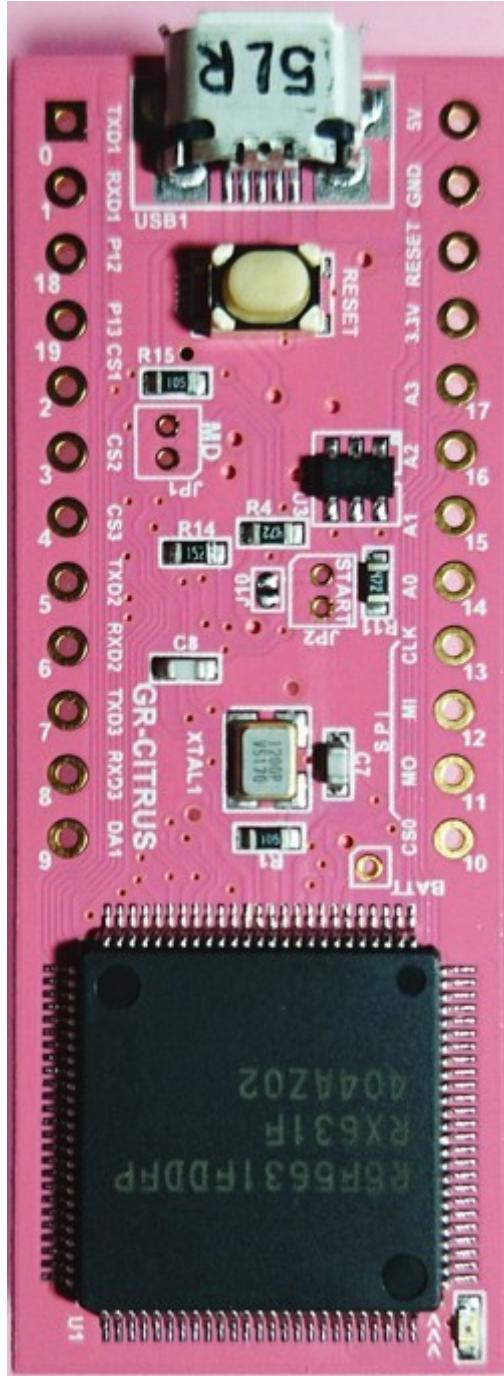


GR-CITRUS 搭載  
Ruby ファーム  
説明資料 ver1.9

Wakayama. rb  
山本三七男(たろサ)

# ハード仕様



## MCU

32ビットCPU RX631(100ピン)

96MHz

FlashROM : 2Mバイト

RAM : 256Kバイト

データ用Flash : 32Kバイト

## ボード機能

USBファンクション端子 (micro-B)

LED 1個

I/Oピン 20ピン

シリアル 6個 (+1個可能)

SPI 1個

A/D 4個

RTC

I2C 4個 (+1個可能)

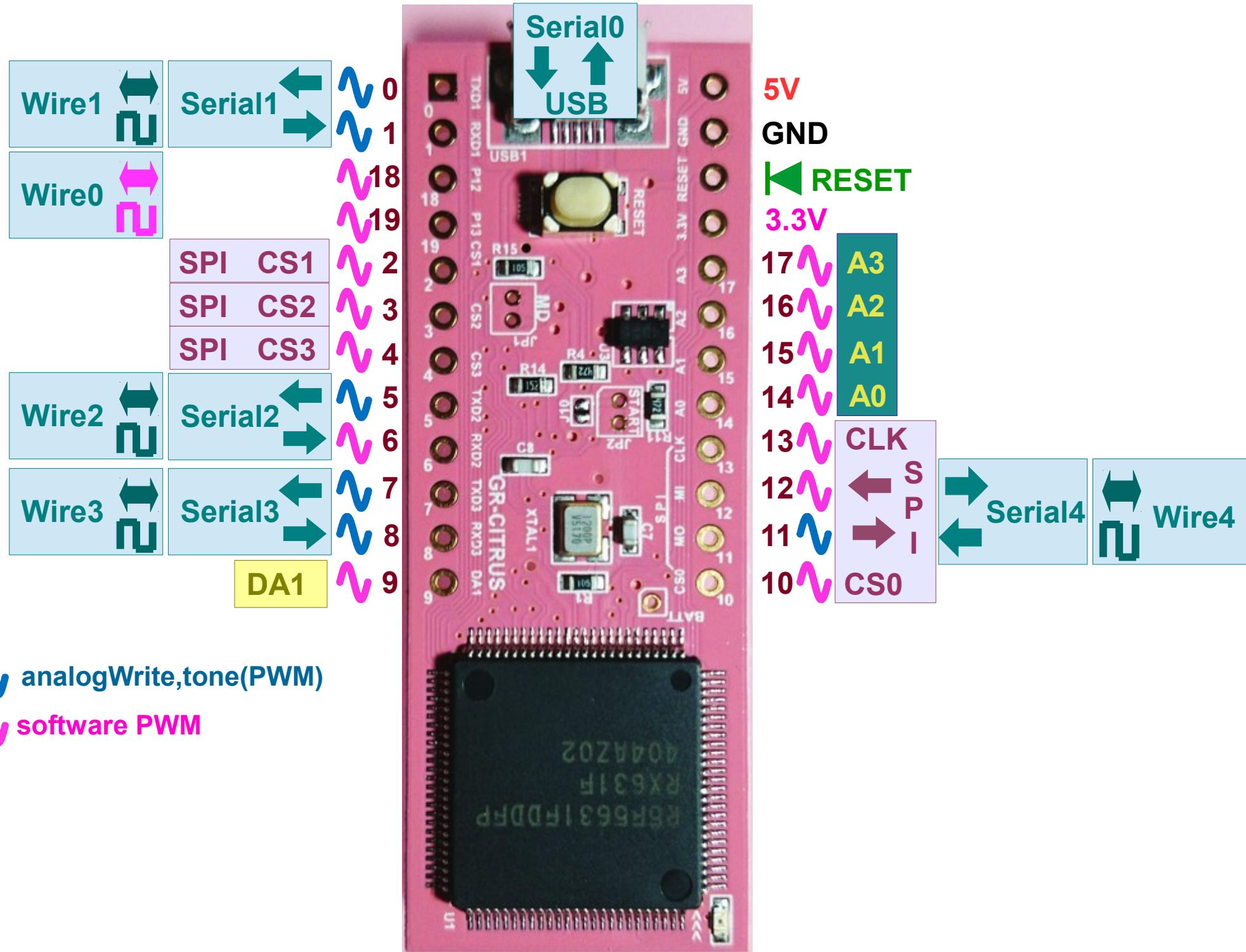
PWM、Servoは自由割当てです。

## 電 源

5V (USBバスパワード)

## サ イ ズ

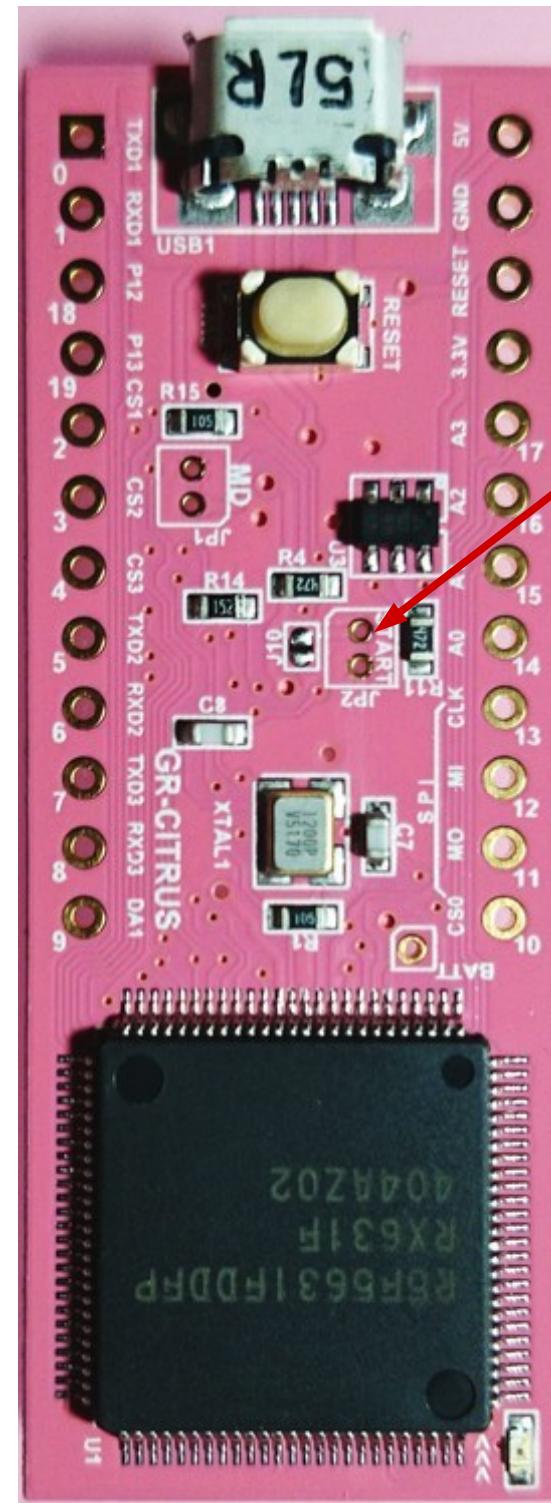
52 × 20mm



## RX631 ピン番号

	P20
0	P21
1	P12
18	P15
19	P13
2	P31
3	PC0
4	P30
5	PC1
6	PC2
7	P25
8	P34
9	P50
10	P55
11	P52
12	P32
13	P33
14	P26
15	P05

赤文字ピン番号は  
5Vトレント



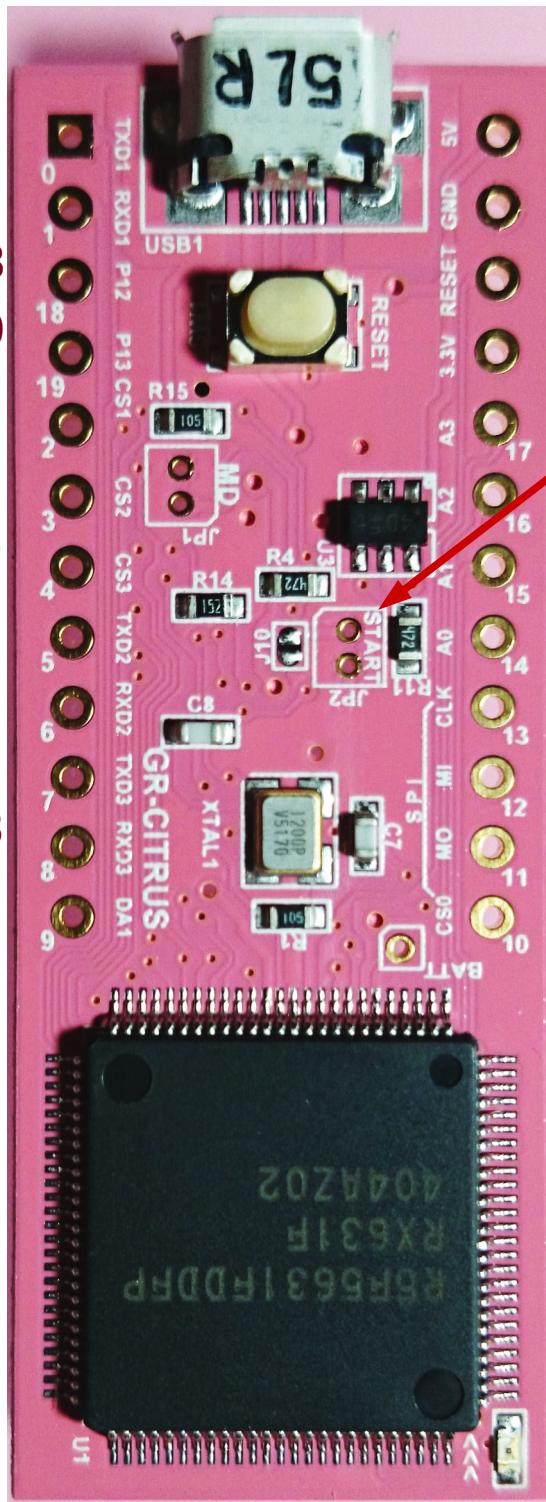
5V	
GND	
RESET	▲
3.3V	
17	P43
16	PE1
15	P42
16	PB5
15	P41
15	PB3
14	P40
14	P27
13	PC5
12	PC7
11	PC6
10	PC4
P35	
NMI	

## RX631 ピン番号

# GR-SAKURAとのピン対比

1	
0	
30	
33	31
TMS	22
TDI	23
8	
5	TRST
29	26
6	
7	
TDO	53

P20	0
P21	1
P12	18
P15	19
P13	
P31	2
PC0	
P30	3
PC1	
PC2	4
P25	5
P34	6
P50	
P55	7
P52	
P32	8
P33	9
P26	
P05	

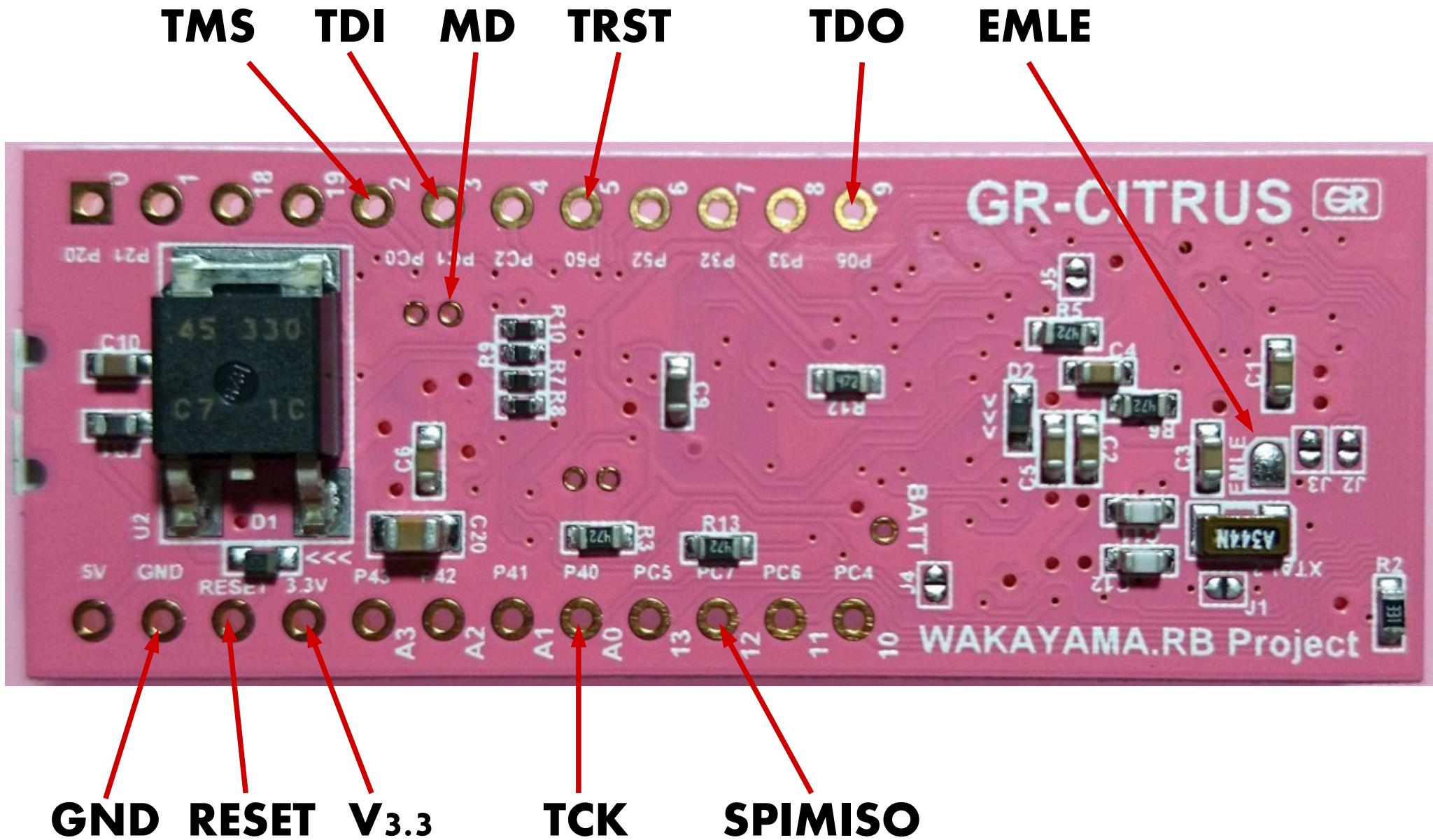


5V	
GND	
RESET	
3.3V	
P35	17
54	16
P43	15
PE1	14
P42	13
PB5	12
P41	11
PB3	10
P40	17
P27	16
PC5	15
14	14
TCK	13
12	12
PC7	11
11	11
PC6	10
10	10

GR-SAKURA割当番号

GR-SAKURA割当番号

# JTAGの端子



# ジャンパの説明

J4

P27(TCK)と14番を接続します。  
P40ともショートになります。

J5

P26と9番を接続します。

J1

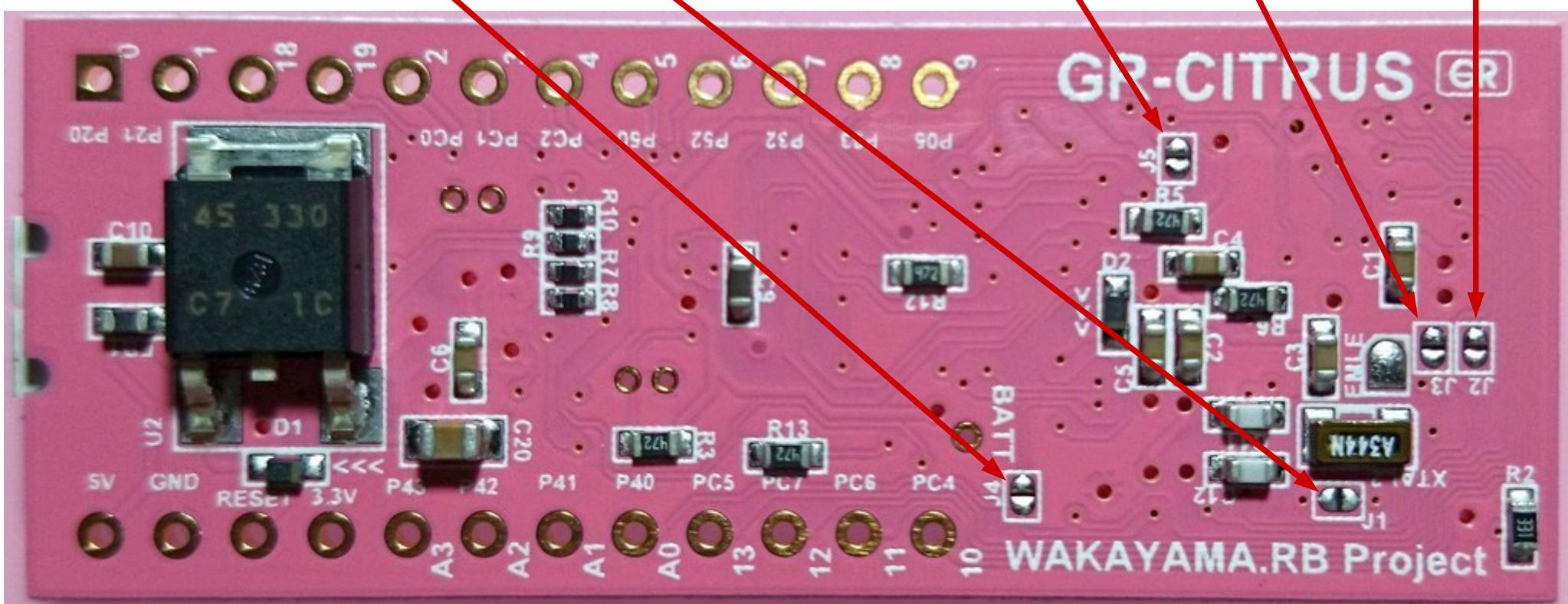
PE1と17番を接続します。  
P43ともショートになります。

J3

PB3と15番を接続します。  
P41ともショートになります。

J2

PB5と16番を接続します。  
P42ともショートになります。

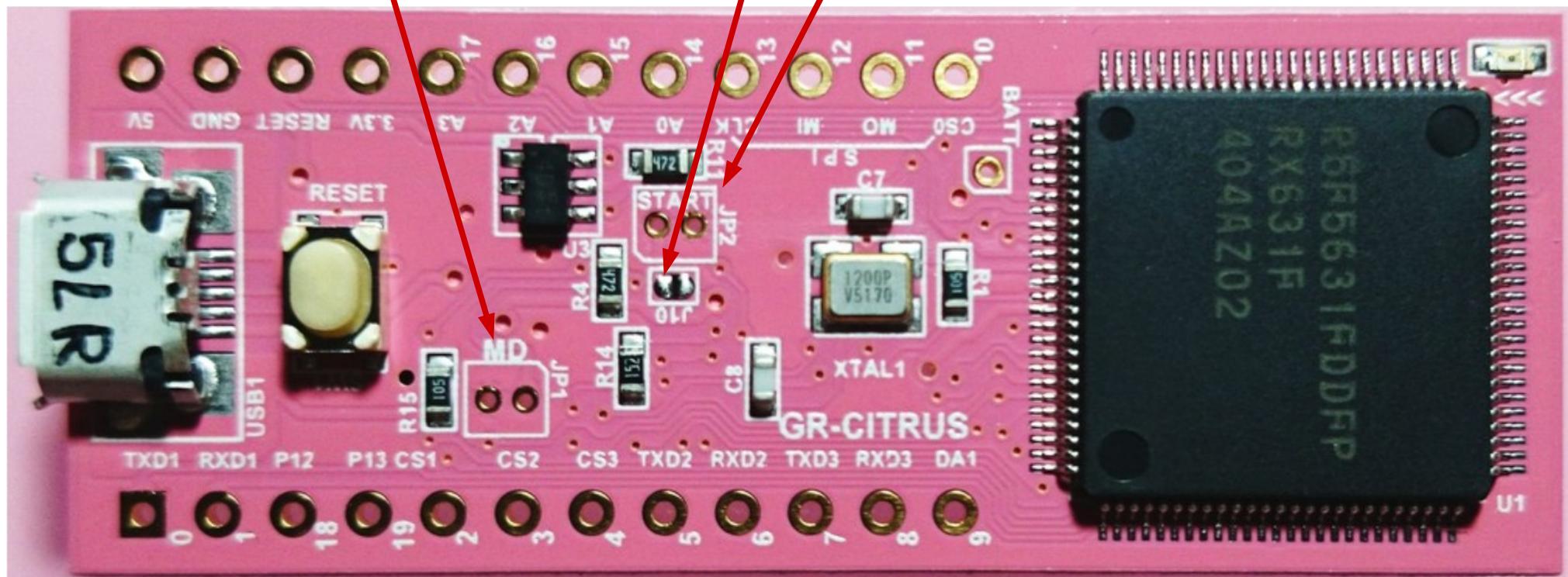


# ジャンパの説明

JP1  
MDをGNDに落とします。  
ファームを書き換えるときに、GND  
とショートさせます。

J10  
P35(NMI)をGNDに落とします。  
電源ON時にデフォルトでGNDにしたいとき  
に使用します。

JP2もJ10と同じです。



## Ruby ファーム仕様(バインドしているmrbgem)

```
mruby-math      #->Math.sin  
mruby-numeric-ext #->10.chr 0.zero?
```

# Ruby ファーム仕様 (V2ライブラリ)

## カーネルクラス

```
pinMode(pin, mode)
digitalWrite(pin, value)
digitalRead(pin)
analogRead(number)
pwm(pin, value)
analogReference(mode)
initDac()
analogDac(value)
delay(value)
millis()
micros()
led(sw)
tone(pin, freq[, duration])
noTone(pin)
randomSeed(value)
random([min, ] max)
```

## ファイルクラス

```
MemFile.open(number, filename[, mode])
MemFile.close(number)
MemFile.read(number)
MemFile.write(number, buf, len)
MemFile.seek(number, byte)
MemFile.cp(src, dst[, mode])
MemFile.rm(filename)
```

## シリアルクラス

```
Serial.new(number [, bps])
bps(bps)
print([str])
println([str])
available()
read()
write(buf, len)
flash()
```

## I2Cクラス

```
I2c.new(num)
write(deviceID, address, data)
read(deviceID, addL [, addH])
begin(deviceID)
lwrite(data)
end()
request(address, count)
lread()
available()
```

## サーボクラス

```
Servo.attach(ch, pin[, min, max])
Servo.write(ch, angle)
Servo.us(ch, us)
Servo.read(ch)
Servo.attached(ch)
Servo.attached?(ch)
Servo.detach(ch)
```

# Rubyファーム仕様(V2ライブラリ)

## システムクラス

**System.exit()**  
**System.setrun(filename)**  
**System.version([r])**  
**System.push(address, buf, length)**  
**System.pop(address, length)**  
**System.fileload()**  
**System.reset()**  
**System.useSD()**  
**System.useWiFi()**  
**System.useMP3(pausePin, stopPin)**  
**System.use(className[, options])**  
**System.use?(className[, options])**  
**System.getMrbPath()**

## リアルタイムクロッククラス

**Rtc.getTime()**  
**Rtc.setTime(array)**  
**Rtc.deinit()**  
**Rtc.init()**

## グローバル変数

ON	= 1
OFF	= 0
HIGH	= 1
LOW	= 0
OUTPUT	= 1
INPUT	= 0

## SDカードクラス

SD.exists(filename)  
SD.mkdir(dirname)  
SD.remove(filename)  
SD.copy(srcfilename, distfilename)  
SD.rmdir(dirname)  
SD.open(number, filename[, mode])  
SD.close(number)  
SD.read(number)  
SD.seek(number, byte)  
SD.write(number, buf, len)  
SD.flush(number)  
SD.size(number)  
SD.position(number)

## MP3クラス

MP3.play(filename)  
MP3.led(sw)

## WiFiクラス

```
WiFi.at(command[, mode])
WiFi.bypass()
WiFi.cClose(number)
WiFi.connect(SSID, Passwd)
WiFi.disconnect()
WiFi.httpGet(URL[, Headers])
WiFi.httpGetSD(Filename, URL[, Headers])
WiFi.httpPost(URL, Headers, Body)
WiFi.httpPostSD(URL, Headers, Filename)
WiFi.httpServer([Port])
WiFi.ipconfig()
WiFi.multiConnect(mode)
WiFi.recv(number)
WiFi.send(number, Data[, length])
WiFi.serialOut(mode[, serialNumber])
WiFi.setMode(mode)
WiFi.udpOpen(number, IP_Address, SendPort, ReceivePort)
WiFi.version()
```

# **rubyプログラムの実行**

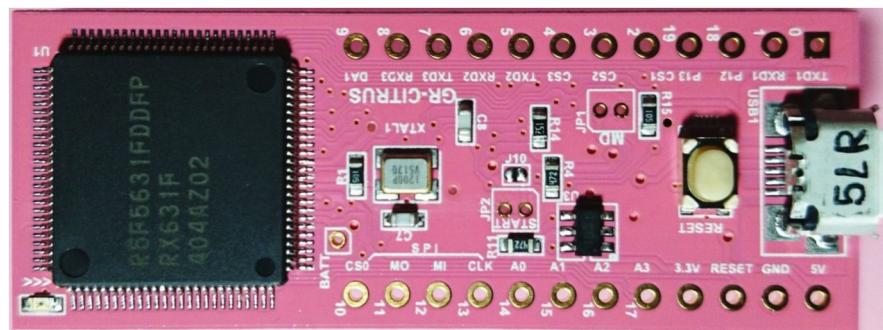
Rubyファームは、内部にrubyプログラムを保存できます。ファイル形式はmrbcによりコンパイルしたmrb形式のファイルとなります。

Rubyファームは、後述する「電源onde即実行するモード」に切り替わっていない限り、通常、電源をオンするとコマンドモードとなります。

# プログラムの書き込み

RubyファームはPCとUSB経由で接続し、シリアル通信を用いて通信します。

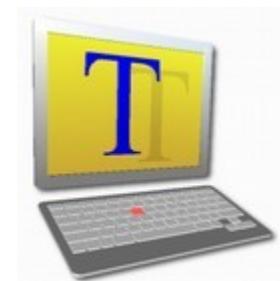
この通信を使って、Rubyのプログラムを書き込んだり、実行したり、RubyファームからデータをPCに出力したりします。



シリアル通信



CoolTerm

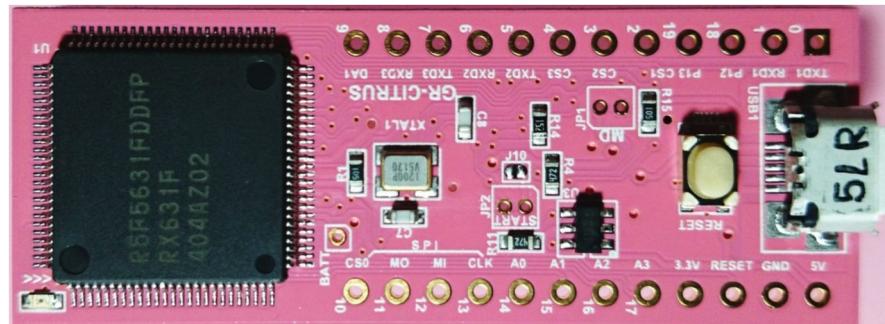


TeraTerm

シリアル通信には、ターミナルソフトを使います。  
代表的なものにTeraTermやCoolTermがあります。

# プログラムの書き込み

きむしゅさんが開発している「Rubic」を使用すると、Rubyファームに接続したまま楽にプログラム開発ができます。mrbファイルへのコンパイルも Rubicが行います。



Rubic

```

1  #!mruby
2  @M = MemFile
3
4  @M.copy("main.mrb", "hello.mrb")
5
6  Serial.begin(0, 115200)
7  k = 1
8  8.times do |n|
9    led k
10   k = 1-k
11   Serial.println(0, "#{n.to_s}:Hello WRB!")
12   delay 300
13 end
14 led 0
15 |

```

5:Hello WRB!  
6:Hello WRB!  
7:Hello WRB!

[Finish main.mrb]

Wakayama.rb Board ▾ COM3 ▾ ⓘ Info

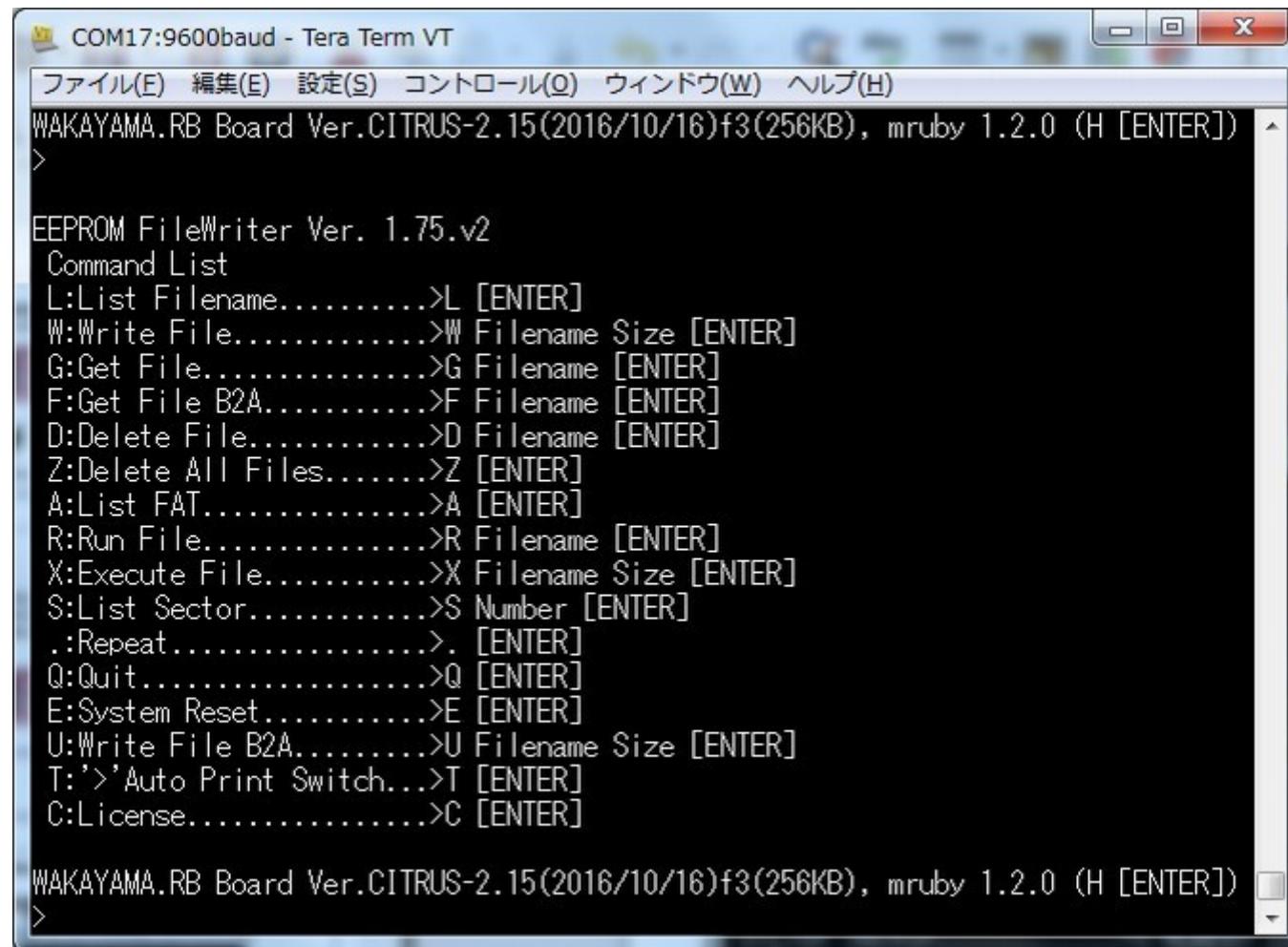
Hide Output

<https://github.com/kimushu/rubic>

# プログラムの書き込み方法

ターミナルソフトを用いてUSBからシリアル通信をしてプログラムを書き込みます。  
ENTERキーで画面にコマンド一覧が表示されます。

アルファベット1文字のコマンドを持っています。



Rubyファームの起動画面

# コマンドの種類

COM17:9600baud - Tera Term VT

WAKAYAMA.RB Board Ver.CITRUS-2.15(2016/10/16)f3(256KB), mruby 1.2.0 (H [ENTER])

>

EEPROM FileWriter Ver. 1.75.v2

Command List

L:List Filename.....	>L [ENTER]	L:保存しているファイルを一覧します。
W:Write File.....	>W Filename Size [ENTER]	W:ファイルを書き込みます。
G:Get File.....	>G Filename [ENTER]	G:ファイルを送信します。
F:Get File B2A.....	>F Filename [ENTER]	F:ファイル内容を16進数テキストで送信します。
D:Delete File.....	>D Filename [ENTER]	D:ファイルを削除します。
Z:Delete All Files.....	>Z [ENTER]	Z:全てのファイルを削除します。
A>List FAT.....	>A [ENTER]	A:セクタを一覧表示します。
R:Run File.....	>R Filename [ENTER]	R:実行ファイルをセットします。
X:Execute File.....	>X Filename Size [ENTER]	X:ファイルを書き込んで直ぐ実行します。
S>List Sector.....	>S Number [ENTER]	S:セクタ情報を表示します。
.:Repeat.....	>. [ENTER]	.:直前のコマンドを再実行します。
Q:Quit.....	>Q [ENTER]	Q:コマンドを終了します。
E:System Reset.....	>E [ENTER]	E:システムをリセットします。
U:Write File B2A.....	>U Filename Size [ENTER]	U:16進数テキストでデータを受信します。
T:'>'Auto Print Switch...>T [ENTER]		T:'>' の自動送信を切り替えます。
C:License.....	>C [ENTER]	C:ライセンスを表示します。

WAKAYAMA.RB Board Ver.CITRUS-2.15(2016/10/16)f3(256KB), mruby 1.2.0 (H [ENTER])

>

# プログラムを書き込みます。(W コマンド)

Wコマンドを用いて、mrbファイルを書き込みます。

Wの後にスペースで区切って、ファイル名とファイルサイズを書き、ENTERキーを押します。

>W ファイル名 ファイルサイズ

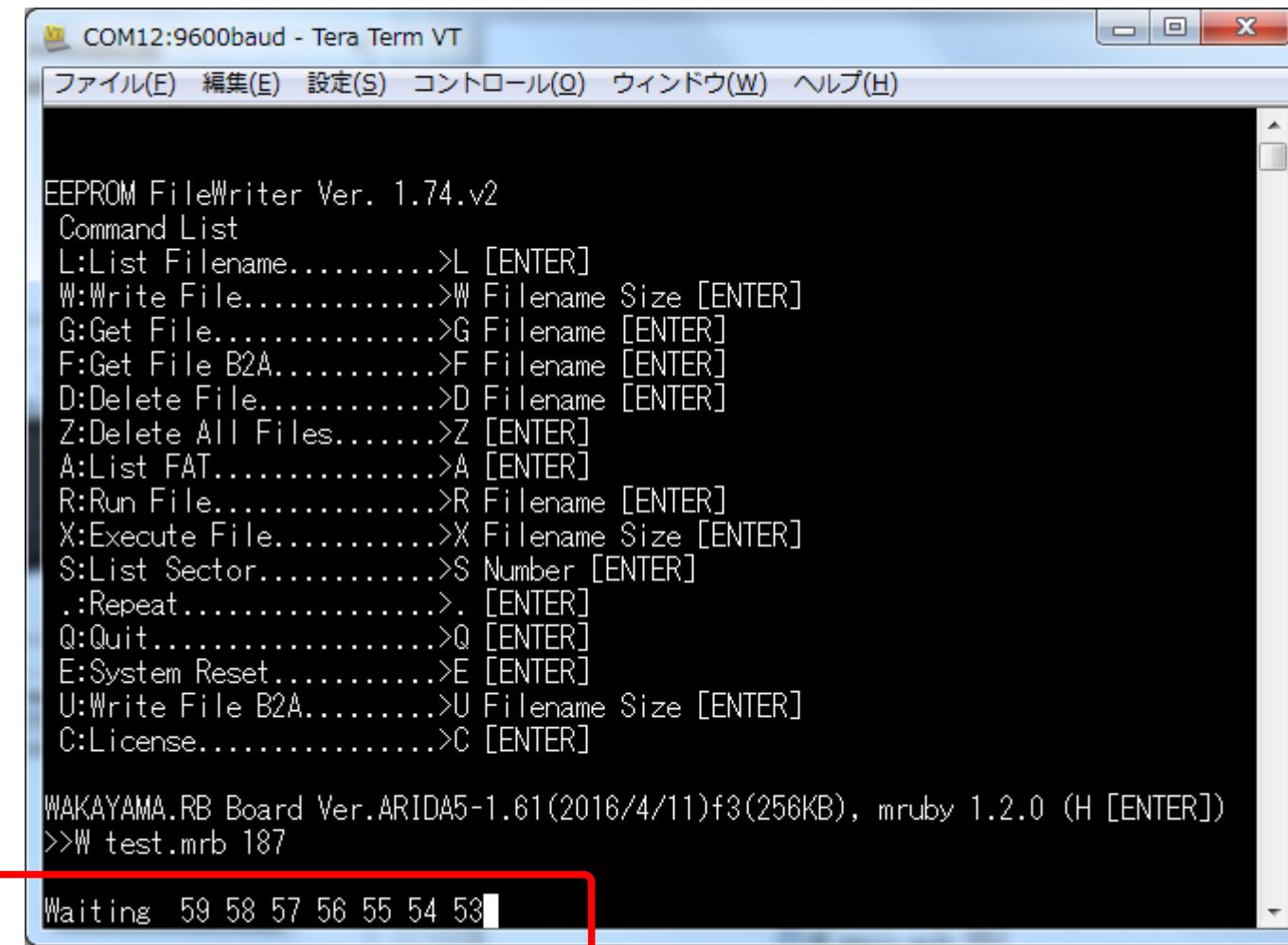
The screenshot shows a terminal window titled "COM12:9600baud - Tera Term VT". The window contains a command list for EEPROM FileWriter Ver. 1.74.v2. At the bottom, there is a red rectangular box highlighting the command "W test.mrb 187".

```
EEPROM FileWriter Ver. 1.74.v2
Command List
L:List Filename.....>L [ENTER]
W:Write File.....>W Filename Size [ENTER]
G:Get File.....>G Filename [ENTER]
F:Get File B2A.....>F Filename [ENTER]
D:Delete File.....>D Filename [ENTER]
Z:Delete All Files.....>Z [ENTER]
A:List FAT.....>A [ENTER]
R:Run File.....>R Filename [ENTER]
X:Execute File.....>X Filename Size [ENTER]
S:List Sector.....>S Number [ENTER]
.:Repeat.....>.[ENTER]
Q:Quit.....>Q [ENTER]
E:System Reset.....>E [ENTER]
U:Write File B2A.....>U Filename Size [ENTER]
C:License.....>C [ENTER]

WAKAYAMA.RB Board Ver.ARIDA5-1.61(2016/4/11)f3(256KB), mruby 1.2.0 (H [ENTER])
>>W test.mrb 187
```

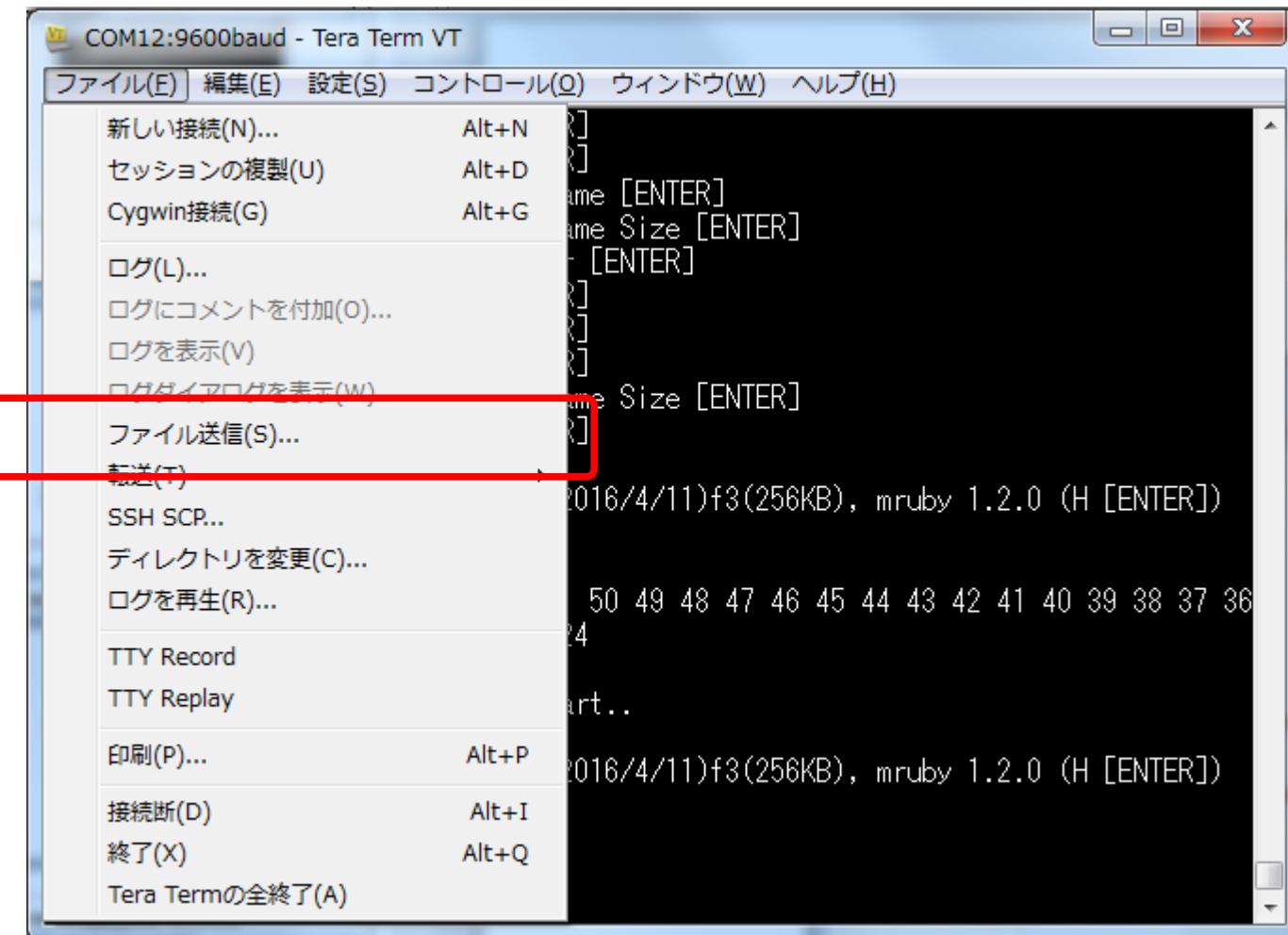
# プログラムを書き込みます。(W コマンド)

ENTERキーを押すと、カウントダウンが始まります。60sec以内にファイルをバイナリ送信してください。



# プログラムを書き込みます。(W コマンド)

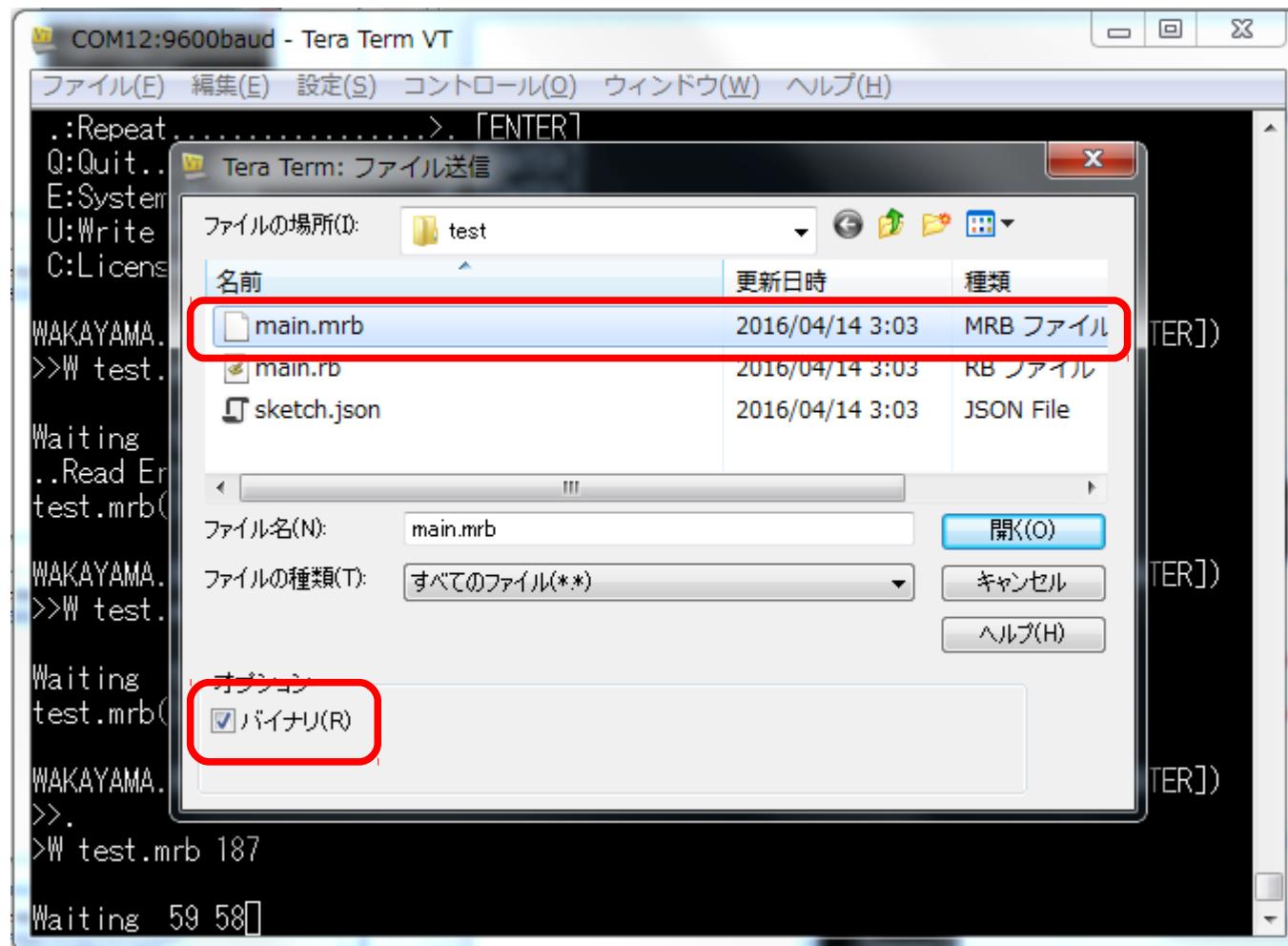
Tera Termの場合、ファイル→ファイル送信 を選択します。



# プログラムを書き込みます。(W コマンド)

Tera Termの場合、オプションのバイナリにチェックを入れます。

その後、送信するファイルを選択して、開く を押します。



# プログラムを書き込みます。(W コマンド)

ファイルの書き込みが終了すると、コマンド入力待ちに戻ります。

The screenshot shows a terminal window titled "COM12:9600baud - Tera Term VT". The menu bar includes "ファイル(F)", "編集(E)", "設定(S)", "コントロール(O)", "ウィンドウ(W)", and "ヘルプ(H)". The window displays the following text:

```
R:Run File.....>R Filename [ENTER]
X:Execute File.....>X Filename Size [ENTER]
S>List Sector.....>S Number [ENTER]
.:Repeat.....>.[ENTER]
Q:Quit.....>Q [ENTER]
E:System Reset.....>E [ENTER]
U:Write File B2A.....>U Filename Size [ENTER]
C:License.....>C [ENTER]

WAKAYAMA.RB Board Ver.ARIDA5-1.61(2016/4/11)f3(256KB), mruby 1.2.0 (H [ENTER])
>>W test.mrb 187

Waiting 59 58 57 56 55 54 53 52
..Read Error!
test.mrb(187) Saving the reading part..

WAKAYAMA.RB Board Ver.ARIDA5-1.61(2016/4/11)f3(256KB), mruby 1.2.0 (H [ENTER])
>>W test.mrb 187

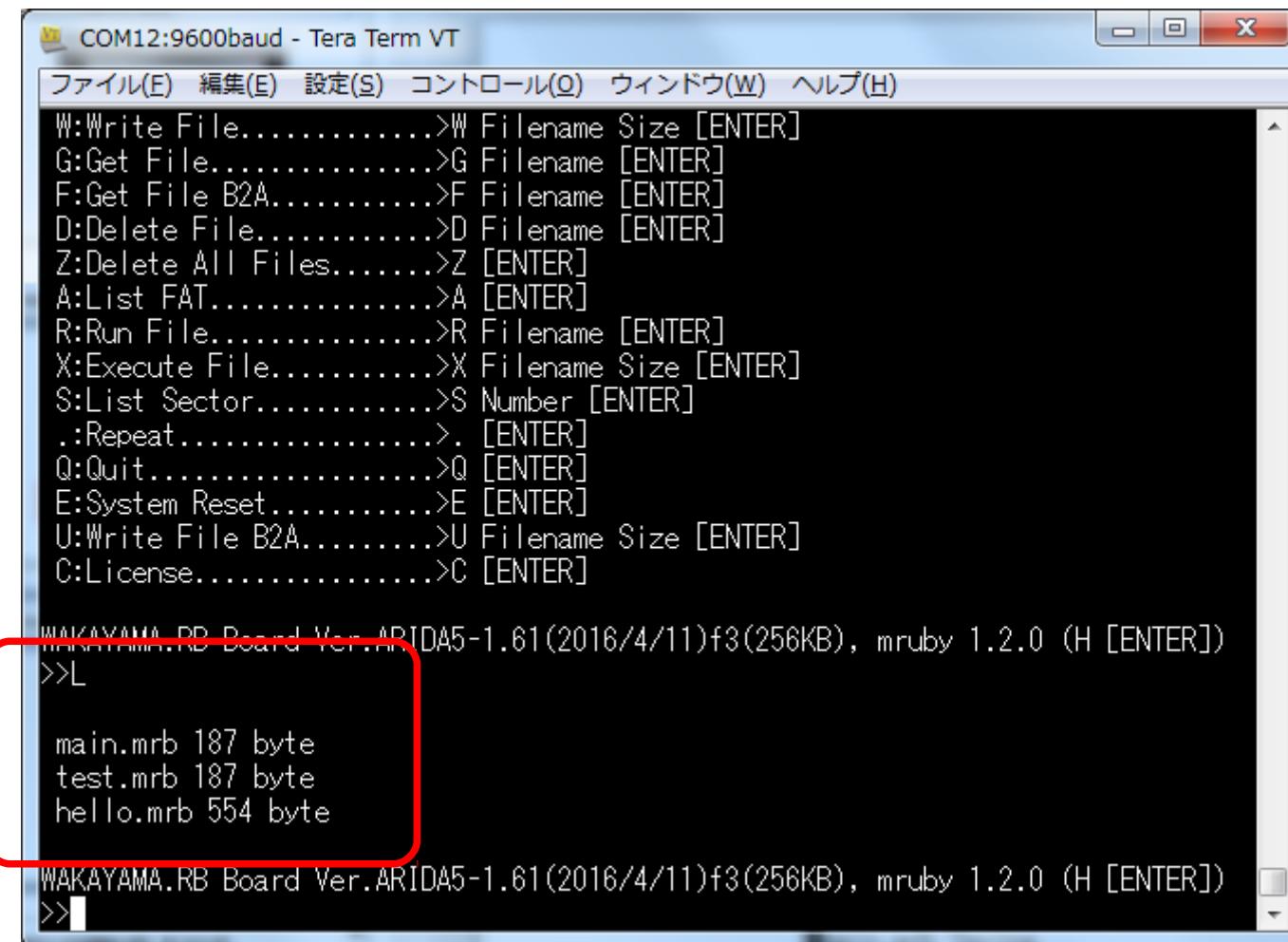
Waiting 59 58 57 56 55 54 53 52 51
test.mrb(187) Saving..

WAKAYAMA.RB Board Ver.ARIDA5-1.61(2016/4/11)f3(256KB), mruby 1.2.0 (H [ENTER])
>>
```

A red rectangle highlights the last three lines of the terminal output, specifically the "Saving.." message and the final prompt ">>".

# ファイルを一覧します。(L コマンド)

Lコマンドを入力すると保存されているファイルの一覧が表示されます。



The screenshot shows a terminal window titled "COM12:9600baud - Tera Term VT". The menu bar includes "ファイル(F)", "編集(E)", "設定(S)", "コントロール(O)", "ウィンドウ(W)", and "ヘルプ(H)". Below the menu is a list of commands:

- W:Write File.....>W F filename Size [ENTER]
- G:Get File.....>G F filename [ENTER]
- F:Get File B2A.....>F F filename [ENTER]
- D:Delete File.....>D F filename [ENTER]
- Z:Delete All Files.....>Z [ENTER]
- A>List FAT.....>A [ENTER]
- R:Run File.....>R F filename [ENTER]
- X:Execute File.....>X F filename Size [ENTER]
- S>List Sector.....>S Number [ENTER]
- .:Repeat.....>[ENTER]
- Q:Quit.....>Q [ENTER]
- E:System Reset.....>E [ENTER]
- U:Write File B2A.....>U F filename Size [ENTER]
- C:License.....>C [ENTER]

At the bottom of the terminal window, the text "WAKAYAMA.RB Board Ver.ARIDA5-1.61(2016/4/11)f3(256KB), mruby 1.2.0 (H [ENTER])" is displayed. Below this, the command ">>L" is entered. The output shows three files: "main.mrb 187 byte", "test.mrb 187 byte", and "hello.mrb 554 byte". The lines for "main.mrb", "test.mrb", and "hello.mrb" are highlighted with a red rectangle.

# mrbファイルを実行します。(R コマンド)

Rコマンドは、mrbファイルを実行することができます。

Rの後にスペースで区切って、実行させたいファイル名を書き、ENTERを押します。

.mrbは省略可能です。

>R ファイル名

The screenshot shows a terminal window titled "COM12:9600baud - Tera Term VT". The menu bar includes "ファイル(F)", "編集(E)", "設定(S)", "コントロール(Q)", "ウィンドウ(W)", and "ヘルプ(H)". Below the menu is a command list:

- W:Write File.....>W Filename Size [ENTER]
- G:Get File.....>G Filename [ENTER]
- F:Get File B2A.....>F Filename [ENTER]
- D:Delete File.....>D Filename [ENTER]
- Z:Delete All Files.....>Z [ENTER]
- A>List FAT.....>A [ENTER]
- R:Run File.....>R Filename [ENTER]
- X:Execute File.....>X Filename Size [ENTER]
- S>List Sector.....>S Number [ENTER]
- .:Repeat.....>. [ENTER]
- Q:Quit.....>Q [ENTER]
- E:System Reset.....>E [ENTER]
- U:Write File B2A.....>U Filename Size [ENTER]
- C:License.....>C [ENTER]

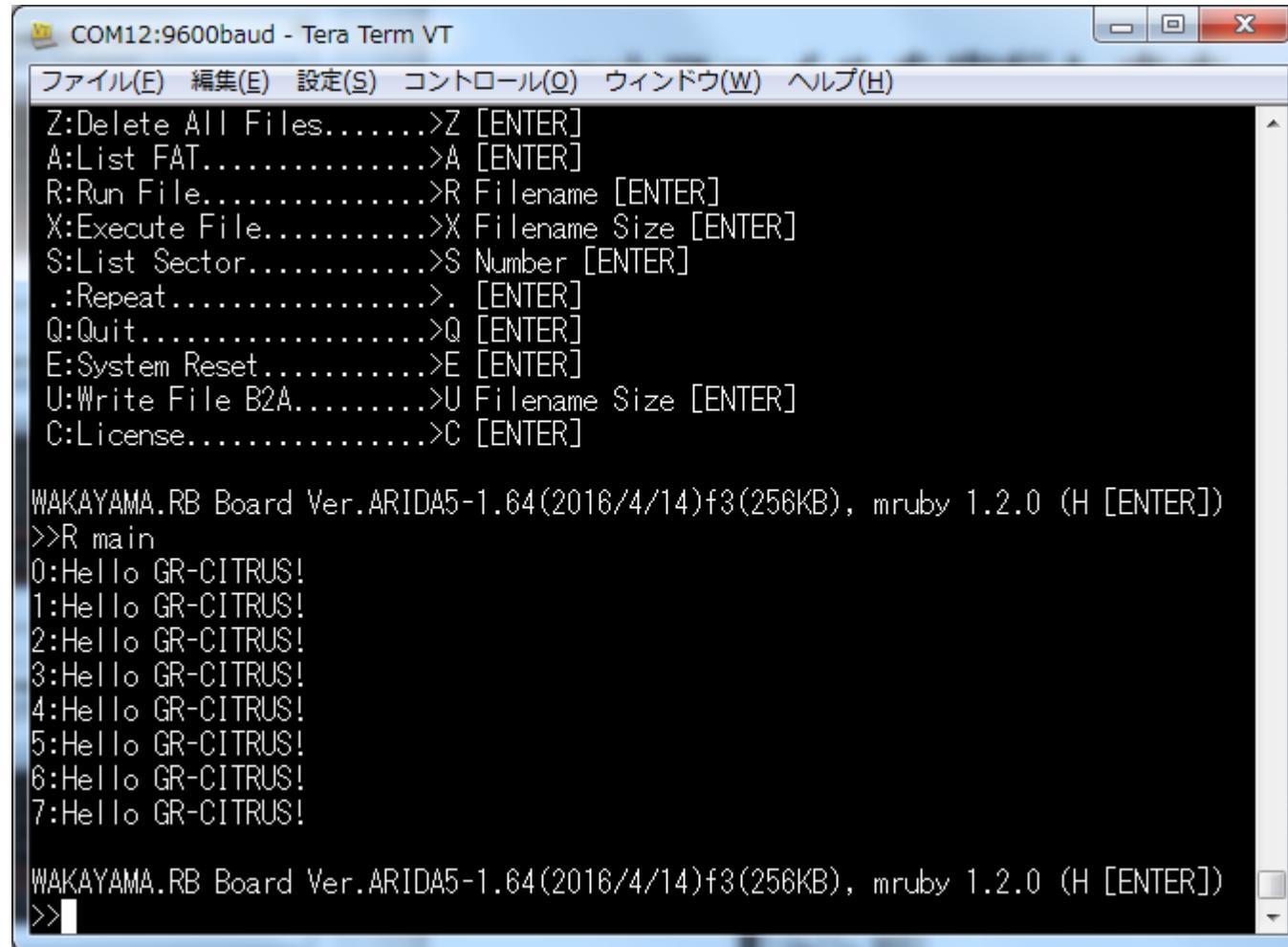
The main window displays the output of a command:

```
MAKAYAMA.RD Board Ver.ARIDA5-1.64(2016/4/14)f3(256KB), mruby 1.2.0 (H [ENTER])
>>R main
0>Hello GR-CITRUS!
1>Hello GR-CITRUS!
2>Hello GR-CITRUS!
3>Hello GR-CITRUS!
4>Hello GR-CITRUS!
5>Hello GR-CITRUS!
```

The first line "0>Hello GR-CITRUS!" is highlighted with a red rectangle.

# mrubyファイルを実行します。(R コマンド)

実行が終了するとコマンドモードに戻ります。



COM12:9600baud - Tera Term VT

ファイル(E) 編集(E) 設定(S) コントロール(Q) ウィンドウ(W) ヘルプ(H)

```
Z:Delete All Files.....>Z [ENTER]
A>List FAT.....>A [ENTER]
R:Run File.....>R Filename [ENTER]
X:Execute File.....>X Filename Size [ENTER]
S>List Sector.....>S Number [ENTER]
.:Repeat.....>. [ENTER]
Q:Quit.....>Q [ENTER]
E:System Reset...>E [ENTER]
U:Write File B2A.....>U Filename Size [ENTER]
C:License.....>C [ENTER]

WAKAYAMA.RB Board Ver.ARIDA5-1.64(2016/4/14)f3(256KB), mruby 1.2.0 (H [ENTER])
>>R main
0>Hello GR-CITRUS!
1>Hello GR-CITRUS!
2>Hello GR-CITRUS!
3>Hello GR-CITRUS!
4>Hello GR-CITRUS!
5>Hello GR-CITRUS!
6>Hello GR-CITRUS!
7>Hello GR-CITRUS!

WAKAYAMA.RB Board Ver.ARIDA5-1.64(2016/4/14)f3(256KB), mruby 1.2.0 (H [ENTER])
>>
```

# プログラムを書き込み実行します。(X コマンド)

Xコマンドを用いて、mrbファイルを書き込み後直ぐ実行します。

Xの後にスペースで区切って、ファイル名とファイルサイズを書き、ENTERキーを押します。

.mrbは省略可能です。

>X ファイル名 ファイルサイズ

あとは、Wコマンドと同様です。プログラムの書き込みが終了後、直ぐに実行されます。

The screenshot shows a terminal window titled "COM12:9600baud - Tera Term VT". The menu bar includes "ファイル(F)", "編集(E)", "設定(S)", "コントロール(Q)", "ウィンドウ(W)", and "ヘルプ(H)". The main window displays a list of commands:

- Z:Delete All Files.....>Z [ENTER]
- A>List FAT.....>A [ENTER]
- R:Run File.....>R Filename [ENTER]
- X:Execute File.....>X Filename Size [ENTER]
- S>List Sector.....>S Number [ENTER]
- .:Repeat.....>.[ENTER]
- Q:Quit.....>Q [ENTER]
- E:System Reset.....>E [ENTER]
- U:Write File B2A.....>U Filename Size [ENTER]
- C:License.....>C [ENTER]

Below the command list, the terminal displays the output of a program:

```
WAKAYAMA.RB Board Ver.ARIDA5-1.64(2016/4/14)f3(256KB), mruby 1.2.0 (H [ENTER])
>>R main
0>Hello GR-CITRUS!
1>Hello GR-CITRUS!
2>Hello GR-CITRUS!
3>Hello GR-CITRUS!
4>Hello GR-CITRUS!
5>Hello GR-CITRUS!
6>Hello GR-CITRUS!
7>Hello GR-CITRUS!
```

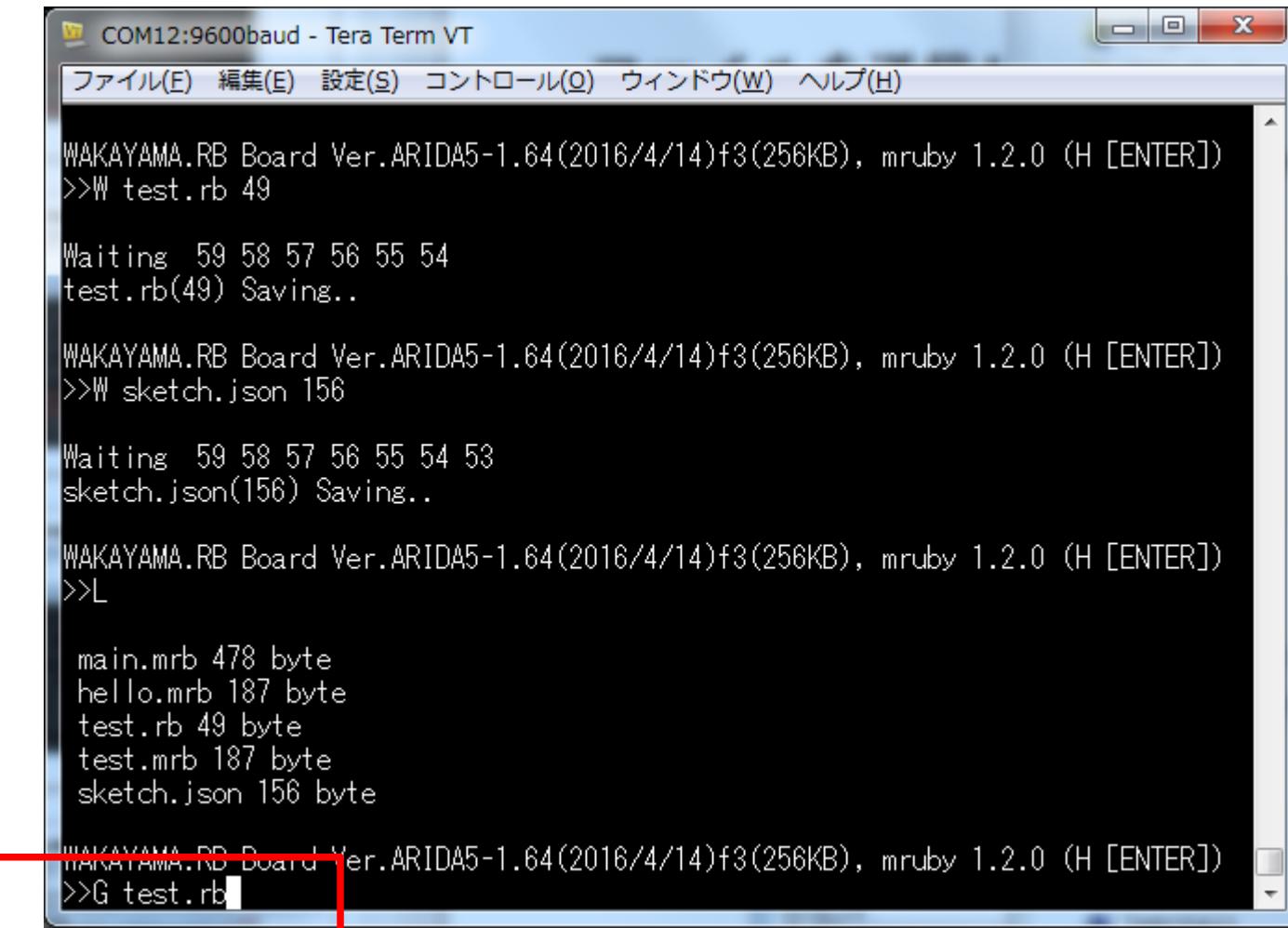
In the bottom right corner of the terminal window, there is a small red rectangular box highlighting the status bar which contains the text "WAKAYAMA.RB Board Ver.ARIDA5-1.64(2016/4/14)f3(256KB), mruby 1.2.0 (H [ENTER])".

# ファイルを送信します。(G コマンド)

Gコマンドを用いるとGR-CITRUSに保存されているファイルをPCに読み出すことができます。

Gの後にスペースで区切って、ファイル名を書き、ENTERキーを押します。

>G ファイル名



The screenshot shows a terminal window titled "COM12:9600baud - Tera Term VT". The window contains the following text output:

```
WAKAYAMA.RB Board Ver.ARIDA5-1.64(2016/4/14)f3(256KB), mruby 1.2.0 (H [ENTER])
>>W test.rb 49

Waiting 59 58 57 56 55 54
test.rb(49) Saving..

WAKAYAMA.RB Board Ver.ARIDA5-1.64(2016/4/14)f3(256KB), mruby 1.2.0 (H [ENTER])
>>W sketch.json 156

Waiting 59 58 57 56 55 54 53
sketch.json(156) Saving..

WAKAYAMA.RB Board Ver.ARIDA5-1.64(2016/4/14)f3(256KB), mruby 1.2.0 (H [ENTER])
>>L

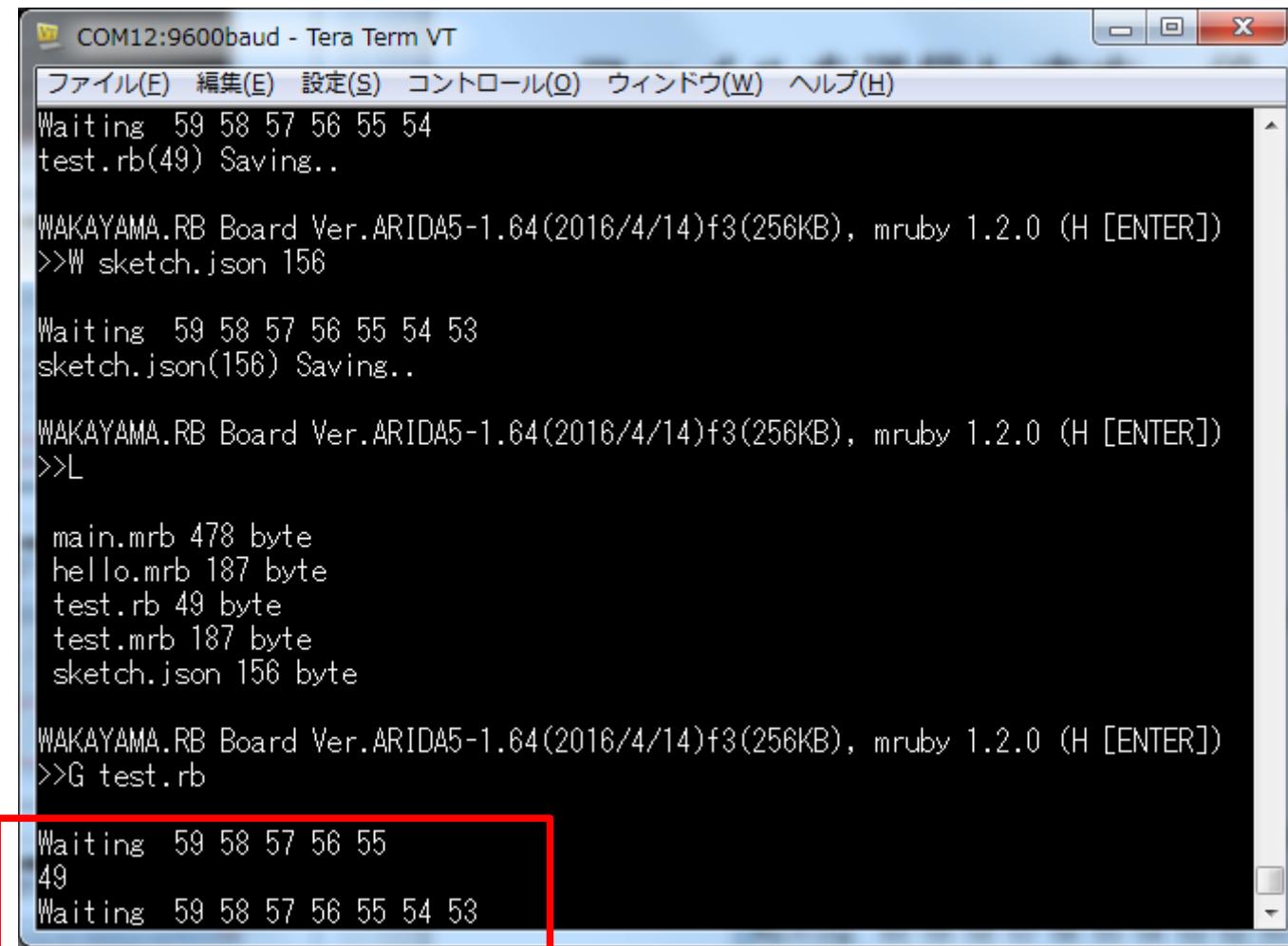
main.mrb 478 byte
hello.mrb 187 byte
test.rb 49 byte
test.mrb 187 byte
sketch.json 156 byte

WAKAYAMA.RB Board Ver.ARIDA5-1.64(2016/4/14)f3(256KB), mruby 1.2.0 (H [ENTER])
>>G test.rb
```

The input command ">>G test.rb" is highlighted with a red rectangle at the bottom left of the terminal window.

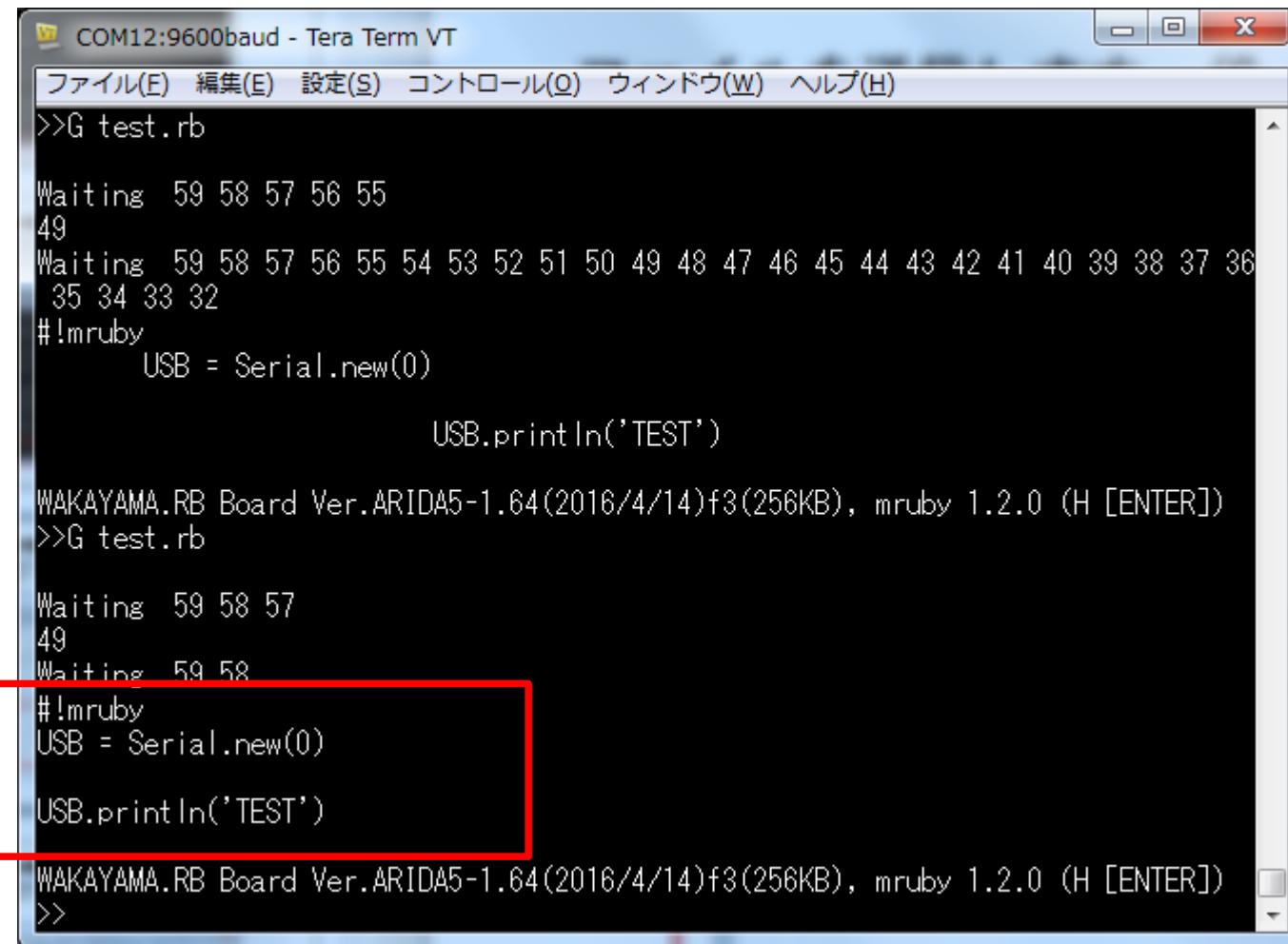
## ファイルを送信します。(G コマンド)

ENTERキーを押すと、カウントダウンが始まります。60sec以内にPCから1バイト送信すると、送信するファイルのファイルサイズが送信されます。  
そして、再びカウントダウンが始まります。



## ファイルを送信します。(G コマンド)

2回目のカウントダウンの60sec以内にPCから1バイト送信すると、指定したファイルが送信されます。  
バイナリファイルの場合もバイナリのまま送信されます。



```
COM12:9600baud - Tera Term VT
ファイル(E) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
>>G test.rb

Waiting 59 58 57 56 55
49
Waiting 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36
35 34 33 32
#!mruby
    USB = Serial.new(0)

        USB.println('TEST')

WAKAYAMA.RB Board Ver.ARIDA5-1.64(2016/4/14)f3(256KB), mruby 1.2.0 (H [ENTER])
>>G test.rb

Waiting 59 58 57
49
Waiting 59 58
#!mruby
USB = Serial.new(0)

USB.println('TEST')

WAKAYAMA.RB Board Ver.ARIDA5-1.64(2016/4/14)f3(256KB), mruby 1.2.0 (H [ENTER])
>>
```

## ファイルを削除する。(D コマンド)

DコマンドはWRBボード保存しているファイルを削除します。

Dの後にスペースで区切って、ファイル名を書き、ENTERキーを押します。

>D ファイル名

The screenshot shows a terminal window titled "COM12:9600baud - Tera Term VT". The window has a menu bar with Japanese labels: ファイル(E), 編集(E), 設定(S), コントロール(O), ウィンドウ(W), ヘルプ(H). The main text area displays the following session:

```
C:License.....>C [ENTER]
WAKAYAMA.RB Board Ver.ARIDA5-1.64(2016/4/14)f3(256KB), mruby 1.2.0 (H [ENTER])
>>L

main.mrb 478 byte
hello.mrb 187 byte
test.rb 49 byte
test.mrb 187 byte
sketch.json 156 byte

WAKAYAMA.RB Board Ver.ARIDA5-1.64(2016/4/14)f3(256KB), mruby 1.2.0 (H [ENTER])
>>D test.mrb
[The line >>D test.mrb is highlighted with a red rectangle]

WAKAYAMA.RB Board Ver.ARIDA5-1.64(2016/4/14)f3(256KB), mruby 1.2.0 (H [ENTER])
>>L

main.mrb 478 byte
hello.mrb 187 byte
test.rb 49 byte
sketch.json 156 byte

WAKAYAMA.RB Board Ver.ARIDA5-1.64(2016/4/14)f3(256KB), mruby 1.2.0 (H [ENTER])
>>
```

# コマンド再実行する。( . コマンド)

. コマンドを入力すると、直前に実行したコマンドを再実行します。

The screenshot shows a terminal window titled "COM12:9600baud - Tera Term VT". The window contains the following text:

```
>>.
>>R main
0>Hello GR-CITRUS!
1>Hello GR-CITRUS!
2>Hello GR-CITRUS!
3>Hello GR-CITRUS!
4>Hello GR-CITRUS!
5>Hello GR-CITRUS!
6>Hello GR-CITRUS!
7>Hello GR-CITRUS!

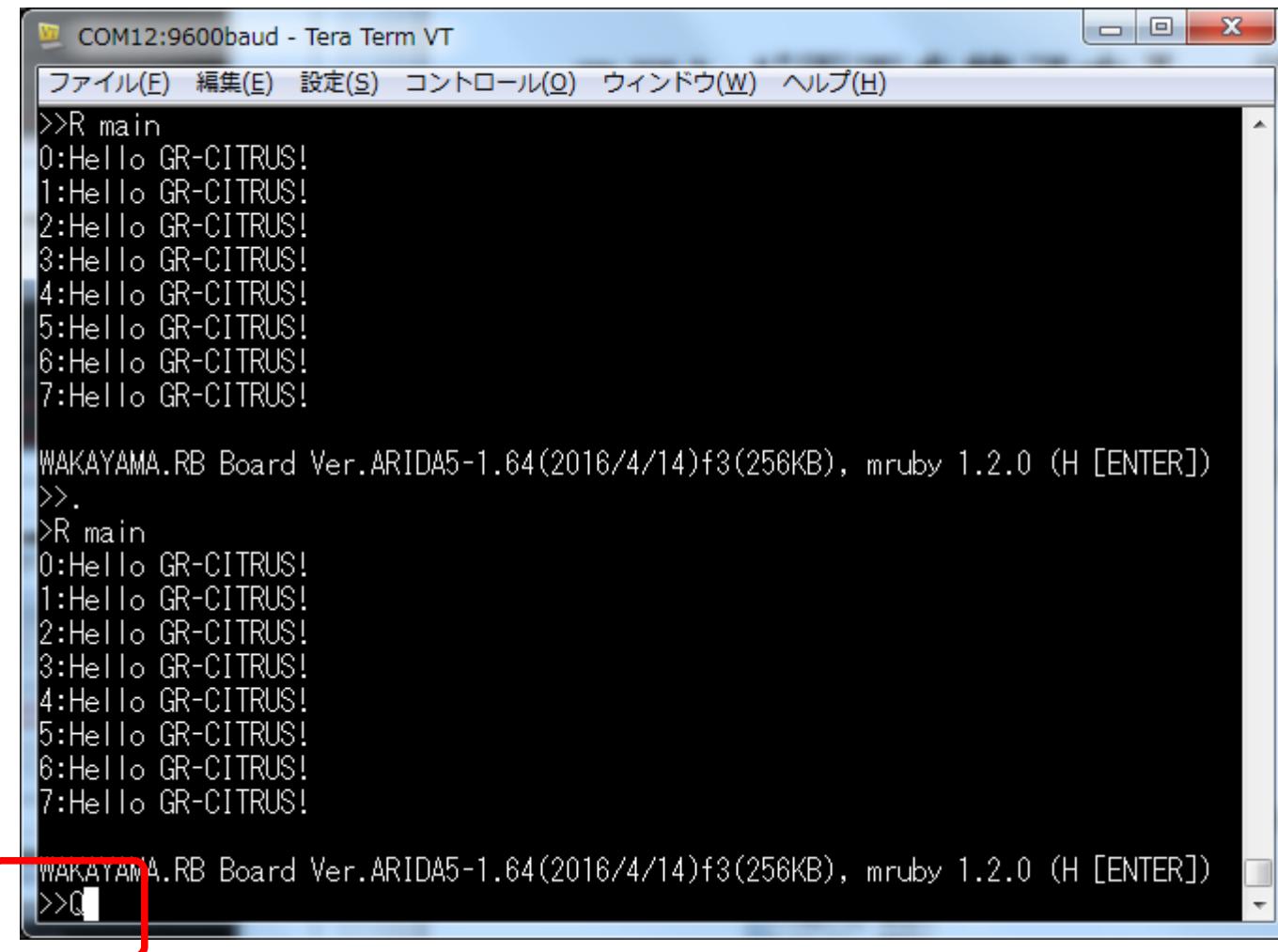
WAKAYAMA.RB Board Ver.ARIDA5-1.64(2016/4/14)f3(256KB), mruby 1.2.0 (H [ENTER])
>>R main
0>Hello GR-CITRUS!
1>Hello GR-CITRUS!
2>Hello GR-CITRUS!
3>Hello GR-CITRUS!
4>Hello GR-CITRUS!
5>Hello GR-CITRUS!
6>Hello GR-CITRUS!
7>Hello GR-CITRUS!

WAKAYAMA.RB Board Ver.ARIDA5-1.64(2016/4/14)f3(256KB), mruby 1.2.0 (H [ENTER])
>>.
```

Two lines of text are highlighted with red boxes: "WAKAYAMA.RB Board Ver.ARIDA5-1.64(2016/4/14)f3(256KB), mruby 1.2.0 (H [ENTER])" and ">>.". The first line is at the top of the second set of output, and the second line is at the bottom of the entire screenshot area.

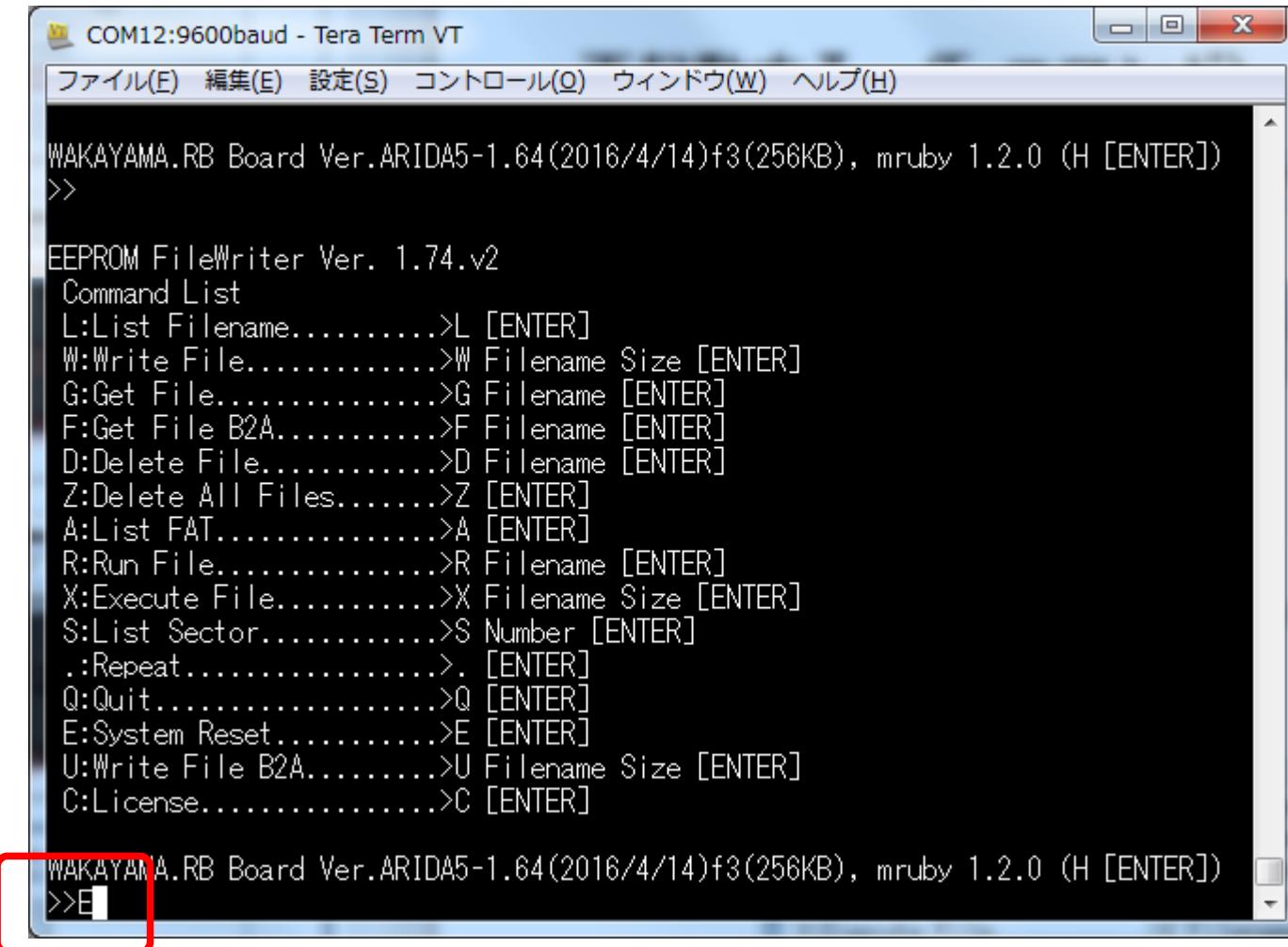
## コマンド画面を終了する。(Q コマンド)

Qコマンドを入力すると、コマンド画面が終了します。プログラムの途中で呼び出されている場合は、元のプログラムに戻ります。



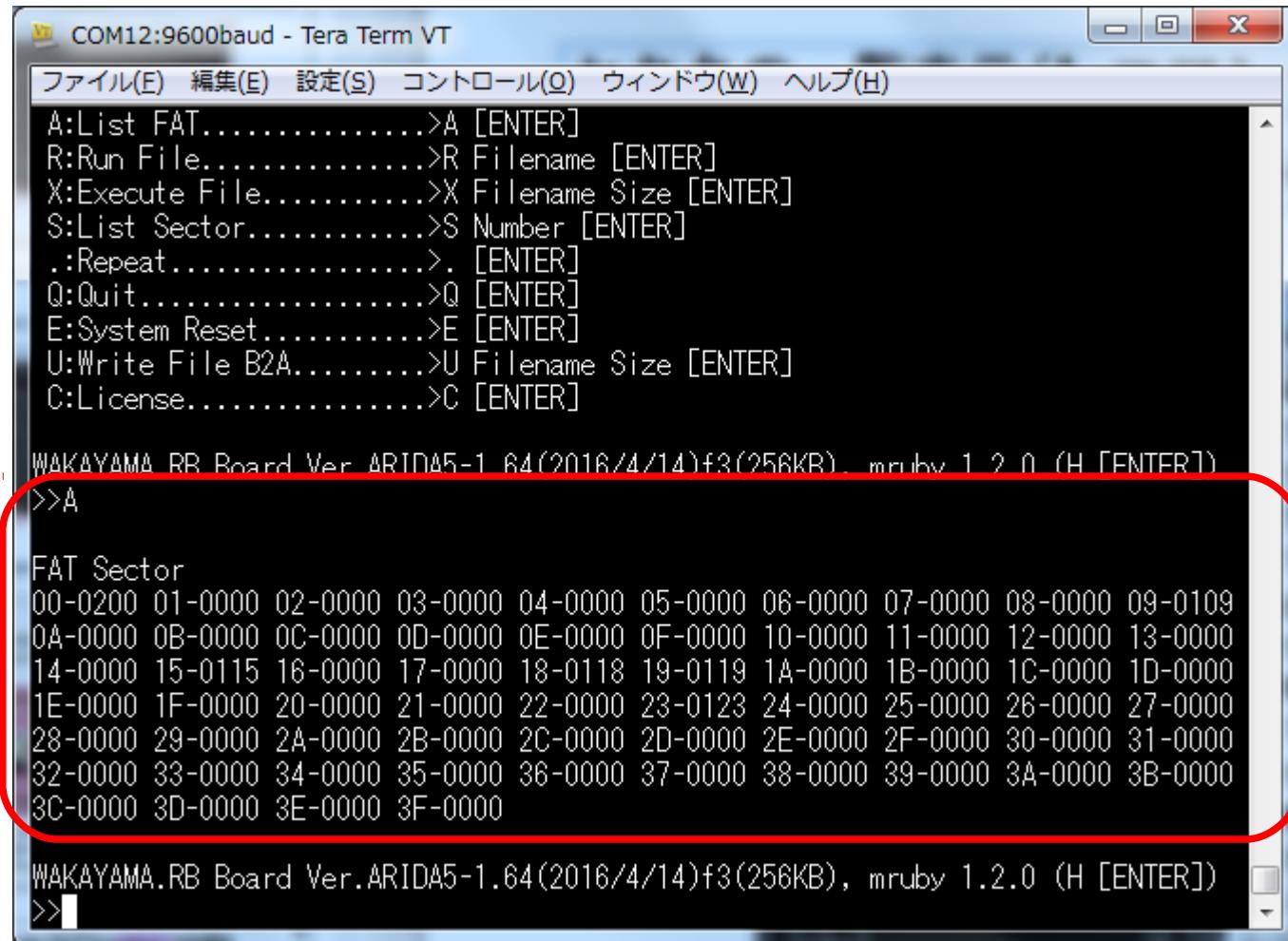
# 再起動する。(E コマンド)

Eコマンドを入力すると、マイコンを再起動します。



# セクタの一覧表示(A コマンド)

Aコマンドを入力すると、セクタを一覧表示します。



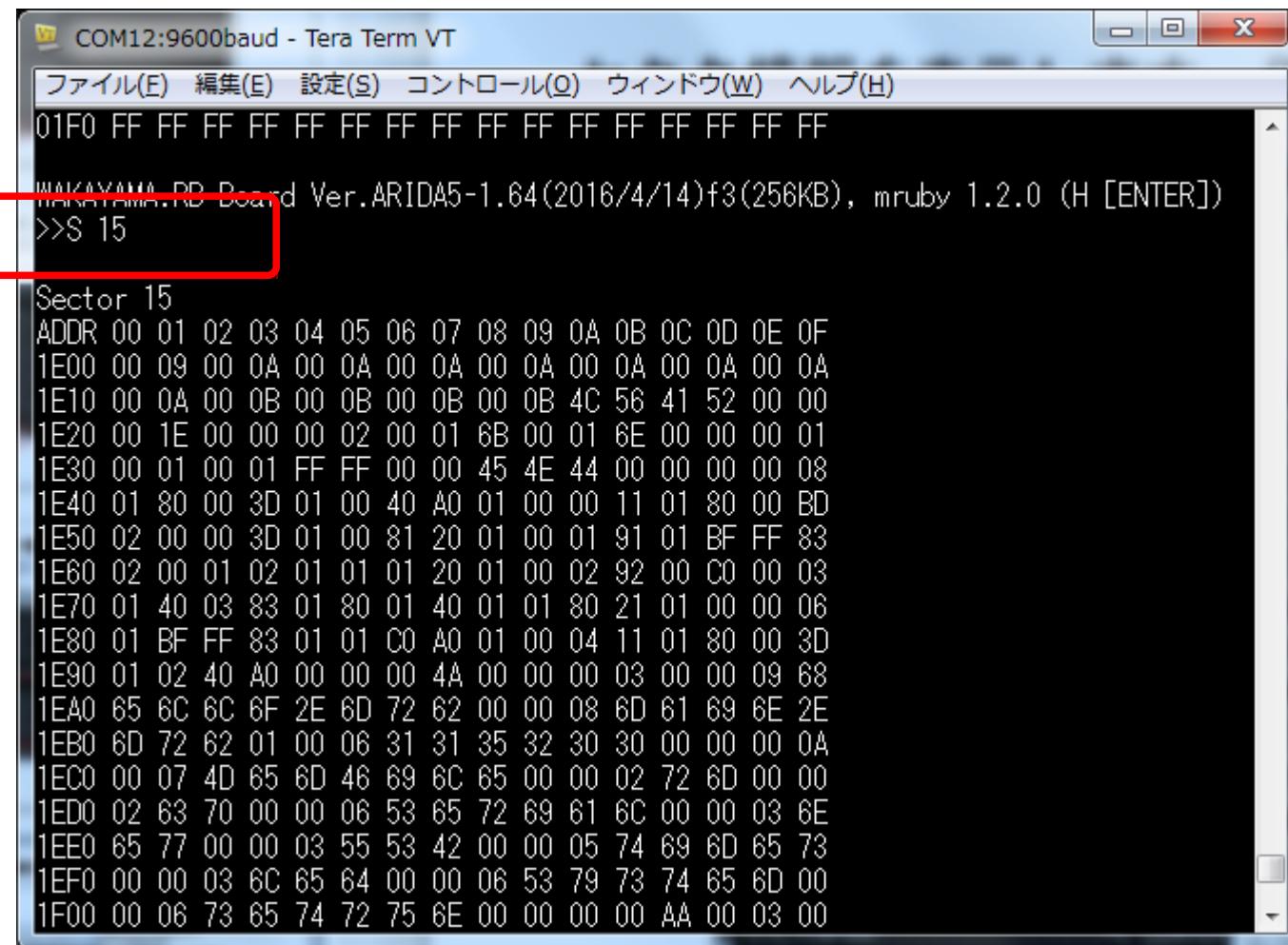
セクタ情報を表示します。(S コマンド)

Sコマンドは、セクタの情報を表示することができます。

Sの後にスペースで区切って、表示したいセクタ番号を書き、ENTERを押します。

番号は、Aコマンドで表示される番号です。

>S 番号



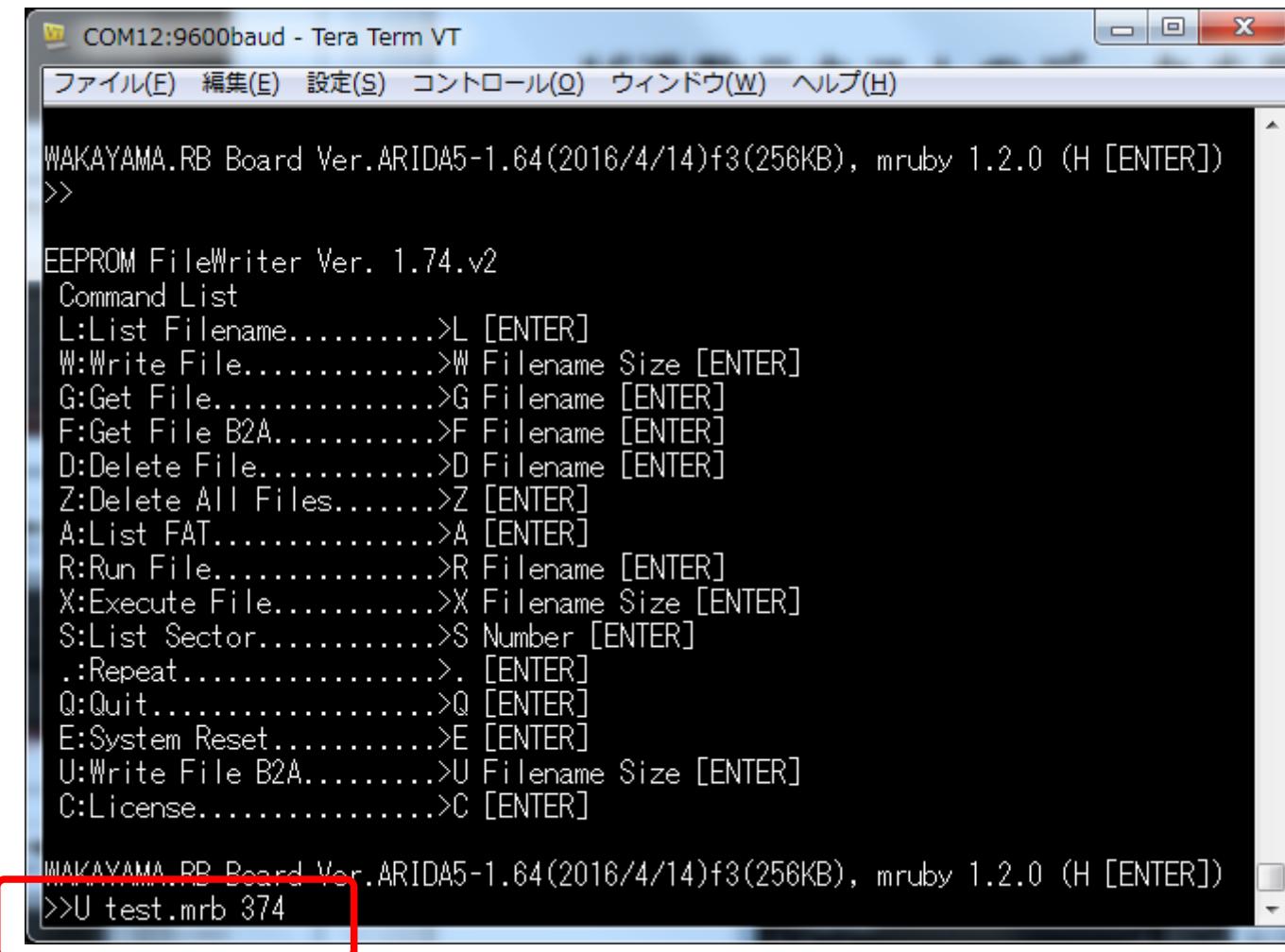
# 16進数テキストのデータを受信します。(U コマンド)

Uコマンドを用いて、16進数テキストデータを書き込みます。

Uの後にスペースで区切って、ファイル名とファイルサイズを書き、ENTERキーを押します。

ファイル受信方法はWコマンドと同じです。

>U ファイル名 ファイルサイズ



```
COM12:9600baud - Tera Term VT
[メニュー] ファイル(F) 編集(E) 設定(S) コントロール(Q) ウィンドウ(W) ヘルプ(H)
WAKAYAMA.RB Board Ver.ARIDA5-1.64(2016/4/14)f3(256KB), mruby 1.2.0 (H [ENTER])
>>

EEPROM FileWriter Ver. 1.74.v2
Command List
L:List Filename.....>L [ENTER]
W:Write File.....>W Filename Size [ENTER]
G:Get File.....>G Filename [ENTER]
F:Get File B2A.....>F Filename [ENTER]
D:Delete File.....>D Filename [ENTER]
Z:Delete All Files.....>Z [ENTER]
A:List FAT.....>A [ENTER]
R:Run File.....>R Filename [ENTER]
X:Execute File.....>X Filename Size [ENTER]
S>List Sector.....>S Number [ENTER]
.:Repeat.....>. [ENTER]
Q:Quit.....>Q [ENTER]
E:System Reset.....>E [ENTER]
U:Write File B2A.....>U Filename Size [ENTER]
C:License.....>C [ENTER]

WAKAYAMA.RB Board Ver.ARIDA5-1.64(2016/4/14)f3(256KB), mruby 1.2.0 (H [ENTER])
>>U test.mrb 374
```

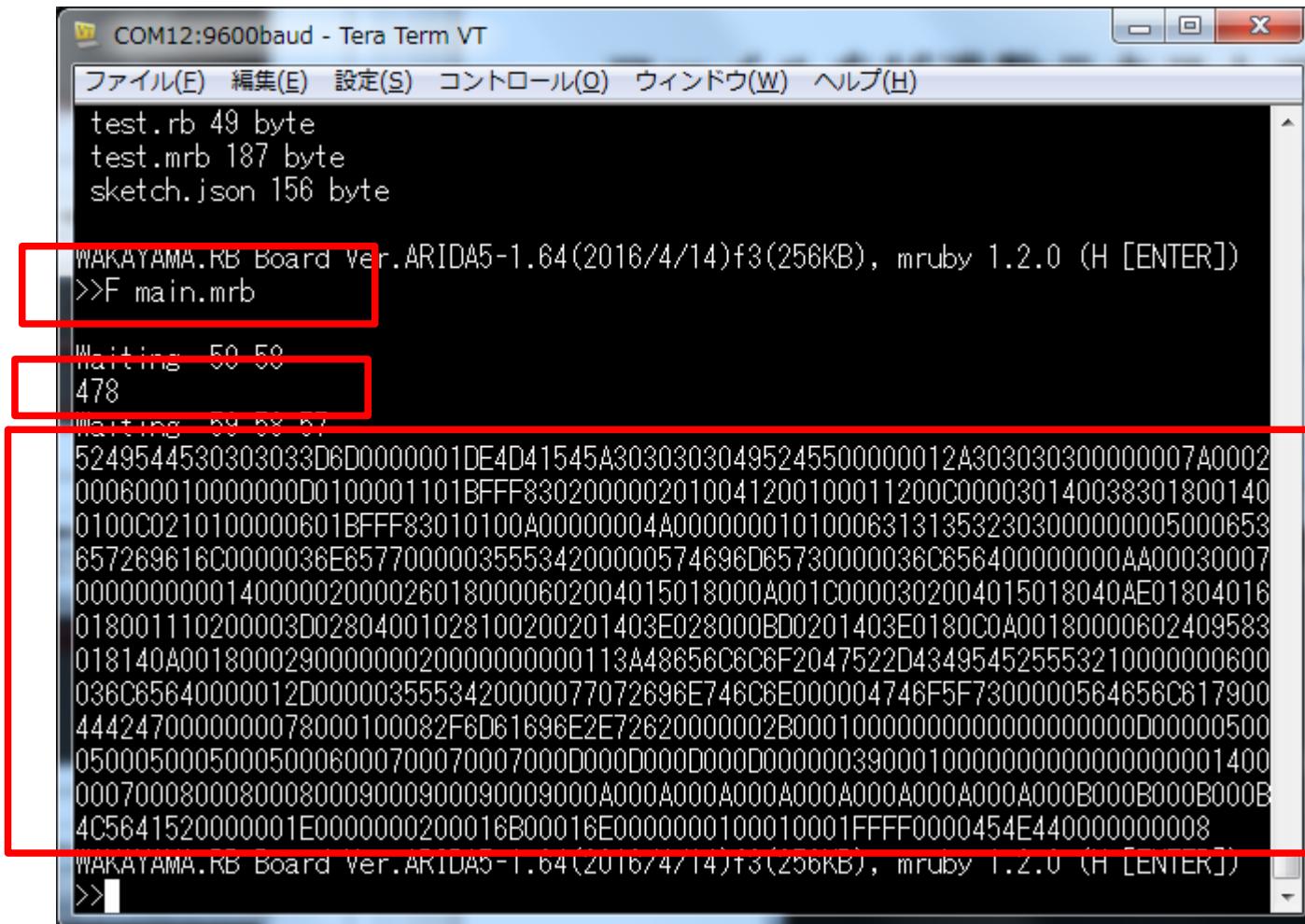
ファイルを16進数テキストで送信します。(F コマンド)

Fコマンドを用いるとGR-CITRUSに保存されているファイルをPCIに読み出すことができます。

Fの後にスペースで区切って、ファイル名を書き、ENTERキーを押します。

送信方法はGコマンドと同じですが、ファイル内容は16進数テキストで送信されます。

## >F ファイル名



# ファイルを全て削除する。(Z コマンド)

ZコマンドはGR-CITRUSに保存しているファイルを削除します。

実際にはファイルシステムを初期化しています。

>Z [ENTER]

The screenshot shows a terminal window titled "COM12:9600baud - Tera Term VT". The menu bar includes "ファイル(E)", "編集(E)", "設定(S)", "コントロール(O)", "ウィンドウ(W)", and "ヘルプ(H)". The main window displays a list of commands:

- S>List Sector.....>S Number [ENTER]
- .:Repeat.....>. [ENTER]
- Q:Quit.....>Q [ENTER]
- E:System Reset.....>E [ENTER]
- U:Write File B2A.....>U Filename Size [ENTER]
- C:License.....>C [ENTER]

Below the commands, the text "WAKAYAMA.RB Board Ver.ARIDA5-1.64(2016/4/14)f3(256KB), mruby 1.2.0 (H [ENTER])" is displayed. The command ">>L" is entered, followed by a list of files and their sizes:  
main.mrb 478 byte  
hello.mrb 187 byte  
test.rb 49 byte  
sketch.json 156 byte

The command ">>Z" is highlighted with a red rectangle. After executing, the screen shows the board information again: "WAKAYAMA.RB Board Ver.ARIDA5-1.64(2016/4/14)f3(256KB), mruby 1.2.0 (H [ENTER])" and ">>L".

# ライセンス情報の表示(T コマンド)

Tコマンドを入力すると、'>' の自動送信を停止/再開できます。

The screenshot shows a terminal window titled "COM17:9600baud - Tera Term VT". The menu bar includes "ファイル(E)", "編集(E)", "設定(S)", "コントロール(O)", "ウィンドウ(W)", and "ヘルプ(H)". The main window displays the "EEPROM FileWriter Ver. 1.75.v2" command list. The "T:">'Auto Print Switch...>T [ENTER]" command is highlighted with a red rectangle. Below the command list, two lines of text are displayed: "WAKAYAMA.RB Board Ver.CITRUS-2.15(2016/10/16)f3(256KB), mruby 1.2.0 (H [ENTER])" and ">>>>>>>T".

```
EEPROM FileWriter Ver. 1.75.v2
Command List
L>List Filename.....>L [ENTER]
W>Write File.....>W Filename Size [ENTER]
G>Get File.....>G Filename [ENTER]
F>Get File B2A.....>F Filename [ENTER]
D>Delete File.....>D Filename [ENTER]
Z>Delete All Files.....>Z [ENTER]
A>List FAT.....>A [ENTER]
R>Run File.....>R Filename [ENTER]
X>Execute File.....>X Filename Size [ENTER]
S>List Sector.....>S Number [ENTER]
.:Repeat.....>[ENTER]
Q>Quit.....>Q [ENTER]
E>System Reset.....>E [ENTER]
U>Write File B2A.....>U Filename Size [ENTER]
T:>'Auto Print Switch...>T [ENTER]
C>License.....>C [ENTER]

WAKAYAMA.RB Board Ver.CITRUS-2.15(2016/10/16)f3(256KB), mruby 1.2.0 (H [ENTER])
>>>>>>>T

WAKAYAMA.RB Board Ver.CITRUS-2.15(2016/10/16)f3(256KB), mruby 1.2.0 (H [ENTER])
>
```

# ライセンス情報の表示(C コマンド)

Cコマンドを入力すると、ライセンス情報を示します。

The screenshot shows a Tera Term VT window titled "COM12:9600baud - Tera Term VT". The menu bar includes ファイル(E), 編集(E), 設定(S), コントロール(Q), ウィンドウ(W), and ヘルプ(H). The main window displays a command menu:

```
G:Get File.....>G Filename [ENTER]
F:Get File B2A.....>F Filename [ENTER]
D:Delete File.....>D Filename [ENTER]
Z:Delete All Files.....>Z [ENTER]
A>List FAT.....>A [ENTER]
R:Run File.....>R Filename [ENTER]
X:Execute File.....>X Filename Size [ENTER]
S>List Sector.....>S Number [ENTER]
.:Repeat.....>.. [ENTER]
Q:Quit.....>Q [ENTER]
E:System Reset.....>E [ENTER]
U:Write File B2A.....>U Filename Size [ENTER]
C:License.....>C [ENTER]
```

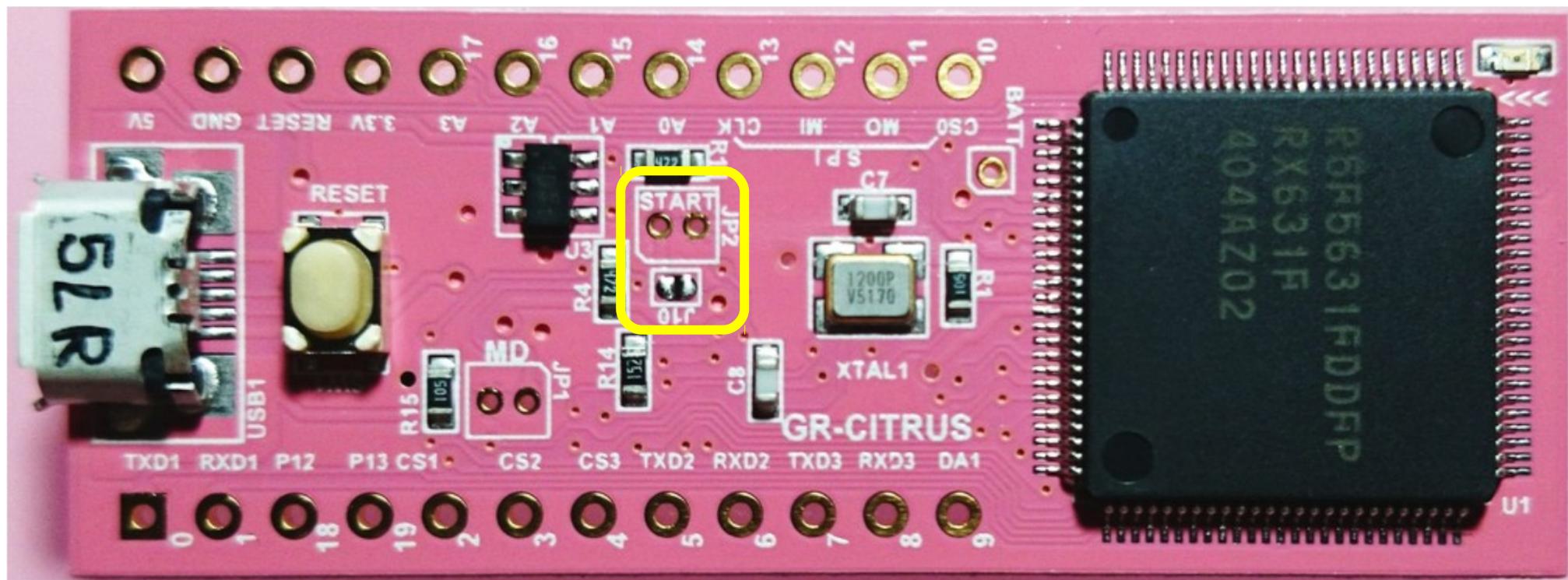
Below the menu, the text "WAKAYAMA RB Board Ver.ARIDA5-1.64(2016/4/14)f3(256KB), mruby 1.2.0 (H [ENTER])" is displayed. A red box highlights the command "C:License.....>C [ENTER]" and the resulting output:

```
>>C
mruby is released under the MIT License.
https://github.com/mruby/mruby/blob/master/MITL
Wakayama-mruby-board is released under the MIT License.
https://github.com/wakayamarb/wrbb-v2lib-firm/blob/master/MITL
```

At the bottom of the window, the text "WAKAYAMA.RB Board Ver.ARIDA5-1.64(2016/4/14)f3(256KB), mruby 1.2.0 (H [ENTER])" is repeated.

# 電源ONで即実行する方法

電源をONしてプログラムを即実行したい場合は、J10をショートさせるか、JP2にハーフピッチジャンパを取り付けて、ジャンパをショートさせてください。



Rubyファームにwrbb.xmlファイルがあり、Startタグで開始プログラム名が書かれているときには、該当プログラムを実行します。無い場合はmain.mrbが実行されます。main.mrbが無い場合は、コマンドモードになります。

# rubyプログラム自動実行の仕組み

自動実行する場合のrubyプログラム実行条件

## 条件(1)

Rubyファームは、先ずwrbb.xml ファイルを検索します。wrbb.xmlとはXML形式で書かれたファイルです。

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<Config>
  <Start file="wrbb.mrb" />
</Config>
```

Startタグのfile要素に実行するmrbファイル名を書いておくと、そのプログラムを実行します。

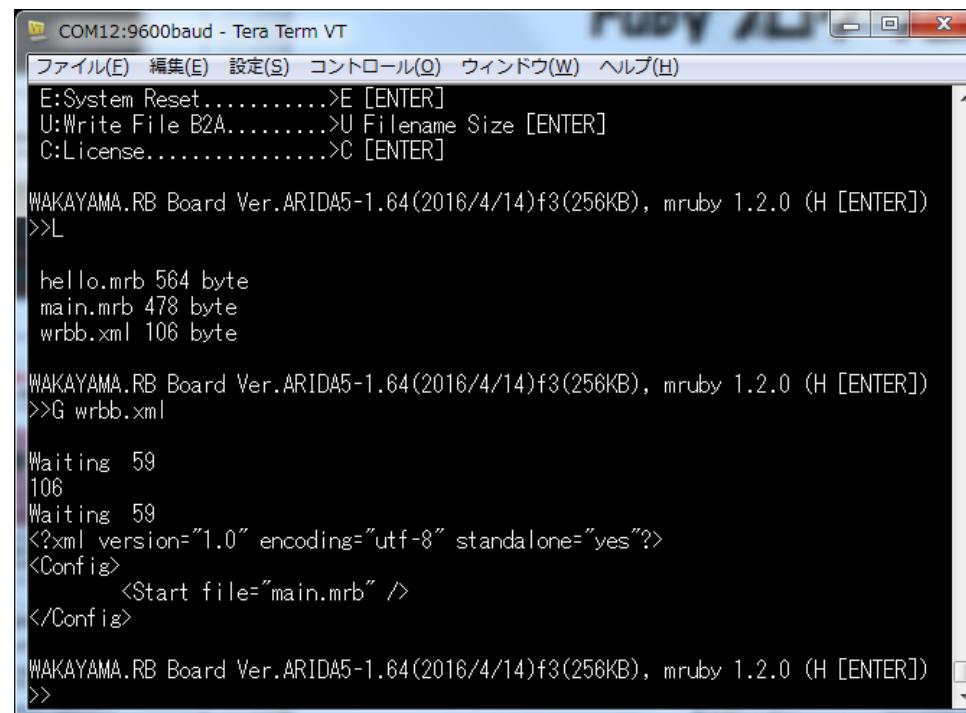
# rubyプログラム自動実行の仕組み

## 条件(2)

wrbb.xml ファイルが見つからない場合は、main.mrbファイルを検索します。  
main.mrb ファイルが見つかれば、main.mrbファイルを実行します。

## 条件(3)

wrbb.xml、main.mrb 両方のファイルが見つからない場合は、USB接続先にコマンド画面を表示します。



The screenshot shows a terminal window titled "COM12:9600baud - Tera Term VT". The window displays the following command-line session:

```
COM12:9600baud - Tera Term VT
E:System Reset.....>E [ENTER]
U:Write File B2A.....>U Filename Size [ENTER]
C:License.....>C [ENTER]

WAKAYAMA.RB Board Ver.ARIDA5-1.64(2016/4/14)f3(256KB), mruby 1.2.0 (H [ENTER])
>>L

hello.mrb 564 byte
main.mrb 478 byte
wrbb.xml 106 byte

WAKAYAMA.RB Board Ver.ARIDA5-1.64(2016/4/14)f3(256KB), mruby 1.2.0 (H [ENTER])
>>G wrbb.xml

Waiting 59
106
Waiting 59
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<Config>
    <Start file="main.mrb" />
</Config>

WAKAYAMA.RB Board Ver.ARIDA5-1.64(2016/4/14)f3(256KB), mruby 1.2.0 (H [ENTER])
>>
```

# rubyプログラム実行の仕組み

rubyプログラム例

LEDを5回 ON/OFFさせます。

```
sw = 1
10.times do
  led(sw)
  sw = 1 - sw
  delay(500)
end
```

以下のように書いても同じです。

```
sw = 1
for i in 1..10 do
  led(sw)
  sw = 1 - sw
  delay(500)
end
```

Hello GR-CITRUS!と10回出力されます。

```
usbout = Serial.new(0)
10.times do
  usbout.println("Hello GR-CITRUS!")
  delay(500)
end
```

# rubyプログラム実行の仕組み

rubyプログラム中に System.setrun 命令を用いて、次に呼び出すrubyプログラムを指定しておくと、実行が終了後、指定されたrubyプログラムが呼び出されます。

main.mrb 実行

```
sw = 1
10.times do
  led(sw)
  sw = 1 - sw
  delay(500)
end
System.setrun("hello.mrb")
```

hello.mrb 実行

```
usbout = Serial.new(0)
usbout.println(0,"Hello GR-CITRUS!")
led(1)
```

hello.mrb 終了

# mrbcファイルの作成方法

コマンドラインからmrbcを実行してください。

```
$ ./mrbc main.rb  
$ ls -l main.mrb  
----rwx---+ 1 minao None 865 6月 26 23:29 main.mrb
```

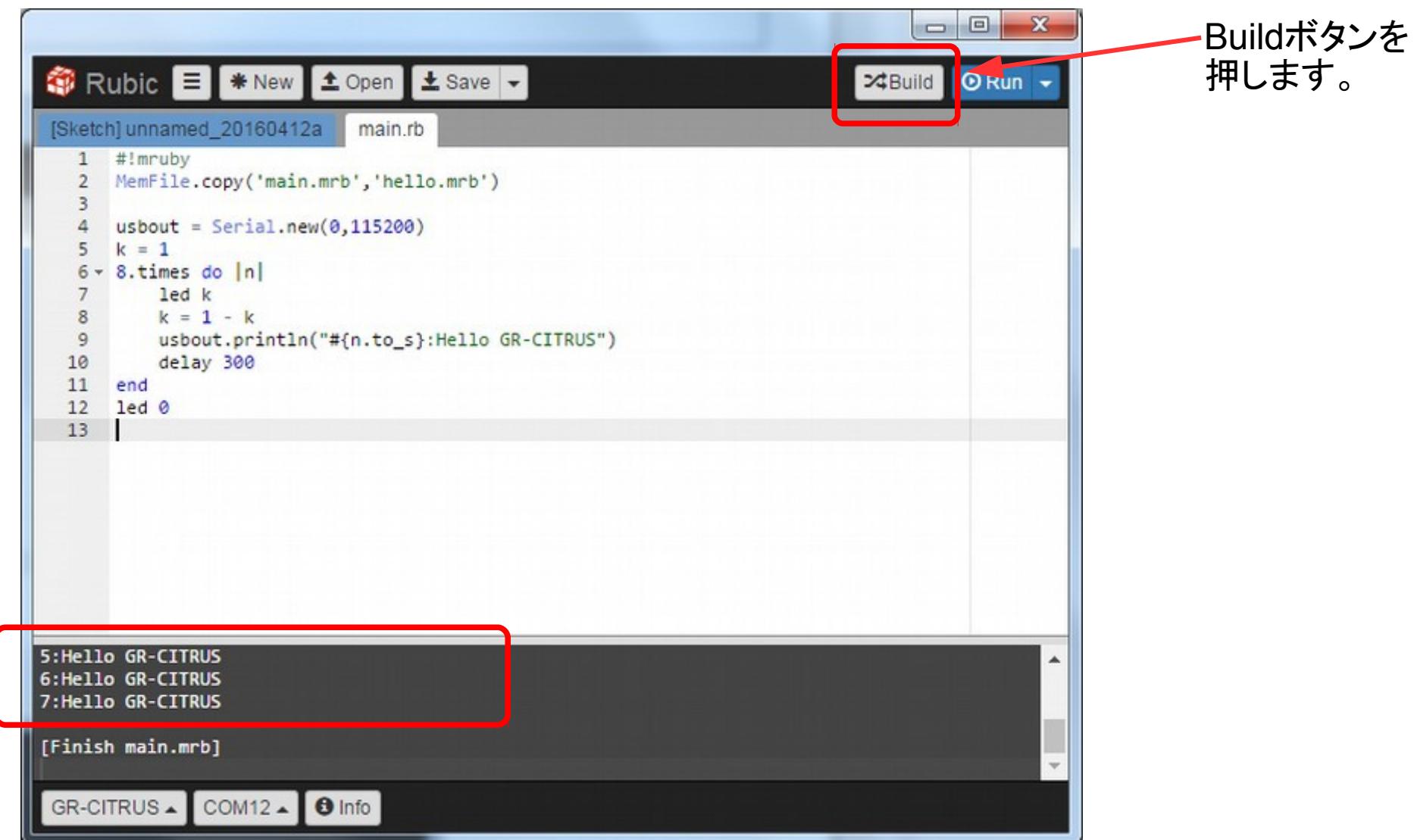
プログラムをデバッグしたい場合は、コンパイルオプションに `-g` を付けてコンパイルすることをお勧めします。エラーの行番号など詳しいエラーメッセージが出力されます。

```
$ ./mrbc -g main.rb
```

```
$ ./mrbc -h  
Usage: ./mrbc [switches] programfile  
switches:  
-c          check syntax only  
-o<outfile> place the output into <outfile>  
-v          print version number, then turn on verbose mode  
-g          produce debugging information  
-B<symbol> binary <symbol> output in C language format  
-e          generate little endian iseq data  
-E          generate big endian iseq data  
--verbose   run at verbose mode  
--version   print the version  
--copyright print the copyright
```

# mrbファイルの作成方法

きむしゅさんが開発している「Rubic」を使用すると、コマンドラインからmrbcを使うことなくmrbファイルを作成することができます。



# メソッドの説明(V2ライブラリ) カーネルクラス

## PINのモード設定 `pinMode(pin, mode)`

ピンのデジタル入力と出力を設定します。

`pin`: ピンの番号

`mode`:  
0: INPUTモード  
1: OUTPUTモード

デフォルトは入力(INPUT)モードです。

## デジタルライト `digitalWrite(pin, value)`

ピンのデジタル出力のHIGH/LOWを設定します。

`pin`: ピンの番号

`value`:  
0: LOW  
1: HIGH

## デジタルリード `digitalRead(pin)`

ピンのデジタル入力値を取得します。

`pin`: ピンの番号

戻り値  
0: LOW  
1: HIGH

# メソッドの説明(V2ライブラリ)

## グローバル定数

グローバル定数として以下の変数が使用可能です。

ON	1
OFF	0
HIGH	1
LOW	0
INPUT	0
OUTPUT	1

# メソッドの説明(V2ライブラリ) カーネルクラス

## アナログリード analogRead(pin)

ピンのアナログ入力値を取得します。

pin: アナログピンの番号(14, 15, 16, 17)

戻り値

10ビットの値(0~1023)

## アナログDACピン初期化 initDac()

アナログ出力ピンを初期化します。

初期化しないとアナログ出力しません。

## アナログDAC出力 analogDac(value)

ピンからアナログ電圧を出力します。

value: 10bit精度(0~4095)で0~3.3V

## LEDオンオフ led(sw)

基板のLEDを点灯します。

sw: 0:消灯

1:点灯

# メソッドの説明(V2ライブラリ) カーネルクラス

## PWM出力 pwm(pin, value)

ピンのPWM出力値をセットします。

pin: ピンの番号

value: 出力PWM比率(0~255)

PWM設定後に、他のピンのpinMode設定をしてください。一度PWMに設定したピンは、リセットするまで変更できません。ショートしているPIOはINPUTに設定しておいてください。

## アナログリファレンス analogReference(mode)

アナログ入力で使われる基準電圧を設定します。

mode: 0:DEFAULT : 5.0V Arduino互換, 1:INTERNAL : 1.1V 内蔵電圧, 2:EXTERNAL : AVREFピン供給電圧,  
3:RAW12BIT : 3.3V

## ディレイ delay(value)

指定の時間(ms)動作を止めます。

value: 時間(msec)

※delay中に強制的にGCを行っています。

## ミリ秒を取得します millis()

システムが稼動してから経過した時間を取得します。

戻り値

起動してからのミリ秒数

# メソッドの説明(V2ライブラリ) カーネルクラス

マイクロ秒を取得します `micros()`

システムが稼動してから経過した時間を取得します。

戻り値

起動してからのマイクロ秒数

トーンを出力 `tone(pin, frequency[, duration])`

トーンを出力します。

`pin`: ピン番号

`frequency`: 周波数 Hz

`duration`: 出力を維持する時間[ms]。省略時、0指定時は出力し続ける。

トーンを停止 `noTone(pin)`

トーンを出力を停止します。

`pin`: ピン番号

# メソッドの説明(V2ライブラリ) カーネルクラス

## 乱数の設定 randomSeed(value)

乱数を得るための種を設定します。

value: 種となる値

## 乱数の random([min, ] max)

乱数を取得します。

min: 亂数の取りうる最小値。省略可

max: 亂数の取りうる(最大値 + 1)

maxは乱数の取りうる最大値に+1したものです。

# メソッドの説明(V2ライブラリ) カーネルクラス

## 使用例

```
pinMode(4, INPUT)
pinMode(5, OUTPUT)

x = digitalRead(4)
digitalWrite(5, 0)

10.times do
  led(ON)
  delay(1000)
  led(OFF)
  delay(1000)
end
```

# メソッドの説明(V2ライブラリ) システムクラス

## システムのバージョン取得 System.version([R])

システムのバージョンを取得します。  
R: 引数があればmrubyのバーションを返します。

## プログラムの終了 System.exit()

プログラムを終了させます。  
System.setRunにより次に実行するプログラムがセットされていれば、そのプログラムが実行されます。

## 実行するプログラムの設定 System.setrun(filename)

次に実行するプログラムを設定します。  
filename: mrbファイル名

## コマンドモードの呼び出し System.onload()

コマンドモードを呼び出します。

# メソッドの説明(V2ライブラリ) システムクラス

## フラッシュメモリに書き込み System.push(address, buf, length)

フラッシュメモリに値を書き込みます。

address: 書き込み開始アドレス (0x0000~0x00ff)

buf: 書き込むデータ

length: 書き込むサイズ(MAX 32バイト)

戻り値

1:成功

0:失敗

※ここに書き込んだ値は、電源を切っても消えません。

## フラッシュメモリから読み出し System.pop(address, length)

フラッシュメモリから値を読み出します。

address: 読み込みアドレス (0x0000~0x00ff)

length: 読み込みサイズ(MAX 32バイト)

戻り値

読み込んだデータ分

## システムのリセット System.reset()

システムをリセットします。電源ONスタート状態となります。

# メソッドの説明(V2ライブラリ) システムクラス

## SDカードを使えるようにします System.useSD()

SDカードを使えるように設定します。

戻り値

0:使用不可, 1:使用可能

## WA-MIKANボード(WiFi)を使えるようにします System.useWiFi()

WA-MIKANボード(WiFi)を使えるように設定します。

戻り値

0:使用不可, 1:使用可能

## MP3再生を使えるようにします: System.useMP3(pausePin, stopPin)

MP3再生を行えるように設定します。

0番ピンとGNDの間にスピーカーを接続してください。

pausePin: 再生中の一時停止に使用するピン番号です。LOWになると一時停止/再開を繰り返します。

stopPin: 再生を止めるときに使用するピン番号です。LOWになると停止します。

戻り値

0:使用不可, 1:使用可能

設定できるピンは、1, 3, 4, 6, 9, 10, 14, 15, 16, 17, 18番ピンの11個です。

## メソッドの説明(V2ライブラリ) システムクラス

実行しているmrbファイルパスを取得します: `System.getMrbPath()`

実行しているmrbファイルパスを取得します。

戻り値

実行しているmrbファイルパス(ファイル名です)。

追加クラスを使用できるようにします: `System.use(ClassName[, Options])`

追加クラスを使用できるようにする。

`ClassName`: クラス名です。'SD'、'WiFi'、'MP3' のいずれかです。

`Options`: オプションの配列です。

SDはオプション無し。

WiFiはオプション無し。

MP3は再生中の一時停止に使用するピン番号と、再生を止めるときに使用するピン番号の配列を指定します。例) [3, 4]

戻り値

0: 使用不可

1: 使用可能

# メソッドの説明(V2ライブラリ) システムクラス

追加クラスを使用できるようにします: `System.use?(ClassName[, Options])`

追加クラスを使用できるようにする。

`ClassName`: クラス名です。'SD'、'WiFi'、'MP3' のいずれかです。

`Options`: オプションの配列です。

SDはオプション無し。

WiFiはオプション無し。

MP3は再生中の一時停止に使用するピン番号と、再生を止めるときに使用するピン番号の配列を指定します。例) [3, 4]

戻り値

`true`: 使用可能

`false`: 使用不可

# メソッドの説明(V2ライブラリ)

## システムクラス

### 使用例

```
#アドレス0x0000から0x0005に[0x3a, 0x39, 0x38, 0x00, 0x36]の5バイトのデータを書き込みます
buf = 0x3a. chr+0x39. chr+0x38. chr+0x0. chr+0x36. chr

System.push( 0x0000, buf, 5 )

#アドレス0x0000から5バイトのデータを読み込みます
ans = System.pop(0x0000, 5)

System.setrun('sample.mrb') #次に実行するプログラム名をセットします

System.exit() #このプログラムを終了します。
```

```
Usb = Serial.new(0, 115200)

if(System.use?("WiFi") == false)then
  Usb.println "WiFi Card can't use."
  System.exit()
end
Usb.println "WiFi Ready"

if(System.use?("MP3", [3, 4]) == false)then
  Usb.println "MP3 can't use."
  System.exit()
end
Usb.println "MP3 Ready"
```

# メソッドの説明(V2ライブラリ)

## シリアルクラス

このクラスはポート毎にインスタンスを生成して使います。

### シリアル通信の初期化 `Serial.new(num, bps)`

シリアル通信を初期化します。シリアル通信を使用する場合は、初めに初期化を行ってください。

`num`: 初期化する通信番号  
 0:USB  
 1:0ピン送信/1ピン受信  
 2:5ピン送信/6ピン受信  
 3:7ピン送信/8ピン受信  
 4:12ピン送信/11ピン受信

`bps`: ボーレート (bps) 基本的に任意の値が設定できます。

戻り値  
シリアルのインスタンス

### ボーレートの設定 `bps(baudrate)`

シリアル通信のボーレートを設定します。

`baudrate`: ボーレート

### シリアルポートへの出力 `print([str])`

シリアルポートに出力します。

`str`: 文字列。省略時は何も出力しません設定できます。

# メソッドの説明(V2ライブラリ)

## シリアルクラス

このクラスはポート毎にインスタンスを生成して使います。

### シリアルポートへの出力(¥r¥n付き) `println([str])`

シリアルポートに¥r¥n付きで出力します。

str: 文字列。省略時は改行のみ

### シリアル受信チェック `available()`

シリアルポートに受信データがあるかどうか調べます。

戻り値

シリアルバッファにあるデータのバイト数。0の場合はデータなし。

### シリアルポートからデータ取得 `read()`

シリアルポートの受信データを取得します。

戻り値

読み込んだデータ配列

データは0x00～0xFFの値

## メソッドの説明(V2ライブラリ)

### シリアルクラス

このクラスはポート毎にインスタンスを生成して使います。

#### シリアルポートへデータ出力 write(buf, len)

シリアルポートにデータを出力します。

buf: 出力データ

len: 出力データサイズ

戻り値

出力したバイト数

#### シリアルデータをフラッシュします flush()

シリアルデータをフラッシュします。

# メソッドの説明(V2ライブラリ)

## シリアルクラス

このクラスはポート毎にインスタンスを生成して使います。

### 使用例

```
USB_Out = Serial.new(0, 115200)
sw = 0

while(USB_Out.available() > 0) do    #何か受信があった
  USB_Out.read()
end

50.times do
  while(USB_Out.available() > 0) do #何か受信があった
    c = USB_Out.read()           #文字取得
    USB_Out.print c             #読み込んだ文字をprintします
  end

  #LEDを点滅させます
  led sw
  sw = 1 - sw

  delay 500
end
```

```
USB_Out = Serial.new(0, 115200)
data = 0x30.chr + 0x31.chr + 0.chr + 0x32.chr + 0x33.chr + 0x0d.chr + 0x0a.chr
USB_Out.write(data, 7)

System.exit()
```

## メソッドの説明(V2ライブラリ)

### MemFileクラス(Flashメモリをメディアのように扱うクラス)

#### ファイルのオープン MemFile.open(number, filename[, mode])

ファイルをオープンします。

number: ファイル番号 0 または 1

filename: ファイル名(8.3形式)

mode: 0:Read, 1:Append, 2:New Create

戻り値

成功: 番号, 失敗: -1

※同時に開けるファイルは2つまでに限定しています。

#### ファイルのクローズ MemFile.close(number)

ファイルをクローズします。

number: クローズするファイル番号 0 または 1

#### ファイルの読み出し位置に移動 MemFile.seek(number, byte)

Openしたファイルの読み出し位置に移動します。

number: ファイル番号 0 または 1

byte: seekするバイト数(-1)でファイルの最後に移動する

戻り値

成功: 1, 失敗: 0

## メソッドの説明(V2ライブラリ)

### MemFileクラス(Flashメモリをメディアのように扱うクラス)

Openしたファイルからの読み込み `MemFile.read(number)`

Openしたファイルから1バイト読み込みます。

`number`: ファイル番号 0 または 1

戻り値

0x00~0xFFが返る。ファイルの最後だったら-1が返る。

Openしたファイルにバイナリデータを書き込む `MemFile.write(number, buf, len )`

Openしたファイルにバイナリデータを書き込みます。

`number`: ファイル番号 0 または 1

`buf`: 書き込むデータ

`len`: 書き込むデータサイズ

戻り値

実際に書いたバイト数

ファイルをコピーします `MemFile.cp(srcFilename, dstFilename[, mode] )`

ファイルをコピーします。

`srcFilename`: コピー元ファイル名

`dstFilename`: コピー先ファイル名

`mode`: 0:上書きしない, 1:上書きする 省略時は上書きしない。

戻り値

成功: 1, 失敗: 0

## メソッドの説明(V2ライブラリ)

### MemFileクラス(Flashメモリをメディアのように扱うクラス)

ファイルを削除します MemFile.rm( Filename )

ファイルを削除します。

Filename: 削除するファイル名

戻り値

成功: 1, 失敗: 000~0xFFが返る。ファイルの最後だったら-1が返る。

# メソッドの説明(V2ライブラリ)

## MemFileクラス

### 使用例

```
MemFile.open(0, 'sample.txt', 2)
MemFile.write(0, 'Hello mruby World', 17)
data = 0x30.chr + 0x31.chr + 0.chr + 0x32.chr + 0x33.chr
MemFile.write(0, data, 5 )
MemFile.close(0)

MemFile.cp('sample.txt', 'memfile.txt', 1)

USB = Serial.new(0, 115200)          #USBシリアル通信の初期化

MemFile.open(0, 'memfile.txt', 0)
while(true) do
  c = MemFile.read(0)
  if(c < 0) then
    break
  end
  USB.write(c.chr, 1)
end
MemFile.close(0)
System.exit()
```

# メソッドの説明(V2ライブラリ)

## I2cクラス

このクラスはポート毎にインスタンスを生成して使います。

### I2C通信を行うピンの初期化 I2c. new(number)

I2C通信を行うピンの初期化を行います。

num: 通信番号

- 1: SDA-0ピン, SCL-1ピン
- 2: SDA-5ピン, SCL-6ピン
- 3: SDA-7ピン, SCL-8ピン
- 4: SDA-12ピン, SCL-11ピン

戻り値

I2cのインスタンス

### アドレスからデータを読み込み: read(deviceID, addressL[, addressH])

アドレスからデータを読み込みます。

deviceID: デバイスID

addressL: 読み込み下位アドレス

addressH: 読み込み上位アドレス

戻り値

読み込んだ値

### アドレスにデータを書き込みます write(deviceID, address, data)

アドレスにデータを書き込みます。

deviceID: デバイスID

address: 書き込みアドレス

data: データ

戻り値

常に 0

# メソッドの説明(V2ライブラリ)

## I2cクラス

このクラスはポート毎にインスタンスを生成して使います。

### I2Cデバイスに対して送信を開始するための準備をする: begin(deviceID)

I2Cデバイスに対して送信を開始するための準備をします。この関数は送信バッファを初期化するだけで、実際の動作は行わない。繰り返し呼ぶと、送信バッファが先頭に戻る。

deviceID: デバイスID 0～0x7Fまでの純粋なアドレス

### デバイスに対してI2Cの送信シーケンスの発行 end()

デバイスに対してI2Cの送信シーケンスを発行します。I2Cの送信はこの関数を実行して初めて実際に行われる。

戻り値  
常に 0

### デバイスに受信シーケンスを発行しデータを読み出す request(address, count)

デバイスに対して受信シーケンスを発行しデータを読み出します。

address: 読み込み開始アドレス  
count: 読み出す数

戻り値  
実際に受信したバイト数

# メソッドの説明(V2ライブラリ)

## I2cクラス

このクラスはポート毎にインスタンスを生成して使います。

### 送信バッファの末尾に数値を追加する `lwrite(data)`

送信バッファの末尾に数値を追加します。

`data`: セットする値

戻り値

送信したバイト数(バッファに溜めたバイト数)を返す。

送信バッファ(260バイト)に空き容量が無ければ失敗して0を返す。

### デバイスに受信シーケンスを発行しデータを読み出す `lread()`

デバイスに対して受信シーケンスを発行しデータを読み出します。

戻り値

読み込んだ値

### 受信バッファ内にあるデータ数を調べる `available()`

デバイスに対して受信バッファ内にあるデータ数を調べます。

戻り値

データ数

# メソッドの説明(V2ライブラリ)

## I2cクラス

このクラスはポート毎にインスタンスを生成して使います。

### 使用例

```

@APTemp = 0x5D          # 0b01011101 圧力・温度センサのアドレス
USB = Serial.new(0, 115200)    #USBシリアル通信の初期化

#センサ接続ピンの初期化(12番SDA, 11番SCL)
sensor = I2c.new(3)
delay(300)

#気圧と温度センサの初期化
@APTemp = 0x5D          # 0b01011101
APTemp_CTRL_REG1 = 0x20  # Control register
APTemp_SAMPLING = 0xA0   # A0:7Hz, 90:1Hz
# 7Hz
sensor.write(@APTemp, APTemp_CTRL_REG1, APTemp_SAMPLING)
delay(100)

#気圧を取得します -----
#Address 0x28, 0x29, 0x2A, 0x2B, 0x2C
v0 = sensor.read( @APTemp, 0x28, 0x29)
v1 = sensor.read( @APTemp, 0x2A)
a = v0 + v1 * 65536
a = a / 4096.0      # hPa単位に直す

#温度を取得します -----
v2 = sensor.read( @APTemp, 0x2B, 0x2C)
if v2 > 32767
    v2 = v2 - 65536
end
t = v2 / 480.0 + 42.5
USB.println(a.to_s + "," + t.to_s)

```

# メソッドの説明(V2ライブラリ)

## I2cクラス

このクラスはポート毎にインスタンスを生成して使います。

### 使用例

```

USB = Serial.new(0, 115200)          #USBシリアル通信の初期化
#センサ接続ピンの初期化(12番SDA, 11番SCL)
sensor = I2c.new(3)
delay(300)
#気圧と温度センサの初期化
@APTemp = 0x5D                      # 0b01011101
APTemp_CTRL_REG1 = 0x20   # Control register
APTemp_SAMPLING = 0xA0    # A0:7Hz, 90:1Hz
sensor.write(@APTemp, APTemp_CTRL_REG1, APTemp_SAMPLING)      # 7Hz
delay(100)

#Address 0x2B, 0x2C
sensor.begin(@APTemp)
sensor.lwrite(0x2B)
sensor.end()
sensor.request(@APTemp, 1)
datL = sensor.lread()

sensor.begin(@APTemp)
sensor.lwrite(0x2C)
sensor.end()
sensor.request(@APTemp, 1)
datH = sensor.read()
v = datL + datH * 256
if v > 32767
    v = v - 65536
end
t = v / 480.0 + 42.5
USB.println(t.to_s)

```

# メソッドの説明(V2ライブラリ) サーボクラス

サーボ出力を任意のピンに割り当てます `Servo.attach(ch, pin[, min, max])`

`ch`: サーボのチャネル 0~9まで指定できます

`pin`: 割り当てるピン番号

`min`: サーボの角度が0度のときのパルス幅(マイクロ秒)。デフォルトは544

`max`: サーボの角度が180度のときのパルス幅(マイクロ秒)。デフォルトは2400

サーボの角度をセットします: `Servo.write(ch, angle)`

`ch`: サーボのチャネル 0~9まで指定できます

`angle`: 角度 0~180バイスに対して受信シーケンスを発行しデータを読み出します。

サーボモータにus単位で角度を指定します: `Servo.us(ch, us)`

`ch`: サーボのチャネル 0~9まで指定できます

`us`: 出力したいパルスの幅 1~19999, 0で出力 OFF

サーボモータに与えられるパルスは20ms周期で、1周期中のHighの時間を直接指定する。

実質的にPWM出力。連続回転タイプのサーボでは、回転のスピードが設定することができる。

最後に設定された角度を読み出します: `Servo.read(ch)`

`ch`: サーボのチャネル 0~9まで指定できます

戻り値

マイクロ秒単位。ただし `us(ch)` で与えた値は読みとれません。

# メソッドの説明(V2ライブラリ) サーボクラス

ピンにサーボが割り当てられているかを確認します: Servo.attached(ch)

ch: サーボのチャネル 0~9まで指定できます

戻り値

1: 割り当てられている  
0: 割り当てはない

ピンにサーボが割り当てられているかを確認します: Servo.attached?(ch)

ch: サーボのチャネル 0~9まで指定できます

戻り値

true: 割り当てられている  
false: 割り当てはない

サーボの動作を止め、割り込みを禁止します: Servo.detach(ch)

ch: サーボのチャネル 0~9まで指定できます

# メソッドの説明(V2ライブラリ) サーボクラス

## 使用例

```
g_pos = 0
g_inc = 10

USB = Serial.new(0, 115200)          #USBシリアル通信の初期化

#8番ピンをサーボ用ピンに割り当てる。
Servo.attach(0, 8)
Servo.write(0, g_pos) #サーボの角度設定

#サーボを10度ずつ50回動かす
50.times do
  delay(100)
  g_pos = g_pos + g_inc
  Servo.write(0, g_pos)
  if(g_pos >= 180 || g_pos <= 0) then
    g_inc = -g_inc
  end
end

Servo.detach(0)
```

# メソッドの説明(V2ライブラリ) リアルタイムクロッククラス

## RTCを起動します: Rtc. init()

戻り値

- 0: 起動失敗
- 1: 起動成功

init()を実行すると日時がリセットされます。

## RTCを停止します: Rtc. deinit()

RTCを停止します。

戻り値

- 0: 失敗
- 1: 成功

## RTCの日時をセットします: Rtc. setTime(array)

RTCの日時をセットします。

array: 年(0000-9999), 月(1-12), 日(1-31), 時(0-23), 分(0-59), 秒(0-59) の配列

戻り値

- 0: 失敗
- 1: 成功

# メソッドの説明(V2ライブラリ) リアルタイムクロッククラス

RTCの日時を取得します: Rtc. getTime()

RTCの日時を取得します。

戻り値は以下の値が配列で返ります

year: 年 (2000–2099)

month: 月 (1–12)

day: 日 (1–31)

hour: 時 (0–23)

minute: 分 (0–59)

second: 秒 (0–59)

weekday: 曜日 (0–6) 0:日, 1:月, 2:火, 3:水, 4:木, 5:金, 6:土

# メソッドの説明(V2ライブラリ) リアルタイムクロッククラス

## 使用例

```
USB = Serial.new(0, 115200)      #USBシリアル通信の初期化  
  
Rtc.init  
  
Rtc.setDateTime([2016, 4, 16, 17, 0, 0])  
  
15.times do|i|  
  led(i % 2)  
  year, mon, da, ho, min, sec = Rtc.getTime()  
  USB.println(year.to_s + "/" + mon.to_s + "/" + da.to_s + " " + ho.to_s + ":" + min.to_s + ":" +  
  sec.to_s)  
  delay(500)  
end
```

# メソッドの説明(V2ライブラリ)

## SDカードクラス

`System.useSD()` を呼んでおく必要があります。

### ファイルのオープン SD.open(number, filename[, mode])

ファイルをオープンします。

number: ファイル番号 0 または 1

filename: ファイル名(8.3形式)

mode: 0:Read, 1:Append, 2:New Create

戻り値

成功: 番号, 失敗: -1

※同時に開けるファイルは2つまでに限定しています。

### ファイルのクローズ SD.close(number)

ファイルをクローズします。

number: クローズするファイル番号 0 または 1

### ファイルの読み出し位置に移動 SD.seek(number, byte)

Openしたファイルの読み出し位置に移動します。

number: ファイル番号 0 または 1

byte: seekするバイト数(-1)でファイルの最後に移動する

戻り値

成功: 1, 失敗: 0

# メソッドの説明(V2ライブラリ)

**SDカードクラス** `System.useSD()` を呼んでおく、または`System.use('SD')` しておく必要があります。

## Openしたファイルからの読み込み `SD.read(number)`

Openしたファイルから1バイト読み込みます。

`number`: ファイル番号 0 または 1

戻り値

0x00~0xFFが返る。ファイルの最後だったら-1が返る。

## Openしたファイルにバイナリデータを書き込む `SD.write(number, buf, len )`

Openしたファイルにバイナリデータを書き込みます。

`number`: ファイル番号 0 または 1

`buf`: 書き込むデータ

`len`: 書き込むデータサイズ

戻り値

実際に書いたバイト数

## Openしたファイルの書き込みをフラッシュします: `SD.flush(number)`

Openしたファイルの書き込みをフラッシュします。

`number`: ファイル番号 0 または 1

## メソッドの説明(V2ライブラリ)

**SDカードクラス** `System.useSD()`を呼んでおく、または`System.use('SD')`しておく必要があります。

Openしたファイルのサイズを取得します: `SD.size(number)`

Openしたファイルのサイズを取得します。

number: ファイル番号 0 または 1

戻り値

ファイルサイズ

Openしたファイルのseek位置を取得します: `SD.position(number)`

Openしたファイルのseek位置を取得します。

number: ファイル番号 0 または 1

戻り値

シーク位置

## メソッドの説明(V2ライブラリ)

**SDカードクラス** `System.useSD()`を呼んでおく、または`System.use('SD')`しておく必要があります。

ディレクトリを作成する: `SD.mkdir(dirname)`

ディレクトリを作成する。

`dirname`: 作成するディレクトリ名

戻り値

成功: 1, 失敗: 0

ファイルを削除します: `SD.remove(filename)`

ファイルを削除します。

`filename`: 削除するファイル名

戻り値

成功: 1, 失敗: 0

ファイルをコピーする: `SD.copy(srcfilename, distfilename)`

ファイルをコピーする。

`srcfilename`: コピー元ファイル名

`distfilename`: コピー先ファイル名

戻り値

成功: 1, 失敗: 0

## メソッドの説明(V2ライブラリ)

**SDカードクラス** `System.useSD()`を呼んでおく、または`System.use('SD')`しておく必要があります。

ディレクトリを削除します: `SD.rmtree(dirname)`

ディレクトリを削除します。

`dirname`: 削除するディレクトリ名

戻り値

成功: 1, 失敗: 0

ファイルが存在するかどうか調べる: `SD.exists(filename)`

ファイルが存在するかどうか調べます。

`filename`: 調べるファイル名

戻り値

存在する: 1, 存在しない: 0

# メソッドの説明(V2ライブラリ)

## SDカードクラス

System.useSD()を呼んでおく、またはSystem.use('SD')しておく必要があります。

### 使用例

```
if(System.use?('SD') == false) then
    System.exit
end

SD.open(0, 'sample.txt', 2)
    SD.write(0, 'Hello mruby World', 17)
    data = 0x30.chr + 0x31.chr + 0.chr + 0x32.chr + 0x33.chr
    Serial.write(0, data, 5)
SD.close(0)

USB = Serial.new(0, 115200)          #USBシリアル通信の初期化

SD.open(0, 'sample.txt', 0)
while(true) do
    c = SD.read(0)
    if(c < 0) then
        break
    end
    USB.write(c.chr, 1)
end

SD.close(0)
System.exit()
```

# メソッドの説明(V2ライブラリ)

## WiFiクラス

`System.WiFi()`を呼んでおく、または`System.use('WiFi')`しておく必要があります。

### ステーションモードを設定する: `WiFi.setMode(mode)`

ステーションモードを設定します。

`mode`: 1:Station, 2:SoftAP, 3:Station + SoftAP1

戻り値

ESP8266の戻り値

### 応答のシリアル出力設定: `WiFi.serialOut(mode[, serialNumber])`

ESP8266に送信したコマンドの応答をシリアル出力するときに設定します。

`mode`: 0:出力しない, 1:出力する

`serialNumber`: 出力先のシリアル番号

戻り値

無し

### ATコマンドを送信する: `WiFi.at(command[, mode])`

ATコマンドを送信します。

`command`: ATコマンド文字列

`mode`: 0: 'AT+' を自動追加する、1: 'AT+' を自動追加しない

戻り値

ESP8266の戻り値

## メソッドの説明(V2ライブラリ)

### WiFiクラス

System.WiFi()を呼んでおく、またはSystem.use('WiFi')しておく必要があります。

#### WiFi接続する: WiFi.connect(SSID, Passwd)

WiFiアクセスポイントの接続します。

SSID: WiFiのSSID

Passwd: パスワード

戻り値

ESP8266の戻り値

#### IPアドレスとMACアドレスの表示: WiFi.ipconfig()

IPアドレスとMACアドレスを表示します

戻り値

ESP8266の戻り値

#### USBポートとESP8266をシリアルで直結します: WiFi.bypass()

USBポートとESP8266をシリアルで直結します。

リセットするまで、処理は戻りません。

# メソッドの説明(V2ライブラリ)

## WiFiクラス

System.WiFi()を呼んでおく、またはSystem.use('WiFi')しておく必要があります。

### ESP8266のソフトのバージョンを取得する: WiFi.version()

ESP8266のソフトのバージョンを取得します。

戻り値

ESP8266の戻り値

### WiFiを切断します: WiFi.disconnect()

WiFiを切断します。

戻り値

ESP8266の戻り値

### 複数接続可能モードの設定: WiFi.multiConnect(mode)

複数接続可能モードの設定をします。

mode: 0:1接続のみ, 1:4接続まで可能

戻り値

ESP8266の戻り値

# メソッドの説明(V2ライブラリ)

## WiFiクラス

`System.WiFi()`を呼んでおく、または`System.use('WiFi')`しておく必要があります。

### http GET結果をSDカードに保存する: WiFi.httpGetSD(Filename, URL[, Headers])

http GET結果をSDカードに保存します。

Filename: 保存するファイル名

URL: URL

Headers: ヘッダに追記する文字列の配列

戻り値

0: 失敗

1: 成功

2: SDカードが使えない

3: 送信データファイルをオープンできなかった

4: 送信データサイズを読み込めなかった

5: 送信データファイルを読み込めなかった

6: 受信用ファイルをオープンできなかった

7: 受信したファイルの生成に失敗した

### http GETプロトコルを送信する: WiFi.httpGet(URL[, Headers])

http GETプロトコルを送信します。送信のみで、結果の受信しません。

URL: URL

Headers: ヘッダに追記する文字列の配列

戻り値

0: 失敗

1: 成功

# メソッドの説明(V2ライブラリ)

## WiFiクラス

`System.WiFi()`を呼んでおく、または`System.use('WiFi')`しておく必要があります。

### TCP/UDPの接続を閉じる: WiFi.cClose(number)

TCP/UDPの接続を閉じます。

number: 接続番号(1~4)

戻り値

ESP8266の戻り値

### UDP接続を開始する: WiFi.udpOpen(number, IP\_Address, SendPort, ReceivePort)

UDP接続を開始します。

number: 接続番号(1~4)

IP\_Address: 通信相手アドレス

SendPort: 送信ポート番号

ReceivePort: 受信ポート番号

戻り値

ESP8266の戻り値

### 指定接続番号にデータを送信する: WiFi.send(number, Data[, length])

指定接続番号にデータを送信します。

number: 接続番号(1~4)

Data: 送信するデータ

length: 送信データサイズ

戻り値

送信データサイズ

# メソッドの説明(V2ライブラリ)

## WiFiクラス

`System.WiFi()`を呼んでおく、または`System.use('WiFi')`しておく必要があります。

### 指定接続番号からデータを受信する: WiFi.recv(number)

指定接続番号からデータを受信します。

number: 接続番号(1~4)

戻り値

受信したデータの配列 ただし、256以下

### SDカードのファイルをhttpPOSTする: WiFi.httpPostSD(URL, Headers, Filename)

SDカードのファイルをhttp POSTします。

URL: URL

Headers: ヘッダに追記する文字列の配列

Filename: POSTするファイル名

戻り値

0: 失敗

1: 成功

2: SDカードが使えない

2: SDカードが使えない

3: 送信データファイルサイズを読み込めなかった

4: 送信ヘッダファイルを書き込みオープンできなかった

5: 送信ヘッダファイルサイズを読み込めなかった

6: 送信ヘッダファイルを読み込みオープンできなかった

7: 送信データファイルを読み込みオープンできなかった

# メソッドの説明(V2ライブラリ)

## WiFiクラス

`System.WiFi()`を呼んでおく、または`System.use('WiFi')`しておく必要があります。

### http POSTする: WiFi.httpPost(URL, Headers, data)

http POSTします。送信のみで結果は受信しません。

URL: URL

Headers: ヘッダに追記する文字列の配列

Data: POSTデータ

戻り値

0: 失敗

1: 成功

### httpサーバを開始します: WiFi.httpServer([Port])

httpサーバを開始します。アクセスの有無で返り値が変わります。

SDカードが必須となります。

ポート番号を省略したときはアクセス確認します。

Port: 待ち受けポート番号

-1: サーバ停止

戻り値

0: アクセスはありません

1: アクセスあり

2: SDカードが使えません

3: ファイルのアクセスに失敗しました

クライアントからアクセスがあるとき

GET: パスが返ります

GET以外、ヘッダの1行目が返ります

# メソッドの説明(V2ライブラリ)

## WiFiクラス

System.WiFi()を呼んでおく、またはSystem.use('WiFi')しておく必要があります。

### 使用例

```
#ESP8266を一度停止させる(リセットと同じ)
pinMode(5, 1)
digitalWrite(5, 0)    # LOW:Disable
delay 500
digitalWrite(5, 1)    # LOW:Disable

Usb = Serial.new(0, 115200)

if( System.use?('WiFi') == false)then
  Usb.println "WiFi Card can't use."
  System.exit()
end

Usb.print WiFi.version
```

# メソッドの説明(V2ライブラリ)

## WiFiクラス

System.WiFi()を呼んでおく、またはSystem.use('WiFi')しておく必要があります。

### 使用例

```
#ESP8266を一度停止させる(リセットと同じ)
pinMode(5, 1)
digitalWrite(5, 0)    # LOW:Disable
delay 500
digitalWrite(5, 1)    # LOW:Disable

Usb = Serial.new(0, 115200)

if( System.use?('WiFi') == false)then
  Usb.println "WiFi Card can't use."
  System.exit()
end

Usb.println "GR-CITRUS[bypass]"

WiFi.bypass()

#GR-CITRUSはESP8266とUSBシリアル通信が接続したままとなります。
#ターミナルを使って、ESP8266との通信テストをすることができます。
```

# メソッドの説明(V2ライブラリ)

## WiFiクラス

System.WiFi()を呼んでおく、またはSystem.use('WiFi')しておく必要があります。

### 使用例

```
#ESP8266を一度停止させる(リセットと同じ)
pinMode(5, 1)
digitalWrite(5, 0)    # LOW:Disable
delay 500
digitalWrite(5, 1)    # LOW:Disable

Usb = Serial.new(0, 115200)

if( System.use?('WiFi') == false)then
  Usb.println "WiFi Card can't use."
  System.exit()
end

Usb.println WiFi.disconnect
Usb.println WiFi.setMode 3 #Station-Mode & SoftAPI-Mode
Usb.println WiFi.connect("TAROSAY", "***")
Usb.println WiFi.ipconfig
Usb.println WiFi.multiConnect 1

for value in 1..10
  Usb.println WiFi.httpGet("192.168.1.58:3000/?value1=" + value.to_s + "&value2=" +
  (value*value).to_s).to_s
end

heds=[User-Agent: curl"]
Usb.println WiFi.httpGetSD("wether1.htm", "wttr.in/wakayama").to_s
Usb.println WiFi.httpGetSD("wether2.htm", "wttr.in/wakayama", heds).to_s
Usb.println WiFi.httpGetSD("yahoo.htm", "www.yahoo.co.jp").to_s
Usb.println WiFi.httpGetSD("google.htm", "www.google.co.jp").to_s
```

# メソッドの説明(V2ライブラリ)

## WiFiクラス

System.WiFi()を呼んでおく、またはSystem.use('WiFi')しておく必要があります。

### 使用例

```
#UDP通信, WA-MIKAN 受信:5555, 送信: 5556
Usb.println WiFi.udpOpen(4, "192.168.1.44", 5555, 5556)

Usb.println "UDP受信した分がarray配列で返ります"
1000.times do
  array = WiFi.recv 4  #受信データがない場合は array[0]に -1 が返ります
  if(array[0] >= 0)then
    for var in array do
      Usb.println var.to_s
    end
  end
  delay 10
end
```

```
#UDP通信, WA-MIKAN 受信:5555, 送信: 5556
Usb.println WiFi.udpOpen(4, "192.168.1.44", 5555, 5556)

100.times do
  WiFi.send 4, "hoho0111122222¥r¥n"
  delay 25
end
Usb.println WiFi.send(4, 0x02.chr + "bcdefghijklmn" + 0x03.chr + "ddd¥r¥n").to_s

Usb.println WiFi.cClose 4
Usb.println WiFi.disconnect
```

# メソッドの説明(V2ライブラリ)

## WiFiクラス

System.WiFi()を呼んでおく、またはSystem.use('WiFi')しておく必要があります。

### 使用例

```
#http POST  
  
header=[“User-Agent: gr-citrus”, “Accept: application/json”, “Content-type: application/json”]  
body = ’{ “name” : “tarosay” }’  
  
WiFi.httpPost(“192.168.1.52:3000”, header, body)  
  
# body.json ファイルをPOSTします  
WiFi.httpPostSD(“192.168.1.52:3000”, header, “body.json”)
```

# メソッドの説明(V2ライブラリ)

## WiFiクラス

System.useWiFi()を呼んでおく必要があります。

### 使用例

```

header0 = "HTTP/1.1 200 OK\r\nServer: GR-CITRUS\r\nContent-Type: text/html\r\n"
header0 += "Date: Sun, 13 Nov 2016 12:00:00 GMT\r\nConnection: close\r\n"
body0 = '<html><head><meta http-equiv="Content-Type" content="text/html; charset=utf-8">'
body0 += '<title>RAMUNE SHOOTER</title></head>'

Usb.println WiFi.httpServer(80).to_s #-> 80ポートでhttp受信します
while(true) do
  res = WiFi.httpServer #-> アクセス確認しています。
  Usb.println res.to_s #-> 0のときはアクセスなし、GETのときはパスをそれ以外はヘッダ先頭行が返る

  if(res == "/exit")
    body1 = '<body><h1 align="center">終了します。</h1></body>' + "\r\n\r\n"
    header1 = "Content-Length: " + (body0 + body1).length.to_s + "\r\n\r\n"
    WiFi.send(0, header0)
    WiFi.send(0, header1)
    WiFi.send(0, body0)
    WiFi.send(0, body1)
    break
  elsif(res != 0)
    Usb.println "Else:" + res.to_s
    body1 = '<body><h1 align="center">エラーです。</h1></body>' + "\r\n\r\n"
    header1 = "Content-Length: " + (body0 + body1).length.to_s + "\r\n\r\n"
    WiFi.send(0, header0)
    WiFi.send(0, header1)
    WiFi.send(0, body0)
    WiFi.send(0, body1)
  end
  delay 100
end
WiFi.httpServer(-1) #-> サーバを停止します

```

# メソッドの説明(V2ライブラリ)

## MP3クラス

System. useMP3(pausePin, stopPin) を呼んでおく、または  
System. use(' MP3' , [pausePin, stopPin]) しておく必要があります。

### MP3ファイルを再生する: MP3. play(filename)

MP3ファイルまたはwavファイルを再生します。  
0番ピンとGNDの間にスピーカーを接続してください。

filename: 再生するMP3ファイル名またはwavファイル名

#### 戻り値

エラーが出たときは、その内容が返ります。エラーが無いときは何も返りません。

System. useMP3(pausePin, stopPin) で設定したピン番号の入力をLOWにすることによって、曲の一時停止や終了を行うことができます。

設定できるピンは、1, 3, 4, 6, 9, 10, 14, 15, 16, 17, 18番ピンの11個です。

### MP3再生中にLEDを点滅させる: MP3. led(sw)

MP3再生中にLEDを点滅させます。

sw: 0:何もしない、1:点滅させる

ポーズ中はLEDは点灯した状態となります。

# メソッドの説明(V2ライブラリ)

## MP3クラス

System. useMP3(pausePin, stopPin) を呼んでおく、または  
System. use(' MP3' , [pausePin, stopPin]) しておく必要があります。

### 使用例

```
Usb = Serial. new(0, 115200)

#3番ピンを一時停止に、4番ピンを再生停止ボタンに設定します。

if(System. use?(' MP3' , [3, 4]) == false)then
    Usb. println "MP3 can't use."
    System. exit()
end
Usb. println "MP3 Ready"

MP3. led 1
Usb. println "/koidance.mp3"
Usb. print MP3. play "/koidance.mp3"

MP3. led 0
Usb. println "/decrain.mp3"
Usb. print MP3. play "/decrain.mp3"
```

# メソッドの説明(V2ライブラリ)

## TFTcクラス

`System.useTFTc()` を呼んでおく必要があります。

プログラムを終了時には`System.clearTFTc()` を呼ぶ必要があります。

### 画面を指定色で塗りつぶす: `TFTc.fillRectScreen(color)`

画面を `color` で指定した色で塗りつぶします。

`color`: 16bit(RGB565) 値

戻り値  
なし

※16bit カラー値はMSBより 5bits red, 6bits green, 5bits blueの並び。

※下記の定義を使用可能。

黒色	<code>C_BLACK</code>	(値:0x0000)
茶色	<code>C_BROWN</code>	(値:0xcb43)
赤色	<code>C_RED</code>	(値:0xf800)
橙色	<code>C_ORANGE</code>	(値:0xfd20)
黄色	<code>C_YELLOW</code>	(値:0xffe0)
緑色	<code>C_LIME</code>	(値:0x07e0)
青色	<code>C_BLUE</code>	(値:0x001f)
紫色	<code>C_VIOLET</code>	(値:0x901a)
灰色	<code>C_GRAY</code>	(値:0xa554)
白色	<code>C_WHITE</code>	(値:0xffff)
水色	<code>C_CYAN</code>	(値:0x07ff)
マゼンタ	<code>C_MAGENTA</code>	(値:0xf81f)

※メソッドにて数値での色指定も可能（参照：`TFTc.color`）

# メソッドの説明(V2ライブラリ)

## TFTcクラス

`System.useTFTc()` を呼んでおく必要があります。

プログラムを終了時には`System.clearTFTc()` を呼ぶ必要があります。

16bitカラー値を得る: `TFTc.color(color_no)`

整数のカラーナンバーから、16bitカラー値を得ます。

`color_no`: 0~11 の値 (下表参照)

戻り値

16bitカラー値

※カラーナンバー定義

カラーナンバー 0:	黒色	(値:0x0000)
カラーナンバー 1:	茶色	(値:0xcb43)
カラーナンバー 2:	赤色	(値:0xf800)
カラーナンバー 3:	橙色	(値:0xfd20)
カラーナンバー 4:	黄色	(値:0xffe0)
カラーナンバー 5:	緑色	(値:0x07e0)
カラーナンバー 6:	青色	(値:0x001f)
カラーナンバー 7:	紫色	(値:0x901a)
カラーナンバー 8:	灰色	(値:0xa554)
カラーナンバー 9:	白色	(値:0xffff)
カラーナンバー 10:	水色	(値:0x07ff)
カラーナンバー 11:	マゼンタ	(値:0xf81f)

## メソッドの説明(V2ライブラリ)

### TFTcクラス

System.useTFTc() を呼んでおく必要があります。

プログラムを終了時にはSystem.clearTFTc() を呼ぶ必要があります。

#### 画面を回転する: TFTc.rotation(rotation)

画面を指定したrotation値に回転します。

rotation: 0:垂直 1:水平 2: 垂直反転 3:水平反転

戻り値  
なし

※ 設定した値を呼び出すメソッドは実装していません。

※ 初期値は0です。

#### 画面の横幅を取得する: TFTc.width()

画面の横幅を取得します。

戻り値は横幅のピクセル数

#### 画面の縦幅を取得する: TFTc.height()

画面の縦幅を取得します。

戻り値は縦幅のピクセル数

# メソッドの説明(V2ライブラリ)

## TFTcクラス

`System.useTFTc()`を呼んでおく必要があります。

プログラムを終了時には`System.clearTFTc()`を呼ぶ必要があります。

### ビットマップの表示: `TFTc.drawBmp(filename, x, y)`

SDカードにあるビットマップファイルを、指定の座標に表示します。

`filename`: ファイル名(8.3形式)

`x`: 表示させるビットマップの左上のx座標

`y`: 表示させるビットマップの左上のy座標

戻り値

0: 描画成功

1: 失敗 (ビットマップサイズが画面よりも大きい)

2: 失敗 (ファイルが存在しない)

3: 失敗 (サポート外のファイル形式)

### ドットを描画する: `TFTc.drawPixel(x, y, [, color])`

指定座標に(指定した色で)ドットを描画します。

`x`: ドットを描画するx座標

`y`: ドットを描画するy座標

`color`: 描画するドットの16bitカラー値

戻り値

なし

※ `color`を省略した場合は、最後にグラフィックを描画した色で描画します。

(ビットマップやテキストの描画色は除きます) 初期値は0x0000(黒色)です。

※ 描画終了点は次のグラフィックが描画されるまで保持します。

(ビットマップやテキストの描画色は除きます)

## メソッドの説明(V2ライブラリ)

### TFTcクラス

System.useTFTc() を呼んでおく必要があります。

プログラムを終了時にはSystem.clearTFTc() を呼ぶ必要があります。

線を描画する: TFTc.drawLine(x0, y0, x1, y1[, color])

2点の指定座標を結ぶ線を（指定した色で）描画します。

x0: 直線の描画開始点のx座標

y0: 直線の描画開始点のy座標

x1: 直線の描画終了点のx座標

y1: 直線の描画終了点のy座標

color: 描画する直線の16bitカラー値

戻り値

なし

※ colorを省略した場合は、最後にグラフィックを描画した色で描画します。

(ビットマップやテキストの描画色は除きます) 初期値は0x0000(黒色)です。

※ 描画終了点は次のグラフィックが描画されるまで保持します。

(ビットマップやテキストの描画色は除きます)

## メソッドの説明(V2ライブラリ)

### TFTcクラス

System.useTFTc() を呼んでおく必要があります。

プログラムを終了時にはSystem.clearTFTc() を呼ぶ必要があります。

連続して線を描画する: TFTc.drawLineto(x, y[, color])

直前に描画した線の最終座標と指定座標を結ぶ直線を（指定した色で）描画します。

x: 直線の描画終了点のx座標

y: 直線の描画終了点のy座標

color: 描画する直線の16bitカラー値

戻り値

なし

※ colorを省略した場合は、最後にグラフィックを描画した色で描画します。

（ビットマップやテキストの描画色は除きます） 初期値は0x0000（黒色）です。

※ 描画終了点は次のグラフィックが描画されるまで保持します。

（ビットマップやテキストの描画色は除きます）

# メソッドの説明(V2ライブラリ)

## TFTcクラス

System.useTFTc() を呼んでおく必要があります。

プログラムを終了時にはSystem.clearTFTc() を呼ぶ必要があります。

円を描画する: TFTc.drawCircle(x, y, r[, color])

塗りつぶしの円を描画する: TFTc.fillRect(x, y, r[, color])

円を（指定した色で）描画します。

x: 円の中心点のx座標

y: 円の中心点のy座標

r: 円の半径(dot)

color: 描画する円の16bitカラー値

戻り値

なし

※ colorを省略した場合は、最後にグラフィックを描画した色で描画します。

（ビットマップやテキストの描画色は除きます） 初期値は0x0000（黒色）です。

※ 描画した円の中心座標は次のグラフィックが描画されるまで保持します。

（ビットマップやテキストの描画色は除きます）

# メソッドの説明(V2ライブラリ)

## TFTcクラス

System.useTFTc()を呼んでおく必要があります。

プログラムを終了時にはSystem.clearTFTc()を呼ぶ必要があります。

四角形を描画する: TFTc.drawRect(x, y, w, h[, color])

塗りつぶしの四角形を描画する: TFTc.fillRect(x, y, w, h[, color])

四角形を（指定した色で）描画します。

x: 四角形の左上のx座標

y: 四角形の左上のy座標

w: 四角形の幅(dot)

h: 四角形の高さ(dot)

color: 描画する四角形の16bitカラー値

戻り値

なし

※ colorを省略した場合は、最後にグラフィックを描画した色で描画します。

(ビットマップやテキストの描画色は除きます) 初期値は0x0000(黒色)です。

※ 四角形の左上の座標は次のグラフィックが描画されるまで保持します。

(ビットマップやテキストの描画色は除きます)

# メソッドの説明(V2ライブラリ)

## TFTcクラス

`System.useTFTc()`を呼んでおく必要があります。

プログラムを終了時には`System.clearTFTc()`を呼ぶ必要があります。

角の丸い四角形を描画する: `TFTc.drawRoundRect(x, y, w, h, r [, color])`

塗りつぶしの角の丸い四角形を描画する: `TFTc.fillRectRoundRect(x, y, w, h, r [, color])`

角の丸い四角形を（指定した色で）描画します。

x: 四角形の左上のx座標

y: 四角形の左上のy座標

w: 四角形の幅(dot)

h: 四角形の高さ(dot)

r: 四角形の角の半径(dot)

color: 描画する四角形の16bitカラー値

戻り値

なし

※ `color`を省略した場合は、最後にグラフィックを描画した色で描画します。

(ビットマップやテキストの描画色は除きます) 初期値は0x0000(黒色)です。

※ 四角形の左上の座標は次のグラフィックが描画されるまで保持します。

(ビットマップやテキストの描画色は除きます)

# メソッドの説明(V2ライブラリ)

## TFTcクラス

`System.useTFTc()` を呼んでおく必要があります。

プログラムを終了時には`System.clearTFTc()` を呼ぶ必要があります。

三角形を描画する: `TFTc.drawTriangle(x0, y0, x1, y1, x2, y2[, color])`

塗りつぶしの三角形を描画する: `TFTc.fillTriangle(x0, y0, x1, y1, x2, y2[, color])`

三角形を（指定した色で）描画します。

`x0`: 三角形の第1頂点のx座標

`y0`: 三角形の第1頂点のy座標

`x1`: 三角形の第2頂点のx座標

`y1`: 三角形の第2頂点のy座標

`x2`: 三角形の第3頂点のx座標

`y2`: 三角形の第3頂点のy座標

`color`: 描画する三角形の16bitカラー値

戻り値

なし

※ `color`を省略した場合は、最後にグラフィックを描画した色で描画します。

（ビットマップやテキストの描画色は除きます） 初期値は0x0000（黒色）です。

※ 四角形の第1頂点の座標は次のグラフィックが描画されるまで保持します。

（ビットマップやテキストの描画色は除きます）

# メソッドの説明(V2ライブラリ)

## TFTcクラス

`System.useTFTc()`を呼んでおく必要があります。

プログラムを終了時には`System.clearTFTc()`を呼ぶ必要があります。

### テキストを1文字描画する: `TFTc.drawChar(x, y, value, color, bgcolor, size)`

指定の位置に5x7サイズのアスキー文字を1文字、指定の文字色と背景色で描画します。

背景色の描画範囲は、文字の左上を基準にした6x8サイズとなります。

x: 文字の左上のx座標

y: 文字の左上のy座標

value: キャラクタ値（アスキーコード）

color: 描画する文字の16bitカラー値

bgcolor: 文字の背景色の16bitカラー値

size: 描画する文字の拡大率(1以上の整数)

戻り値

なし

### 文字列の描画位置を設定する: `TFTc.textCursor(x, y)`

文字列を描画するにあたり、カーソル位置の座標を設定します。

x: あとで描画する文字列の左上のx座標

y: あとで描画する文字列の左上のy座標

戻り値

なし

※ 文字列の描画は、`TFTc.print`, `TFTc.println` メソッドを用います。

※ 初期値は x=0, y=0 です。

# メソッドの説明(V2ライブラリ)

## TFTcクラス

`System.useTFTc()` を呼んでおく必要があります。

プログラムを終了時には`System.clearTFTc()` を呼ぶ必要があります。

### 文字列の描画色を設定する: `TFTc.textColor(color[, bgcolor])`

文字列を描画するにあたり、文字色（および背景色）を設定します。

`color`: あとで描画する文字列の16bitカラー値

`bgcolor`: あとで描画する文字列の背景色の16bitカラー値

戻り値  
なし

※ 文字列の描画は、`TFTc.print`, `TFTc.println` メソッドを用います。

※ 初期値は `color`、`bgcolor`ともに白 (=0xFFFF) です。

### 文字列のサイズを設定する: `TFTc.setTextSize(size)`

文字列を描画するにあたり、文字のサイズ（拡大率）を設定します。

`size`: 描画する文字列の拡大率(1以上の整数)

戻り値  
なし

※ 文字列の描画は、`TFTc.print`, `TFTc.println` メソッドを用います。

※ 初期値は 1 です。

## メソッドの説明(V2ライブラリ)

### TFTcクラス

`System.useTFTc()` を呼んでおく必要があります。

プログラムを終了時には`System.clearTFTc()` を呼ぶ必要があります。

#### 文字列を改行なしで描画する: `TFTc.print([str])`

予め指定したカーソル位置に、予め設定した文字色、文字サイズで、指定の文字列を改行なしで描画します。

[str]: アスキーワード

戻り値  
なし

#### 文字列を描画し改行する: `TFTc.println([str])`

予め指定したカーソル位置に、予め設定した文字色、文字サイズで、指定の文字列を描画し、改行します。

[str]: アスキーワード

戻り値  
なし

#### 文字列の折り返しを設定する: `TFTc.textWrap(mode)`

文字列を描画する際、画面の右端に差し掛かった場合に折り返すか折り返さないかを設定します。

mode: 0: 折り返さない 1: 折り返す

戻り値  
なし

※ 初期値は 1 です。

## メソッドの説明(V2ライブラリ)

### TFTcクラス

System.useTFTc()を呼んでおく必要があります。

プログラムを終了時にはSystem.clearTFTc()を呼ぶ必要があります。

文字および文字列描画で使用するフォントを設定する: TFTc.textFont(fontno)

【オプション機能】 使用するフォントを設定します。

fontno: フォントno 0~48

戻り値

なし

※ファームウェアサイズを圧縮するため、デフォルトでは標準フォント (fontno=0) 以外は使えません。

必要に応じて sTFTc.cpp、およびsTFTc.hのコメントアウトを削除し、有効にしてください。

## メソッドの説明(V2ライブラリ)

### TFTcクラス

`System.useTFTc()`を呼んでおく必要があります。

プログラムを終了時には`System.clearTFTc()`を呼ぶ必要があります。

#### タッチを検出する: `TFTc.panelTouched()`

タッチパネルがタッチされているかどうかを検出します。タッチされている場合は内部変数にタッチされた座標が記録されます。

戻り値

- 0: タッチされていない
- 1: タッチされている

※画面回転の設定に応じて座標を自動変換します。

#### タッチされた座標のx値を得る: `TFTc.touchedX()`

`TFTc.panelTouched()`関数で1が返された場合、本メソッドを用いてタッチされた座標のx値を得ることができます。

戻り値は、タッチされた座標のxの値

#### タッチされた座標のy値を得る: `TFTc.touchedY()`

`TFTc.panelTouched()`関数で1が返された場合、本メソッドを用いてタッチされた座標のy値を得ることができます。

戻り値は、タッチされた座標のyの値

## メソッドの説明(V2ライブラリ)

### TFTcクラス

`System.useTFTc()` を呼んでおく必要があります。

プログラムを終了時には`System.clearTFTc()` を呼ぶ必要があります。

#### ボタンの最大数を設定する: `TFTc.setButtonMax(buttonmax)`

使用するボタンの最大数を設定します。（ボタン構造体のインスタンスを生成するためメモリを消費します）  
プログラムで複数画面を使用する場合、画面毎にボタンの数が異なるため、一番多いボタン数のメモリをこのメソッドで確保します。

`buttonmax`: プログラムで使用するボタン数の最大値(1~32)

戻り値は、確保したボタン構造体の数

※ 最大のボタン数は 32個です。これ以上のボタン数を指定した場合は32個に制限されます。

#### 設定したボタンの最大値をクリアする: `TFTc.deleteButtons()`

確保してあるボタン構造体のインスタンスを開放します。`TFTc.setButtonMax` で設定した最大値を変更したい場合はこの手続を実施したあとで設定し直します。

戻り値は、開放したボタン構造体の数

※ 最大のボタン数は 32個です。これ以上のボタン数を指定した場合は32個に制限されます。

# メソッドの説明(V2ライブラリ)

## TFTcクラス

`System.useTFTc()`を呼んでおく必要があります。

プログラムを終了時には`System.clearTFTc()`を呼ぶ必要があります。

使用するボタンの数を設定する: `TFTc.setDisplayButtonMax(buttonmax)`

画面に表示し、制御するボタンの数を設定します。

`buttonmax`: 画面に表示するボタンの数(1~32)

戻り値は、設定したボタン数

※設定するボタンの数は、`TFTc.setButtonMax`メソッドで設定した数を超えてはなりません。

ボタンを個別に初期化する: `TFTc.initButton(no, x, y, w, h, edgec, backc, textc, [str], size)`

表示および制御するボタンを個別に初期化します。

`no`: ボタンno (1~32)

`x`: ボタンの中央のx座標

`y`: ボタンの中央のy座標

`w`: ボタンの幅

`h`: ボタンの高さ

`edgec`: ボタンの縁の16bitカラー値

`backc`: ボタンの背景の16bitカラー値

`textc`: ボタンの中に表示する文字の16bitカラー値

`[str]`: ボタンの中に表示するアスキーワード

`size`: ボタンの中に描画する文字の拡大率(1以上の整数)

戻り値

0: 正しく設定できなかった場合 1: 正しく設定できた場合

※設定するボタンnoは、`TFTc.setDisplayButtonMax`メソッドで設定した数を超えてはなりません。

## メソッドの説明(V2ライブラリ)

### TFTcクラス

`System.useTFTc()`を呼んでおく必要があります。

プログラムを終了時には`System.clearTFTc()`を呼ぶ必要があります。

全てのボタンを描画する: `TFTc.drawButtons()`

`TFTc.setDisplayButtonMax`で設定したボタン数のボタンを、全て通常表示で描画します。

`TFTc.initButton`で個別に全てのボタンを設定した後、このメソッドを用いて最初の描画を行います。

戻り値  
なし

ボタンを個別に描画する: `TFTc.drawButton(no, invert)`

ボタンを個別に描画します。

`no`: ボタンno (1~32)

`invert`: 反転表示フラグ 0:反転しない(通常表示) 1:反転する(反転表示)

戻り値  
なし

ボタンがタッチされたどうか調べる: `TFTc.searchTouchedButton(x, y)`

タッチが検出された場合に、その座標がボタンかどうかを調べます。

検索の結果は、各ボタンの状態変数に記録されます。

`x`: タッチ検出位置のx座標

`y`: タッチ検出位置のy座標

戻り値  
なし

## メソッドの説明(V2ライブラリ)

### TFTcクラス

`System.useTFTc()`を呼んでおく必要があります。

プログラムを終了時には`System.clearTFTc()`を呼ぶ必要があります。

#### 個別にボタンの変化を調べる: `TFTc.searchButtonState(no)`

指定したボタンがリリース状態から押下状態に変化したか、押下状態からリリース状態に変化したか、それとも変化がないか、判定を行う。

また、押下状態からリリース状態に変化した場合は、ボタンの描画を通常の描画に変更する。

リリース状態から押下状態に変化した場合は、ボタンの描画を反転させる。

どちらでもない場合（変化がない場合）は、描画を変更しない。

no: ボタンno

戻り値

1:押下→リリースと変化 -1:リリース→押下と変化 0:変化なし

※同時押しや長押し検出など、細かな制御を行いたい場合に使用します。

#### 押下されたボタンを検出する: `TFTc.searchPressedButton()`

`TFTc.setDisplayButtonMax`メソッドで設定したボタン数のボタンについて、リリース状態から押下状態へ変化したボタンnoを検出する。

押下されたボタンは自動的に反転表示となる。また、押下されていないボタンは自動的に通常表示となる。

戻り値

0:どのボタンも押下されていない 1~32:数値のボタンがリリース状態から押下状態へ変化

## メソッドの説明(V2ライブラリ)

### TFTcクラス

System.useTFTc() を呼んでおく必要があります。

プログラムを終了時にはSystem.clearTFTc() を呼ぶ必要があります。

リリースされたボタンを検出する: TFTc.searchReleasedButton

TFTc.setDisplayButtonMax メソッドで設定したボタン数のボタンについて、押下状態からリリース状態へ変化したボタンnoを検出する。

押下されたボタンは自動的に反転表示となる。また、押下されていないボタンは自動的に通常表示となる。

戻り値

0:どのボタンもリリースされていない 1~32:数値のボタンが押下状態からリリース状態へ変化

# メソッドの説明(V2ライブラリ)

## TFTcクラス

System.useTFTc() を呼んでおく必要があります。  
プログラムを終了時にはSystem.clearTFTc() を呼ぶ必要があります。

### 使用例

※ 現状のファームウェアに、SDカードアクセス時に10pinよりノイズが outputされる不具合があります。  
そのため、このpinをCS信号として使用しているTFTタッチパネル液晶にゴミが描画されてしまう場合があります。  
バグが取り切れていない現状では、下記のコードをrubyコードの最初に挿入することで回避できます。

```
# 初期化開始 (SDカードとTFTcapacitiveタッチパネルの共存)
# 10pinからのノイズ出力の不具合対策
# 予めSDカードにsample.txtを作り、カードを挿入しておく
pinMode(10, 1)      # pin10 output
digitalWrite(10, 1)  # pin10をHighに
USB.println("Checking SD")
if (0 == System.useSD()) then
    USB.println("SD card is not inserted.")
else
    USB.println("SD card is ready.")
    if (0==SD.open(0,"sample.txt",0)) then
        while(true) do
            c = SD.read(0)
            if(c < 0) then
                break
            end
            USB.print(c.chr)
        end
        SD.close(0)
    end
end
tone(10, 1000, 100)
digitalWrite(10, 1)  # pin10をHighに (tone off)
delay(100)
```

```
if(1==System.useTFTc()) then
    TFTc.drawLine(1, 1, TFTc.color(1))
end
USB.println("Checking SD") #2回目
if (0 == System.useSD()) then
    USB.println("SD card is not inserted.")
else
    USB.println("SD card is ready.")
    if (0==SD.open(0,"sample.txt",0)) then
        while(true) do
            c = SD.read(0)
            if(c < 0) then
                break
            end
            USB.print(c.chr)
        end
        SD.close(0)
    end
end
tone(10, 1000, 100)
digitalWrite(10, 1)  # pin10をHighに (tone off)
USB.println("Clear screen")
if(1==System.useTFTc()) then
    TFTc.fillRect(TFTc.color(0))
end #if
```

# メソッドの説明(V2ライブラリ)

## TFTcクラス

System.useTFTc() を呼んでおく必要があります。

プログラムを終了時にはSystem.clrTFTc() を呼ぶ必要があります。

### 使用例

# 基本的なグラフィック描画メソッドの動作

```
if(1==System.useTFTc()) then
```

#drawCircle & fillCircle test

```
x=50;y=50;
TFTc.drawCircle(x,y,15,TFTc.color(6))
TFTc.drawCircle(x,y+5,12,TFTc.color(7))
TFTc.fillRect(x,y+10,10,C_YELLOW)
TFTc.fillRect(x,y,5,C_GRAY)
```

#drawPixel test

```
x=100;y=50;a=20;
TFTc.drawPixel(x,y,TFTc.color(1))
TFTc.drawPixel(x,y+a)
TFTc.drawPixel(x+a,y+a)
TFTc.drawPixel(x+a,y,0xcb43)
TFTc.drawPixel(x+a/2,y+a/2,TFTc.color(4))
```

#drawLine , drawLineto test

```
x=150;y=40;
10.step(35,5){|a|
  TFTc.drawLine(x,y,x,y+a,TFTc.color(5))
  TFTc.drawLineto(x+a,y+a)
  TFTc.drawLineto(x+a,y)
  TFTc.drawLineto(x,y)
  TFTc.drawLineto(x+a/2,y+a/2,TFTc.color(7))
}
```

#drawRect & fillRect test

```
x=50;y=100;
TFTc.drawRect(x,y,10,4,TFTc.color(8))
TFTc.drawRect(x+5,y+5,20,8)
```

TFTc.fillRect(x+10,y+10,15,20)

TFTc.fillRect(x+20,y+15,5,10,TFTc.color(9))

#drawRoundRect & fillRoundRect test

```
x=100;y=100;
TFTc.drawRoundRect(x,y,20,14,5,TFTc.color(8))
TFTc.drawRoundRect(x+5,y+5,25,20,4)
TFTc.fillRect(x+10,y+10,15,30,3)
TFTc.fillRect(x+20,y+15,5,10,2,TFTc.color(9))
```

#drawTriangle & fillTriangle test

```
x=180;y=110;
TFTc.fillTriangle(x,y,x+5,y+10,x-5,y+10,TFTc.color(2))
x=150;y=100;
TFTc.drawTriangle(x+10,y+10,x+15,y+30,x+5,y+30,C_ORANGE)
TFTc.fillTriangle(x+15,y+20,x+23,y+35,x+7,y+35,C_BROWN)
TFTc.drawTriangle(x+25,y+30,x+35,y+45,x+15,y+45,C_YELLOW)
```

TFTc.drawRoundRect(20,20,200,150,5,TFTc.color(8))

```
x=120;y=240;
for i in 1..5 do
  for j in 1..5 do
    TFTc.drawCircle(x+j*2,y+j*2,i*10,TFTc.color(6)-j*3-i*2)
  end
end
```

System.clrTFTc()

end #if

# メソッドの説明(V2ライブラリ)

## TFTcクラス

System.useTFTc()を呼んでおく必要があります。

プログラムを終了時にはSystem.clrTFTc()を呼ぶ必要があります。

### 使用例

```
#Screen rotation test

if(1==System.useTFTc()) then
    for @Rot in 0..4 do
        TFTc.fillRect(TFTc.color(11))
        TFTc.fillRect(0x1111)
        TFTc.textCursor(0, 0)
        TFTc.print("abcd")
        w=TFTc.width()
        h=TFTc.height()
        Usb.print("w=");
        Usb.print(w.to_s);
        Usb.print("h=");
        Usb.println(h.to_s);
        @Rot += 1
        if(4==@Rot) then
            @Rot = 0
        end
        TFTc.rotation(@Rot)
        Usb.print("rotation");
        Usb.println(@Rot.to_s);
        w=TFTc.width()
        h=TFTc.height()
        Usb.print("w=");
        Usb.print(w.to_s);
        Usb.print("h=");
        Usb.println(h.to_s);
        delay(1000)
    end #for
    System.clrTFTc()
end #if
```

# メソッドの説明(V2ライブラリ)

## TFTcクラス

`System.useTFTc()`を呼んでおく必要があります。

プログラムを終了時には`System.clrTFTc()`を呼ぶ必要があります。

### 使用例

```
#Font test
if(1==System.useTFTc()) then
    TFTc.fillRect(TFTc.color(0))
    TFTc.textFont(0)      # デフォルトフォントに設定
    TFTc.setTextSize(0)   # 倍率を1に設定
    TFTc.textCursor(10, 0)
    TFTc.setTextColor(TFTc.color(2))
    TFTc.print("Size:0, RED ")
    TFTc.setTextColor(TFTc.color(4))
    TFTc.println("YELLOW")
    TFTc.setTextColor(TFTc.color(3))
    TFTc.textWrap(0)
    TFTc.println("Text Wrap 0, ORANGE:")
    TFTc.println("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz_0123456789. !?#")
    hijklmnopqrstuvwxyz_0123456789. !?#
    TFTc.textWrap(1)
    TFTc.setTextColor(TFTc.color(5))
    TFTc.println("Text Wrap 1, GREEN :");
    TFTc.println("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz_0123456789. !?#")
    hijklmnopqrstuvwxyz_0123456789. !?#
    TFTc.setTextSize(1)
    TFTc.setTextColor(TFTc.color(6))
    TFTc.println("Size:1, BLUE: ABC")
    TFTc.setTextSize(2)
    TFTc.setTextColor(TFTc.color(7))
    TFTc.println("Size:2, PURPLE: ABC")
    TFTc.setTextSize(3)
    TFTc.setTextColor(TFTc.color(8))
    TFTc.println("Size:3, GRAY: ABC")
    TFTc.setTextSize(1)
    TFTc.textCursor(50, 130)
    TFTc.println("Size:1, cursor (50, 100) [CR]")
    TFTc.setTextColor(TFTc.color(9))
    TFTc.print("White:ABCD")
    TFTc.setTextColor(TFTc.color(10))

    TFTc.println("CYAN:EFGHI")
    TFTc.setTextSize(2)
    TFTc.setTextColor(TFTc.color(0), TFTc.color(2))
    TFTc.print("          ")
    TFTc.print("      BLACK/RED      ")
    TFTc.print("          ")
    TFTc.print("          ")
    TFTc.drawChar(100, 200, "*", TFTc.color(4), 0, 1)
    TFTc.drawChar(108, 192, "*", TFTc.color(4), 0, 1)
    TFTc.drawChar(116, 200, "*", TFTc.color(4), 0, 1)
    TFTc.textCursor(0, 210)
    TFTc.setTextSize(1)
    TFTc.setTextColor(11)
    for i in 1..8 do
        TFTc.print("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz_0123456789. !?#")
    end
    TFTc.drawChar(100, 250, "A", TFTc.color(4), 0, 5)
    # 以下はフリーフォントを組み込んでいる場合に実行可能
    # TFTc.textCursor(0, 100)
    # TFTc.setFont(5)
    # TFTc.print("ab")
    # TFTc.setFont(6)
    # TFTc.print("cd")
    # TFTc.setFont(7)
    # TFTc.print("ef")
    # TFTc.setFont(8)
    # TFTc.println("g")
    TFTc.setFont(0)
    TFTc.setTextSize(0)
    System.clrTFTc()
end #if
```

# メソッドの説明(V2ライブラリ)

## TFTcクラス

System.useTFTc() を呼んでおく必要があります。

プログラムを終了時にはSystem.clrTFTc() を呼ぶ必要があります。

### 使用例

```
# Draw Bitmap test (Only 24bit color BMP file)

Usb.println("Checking SD")
if (0 == System.useSD()) then
    Usb.println("SD card is not inserted.")
else
    Usb.println("SD card is ready.")
    if(1==System.useTFTc()) then
        Usb.println("TFT is ready.")
        delay(100)
        Usb.println("Draw 24bit color Bitmap.")
        a=TFTc.drawBmp("purple.bmp", 0, 0)
        Usb.print("drawBmp return code is :"); Usb.println(a.to_s)
        System.clrTFTc()
    else
        Usb.println("TFT is not ready.")
    end #if
    SD.close(0)
end
```

# メソッドの説明(V2ライブラリ)

## TFTcクラス

`System.useTFTc()`を呼んでおく必要があります。

プログラムを終了時には`System.clrTFTc()`を呼ぶ必要があります。

### 使用例

```
# Button test 1 ボタン使用例（簡易）押下あるいはリリースのどちらかを検出する場合
if(1==System.useTFTc()) then
    TFTc.fillRect(TFTc.color(0))
    #button初期化
    TFTc.setButtonMax(5)
    TFTc.setDisplayButtonMax(3)
    TFTc.initButton(1, 50, 100, 50, 50, TFTc.color(9), TFTc.color(0), C_YELLOW, "On", 2)
    TFTc.initButton(2, 120, 100, 50, 50, TFTc.color(9), TFTc.color(0), C_RED, "Off", 2)
    TFTc.initButton(3, 195, 100, 60, 50, TFTc.color(9), TFTc.color(0), C_BLUE, "Exit", 2)
    TFTc.drawButtons()
    TFTc.searchTouchedButton(-1, -1);    # buttonのタッチ履歴をクリア
    while(true) do
        x=(-1); y=(-1); # release トラッキング用に座標をクリアする。
        if (TFTc.panelTouched() == 1) then
            x=TFTc.touchedX;
            y=TFTc.touchedY;
        end #if
        TFTc.searchTouchedButton(x, y)
        bno = TFTc.searchReleasedButton() # リリース時検出の場合
        if(bno!=0) then
            Usb.print("Released Btn no:");Usb.println(bno.to_s)
        end #if
        bno = TFTc.searchPressedButton() # 押下時検出の場合
        if(bno!=0) then
            Usb.print("Pressed Btn no:");Usb.println(bno.to_s)
        end #if
        if (bno==3) then # 押下/リリースにかかわらずモードを抜ける。
            TFTc.deleteButtons() # 生成したボタンのインスタンスを全て消去(ここでは5個分)
            TFTc.fillRect(TFTc.color(0))
            break #exit this mode
        end
    end #while
    System.clrTFTc()
end #if
```

# メソッドの説明(V2ライブラリ)

## TFTcクラス

System.useTFTc() を呼んでおく必要があります。

プログラムを終了時にはSystem.clrTFTc() を呼ぶ必要があります。

### 使用例

```
# Button test 2 ボタン使用例（詳細）：押下とリリースを区別する場合。同時押し検出をする場合。
if(1==System.useTFTc()) then
    TFTc.fillRect(TFTc.color(0))
    TFTc.setButtonMax(5)
    TFTc.setDisplayButtonMax(3)
    TFTc.initButton(1, 50, 100, 50, 50, TFTc.color(9), TFTc.color(0), TFTc.color(4), "On", 2)
    TFTc.initButton(2, 120, 100, 50, 50, TFTc.color(9), TFTc.color(0), TFTc.color(2), "Off", 2)
    TFTc.initButton(3, 195, 100, 60, 50, TFTc.color(9), TFTc.color(0), TFTc.color(6), "Exit", 2)
    TFTc.drawButtons()
    TFTc.searchTouchedButton(-1, -1);    # buttonのタッチ履歴をクリア
    exit_flag=0;
    while(true) do
        x=(-1); y=(-1); # release トラッキング用に座標をクリアする
        if (TFTc.panelTouched() == 1) then
            x=TFTc.touchedX; y=TFTc.touchedY;
        end #if
        TFTc.searchTouchedButton(x, y)
        for i in 1..3 do
            ret = TFTc.searchButtonState(i)
            case ret
            when -1 then
                Usb.print("Pressed Btn no:");Usb.println(i.to_s)
            when 1 then
                Usb.print("Released Btn no:");Usb.println(i.to_s)
                if (3==i) then
                    exit_flag=1
                end
            end #case
        end #for
        if exit_flag == 1 then
            TFTc.deleteButtons()    # 生成したボタンのインスタンスを全て消去(ここでは5個分)
            TFTc.fillRect(TFTc.color(0))
            break    #exit this mode
        end #if
    end #while
    System.clrTFTc()
end #if
```

# メソッドの説明(V2ライブラリ)

## TFTcクラス

`System.useTFTc()`を呼んでおく必要があります。

プログラムを終了時には`System.clearTFTc()`を呼ぶ必要があります。

### 使用例

```
# Draw Screen test Rubyにてボタン生成、タッチ検出、グラフィック描画を制御
if(1==System.useTFTc()) then
    TFTc.fillRect(TFTc.color(0))
    w=TFTc.width() #ディスプレイの幅を取得
    h=TFTc.height() #ディスプレイの高さを取得
    #button初期化
    TFTc.setButtonMax(10) #ボタンのインスタンスを生成
    TFTc.setDisplayButtonMax(8) # この画面で使用するボタン数を指定
    TFTc.initButton(1, 20, 20, 40, 40, TFTc.color(9), TFTc.color(0), TFTc.color(2), "RED", 1)
    TFTc.initButton(2, 60, 20, 40, 40, TFTc.color(9), TFTc.color(0), TFTc.color(4), "YELLOW", 1)
    TFTc.initButton(3, 100, 20, 40, 40, TFTc.color(9), TFTc.color(0), TFTc.color(5), "GREEN", 1)
    TFTc.initButton(4, 140, 20, 40, 40, TFTc.color(9), TFTc.color(0), TFTc.color(10), "CYAN", 1)
    TFTc.initButton(5, 180, 20, 40, 40, TFTc.color(9), TFTc.color(0), TFTc.color(6), "BLUE", 1)
    TFTc.initButton(6, 220, 20, 40, 40, TFTc.color(9), TFTc.color(0), TFTc.color(11), "MAGEN", 1)
    TFTc.initButton(7, 220, 60, 40, 40, TFTc.color(9), TFTc.color(0), TFTc.color(1), "EXIT", 1)
    TFTc.initButton(8, 20, 60, 40, 40, TFTc.color(9), TFTc.color(0), TFTc.color(7), "CLEAR", 1)
    TFTc.drawButtons()
    TFTc.searchTouchedButton(-1, -1); # buttonのタッチ履歴をクリア
    currentcolor = 2 # RED
    old_bno = 1 # ハイライトのための処理:前回タッチしてハイライトされているボタンno
    bno = 1 # ハイライトのための処理:現在のタッチ検出ボタンno
    TFTc.drawButton(1, 1); # button 1 を反転させる
    exit_flag = 0
    while(true) do
        x=(-1); y=(-1); # releaseトラッキング用に座標をクリアする。
        if (TFTc.panelTouched() == 1) then # パネル全体のタッチ検出
            x=TFTc.touchedX;
            y=TFTc.touchedY;
        end #if
        if(x<w or y<h) then
            TFTc.searchTouchedButton(x, y) # 全てのボタンについてタッチされたか調査
            bno = TFTc.searchReleasedButton() # リリースされたボタンを検出
            if(bno!=0) then # 何れかのボタンがタッチされていた場合
                Usb.print("Released Btn no:");Usb.println(bno.to_s)
                if(bno!=8) then #no8のボタンは画面クリア用なので、反転させる必要なし
                    TFTc.drawButton(old_bno, 0) # 前回タッチされたボタンの反転表示をクリア
                    TFTc.drawButton(bno, 1) # 今回タッチされたボタンを反転表示する
                end #if
            end #if
        end #if
    end #while
end #if
```

# メソッドの説明(V2ライブラリ)

## TFTcクラス

`System.useTFTc()`を呼んでおく必要があります。

プログラムを終了時には`System.clrTFTc()`を呼ぶ必要があります。

### 使用例

```

        old_bno = bno          # 「前回タッチされたボタンno」を記憶する
    end #if
end #if
case bno
when 1 then
    currentcolor = 2;
when 2 then
    currentcolor = 4;
when 3 then
    currentcolor = 5;
when 4 then
    currentcolor = 10;
when 5 then
    currentcolor = 6;
when 6 then
    currentcolor = 11;
when 7 then # EXIT ボタンでこのモードを終了する
    TFTc.deleteButtons()  # 生成したボタンのインスタンスを全て消去(ここでは10個分)
    TFTc.fillRect(TFTc.color(0)) # 画面を暗転する
    exit_flag=1
when 8 then # 画面をクリアする
    TFTc.fillRect(TFTc.color(0))
    TFTc.drawButtons()
    TFTc.drawButton(old_bno, 1)    # 前回タッチされたボタンを反転表示する
when 0 then
    if x!=(-1) then #画面がタッチされており、かつボタンがタッチされていない場合
        TFTc.fillCircle(x, y, 3, TFTc.color(currentcolor))
    end # then
end #when
end #if
if exit_flag==1 then
    break  #exit while(true)
end #if
end #while
System.clrTFTc()
end #if

```