



# 組込みハードウェアモジュールへの mrubyアプリケーション適用試行

2017/11/01

株式会社 日立ソリューションズ  
業務革新統括本部 IT技術推進センター

三好 秀徳

# Contents

---

1. 自己紹介、Rubyに対する取り組み
2. 昨年実施内容の振り返りと今年の取り組み内容
3. 組込みハードウェアモジュール利用時のmruby適用
4. センサーデータの取得とその利用
5. まとめと今後の課題

---

# 1. 自己紹介、Rubyに対する取り組み

## ■ (株)日立ソリューションズ

- さまざまな業務システムをRubyを使って構築
- RubyWorld Conference / Ruby Associationにて事例を数多く紹介

### ■ 活文メールセキュリティ・利活用基盤

- メール誤送信防止のためのソリューション
- <http://www.ruby.or.jp/ja/showcase/case37>

### ■ 作業実施依頼システム

- メール・口頭で行っていた作業実施依頼を管理するシステム
- <http://www.ruby.or.jp/ja/showcase/case43>

## ■ (株)日立ソリューションズ

- さまざまな業務システムをRubyを使って構築
- RubyWorld Conference / Ruby Associationにて事例を数多く紹介

## ■ 日本OSS推進フォーラム アプリケーション部会

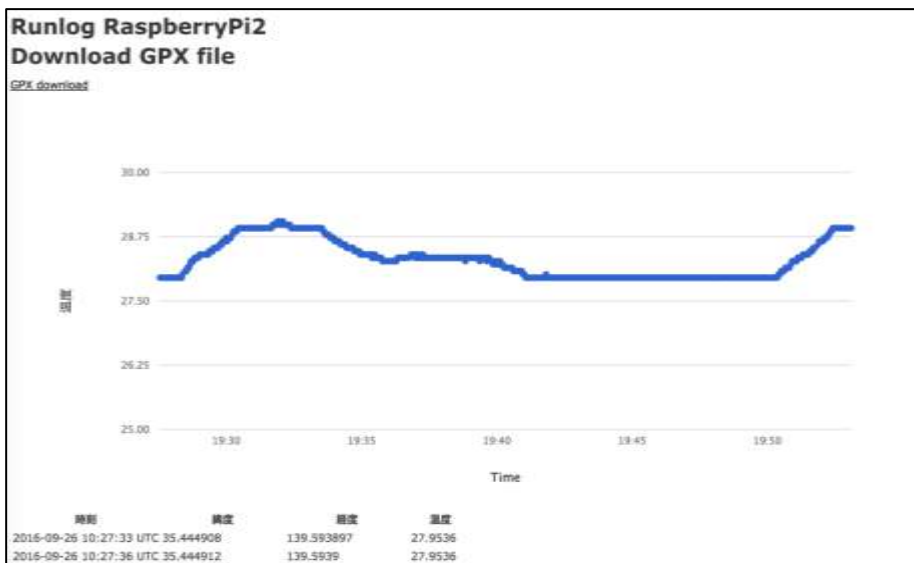
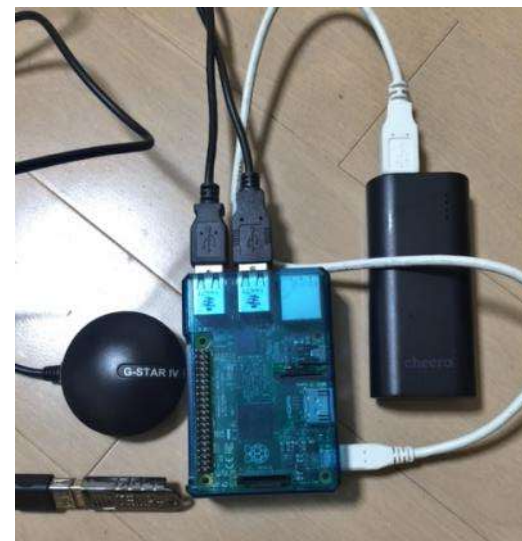
- ITビジネスの場へOSSの適用・普及・推進をはかる民間団体
- RubyWorld Conferenceにて事例を数多く紹介
  - ◆『IoTへのRuby/mruby適用試行』(2015年 廣田哲也氏)
  - ◆『組込みハードウェアへのmrubyアプリケーション適用試行』  
(2016年 原嘉彦氏/三好秀徳)

---

## 2. 昨年実施内容の振り返りと今年取り組み内容

## 2-1 去年の実施内容の振り返り

- Raspberry Piに温度センサー、GPSセンサーをUSB接続
- Raspberry Pi上で動作するmrubyアプリがセンサーデータを取得して、クラウドにデータ送信・可視化





### ■ メリット

- 動作検証が容易

- ◆ Linuxが動くため、数多くのツールやmrbgemsが動作する
- ◆ センサーはUSB接続するだけで値を取得できる

### ■ デメリット

- 高スペックすぎる

- ◆ 消費電力が多い
- ◆ CRubyが動いてしまうため、mrubyを使う意味があまりない

- Raspberry Piよりも非力なハードで去年と同等のことはする
  - mrubyは動くが、CRubyは動かないハードウェアで実装する
    - ◆ 32bit CPU
    - ◆ 400KB程度のメモリ
  - センサーデータを読み込んで、外部に送信する

### ■ ESP-WROOM-32を選定

項目	値(ESP-WROOM-32)
CPU	Xtensa LX6 160~240 MHz 2コア
メモリ	520KB
Flash	内蔵(4MB)
Network	802.11 b/g/n + Bluetooth 4.2
USB 2.0	なし
消費電力	80mA
OS	FreeRTOS



### ■ 選定理由

- mrubyのフットプリントより大きいメモリを搭載している
- WiFi + Bluetooth内蔵
- 工事設計認証(技適)番号:211-161007

項目	値(Raspberry Pi 2 Model B)	値(ESP-WROOM-32)
CPU	ARM Cortex-A7/900MHz 4コア	Xtensa LX6 160~240 MHz 2コア
メモリ	1.0GB( <b>1,048,576KB</b> )	<b>520KB</b>
Flash	microSDカードに保存(GB単位)	内蔵(4MB)
Network	10/100Mbpsイーサネット	802.11 b/g/n + Bluetooth 4.2
USB 2.0	4口	なし
消費電流	<b>900mA</b>	<b>80mA</b>
OS	Linux	FreeRTOS

- Raspberry Piと比較すると、非常に低スペック
  - CRubyは当然動かない
- Linuxが動かないため、ツールや昨年使用したmrbgemsの動作は期待できない
  - センサーはつなげるだけでは動作しない
  - ライブラリー不足のため、mrbgemsのコンパイルができない

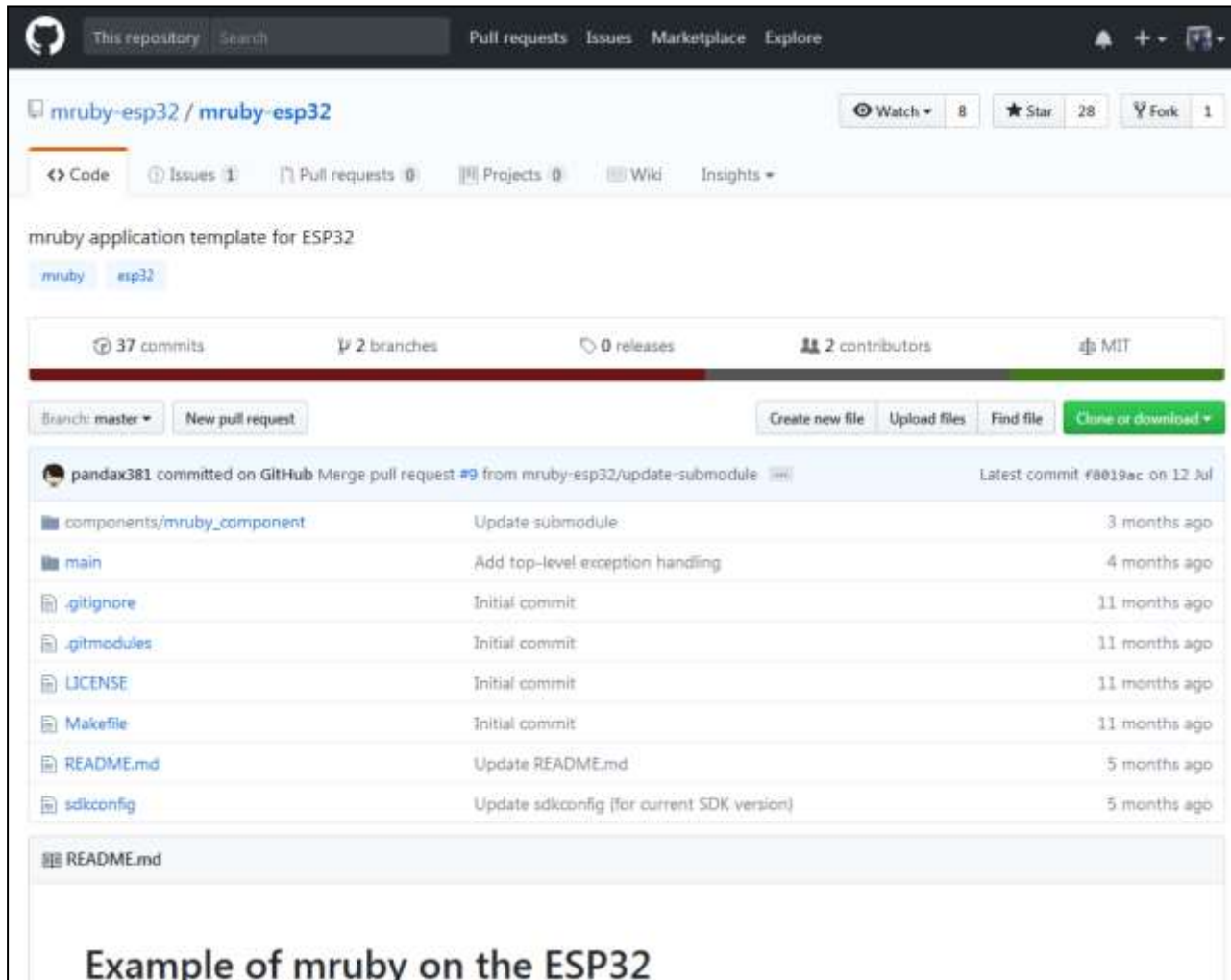
---

## 3.組み込みハードウェアモジュール利用時のmruby適用

1. mrubyアプリをESP-WROOM-32上で動かす
2. 各種センサーデータを取得してmruby上で扱う
  - ① 温度センサーの場合
  - ② GPSセンサーの場合

1. mrubyアプリをESP-WROOM-32上で動かす
2. 各種センサーデータを取得してmruby上で扱う
  - ① 温度センサーの場合
  - ② GPSセンサーの場合

## 3-2 mruby-esp32のご紹介



mruby-esp32 / mruby-esp32

37 commits 2 branches 0 releases 2 contributors MIT

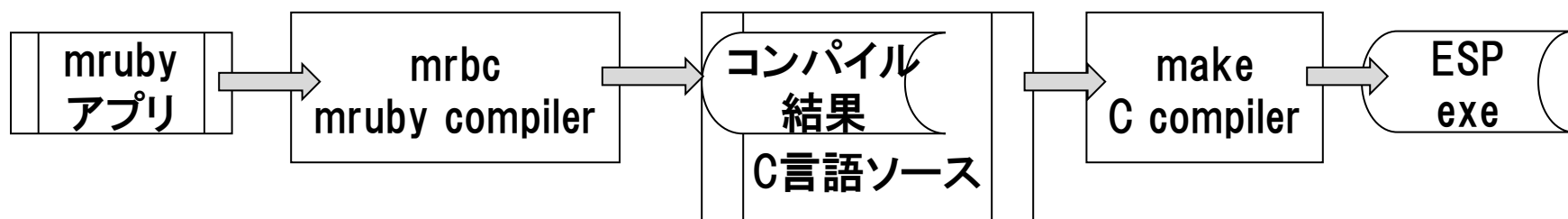
File	Commit Message	Time Ago
components/mruby_component	Update submodule	3 months ago
main	Add top-level exception handling	4 months ago
.gitignore	Initial commit	11 months ago
.gitmodules	Initial commit	11 months ago
LICENSE	Initial commit	11 months ago
Makefile	Initial commit	11 months ago
README.md	Update README.md	5 months ago
sdkconfig	Update sdkconfig (for current SDK version)	5 months ago

Example of mruby on the ESP32

<https://github.com/mruby-esp32/mruby-esp32>



- ESP-WROOM-32上でmrubyを動かすプロジェクト
- 動作原理は単純
  1. mrubyアプリをコンパイルし、C言語の配列として出力する
  2. コンパイル結果をESP-WROOM-32用アプリのC言語ソースで読み込む
  3. C言語ソースをコンパイルしてESP-WROOM-32用バイナリを作成する
- mrubyに手を加えていないので、mruby自身のバージョンアップも容易



# 3-4 mruby-esp32の動作説明(1)

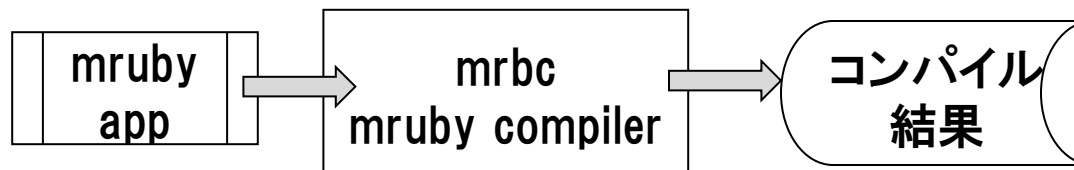
```
100.times do  
  puts 'Hello mruby World!'  
end
```

mruby app

mrbc  
mruby compiler

コンパイル結果

```
#include <stdint.h>  
extern const uint8_t out[];  
(snip)  
out[] = {  
0x45,0x54,0x49,0x52,0x30,0x30,0x30,0x34,0xc5,0x77,0x00,0x00,0x00,0x9c,  
0x4d,0x41,  
(snip)  
0x75,0x74,0x73,0x00,0x45,0x4e,0x44,0x00,0x00,0x00,0x00,0x08,  
};
```



## 3-5 mruby-esp32の動作説明(2)

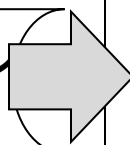
コンパイル結果

```
#include <stdint.h>
extern const uint8_t out[];
(snip)
out[] = {
0x41, 0x54, 0x49, 0x52, 0x30, 0x30, 0x30, 0x34, 0xc5, 0x77, 0x00, 0x00, 0x00, 0x9c, 0x4d,
(snip)
0x75, 0x73, 0x00, 0x45, 0x4e, 0x44, 0x00, 0x00, 0x00, 0x00, 0x08,
};
```

C言語ソース

```
void mruby_task(void *pvParameter){
    mrb_state *mrb = mrb_open();
    mrbc_context *context = mrbc_context_new(mrb);
    mrb_load_irep_cxt(mrb, out, context);
(snip)
```

コンパイル  
結果



C言語ソース

## 3-6 mruby-esp32の動作説明(3)

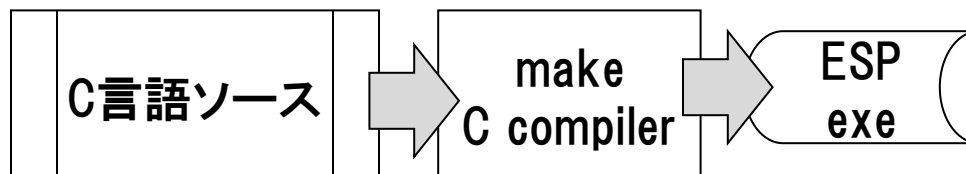
```
$ make MRUBY_EXAMPLE=hello_mruby_world.rb flash monitor
```

コンパイル

make  
C compiler

実行

```
CC mruby_main.o  
AR libmain.a  
LD mruby_example.elf  
esptool.py v2.0-beta3  
Flashing binaries to serial port /dev/cu.SLAB_USBtoUART (app at offset 0x10000)...  
esptool.py v2.0-beta3  
(snip)  
I (1372) cpu_start: Pro cpu start user code  
I (1431) cpu_start: Starting scheduler on PRO CPU.  
I (1951) mruby_task: Loading binary...  
Hello mruby world!  
Hello mruby world!  
Hello mruby world!  
(snip)
```

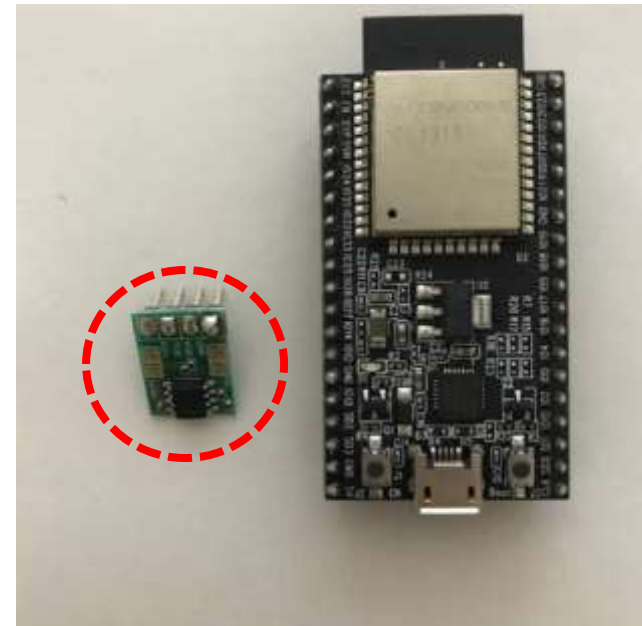
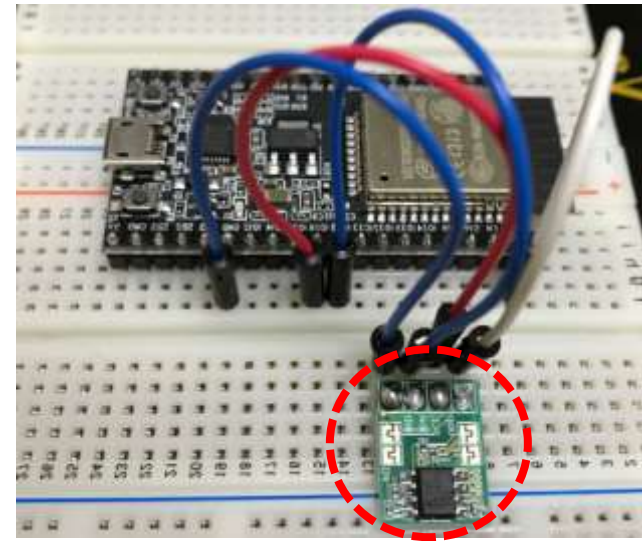


---

## 4. センサーデータの取得とその利用

1. mrubyアプリをESP-WROOM-32上で動かす
2. 各種センサーデータを取得してmruby上で扱う
  - ① 温度センサーの場合
  - ② GPSセンサーの場合

- ANALOG DEVICES社製温度センサー
  - I<sup>2</sup>C(アイ・スクエアド・シー)互換インタフェースを持ち、I<sup>2</sup>Cバスに測定温度データを出力
  - I<sup>2</sup>Cバスに出力される計測温度データに対してビット演算などを行うことで摂氏データを取得することができる



### ■ I<sup>2</sup>C(アイ・スクエアド・シー)通信でデータを取得する必要がある

- Raspberry Piではi2c-devを設定ファイルに追加するだけ
  - ◆ 既存のmrbgemsはLinux上での動作を前提としていて使えない
- ESP-WROOM-32ではC言語APIを使ってI<sup>2</sup>C通信でデータを取得できる

### ■ そこでmrbgemsを作りました

- [https://github.com/miyohide/mruby-esp32-i2c/tree/add\\_read](https://github.com/miyohide/mruby-esp32-i2c/tree/add_read)
- I<sup>2</sup>C通信でデータを取得する部分だけを実装
- 得られたデータを加工・編集する処理はmrubyアプリ側で実装



## 4-4 作成したmrbgemsの概要

mrbgems

```
static mrb_value  
mrb_esp32_i2c_init(mrb_state *mrb, mrb_value self)  
{  
  (省略)  
}
```

I<sup>2</sup>C通信関連の処理を  
C言語で実装

```
static mrb_value  
mrb_esp32_i2c_receive(mrb_state *mrb, mrb_value self)  
{  
  (省略)  
}
```

C言語の関数と  
mrubyのメソッドとを  
紐付け

```
void  
mrb_mruby_esp32_i2c_gem_init(mrb_state* mrb)  
{
```

```
  mrb_define_method(mrb, i2c, "_init", mrb_esp32_i2c_init, MRB_ARGS_REQ(7));  
  mrb_define_method(mrb, i2c, "receive", mrb_esp32_i2c_receive, MRB_ARGS_REQ(2));  
  (以下略)  
}
```

```
class ADT7410
  def receive
    word_data = @i2c.receive(2, @addr)
    first_bit = word_data[0].unpack("C*").first
    second_bit = word_data[1].unpack("C*").first
    ((first_bit << 8 | second_bit) >> 3) * 0.0625
  end
end
```

I<sup>2</sup>C通信の汎用的な部分を  
mrbgems化

```
i2c = I2C.new(I2C::PORT0, scl: I2C::SCL1, sda: I2C::SDA1).init(I2C::MASTER)
```

```
adt = ADT7410.new(i2c)
# 初期化処理等は省略
100.times do
  puts "#{Time.now} temp=#{adt.receive}"
  ESP32::System.delay(1000)
end
```

## 4-6 mrubyアプリの概要(2)

mrubyアプリ

```
class ADT7410
```

```
  def receive
```

```
    word_data = @i2c.receive(2, @addr)
```

```
    first_bit = word_data[0].unpack("C*").first
```

```
    second_bit = word_data[1].unpack("C*").first
```

```
    ((first_bit << 8 | second_bit) >> 3) * 0.0625
```

```
  end
```

```
end
```

```
i2c = I2C.new(I2C::PORT0, scl: I2C::SCL1, sda: I2C::SDA1).init(I2C::MASTER)
```

```
adt = ADT7410.new(i2c)
```

```
# 初期化処理等は省略
```

```
100.times do
```

```
  puts "#{Time.now} temp=#{adt.receive}"
```

```
  ESP32::System.delay(1000)
```

```
end
```

センサー固有の処理は  
mrubyで実装

## 4-7 温度センサーからデータを取得している例

(省略)

I (1551) cpu\_start: Pro cpu start user code

I (1610) cpu\_start: Starting scheduler on PRO CPU.

I (2202) mruby\_task: Loading binary...

Thu Jan 01 00:00:00 1970 temp=27.4375

Thu Jan 01 00:00:01 1970 temp=27.4375

Thu Jan 01 00:00:02 1970 temp=27.4375

Thu Jan 01 00:00:03 1970 temp=27.4375

Thu Jan 01 00:00:04 1970 temp=27.375

Thu Jan 01 00:00:05 1970 temp=27.4375

Thu Jan 01 00:00:06 1970 temp=27.4375

Thu Jan 01 00:00:07 1970 temp=27.4375

Thu Jan 01 00:00:08 1970 temp=27.375

Thu Jan 01 00:00:09 1970 temp=27.4375

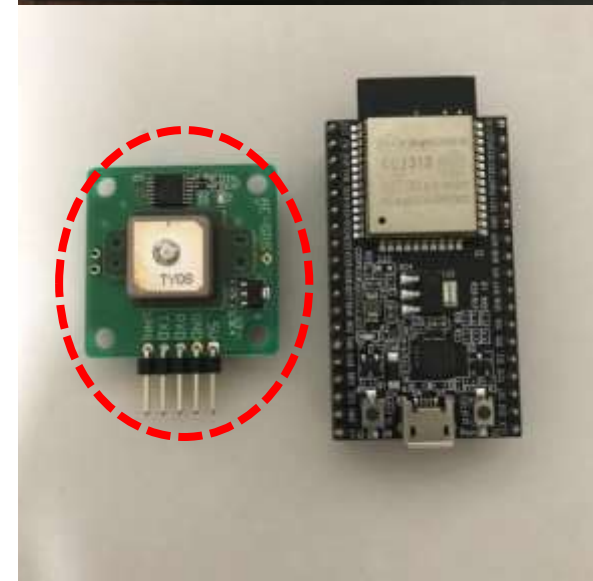
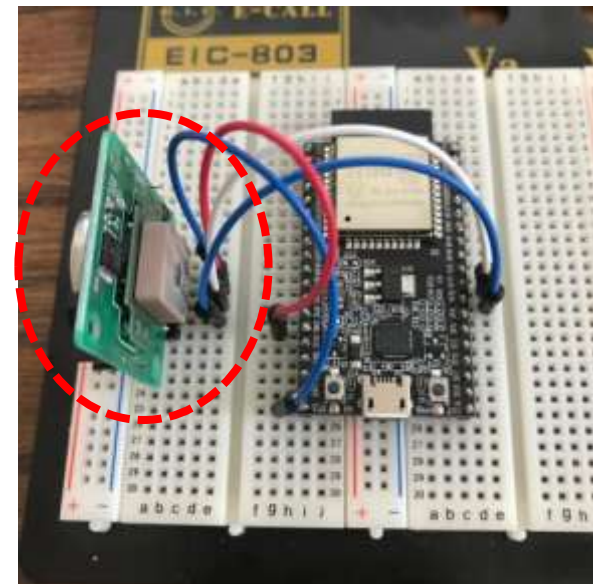
Thu Jan 01 00:00:10 1970 temp=27.4375

温度が正しく取れている

1. mrubyアプリをESP-WROOM-32上で動かす
2. 各種センサーデータを取得してmruby上で扱う
  - ① 温度センサーの場合
  - ② GPSセンサーの場合

### ■ 太陽誘電社製 GPSセンサー

- UARTシリアル通信にてGPSデータ (NMEA0183形式)を取得可能
- NMEA0183形式で得られたデータを加工・編集することで測定位置の緯度・経度データを取得することができる



- UARTシリアル通信でデータを読み込む必要がある
  - Raspberry Piでは配線するだけで/dev/ttyAMA0にデータが出力される
    - 既存のmrbgemsはLinux上での動作を前提としているため使えない
  - ESP-WROOM-32ではC言語APIが提供されている
  
- そこでmrbgemsを作成しました
  - <https://github.com/miyohide/mruby-esp32-gps>
  - UARTシリアル通信でデータを取得する部分のみを実装
  - 得られたデータを加工するのはmrubyアプリにて実装する

## 4-11 作成したmrbgemsの概要(UART編)

mrbgems

```
static mrb_value  
mrb_esp32_gps_init(mrb_state *mrb, mrb_value self) {  
(省略)  
}
```

```
static mrb_value  
mrb_esp32_gps_dogps(mrb_state *mrb, mrb_value self) {  
(省略)  
}
```

```
void mrb_mruby_esp32_gps_gem_init(mrb_state* mrb) {  
(省略)
```

```
    mrb_define_module_function(mrb, gps, "init", mrb_esp32_gps_init,  
        MRB_ARGS_NONE());  
    mrb_define_module_function(mrb, gps, "doGPS", mrb_esp32_gps_dogps,  
        MRB_ARGS_NONE());  
}
```

UARTシリアル通信関連の  
処理をC言語で実装

C言語の関数と  
mrubyのメソッドとを  
紐付け



mrubyアプリ

```
include ESP32::GPS
```

```
init
```

```
1000.times {  
  puts doGPS  
}
```

mrbgems化した  
UARTシリアル通信の部分を  
呼び出し

## 4-13 GPSセンサーから値(NMEA0183形式)を取得

```
I (1950) uart: queue free spaces: 10
$GPGGA,044241.000,xxxx.xxxx,N,yyyy.yyyy,E,1,5,2.56,105.7,M,39.5,M,,*5B
$GPGSA,A,3,193,06,19,17,09,,,,,,,,,2.74,2.56,0.97*36
$GPRMC,044241.000,A,xxxx.xxxx,N,yyyy.yyyy,E,0.37,98.78,081017,,,A*54
$GPZDA,044241.000,08,10,2017,,*5C
$GPGGA,044242.000,xxxx.xxxx,N,yyyy.yyyy,E,1,5,2.56,105.7,M,39.5,M,,*5A
$GPGSA,A,3,193,06,19,17,09,,,,,,,,,2.74,2.56,0.97*36
$GPRMC,044242.000,A,xxxx.xxxx,N,yyyy.yyyy,E,0.30,98.78,081017,,,A*52
$GPZDA,044242.000,08,10,2017,,*5F
$GPGGA,044243.000,xxxx.xxxx,N,yyyy.yyyy,E,1,5,2.56,105.7,M,39.5,M,,*58
$GPGSA,A,3,193,06,19,17,09,,,,,,,,,2.74,2.56,0.97*36
$GPRMC,044243.000,A,xxxx.xxxx,N,yyyy.yyyy,E,0.27,98.78,081017,,,A*56
$GPZDA,044243.000,08,10,2017,,*5E
$GPGGA,044244.000,xxxx.xxxx,N,yyyy.yyyy,E,1,5,2.56,105.7,M,39.5,M,,*5E
$GPGSA,A,3,193,06,19,17,09,,,,,,,,,2.74,2.56,0.97*36
$GPGSV,4,1,13,193,83,295,23,06,71,081,27,00,57,321,00,42,051,02,*46
$GPGSV,4,2,13,19,36,181,25,05,28,270,,07,19,107,2
$GPGSV,4,3,13,23,15,042,,30,12,139,,13,06,205,,12,
$GPGSV,4,4,13,29,03,327,*45
```

緯度・経度が取れている  
(xxxx.xxxx / yyyy.yyyyの部分)

```
include ESP32::GPS  
init
```

```
n = NMEA.new
```

```
1000.times {  
  n.parse(doGPS)  
}
```

UARTシリアル通信の結果を  
加工・編集して出力

```
class NMEA  
  def parse  
    (省略)  
  end  
end
```

NMEA0183形式の文字列を  
加工・編集して、日時・緯度・  
経度を求める。  
(実際はmrbgems化)

```
l (1969) uart: queue free spaces: 10
2017/10/08 04:48:00 lat = Nxx.xxxxx , lon = Eyyy.yyyyy
2017/10/08 04:48:01 lat = Nxx.xxxxx , lon = Eyyy.yyyyy
2017/10/08 04:48:01 lat = Nxx.xxxxx , lon = Eyyy.yyyyy
2017/10/08 04:48:02 lat = Nxx.xxxxx , lon = Eyyy.yyyyy
2017/10/08 04:48:02 lat = Nxx.xxxxx , lon = Eyyy.yyyyy
2017/10/08 04:48:03 lat = Nxx.xxxxx , lon = Eyyy.yyyyy
2017/10/08 04:48:03 lat = Nxx.xxxxx , lon = Eyyy.yyyyy
2017/10/08 04:48:04 lat = Nxx.xxxxx , lon = Eyyy.yyyyy
2017/10/08 04:48:04 lat = Nxx.xxxxx , lon = Eyyy.yyyyy
2017/10/08 04:48:05 lat = Nxx.xxxxx , lon = Eyyy.yyyyy
2017/10/08 04:48:05 lat = Nxx.xxxxx , lon = Eyyy.yyyyy
2017/10/08 04:48:06 lat = Nxx.xxxxx , lon = Eyyy.yyyyy
2017/10/08 04:48:06 lat = Nxx.xxxxx , lon = Eyyy.yyyyy
2017/10/08 04:48:07 lat = Nxx.xxxxx , lon = Eyyy.yyyyy
2017/10/08 04:48:07 lat = Nxx.xxxxx , lon = Eyyy.yyyyy
2017/10/08 04:48:08 lat = Nxx.xxxxx , lon = Eyyy.yyyyy
2017/10/08 04:48:08 lat = Nxx.xxxxx , lon = Eyyy.yyyyy
2017/10/08 04:48:09 lat = Nxx.xxxxx , lon = Eyyy.yyyyy
```

NMEA0183形式から  
日時・緯度・経度データ  
があるものだけを抜き出して  
出力

---

## 5. まとめと今後の課題

- ESP-WROOM-32上でmrubyアプリを動かすことに成功しました
- 各種センサーからのデータをmrubyアプリから取得できる  
mrbgemsを作成しました
- mrbgemsを作成するときに役立つ資料
  - mrubyのGitHubリポジトリにあるdocディレクトリ
    - <https://github.com/mruby/mruby/blob/master/doc/guides/mrbgems.md>
  - CRubyのGitHubリポジトリにある「Rubyの拡張ライブラリの作り方」
    - <https://github.com/ruby/ruby/blob/trunk/doc/extension.ja.rdoc>
  - mruby-mrbgem-template
    - <https://github.com/matsumotory/mruby-mrbgem-template>

### ■ 昨年と同等のことはまだできていない(https通信)

- ライブラリー不足によりコンパイルに失敗する
- ESP-WROOM-32の機能を使って何とか解決できないか調査中

```
(省略)
CC build/mrbgems/mruby-polarssl/src/polarssl.c -> build/esp32/mrbgems/mruby-polarssl/src/polarssl.o
/Users/miyohide/work/mruby-esp32/components/mruby_component/mruby/build/mrbgems/mruby-
polarssl/src/polarssl.c:20:23: fatal error: sys/ioctl.h: No such file or directory
compilation terminated.
/Users/miyohide/work/mruby-compilation terminated.
rake aborted!
Command Failed: (省略)
make[2]: *** [all] Error 1
make[1]: *** [build] Error 2
make: *** [mruby_component-build] Error 2
```

- ハードウェアごとに製品仕様を調べて、mrbgemsを作成するのは大変
  - mruby IoTフレームワーク “Plato” (プラトン)



<http://plato.click/>



**END**



## 組込みハードウェアモジュールへの mrubyアプリケーション適用試行

2017/11/01

株式会社 日立ソリューションズ  
業務革新統括本部 IT技術推進センター

三好 秀徳

活文は、株式会社日立ソリューションズの登録商標です。

LinuxはLinus Torvalds氏の米国、日本およびその他の国における登録商標または商標です。

Raspberry Piは、英国Raspberry Pi財団の米国およびその他の国における商標または登録商標です。

Cortexは、ARM Limitedの米国およびその他の国における商標または登録商標です。

その他、記載されている会社名、商品名は、各社の登録商標または商標です。

**HITACHI**  
**Inspire the Next**