VS-RC202 取扱説明書

内容

1. はじめに

本書は、ESP-WROOM-02 を搭載したワイヤレスロボット制御ボード「VS-RC202」の使用方法と仕様について解説した取扱説明書です。ご使用になる前に、かならず本書をよくお読みいただき、安全にお使いください。

2. 仕様

サイズ	W40×D52 (mm) アンテナを含まない PCB 部分のサイズ
重量	16.6g
CPU	ESP-WROOM-02
電源	DC 4.5～8 V ニッケル水素充電池 4 本
出力	サーボモータ(または LED) ×10 圧電ブザー×1
入力	アナログセンサ入力×3 超音波センサ入力×1
インタフェース	USB microB ×1 シリアルポート(3.3v レベル)×1 I2C ポート(3.3v レベル)×1

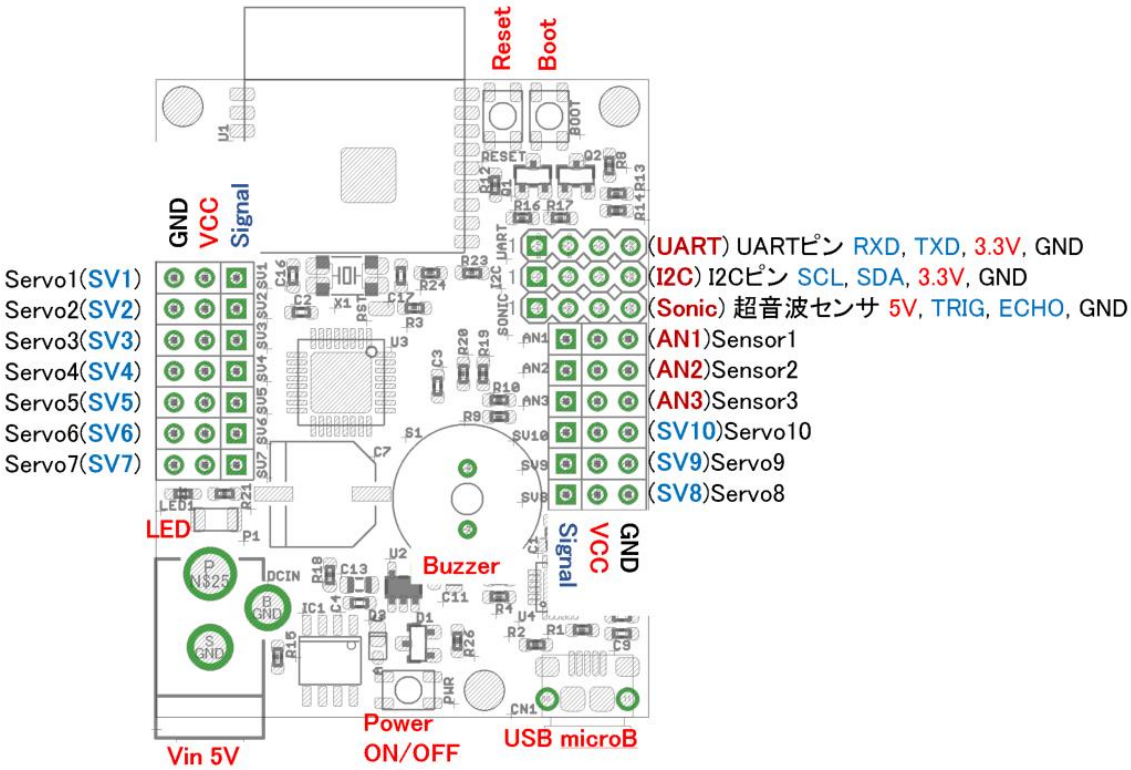
3. ご注意

本製品を取り扱う際には、注意事項に従い正しくお使いください。

- ○ VS-RC202(以降 本ボード)に強い衝撃を与えないでください。
- ○ 本ボードを水に濡らしたり、湿気やほこりの多い場所で使用したりしないでください。ショートなどによる故障が発生する恐れがあります。
- ○ 本ボードから煙が発生した場合、すぐに電源をお切りください。
- ○ 本ボードを幼児の近くで使用したり、幼児の手の届くところに保管したりしないでください。
- ○ 動作中、基板上の素子が高温になることがありますので、絶対に触れないでください。
- ○ 基板上の端子(金属部分)に触れると静電気により故障する恐れがあります。かならず基板の縁を触るようにしてください。
- ○ 基板上の端子同士が金属などでショートすると、過電流により故障する可能性があります。

4. 本体外観

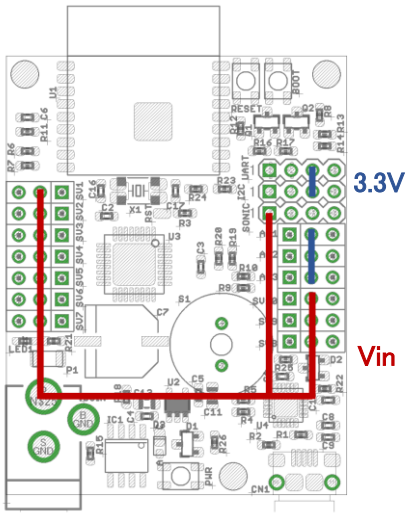
5. 各部詳細


Reset
Boot
GND
VCC
Signal
Servo1(SV1)
Servo2(SV2)
Servo3(SV3)
Servo4(SV4)
Servo5(SV5)
Servo6(SV6)
Servo7(SV7)
(UART) UARTピン RXD, TXD, 3.3V, GND
(I2C) I2Cピン SCL, SDA, 3.3V, GND
(Sonic) 超音波センサ 5V, TRIG, ECHO, GND
(AN1)Sensor1
(AN2)Sensor2
(AN3)Sensor3
(SV10)Servo10
(SV9)Servo9
(SV8)Servo8
LED
Buzzer
Signal
VCC
GND
Vin 5V
Power
ON/OFF
USB microB

[Vin]

電源入力範囲は 4.5V-8V です。通常は 4.8-5V での使用を推奨しています。アナログセンサ・I2C・UART の VCC は 3.3V となります。サーボモータと超音波センサの VCC は電源入力直結になります。

動作電圧 5V のサーボモータ・超音波センサを使用する場合、電源電圧を 5V にしてください。



[Power ON/OFF]

Power ボタンを押すと電源 ON、3 秒長押しで電源 OFF となります。ただし、USB 給電がある場合は自動的に電源 ON になります。尚、デフォルトでは電源電圧が 4.6V を下回り、USB 給電がない場合は自動で電源が消えます。

[LED]

電源 ON 時に点灯します。

[USB microB]

ESP-WROOM-02 の UART ポートと繋がっています。Arduino IDE でプログラミングするときに使用します。

[Reset・Boot ボタン]

Reset ボタンを押すと、ESP-WROOM-02 が再起動します。Boot ボタンを押しながら、Reset ボタンを押すと、ESP-WROOM-02 がブートモードで起動します。

[Servo1-10(SV1-10)]

サーボモータ、もしくは、LED を接続して使用してください。LPC1113 から直接大量の電流を流そうとすると故障の原因になります。LED を複数ドライブする場合は FET を使用してください。もしくはロボットショップで OctopusLED ライトブリック(青)を購入されると大変便利です。ブザー機能を有効にした場合は SV9,10 は使用することができないのでご注意ください。

[Sensor1-3(AN1-3)]

ADC への入力ポートです。アナログのセンサを接続します。

[SONIC]

超音波センサの入力ポートです。HC-SR04 を接続できます。

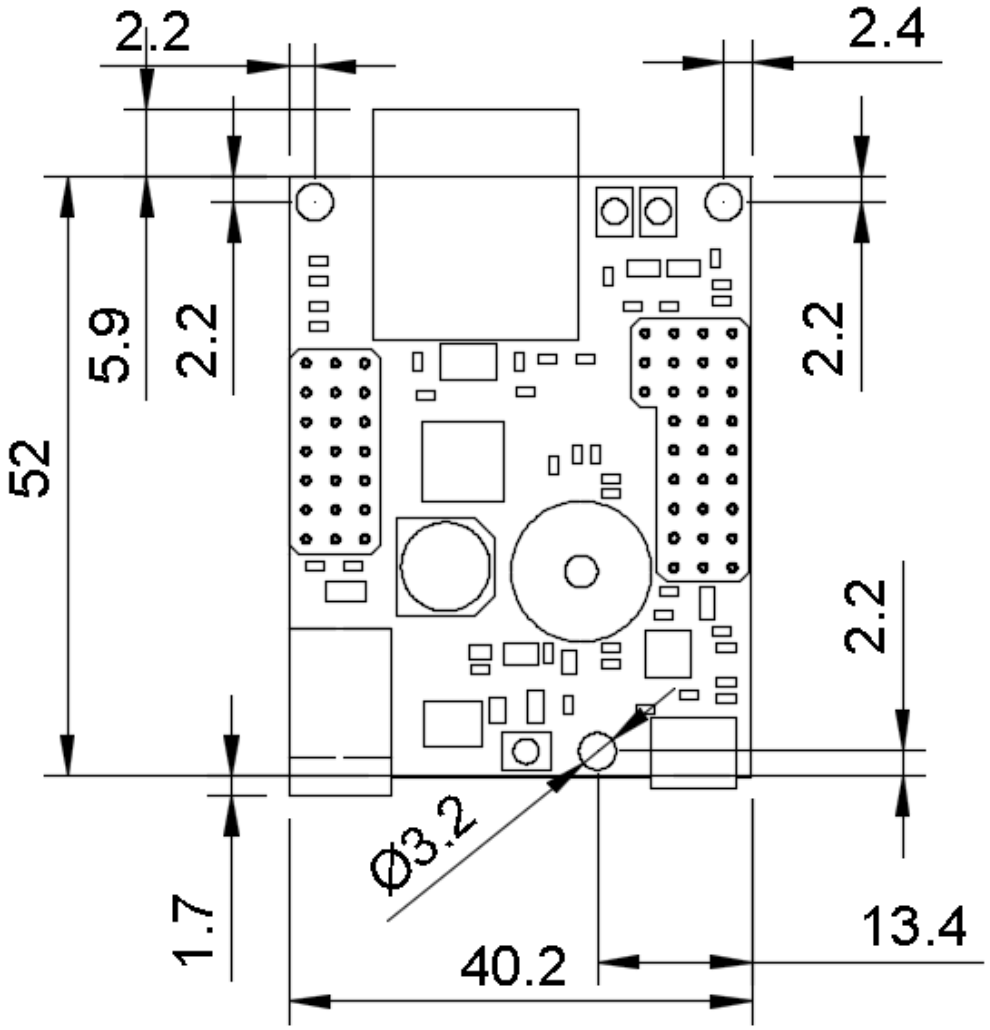
[I2C・UART]

I2C・UART バスの入力ポートです。UART ポートは ESP-WROOM-02 のプログラムの書き込みにも使用するので、Arduino IDE 等でプログラムを書き込む際は何も繋げないようにしてください。

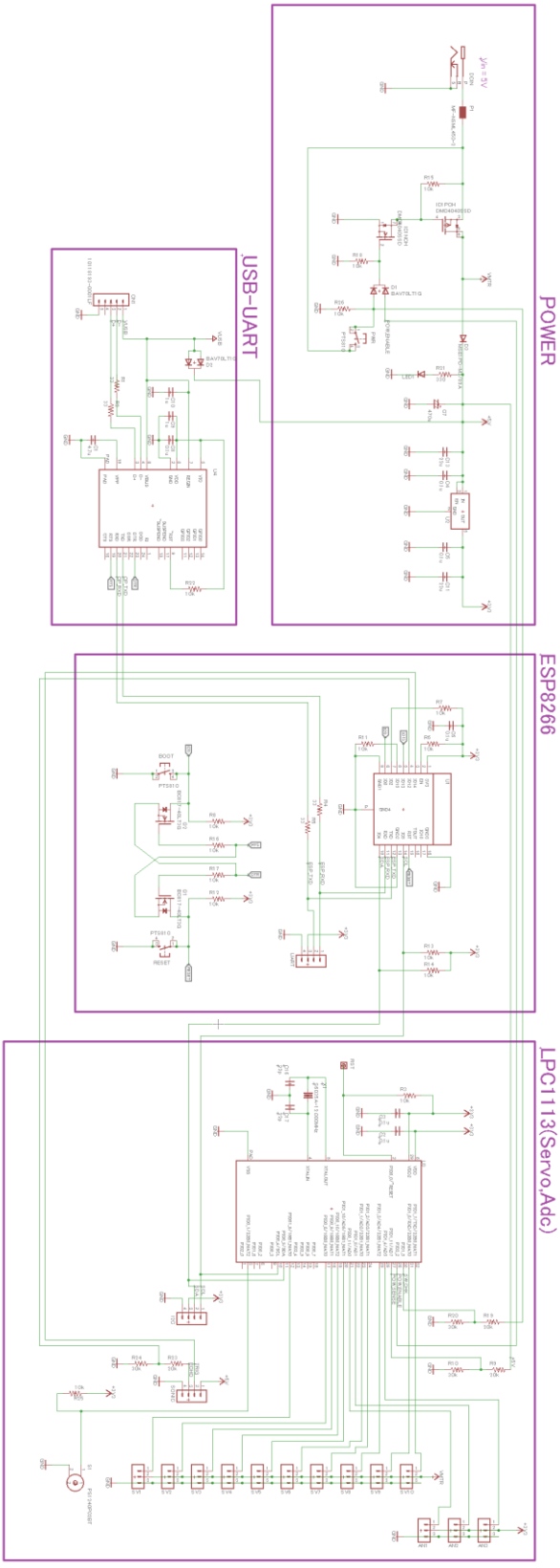
[Buzzer]

圧電ブザーで音を鳴らすことができます。

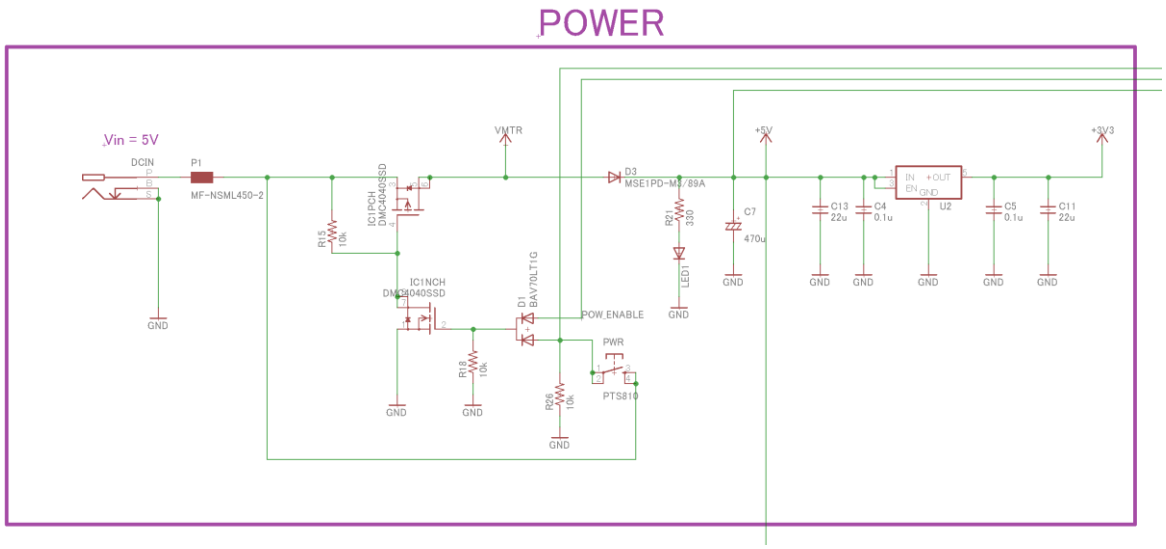
6. 基板外形

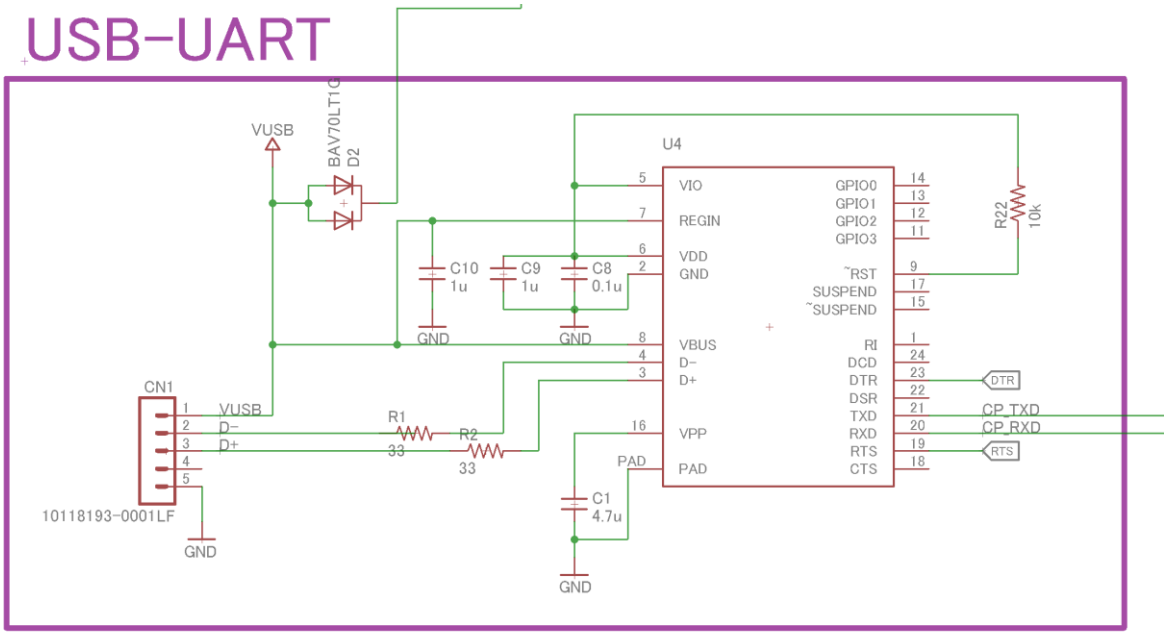


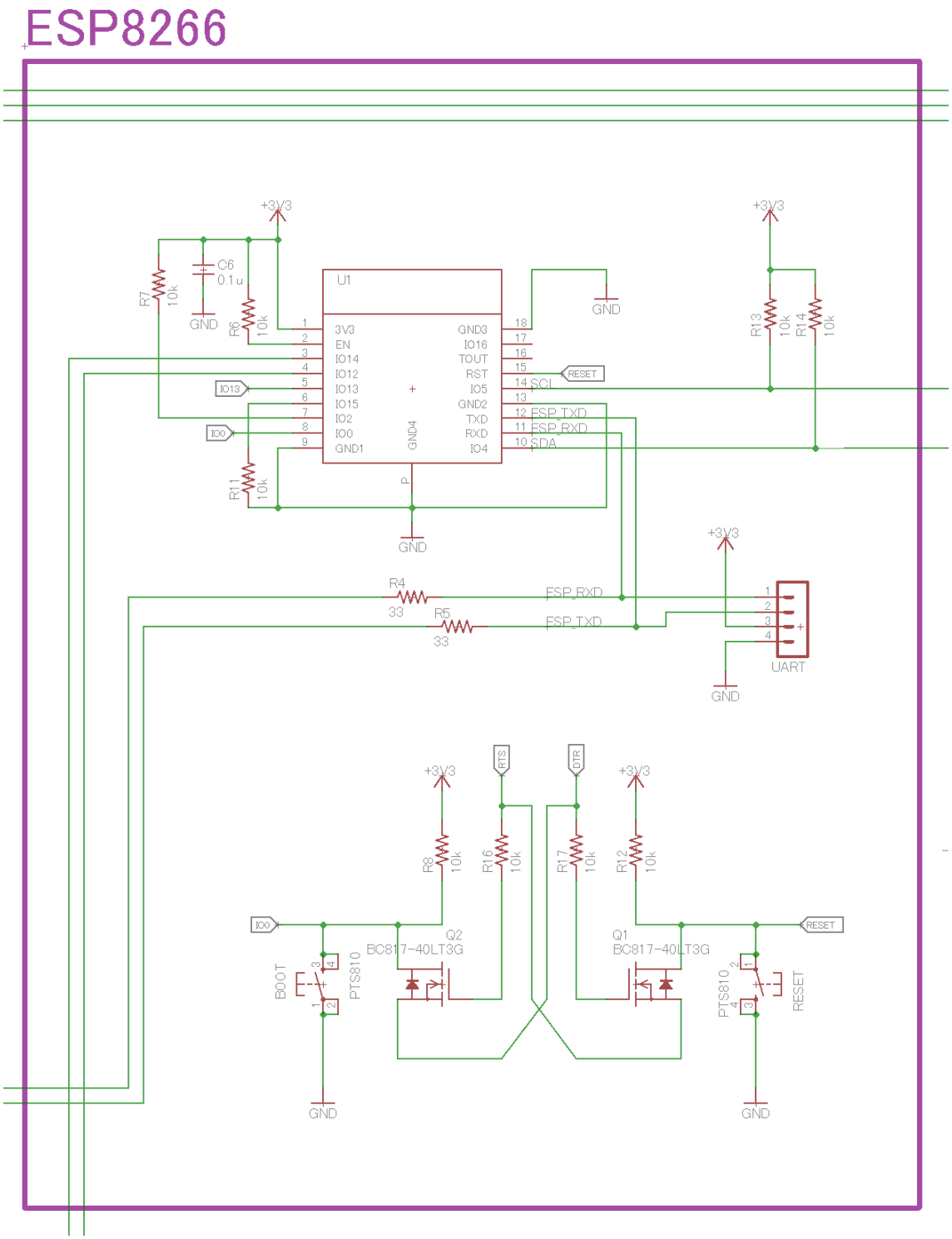
7. 回路図



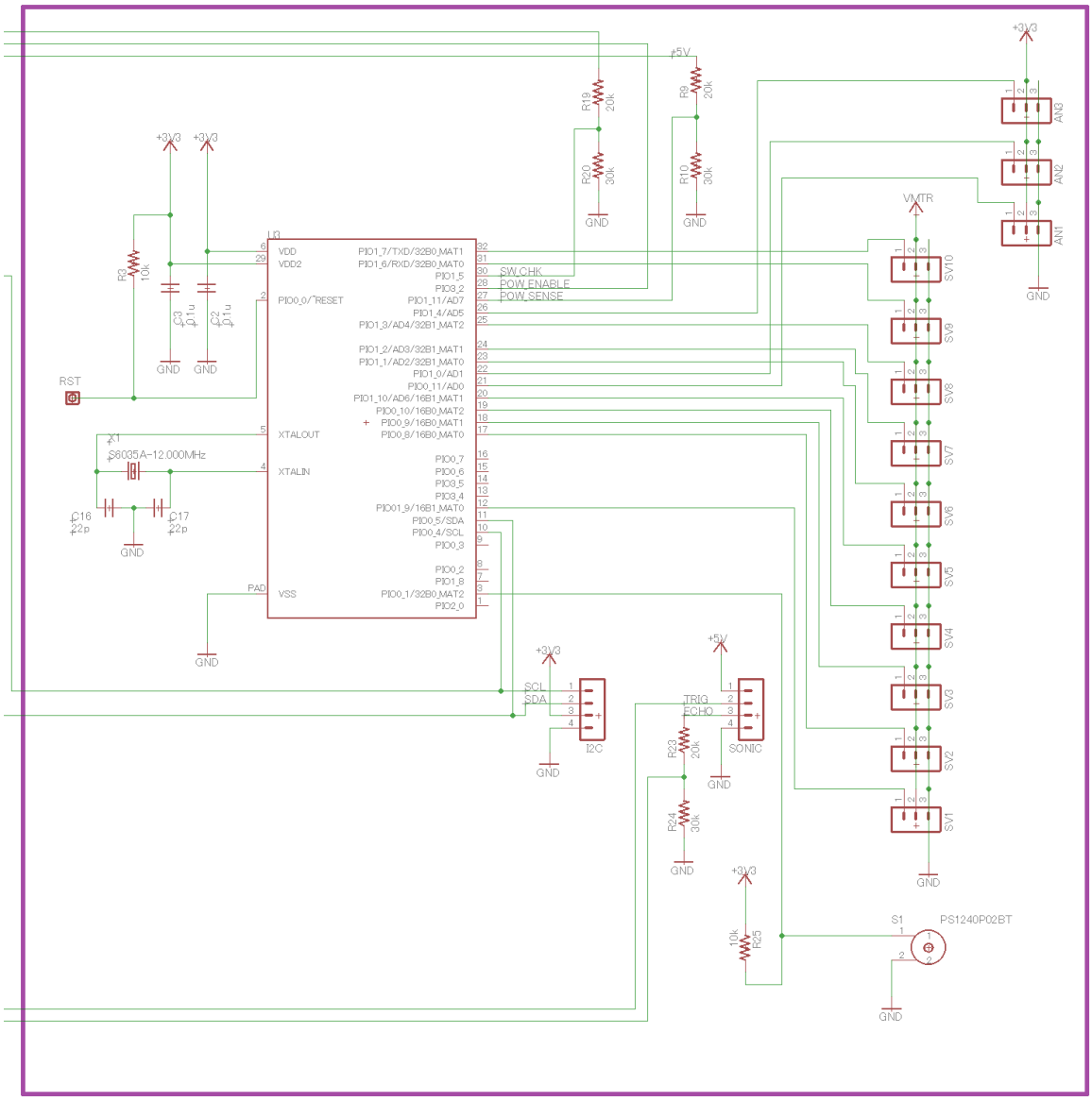
[拡大図]


POWER
Vin = 5V
DCIN
P1
MF-NSML450-2
GND
VMTR
IC1:PCH
DMC4040SSD
IC1:NCH
DMC4040SSD
R15
10k
R18
10k
R26
10k
D1
BAV70LT1G
POW_ENABLE
PWR
PTS810
D3
MSE1PD-M3/89A
R21
330
LED1
C7
470u
+5V
+3V3
C13
22u
C4
0.1u
U2
C5
0.1u
C11
22u


USB-UART
VUSB
BAV70LT1G
D2
U4
VIO
REGIN
VDD
GND
VBUS
D-
D+
VPP
PAD
GPIO0
GPIO1
GPIO2
GPIO3
~RST
SUSPEND
~SUSPEND
RI
DCD
DTR
DSR
TXD
RXD
RTS
CTS
R22
10k
C10
1u
C9
1u
C8
0.1u
CN1
VUSB
D-
D+
R1
33
R2
33
C1
4.7u
10118193-0001LF
CP_TXD
CP_RXD
DTR
RTS


ESP8266
+3V3
C6
0.1 u
R7
10k
R6
10k
GND
U1
3V3
EN
IO14
IO12
IO13
IO15
IO2
IO0
GND1
GND3
IO16
TOUT
RST
IO5
GND2
TXD
RXD
IO4
GND4
RESET
SCL
ESP_TXD
ESP_RXD
SDA
IO13
IO0
R11
10k
R13
10k
R14
10k
R4
33
R5
33
ESP_RXD
ESP_TXD
UART
R8
10k
R16
10k
R17
10k
R12
10k
RTS
DTR
Q2
BC817-40LT3G
Q1
BC817-40LT3G
BOOT
PTS810
RESET

LPC1113(Servo,Adc)



8. VS-RC202 のソフトウェア概要

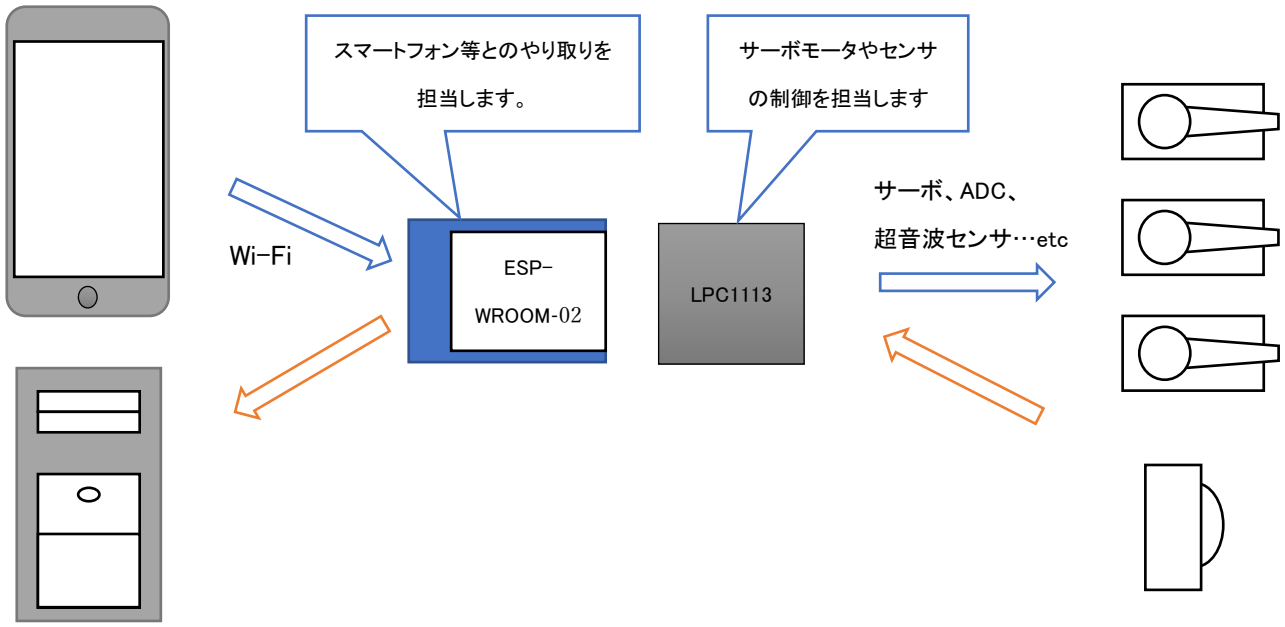
VS-RC202 は“ネットワークにつながるロボットを簡単に作る“をコンセプトにしています。ESP-WROOM-02 という Wi-Fi モジュールが搭載されており、Arduino IDE でプログラムできます。ユーザが直接プログラムするのはこの ESP-WROOM-02 になります。

ESP-WROOM-02 はモータやセンサを直接接続して制御することが苦手です。そこで、VS-RC202 には LPC1113 という ARM チップが搭載されており、これがサーボモータやセンサとのやり取りを全て代わりに行ってくれます。さらに、サーボモータの補間処理も行います。

プログラムの流れは基本的に以下のどちらかの処理になります。

“Wi-Fi 経由でスマートフォンなどから命令を受け取ったので、LPC1113 に何かを依頼する”

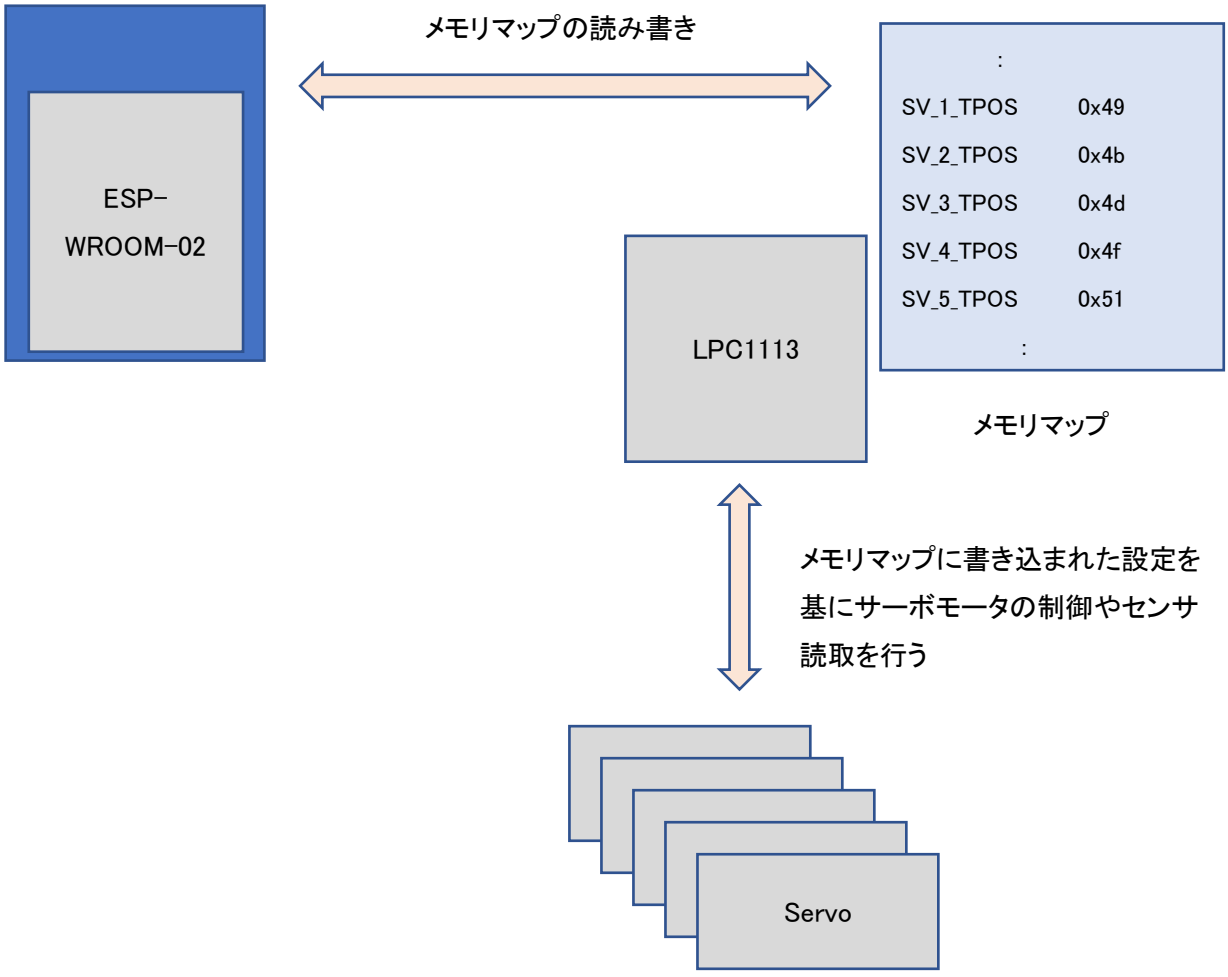
“LPC1113 に何かを依頼して、その結果をクラウドやウェブサービスに報告する”



9. プログラムの仕様

VS-RC202 に搭載されている LPC1113 はメモリマップという仮想的なレジスタメモリを持っています。このメモリマップの値を読み書きすることにより、サーボモータやセンサを制御します。

ユーザは Arduino IDE で専用のライブラリを使い、ESP-WROOM-02 が LPC1113 のメモリマップを操作するプログラムを作成します。



10. セットアップ

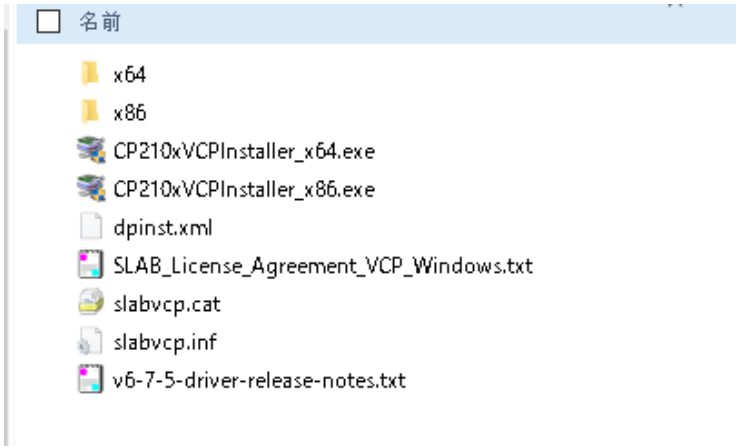
VS-RC202 の詳細なセットアップとサンプルの使い方に関しては VS-RC202 チュートリアルをご参照ください。VS-RC202 チュートリアルは以下の URL からダウンロード可能です。
https://www.vstone.co.jp/products/vs_rc202/download.html

[セットアップに必要なソフトウェア一覧]
Windows10 を例に PC で VS-RC202 を使用するためのセットアップ方法を説明します。2018/1/26 現在の情報を基にしています。

A) USB-UART 変換チップのドライバをインストールする

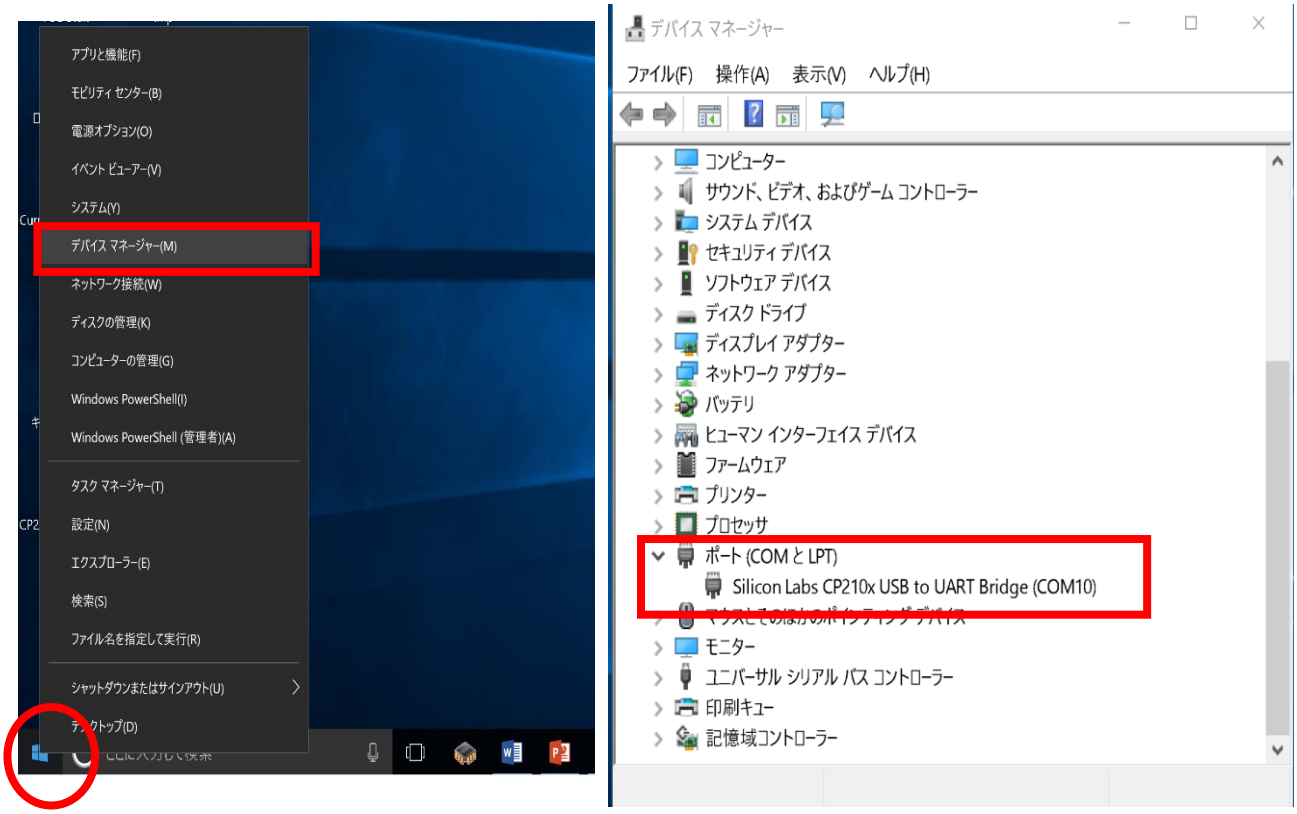
VS-RC202 を USB で PC に接続して、認識できるようにするためにドライバをインストールします。下記の URL から、Windows 用ドライバをダウンロードします。
https://jp.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers
[注]ドライバのインストール前に VS-RC202 を PC に接続しないでください。

CP210x_Windows_Drivers.zip を解凍すると、以下のファイルができるので、**64bitOS を使用している場合は CP210xVCPInstaller_x64.exe** をダブルクリックしてインストールします。**32bitOS を使用している場合は CP210xVCPInstaller_x86.exe** をダブルクリックしてインストールします。



ドライバをインストールしたら、VS-RC202 を付属の USB ケーブルで PC に接続してデバイスマネージャーを確認します。Windows10 の場合は、スタートボタン上で右クリックして、デバイスマネージャーを選択します。

デバイスマネージャーウィンドウのポート（COM と LPT）に、**Silicon Labs CP210x USB to UART Bridge(COMxx)**と表示されていれば正常にデバイスを認識しています。



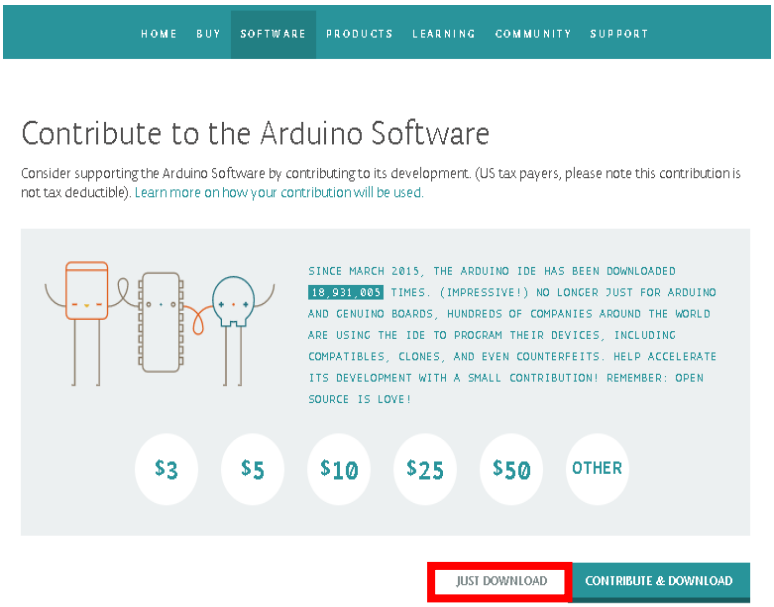
B) Arduino IDE のダウンロード

VS-RC202 は Arduino IDE でプログラムを作成します。下記の URL にアクセスして、最新の Arduino IDE の Windows Installer を選択します。

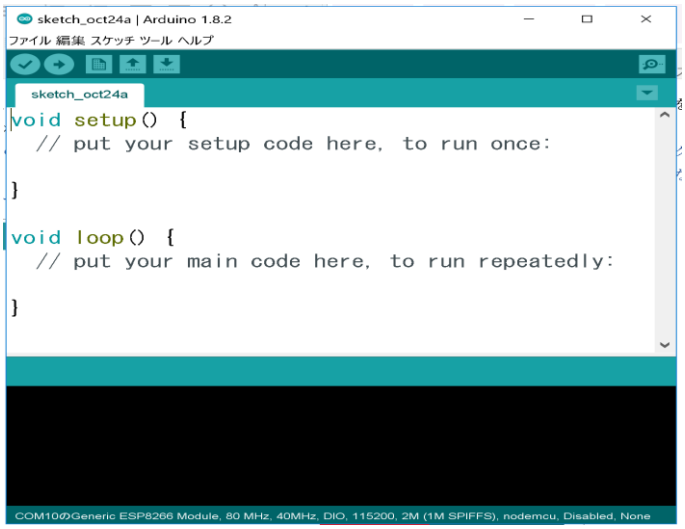
https://www.arduino.cc/en/Main/Software

Windows Installer を選択すると、以下のような画面が現れるので、JUST DOWNLOAD をクリックすれば、ダウンロードが開始されます。

（注）金額のようなものが出ていますが、これは Arduino IDE を開発しているプロジェクトに寄付できるというだけで、JUST DOWNLOAD を選択すれば、料金が発生することなどはありません。



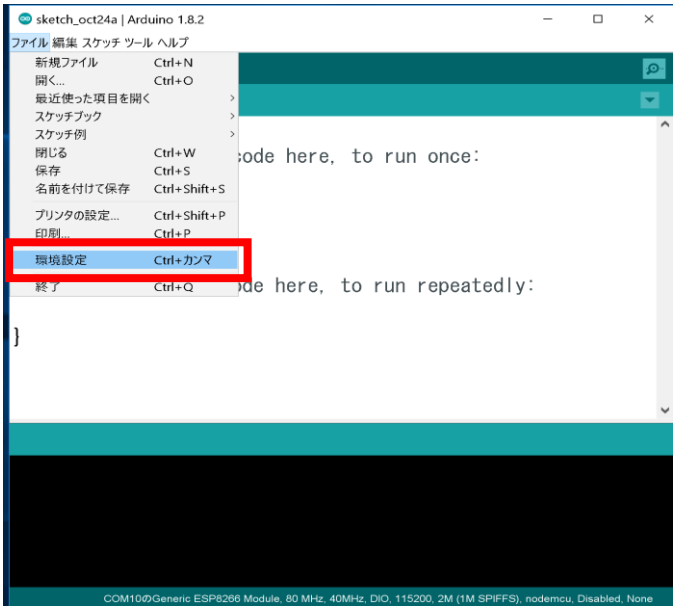
ダウンロードした arduino-xxxx-windows.exe をダブルクリックして指示に従いインストールすれば、準備完了です。デスクトップの Arduino IDE のアイコンをダブルクリックして、以下のウィンドウが表示されれば正常にインストールされています。



C) VS-RC202 を Arduino IDE でプログラムできるようにする

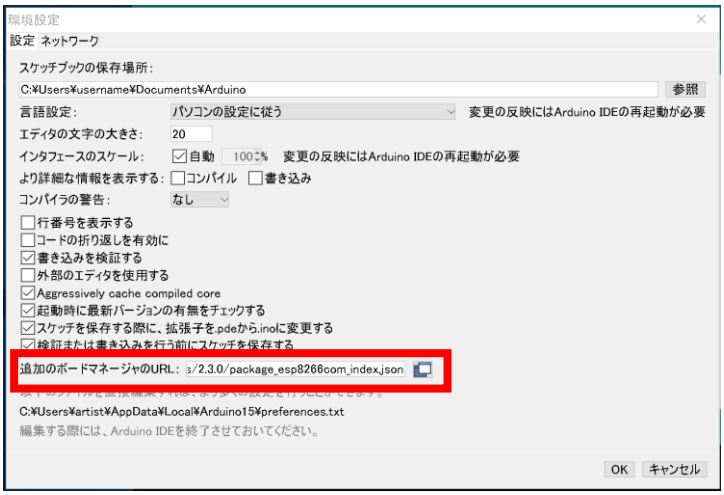
VS-RC202 を Arduino IDE でプログラムできるようにするには、追加の設定ファイルをインストールする必要があります。

Arduino IDE を起動して、メニューのファイルから環境設定を選択します。

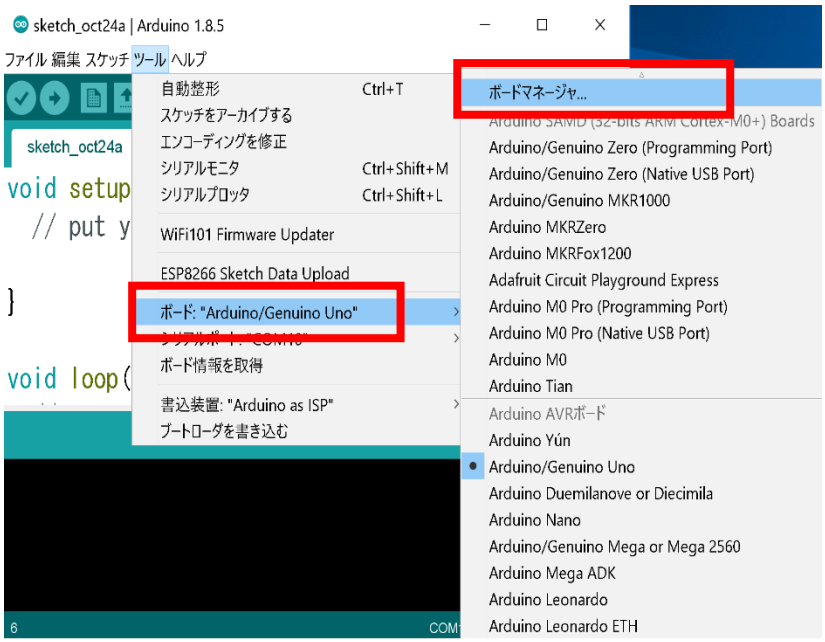


環境設定パネルの追加のボードマネージャ URL に以下の URL を張り付けて、OK ボタンを押します。

http://arduino.esp8266.com/stable/package_esp8266com_index.json



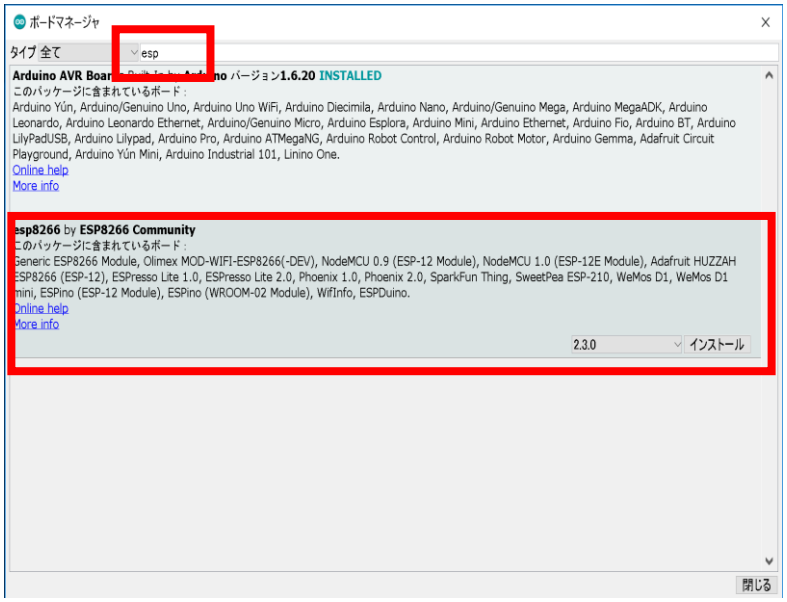
メニューのツールから、ボード＞ボードマネージャを選択します。



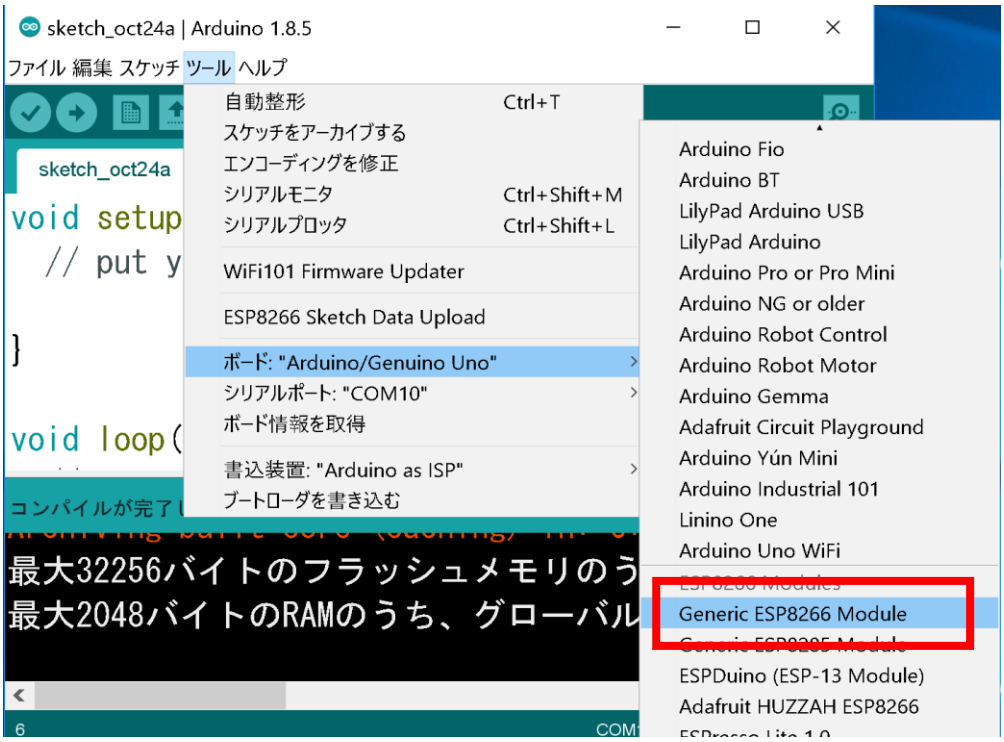
ボードマネージャが表示されたら上部にある検索ボックスで”esp”と入力します。

esp8266 by ESP8266 Community が見つかるので、バージョンを選択して、インストールします。

2019 年 8 月 1 日現在、2.5.2 で動作することを確認しています。

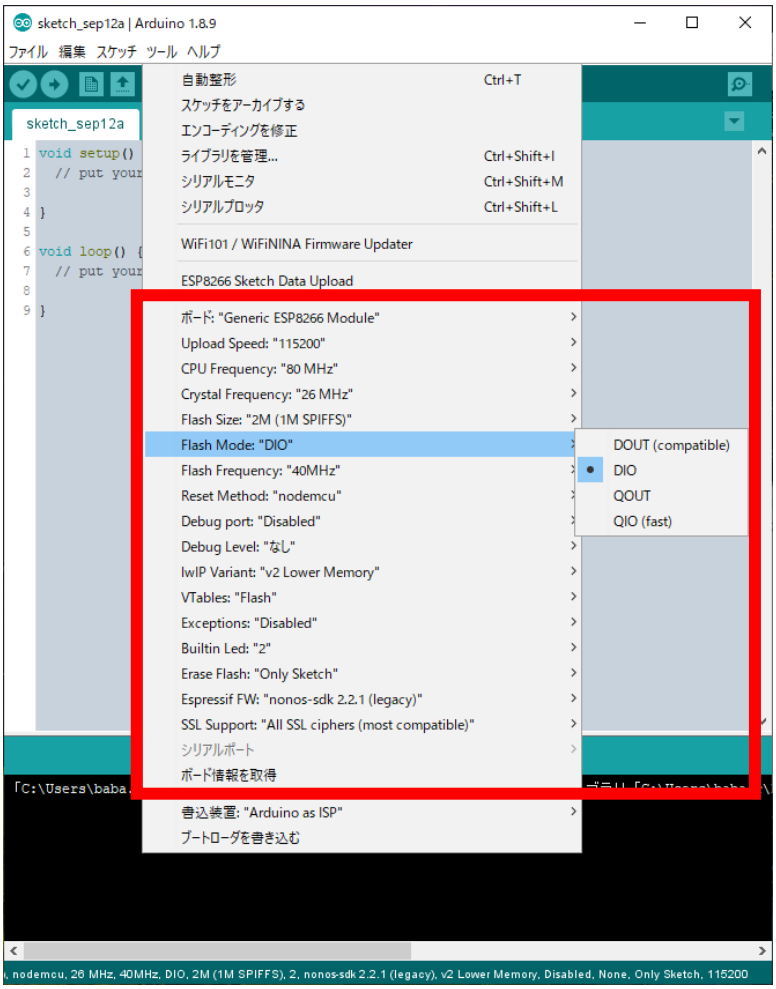


Esp8266 のインストールが終わったら、メニューのツールからボードで **Generic ESP8266 Module** が選択できるようになっていれば、正常にインストールされています。(ボード選択画面で下のほうにスクロールすると見つかります) Generic ESP8266 Module を選択してください。



Generic ESP8266 Module を選択すると、メニューのツールに複数の設定がでてくるので、以下のように設定します。一度設定すれば、次回以降は自動的に選択されます。

- Flash Mode : DIO
- Flash Frequency : 40Mhz
- Crystal Frequency : 26Mhz
- CPU Frequency : 80Mhz
- Flash Size : 2M(1M SPIFFS)
- Debug port : disabled
- Debug level : なし
- Reset Method : nodemcu
- Upload Speed : 115200
- シリアルポート : VS-RC202 が接続されているポート



D) VS-RC202 にプログラム以外のファイルを書き込めるようにする

VS-RC202 は Wi-Fi 機能を有し、簡易ウェブサーバーとして稼働し、HTML ファイルなどを配信することができます。ここでは配信する HTML 等を事前に VS-RC202 に書き込む準備をします。

esp8266fs-plugin を使うことにより Arduino IDE から VSRC202 のフラッシュメモリに HTML ファイル等を書き込むことができます。以下の URL にアクセスし、**ESP8266FS-×××.zip** をダウンロードして下さい。

※×××にはバージョン名が入ります。

2019 年 6 月 13 日現在、0.4.0 で動作することを確認しています。

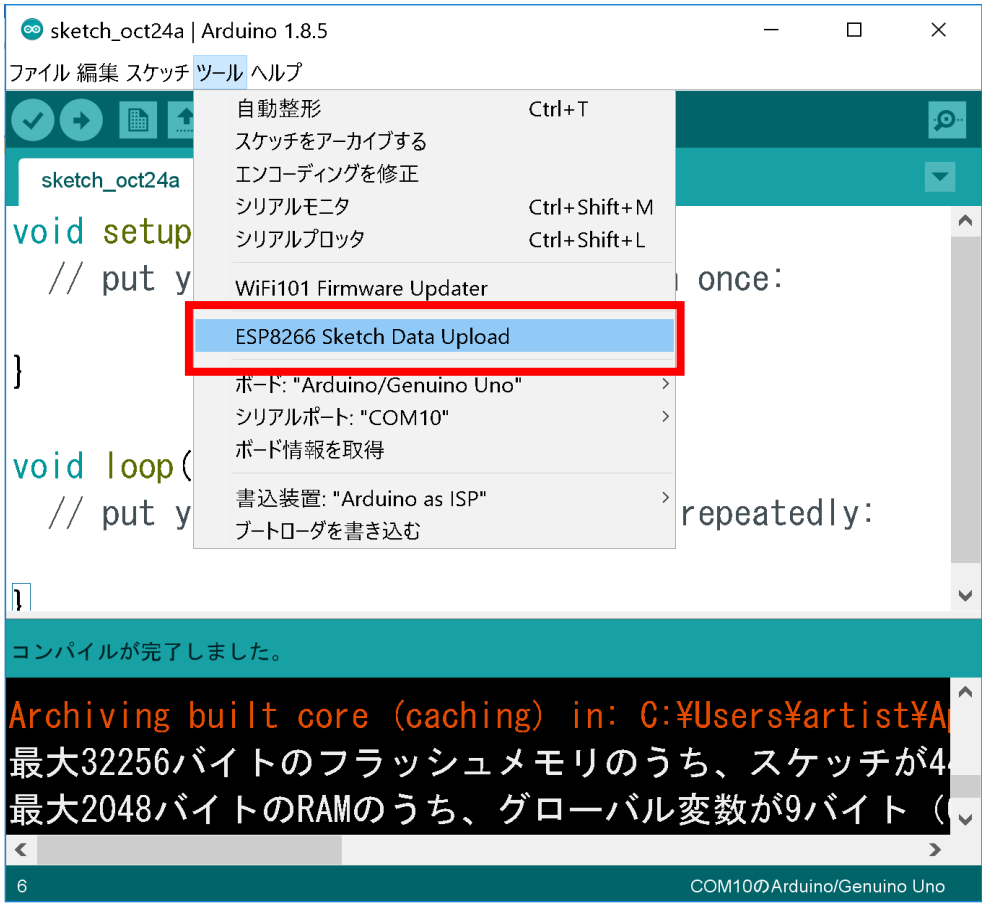
https://github.com/esp8266/arduino-esp8266fs-plugin/releases/

ESP8266FS-×××.zip を解凍すると、ESP8266FS というフォルダができます。Arduino のスケッチフォルダに tools というフォルダを作成して、ESP8266FS を tools の中に入れます。

デフォルトでは Arduino のスケッチフォルダはドキュメントフォルダの直下にあります。
C:¥Users¥username¥Documents¥Arduino

以下のように配置します。
C:¥Users¥username¥Documents¥Arduino**¥tools¥ESP8266FS**

配置したら一度 Arduino IDE を再起動して、メニューのツール＞ESP8266 Sketch Data Upload が表示されていれば正常にインストールされています。

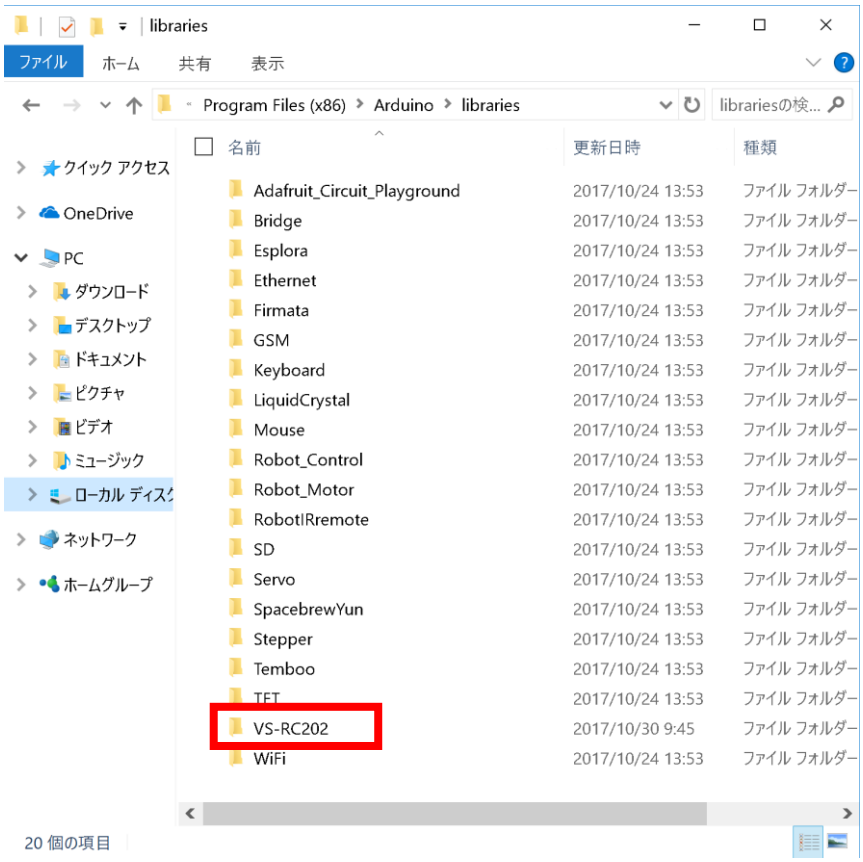


E)　VS-RC202 のライブラリを使用できるようにする

以下の URL から V-duino-ver.×××.zip をダウンロードします。
※×××にはバージョン名が入ります。
　基本的に最新のバージョンを使用してください。
https://github.com/vstoneofficial/V-duino/releases

　解凍すると、V-duino-ver.×××というフォルダができます。本フォルダ内に VS-RC202 というフォルダがあるので、このフォルダを Arduino の libraries フォルダの中に移動させてください。

　Windows10 (64bit 版)であれば、libraries フォルダは以下の位置にあります。
C:¥Program Files (x86)¥Arduino¥libraries



　以上で、ソフトウェアのセットアップは完了です。

11. vs-rc202.cpp 関数一覧

A) 初期化

関数	void initLib()
概要	初期化処理。プログラムの最初に必ず実行する
引数と戻値	[引数]なし [戻値]なし

使用例	setup(){
	　　initLib();
	}

B) センサ/電源管理

関数	void setSensEnable(int flag)
概要	センサ読込の Enable/Disable の切り替え。デフォルトでは有効(Enable)
引数と戻値	[引数]Enable(1)/Disable(0) [戻値] なし

使用例	setup(){
	　　setSensEnable(1);　　//センサ読込有効
	　　setSensEnable(0);　　//センサ読込無効
	}

関数	void setPullupEnable(int id, int flag)
概要	センサピンのプルアップ抵抗の Enable/Disable 切り替え。デフォルトでは無効(Disable)
引数と戻値	[引数] Enable(1)/Disable(0) [戻値] なし

使用例	setup(){
	　　setPullupEnable (1);　　//プルアップ有効
	　　setPullupEnable (0);　　//プルアップ無効
	}

関数	float getDist()
概要	超音波センサの値を取得
引数と戻値	[引数]なし [戻値]超音波センサの距離(cm)

使用例	loop(){
	float dist = getDist(); //距離を取得
	Serial.println(dist); //距離を表示
	}

関数	int readSens(int id)
概要	センサの値を取得する
引数と戻値	[引数]センサ番号(1～3) [戻値]センサ値

使用例	loop(){
	int sens[3];
	sens[0] = readSens(1);
	sens[1] = readSens(2);
	sens[2] = readSens(3);
	
	Serial.println(sens[0]);
	Serial.println(sens[1]);
	Serial.println(sens[2]);
	delay(500);
	}

関数	int readPow()
概要	電源電圧を取得する
引数と戻値	[引数]なし [戻値]電源電圧(mV)

使用例	loop(){
	int power_voltage = readPow(); //電源電圧取得
	Serial.print("Power voltage : ");
	Serial.print(power_voltage); //電源電圧を表示
	Serial.println();
	delay(500);
	}

関数	void powerOff ()
概要	電源を OFF にする。(遠隔操作で電源を切りたい場合に使用する)
引数と戻値	[引数]なし [戻値]なし

使用例	eventHandler{
	powerOff(); //WiFi 経由で特定のコマンドを受信した場合などに
	//実行すると、遠隔で電源を OFF にできる
	}

C) サーボモータ制御

関数	void servoEnable(int id, int flag)
概要	指定したピン番号のサーボモータの PWM 信号の Enable/Disable を切り替える。
引数と戻値	[引数]サーボモータ ID, Enable(1)/Disable(0) [戻値]なし

使用例	setup(){
	servoEnable(1, 1); //SV1 PWM Enable
	servoEnable(2, 0); //SV2 PWM Disable
	}

関数	void setServoOffset(int id, int deg)
概要	サーボモータの基準位置にオフセットを設定する。主にサーボモータの個体差によるずれの補正に使用。 設定範囲は-500～500
引数と戻値	[引数]サーボモータ ID, オフセット [戻値]なし

使用例	setup(){
	setServoOffset(1, 500); //SV1 offset 500
	setServoOffset(2, -500); //SV2 offset -500
	}

関数	void setServoLimitL(int deg)
概要	サーボモータの反時計回りの限界位置を設定する。設定範囲は-2500～2500
引数と戻値	[引数]限界位置 [戻値]なし

使用例	Setup(){
	setServoLimitL(-1800); //サーボモータの反時計まわりの限界位置-1800
	serServoLimitL(500); //サーボモータの反時計まわりの限界位置 500
	}

関数	void setServoLimitH(int deg)
概要	サーボモータの時計回りの限界位置を設定する。設定範囲は-2500～2500
引数と戻値	[引数]限界位置　[戻値]なし

使用例	Setup(){
	　setServoLimitH(1800);　　//サーボモータの時計まわりの限界位置 1800
	　serServoLimitH(-500);　　//サーボモータの時計まわりの限界位置-500
	}

関数	void setServoMode(int sv_mode)
概要	サーボモータの動作モードを設定。デフォルトでは順次実行モード(1) オーバーライドモード(0):目標位置を更新すると、現在の動作をキャンセルして即実行 順次実行モード(1):目標位置を設定すると、現在の補間動作が完了してから実行
引数と戻値	[引数]オーバーライドモード(0)/順次実行モード(1)　[戻値]なし

使用例	Setup(){
	　setServoMode(0);　　//オーバーライドモード
	　serServoMode(1);　　//順次実行モード
	}

関数	void setServoMovingTime(int moving_time)
概要	サーボモータの目標位置への遷移時間を設定。setServoMovingTime()を実行後サーボモータは全て設定された時間をかけて目標位置へ移動する。再度 setServoMovingTime()を実行すると、遷移時間が更新される。設定範囲は 20～100000(ミリ秒)
引数と戻値	[引数]遷移時間(ミリ秒)　[戻値]なし

使用例	Setup(){
	　setServoMovingTime(1000);　//遷移時間を 1000ms に設定
	　setServoMovingTime(600);　　//遷移時間を 600ms に更新
	}

関数	void setServoDeg(int id, int deg)
概要	サーボモータに目標位置を設定する。setServoMovingTime(int mv_time), moveServo()とセットで利用する。
引数と戻値	[引数]サーボモータ ID, 目標位置 [戻値]なし
使用例	Loop(){ 　setServoMovingTime(1000);　　//遷移時間 1000msec 　setServoDeg(1, 1500);　　//SV1 のサーボモータの目標位置を 1500 に設定

	setServoDeg(2,-1200); //SV2 のサーボモータの目標位置を-1200 に設定
	moveServo(); //moveServo()の実行から遷移時間 1000 をかけて、
	// SV1,2 のサーボモータが目標位置に到達する
	delay(1200); //遷移時間+α待ってから、次の動作を実行
	setServoMovingTime(1500);
	setServoDeg(1, -1200);
	setServoDeg(2,1500);
	moveServo();
	}

関数	void moveServo()
概要	目標位置の設定されたサーボモータを動かす。速度は setServoMovingTime()で設定された遷移時間による
引数と戻値	[引数]なし [戻値]なし

使用例	loop(){
	setServoMovingTime(1500); //遷移時間設定
	setServoDeg(1, 1500); //サーボモータの目標位置設定
	moveServo(); //移動開始
	}

注意事項	下記のように連続で同じ目標位置にサーボモータを動かそうとすると、正常に動作しなくなる場合があるのでご注意ください。
	setServoDeg(1, 1500);
	moveServo();
	setServoDeg(1, 1500);
	moveServo();

関数	void stopServo()
概要	回転中のサーボモータを停止させる。補間もリセットされる
引数と戻値	[引数]なし [戻値]なし

使用例	loop(){
	setServoMovingTime(1500);
	setServoDeg(1, 1500);
	moveServo(); //移動開始
	delay(500);
	stopServo(); //途中でサーボモータが停止する
	}

関数	int readServoLimitL()
概要	サーボモータの反時計回りの限界位置の設定を取得する
引数と戻値	[引数]なし [戻値]反時計回りの限界位置

使用例	loop(){
	int limit_l = reasServoLimitL(); //反時計回りの限界位置取得
	Serial.println(limit_l);
	delay(500);
	}

関数	int readServoLimitH()
概要	サーボモータの時計回りの限界位置の設定を取得する
引数と戻値	[引数]なし [戻値] 時計回りの限界位置

使用例	loop(){
	int limit_h = reasServoLimitH(); //時計回りの限界位置取得
	Serial.println(limit_h);
	delay(500);
	}

関数	int readServoOffset(int id)
概要	ID で指定したサーボモータのオフセットの設定を取得する
引数と戻値	[引数] サーボモータ ID [戻値]オフセット

使用例	loop(){
	int offset = readServoOffset(1); //SV1 のオフセットを取得
	Serial.println(offset);
	delay(500);
	}

関数	int readServoMovingTime()
概要	サーボモータの遷移時間の設定を取得する
引数と戻値	[引数]なし [戻値]目標位置までのサーボモータの遷移時間

使用例	loop(){
	int mv_time = readServoMovingTime(); //遷移時間を取得
	Serial.println(mv_time);
	delay(500);
	}

関数	int readServoPos(int id)
概要	サーボモータの現在位置を取得
引数と戻値	[引数]サーボモータ ID ［戻値]ID で指定したサーボモータの現在位置

使用例	loop(){
	int sv1_pos = readServoPos (1); //SV1 の現在位置取得
	Serial.println(sv1_pos);
	delay(500);
	}

関数	int servoAvailable()
概要	サーボモータが動作中（補間中）かどうかを取得する。順次実行モード（setServoMode(1)）の時に使用する。順次実行モードの場合、メモリマップの SV_n_TPOS が更新された場合、現在実行中のモーションを完了してから、SV_n_TPOS のモーションを読み出す。 更新した SV_n_TPOS の内容が実行されていない間は servoAvailable()は 0 を返す。SV_n_TPOS の内容が読みこまれたら（次のモーションを書き込めるようになったら）、servoAvailable()は 1 を返す。
引数と戻値	[引数]なし ［戻値]0/1

使用例	if(servoAvailable()){ //前回書きこんだモーションが実行されたら
	setMotion (motion[SV_NUM+1]); //次のモーションを書き込む
	}

関数	void setMotion(int motion[SV_NUM+1])
概要	遷移時間、及び、全てのサーボモータの目標位置を設定する。通常は servoAvailable()とセットで使用する。servoAvailable()=0 の時に実行すると、バッファ（次のモーション）が強制的に上書きされる。
引数と戻値	[引数]モーション配列 ［戻値]なし

使用例	//要素1から、遷移時間、 SV1-SV10 の目標位置
	int nextMotion[SV_NUM+1] = {600,1000,1000,1000,1000,0,0,0,0,0,0};
	
	if(servoAvailable()){ //書き込み可能かチェック
	setMotion (nextMotion); //次のモーションを書き込む
	}
	moveServo(); //移動開始

関数	void setMotion(int motion[SV_NUM+1], int sv_num)
概要	遷移時間と SV1 から指定した数のサーボモータに対し、次の目標位置を1回で設定する。 通常は servoAvailable()とセットで使用する。 SV1-4 をサーボモータ用に使い、SV7-10 を LED の制御に使う場合など、特定のサーボモータ(この場合は SV1-4)だけ遷移させたい場合に使用する。
引数と戻値	[引数]モーション配列、サーボモータ数 [戻値]なし

使用例	#define sv_num = 4;
	//要素1から遷移時間、 SV1-SV4 の目標位置
	int nextPose[sv_num+1] = {600,1000,1000,1000,1000};
	
	if(servoAvailable()){ //書き込み可能かチェック
	setMotion (nextPose, sv_num); //次のモーションを書き込む
	}
	moveServo(); //移動開始

D） モーションの再生

以下の4つの関数は組み合わせて使用する

関数	void setMotionNumber(int motion_number)
概要	Global 変数 motion_number に値を設定する 外部から操作があった場合等に再生したいモーションの番号を設定する。getMotionNumber()とセットで使用する どのモーション番号がどのモーションに対応するかは任意
引数と戻値	[引数]モーション番号 [戻値]なし

関数	int getMotionNumber()
概要	Global 変数 motion_number の値を取得する
引数と戻値	[引数]なし [戻値]モーション番号

関数	void playMotion(int motion[][SV_NUM+1], int array_length)
概要	モーションを繰り返し再生する。 モーションは 2 次元配列{遷移時間,SV1 目標位置 ～ ,SV10 目標値} array_length に 10 を設定した場合、 motion[0]->motion[1]-> ‥‥->motion[9]-> motion[0]…のように繰り返し実行される
引数と戻値	[引数]モーションの 2 次元配列、モーション数 [戻値]なし

関数	void playMotionOnce(int motion[][SV_NUM+1], int array_length)
概要	モーションを一度だけ再生する モーションは 2 次元配列{遷移時間,SV1 目標位置 ～ ,SV10 目標値} array_length に 10 を設定した場合、 motion[0]->motion[1]-> ‥‥->motion[9]の順に1度だけ実行される
引数と戻値	[引数]モーションの 2 次元配列、モーション数 [戻値]なし

モーション再生のサンプルコード	

使用例	//モーションを2次元配列で用意
	int motion1[3][11] = {{300,0,0,-600,0,0,0,0,0,0,0},
	{300,300,0,-600,0,0,0,0,0,0,0},
	{500,300,0,600,0,0,0,0,0,0,0}};
	int motion2[4][11] = {…};
	
	int getCommandFromUart(){

```
        //UART からコマンドを受信する関数
}
//入力やセンサ値をトリガーにして、setMotionNumber でモーション番号を設定する
getCommand(){
    int cmd = getCommandFromUart();
    swtich(cmd){
        case 1:
            setMotionNumber(1);
            break;
        case 2:
            setMotionNumber(2);
            break;
             :
    }
}

//現在設定されているモーション番号を参照して、モーションを実行
void selectMotion(){
   switch(getMotionNumber()){
      case 1:
         playMotion(motion1, 5);
         break;
      case 2:
         playMotion(motion2, 5);
         break;
             :
   }
}

//メインループ内でコマンド読込とモーション実行を繰り返す
void loop() {
   getCommand();
   selectMotion();
}
```

E) LED の制御

関数	void setLedMode(int id, int flag)
概要	指定したサーボモータのピンを LED 制御用に切り替える
引数と戻値	[引数] サーボモータ ID、Enable(1)/Disable(0) [戻値]なし

使用例	servoEnable(1, 1); //SV1 の PWM 信号を有効にする
	servoEnable(2, 1); //SV2 の PWM 信号を有効にする
	setLedMode(1, 1); //SV1 を LED モードに設定
	setLedMode(2, 1); //SV2 を LED モードに設定
	setLedMode(2, 0); //SV2 を通常モードに戻す

関数	void setLedBrightness(int id, int brightness)
概要	指定したピンの LED 輝度を設定する。範囲は 0～1000。 moveServo()を実行すると、輝度が変化する。
引数と戻値	[引数]LED の ID, 輝度 [戻値]なし

使用例	servoEnable(1, 1);
	setLedMode(1, 1); //SV1 を LED モードに設定
	setServoMovingTime(1000);
	setLedBrightness(1, 500); //SV1 の輝度を 500 にする
	moveServo(); //LED の輝度が設定した遷移時間をかけて変化する

関数	void setLedBrightnessDirect(int id, int brightness)
概要	指定したピンの LED 輝度を設定する。範囲は 0～1000。 実行した直後に指定した輝度になる。 setLedBrightness(int id, int brightness)よりも優先される。 setLedBrightnessDirect(ID, 0)とした場合（輝度を 0 に設定）、setLedBrightness(int id, int brightness)が有効になる
引数と戻値	[引数] LED の ID, 輝度 [戻値]なし

使用例	servoEnable(1, 1);
	setLedMode(1, 1); //SV1 を LED モードに設定
	setLedBrightnessDirect(1, 500); //SV1 の輝度を 500 にする(即座に反映される)

F) ブザーの制御

関数	void buzzerEnable(int flag)
概要	基板上の圧電ブザーの Enable/Disable を切り替える。 （注意）圧電ブザーを有効にすると SV9, 10 は使えなくなる。
引数と戻値	[引数]Enable(1)/Disable(0) [戻値]なし

使用例	buzzerEnable(1); //ブザーを有効にする。SV9, 10 は使えない
	buzzerEnable(0); //ブザーを無効にする。SV9, 10 が使えるようになる

関数	void setBuzzer(int scale, int beat, int tang)
概要	ブザーを鳴らす。各引数の値は次のページのマクロを使用する。 scale = 音程（ドレミファソラシド） beat = 音の長さ（全音符、2分音符など） tang = 音をつなげるか(タンギングかスラー)
引数と戻値	[引数]音程、 音の長さ、タンギングかスラー [戻値]なし

使用例	buzzerEnable(1);
	setBuzzer(PC4, BEAT4, TANG); //ドの4分音符をタンギングしながら鳴らす
	setBuzzer(PC4, BEAT4, TANG);
	setBuzzer(PC4, BEAT4, TANG);
	
	setBuzzer(PC4, BEAT4, SLUR); //ドの四分音符をスラーで鳴らす。
	setBuzzer(PC4, BEAT4, SLUR);
	setBuzzer(PC4, BEAT4, SLUR);
	
	/*
	テンポは全音符(BEAT1)の時間(1000 ミリ秒)を基準に設定される
	テンポを変更する場合は vs-rc202.h の BEAT1 の値を変更して調整可能
	#define BEAT1 1000 //全音符
	#define BEAT2 BEAT1/2 //2分音符
	#define BEAT4 BEAT1/4 //4分音符
	#define BEAT8 BEAT1/8 //8分音符
	
	#define TANG_LENGTH 20 //タンギングの息継ぎ時間は 20 ミリ秒
	*/

ブザー用マクロー覧	
音程	0 = 無音 1～88 = 鍵盤の 1～88 SH = # #define PN 0 #define PA0 1 #define PA0_SH 2 #define PB0 3 #define PC1 4 #define PC1_SH 5 #define PD1 6 #define PD1_SH 7 #define PE1 8 #define PF1 9 #define PF1_SH 10 #define PG1 11 #define PG1_SH 12 #define PA1 13 #define PA1_SH 14 #define PB1 15 #define PC2 16 #define PC2_SH 17 #define PD2 18 #define PD2_SH 19 #define PE2 20 #define PF2 21 #define PF2_SH 22 #define PG2 23 #define PG2_SH 24 #define PA2 25 #define PA2_SH 26 #define PB2 27 #define PC3 28 #define PC3_SH 29 #define PD3 30 #define PD3_SH 31 #define PE3 32

```
#define PF3      33
#define PF3_SH 34
#define PG3      35
#define PG3_SH 36
#define PA3      37
#define PA3_SH 38
#define PB3      39
#define PC4      40
#define PC4_SH 41
#define PD4      42
#define PD4_SH 43
#define PE4      44
#define PF4      45
#define PF4_SH 46
#define PG4      47
#define PG4_SH 48
#define PA4      49
#define PA4_SH 50
#define PB4      51
#define PC5      52
#define PC5_SH 53
#define PD5      54
#define PD5_SH 55
#define PE5      56
#define PF5      57
#define PF5_SH 58
#define PG5      59
#define PG5_SH 60
#define PA5      61
#define PA5_SH 62
#define PB5      63
#define PC6      64
#define PC6_SH 65
#define PD6      66
#define PD6_SH 67
#define PE6      68
```

	#define PF6 69 #define PF6_SH 70 #define PG6 71 #define PG6_SH 72 #define PA6 73 #define PA6_SH 74 #define PB6 75 #define PC7 76 #define PC7_SH 77 #define PD7 78 #define PD7_SH 79 #define PE7 80 #define PF7 81 #define PF7_SH 82 #define PG7 83 #define PG7_SH 84 #define PA7 85 #define PA7_SH 86 #define PB7 87 #define PC8 88
音の長さ[ms]	#define BEAT1 1000 //全音符 #define BEAT2 BEAT1/2 //2分音符 #define BEAT4 BEAT1/4 //4分音符 #define BEAT8 BEAT1/8 //8分音符
タンギングと スラー	#define TAN 20 //タンギングの長さ[ms] #define SLUR 0 //スラー #define TANG 1 //タンギング

G) メモリマップの読み書き

メモリマップのアドレスは巻末のメモリマップを参照してください。

関数	int read2byte(unsigned char addr)
概要	2バイトデータをメモリマップから読み出す
引数と戻値	[引数]メモリマップのアドレス [戻値]2 バイトデータ

使用例	int sv_1_pos = read2byte(SV_1_POS); //SV1 の現在位置の読出
	int sv_2_pos = read2byte(SV_2_POS); //SV2 の現在位置の読出

関数	int read1byte(unsigned char addr)
概要	1 バイトデータをメモリマップから読み出す
引数と戻値	[引数]メモリマップのアドレス [戻値]1 バイトデータ

使用例	int sens_enable = read1byte(SENS_ENABLE); //センサ読込有効フラグの読出
	int pwm_enable = read1byte(PWM_ENABLE1); //SV1 PWM 信号の有効フラグの読出

関数	int write2byte(unsigned char addr, short data)
概要	2 バイトデータをメモリマップに書き込む
引数と戻値	[引数]メモリマップのアドレス, 2バイトデータ [戻値]1

使用例	write2byte(SV_1_TPOS, 1000); //SV1 の目標位置を 1000 に設定
	write2byte(SV_2_TPOS, 500); //SV2 の目標位置を 500 に設定

関数	int write1byte(unsigned char addr, short data)
概要	1 バイトデータをメモリマップに書き込む
引数と戻値	[引数] メモリマップのアドレス, 1 バイトデータ [戻値]1

使用例	write1byte(PWM_ENABLE1, 1); //SV1 の PWM を有効にする
	write1byte(LED_MODE1, 1); //SV1 を LED モードに設定する

12. メモリマップの直接編集

ESP-WROOM-02 と LPC1113 は I2C で接続されており、用意された関数を使う以外に、直接メモリマップを書き換えることによっても VS-RC202 を操作可能です。なお、VS-RC202 ライブラリにデバイスとレジスタのアドレスはマクロ登録されています。

デバイスアドレス　　: DEV_ADDR
各レジスタのアドレス　　:巻末のレジスタ名
データフォーマット　　:リトルエンディアン

例1　Arduino の Wire 関数でサーボモータの遷移時間を書き込む

```
int moving_time = 1000;                                  //遷移時間 1000msec
char upper_sv_movint_time = moving_time << 8;
char lower_sv_movint_time = moving_time;

Wire.beginTransmission(DEV_ADDR);                        //デバイスアドレス指定
Wire.write(SV_MV_TIME);                                  //先頭レジスタアドレス指定
Write.write(lower_sv_movint_time);                       //SV_MV_TIME の下 1 バイト
Write.write(upper_sv_movint_time);                       // SV_MV_TIME の上 1 バイト
Wire.endTransmission();                                  //Stop コンディション送信
```

アドレスが連続するレジスタに関しては、1 回の通信で全て書き込むことができます。

例2　Arduino の Wire 関数でサーボモータの遷移時間と目標位置を 1 回で読みこむ

```
Wire.beginTransmission(DEV_ADDR);                        //デバイスアドレス指定
Wire.write(SV_MV_TIME);                                  //先頭レジスタアドレス指定
Write.write(upper_sv_movint_time);                       //SV_MV_TIME の上 1 バイト
Write.write(lower_sv_movint_time);                       // SV_MV_TIME の下 1 バイト
Write.write(lower_sv_1_tpos);                            //SV_1_TPOS の下 1 バイト
Write.write(upper_sv_1_tpos);                            // SV_1_TPOS の上 1 バイト
            :
Wire.endTransmission();                                  //Stop コンディション送信
```

読み出しを行う場合は、読み出したいレジスタのアドレスと、そのアドレスから何バイト読み出すかを指定します。一度 Write モードで読み出したいレジスタのアドレスを書き込み、通信を終了します。次に、Read モードで接続すると、書き込んだアドレスからデータを読み出します。

例3　Arduino の Wire 関数でサーボモータの現在位置を読み出します。

```
Wire.beginTransmission(DEV_ADDR);        //Write モードで通信
Wire.write(SV_1_POS);                    //レジスタのアドレスを書き込む
Wire.endTransmission();                  //一旦通信終了

unsigned char tmp[20];
int index = 0;
Wire.requestFrom(DEV_ADDR, 20);          //Read モードで 20byte 読みだす
while (Wire.available()) {               //アドレス SV_1_POS から 20byte 分読み込む
    tmp[index++] = Wire.read();
}
```

13. メモリマップ

[Addr] ： レジスタアドレス

[R/W] ： R[Readable], W[Writable]

[符号] ： U(unsigned), S(signed)

[Byte] ： レジスタのサイズ

Addr	R/W	レジスタ名	符号	Byte	初期値	備考
0x00	R/W	SENS_ENABLE	U	1	0x01	ADC の有効無効設定 0x00=無効 0x01=有効
0x01	R/W	POW_OFF	U	1	0x00	電源の遠隔 OFF 0x01 を書きこむとシャットダウンする (USB 電源は OFF にならない)
0x02	R	P_SW	U	1	0xff	電源ボタンフラグ、押されている間は 0x01 離すと 0x00
0x03	R/W	POW_ENABLE	U	1	0x01	電源電圧による自動シャットダウンの ON/OFF 電源電圧が P_TH の値を下回ったらシャットダウンする 0x00=無効 0x01=有効
0x04	R	SENS_1	U	2	0x0000	センサ(AN1～AN3)の値 Range 0 ～ 1023
0x06	R	SENS_2	U	2	0x0000	
0x08	R	SENS_3	U	2	0x0000	
0x0a	R/W	PULLUP_1	U	1	0x00	センサ(AN1～AN3)のプルアップ抵抗の設定 0x00=無効 0x01=有効
0x0b	R/W	PULLUP_2	U	1	0x00	
0x0c	R/W	PULLUP_3	U	1	0x00	
0x0e	R	POWER	U	1	0xff	電源電圧 (mA)
0x10	R/W	P_TH	U	2	0xf811	シャットダウンする閾値電圧(mA) (リトルエンディアン)0xf811 = 0x11f8 = 4600 mA
0x12	R/W	SV_L_LIMIT	S	2	0xf8f8	反時計回りの限界位置 range -2500 ～ 2500 **可動範囲の狭いサーボモータの場合、実際の可動範囲を超えて、設定可能なため注意**
0x14	R/W	SV_H_LIMIT	S	2	0x0807	時計回りの限界位置 range -2500～2500 **可動範囲の狭いサーボモータの場合、実際の可動範囲を超えて、設定可能なため注意**

0x16	R/W	SV_1_OFFSET	S	2	0x0000	サーボモータ 1～10 のオフセット range -500 ～ 500
						(例)サーボモータ1にオフセット 200 を設定する
						SV_1_OFFSET = 0x00c8 //オフセット 200
						SV_1_TPOS = 0x05dc //目標位置 1500

0x18	R/W	SV_2_OFFSET	S	2	0x0000	
0x1a	R/W	SV_3_OFFSET	S	2	0x0000	
0x1c	R/W	SV_4_OFFSET	S	2	0x0000	

0x1e	R/W	SV_5_OFFSET	S	2	0x0000	//オフセット+目標位置 1700
						実際に移動する位置 = 0x06a4

0x20	R/W	SV_6_OFFSET	S	2	0x0000	
0x22	R/W	SV_7_OFFSET	S	2	0x0000	
0x24	R/W	SV_8_OFFSET	S	2	0x0000	
0x26	R/W	SV_9_OFFSET	S	2	0x0000	
0x28	R/W	SV_10_OFFSET	S	2	0x0000	
0x2a	R	SV_1_POS	S	2	0x0000	サーボモータ1～10 の現在位置
0x2c	R	SV_2_POS	S	2	0x0000	
0x2e	R	SV_3_POS	S	2	0x0000	
0x30	R	SV_4_POS	S	2	0x0000	
0x32	R	SV_5_POS	S	2	0x0000	
0x34	R	SV_6_POS	S	2	0x0000	
0x36	R	SV_7_POS	S	2	0x0000	
0x38	R	SV_8_POS	S	2	0x0000	
0x3a	R	SV_9_POS	S	2	0x0000	
0x3c	R	SV_10_POS	S	2	0x0000	
0x3e	R/W	PWM_ENABLE1	U	1	0x00	PWM 信号の ONOFF 0x00=PWM_OFF 0x01 = PWM_ON 起動直後は PWM OFF（サーボモータが脱力した状態）となっている **サーボモータを動かす際は最初に PWM_ENABLEn に 0x01 を書きこむ**
0x3f	R/W	PWM_ENABLE2	U	1	0x00	
0x40	R/W	PWM_ENABLE3	U	1	0x00	
0x41	R/W	PWM_ENABLE4	U	1	0x00	
0x42	R/W	PWM_ENABLE5	U	1	0x00	
0x43	R/W	PWM_ENABLE6	U	1	0x00	
0x44	R/W	PWM_ENABLE7	U	1	0x00	
0x45	R/W	PWM_ENABLE8	U	1	0x00	
0x46	R/W	PWM_ENABLE9	U	1	0x00	
0x47	R/W	PWM_ENABLE10	U	1	0x00	
0x48	R/W	SV_MV_TIME	U	2	0x0000	サーボモータの現在位置から目標位置までの移動時間 [ms] (例) 0x03e8 = 1000 msec
0x4a	R/W	SV_1_TPOS	S	2	0x0000	サーボモータ1～10 の目標位置 設定可能範囲 = 限界位置以内 **実際の可動範囲 = 限界位置 + オフセット** （例） SV_L_LIMIT = −1800 SV_H_LIMIT = 1800
0x4c	R/W	SV_2_TPOS	S	2	0x0000	
0x4e	R/W	SV_3_TPOS	S	2	0x0000	
0x50	R/W	SV_4_TPOS	S	2	0x0000	
0x52	R/W	SV_5_TPOS	S	2	0x0000	
0x54	R/W	SV_6_TPOS	S	2	0x0000	

0x56	R/W	SV_7_TPOS	S	2	0x0000	SV_1_OFFSET = 500 設定可能範囲 −1800 ～ 1800 実際の可動範囲 −1300 ～ 2300
0x58	R/W	SV_8_TPOS	S	2	0x0000	
0x5a	R/W	SV_9_TPOS	S	2	0x0000	
0x5c	R/W	SV_10_TPOS	S	2	0x0000	
0x5e	R/W	SV_START	U	1	0x00	0x01 を設定するとサーボモータが動き出す SV_MV_TIME と SV_n_TPOS を設定し終わった後に 0x01 を書き込むとサーボモータが稼働する。
0x5f	R/W	SV_CANCEL	U	1	0x00	0x01 を設定すると現在の動きをキャンセルする
0x60	R	SV_STATUS	U	1	0x00	0x00 サーボモータが止まっている 0x01 サーボモータが動いている
0x61	R	BUF_STATUS	U	1	0x00	0x00 バッファが空 0x01 バッファに次の値が入っている
0x62	R/W	SV_MODE	U	1	0x00	0x00 オーバーライド　0x01 順次実行モード
0x64	R/W	LED_MODE1	U	1	0x00	サーボモータピンを LED モードに切り替える 0x00 =サーボモータモード 0x01=LED モード LED モードでは、SV_n_TPOS は LED の輝度となる。 range 0～1000 （例）SV10 を LED モードにして最大輝度で光らせる LED_MODE10 = 0x01 SV_10_TPOS = 1000 SV_START = 0x01
0x65	R/W	LED_MODE2	U	1	0x00	
0x66	R/W	LED_MODE3	U	1	0x00	
0x67	R/W	LED_MODE4	U	1	0x00	
0x68	R/W	LED_MODE5	U	1	0x00	
0x69	R/W	LED_MODE6	U	1	0x00	
0x6a	R/W	LED_MODE7	U	1	0x00	
0x6b	R/W	LED_MODE8	U	1	0x00	
0x6c	R/W	LED_MODE9	U	1	0x00	
0x6d	R/W	LED_MODE10	U	1	0x00	
0x6e	R/W	LED_BRIGHT1	U	2	0x0000	LED モードで、LED_BRIGHTn に値（LED 輝度）を書き込むと、すぐに LED に反映される。 LED_BRIGHTn は SV_n_TPOS より優先される。 range 0～1000 LED_BRIGHTn は即座に光らせる場合に使用する。 SV_n_TPOS はゆっくり光らせる場合に使用する。 LED モードで SV_n_TPOS を有効にするには、LED_BRIGHTn を 0x0000 にする必要がある。
0x70	R/W	LED_BRIGHT2	U	2	0x0000	
0x72	R/W	LED_BRIGHT3	U	2	0x0000	
0x74	R/W	LED_BRIGHT4	U	2	0x0000	
0x76	R/W	LED_BRIGHT5	U	2	0x0000	
0x78	R/W	LED_BRIGHT6	U	2	0x0000	
0x7a	R/W	LED_BRIGHT7	U	2	0x0000	
0x7c	R/W	LED_BRIGHT8	U	2	0x0000	
0x7e	R/W	LED_BRIGHT9	U	2	0x0000	
0x80	R/W	LED_BRIGHT10	U	2	0x0000	

0x82	R/W	BUZZER_ENABLE	U	1	0x00	ブザーの Enable/Disable 0x00=ブザーDisable　0x01=ブザーEnable **ブザーEnable にすると、サーボモータ 9,10 が使用不可**
0x83	R/W	BUZZER_SCALE	U	1	0x00	ブザーの音色 0x00 = 無音 0x01～0x58 = ピアノの 88 盤に対応 （例） BUZZER_SCALE = 0x34(=52) C5 BUZZER_SCALE = 0x49(=73) A6