Vol.2016-ARC-219 No.8
Vol.2016-SLDM-175 No.8
Vol.2016-EMB-40 No.8
2016/3/24

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，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 プログラムを効率良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. 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.

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，複雑・大規模なソフトウェアを効率的に開発できる．システムの拡張や仕様変更に柔軟に対応できる．組込みシステムのコンポーネント技術として，TECS (TOPPERS Embedded Component System) [1] や AUTOSAR [2] 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．現在の組込み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持っているため，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，Ruby，Perl，Python，Lua などがある．

しかし，スクリプト言語は生産性が高い反面，C 言語よりも実行時間が遅く，実行時間にばらつきがある．多くの組込みシステムは，リアルタイム性が重要であるため，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 [3] は，mruby(軽量 Ruby) [4] と，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 を利用したフレームワークである．mruby on TECS では，

[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

Vol.2016-ARC-219 No.8
Vol.2016-SLDM-175 No.8
Vol.2016-EMB-40 No.8
2016/3/24

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，SD カードや ROM への書き込みにしか対応していないため，作業効率が悪い．例えば，LEGO MINDSTORMS EV3 のプラットフォーム [5] を利用した場合，mruby プログラムを修正する度に，SD カードの抜き差しを行う必要がある．さらに，mruby on TECS はマルチ VM に対応しているが，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．

本論文では，上記の問題を解決するため，mruby on TECS を拡張して，Bluetooth を用いた mruby バイトコードローダおよび RiteVM スケジューラを提案する．

本論文における貢献は次の通りである．mruby バイトコードローダにより，ソフトウェア開発における作業効率を向上させる．RiteVM スケジューラにより，複数の mruby プログラムを効率的に並行動作させる．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．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，提案フレームワークの作業効率や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. mruby on TECS (既存フレームワーク)

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 (軽量 Ruby)

mruby は，Ruby の高い生産性を持ち，Ruby より少ないリソースで動作するため，組込みシステムに適している．mruby には，RiteVM と呼ばれる VM が実装されており，どの OS 上でも同じ mruby プログラムを実行できる．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．

2.2 TECS [1]

TECS によるコンポーネントベース開発は，ソフトウェアの再利用性を上げるため，生産性を向上できる．

2.2.1 コンポーネントモデル

図 1 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，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

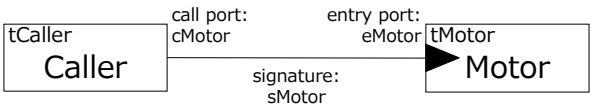

図 1　TECS コンポーネント図の例

```
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 2　シグニチャ記述

```
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 3　セルタイプ記述

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，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，組上げ記述に分類され，cdl ファイルに記述する．図 1 の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．図 2 に示す通り，*signature* キーワードに続けて，シグニチャ名 (sMotor) を記述する．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] という指定子が付けられる．

セルタイプ記述は，受け口，呼び口，属性，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 (tCaller) を記述する．図 3 に示す通り，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．

組上げ記述は，セルをインスタンス化し，セルを結合する．*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，セルを結合する．図 4 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 を用いたコンポーネント

Vol.2016-ARC-219 No.8
Vol.2016-SLDM-175 No.8
Vol.2016-EMB-40 No.8
2016/3/24

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

図 4 組上げ記述

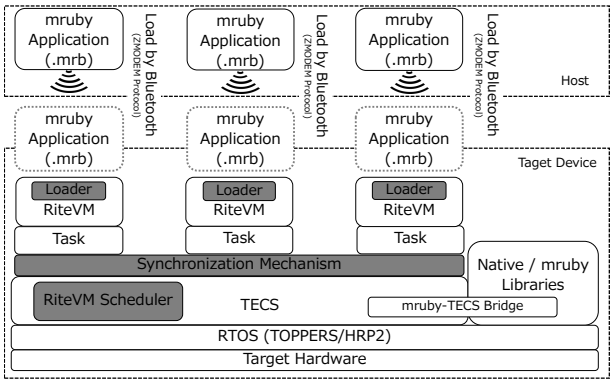

図 5 提案フレームワークのシステムモデル

ベース開発が可能なフレームワークである．mruby バイトコードはそれぞれ，コンポーネント化された RTOS (リアルタイム OS) のタスクとして，RiteVM 上で実行される．さらに，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．

本研究では，RTOS として，TOPPERS/HRP2 [6] を使用した．TOPPERS/HRP2 は，μITRON [7] 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 [8] や TOPPERS/ASP [9] といった RTOS にも対応しているため，mruby on TECS は RTOS に依存しない．

3. 設計と実装

図 5 に提案フレームワークの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，同期処理され実行される．RiteVM スケジューラは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できる．mruby アプリケーションの起動はイベントフラグを用いた同期機構により同期される．

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．既存フレームワークでは，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 に再度書き込み，OS を再起動する必要がある．提案フレームワークでは，一度だけ SD カードや ROM に書き込むため，作業効率を向上できる．

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．アプリケーションとライブラリをまとめたバイトコードを転送・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．この設計により，転送するバイトコードのサイズや処理時間の無駄を省ける．さらに，各 RiteVM でライブラリを共有することや，固有のライブラリを持つことも可能になる．

図 6 ローダを実装した RiteVM のコンポーネント図

3.1.1 ローダを実装した RiteVM のコンポーネント

ローダは，mruby on TECS で提供されている RiteVM コンポーネント [3] を拡張して実装した．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 6 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 RTOS (TOPPERS/HRP2) のタスクである．MrubyBluetooth1 は，ローダを実装した RiteVM コンポーネントであり，転送されるバイトコードを受信する．バイナリ転送プロトコルは，汎用性が良く，パフォーマンスが高いため，ZMODEM [10] を適用した．

図 7 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フローチャート，図 9 に処理のソースコードを示す．

はじめに，mruby で使われる状態と変数のセットである *mrb_state* と *mrbc_context* のポインタを初期化する (図 7 : A，図 9 : 2-5 行目)．

次に，mruby ライブラリのバイトコードを読み込む (図 7 : B，図 9 : 8 行目)．図 8 に示す通り，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われるため，この属性はメモリを消費しない．*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 (図 7 : C，図 9 : 10，12 行目)．mruby アプリケーションのバイトコードは，変数 uint8_t 型の配列 *irepApp* に格納される．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 (図 7 : D，図 9 : 12 行目)．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．

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．

Vol.2016-ARC-219 No.8
Vol.2016-SLDM-175 No.8
Vol.2016-EMB-40 No.8
2016/3/24

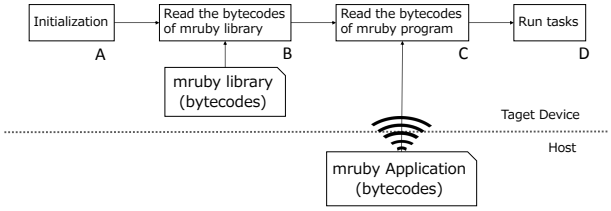

図 7　ローダの処理フローチャート

```
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 8　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 mrb_state*/
4    /*Omit: error check for mrb_state*/
5    c=mrbc_context_new( mrb ); /*New mrbc_context*/
6    /*Omit: initialization of mruby−TECS bridge*/
7    /*Load mruby library bytecode and run*/
8    mrb_load_irep_cxt( mrb, ATTR_irep, c );
9    /*Receive the bytecode via Bluetooth*/
10   bluetooth_loader( VAR_irepApp );
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 9　ローダを実装した RiteVM のメインコード

マルチタスク処理のアプローチの一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行動作できるが，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ため，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．Ruby のコルーチンは，Fiber クラス [11] に定義されている．

その他にも，μITRON のサービスコールである *delay()* を使った手法がある．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 スケジューラを提供し，複数の VM を平等に並行動作させる．RiteVM スケジューラは RiteVM 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なく使うことができる．

3.2.1　RiteVM スケジューラ

RiteVM スケジューラは，周期ハンドラを利用して実装し，同優先度のタスクの実行を切り替える μITRON のサービスコールである *rotateReadyQueue* を周期的に呼び出す．

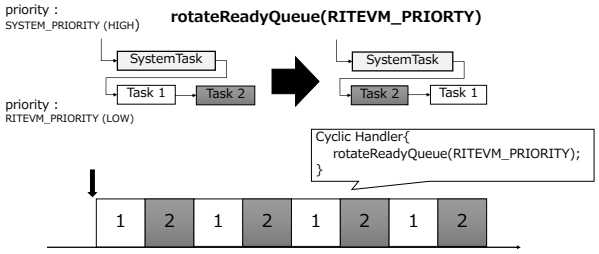

図 10　RiteVM スケジューラの設計

```
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 11　RiteVM スケジューラの組上げ記述

RiteVM スケジューラの設計を図 10 に示す．

rotateReadyQueue が呼ばれると，図 10 に示す通り，同優先度タスクの実行順序が切り替わる．*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 A，B，C が順に起動する場合，*rotateReadyQueue* が呼ばれると，B，C，A の順にタスクの実行順序は切り替わる．

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，3 つの属性 (attribute，cyclicTime，cyclicPhase) を持っており，μITRON の周期ハンドラ [7] の設定を行う．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．呼び口は，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受け口 (*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 11 に，RiteVM スケジューラの組上げ記述を示す．attribute には，OS が起動すると実行状態になる属性である *TA_STA*, 周期時間には, 1ms が与えられる．CyclicMain セル部分は，属性 priority を持っており，mruby タスクの優先度である RITEVM_PRIORITY が与えられる．

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，イベントフラグを用いて同期処理を行った．RiteVM が 4 つの場合，各 VM はそれぞれ，イベントフラグに 0x01(0001), 0x02(0010), 0x04(0100), 0x08(1000) のパターンをセットし，待ち状態へと遷移する (図 12 (A))．イベントフラグが待ちパターン 0x0f(1111) になると，各 VM は実行可能状態になり，アプリケーションを同時に実行する．

さらに，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する．終了も同期する

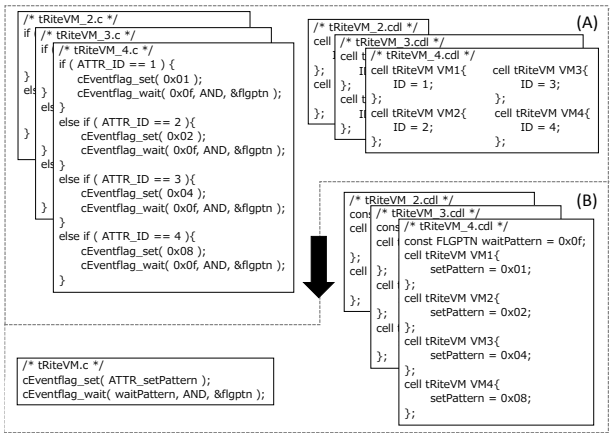

図 12 TECS を用いたイベントフラグの設計 *差分のみ示す

ことで，すぐに終了するアプリケーションの RiteVM が次のロード待機状態に入るのを防げ，すべての RiteVM は同時に次のロード待機状態になる．

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は，RiteVM や RiteVM スケジューラ，イベントフラグをコンポーネントとして提供しているため，開発者はそれらのコンポーネントを容易に付け外し，再利用できる．例えば，RiteVM スケジューラの機能を外したい場合，図 11 に示される cdl ファイルを*//import(<tRiteVMScheduler.cdl>);* とコメントアウトするだけで良いため，開発者はカーネルの設定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．図 12 (B) に示す通り，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)* のように，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 *[optional]* によって，呼び口が結合した場合のみ処理を行うため，イベントフラグを利用しない場合でも同じ C ファイルを利用できる．

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，転送する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の評価を行った．

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，ライブラリとアプリケーションを合わせたバイトコードのサイズ，ロード処理時間，コンパイル時間の比較を示す [*1]．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．上記の結果は，RiteVM 一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 を上書きする手間や，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できる．

表 1 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
Loading Time	305.081 msec	7.774 msec	×39.2
Compile Time	8.7 msec	0.3 msec	×29.0

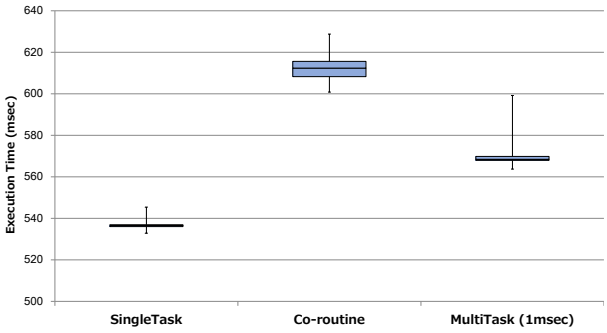

図 13 シングルタスク，コルーチン，マルチタスクの実行時間

次に，図 13 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．評価用アプリケーションとして，100,000 回ループするプログラム (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 回ループ) を用いた．実行時間の観点では，提案手法のマルチタスクは，コルーチンより有効である．シングルタスクと比べても，RiteVM スケジューラのオーバヘッドは約 5%と実用的に問題ない．タスクの切り替えそのものにかかる時間は，約 3μs 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 14 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 1ms とした．一方で，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 8ms 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．1ms と 8ms では，約 1%しか減少していない．小さい周期時間でもオーバヘッドは大きくないため，効率的にタスクを並行動作できる．

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 12 の上

[*1] 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 ms 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 ms である．

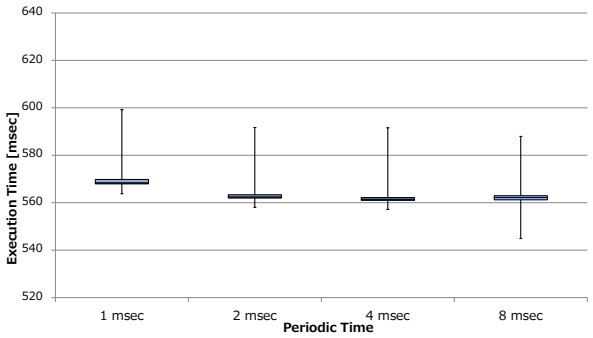

図 14　周期時間によるオーバヘッド

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 (Modification)	10×α−2	0	10×α−2
cdl	18×α+25	18×α+25	0

α：the number of RiteVM

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 の数に関わらず，修正することなく同一のファイルを使用できる．cdl ファイルに差はない．コンポーネントベース開発を上手く活用することで，コードの行数が減ることを示した．その上，RiteVM の数に限らず同じ c ファイルを利用できる．

5. 関連研究

現在，スクリプト言語を適用した組込みシステム向け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 [12]，Owl system [13]，eLua [14]，mruby [4]，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，低リソースで Python プログラムを実行する．p14p では，複数のグリーンスレッドを並行動作できる．

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 マイコン上で動く．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 の組込みシステム向け実装である．Lua は，コルーチンをサポートしているが，ノンプリエンプティブであるため，マルチタスク処理を行うのは難しい．

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，マルチタスクに対応しているが，開発者が RTOS の機能を呼び出す必要がある．

6. おわりに

本研究では，mruby on TECS の拡張として，Bluetooth 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．ローダによって，開発者は SD カードや ROM を上書きする手間や，OS を再起動する手間が省けるため，ソフトウェア開発の作業効率を向上できる．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し，同期機構によりタスク起動を同期する．実験評価では，ローダによる作業効率の向上や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．

ソフトウェア開発をさらに効率良く行うために，VM 間の通信機能や，コマンドライン上でのバイトコード転送機能を提供することが，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, *in 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, *in Proc. IEEE ISORC*, pp. 252–259 (2015).
[4] Tanaka, K., Matsumoto, Y. and Arimori, H.: Embedded System Development by Lightweight Ruby, *in Proc. ICCSA*, pp. 282–285 (2011).
[5] 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).
[6] Ishikawa, T., Azumi, T., Oyama, H. and Takada, H.: HR-TECS: Component technology for embedded systems with memory protection, *in Proc. IEEE ISORC*, pp. 1–8 (2013).
[7] Takada, H. and Sakamura, K.: μITRON for Small-Scale Embedded Systems, *IEEE Micro*, Vol. 15, No. 6, pp. 46–54 (1995).
[8] 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).
[9] Azumi, T., Takada, H., Ukai, T. and Oyama, H.: Wheeled Inverted Pendulum with Embedded Component System: A Case Study, *in Proc. IEEE ISORC*, pp. 151–155 (2010).
[10] Forsberg, C.: The ZMODEM Inter Application File Transfer Protocol, `http://pauillac.inria.fr/~doligez/zmodem/zmodem.txt` (1988).
[11] class Fiber: `http://docs.ruby-lang.org/en/2.3.0/Fiber.html`.
[12] python-on-a-chip: `http://code.google.com/archive/p/python-on-a-chip/`.
[13] Barr, T. W., Smith, R. and Rixner, S.: Design and Implementation of an Embedded Python Run-Time System, *in Proc. USENIX ATC 12*, pp. 297–308 (2012).
[14] eLua: `http://www.eluaproject.net`.