPPERS/ASP [8] といった RTOS にも対応しているので，mruby on TECS は RTOS に依存しない．

3.　設計と実装

図 6 に提案フレームワークの詳細なシステムモデルを示す．ホストから転送された mruby アプリケーションのバイトコードは，それぞれの RiteVM に実装されたローダで受信され，同期処理により同時に実行される．RiteVM スケジューラがタスクを周期的に切り替えるため，mruby アプリケーションは並行動作することができる．

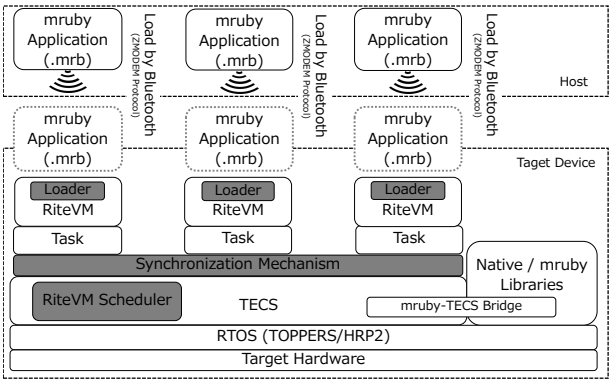

図 6 提案フレームワークの詳細なシステムモデル

3.1 Bluetooth を用いた mruby バイトコードローダ

この章では，Bluetooth を用いた mruby バイトコードローダについて述べる．現状の mruby on TECS では，mruby バイトコードをプラットフォームに組み込んでいるため，mruby プログラムを修正する度にプラットフォーム部分をもう一度コンパイル・リンクし，SD カードや ROM といったデバイスに再度書き込み，OS を再起動する必要がある．このような作業を繰り返すことで，作業効率は悪くなる．提案フレームワークでは，プラットフォーム部分をコンパイル・リンクし記憶デバイスに書き込むのは，はじめの一度だけで良いため，作業効率を向上させることができる．

mruby プログラムは，アプリケーション部分とライブラリ部分に分けられる．mruby アプリケーションは，開発者が主にプログラムするメインのコードであり，mruby ライブラリには，motor クラスや sensor クラスといった Ruby クラスのようにアプリケーションで利用される関数群が定義されている．アプリケーションとライブラリを含んだ mruby バイトコードを転送し，実行することもできるが，提案フレームワークでは，ライブラリ部分はプラットフォームに組み込み，アプリケーション部分のみを転送する設計にした．この設計にすることで，転送するバイトコードのサイズや処理時間の無駄を省くことができる．さらに，各 RiteVM でライブラリを共有することや，RiteVM 固有のライブラリを持つことも可能になる．

提案フレームワークでは，mruby バイトコードの連続ローディングにも対応している．

3.1.1 ローダを実装した RiteVM のコンポーネント

提案フレームワークでは，mruby バイトコードローダは TECS のコンポーネントとして提供されている．このコンポーネントは，RiteVM のコンポーネントの拡張として実装されている．(詳細については [3] を参照) このコンポーネントでは，転送されたバイトコードの受信に加え，RiteVM のコンフィグレーションが行われる．

図 7 に，MrubyTask1 と MrubyBluetooth1 のコンポーネントの例を示す．MrubyTask1 は，コンポーネント化された

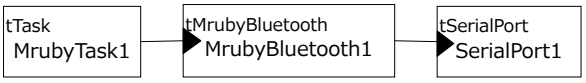

図 7 ローダを実装した RiteVM のコンポーネント図

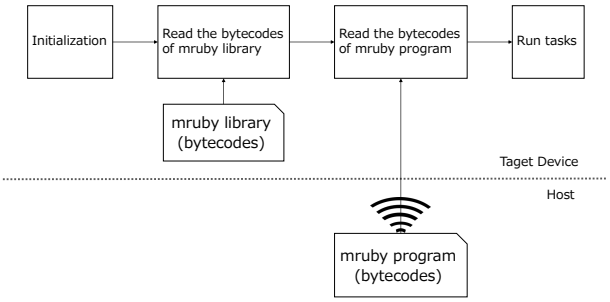

図 8 ローダの処理フローチャート

RTOS (TOPPERS/HRP2 [**?**]) のタスクである．Mruby-Bluetooth1 は，mruby バイトコードを実装した RiteVM のコンポーネントであり，このコンポーネントのはじめで転送されてきたバイトコードを受信する．提案フレームワークでは，バイナリ転送プロトコルとして，ZMODE を適用した．

図 8 に，ローダを実装した RiteVM が mruby プログラムを実行するまでの制御フロー，図 10 には，RiteVM コンポーネントのソースコードを示す．

はじめに，*mrb_state* と *mrbc_context* のポインタを初期化する．*mrb_state* は，mruby で使われる状態と変数のセットである．

次に，mruby ライブラリのバイトコードを読み込む．図 9 に示すように，tMrubyBluetooth のセルは属性を持っている．*mrubyFile* は mruby ライブラリのファイルを示しており，*[omit]* は TECS ジェネレータによってのみ使えわれるため，この属性はメモリを消費することない．*irep* は，mruby ライブラリのバイトコードが格納されている配列へのポインタである．つまり，mruby ライブラリは始めのコンパイル時にコンポーネントの属性として，配列に保存されている．

次に，転送されてきた mruby アプリケーションのバイトコードを読み込む．mruby アプリケーションのバイトコードは，図 9 に示す，内部変数 uint8_t 型の配列 *irepApp* に格納されている．２つのバイトコードは，それぞれ別の配列に保存されており，別々に読み込まれる．

最後に，RiteVM は mruby アプリケーションであるタスクを実行する．mruby アプリケーションのプログラムが修正された場合は，そのバイトコードのみを再転送する．

```
1  celltype tMrubyBluetooth{
2    entry sTaskBody eMrubyBody;
3    attr{
4      [omit]char_t *mrubyFile;
5      char_t *irep = C_EXP("&$cell_global$_irep");
6      uint32_t irepAppSize = C_EXP( BUFFER_SIZE );
7    };
8    var{ [size_is(irepAppSize)] uint8_t *irepApp; };
9  };
```

図 9 ローダを実装した RiteVM のセルタイプ記述

```
1  void eMrubyBody_main( CELLIDX idx ){
2    /*Omit: Declaration variables*/
3    mrb=mrb_open(); /*New interpreter instance*/
4    /*Omit: error check for mrb_state*/
5    c=mrbc_context_new( mrb ); /*New mruby context*/
6    /*Omit: initialization of mruby−TECS bridge*/
7    /*Receive the bytecode via Bluetooth*/
8    bluetooth_loader( VAR_irepApp );
9    /*Load mruby library bytecode and run*/
10   mrb_load_irep_cxt( mrb, ATTR_irep, c );
11   /*Load mruby application bytecode and run*/
12   mrb_load_irep_cxt( mrb, VAR_irepApp, c );
13   mrbc_context_free( mrb, c ); /*Free mruby context*/
14   mrb_close( mrb ); /*Free interpreter instance*/
15 }
```

図 10 ローダを実装した RiteVM のメインコード

3.2 マルチタスク処理

この章では，提案フレームワークでのマルチタスク処理の設計について述べる．

マルチタスク処理のアプローチのひとつにコルーチンがある．コルーチンは，協調的スレッドであり，開発者が *resume* や *yield* といった関数を呼び出すことで並列動作を行うことができる．(Ruby のコルーチンは，Fiber クラス [9] に定義されている．) コルーチンは，ノンプリエンプティブな処理で，開発者自身がタスクの切り替えを行う必要があるので，OS のサポートやマルチコアの恩恵を受けることができない．

その他のアプローチとして，μ ITRON のサービスコールである *delay()* 使った手法がある．このサービスコールは，引数として与えられた時間だけ，そのタスクの起動を遅らせる．*delay()* は，固定優先度スケジューリングの場合に用いられるが，フェアスケジューリングの場合には使用することが難しい．

提案フレームワークでは，フェアスケジューラである RiteVM スケジューラを実装し，複数のタスクを平等に並行動作させる．このスケジューラはアプリケーションタスクが同じ優先度を持つ場合に利用でき，開発者が OS の関数を呼び出すことなく，マルチタスク処理が可能になる．その上，既存のアプリケーションプログラムの構造を変えること使うことができる．

3.2.1 RiteVM スケジューラ

RiteVM スケジューラは，周期ハンドラであり，μ ITRON のサービスコールである *rotateReadyQueue* を呼び出す．

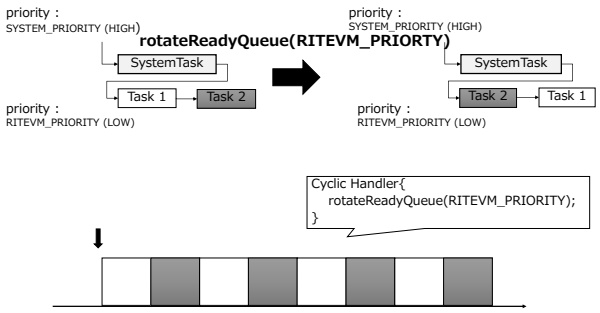

図 11 RiteVM スケジューラの設計

rotateReadyQueue は，同優先度のタスクの実行を切り替える関数である．RiteVM スケジューラの設計を図 11 に示す．

無限ループを持った同優先度の 2 つのタスクの場合を考える．現状のシステムでは，始めにタスクが起動すると，そのタスクが無限ループに入るため，もう片方のタスクが起動することができない．

rotateReadyQueue が呼ばれると，図 11 に示すように，タスクの起動が切り替わる．*rotateReadyQueue* の引数は，優先度である．

rotateReadyQueue は，2 つ以上のタスクの場合にでも利用できる．例えば，3 つのタスク，task 1, 2, 3 が順に起動する場合，*rotateReadyQueue* が呼ばれると，task 2, 3, 1 の順にタスクの起動は切り替わる．

3.2.2 RiteVM スケジューラのコンポーネント

RiteVM スケジューラのコンポーネントは，CyclicHandler と CyclicMain から構成される．CyclicHandler セルは，μ ITRON の周期ハンドラの設定を行う．μ ITRON の周期ハンドラは，[6] に詳しく書かれている．

周期ハンドラは, 5 つの引数を持っている．(ID, attribute, cyclic time, cyclic phase, access pattern) CyclicHandler セルは，3 つの引数をセルの属性として持っている．CyclicMain セルは，周期ハンドラの処理を行うコンポーネントであり，*rotateReadyQueue* が実装されている．*call* ポートは，カーネルの機能である *rotateReadyQueue* を使用するためにカーネルセルの *entry* ポート (*tkernel.eiKernel*) に接続されている．属性は，*rotateReadyQueue* の引数として使われる．

図 12 に，RiteVM スケジューラのコンポーネント組上げ記述を示す．

CyclicHandler セル部分では，cyclicTime や cyclicPhase といった属性で与えられる周期ハンドラの設定を行う．attribute は *TA_STA* であるので，OS が起動すると周期ハンドラが実行状態となる．周期時間は，1msec である．CyclicMain セル部分には，属性として priority が記述されている．RITEVM_PRIORITY は mruby タスクの優先度として定義されている．この属性は，*rotateReadyQueue* の

```
1  cell tCyclicHandler CyclicHandler{
2      ciBody = CyclicMain.eiBody;
3      attribute = C_EXP("TA_STA");
4      cyclicTime = 1;
5      cyclicPhase = 1;
6  };
7  cell tCyclicMain CyclicMain{
8      priority = C_EXP("RITEVM_PRIORITY");
9  };
```

図 12 RiteVM スケジューラの組上げ記述

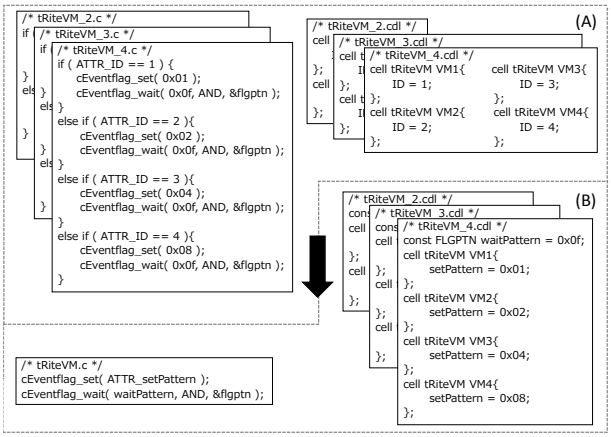

図 13 TECS を用いたイベントフラグの設計 *差分のみ示す

引数として与えられる．

3.2.3 mruby アプリケーションの同期機構

複数の mruby アプリケーションの起動を同期させるために，提案フレームワークでは，同期処理の手法としてイベントフラグを適用した．タスクはそれぞれ 0x01(01), 0x02(10) のパターンをセットし，待ちパターン 0x3(11) で AND 待ちする．この機構は，2 つ以上のタスクの場合でも適用できる．例えば，タスクが 4 つの場合，タスクはそれぞれ，0x01(0001), 0x02(0010), 0x04 (0100), 0x08 (1000) のパターンをセットし，待ちパターン 0x0f(1111) で AND 待ちする．(図 13 (A))

さらに，提案フレームワークでは，バイトコードの連続ローディングに対応するため，アプリケーションの終了も同期を行う．これによって，すぐに終了するようなアプリケーションの RiteVM が，次のロードの待機状態に入るのを防ぐことができる．すべてのアプリケーションは同時に終了し，すべての RiteVM は同時に次のロードの待機状態へと入る．

3.3 コンポーネントベース開発の利点

この章では，コンポーネントベース開発の利点を使った設計について述べる．

提案フレームワークでは，RiteVM や RiteVM スケジューラ，イベントフラグはコンポーネントとして提供されているため，開発者はそれらのコンポーネントを容易に付け外し，再利用することができる．例えば，RiteVM スケジューラの機能を外したい場合，図 12 に示される cdl ファイルを *//import(<tRiteVMScheduler.cdl>);* のようにコメントアウトすることで容易に実現できる．開発者は，カーネルのコンフィグレーションファイルを修正する手間を省くことができる．

それに加えて，コンポーネントベース開発ではコード量を減らすことができる．提案フレームワークでは，イベントフラグにこの利点は見られる．図 13 に示されるように，イベントフラグのセットパターンと待ちパターンを属性として定義する．*cEventflag_set(ATTR_setPattern)* に見られるこの設計によって，if 文なしでプログラムを記述でき，RiteVM の数に関わらず同じ C ファイルを使用できる．さらに，TECS のオプションである *[optional]* によって，呼び口が結合した場合のみ処理を行うため，イベントフラグの結合を切った場合にでも同じ C ファイルを利用することができる．

4. 評価実験

この章では，評価実験の結果と考察について述べる．提案フレームワークの利点を分析するため，以下の評価を行った．

- 転送される mruby バイトコードのサイズと処理時間およびコンパイル時間
- シングルタスク，コルーチン，マルチタスクの実行時間
- 周期時間によるオーバヘッド
- コンポーネントベース開発を利用したコード行数

本評価実験は，ターゲットデバイスとして，LEGO MINDSTORMS EV3 (300MHz ARM9-based Sitara AM1808 system-on-a-chip) を使用した．コンパイラは，gcc 4.9.3 -O2 および mruby のバージョンは，1.2.0 である．

表 1 に，mruby アプリケーションのバイトコードと，mruby ライブラリとアプリケーションのバイトコードのサイズ，ロードにかかる処理時間，コンパイル時間の比較を示す．0 バイトのバイトコードを処理した場合にかかるロード処理時間のオーバヘッドは，50.933 msec である．同様に 0 バイトのバイトコードをコンパイルした場合にかかるコンパイルのオーバヘッドは，46.9 msec である．mruby アプリケーションのみのバイトコードの方が，ライブラリを含めた場合よりも，すべての点で優れている．これは，RiteVM ひとつの評価であるため，RiteVM の数が増えるにつれ，この差は広がっていく．さらに，この設計では，SD カードや ROM を接続したり，OS を再起動する手間も省くことができる．以上より，提案フレームワークは，既存フレームワークより作業効率を向上させることができる．

次に，図 14 に，シングルタスク，コルーチン，マルチタスクの実行時間を示す．評価用のアプリケーションとして，100,000 回ループするプログラムを適用した．マルチタスクおよびコルーチンは，2 つのタスクでそれぞれ 50,000 回

表 1　転送される mruby バイトコードのサイズと処理時間およびコンパイル時間

	App&Lib	App	App&Lib/App
Bytecode Size	14,044 bytes	199 bytes	×70.6
Load Process Time	305.081 msec	7.774 msec	×39.2
Compile Time	8.7 msec	0.3 msec	×29.0

表 2　コンポーネントベース開発を利用したコード行数

	(A)	(B)	Diff
.c (Total)	8×α+134	130	8×α+4
.c (Modified)	10×α−2	0	10×α−2
.cdl	18×α+25	18×α+25	0

α : the number of RiteVM

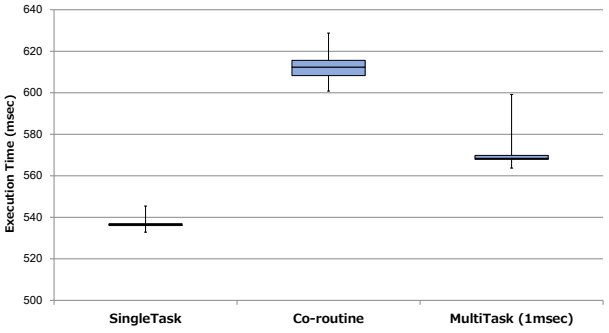

図 14　シングルタスク，コルーチン，マルチタスクの実行時間

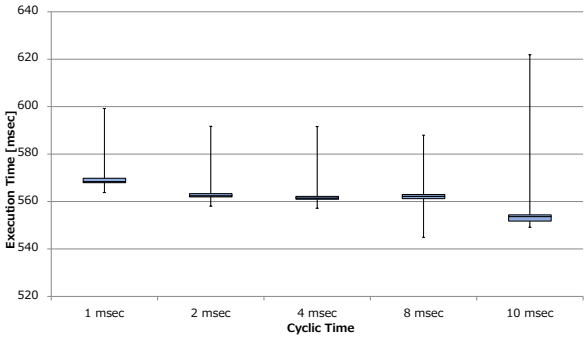

図 15　周期時間によるオーバヘッド

ループするプログラムを用いた．マルチタスクの実行時間は，シングルタスクよりも遅いが，コルーチンより速い．この結果から，提案フレームワークのマルチタスク処理は，コルーチンより有効である．RiteVM スケジューラのオーバヘッドは，約 5%である．タスクの切り替えそのものにかかる時間は，約 3 μ sec である．RiteVM スケジューラは，周期的に割り込み，タスクの切り替えの処理を行うため，これがオーバヘッドとなっている．

図 15 は，RiteVM スケジューラの周期時間ごとの実行時間を示している．この評価では，TOPPERS/HRP2 の仕様により周期時間の下限を 1msec とした．また，周期時間が大きいとアプリケーションに影響を与える可能性があるため，上限を 10msec とした．周期時間が大きいほど，タスクの切り替え回数は減少するため，実行時間は小さくなることが分かる．1msec と 10msec では，約 3%減少している．1msec と 8msec では，約 1%減少している．タスクを並行動作させるには，より小さい周期時間の方が好まれるが，オーバヘッドは大きくなる．しかし，周期時間によるオーバヘッドは，誤差に埋もれるほど小さい．

表 2 に，2 つの.c ファイルと.cdl ファイルのコード行数と変更行数の比較を示す．(A) と (B) はそれぞれ，図 13 の上部と下部を指している．(A) の.c ファイルは，RiteVM の数に比例して増えているのに対して，(B) の.c ファイルは同じである上に，RiteVM の数にかかわらず，修正することなく同一のファイルを使用できる．さらに，.cdl ファイルは両者違いはない．コンポーネントベース開発を上手く活用することで，コードの行数を減らすことができ，高い生産性やメンテナンス性につながる．その上，同じ.c ファイルを利用できるため，開発者がファイルを修正する手間を省くことも可能になる．

5. 関連研究

現在，オープンソースのスクリプト言語のランタイムシステムとして，次のものが開発されている．python-on-a-chip [10], the Owl system [11], eLua [12], mruby [13], [14], mruby on TECS [3].

python-on-a-chip(p14p) は，PyMyte と呼ばれる PythonVM を使用する，Python ランタイムシステムである．PyMyte は，マイクロコントローラー上で低リソースで Python プログラムを実行する．p14p は，スタックレスなグリーンスレッドを複数動作させることができる．

Owl system は，組込みシステム向けの Python ランタイムシステムであり，ARM Cortex-M3 マイクロコントローラー上で動く．Owl ツールチェーンは，Python コードから，マイコン上で直接起動するメモリイメージを生成する．Owl system は，p14p のインタプリタを使用している．

eLua は，Lua を組込みシステム向けの完全な実装である．Lua は，協調的マルチタスク処理が可能なコルーチンをサポートしているが，コルーチンは，ノンプリエンプティブであるため，マルチタスク処理のようなプログラムを行うのは難しい．

mruby は，コルーチンをサポートしているが，RTOS のマルチタスクには対応していない．

mruby on TECS は，マルチタスクをサポートしているが，マルチタスク処理を行うには，開発者が RTOS の機能を呼び出す必要がある．

表 3 に，提案フレームワークと既存研究の比較を示す．提案フレームワークでは，ローダに加えて，VM スケジューラとアプリケーションの同期機構を提供する．

6. おわりに

本研究では，mruby on TECS の拡張として，Bluetooth

表 3 関連研究

	Bluetooth Loader	Call C Function	Legacy Code of Embedded System	VM Managenment	VM Scheduler	Synchronization of Application	Co-routine
python-on-a-chip [10]							✓
Owl system [11]		✓	Partially				✓
eLua [12]		✓	Partially				✓
mruby [13]		✓					✓
mruby on TECS [3]		✓	✓	✓			✓
Proposed framework	✓	✓	✓	✓	✓	✓	✓

を用いた mruby バイトコードローダと RiteVM スケジューラを提案した．mruby バイトコードローダによって，開発者は SD カードや ROM を上書きしたり，OS を再起動する手間が省けるため，mruby on TECS でのソフトウェア開発の作業効率を向上させる．ローダは，Bluetooth だけでなく，有線のシリアル通信にも対応しているため，様々な組込みシステムに適用できる．RiteVM スケジューラは複数の mruby アプリケーションを効率良く並行実行する．実験評価では，ローダによる作業効率の向上やオーバヘッドの少ないマルチタスク設計の有用性，コンポーネントベース開発の利点を示した．

さらに，提案フレームワークで提供される機能はコンポーネントであるため，機能の取り外しや再利用が容易になる．RiteVM スケジューラも容易に付け外しできるため，開発者は，フェアスケジューリングか固定優先度スケジューリングかを選択することもできる．コンポーネントベース開発は，生産性を向上させ，システムの複雑さを減らすことができる．

ソフトウェア開発をさらに効率よく行うために，プラグインを使用した RiteVM や mruby-TECS ブリッジの.cdl ファイルの自動生成や，コマンドライン上でのバイトコードの転送などを提供することが，今後の課題である．

参考文献

[1] Azumi, T., Yamamoto, M., Kominami, Y., Takagi, N., Oyama, H. and Takada, H.: A New Specification of Software Components for Embedded Systems, *Proc. IEEE ISORC*, pp. 46–50 (2007).
[2] : AUTOSAR, `http://www.autosar.org/`.
[3] Azumi, T., Nagahara, Y., Oyama, H. and Nishio, N.: mruby on TECS: Component-Based Framework for Running Script Program, *Proc. IEEE ISORC*, pp. 252–259 (2015).
[4] Li, Y., Ishikawa, T., Matsubara, Y. and Takada, H.: A Platform for LEGO Mindstorms EV3 Based on an RTOS with MMU Support, *OSPERT 2014*, p. 51 (2014).
[5] Ishikawa, T., Azumi, T., Oyama, H. and Takada, H.: HR-TECS: Component technology for embedded systems with memory protection, *Proc. IEEE ISORC*, pp. 1–8 (2013).
[6] Takada, H. and Sakamura, K.: μITRON for Small-Scale Embedded Systems, *IEEE Micro*, Vol. 15, No. 6, pp. 46–54 (1995).
[7] Ohno, A., Azumi, T. and Nishio, N.: TECS Components Providing Functionalities of OSEK Specifications for ITRON OS, *Journal of Information Processing*, Vol. 22, No. 4, pp. 584–594 (2014).
[8] TOPPERS: TOPPERS/ASP kernel, `https://www.toppers.jp/en/asp-kernel.html`.
[9] : class Fiber, `http://docs.ruby-lang.org/en/2.3.0/Fiber.html`.
[10] : python-on-a-chip, `http://code.google.com/archive/p/python-on-a-chip/`.
[11] Barr, T. W., Smith, R. and Rixner, S.: Design and Implementation of an Embedded Python Run-Time System, *Proc. USENIX ATC 12*, pp. 297–308 (2012).
[12] : eLua, `http://www.eluaproject.net`.
[13] Tanaka, K., Matsumoto, Y. and Arimori, H.: Embedded System Development by Lightweight Ruby, *Proc. ICCSA*, pp. 282–285 (2011).
[14] : mruby, `https://github.com/mruby/mruby`.