軽量 Ruby のハードウェア制御ライブラリの実装に関する研究

永山将成† 田中和明†

近年，組込み製品の多機能・高付加価値化により組込みシステムは大規模で複雑なものになっている．また，製品ライフサイクルの短期化により開発期間を短縮しなければならず，開発効率や品質の向上が課題となっている．そこで，生産性の高いプログラム言語の Ruby を組込み向けに適用した軽量 Ruby が公開されている．しかし，軽量 Ruby から RTOS の操作やハードウェア制御を行うためには，実行環境に合わせたドメイン別ライブラリの整備が必要になる．そこで，本研究では軽量 Ruby から RTOS の操作及びハードウェア制御を行うライブラリの実装法を提案し，軽量 Ruby を用いた組込みシステムの開発・検証を行った．

Studies on the implementation of the hardware control library for mruby

MASANARI NAGAYAMA† KAZUAKI TANAKA†

In recent years, embedded systems have become large and complex ones. In addition, development time is shortened. Therefore, the development efficiency and improve the quality has become an issue. So, mruby applied to the embedded Ruby has been published. In this study, we propose a method to the implementation of the library for hardware control and RTOS from mruby. In this paper, we show the results of out library implementation and its applications. As these results, some tasks in RTOS are dispatched by ruby monitor and communicate each other.

1. はじめに

近年，組込み製品の多機能・高付加価値化により組込みシステムは大規模で複雑なものになっている．また，製品ライフサイクルは短期化しており，組込みシステムの開発期間も短縮傾向にある．そのため，大規模で複雑なシステムを短期間で開発しなければならず，開発効率や品質の向上が課題となっている．

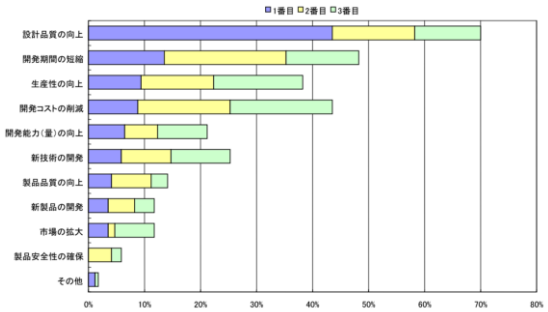

図 1 組込み開発の課題

そこで，生産性の高いプログラム言語の Ruby を組込み向けに適用した軽量 Ruby の開発が行われている．この軽量 Ruby にはミニマル，スタンダード，フルの基本ライブラリと実装環境用のドメイン別の 4 種類のライブラリ仕様がある．しかし，現在は最小構成のミニマルライブラリのみの公開となっており，実際に組込み機器で使用するためにはドメイン別ライブラリの整備が必要になる．

そこで，本研究では移植性の高いドメイン別ライブラリの構成を提案し，幅広い製品での軽量 Ruby の利用を目指す．また，実際に RTOS 用やハードウェア制御用のライブラリの実装及び動作検証を行った．

本論文では 2 章で軽量 Ruby の概要及びライブラリの構成について説明する．次に，3 章で本研究に使用する実装環境及び軽量 Ruby の実装法について説明する．4 章では実装したライブラリの動作検証を行い，5 章で本研究のまとめを述べる．

2. 軽量 Ruby

2.1 軽量 Ruby の概要

軽量 Ruby は組込みシステムのようなリソースの限られた環境で動作するように Ruby を適用した言語である．2012 年 4 月にオープンソースソフトウェアとして公開され，組込みシステムをはじめとした様々なシステムでの利用が期待されている．

2.2 軽量 Ruby の特徴

Ruby はインタプリタ型言語であり，多くのリソースを必要とするため，リソースに限りがある組込みシステムでは

† 九州工業大学大学院情報工学府
Graduate School of Information Engineering, Kyushu Institute of Technology

動作が困難である．また，Ruby スクリプトの逐次解析・実行や GC によるプログラムの一時停止などリアルタイム制御が必要不可欠な組込みシステムに対して大きな欠点がある．そこで，軽量 Ruby はコンパイラにより出力されるバイトコードを軽量 Ruby の実行基盤である RiteVM 上で動作させる． VM 方式を採用することで Ruby に比べて実行時のリソースが少なく，処理速度が速いという利点がある．

また，C/C++言語との親和性が高く，C/C++言語などで開発されたプログラムの呼び出しが容易で，拡張ライブラリとして使用することができる．図 2 に軽量 Ruby のソフトウェア構成を示す．さらにインクリメンタル GC を採用することで，プログラムの中断時間を短縮し，リアルタイム性を確保している．

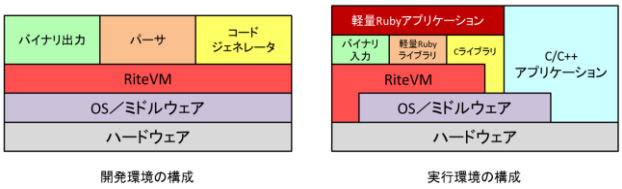

図 2　軽量 Ruby のソフトウェア構成

2.3 ライブラリ構成

軽量 Ruby は様々な環境で動作するように実行環境に合わせたライブラリを実装することができるように設計されている．軽量 Ruby ライブラリの種類とそれぞれの内容について表 1 及び図 3 に示す．

本研究ではドメイン別の拡張ライブラリを対象に実装及び動作検証を行った．

表 1　ライブラリの種類

ライブラリ	内容
ミニマル	実行に必要な最低限の機能
スタンダード	JIS X 3017 規格相当
フル	MRI(CRuby)のフル機能相当
ドメイン別	製品ドメイン別の拡張ライブラリ

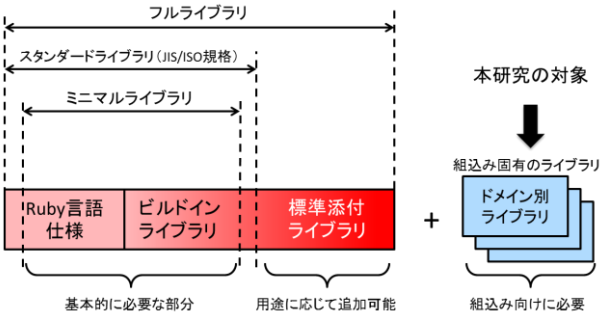

図 3　軽量 Ruby ライブラリの位置づけ

2.4 ライブラリ開発方法

軽量 Ruby のライブラリを実装し使用するためには，RiteVM にライブラリとして使用するメソッドやクラスの定義を行う必要がある．表 2 にメソッド定義関数について示す．この関数を RiteVM に定義することで，軽量 Ruby から C 言語の機能関数を呼び出すことができるようになり，実装環境の機能が使用可能になる．

表 2　メソッド定義関数

関数型	void mrb define method(mrb state *mrb, struct RClass *c, const char *name, mrb func t func, int aspec)	
引数	mrb	mrb state ポインタを指定
	c	クラス/モジュールのオブジェクトを指定
	name	mruby から呼び出すメソッド名
	func	呼び出し対象の C 言語機能関数を指定 mrb value func(mrb state *, mrb value)
	aspec	メソッドが受け取る引数の数を指定
戻り値	なし	

2.5 ライブラリ構成の提案

軽量 Ruby のライブラリは C 言語の機能関数を RiteVM にメソッドとして登録することで使用できる．ライブラリの基本構成を図 4 に示す．

しかし，基本構成でライブラリを実装した場合，RiteVM への機能関数の定義が実装環境に依存してしまい，組込みシステムで重要な移植性を損なってしまう．そこで，本研究では図 5 に示すライブラリ構成を提案した．RiteVM と機能関数の間に C 言語関数を挟むことで，RiteVM と機能関数が直接データのやり取りを行うことがなくなり，ソフトウェアやライブラリの移植性が向上することが期待できる．

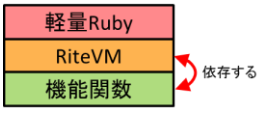

図 4　基本構成

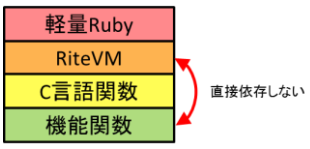

図 5　本研究の構成

3. 実装

3.1 RTOS

3.1.1 ITRON

ITRON は組込み向けリアルタイムカーネル仕様である．ITRON 系の RTOS は組込み OS の 40%以上のシェアを占めており，様々な製品に使用されている．軽量 Ruby の幅広い製品での使用を目指す本研究では ITRON 系 RTOS を採用した．図 6 に組込み OS のシェアについて示す．

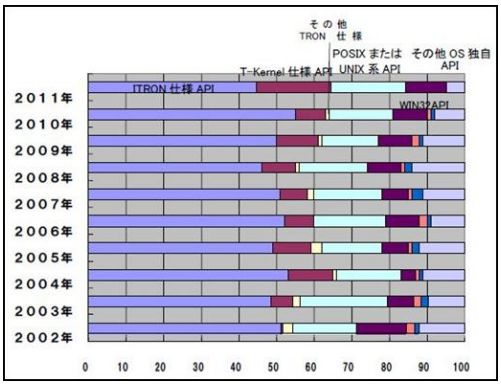

図 6　組込み OS のシェア

3.1.2 TOPPERS/ASP カーネル

TOPPERS は, オープンソースの組込み向けリアルタイムカーネルおよびソフトウェア部品群である．その中でも ASP カーネルは μTRON4.0 仕様を基盤とし，組込みシステムの多様化に対応するために開発された基本的なカーネルである．拡張性や信頼性の改良をはじめ，割り込み処理機能の実装を行ったものになっている．

ASP カーネルは TOPPERS プロジェクトの中軸で，ASP カーネルを基に様々なカーネルが開発されている．また，ターゲットプロセッサが多いことも特徴である．本研究では, 多くのカーネルの基になる ASP カーネル向けのライブラリを実装した．

3.2 SH マイコン

SH マイコンは，組込み向け 32bit マイクロコンピュータで，携帯電話やカーナビといった小型の製品から，自動車や産業ロボットのような大型の製品まで幅広く使用されている．本研究では，その中でも高性能な SH-4 プロセッサをターゲットに, ハードウェア制御ライブラリを実装した．本研究では CPU ボードにアルファプロジェクト製の SH-4 マイコンボードを用いた．主な仕様を表 3 に，外観を図 7 に示す．

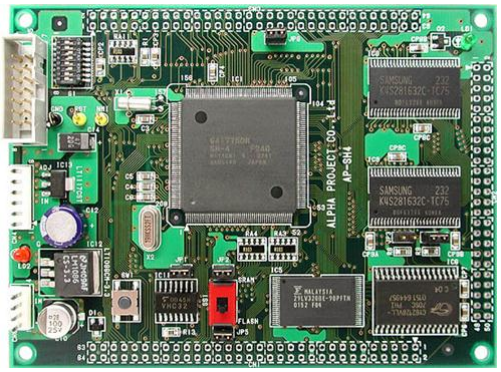

図 7　AP-SH4-1A（CPU ボード）外観

表 3　主な仕様

項目	仕様
CPU	HD6417750RF240　（SH7750R）
クロック	最大 235.9296MHz
Flash ROM	4Mbyte
SDRAM	32Mbyte
SRAM	256Kbyte

3.3 mruby の実装

軽量 Ruby のソフトウェアは mruby と呼ばれる．この mruby の実装方法には大きく分けて 2 種類ある．1 つ目がファイルシステムを持つ環境用の方法である．コンパイル済みのバイナリファイル（mrb ファイル）を実行環境上のファイルシステムに置き，実行ファイル内の RiteVM に読み込ませる方法である．

2 つ目がファイルシステムを持たない環境用の方法である．コンパイル済みのバイナリデータ（C ファイル）を，RiteVM を含む C 言語プログラムと一緒にクロスコンパイルし，実行ファイルを生成する．そして，生成した実行ファイルを実行環境上で動作させる方法である．

本研究で使用する ASP カーネルはファイルシステムを持たないため，二つ目の実装方法を用いる．図 8 にファイルシステムを持たない環境用の実装の流れを示す．

また，mruby の RiteVM を ASP カーネルのタスクとして組込むことで動作させる．図 9 にプログラム構成のイメージ図を示す．

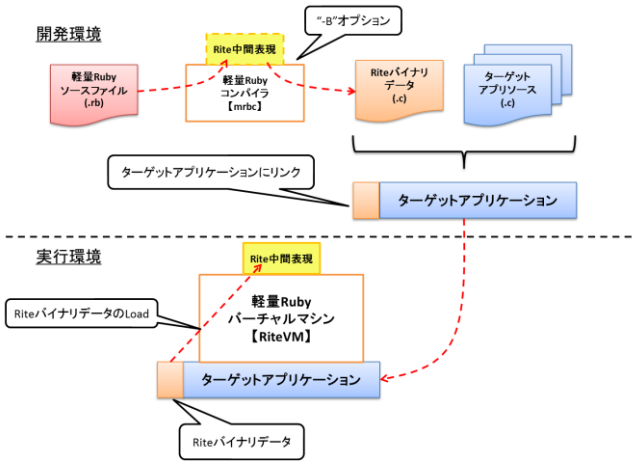

図 8　ファイルシステムを持たない環境用の実装法

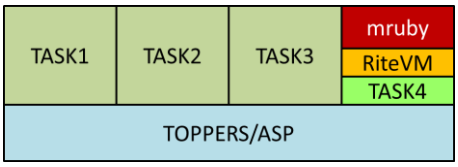

図 9　プログラム構成イメージ

4. 動作検証

4.1 検証方法

本研究では，ASP カーネルのタスク操作機能とタスク間通信機能や AP-SH4-1A の汎用 I/O ポートの入出力機能を mruby のメソッドとして利用するためのライブラリを作成した．作成したライブラリを mruby に定義し，ASP カーネルと一緒にクロスコンパイルして，実行ファイルを生成する．生成した実行ファイルを，デバッグツールを使用し AP-SH4-1A 上で実行する．実行状況は Serial 通信で PC 上のターミナルソフトにログ出力することで確認する．動作検証に用いたシステム構成を図 10 に示し，使用した環境を表 4 に示す．

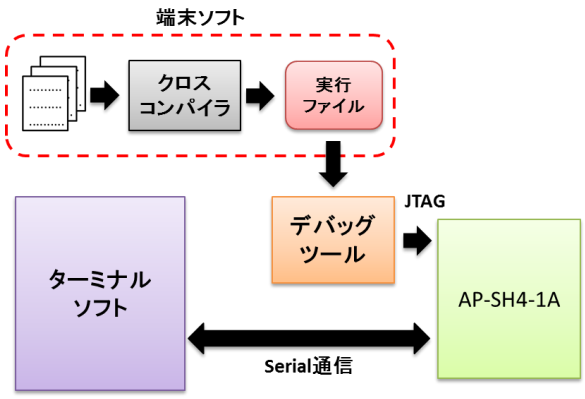

図 10　システム構成

表 4　検証環境

検証環境	
端末ソフト	cygwin　1.7.17-1
デバッグツール	XrossFinder Evo
	XsSight 2.0.0.5
ターミナルソフト	Tera Term 4.73

4.2 検証結果

4.2.1 タスク操作機能

mruby からタスクの操作を行うために C 言語 API を mruby のメソッドとして定義した．その際，タスク操作メソッドはタスク操作用の TASK クラスを作り，TASK クラスのメソッドとした．また，タスク操作メソッドは移植性を高めるためにクラス・メソッドの定義及び C 言語 API 関数を mruby 本体の関数から分割した．タスク操作ライブラリの構成は以下の通りである．

- mrb_init_task.c
  TASK クラス及びメソッド定義，呼び出し対象関数．
- mrb_init_task.h
  呼び出し対象関数のプロトタイプ宣言．
- mrb_task.c
  C 言語 API 関数．
- mrb_ task.h
  C 言語 API 関数のプロトタイプ宣言．

上記構成で定義したタスク操作メソッドの一覧を表 5 に示す．

表 5　タスク操作メソッド

軽量 Ruby	動作
TASK クラス.act(int tskid)	起動
TASK クラス.iact(int tskid)	起動
TASK クラス.cact(int tskid)	起動取り消し
TASK クラス.ext	タスクの終了
TASK クラス.ter(int tskid)	強制終了
TASK クラス.cpri(int tskid, int pri)	優先度の変更
int　TASK クラス.gpri(int tskid)	優先度の参照
TASK クラス.slp	起床待ち
TASK クラス.tslp (int tmout)	待ち（タイムアウト）
TASK クラス.wup(int tskid)	タスクの起床
TASK クラス.iwup(int tskid)	タスクの起床
TASK クラス.cwup(int tskid)	起床要求の無効化
TASK クラス.rwai(int tskid)	待ち状態の強制解除
TASK クラス.irwai(int tskid)	待ち状態の強制解除
TASK クラス.sus(int tskid)	強制待ちへ移行
TASK クラス.rsm(int tskid)	強制待ちからの再開
TASK クラス.dly(int dlytim)	自タスクの遅延

mruby のタスクを 2 つ使い，定義したタスク操作ライブラリの動作検証を行った．以下に検証の内容と検証結果について示す．

1. 起動，終了

2 つの mruby タスクを使い，タスクの起動，終了についてテストプログラムを作成し，動作検証を行った．テストプログラムの動作を以下に示す．

① メインタスクはサブタスクを起動する．
② サブタスクは処理終了後に終了する．
③ ①，②を繰り返す．
④ タスクが終了するとタスク内のデータは初期化される．

メインタスクとサブタスクのスクリプトを示し，実行結果を図 11 に示す．実行結果からサブタスクが起動されていることがわかる．また，サブタスク内のデータ i はタスク

終了に伴い初期化され，常に初期値での出力になっていることがわかる．このことから，タスクの終了動作が正常に行えていることが確認できた．

メインタスク

```
while (1) do
  syslog_0("main task")
  task.act(subtask)
  task.dly(500)
end
```

サブタスク

```
i = 1
while (1) do
  syslog _1("count:%d",i)
  i = i +1
  task.ext
end
```

リスト 1　テストプログラム 1

```
Graduate School of In

System logging task is started on
main task
count:1
main task
count:1
main task
count:1
main task
count:1
main task
count:1
main task
```

図 11　実行結果 1

2. 待ち移行，起床

タスクの待ち移行及び起床命令の動作確認を行った．テストプログラムの動作を以下に示す．

① メインタスクにより起動されたサブタスクは，起床待ち状態に入る．
② メインタスクからの起床命令により，サブタスクは起床し，処理を行う．
③ 処理終了後にサブタスクは待ち状態へ移行する．
④ ②，③の動作を繰り返す．
⑤ 待ち状態の場合，タスク内の状態は保持される．

テストプログラム及び実行結果を以下に示す．実行結果から起床命令及び待ち移行命令が正常に動作し，サブタスク内でカウントできていることが確認できた．

メインタスク

```
while (1) do
  syslog_0("main task")
  task.wup(subtask)
  task.dly(500)
end
```

サブタスク

```
i = 1
while (1) do
  task.slp
  syslog _1("count:%d",i)
  i = i +1
end
```

リスト 2　テストプログラム 2

```
System logging task is started on
main task
count:1
main task
count:2
main task
count:3
main task
count:4
main task
count:5
main task
count:6
```

図 12　実行結果 2

4.2.2 タスク間通信機能

複数のタスクを効率よく制御し，処理を行うために，タスク間通信機能のデータ・キューを DTQ クラスのメソッドとして定義した．ライブラリの構成はタスク操作ライブラリと同様である．

定義したタスク間通信メソッドの一覧を表 6 に示す．

表 6　タスク間通信メソッド

軽量 Ruby	動作
DTQ クラス.snd(int dtqid, int data)	送信
DTQ クラス.psnd(int dtqid, int data)	送信（ポーリング）
DTQ クラス.ipsnd(int dtqid, int data)	送信（ポーリング）
DTQ クラス.tsnd(int dtqid, int data, int tmout)	送信（タイムアウト）
DTQ クラス.fsnd(int dtqid, int data)	データの強制送信
DTQ クラス.ifsnd(int dtqid, int data)	データの強制送信
int DTQ クラス.rcv(int dtqid)	データの受信
int DTQ クラス.prcv(int dtqid)	受信（ポーリング）
int DTQ クラス.trcv(int dtqid, int tmout)	受信（タイムアウト）

2 つのタスク間でデータ・キューを用いたデータ通信プログラムを作成し，タスク間通信の動作検証を行った．テストプログラムの流れを以下に示す．

① タスク 1 は一定間隔でデータ･キューへデータを送る．
② タスク 2 はデータ・キューにデータの入力があったとき，データを読み出す．
③ 読み出したデータを出力し，受信待ち状態に移行する．
④ ①～③の動作を繰り返す．

タスク 1 とタスク 2 のスクリプトと実行結果を以下に示す．実行結果からタスク 1 からデータ・キューへ送信したデータをタスク 2 で正常に受信できていることが確認できた．また，タスク 2 はデータ・キューがデータを受け取るまで受信待ちの動作を行っていることが確認できた．

タスク1

```
while (1) do
  syslog_1 ("task1 snd data:%d", i)
  dtq.snd(dtqid, i)
  task.dly(1000)
  i = i + 1
end
```

タスク2

```
while (1) do
  data = dtq.rcv(dtqid)
  syslog_1("task2 rcv data:%d", data)
end
```

リスト 3　テストプログラム 3

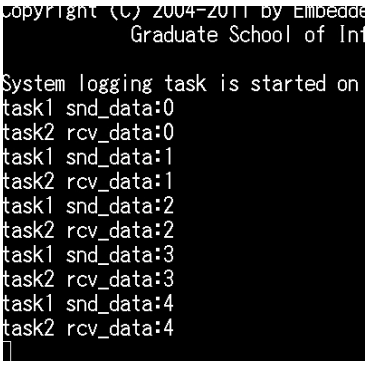

図 13　実行結果 3

4.2.3 ハードウェア制御

mruby からハードウェア制御を行うためのライブラリを作成した．タスク制御ライブラリと同様に，RiteVM とハードウェア操作関数の間に C 言語関数を挟むことで移植性を考慮した構造とした．

本研究では AP-SH4-1A の 20bit 汎用 I/O ポートを制御するライブラリの作成を行った．ライブラリの構成は以下に示す．また，定義したメソッドを表 7 に示す．

- mrb_init_io.c

  IO クラス及びメソッド定義，呼び出し対象関数．
- mrb_init_io.h

  呼び出し対象関数のプロトタイプ宣言．
- mrb_io.c

  ハードウェア制御用の C 言語関数．
- sh7750.h

  SH7750 用のレジスタ定義ヘッダーファイル．このファイル内に I/O ポートのレジスタ定義があり，ハードウェア制御用の C 言語関数から呼び出し，利用する．

表 7　I/O ポート制御メソッド

軽量 Ruby	動作
PORT クラス.set(int pin, int dir, int pup)	bit 単位入出力設定
PORT クラス.set_reg(char *reg, int value)	レジスタ単位入出力設定
PORT クラス.write(int pin, int value)	bit 単位出力
PORT クラス.write_reg(char *reg, int value)	レジスタ単位出力
int PORT クラス.read(int pin)	bit 単位でポートの入力

定義したメソッドを用いて，ハードウェア制御プログラムを作成した．プログラムの動作を以下に示す．

① 使用する I/O ポートの入出力設定を行う．
② 無限ループ内で入力ポートの値を監視する．
③ 入力ポートが”1”の場合，出力ポートへ”1”を出力し，LED を点灯させる．
④ 入力が”0”の場合は出力”0”となり，LED は消灯する．

実際に作成したプログラム及び回路図を以下に示す．また，実行結果を図 15 に示す．実行結果から SW を押したときに入力ポートに”1”が入り，出力ポートから”1”が出力され LED が点灯したことが確認できた．このことから，I/O ポートの制御を mruby から行えることが確認できた．

```
port.set(18, 0, 0)
port.set(16, 1, 1)

while (1) do
  val = port.read(18)
  if val == 1
    port.write(16, 1)
  else
    port.write(16, 0)
  end
end
```

リスト 4　テストプログラム 4

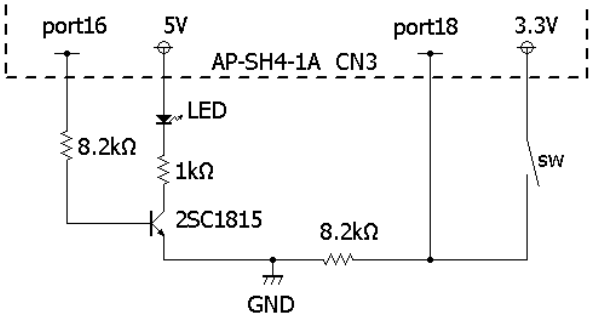

図 14　テスト回路図

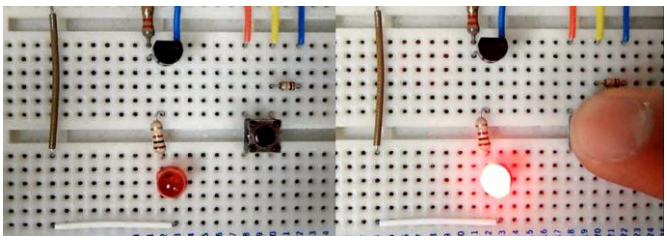

図 15　動作結果 4

5. まとめ

本研究では拡張ライブラリの構成を提案した．また，提案した構成を用いて RTOS 用とハードウェア制御用のライブラリを実装し，動作検証を行った．

提案したライブラリ構成で実装したライブラリを用いて作成したテストプログラムはいずれも正常に動作することが確認できた．

今後は今回実装していないタスク制御機能や PWM 出力などをメソッドとして定義し，さらに高度な制御を行えるようにすることで，様々な組込み製品での軽量 Ruby の利用を目指す．

また, TOPPERS の他のカーネルや ITRON 系の他の RTOS への移植，他の SH マイコンへの移植を行うことで，本研究で提案したライブラリ構成の移植性の検証を行う．

参考文献

1) 経済産業省，2009 年版組込みソフトウェア産業実態調査
2) T-Engine フォーラム，2011 年度組込みシステムにおけるリアルタイム OS の利用動向に関するアンケート調査報告書
3) インターフェース編集部，μITRON 準拠 TOPPERS の実践活用，2007 年 3 月