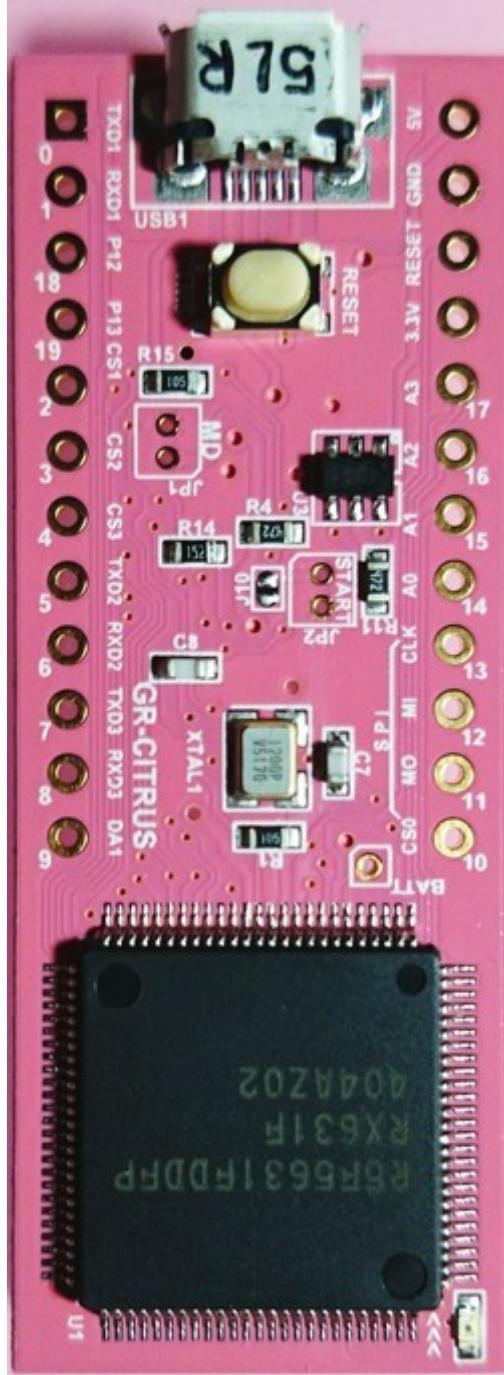


GR-CITRUS 搭載
Ruby ファーム
説明資料 ver1.5

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)

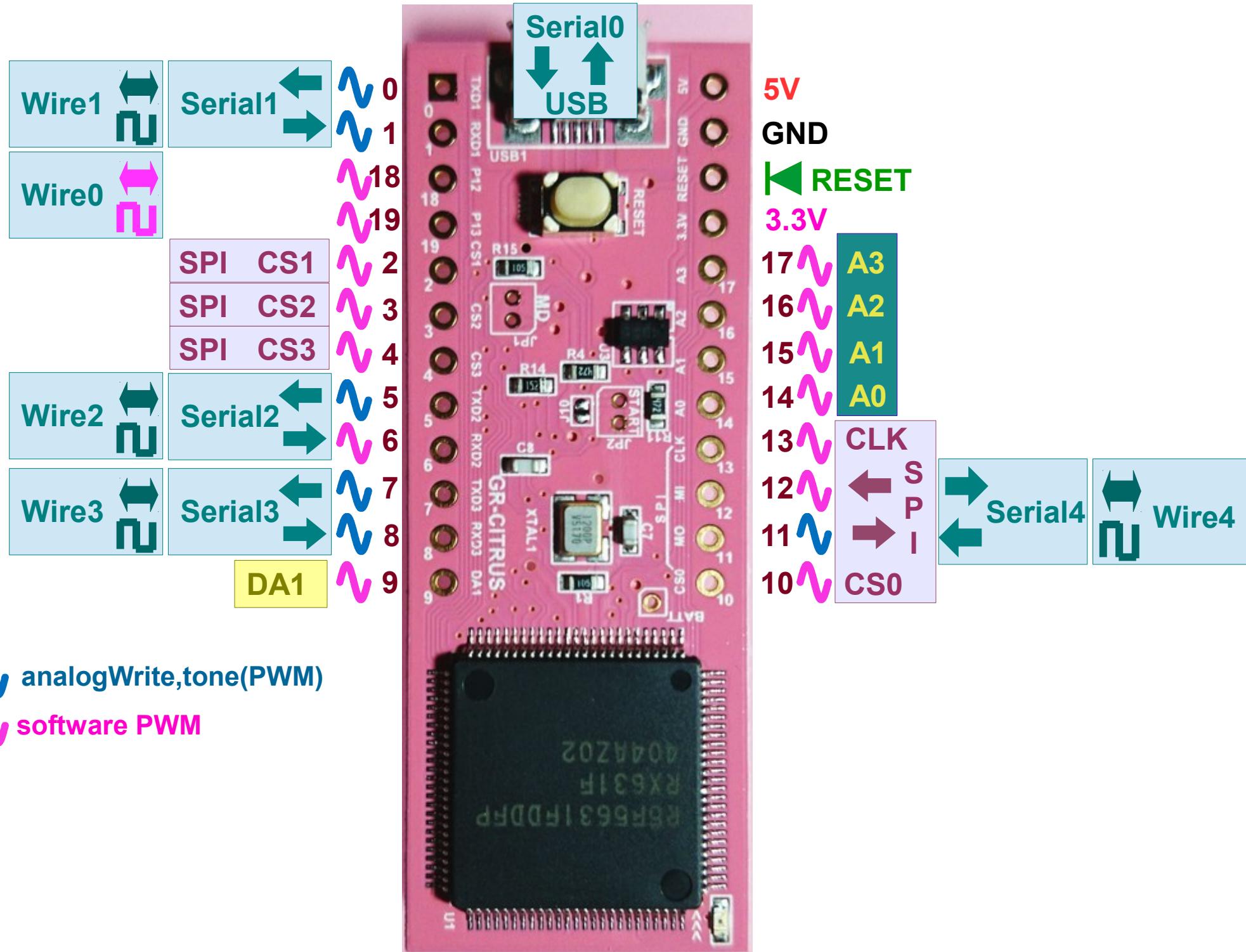
PWM、Servoは自由割当てです。

電 源

5V (USBバスパワード)

サ イ ズ

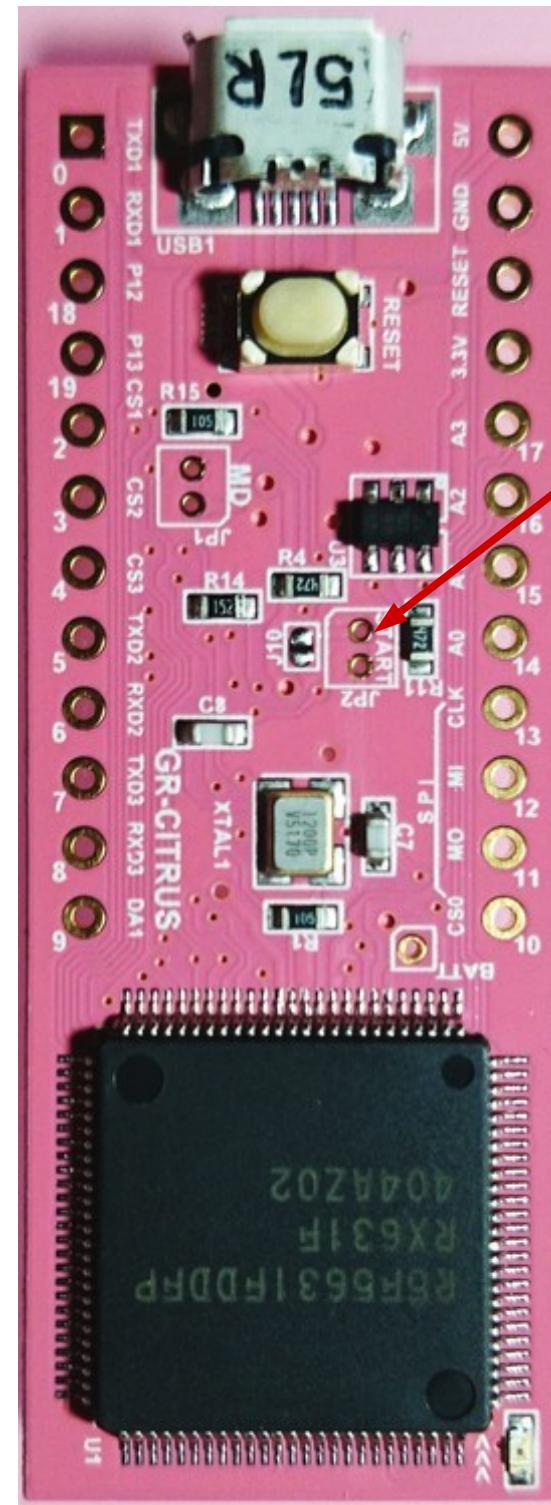
52 × 20mm



RX631 ピン番号

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

赤文字ピン番号は
5Vトレント



17 P43 PE1
16 P42 PB5
15 P41 PB3
14 P40 P27
13 PC5
12 PC7
11 PC6
10 PC4

5V

GND

RESET

3.3V

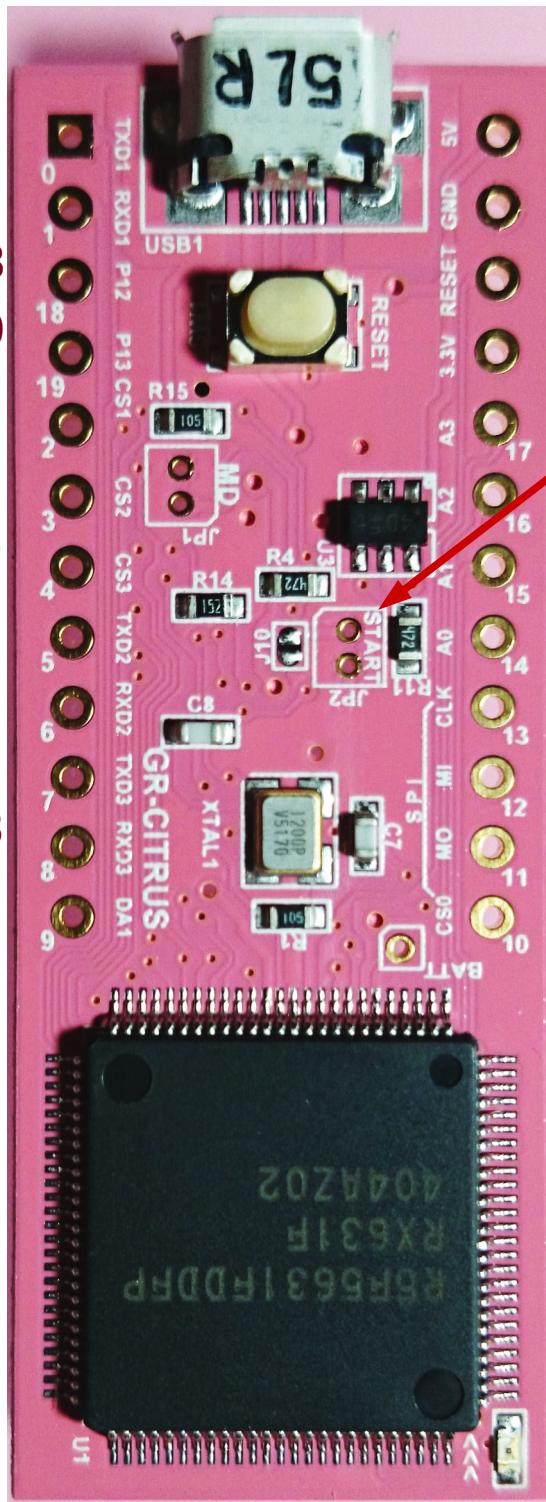
RX631 ピン番号

P35 NMI

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	

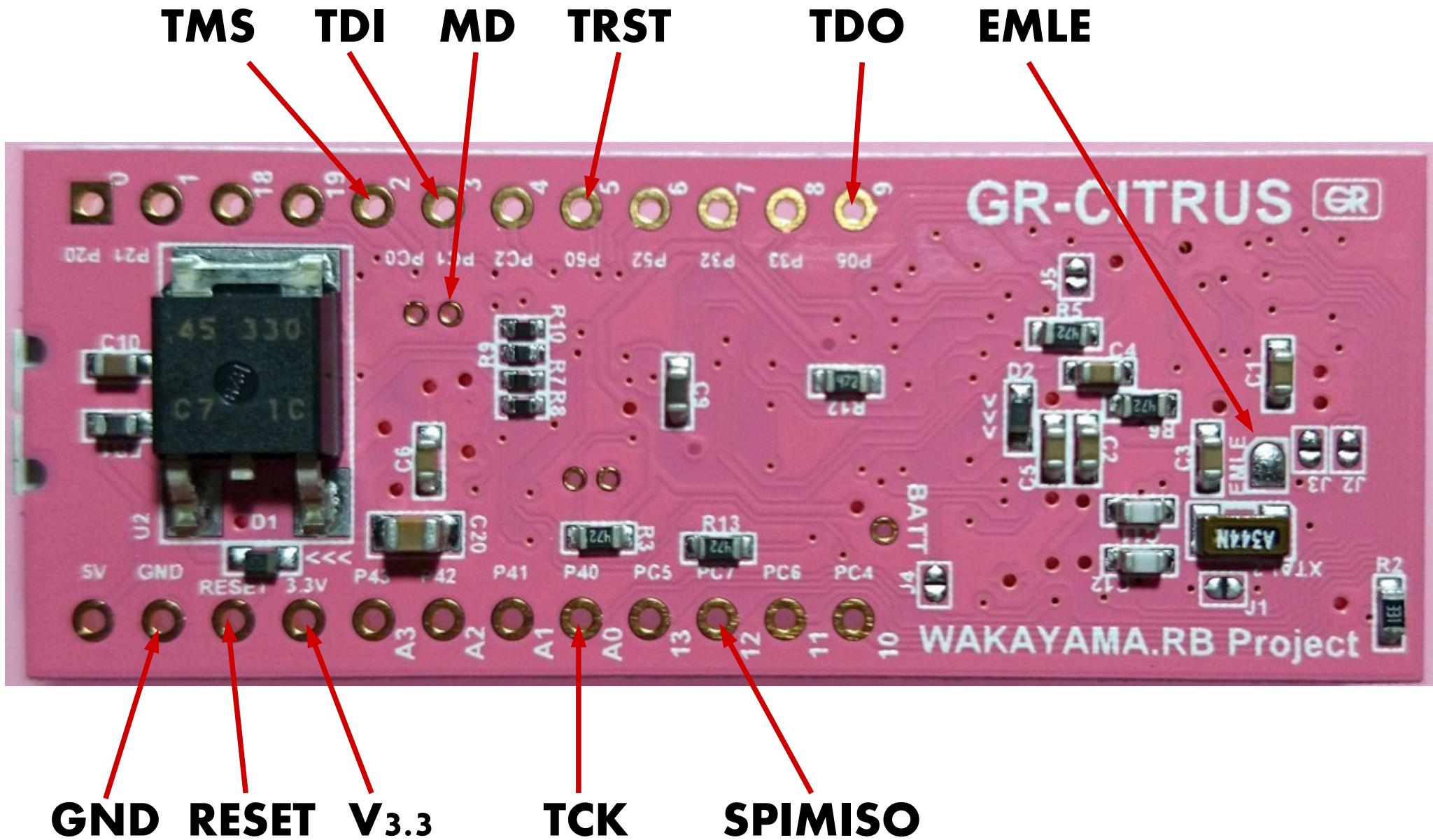


5V	
GND	
RESET	
3.3V	
P35	17
54	16
P43	15
PE1	14
P42	13
PB5	12
P41	11
PB3	10
P40	17
P27	16
PC5	15
14	14
TCK	13
12	12
PC7	11
11	11
PC6	10
10	10

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

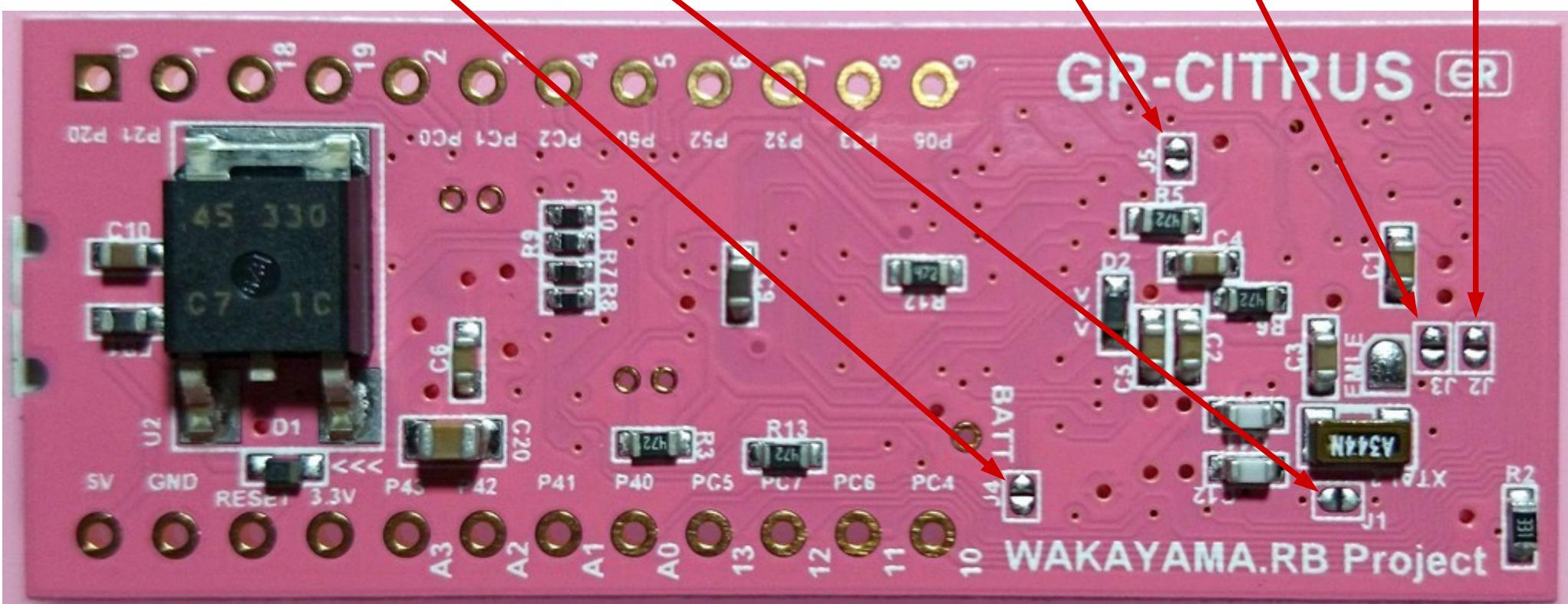
PE1と17番を接続します。
P43ともショートになります。

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。

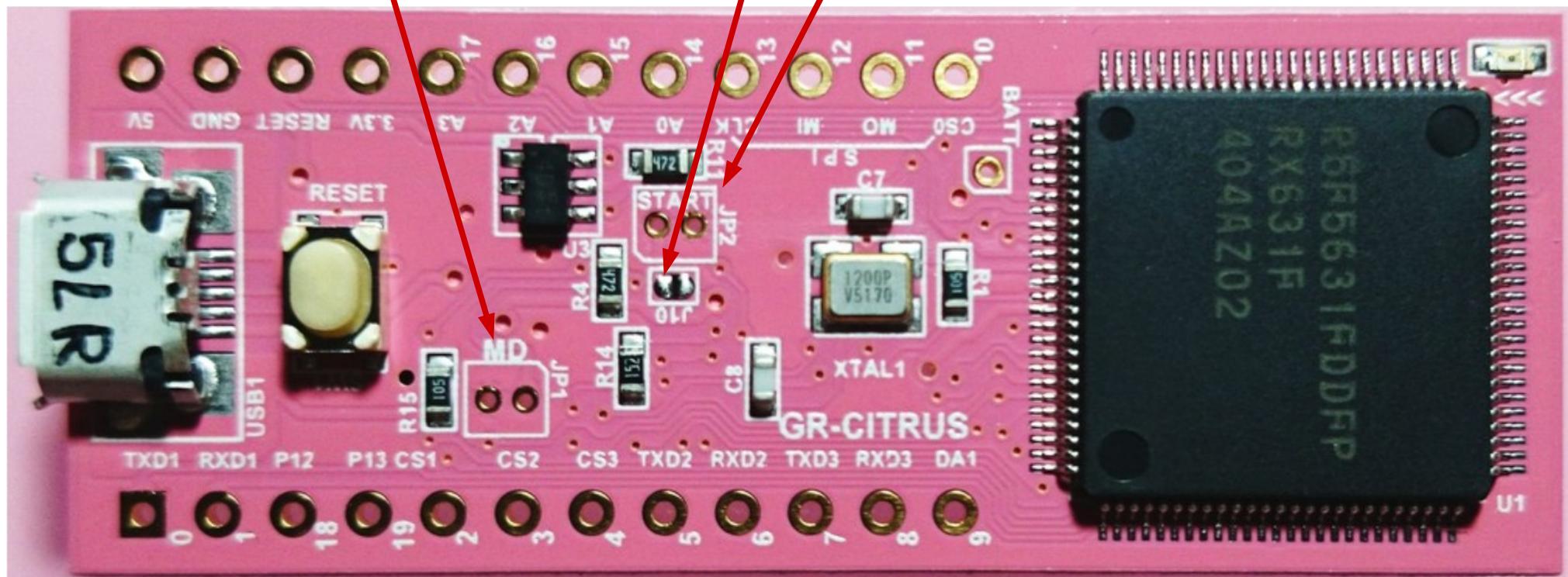


ジャンパの説明

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 (V2 ライブライアリ)

9

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)

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.getMrbPath()

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()

基本ソフト仕様(V2ライブラリ)

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.detach(ch)
```

リアルタイムクロッククラス

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

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)
```

基本ソフト仕様(V2ライブラリ)

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

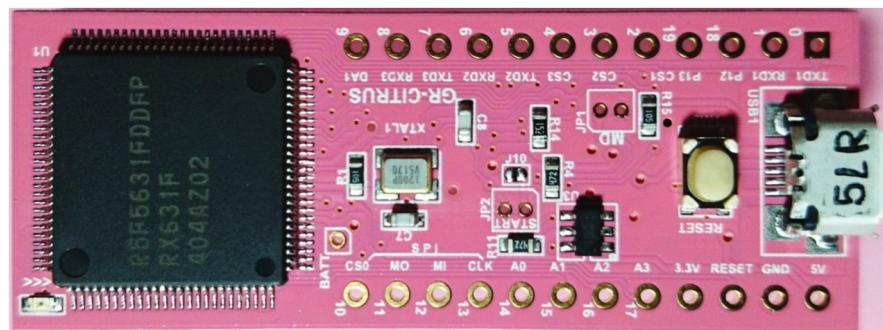
Rubyファームは、内部にrubyプログラムを保存できます。ファイル形式はmrbcによりコンパイルしたmrb形式のファイルとなります。

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

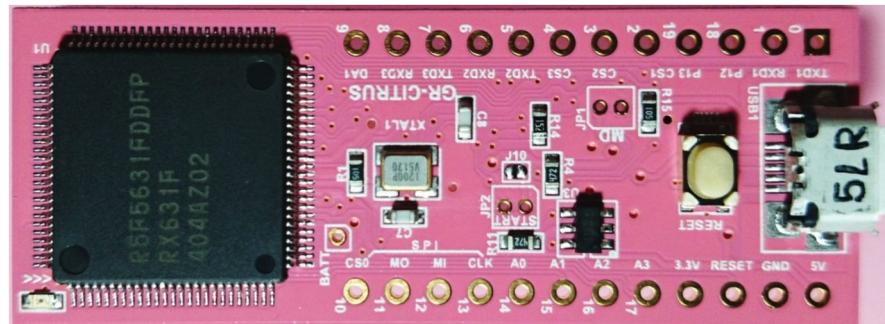


TeraTerm

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。



The screenshot shows the Rubic software interface. The code editor window displays the following Ruby script:

```
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 |
```

The terminal window below shows the output of the program execution:

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

[Finish main.mrb]
```

At the bottom of the interface, there are buttons for 'Wakayama.rb Board' and 'COM3', and an 'Info' button.

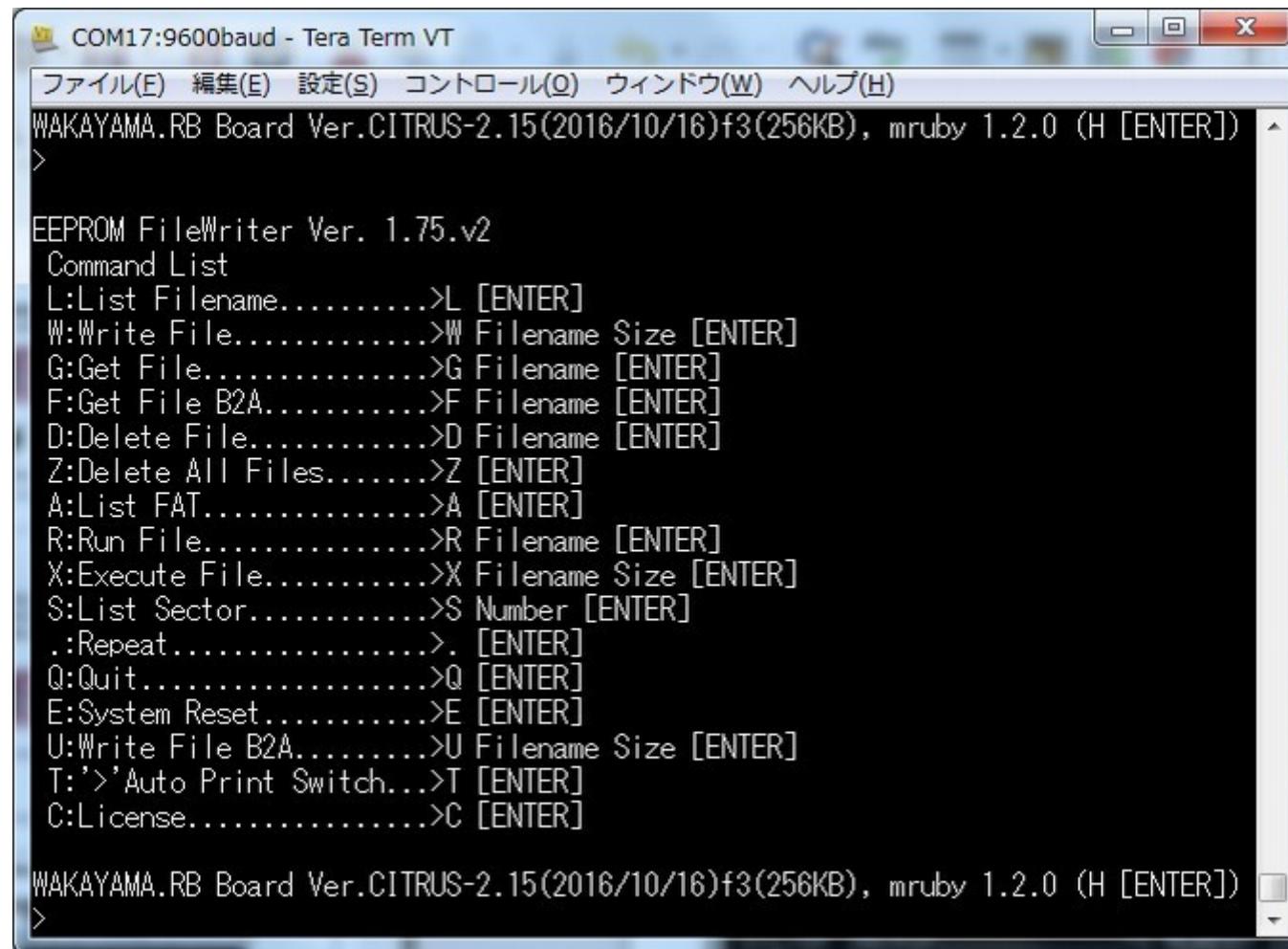
Rubic

<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

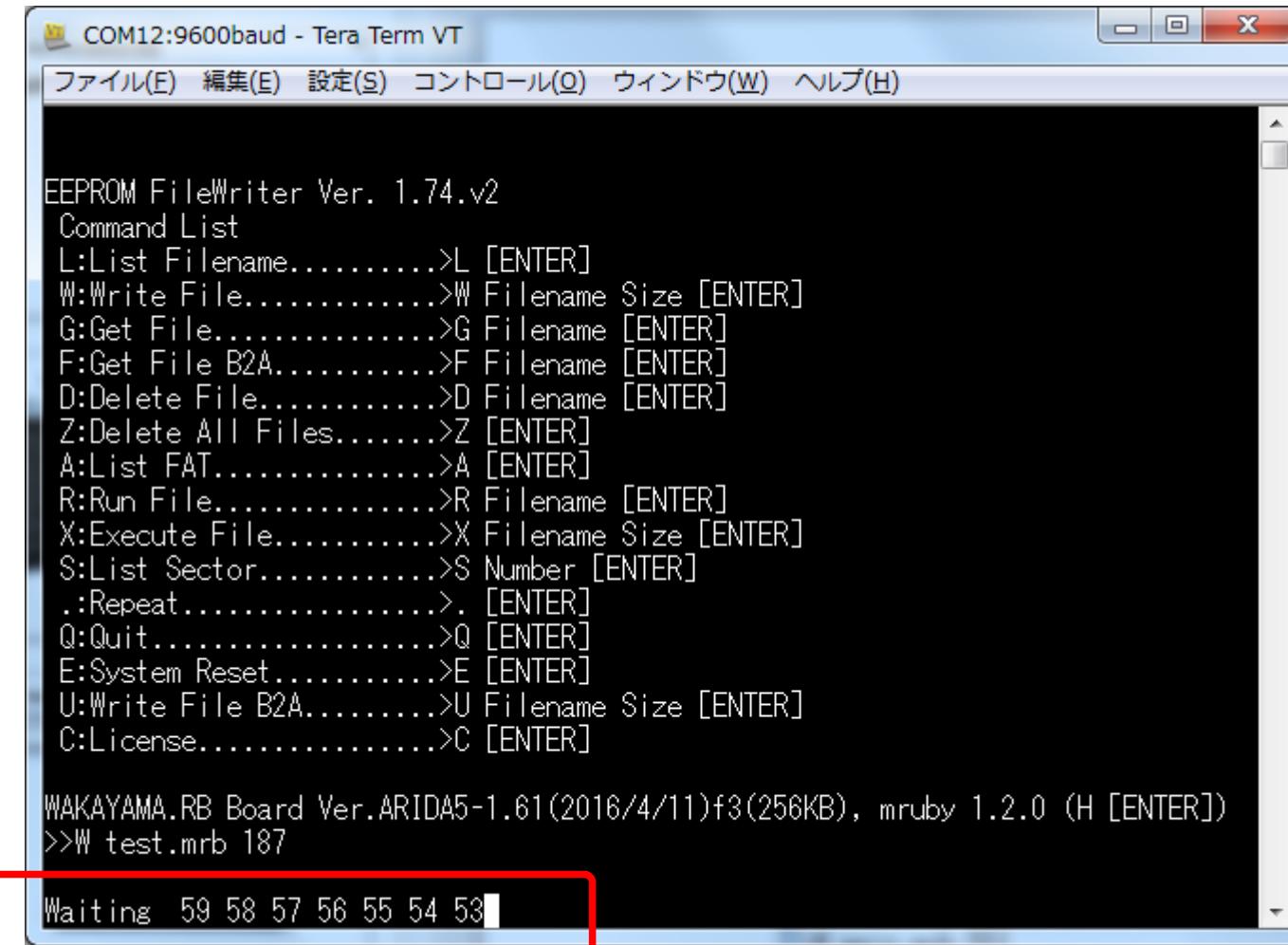
The screenshot shows a terminal window titled "COM12:9600baud - Tera Term VT". The window contains a command list for EEPROM FileWriter Ver. 1.74.v2. At the bottom, there is an input field containing the command "W test.mrb 187" which is highlighted with a red rectangle.

```
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
```

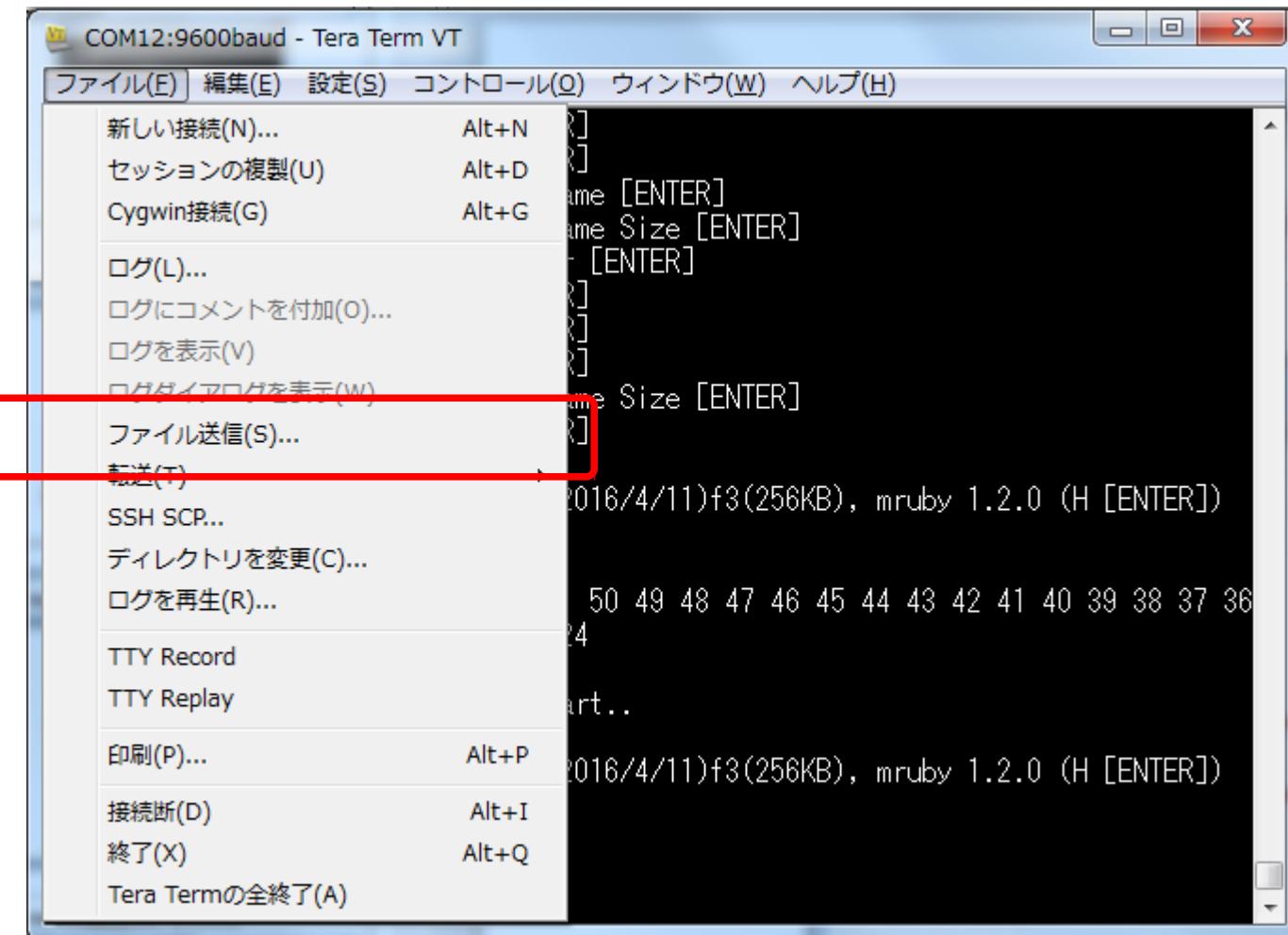
プログラムを書き込みます。(W コマンド)

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)

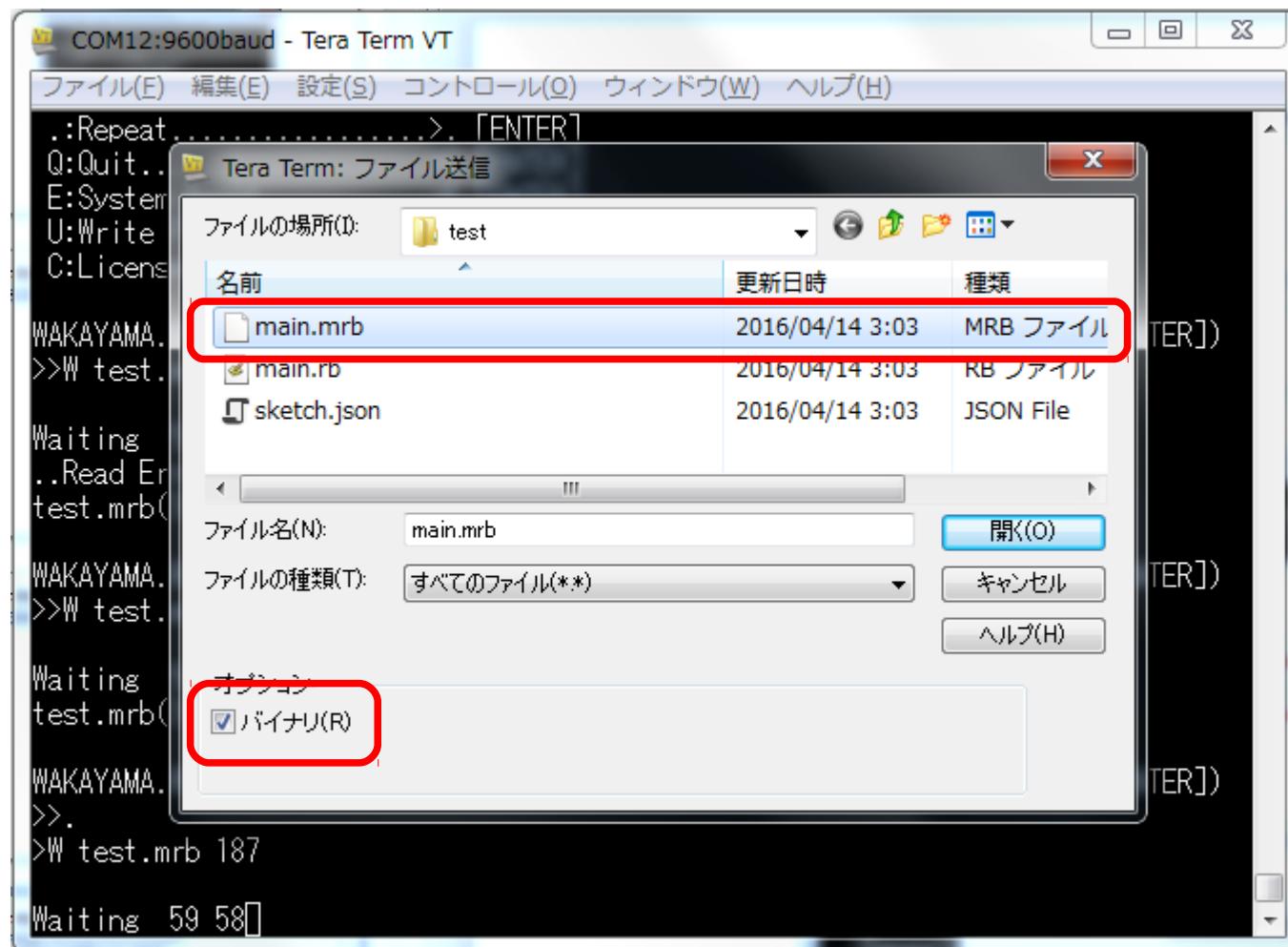
Tera Termの場合、ファイル→ファイル送信 を選択します。



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 a command menu and the board's version information:

```
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

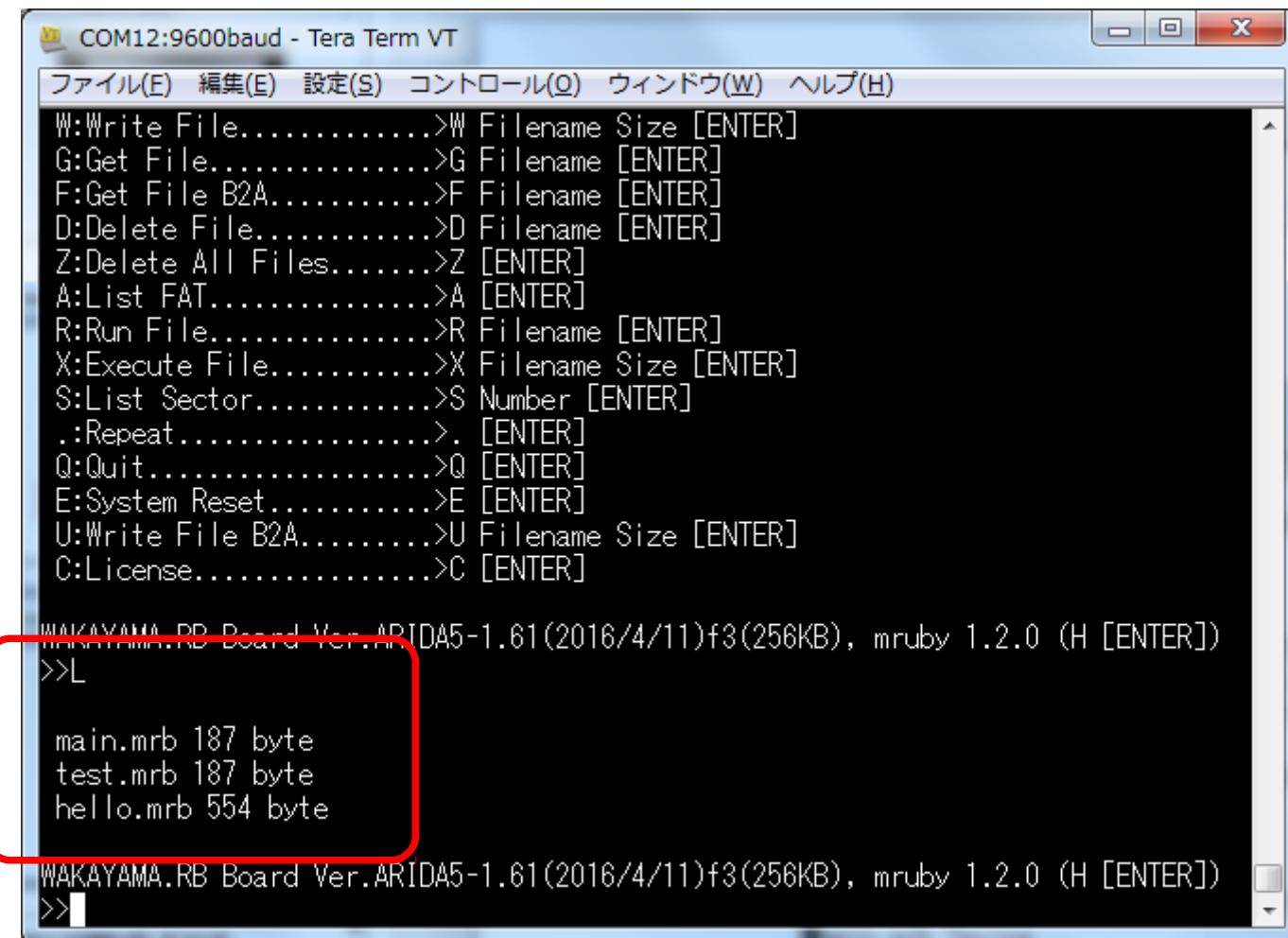
Waiting 59 58 57 56 55 54 53 52 51
test.mrb(187) Saving..

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

A red rectangle highlights the last two lines of output, which show the progress of saving the file to the board.

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.

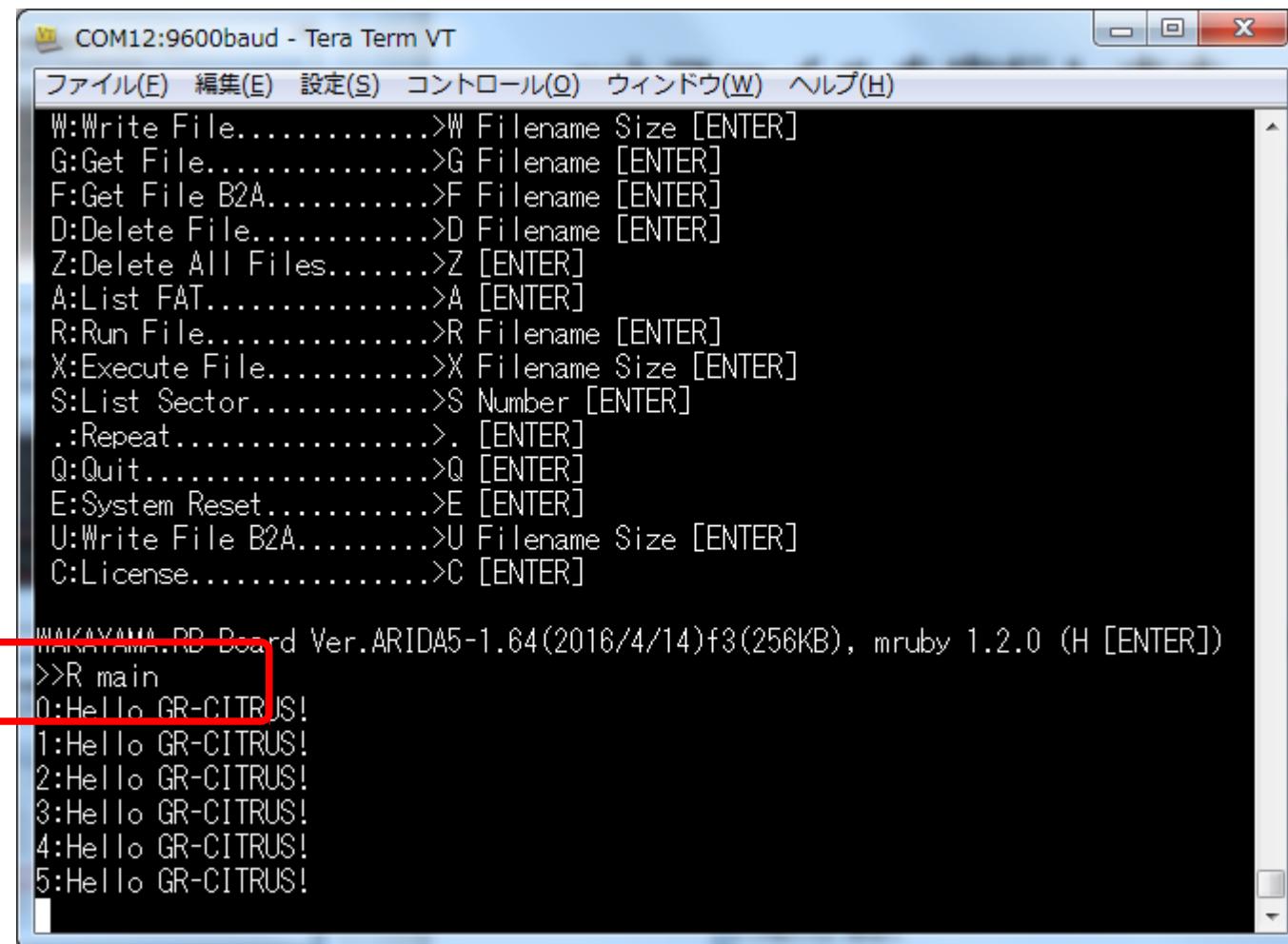
mrbファイルを実行します。(R コマンド)

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 list of commands:

- 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]

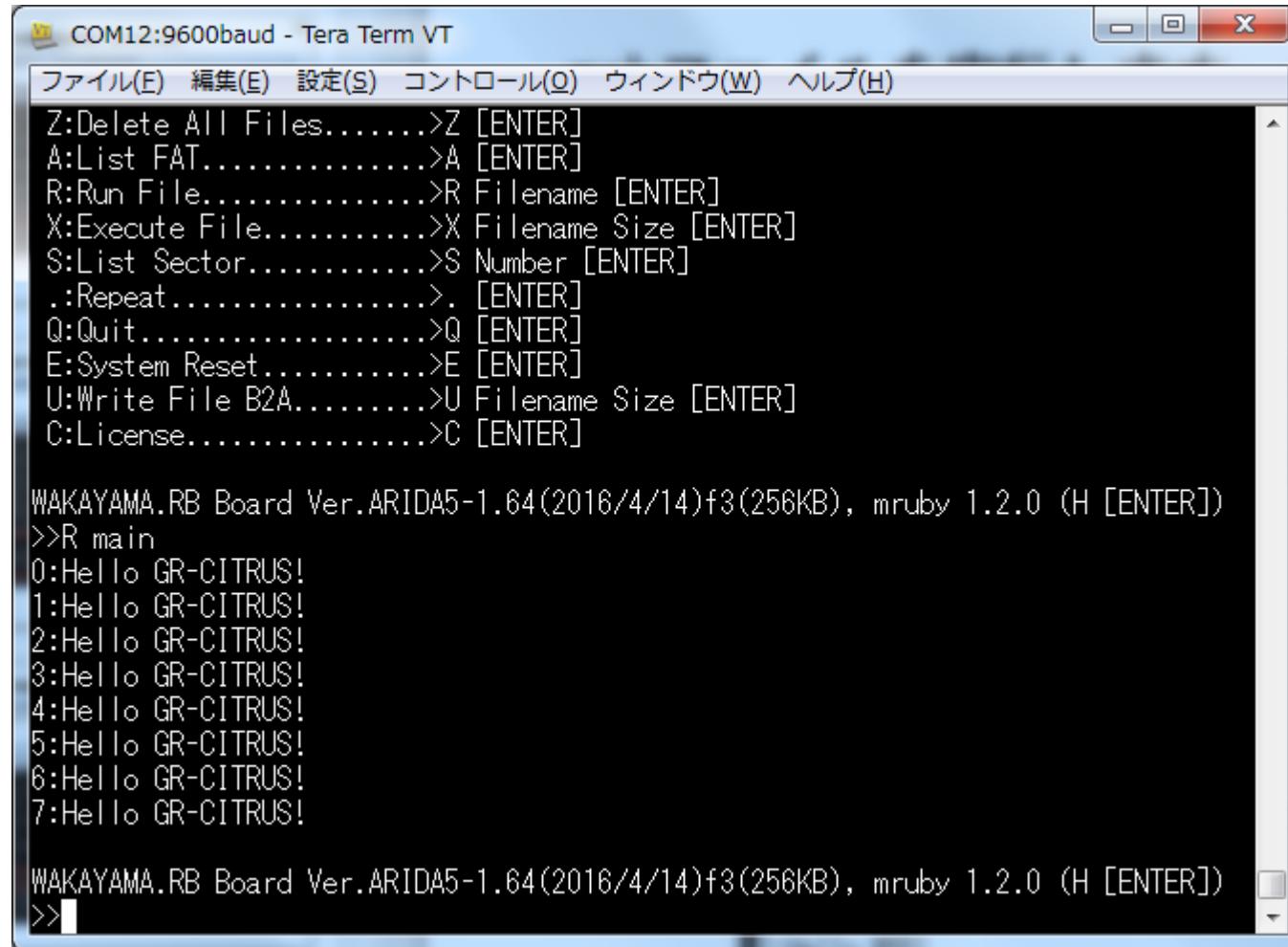
The command history at the bottom of the window shows:

```
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 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)". Below the menu is 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]

The main window displays the output of the program. It starts with the board information: "WAKAYAMA.RB Board Ver.ARIDA5-1.64(2016/4/14)f3(256KB), mruby 1.2.0 (H [ENTER])". Then it shows the output of the "R main" command, which prints "Hello GR-CITRUS!" seven times. Finally, the "X hello.mrb 187" command is entered at the bottom, with the input area highlighted by a red box.

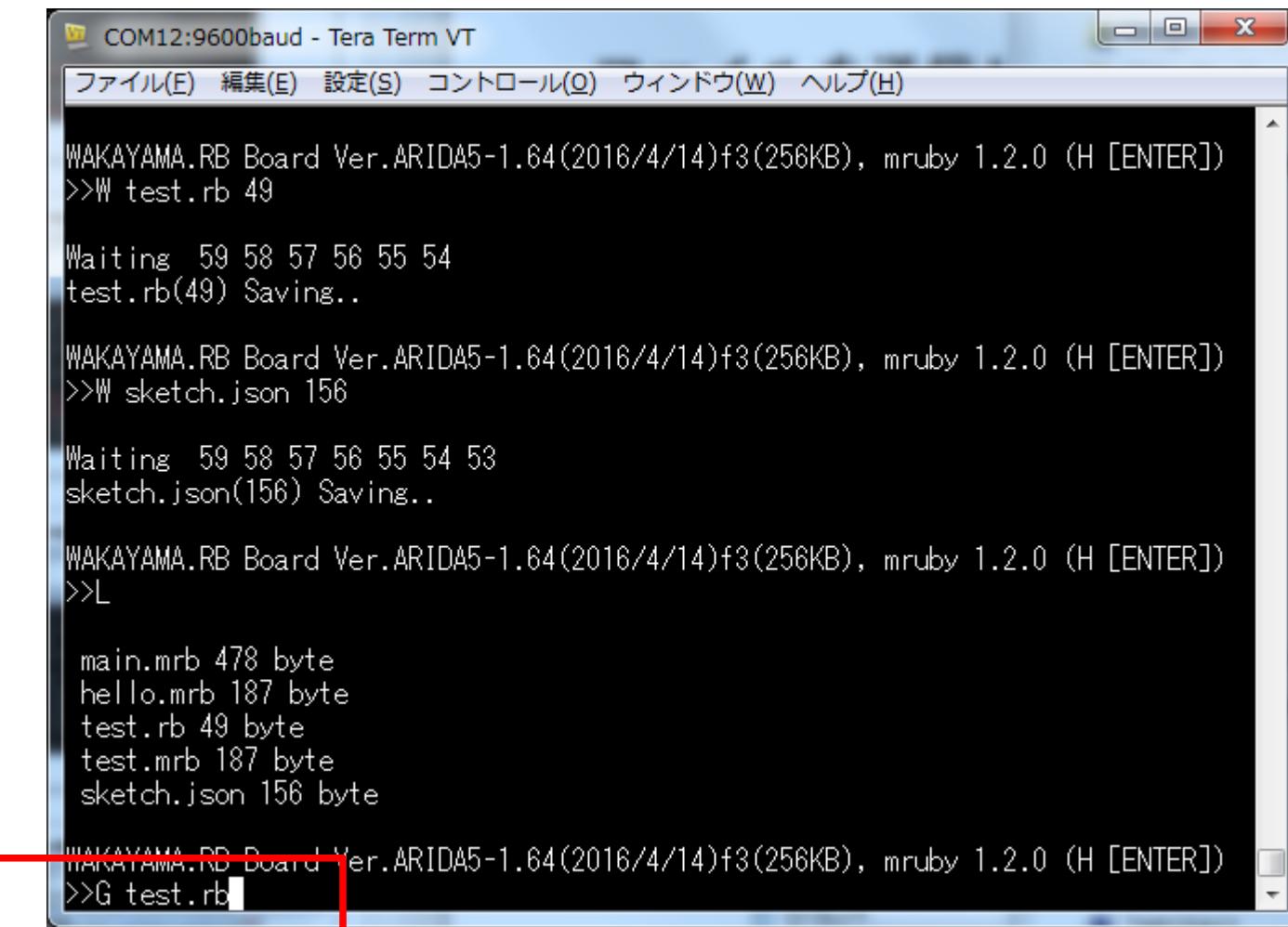
```
WAKAYAMA.RB Board Ver.ARIDA5-1.64(2016/4/14)f3(256KB), mruby 1.2.0 (H [ENTER])
>>R main
0:Hello GR-CITRUS!
1:Hello GR-CITRUS!
2:Hello GR-CITRUS!
3:Hello GR-CITRUS!
4:Hello GR-CITRUS!
5:Hello GR-CITRUS!
6:Hello GR-CITRUS!
7:Hello GR-CITRUS!
WAKAYAMA.RB Board Ver.ARIDA5-1.64(2016/4/14)f3(256KB), mruby 1.2.0 (H [ENTER])
>>X hello.mrb 187
```

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

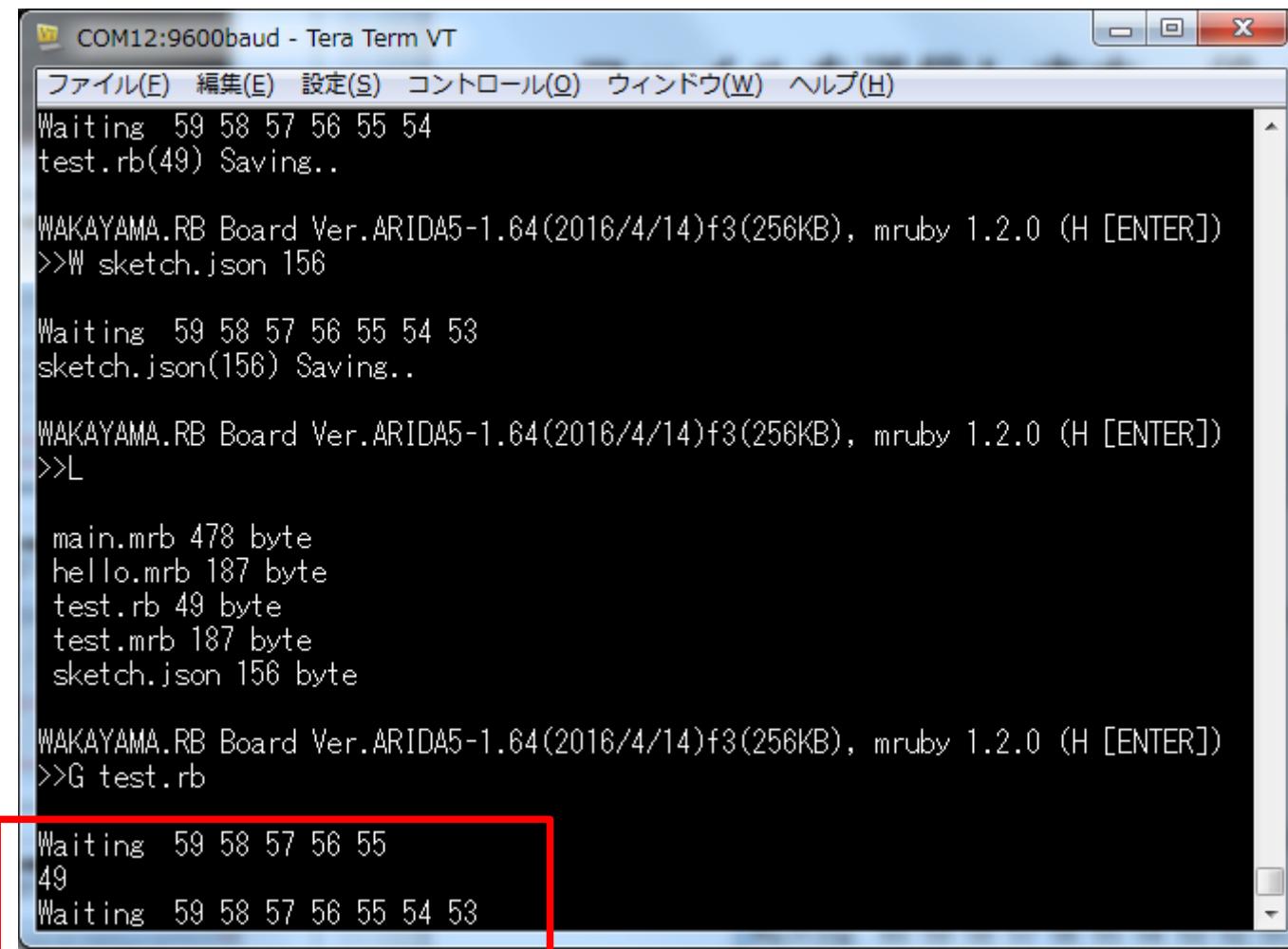
main.mrb 478 byte
hello.mrb 187 byte
test.rb 49 byte
test.mrb 187 byte
sketch.json 156 byte

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.

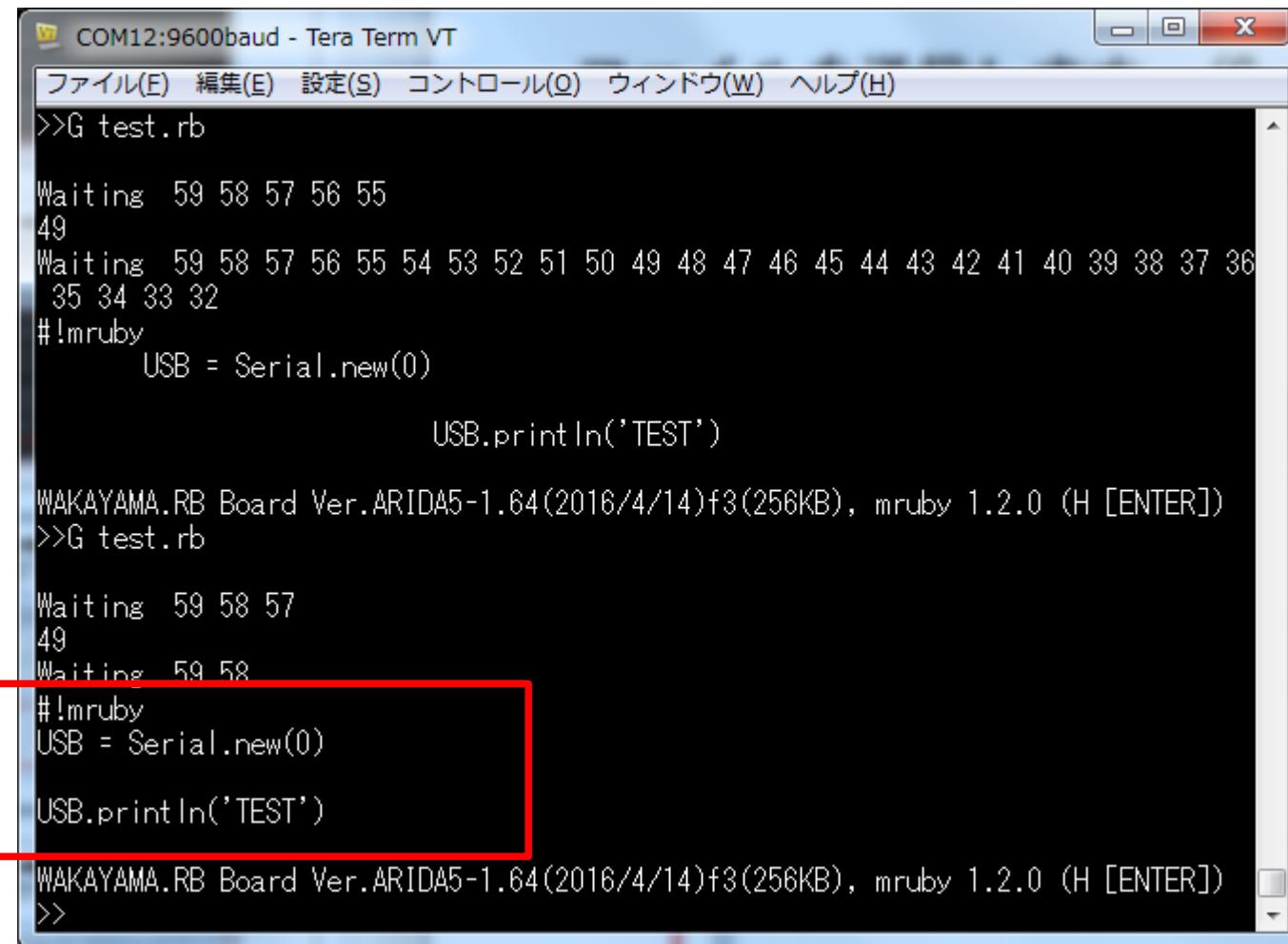
ファイルを送信します。(G コマンド)

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。



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 the following text:

```
>>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])
>>
```

A red rectangular box highlights the first transmission of the file content, specifically the line starting with "#!mruby".

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。

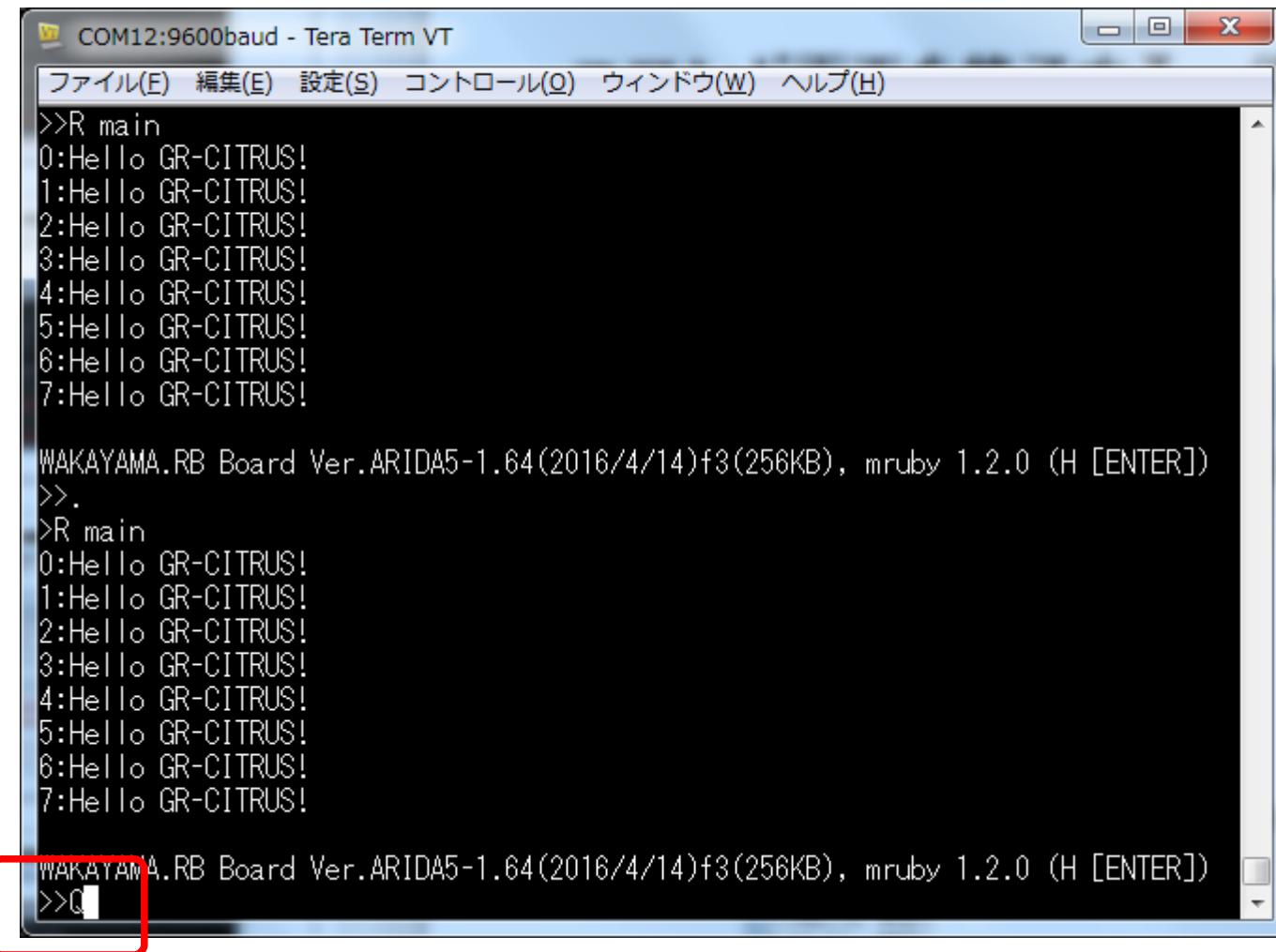
The screenshot shows a terminal window titled "COM12:9600baud - Tera Term VT". The window contains the following text:

```
>>.  
>>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 the response to the ".R main" command, and the second line is the prompt for the next command.

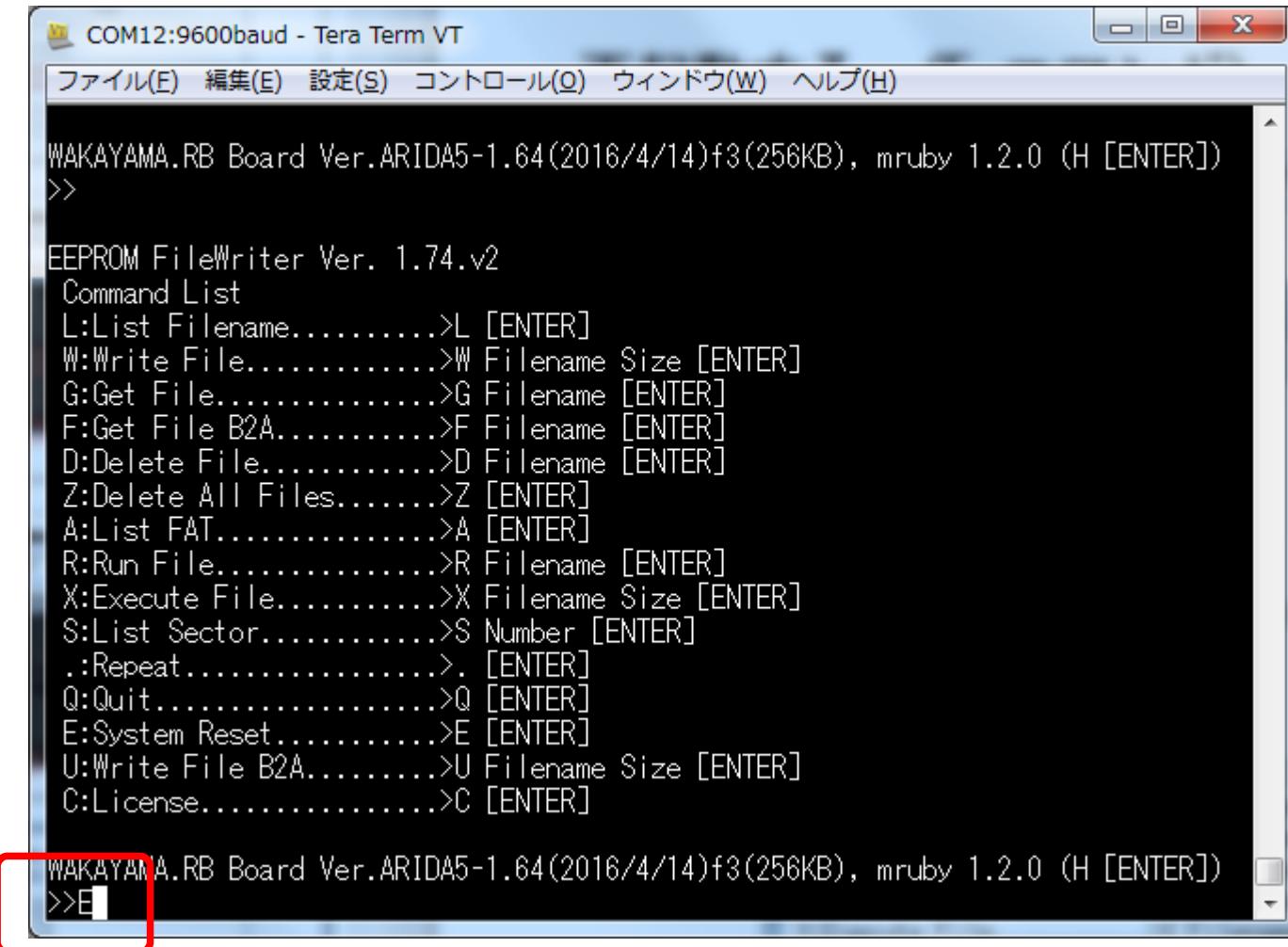
コマンド画面を終了する。(Q コマンド)

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。



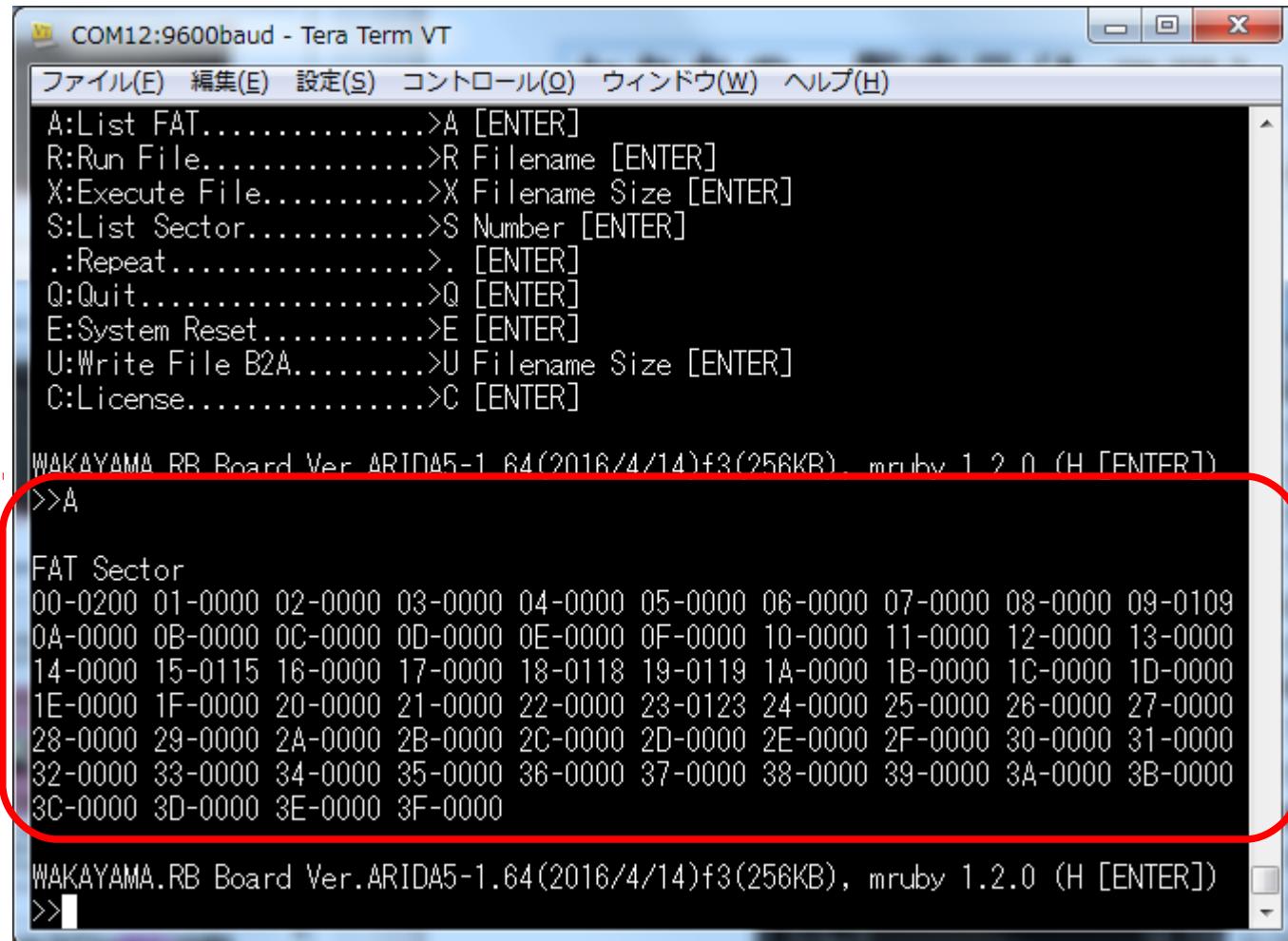
再起動する。(E コマンド)

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。



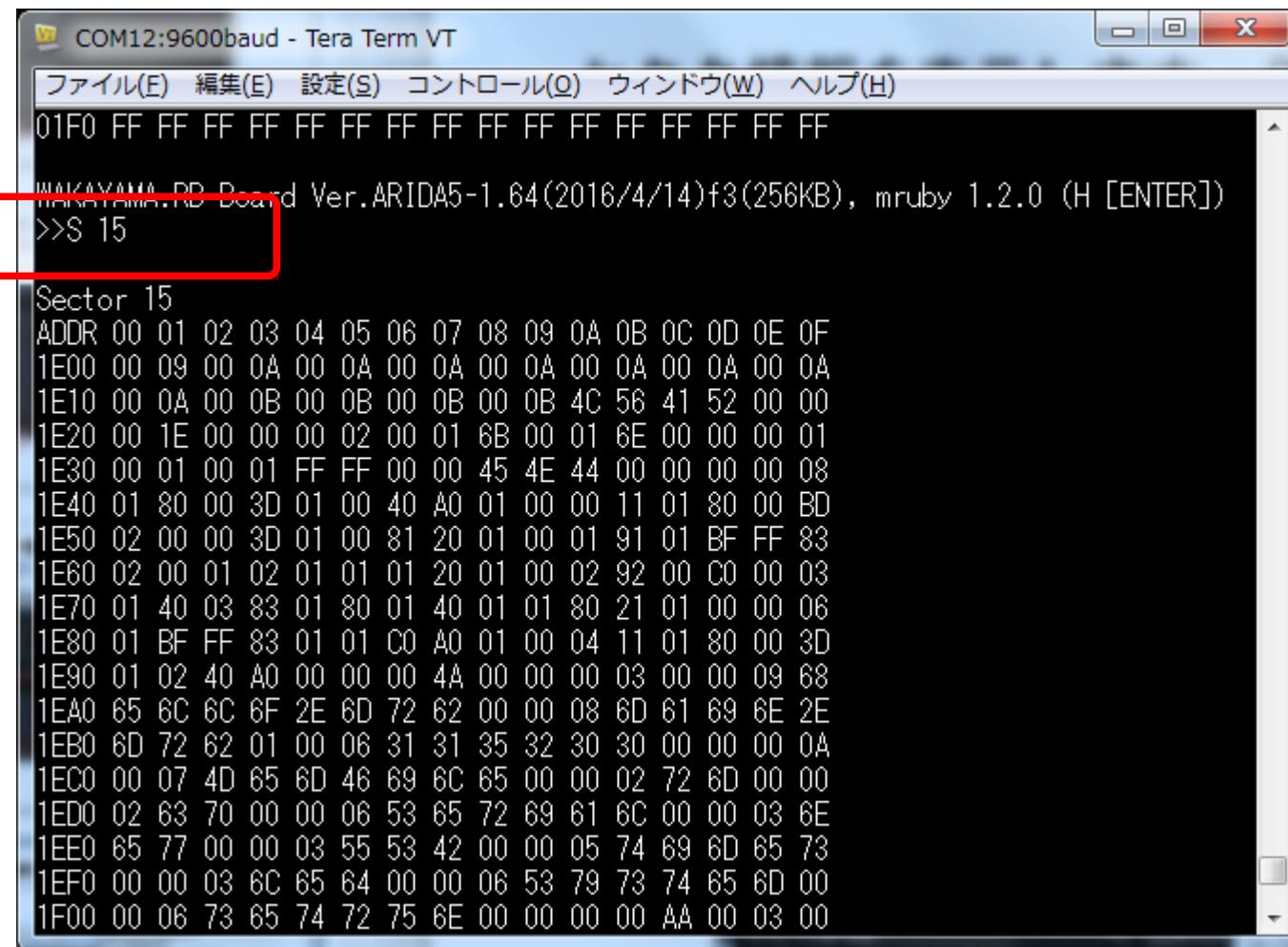
セクタ情報を表示します。(S コマンド)

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



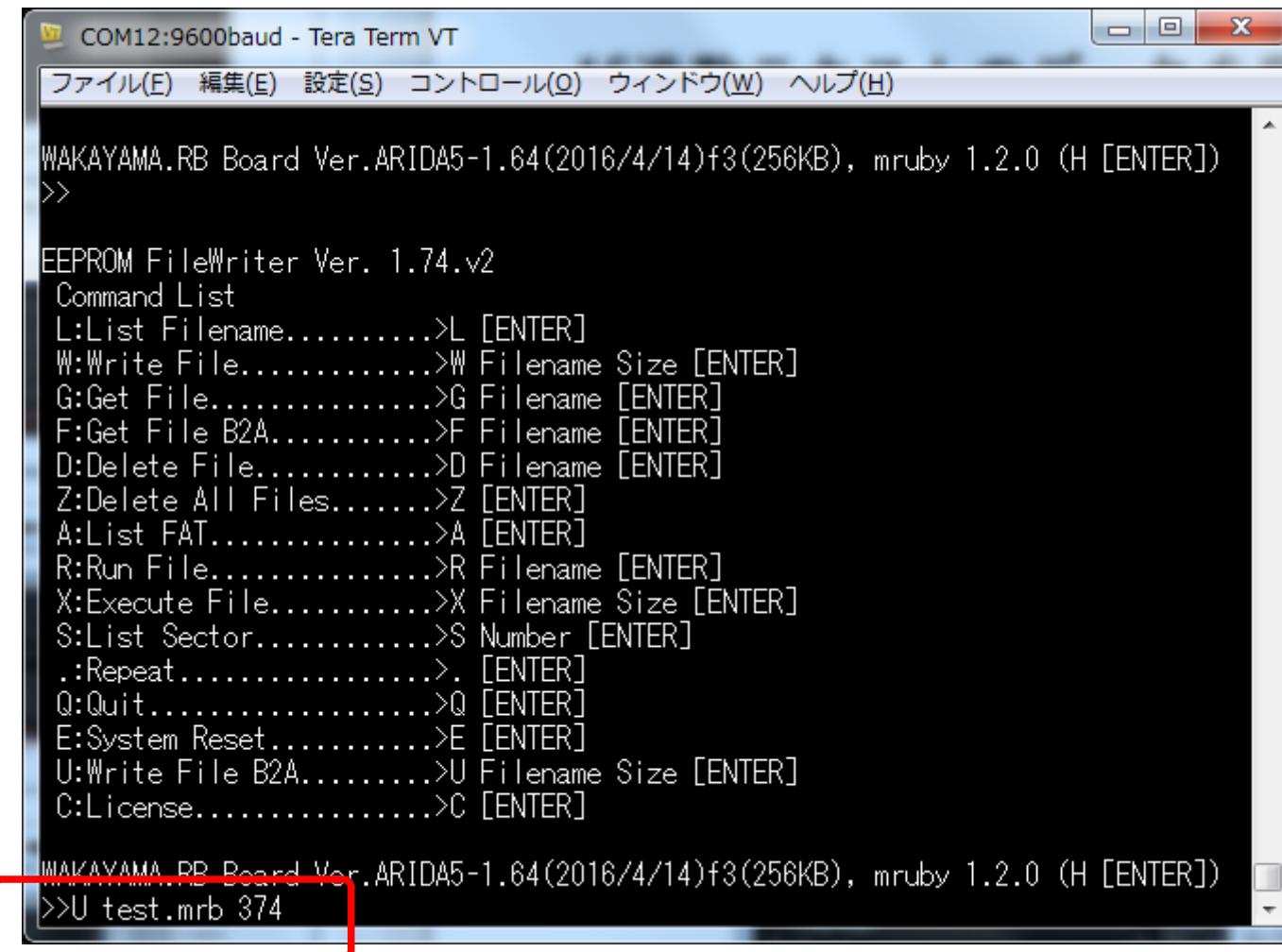
16進数テキストのデータを受信します。(U コマンド)

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
```

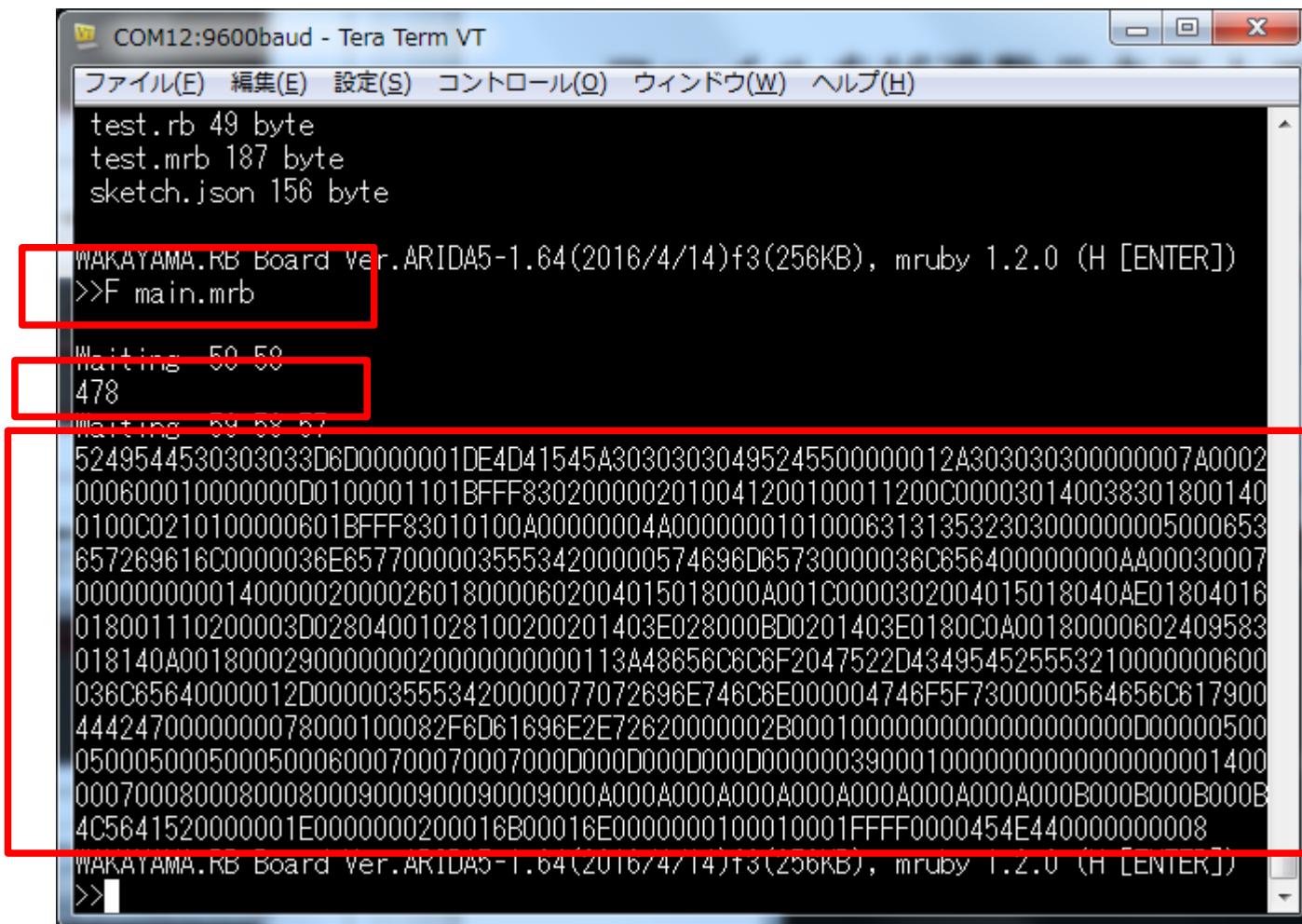
ファイルを16進数テキストで送信します。(F コマンド)

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:

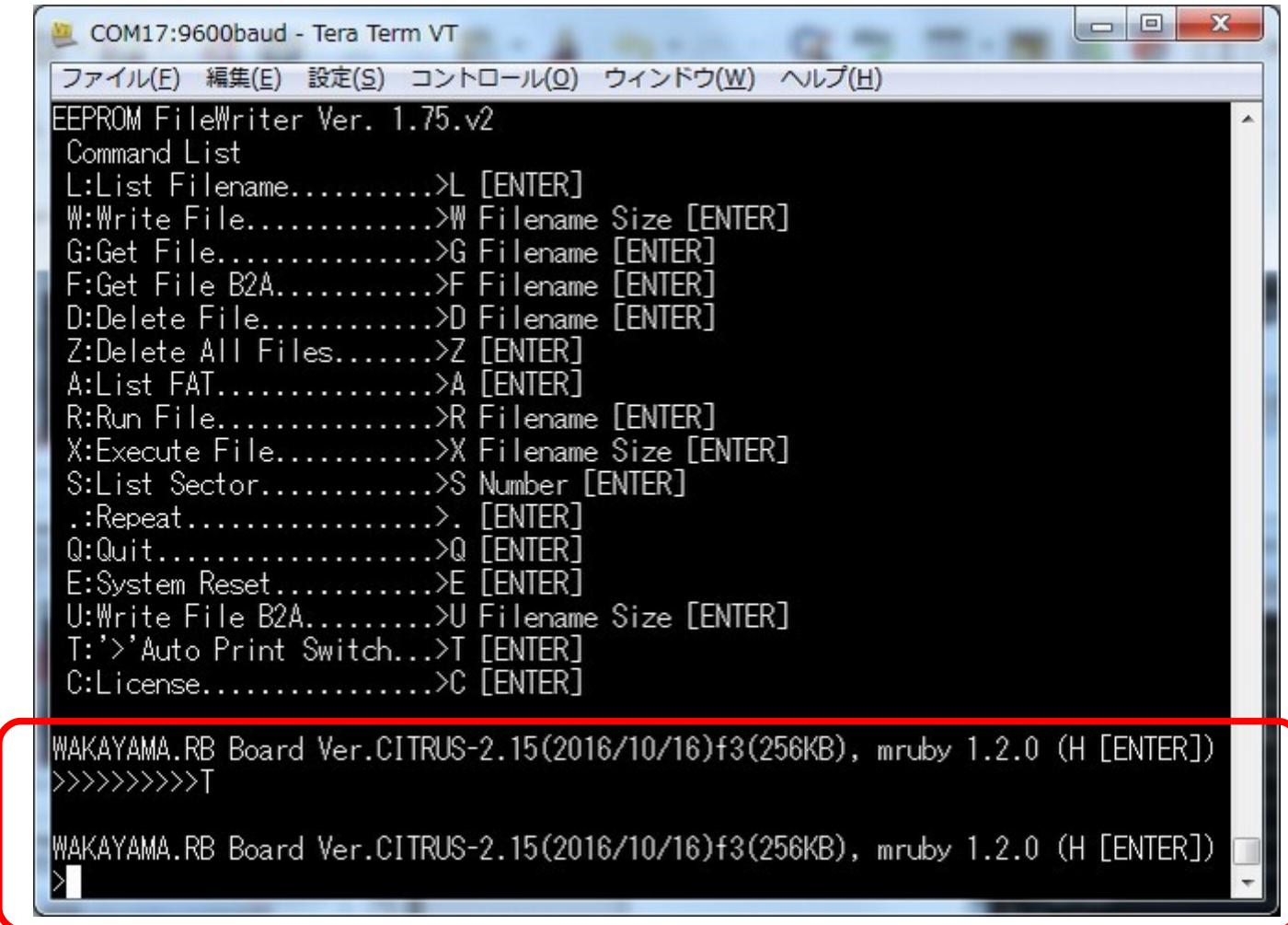
- S>List Sector.....>S Number [ENTER]
- .:Repeat.....>. [ENTER]
- Q:Quit.....>Q [ENTER]
- E:System Reset.....>E [ENTER]
- U:Write File B2A.....>U Filename Size [ENTER]
- C:License.....>C [ENTER]

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 execution, 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。



COM17:9600baud - Tera Term VT

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 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 command menu:

```
G:Get File.....>G F ilename [ENTER]
F:Get File B2A.....>F F ilename [ENTER]
D:Delete File.....>D F ilename [ENTER]
Z:Delete All Files.....>Z [ENTER]
A>List FAT.....>A [ENTER]
R:Run File.....>R F ilename [ENTER]
X:Execute File.....>X F ilename 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 ilename Size [ENTER]
C:License.....>C [ENTER]
```

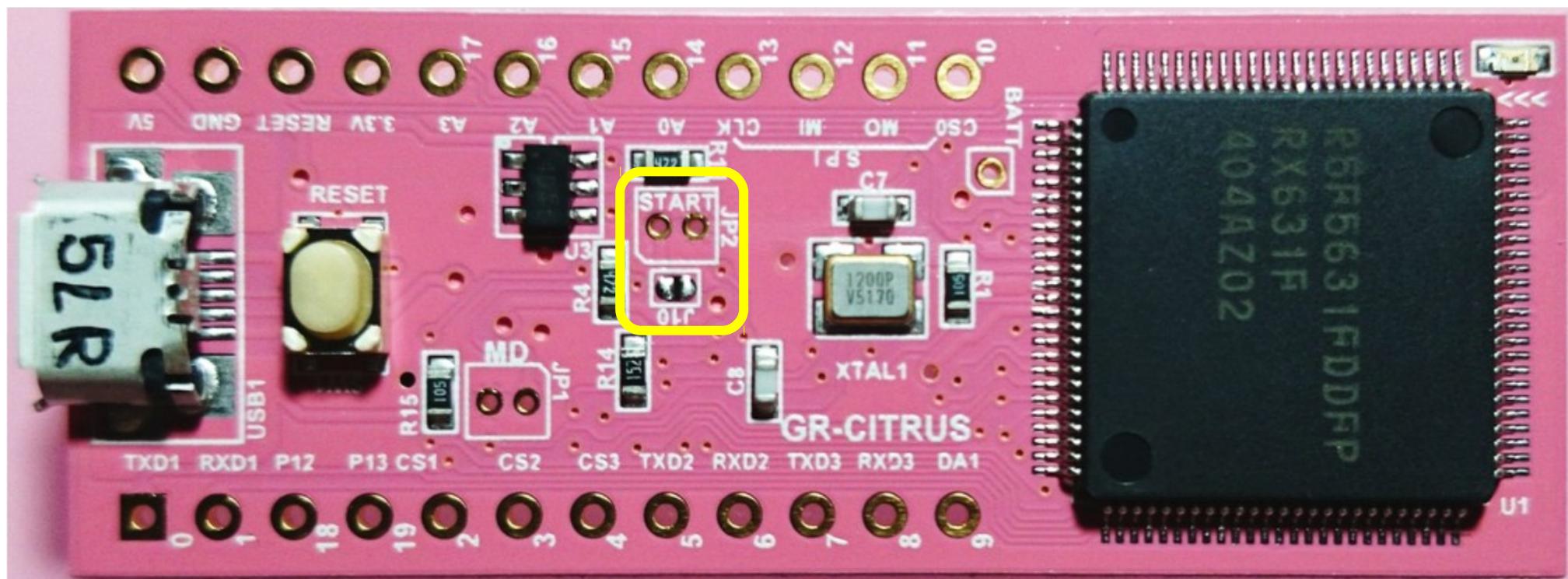
Below the menu, the text "WAKAYAMA RB Board Ver.ARIDA5-1.64(2016/4/14)f3(256KB), mruby 1.2.0 (H [ENTER])" is displayed. A red box highlights the command "C:License.....>C [ENTER]" and the resulting output:

```
>>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。

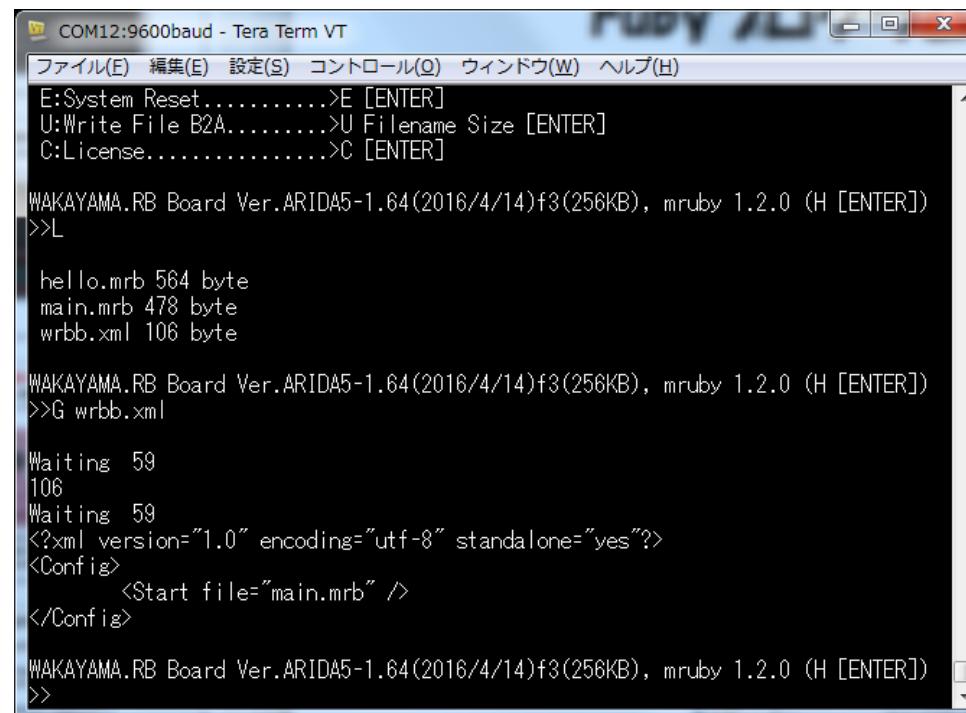
rubyプログラム自動実行の仕組み

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
```

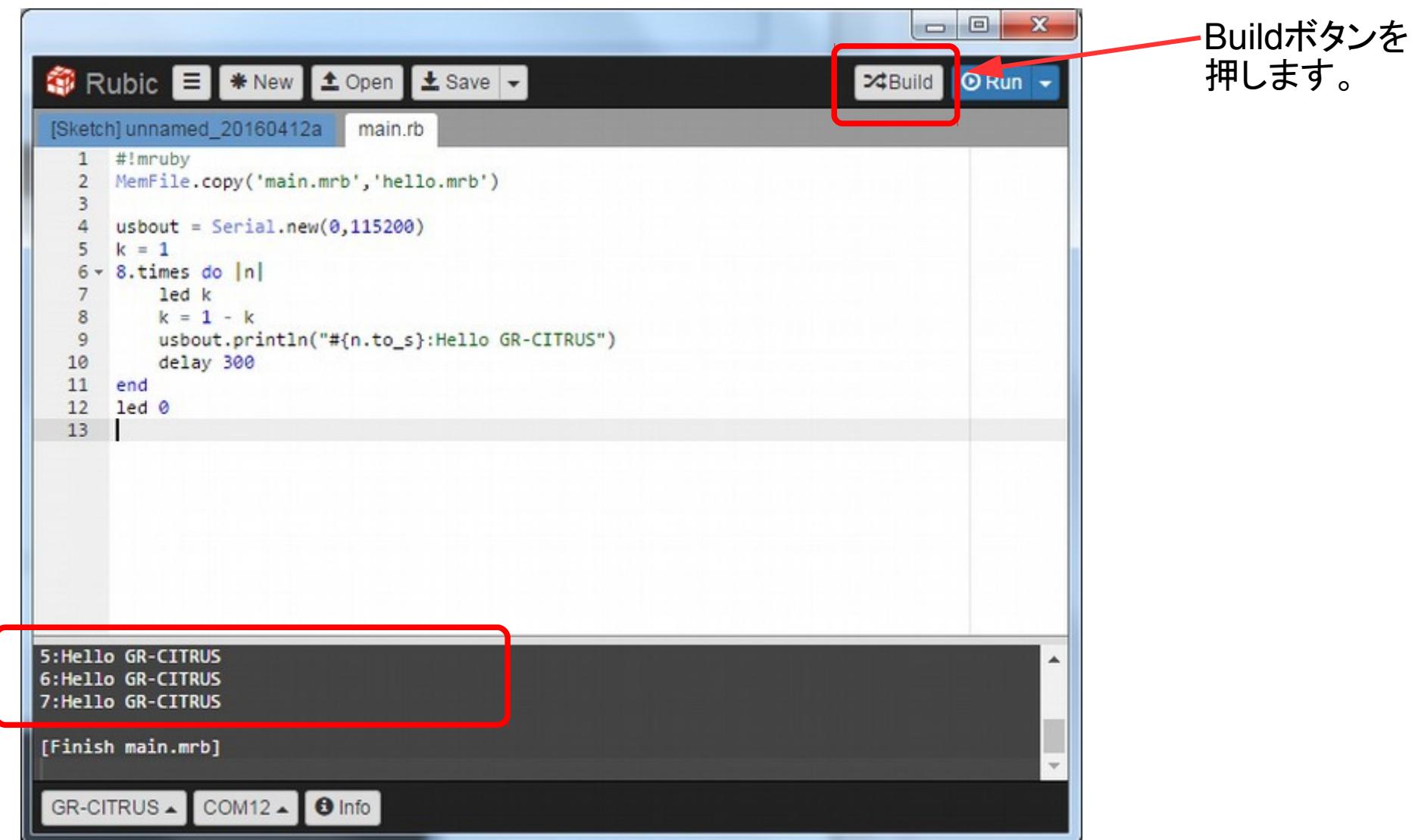
プログラムをデバッグしたい場合は、コンパイルオプションに `-g` を付けてコンパイルすることをお勧めします。エラーの行番号など詳しいエラーメッセージが出力されます。

```
$ ./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) 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

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, 0)
pinMode(5, 1)

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

10.times do
  led(1)
  delay(1000)
  led(0)
  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

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)。

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。
```

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

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(V1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()を呼んでおく必要があります。

ディレクトリを削除します: SD.rmdir(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 SDカードクラス

System.useSD()を呼んでおく必要があります。

使用例

System.useSD

```
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.useWiFi()`を呼んで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。

commnad: 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.useWiFi()を呼んで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.useWiFi()を呼んで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.useWiFi()`を呼んで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.useWiFi()`を呼んで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.useWiFi()`を呼んで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.useWiFi()`を呼んで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

WiFiクラス

System.useWiFi()を呼んで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.useWiFi() == 0)then
  Usb.println "WiFi Card can't use."
  System.exit()
end

Usb.print WiFi.version
```

メソッドの説明

WiFiクラス

System.useWiFi()を呼んで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.useWiFi() == 0)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

WiFiクラス

System.useWiFi()を呼んで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.useWiFi() == 0)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

WiFiクラス

System.useWiFi()を呼んで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, "hoho01111122222¥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

WiFiクラス

`System.useWiFi()`を呼んで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”)
```

`Usb.println WiFi.httpServer(80).to_s` #→ 80ポートでhttp受信します

```
20.times 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 != 0)then
    break
  end
  delay 1000
end
```

```
WiFi.send 0, “OK”      #→ OK を返しています。
WiFi.httpServer(-1)    #→ サーバを停止します
```