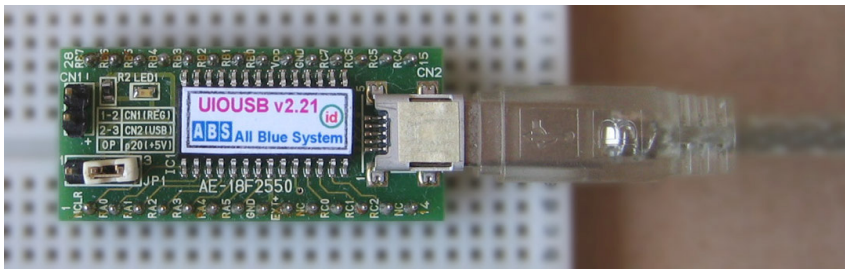
# UIOUSB ユーザーマニュアル

**ABS-9000 UIOUSB**

**UIOUSB User Manual Rev A.2.7**

**2013/03/14**

**Firmware version: “2.21”**





オールブルーシステム (All Blue System)

ウェブページ: www.allbluesystem.com

コンタクト:contact@allbluesystem.com

# 1 このマニュアルについて

## 1.1 著作権および登録商標



WindowsXP,Windows2000 はマイクロソフト社の登録商標です。

PIC は Microchip Technology Incorporatedの登録商標です。

## 1.2 連絡先

オールブルーシステム (All Blue System)

ウェブページ http://www.allbluesystem.com

メール contact@allbluesystem.com

# 2 使用条件およびライセンス

本ソフトウェア(ファームウエア)はオールブルーシステムの ABS-9000 DeviceServer のライセンスを購入されて、その DeviceServerと組み合わせて使用する場合には、複数のプロセッサにインストールして使用することができます。この場合には本ソフトウェアを他の製品に組み込んだり、サポート業務を行なうこともできます。

前述の ABS-9000 DeviceServer のライセンスを購入された場合の他に、オールブルーシステムから 本ソフトウェアのライセンスを購入された場合に、本ソフトウェア(ファームウエア)の使用ライセンスをお客様に提供いたします。ハードウエアと組み合わせたアプリケーション全体についてのライセンスや、ハードウエアと組み合わせたアプリケーションのサポートは、オールブルーシステムは提供致しません。

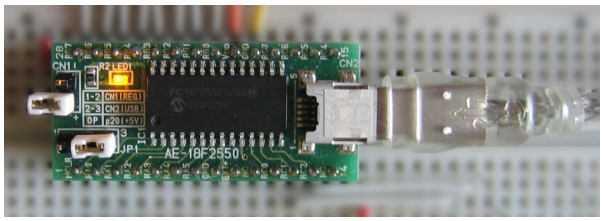
本ソフトウェアをオールブルーシステムの ABS-9000 DeviceServer と組み合わせずに単体で使用する場合や、本ソフトウェアライセンスを別途購入していない場合には、個人目的でのみこのソフトウェアを使用することができます。この場合は、業務用途や商用目的で本ソフトウェアを使用したり、他の製品に組み合わせて使用したり、サポート業務をおこなうことはできません。

本ソフトウェアはハイリスクな目的に使用することはできません。ハイリスクな目的とは、原子力、航空、直接的または間接的に人体に死傷を及ぼす可能性のある装置等に使用することを指します。オールブルーシステムは、本ソフトウェアにエラー、バグ等の不具合がないこと、若しくは中断なく稼動すること又は本ソフトウェアの使用がお客様及び第三者に損害を与えないことを保証しません。この事に同意していただけない場合は使用することはできません。

本ソフトウェアは、Microchip 社の”MICROCHIP USB FRAMEWORK SOFTWARE” を使用して作成されています。

# 3　　イントロダクション

UIOUSB は PIC18F2550にUSBポートを搭載したマイコンボードを汎用 I/O装置として使用するためのソフトウェア（ファームウエア）です。 PC からは、USB 仮想シリアルポート経由でコマンドを送信して、I/Oを操作することができます。



UIOUSBファームウエアを組み込んだ USB インターフェイス付きのCPU ボード

UIOUSB の主な機能は以下のものがあります。

- 8 ビットI/O ポート(RB) 出力
- 8 ビットI/O ポート(RB) 入力
- 4 チャンネル 10ビットA/D 変換入力 (RA0,1,2,3)
- 2 チャンネル PWM信号出力 (CCP1,CCP2)
- 8 チャンネル 簡易サーボ信号出力(RB)
- 8 ビットI/O ポート(RB) 入出力切り替え設定
- 入力ポートのプルアップ切り替え設定
- 設定値をPIC内部 EEPROM に保存（デバイスリセット時に自動読み込み）
- I/O ポート入力変化時にイベントデータをシリアルポートに出力
- 予め設定した カウンタ値を越えて、I/O ポート入力値が変化した場合に、イベントデータをシリアルポートに出力
- 予め設定したA/D変化許容値を越えて、A/Dポート入力値が変化した場合に、イベントデータをシリアルポートに出力
- 予め設定した間隔で、定期的に I/O ポート値、A/D 変換値、カウンタ値(入力ポート変化数)をシリアルポートに出力(自動サンプリング)
- コマンドから任意のタイミングで、I/O ポート値、A/D 変換値、カウンタ値(入力ポート変化数)をシリアルポートに出力(手動サンプリング)
- シグナル(LED)の点滅・点灯、ブザー出力(２種類のパターン)をコマンドで設定、またイベントデータ送信条件になった場合にシグナルを同時に設定することも可能

**⚠ 注意**

上記の幾つかの機能は、同一のポートを使用していますので同時に使用することはできません。例えば I/O ポート

の下位4 ビットを出力で使用したときは、入力で使用可能なポートは上位 4 bitになります。

これらのコマンドを、手動もしくはプログラムから実行することで簡単に I/O 制御を実現することができます。
UIOUSB デバイスを利用して、下記のようなアプリケーションを作成することができます。オールブルーシステムの ABS-9000 DeviceServer と組み合わせるとでシステムを簡単に構築できます。UIOUSB デバイス単体でもシリアルポート経由のコマンドを操作するためのプログラムを作成してシステムを構築することが可能です。

- **パトライトにアラームや警報を出力する**

UIOUSB デバイスのシグナルポート出力(LED,ブザー) にドライバまたはリレー経由で市販のパトライトや警報器を接続できます。シグナルポートでは LED の点灯・消灯、点滅,ブザーは２種類の出力パターンをコマンド経由で指定できますので、簡単に PC に接続したアラーム装置を作成することができます。
これを応用して、アプリケーションプログラムの異常やメール着信などのイベントを外部に通知できます。
シグナルポートに出力設定した一定時間経過後に、自動的に LEDとブザー出力を停止するように設定することもできます。

- **フットスイッチでアプリケーションを操作する**

市販のフットスイッチ等を UIOUSB の I/O ポートに接続できます。フットスイッチの ON,OFF のイベントを検出して、外部のプログラムに UDP パケット経由でイベントデータを送信できます。既存のアプリケーションの機能に、UDP パケットの受信部分を追加するだけで、キーやマウス操作が難しい現場でのアプリケーションの使い勝手を向上させることができます。
UIOUSB はファームウエアでチャタリング抑止や入力変化回数によるイベント送信機能がありますので、各種インターフェイスデバイスの接続にも適しています。また A/D 変換機能もありますので様々なアナログ入力にも対応できます。

- **外出先からWebや電子メールで電化製品の ON/OFF を操作する**

UIOUSB デバイスの I/O ポートにリレー等のドライバを接続して商用電源の ON/OFF を操作することが出来ます。
UIOUSB を ABS-9000 DeviceServer に接続すると、外出先の携帯電話やパソコンから電子メールでログイン認証を行なって、予め作成したスクリプトを実行することでUIOUSB をリモートコントロールできます。
現在のポート状態を読み込んで、自動でメール送信することも簡単にできます。

- **各種センサーを接続して在宅監視システムを構築**

UIOUSB デバイスにリードセンサーを接続して窓やドアの開閉の検知ができます。また 赤外線センサー（人感センサー）をI/O ポートのカウンタ入力に接続して、人や動物などの動きを検出してメールを送信したり、アラーム装置に出力できます。UIOUSB のカウンタ入力機能には、閾値の設定や連続してイベントを検出するための機能が予め入っていますので、監視システムが簡単に構築できます。

- **一定間隔でセンサーのサンプリングを行って集計**

UIOUSB デバイスに温度・湿度・照度センサーを接続して、それらのA/D 変換値を、予め設定したサンプリング間隔でサーバーに送信してデータ収集を行なうシステムを構築できます。サンプリング間隔毎に UIOUSB からイベント送

信が行われますので、サーバー側のシステムではポーリングやタイマー監視等の仕組みは必要ありません。

また、ABS-9000 DeviceServer ではデータベースへの集計データの格納や、データ統計のライブラリが用意されていますので、センサーシステム構築が簡単にできます。

# 4 UIOUSB 動作仕様

UIOUSB プログラムは下記の動作環境で機能するように設計されています。

- PIC18F2550 CPUボード(USBポート付き)

  株式会社 秋月電子通商 の“ＵＳＢマイコンボード 完成品（ＰＩＣ１８Ｆ２５５０）” (2008/5/1 入手）で動作確認しています。但しこれは同一製品の全てのボードに対しての動作保障や、将来に仕様変更が行われたとき等の動作保証を行うものではありません。

- 仮想シリアルポート経由でコマンド操作を行う側のコンピュータ

  Windows XP(32bit) SP2/SP3

  動作させるコンピュータに、USB 2.0 ポートがあること

**⚠ 注意**

本ソフトウェア(UIOUSB)はオールブルーシステム独自に開発・提供していますので、本ソフトウェアに関する質問等を ”株式会社　秋月電子通商様” に行わないで下さい。

## 4.1 マイクロコントローラ PIC18F2550(USB I/F 付)

- 主なUIOUSB機能の PIC18F2550ピン配置


- PIC18F2550 の各機能アサイン

| | 設定値 |
|---|---|
| CPU 外部クロック | 20MHz(PLL 動作で内部は 48Mhz) |

| | | | | | | |
|---|---|---|---|---|---|---|
| 電源 | | USBバスから供給(5V) | | | | |
| Flash memory | | USBスタック、UIOUSBファームウエア格納用 | | | | |
| EEPROM | | UIOUSBコンフィギュレーション保存用 | | | | |
| PORTA | AN0 | A/D 変換入力 ad0 | | | | |
| | AN1 | A/D 変換入力 ad1 | | | | |
| | AN2 | A/D 変換入力 ad2 | | | | |
| | AN3 | A/D 変換入力 ad3 | | | | |
| | RA4 | シグナル LED I/O 出力 | | | | |
| | RA5 | シグナルブザー I/O 出力 | | | | |
| PORTB | | I/O 入出力 | pullup | servo出力 | change_detect検出 | カウンタ入力 |
| | RB0 | ○ | ○ | ○ | ○ | × |
| | RB1 | ○ | ○ | ○ | ○ | × |
| | RB2 | ○ | ○ | ○ | ○ | × |
| | RB3 | ○ | ○ | ○ | ○ | × |
| | RB4 | ○ | ○ | ○ | ○ | ○ |
| | RB5 | ○ | ○ | ○ | ○ | ○ |
| | RB6 | ○ | ○ | ○ | ○ | ○ |
| | RB7 | ○ | ○ | ○ | ○ | ○ |
| PORTC | RC1(CCP2) | PWM2 出力 | | | | |
| | RC2(CCP1) | PWM1 出力 | | | | |

## 4.2 ファームウエア書き込み

書き込みには PIC18F2550 CPU ボードに対応した書き込み装置が必要になります。このマニュアルでは “株式会社秋月電子通商の ＡＫＩ－ＰＩＣプログラマー Ｖｅｒ．４”を使用した書き込み手順を示します。 書き込みソフトの詳しい操作方法やPC との接続方法などについてはPICプログラマーのマニュアルを参照してください。

ファームウエアとドライバをダウンロードして適当なフォルダに展開します。以下のファイルがそろっているかを確認してください。

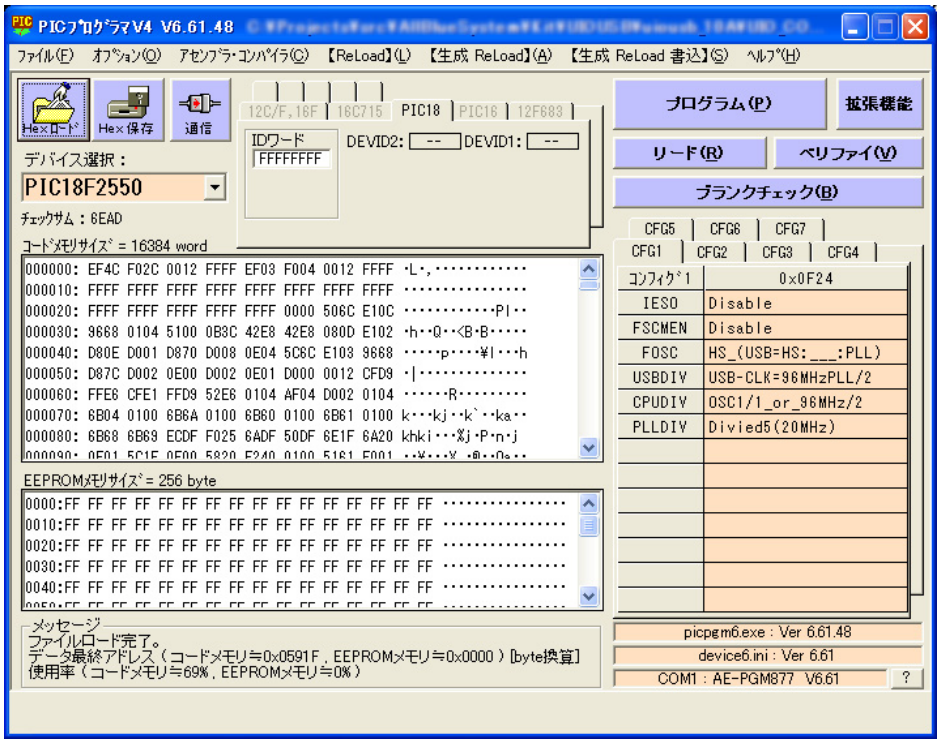
| | |
|---|---|
| ファームウエア本体 | UIO_COMMAND_USB_2550.hex |
| デバイスドライバ<br>(Microchip 社製 CDC ドライバ) | inf_uioudb¥win2k_winxp_winvista32¥mchpcdc.inf |

## 4.3 書き込み時の CONFIG 設定値

PICプログラマを起動して、ファームウエアの HEX ファイル(UIO_COMMAND_USB_2550.hex)をロードします。
HEX ファイル中には下記のコンフィギュレーション値が予め書き込まれていますので、PIC プログラマ側での変更は通常必要ありませんが、念のために設定値を確認してから書き込み操作を行って下さい。設定値が間違っているとUIOUSB は正常に動作しませんので注意してください。

| UIOUSB コンフィグ設定値 | | |
|---|---|---|
| 項目 | Name | 設定値 |
| Internal/External Oscillator Switchover bit | IESO | Disable |
| Fail-Safe Clock Monitor Enable bit | FSCMEN | Disable |
| Clock Mode | FOSC | HS_PLL |
| USB Clock Selection bit | USBDIV | 96MHzPLL/2 |
| System Clock Postscaler Selection bit | CPUDIV | OSC1/1_or_96MHz/2 |
| PLL Prescaler Selection bits | PLLDIV | Divied5(20MHz) |
| USB Internal Voltage Regulator Enable bit | VREGEN | Enable |
| Brown-out Reset Voltage bits | BORV | 2.7V |
| Brown-out Reset Enable bits | BOREN | Enable(H/W) |
| Power-up Timer Enable | PWRTEN | Enable |
| Watchdog Timer Enable bit | WDTEN | Disable |
| MCLR Pin Enable bit | MCLRE | Enable |
| CCP2 MUX bit | CCP2MX | CCP2=RC1 |
| Single-Supply ICSP Enable bit | LVP | Disable |
| Stack Full/Underflow Reset Enable bit | STVR | Enable |

下記は、PIC ライターでUIOUSB ファームエアの書き込みを行っている画面です。




PIC ライターで書き込みを行ってベェリファイが正常に完了すれば、UIOUSB デバイスの準備は完了です。

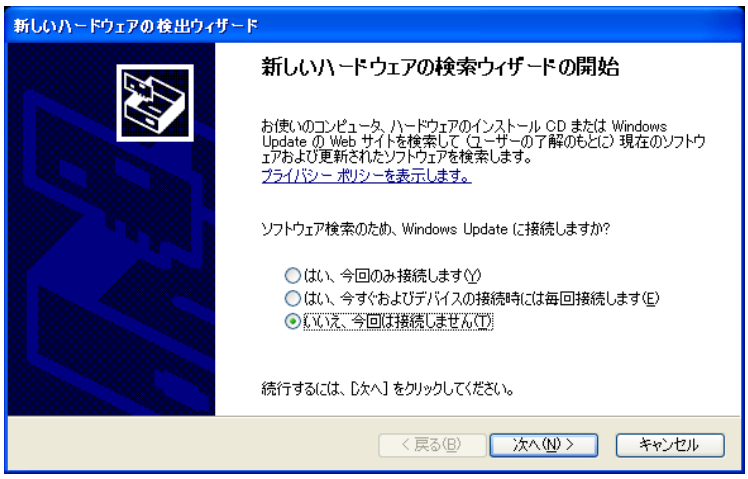
# 5　　インストール(Windows PC に接続)

UIOUSB ファームウエアを書き込んだ PIC2550 CPU ボードをUSBケーブルで PC に接続します。

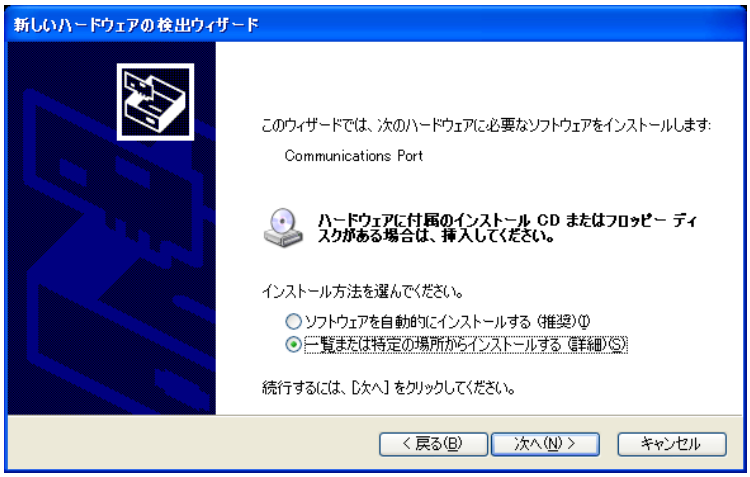
この時 CPUボードの電源をUSBから供給するか、別電源を供給するかの切り替えをボード上のジャンパで正しく設定してください。

始めて UIOUSB デバイスをPC に接続すると、ハードウエアの検出画面が開きますのでドライバの設定を行います。

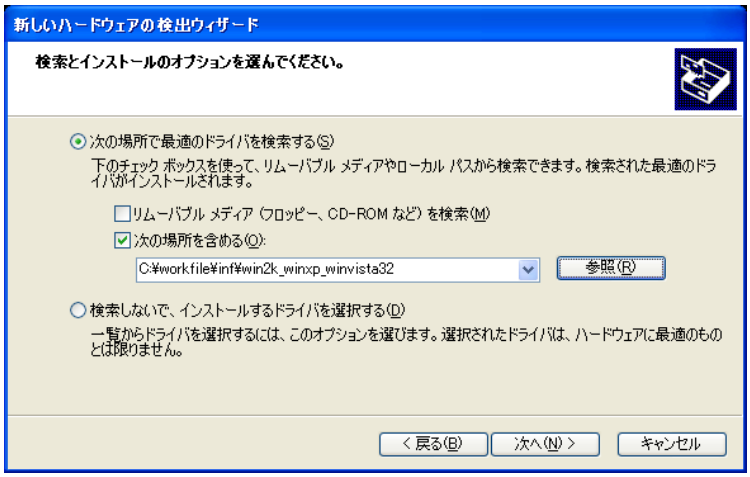
以降の画面の説明は Windows XP でのデバイスドライバインストールの操作例を示します。



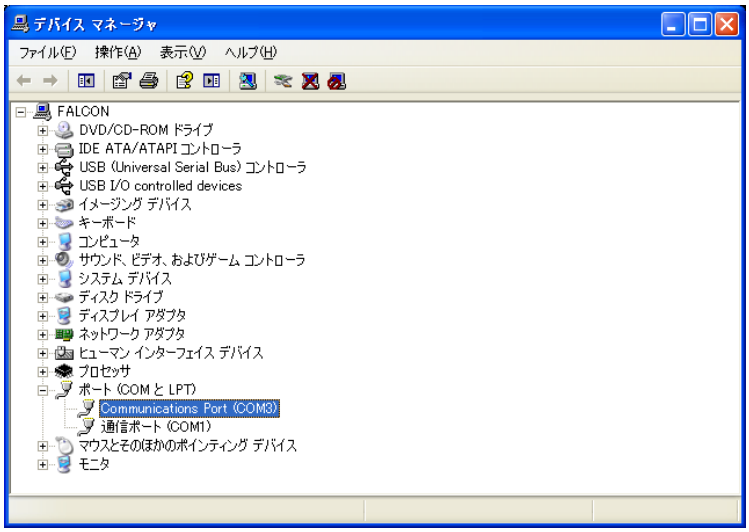

ドライバをローカルディスクから指定しますので Windows Update には接続しません。




ダウンロード済みの Inf ファイルを指定しますので特定の場所からインストールするを選択します。




ダウンロードした Microchip 社製 CDC ドライバ の保管してあるフォルダを指定します。




ドライバがインストールされるとデバイスマネージャでシリアルポートとして認識されているのが確認できます。

ドライバが一度インストールされると、次回からは UIOUSB は自動で認識されます。

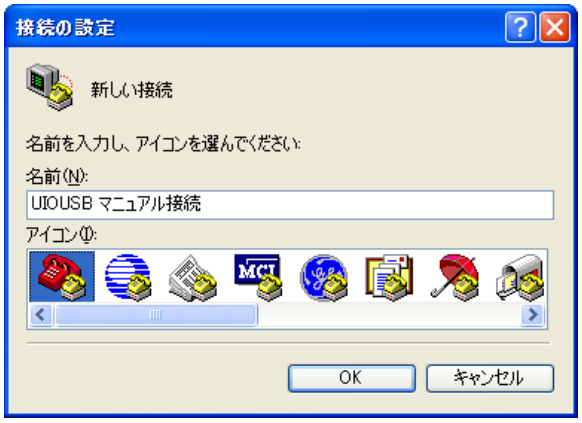
デバイスマネージャ上でアサインされた COM ポート名（画面の例では COM3）をメモしておいてください。シリアルポート経由でUIOUSB のコマンド操作を手動で操作する場合と、DeviceServer でUIOUSB を使用するときにここで確認したシリアル名が必要になります。

ドライバのインストールが完了しても、デバイスが認識されないときには 一度 USB ケーブルを抜いてからもう一度接続することで正常に認識される場合があります。

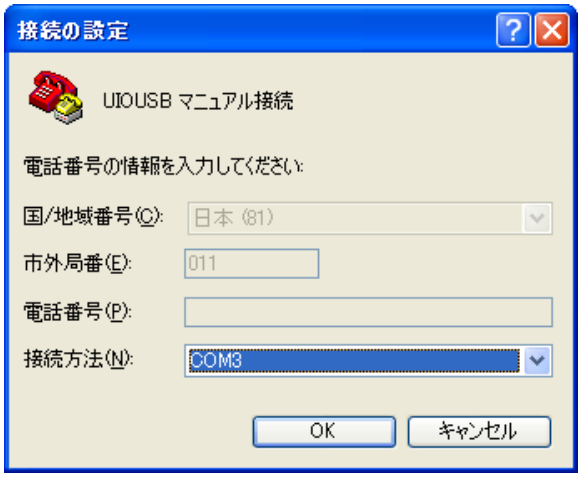
# 6 動作確認

ハイパーターミナルプログラムを使用して動作確認を行います。アクセサリからハイパーターミナルを選択して起動します。ハイパーターミナル以外でもシリアルポートに接続可能なソフトウェアならば動作確認を行うことができます。

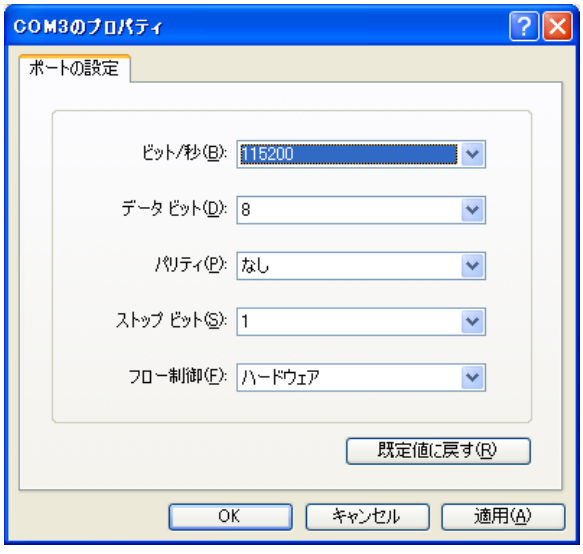
作業の途中でソフトを終了した場合などに、UIOUSB のCOMポートが使用中で開けなくなる場合があります。このときは、一度 USB ケーブルを抜いて再度デバイスを認識させるとうまくつながる場合があります。




ハイパーターミナルを起動して、接続名に適当なものを入力します。



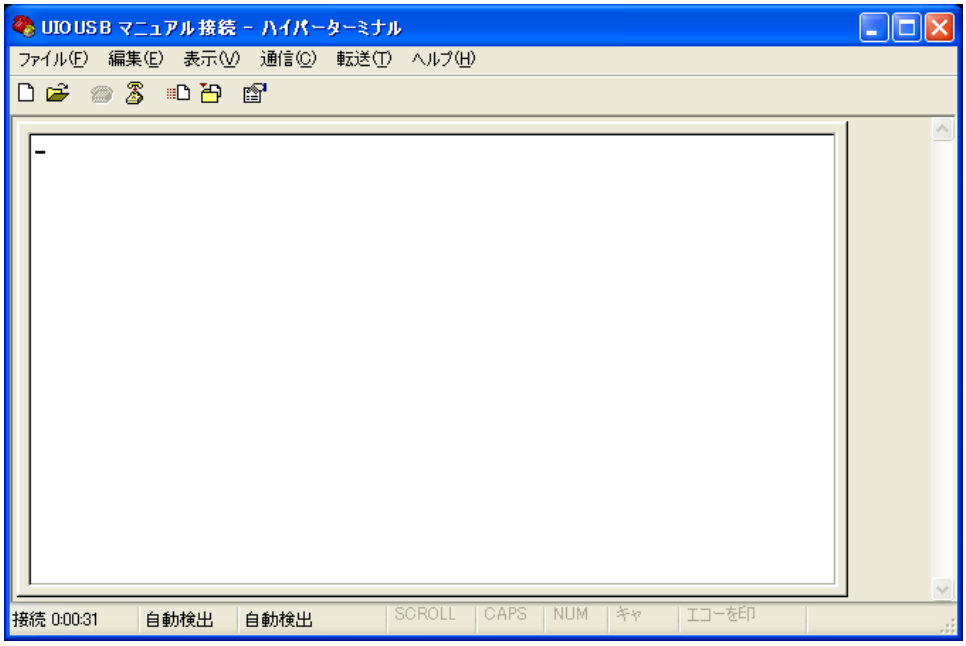

デバイスマネージャで確認した UIOUSB が接続されたCOM ポートを指定します。



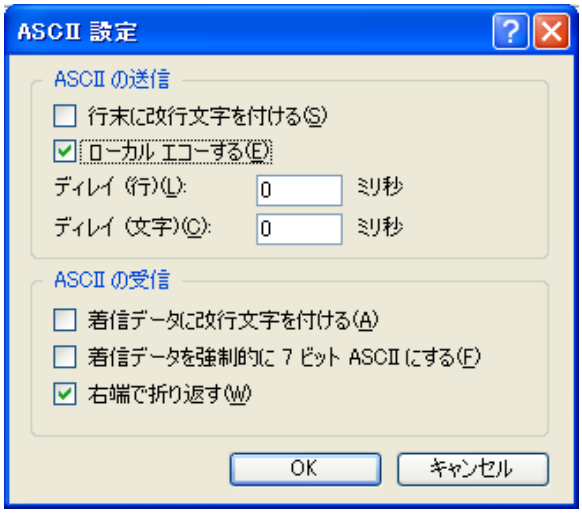

ポートの設定は適当な値で構いませんので入力します。

ポートに接続するとターミナル画面が表示されます。

UIOUSB は入力文字をデバイスからエコーバックを行いません。ターミナルからコマンド文字列を入力しても画面にはエコーバック表示されずにUIOUSB のレスポンスのみが表示されます。

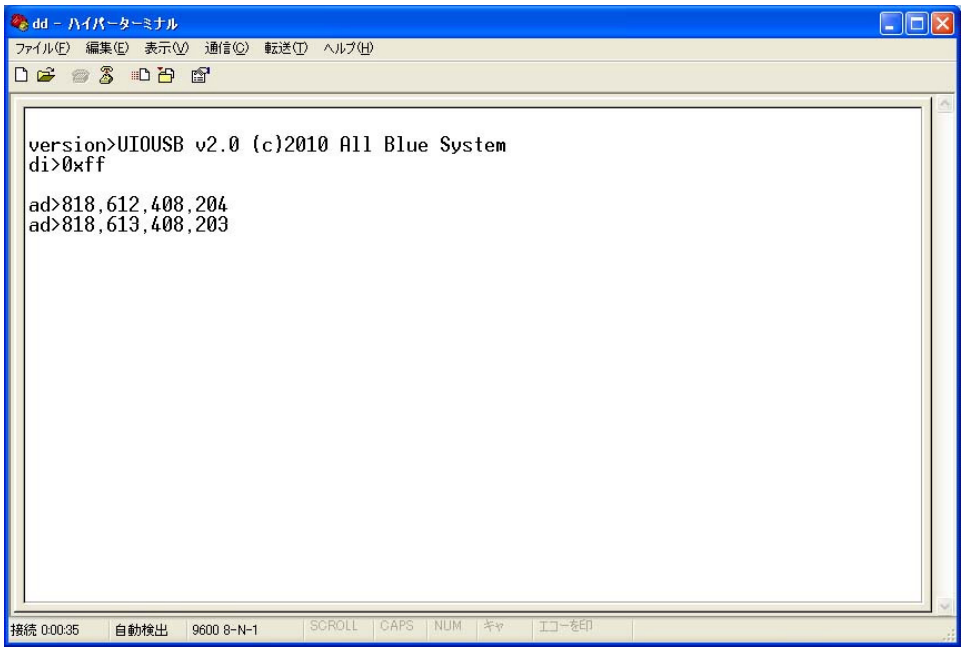



プログラム等からコマンドを実行する場合を除いて、手動でシリアルターミナルからコマンドを実行する場合には、入力間違いを防ぐために、ハイパーターミナルの設定を変更してローカルエコーを有効にします。




ファイルメニューの中のプロパティを選択して、ASCII 設定画面を開いてローカルエコーを有効にします。

この状態で全ての UIOUSB コマンドが実行できますので、 version<Enter> を入力すると画面に UIOUSB のバージョン番号が表示されます。




# 7 UIOUSB コマンドリファレンス

## 7.1 シリアルポート経由のコマンド操作について

UIOUSB のコマンド入力とレスポンスデータを USB 仮想COM ポート経由で操作する場合の説明をします。

- 文字コードセット

  UIOUSB で使用できる文字は ASCII と数字のみで日本語等の文字を使用することはできません。

  コマンド入力のときのスペース文字はASCII スペース(0x20) のみが有効で全角スペースはエラーになりますので注意してください。

- コマンドとパラメータの区切り

  UIOUSB コマンドにパラメータを指定する場合には、必ず間にスペース文字(0x20)を1つ入れて下さい。

- 入力文字エコーバック

  UIOUSB は入力文字のエコーバックは行いません。手動で操作する場合にはターミナルソフト側のローカルエコー機能などを使用してください。

- コマンドのデリミタ

  １つのコマンドはキャリッジリターン (0x0d) をコマンドデリミタ文字として認識します。複数のコマンドを連続して送信する場合には、各々のコマンドの終端にキャリッジリターン付けてください。

- レスポンス文字列の表示

  UIOUSB からのレスポンス文字列は終端にキャリッジリターン(0x0d)とラインフィード(0x0a)の２文字が追加

されて返されます。

レスポンス文字列のフォーマットは必ず “入力コマンド文字列” + ‘>’ で始まり、続いてUIOUSB のレスポンス文字、最後にキャリッジリターン、ラインフィード文字になります。

エラーの場合はUIOUSBレスポンス文字列の先頭に “*ERROR*” が付きます。その後にエラーメッセージが続きます。

- ヌルコマンドの応答

  有効なコマンドをなにも入れないで、単にキャリッジコードを送信した場合は、レスポンス文字列にはキャリッジリターン、ラインフィード文字のみが返されます。

- 連続してコマンドを実行する場合の、レスポンス応答待ち(重要)

  プログラムから UIOUSB を制御する場合は、入力コマンドとキャリッジリターンを送信した後、必ず ‘>’ 文字を含むレスポンス文字列とキャリッジリターンとラインフィード文字までを確実に受信してから、次の入力コマンドを送信してください。

  レスポンス文字列には ‘>’文字が必ず1つだけ含まれますので、この文字が含まれた文字列が UIOUSB デバイスから送信されてきた場合には、直近のコマンド処理が終了してその実行結果が ’>’ 文字以降に入っています。“*ERROR*” の文字列がレスポンス文字列に含まれている場合には、実行結果がエラーになったことを示します。

- イベント文字列の受信

  UIOUSB デバイスで自動サンプリングやカウンタイベント等の使用する設定を行った場合には、イベント発生条件になると、UIOUSB デバイスからは “$$$” で始まるイベント文字列が送信されます。イベント発生のタイミングが、コマンド入力とそのレスポンス出力の間になる場合もあります。イベントについての詳しい説明は “UIOUSB イベントリファレンス”の章を参照してください。

## 7.2 ad

- **機能概要**

  全てのA/D 入力チャンネルの値を返す。

- **リクエストフォーマット**

| command | ad |
|---|---|

- **リプライデータフォーマット**

| reply | ad>[ad0],[ad1],[ad2],[ad3] |
|---|---|

- **パラメータとリターン値**

| | |
|---|---|
| [ad0] | ad0 A/D入力変換値。10進数で 0 から 1023 までの整数が入る。 |
| [ad1] | ad1 A/D入力変換値。10進数で 0 から 1023 までの整数が入る。 |
| [ad2] | ad2 A/D入力変換値。10進数で 0 から 1023 までの整数が入る。 |
| [ad3] | ad3 A/D入力変換値。10進数で 0 から 1023 までの整数が入る。 |

- **備考**

- **実行例 “リクエスト文字列” -> “リプライ文字列”**

| |
|---|
| “ad” -> “ad>671,374,845,324” |

## 7.3 ad(ch#)

- **機能概要**

指定したA/D 入力チャンネルの値を返す。

- **リクエストフォーマット**

| | |
|---|---|
| command | ad0 |
| | ad1 |
| | ad2 |
| | ad3 |

- **リプライデータフォーマット**

| | |
|---|---|
| reply | ad0>[0-1023] |
| | ad1>[0-1023] |
| | ad2>[0-1023] |
| | ad3>[0-1023] |

- **パラメータとリターン値**

| | |
|---|---|
| [0-1023] | A/D入力変換値。10進数で 0 から 1023 までの整数が入る。 |

- **備考**

- **実行例 “リクエスト文字列” -> “リプライ文字列”**

| |
|---|
| “ad0” -> “ad0>671” |

| |
|---|
| “ad1” -> “ad0>374” |
| “ad2” -> “ad0>875” |
| “ad3” -> “ad0>323” |
| “ad4>” -> “ad4>*ERROR* unknown command |

## 7.4 di

- **機能概要**

  I/O ポート(PORTB) の現在値を返す。

- **リクエストフォーマット**

| | |
|---|---|
| command | di |

- **リプライデータフォーマット**

| | |
|---|---|
| reply | di>[0x00-0xff] |

- **パラメータとリターン値**

  [x00-0xff]　　PORTB の現在の値。16進数で 0x00 から 0xffまでの値が入る。

- **備考**

  I/O ポートの各ビットが入力・出力どちらの設定でも、現在の値が返されます。

- **実行例 “リクエスト文字列” -> “リプライ文字列”**

| |
|---|
| “di” -> “di>0xff” |
| “do 0x12” -> “do 0x12>0x12” |
| “di” -> “di>0x12” |

## 7.5 di(bit#)

- **機能概要**

  I/O ポート(PORTB) の指定ビットの現在値を返す。

- **リクエストフォーマット**

| | |
|---|---|
| command | di0<br>di1 |

| | |
|---|---|
| | di2 |
| | di3 |
| | di4 |
| | di5 |
| | di6 |
| | di7 |

- **リプライデータフォーマット**

| | |
|---|---|
| reply | di0>[0\|1] |
| | di1>[0\|1] |
| | di2>[0\|1] |
| | di3>[0\|1] |
| | di4>[0\|1] |
| | di5>[0\|1] |
| | di6>[0\|1] |
| | di7>[0\|1] |

- **パラメータとリターン値**

[0|1]　　I/O ポート(PORTB)中の指定したビットの現在の値。'0' または '1'

- **備考**

I/O ポートの各ビットが入力・出力どちらの設定でも、現在の値が返されます。

I/O ポートの指定ビットの値がHigh の場合は '1'、Lowの場合は '0' が返ります。

- **実行例 "リクエスト文字列" -> "リプライ文字列"**

| |
|---|
| "do 0x12" -> "do 0x12>0x12" |
| "do0" -> "di0>0 |
| "di1" -> "di1>1" |
| "di8" -> "di8>*ERROR* unknown command" |

## 7.6　　do

- **機能概要**

I/O ポートに指定した値を出力。

- **リクエストフォーマット**

| command | do [0-255\|0x00-0xff] |
|---|---|

- **リプライデータフォーマット**

| reply | do [0-255\|0x00-0xff]>[0x00-0xff] |
|---|---|

- **パラメータとリターン値**

[0-255|0x00-0xff]　　PORTB に出力する値。10進数または16進数の値を指定する。
16進数で指定する場合には “0x” が先頭に必要。

[0x00-0xff]　　PORTB の変更後の値。16進数で 0x00 から 0xffまでの値が入る。

- **備考**

8 bit I/O Port(RB)に指定した値を出力します。

値は16進表記もしくは10進表記で指定します。ポートの値をHigh にしたい場合は対応するビットに 1 を指定し、Lowの場合は 0 を指定します。

- **実行例 “リクエスト文字列” -> “リプライ文字列”**

“do 0x0” -> “do 0x0>0x00”
“do 255” -> “do 255>0xff”

## 7.7　　do(bit#)

- **機能概要**

I/O ポート(PORTB) の指定ビットの値を設定する。

- **リクエストフォーマット**

| command | do0 [0\|1] |
|---|---|
| | do1 [0\|1] |
| | do2 [0\|1] |
| | do3 [0\|1] |
| | do4 [0\|1] |
| | do5 [0\|1] |
| | do6 [0\|1] |
| | do7 [0\|1] |

- **リプライデータフォーマット**

| reply | do0 [0\|1]>[0\|1] |
|---|---|
| | do1 [0\|1]>[0\|1] |
| | do2 [0\|1]>[0\|1] |
| | do3 [0\|1]>[0\|1] |
| | do4 [0\|1]>[0\|1] |
| | do5 [0\|1]>[0\|1] |
| | do6 [0\|1]>[0\|1] |
| | do7 [0\|1]>[0\|1] |

- **パラメータとリターン値**

  [0|1]　　PORTB の指定したビットに設定する値。リプライの場合は現在の値を示す。

  '0' または '1'

- **備考**

  I/O ポート中の指定したビットの値を High または Low に設定します。

  ポート中の指定したビットの値をHigh にする場合は1 を指定し、Lowの場合は 0 を指定します。

- **実行例 "リクエスト文字列" -> "リプライ文字列"**

| "do0 1" ->1 "do0 1>1" |
|---|
| "do7 1" -> "do7 1>1" |
| "di" -> "di>0x81" |
| "do8 1" -> "do8 1>*ERROR* unknown command" |

## 7.8　dcfg

- **機能概要**

  I/O ポートの各ビットの入出力モードの設定または現在の設定値の取得。

  "save" コマンドによる EEPROM への設定値保存対象

- **リクエストフォーマット**

| command | dcfg |
|---|---|
| | dcfg [0-255\|0x00-0xff] |

- **リプライデータフォーマット**

| reply | dcfg>[0x00-0xff] |
|---|---|
| | dcfg [0-255\|0x00-0xff]>[0x00-0xff] |

- **パラメータとリターン値**

| | |
|---|---|
| [0-255\|0x00-0xff] | PORTB の入出力モードを設定する。10進数または16進数の値を指定する。<br>16進数で指定する場合には “0x” が先頭に必要。<br>ポートの各ビットに対応するビット位置の値が 0 で出力、1 で入力になる。 |
| [0x00-0xff] | リプライ中の現在の設定値を示す。16進数で 0x00 から 0xffまでの値が入る。 |

- **備考**

パラメータを指定しない場合は、現在のI/O ポートの入出力モードを表示します。

パラメータを指定する場合は、I/O ポートの入出力モードを設定します。入力モードにする場合は、対応するビットを 1 、出力の場合は 0 に設定します。

入力に設定した場合は、CPU内部のプルアップ設定を使用するかどうかの指定を、使用環境に合った適切な値に設定して下さい。(“pullup” コマンド参照)

リセット直後の dcfg の初期値は 0xff で全ビットが入力モードになっています。“save” コマンドで以前に設定値を保存していた場合は、リセット後に保存されていた設定値がロードされます。

- **実行例 “リクエスト文字列” -> “リプライ文字列”**

| |
|---|
| “dcfg” -> “dcfg>0xff” |
| “dcfg 0x00” -> “dcfg 0x00>0x00” |
| “dcfg” -> “dcfg>0xff” |

## 7.9 pullup

- **機能概要**

CPU内部のI/O ポートプルアップ設定を指定または表示する。

“save” コマンドによる EEPROM への設定値保存対象

- **リクエストフォーマット**

| | |
|---|---|
| **command** | pullup<br>pullup [1\|0] |

- **リプライデータフォーマット**

| | |
|---|---|
| **reply** | pullup>[1\|0]<br>pullup [1\|0]>[0\|1] |

- **パラメータとリターン値**

  [0|1]　　プルアップを使用する場合は 1 に設定する。使用しない場合は 0 に設定する。
  リプライの場合は現在の値を示す。

- **備考**

  パラメータを指定しない場合は、現在のプルアップ設定値を表示します。

  リセット直後の pullup の初期値は '0' になっています。"save" コマンドで以前に設定値を保存していた場合は、リセット後に保存されていた設定値がロードされます。

- **実行例 "リクエスト文字列" -> "リプライ文字列"**

| |
|---|
| "pullup 1" -> "pullup 1>1" |
| "pullup 0" -> "pullup 0>0 |

## 7.10　save

- **機能概要**

  現在の設定値を CPU 内部のEEPROM に保存

- **リクエストフォーマット**

| command | save |
|---|---|

- **リプライデータフォーマット**

| reply | save>ok |
|---|---|

- **パラメータとリターン値**

  [ok]　　保存に成功した場合に ok が返答される。
  エラーの場合には "*ERROR*" で始まるエラーメッセージが返る

- **備考**

  保存対象となる設定項目については、各コマンドの機能概要を参照してください。

  EEPROM に保存した設定値は、リセット時に自動的に EEPROM から読み込まれてコンフィギュレーションが設定されます

- **実行例 "リクエスト文字列" -> "リプライ文字列"**

| |
|---|
| “save” -> “save>ok” |

## 7.11 clear_eeprom

- **機能概要**

  CPU 内部のEEPROM に保存されている設定値を消去する

- **リクエストフォーマット**

| | |
|---|---|
| **command** | clear_eeprom |

- **リプライデータフォーマット**

| | |
|---|---|
| **reply** | clear_eeprom>ok |

- **パラメータとリターン値**

  [ok]　　消去に成功した場合に ok が返答される。

  エラーの場合には “*ERROR*” で始まるエラーメッセージが返る

- **備考**

  このコマンド実行後、デバイスをリセット(USBケーブルを外すか、外部供給電源をOFF にする)することで、各コマンドの設定値が全て初期値に戻ります。

- **実行例 “リクエスト文字列” -> “リプライ文字列”**

| |
|---|
| “clear_eeprom” -> “clear_eeprom>ok” |

## 7.12 pwm(ch#)

- **機能概要**

  PWM 出力を有効または無効にするフラグの設定または表示。

  “save” コマンドによる EEPROM への設定値保存対象

- **リクエストフォーマット**

| | |
|---|---|
| **command** | pwm1<br>pwm2<br>pwm1 [0\|1]<br>pwm2 [0\|1] |

- **リプライデータフォーマット**

| reply | pwm1>[0|1]<br>pwm2>[0|1]<br>pwm1 [0|1]>[0|1]<br>pwm2 [0|1]>[0|1] |
|---|---|

- **パラメータとリターン値**

  [0|1]　　PWM の指定したチャンネルを有効または無効に設定する。'0' で無効 '1' で有効に設定される。リプライの場合は現在の値を示す。

- **備考**

  パラメータを指定しない場合は、現在の PWM 出力設定状態を表示します。

  PWM 信号は CPU の CCP1 と CCP2 端子より出力されます。 PWM 信号の周波数は 約 2.93KHz(固定) で duty 値は "duty" コマンドで設定した値になります。

  リセット直後の pwm1,pwm2 の初期値は 0 で無効になっています。

  "save" コマンドを実行した場合は、その時の値にリセット後初期化されます。

- **実行例 "リクエスト文字列" -> "リプライ文字列"**

| "pwm1 0" -> "pwm1 0>0"<br>"pwm2 1" -> "pwm2 1>1"<br>"pwm1" -> "pwm1>0"<br>"pwm2" -> "pwm2>1" |
|---|

## 7.13　duty(ch#)

- **機能概要**

  PWM 信号の duty 値の設定または表示。

  "save" コマンドによる EEPROM への設定値保存対象

- **リクエストフォーマット**

| command | duty1<br>duty2<br>duty1 [0-1023]<br>duty2 [0-1023] |
|---|---|

- **リプライデータフォーマット**

| reply | duty1>[0-1023] |
|---|---|
| | duty2>[0-1023] |
| | duty1 [0-1023]>[0-1023] |
| | duty2 [0-1023]>[0-1023] |

- **パラメータとリターン値**

[0-1023]　　PWM duty 値を表示する。値は 0 から 1023の間の整数値。
リプライの場合は現在の値を示す。

- **備考**

パラメータを指定しない場合は、現在の DUTY設定値を表示します。
PWM 信号を出力する時には pwm1 pwm2 コマンドで出力を有効に設定しください。

PWM 信号は CPU の CCP1 と CCP2 端子より出力されます。 PWM 信号の周波数は 約 2.93KHz(固定) で duty 値は “duty” コマンドで設定した値になります。リセット直後の duty1,duty2 の初期値は 512になっています。

“save” コマンドを実行した場合は、その時の値にリセット後初期化されます。

- **実行例 “リクエスト文字列” -> “リプライ文字列”**

| “duty1 0” -> “duty1 0>0” |
|---|
| “duty2 1023” -> “duty2 1023>1023” |
| “duty1” -> “duty1>0” |
| “duty2” -> “duty2>1023” |
| “duty1 3000” -> ”duty1 3000>1023 |

## 7.14　servo(bit#)

- **機能概要**

I/O ポート(PORTB) の指定ビットについて、servo 信号出力を有効または無効にするフラグの設定または表示。
“save” コマンドによる EEPROM への設定値保存対象

- **リクエストフォーマット**

| command | servo0 |
|---|---|
| | servo1 |
| | servo2 |

| | |
|---|---|
| | servo3 |
| | servo4 |
| | servo5 |
| | servo6 |
| | servo7 |
| | servo0 [0\|1] |
| | servo1 [0\|1] |
| | servo2 [0\|1] |
| | servo3 [0\|1] |
| | servo4 [0\|1] |
| | servo5 [0\|1] |
| | servo6 [0\|1] |
| | servo7 [0\|1] |

- **リプライデータフォーマット**

| | |
|---|---|
| **reply** | servo0>[0\|1] |
| | servo1>[0\|1] |
| | servo2>[0\|1] |
| | servo3>[0\|1] |
| | servo4>[0\|1] |
| | servo5>[0\|1] |
| | servo6>[0\|1] |
| | servo7>[0\|1] |
| | servo0 [0\|1]>[0\|1] |
| | servo1 [0\|1]>[0\|1] |
| | servo2 [0\|1]>[0\|1] |
| | servo3 [0\|1]>[0\|1] |
| | servo4 [0\|1]>[0\|1] |
| | servo5 [0\|1]>[0\|1] |
| | servo6 [0\|1]>[0\|1] |
| | servo7 [0\|1]>[0\|1] |

- **パラメータとリターン値**

[0|1]　　PORTB の指定したビットの servo信号出力を有効または無効に設定する。

'0' で無効 '1' で有効に設定される。

リプライの場合は現在の値を示す。

- **備考**

パラメータを指定しない場合は、現在の servo 信号出力設定状態を表示します。有効の場合は 1 が返り、無効の場合は 0 が返ります。パラメータを指定する場合は、I/O ポート(PORTB) の対応するビットに servo 信号を出力するかどうかを設定します。1 で有効に設定され、0 で無効に設定されます。

Servo 信号を有効にする場合は、予め対応するポートのビットを出力モードにする必要があります。“dcfg” コマンドを参照してください。

Servo信号を出力中に、”do” コマンドや “do(bit#)” コマンドを使用して該当ビットの値を更新すると、予期しないservo 信号となってサーボが誤動作する可能性がありますので注意してください。

出力されるservo 信号の周期は約 2.5ms で固定です。パルス幅（サーボ位置）は “pos” コマンドで設定した値になります。

リセット直後の servo(bit#) の初期値は 0 で無効に設定されています。
“save” コマンドを実行した場合は、その時の値にリセット後初期化されます。

- **実行例 “リクエスト文字列” -> “リプライ文字列”**

| |
|---|
| “servo0 1” -> “servo0 1>1” |
| “servo0” -> “servo0>1” |

## 7.15 pos(bit#)

- **機能概要**

I/O ポート(PORTB) の指定ビットについて、servo 信号のパルス幅(サーボ位置)の設定または表示。
“save” コマンドによる EEPROM への設定値保存対象

- **リクエストフォーマット**

| | |
|---|---|
| **command** | pos0 |
| | pos1 |
| | pos2 |
| | pos3 |
| | pos4 |
| | pos5 |
| | pos6 |
| | pos7 |
| | pos0 [60-240] |
| | pos1 [60-240] |
| | pos2 [60-240] |

| | |
|---|---|
| | pos3 [60-240] |
| | pos4 [60-240] |
| | pos5 [60-240] |
| | pos6 [60-240] |
| | pos7 [60-240] |

- **リプライデータフォーマット**

| | |
|---|---|
| reply | pos0>[60-240] |
| | pos1>[60-240] |
| | pos2>[60-240] |
| | pos3>[60-240] |
| | pos4>[60-240] |
| | pos5>[60-240] |
| | pos6>[60-240] |
| | pos7>[60-240] |
| | pos0 [60-240]>[60-240] |
| | pos1 [60-240]>[60-240] |
| | pos2 [60-240]>[60-240] |
| | pos3 [60-240]>[60-240] |
| | pos4 [60-240]>[60-240] |
| | pos5 [60-240]>[60-240] |
| | pos6 [60-240]>[60-240] |
| | pos7 [60-240]>[60-240] |

- **パラメータとリターン値**

[60-240]　　PORTB の指定したビットの servo信号のパルス幅(サーボ位置)を指定する。
値は 10ns 単位で 60 から 240 の間の整数値。(0.6ms から 2.4msに相当する)
リプライの場合は現在の値を示す。

- **備考**

パラメータを指定しない場合は、現在の servo信号のパルス幅を表示します。値は 10ns 単位で 60 から 240 の間の整数値。(0.6ms から 2.4ms)

パラメータを指定する場合は、servo信号のパルス幅を 10ns の整数倍で設定します。有効な値は 60 から 240 の間の整数値で、0.6ms から 2.4ms のパルス幅で出力されます。
一般的なラジコンやロボット用のサーボの中間位置は 1.5ms なので、そのときのpos コマンドの値は 150 になります。

リセット直後の pos(bit#) の初期値は 150 に設定されています。

“save” コマンドを実行した場合は、その時の値にリセット後初期化されます。

- **実行例 “リクエスト文字列” -> “リプライ文字列”**

| |
|---|
| “pos0 150” -> “pos0 150>150” |
| “pos0” -> “pos0>150” |

## 7.16 sampling_rate

- **機能概要**

自動サンプリング間隔の設定または取得

“save” コマンドによる EEPROM への設定値保存対象

- **リクエストフォーマット**

| | |
|---|---|
| **command** | sampling_rate |
| | sampling_rate [rate] |

- **リプライデータフォーマット**

| | |
|---|---|
| **reply** | sampling_rate>[rate] |
| | sampling_rate [rate]>[rate] |

- **パラメータとリターン値**

[rate] 自動サンプリング間隔(100ミリ秒単位)を指定する (0 から 2^31 までの整数)

0 を指定した場合は 自動サンプリングを行わない。

- **備考**

<rate> に 0 を設定した場合は、自動サンプリングを行いません。

<rate> は 100ms 単位で指定しますので、”1” を設定すると 100ms に一回サンプリングされてデータが送信されます。10 を設定すると1 秒毎のサンプリングになります。

リクエストコマンド [rate]パラメータを省略した場合には、現在の設定値が返されます。

設定したサンプリング間隔毎に “SAMPLING” イベントが発生して、サンプリングイベントデータを出力します。サンプリングイベントデータの内容は “イベントリファレンス” の章を参照してください。

サンプリングイベントで設定した秒間隔でイベントが発生しますが、時間精度は マイクロコントローラに接続されたクリスタルの精度に依存しています。そのため、正確なサンプリング間隔を必要とする場合には、サー

バー等から “force_sample” コマンドを使用する方法を検討してください。

リセット直後の sampling_rate の初期値は 0 に設定されています。

“save” コマンドを実行した場合は、その時の値にリセット後初期化されます。

- **実行例 “リクエスト文字列” -> “リプライ文字列”**

| |
|---|
| “sampling_rate 3600” -> “sampling_rate 3600>3600” |

## 7.17 change_detect

- **機能概要**

入出力ポート値の変化を検出するビット位置の、設定または取得

“save” コマンドによる EEPROM への設定値保存対象

- **リクエストフォーマット**

| | |
|---|---|
| **command** | change_detect<br>change_detect [0x00-0xff] |

- **リプライデータフォーマット**

| | |
|---|---|
| **reply** | change_detect>[0x00-0xff]<br>change_detect [0x00-0xff]>[0x00-0xff] |

- **パラメータとリターン値**

[0x00-0xff]　　I/Oポート(PORTB)値の変化の検出するビットを “1” 、検出を行わないビットを“0” で表した値を16進数表記で指定する。(デフォルト設定値:0x00)
リプライの場合は現在の値を示す。

- **備考**

“change_detect” では、入力ポートに加えて、出力ポートに対しても検出対象に指定することができます。

“change_detect” でポート値の変化を検出する毎に、”CHANGE_DETECT” イベントが発生して、イベントデータを送信します。”CHANGE_DETECT” イベントの詳しい説明は、”イベントリファレンス”の章を参照して下さい。

“change_detect” コマンド設定値は常に 8bit幅で保存されています。

“count_check” コマンドでポート入力のカウント機能を使用する場合には、予め “change_detect” コマンドでカウンタ入力に使用するビット位置を検出対象に指定しておく必要があります。

リクエストコマンドでパラメータを省略した場合には、現在の設定値がリプライデータに返されます。

リセット直後の change_detect の初期値は 0x00に設定されています。

“save” コマンドを実行した場合は、その時の値にリセット後初期化されます。

- **実行例 “リクエスト文字列” -> “リプライ文字列”**

| |
|---|
| “change_detect 0xff” -> “change_detect 0xff>0xff” |

## 7.18 count_check

- **機能概要**

ポート入力カウンタの制限値チェックフラグの設定または取得

“save” コマンドによる EEPROM への設定値保存対象

- **リクエストフォーマット**

| | |
|---|---|
| **command** | count_check<br>count_check [flag] |

- **リプライデータフォーマット**

| | |
|---|---|
| **reply** | count_check>[flag]<br>count_check [flag]>[flag] |

- **パラメータとリターン値**

[flag] “0” : カウンタ値チェックを行わない

“1” : 1回だけカウンタ値チェックを行う

“2” : 連続してカウンタ値チェックを行う

リプライの場合は現在の値を示す。

- **備考**

カウンタ用入力ポートに接続されたデジタル入力が変化した場合に、カウンタ値をインクリメントする設定を行います。

カウンタ入力ポートに使用するビットは、”change_detect” コマンドを使用して該当ビットを “1” に設定しておく必要があります。

入力ポートカウンタ値のチェックについては、”イベントリファレンス”の章内の “COUNT_EXCEED” イベントの

項目を参照してください。

リクエストコマンドで [flag]パラメータを省略した場合には、現在の設定値がリプライデータに返されます。

リセット直後の count_check の初期値は 0 に設定されています。

“save” コマンドを実行した場合は、その時の値にリセット後初期化されます。

- **実行例 “リクエスト文字列” -> “リプライ文字列”**

| |
|---|
| “count_check 1” -> “count_check 1>1” |
| “count_check” -> “count_check>1” |

## 7.19 count_high

- **機能概要**

ポート入力カウンタの制限値(上限値)の設定または取得

“save” コマンドによる EEPROM への設定値保存対象

- **リクエストフォーマット**

| | |
|---|---|
| **command** | count_high |
| | count_high [limit_value] |

- **リプライデータフォーマット**

| | |
|---|---|
| **reply** | count_high >[limit_value] |
| | count_high [limit_value]>[ limit_value] |

- **パラメータとリターン値**

<limit_value> ポート入力カウンタの制限値を10進数表記で指定する。(0 から 2^31 までの整数)
(デフォルト設定値:10)
リプライの場合は現在の値を示す。

- **備考**

カウンタ入力ポートの変化した数（カウンタ数）が指定された上限値を超えた場合に“COUNT_EXCEED” イベントが発生します。イベントの詳細については ”イベントリファレンス”の章を参照してください。

リセット直後の count_high の初期値は 10 に設定されています。

“save” コマンドを実行した場合は、その時の値にリセット後初期化されます。

- **実行例 “リクエスト文字列” -> “リプライ文字列”**

| |
|---|
| “count_high 200” -> “count_high 200>200” |
| “count_high” -> “count_high>200” |

## 7.20 count_reset

- **機能概要**

  I/O ポート入力カウンタ値を 0 にする。

- **リクエストフォーマット**

| | |
|---|---|
| **command** | count_reset |

- **リプライデータフォーマット**

| | |
|---|---|
| **reply** | count_reset>ok |

- **パラメータとリターン値**

- **備考**

  カウンタ入力ポートの変化した数（カウンタ数）を 0 に初期化します。

- **実行例 “リクエスト文字列” -> “リプライ文字列”**

| |
|---|
| “count_reset” -> “count_reset>ok” |

## 7.21 signal_write

- **機能概要**

  シグナルステータスの設定

- **リクエストフォーマット**

| | |
|---|---|
| **command** | signal_write [signal_state] |

- **リプライデータフォーマット**

| | |
|---|---|
| **reply** | signal_write [signal_state]>[signal_state] |

- **パラメータとリターン値**

[signal_state]　　シグナルステータス(8bit)を16進数表記で設定する。
リプライの場合は現在の値を示す。

- **備考**

シグナルステータスの各ビット毎にアサインされた機能については“シグナル出力機能”の章を参照して下さい。シグナルステータスは更新直後から有効になります。

リセット直後の シグナルステータスは 0x00 に設定されています。シグナルステータスは“save”コマンドによる保存対象ではありません。

- **実行例“リクエスト文字列”->“リプライ文字列”**

| “signal_write 0x18” -> “signal_write 0x18>0x18” |
|---|

## 7.22　signal_read

- **機能概要**

現在のシグナルステータスの取得

- **リクエストフォーマット**

| command | signal_read |
|---|---|

- **リプライデータフォーマット**

| reply | signal_read [signal_state] |
|---|---|

- **パラメータとリターン値**

<signal_state>　　現在のシグナルステータス(8bit)の内容を16進数表記で表す。

- **備考**

シグナルステータスの各ビット毎にアサインされた機能については“シグナル出力機能”の章を参照して下さい。

- **実行例“リクエスト文字列”->“リプライ文字列”**

| “signal_read” -> “signal_read>0x18” |
|---|

## 7.23 signal_bit(bit#)

- **機能概要**

  シグナルステータスの設定(ビット単位の指定)

- **リクエストフォーマット**

| | |
|---|---|
| command | signal_bit0 [0\|1] |
| | signal_bit3 [0\|1] |
| | signal_bit4 [0\|1] |
| | signal_bit7 [0\|1] |

- **リプライデータフォーマット**

| | |
|---|---|
| reply | signal_bit0 [0\|1]>[0\|1] |
| | signal_bit3 [0\|1]>[0\|1] |
| | signal_bit4 [0\|1]>[0\|1] |
| | signal_bit7 [0\|1]>[0\|1] |

- **パラメータとリターン値**

  [0|1]　　シグナルステータス中の指定したビットの値を設定する。
  リプライの場合は現在の値を示す。

- **備考**

  指定可能なシグナルステータス中のビットは 0,3,4,7 のみです。

  シグナルステータスの各ビット毎にアサインされた機能については“シグナル出力機能”の章を参照して下さい。シグナルステータスは更新直後から有効になります。

  リセット直後の シグナルステータスは 0x00 に設定されています。シグナルステータスは“save”コマンドによる保存対象ではありません。

- **実行例“リクエスト文字列”->“リプライ文字列”**

| |
|---|
| “signal_bit7 1” -> “signal_bit7 1>1” |

## 7.24 signal_timer

- **機能概要**

  シグナルの自動リセットタイマー値の設定または取得

  “save”コマンドによる EEPROM への設定値保存対象

- **リクエストフォーマット**

| command | signal_timer<br>signal_timer [timer] |
|---|---|

- **リプライデータフォーマット**

| reply | signal_timer>[timer]<br>signal_timer [timer]>[timer] |
|---|---|

- **パラメータとリターン値**

[timer]　　自動リセットまでの間隔(秒)を指定する (0 から 2^31 までの整数)
0 を指定した場合はシグナルステータスの自動リセットを行わない。
リプライの場合は現在の値を示す。

- **備考**

タイマー値(秒) を 0 以外に設定すると、シグナルステータスを更新してからタイマー値に設定した秒を超えると、自動的にシグナルステータスが 0x00 にリセットされて全てのシグナル出力が停止します。

シグナルステータスをタイマーで設定した秒以前に連続して更新した場合には、最後に更新したタイミングからタイマーはカウントします。

リセット直後の signal_timer の初期値は 0 で無効に設定されています。
“save” コマンドを実行した場合は、その時の値にリセット後初期化されます。

- **実行例 “リクエスト文字列” -> “リプライ文字列”**

| “signal_timer 200” -> “signal_timer 200>200” |
|---|

## 7.25 signal_event

- **機能概要**

イベント発生と同時にセットするシグナルステータスの設定または取得
“save” コマンドによる EEPROM への設定値保存対象

- **リクエストフォーマット**

| command | signal_event |
|---|---|

| | |
|---|---|
| | signal_event [signal_state] |

- **リプライデータフォーマット**

| | |
|---|---|
| **reply** | signal_event>[signal_state]<br>signal_event [signal_state]>[signal_state] |

- **パラメータとリターン値**

[signal_state]　　イベント発生条件になったときにセットするシグナルステータス(8bit)を16進数表記で設定する。
リプライの場合は現在の値を示す。

- **備考**

0 以外の値を設定すると、イベント送信条件になった場合にシグナルステータスを変更してアラーム出力をすることができます。

”イベントリファレンス” の章にイベントについての詳細と、どのイベントが “signal_event” の対象になるかが記述してありますの参照して下さい。

リセット直後の signal_event の初期値は 0x00 に設定されています。
“save” コマンドを実行した場合は、その時の値にリセット後初期化されます。

- **実行例 “リクエスト文字列” -> “リプライ文字列”**

| |
|---|
| “signal_event 0x18” -> “signal_event 0x18>0x18” |

## 7.26　　force_sample

- **機能概要**

手動サンプリングの実行とサンプリングデータの取得

- **リクエストフォーマット**

| | |
|---|---|
| **command** | force_sample |

- **リプライデータフォーマット**

| | |
|---|---|
| **reply** | force_sample>[port_val],[counter_val],[ad0],[ad1],[ad2],[ad3] |

- **パラメータとリターン値**

| | |
|---|---|
| [port_val] | PORTB の値。16進数で 00 から FF までの値で、先頭の"0x"部分を取り除いて大文字で表記したものが入る。 |
| [counter_val] | カウンタ入力値。10進数で 0 から 2^31 までの整数が入る。 |
| [ad0] | A/D入力チャンネル ad0 の変換値。10進数で 0 から 1023 までの整数が入る。 |
| [ad1] | A/D入力チャンネル ad1 の変換値。10進数で 0 から 1023 までの整数が入る。 |
| [ad2] | A/D入力チャンネル ad2 の変換値。10進数で 0 から 1023 までの整数が入る。 |
| [ad3] | A/D入力チャンネル ad3 の変換値。10進数で 0 から 1023 までの整数が入る。 |

- **備考**

sampling_rate コマンドで設定した定期的に発生するサンプリングイベントとは別に、直ぐにサンプリングを行ってその結果をリプライデータで取得します。
このコマンド実行時を行っても、定期的に実行されるサンプリングイベントには影響しません。

このコマンド実行時にはカウンタ値(“count_check”,”count_reset”,”count_high” コマンド参照の事) はクリアされません。

- **実行例 “リクエスト文字列” -> “リプライ文字列”**

| |
|---|
| “force_sample” -> “force_sample>FF,0,817,613,407,203” |

## 7.27 version

- **機能概要**

UIOUSB ファームウエアバージョン文字列を取得する

- **リクエストフォーマット**

| | |
|---|---|
| **command** | version |

- **リプライデータフォーマット**

| | |
|---|---|
| **reply** | version>[version_string] |

- **パラメータとリターン値**

| | |
|---|---|
| [version_string] | UIOUSB ファームエアバージョン文字列。 |

- **備考**

- **実行例 “リクエスト文字列” -> “リプライ文字列”**

| |
|---|
| “version” -> “version>UIOUSB v2.0 (c)2010 All Blue System” |

## 7.28 ad_margin(ch#)

- **機能概要**

  A/D 変換値変化(ADVAL_UPDATE)イベント検出に使用する許容値(マージン)値の設定または取得

  “save” コマンドによる EEPROM への設定値保存対象

- **リクエストフォーマット**

| | |
|---|---|
| command | ad_margin0 |
| | ad_margin1 |
| | ad_margin2 |
| | ad_margin3 |
| | ad_margin0 [margin0] |
| | ad_margin1 [margin1] |
| | ad_margin2 [margin2] |
| | ad_margin3 [margin3] |

- **リプライデータフォーマット**

| | |
|---|---|
| reply | ad_margin0>[margin0] |
| | ad_margin1>[margin1] |
| | ad_margin2>[margin2] |
| | ad_margin3>[margin3] |
| | ad_margin0 [margin0]>[margin0] |
| | ad_margin1 [margin1]>[margin1] |
| | ad_margin2 [margin2]>[margin2] |
| | ad_margin3 [margin3]>[margin3] |

- **パラメータとリターン値**

  [margin0]

  [margin1]

  [margin2]

  [margin3] 各 A/D チャンネル毎のA/D 変換値が変化したかどうかを判断するための
  マージン値。0 から 1023 までの整数を指定する
  0 を指定した場合は A/D変換値の変化を検出しない。

- **備考**

  margin(ch#)に 0 を設定した場合は、その A/D チャンネルに対するA/D 変換値変化(ADVAL_UPDATE)イベント検出を行いません。

  margin(ch#) で指定した値よりも A/D 変換値(絶対値)が変化した場合に、ADVAL_UPDATE イベントが送信されます。A/D 変換値は、各 A/D チャンネル毎に独立して 前回 ADVAL_UPDATE イベントを送信した時の A/D 変換値と比較しています。

  **⚠ 注意**

  A/D 変換値の変化範囲が予め予測できない状態で margin(ch#) の値を設定するときは、最初は大きめの値を設定してください。その後、 “ADVAL_UPDATE” イベント発生が目的とする頻度に達するまで徐々に margin(ch#) 値を小さくしていきます。最初から margin(ch#) に小さい値を指定すると、イベントが連続発生してサーバー側のイベントハンドラ処理が間に合わなくなり、エラーが発生する場合があります。

  リクエストコマンド margin(ch#)パラメータを省略した場合には、現在の設定値が返されます。

  A/D 変換値変化(ADVAL_UPDATE)イベントデータの内容は “イベントリファレンス” の章を参照してください。

  リセット直後の ad_margin(#ch) の初期値は 0 に設定されています。
  “save” コマンドを実行した場合は、その時の値にリセット後初期化されます。

- **実行例 “リクエスト文字列” -> “リプライ文字列”**

  “ad_margin0 3” -> “ad_margin0 3>3”

# 8 UIOUSB イベントリファレンス

UIOUSB では 予め決められた I/O ポート値が変化した場合や、自動サンプリングを設定した場合にイベントデータを出力する機能があります。これによって、監視システムに応用する時のサーバーとリモートデバイス間のポーリング動作を最低限にすることで、データ通信量を減らすことができます。

イベント発生条件は UIOUSB コマンドで操作・設定可能ですので、システムの運用中に動作条件をダイナミックに変更可能です。

**⚠ 注意**

UIOUSB 内部では、10ms 間隔 “interval” でイベント発生条件をチェックして、条件に一致した場合にイベントデー

タを送信します。同一 “interval” 内で複数のイベントが発生する条件になっている場合には、後のイベントデータ発生条件のチェックを、次の “interval” まで遅延させて実行します。

**⚠ 注意**

イベントデータは、UIOUSB コマンド実行中でも送信される場合があります。このため、リクエストコマンド送信からリプライ文字列を受信する間にイベントデータを複数受信する場合があります。この場合でもリプライ文字列の途中(改行文字の前)にイベントデータが入り込むことはありません。

## 8.1 CHANGE_DETECT イベント

- **イベント発生条件**

I/O ポートの値が変化した場合に発生します。

監視対象のポートのビット位置は “change_detect” コマンドで設定します。

- **イベントデータフォーマット**

```
$$$,CHANGE_DETECT,[diff_bits],[port_val]
```

[diff_bits] には、変化したビットを 1、変化していないビットを 0にした値が、16進数で 00 から FF までの値で、先頭の”0x”部分を取り除いて大文字で表記したものが入ります。

[port_val] には 現在のポート値が16進数で 00 から FF までの値で、先頭の”0x”部分を取り除いて大文字で表記したものが入ります。

- イベント関連コマンド(コマンドの詳細は “TDCPコマンドリファレンス” の章を参照してください)

“change_detect” 検出対象のポートとそのビットを指定する

- **備考**

**“signal_event” で設定したシグナル更新の対象イベントです。**

イベントの発生条件になる毎に、このイベントが発生してイベントデータが送信されます。

ポート値は、10ms間隔で値を常に取得して、データ値が連続して同一の値を示した時に新しいポート値として内部に保存しています。この保存するポート値が前回保存していたポート値と比べて、変化していた場合にこのイベントが発生します。

- **イベントデータ例**

```
$$$,CHANGE_DETECT,02,FF
```

## 8.2 COUNT_EXCEED イベント

- **イベント発生条件**

UIOUSB の カウンタ入力ポートに対する入力値の変化回数が、予め決められた上限値を越えた場合に発生します。

カウント上限値は、”change_count_high” コマンドで設定します。

- **イベントデータフォーマット**

```
$$$,COUNT_EXCEED,[change_count]
```

[change_count] には、現在のカウント値(入力ポートの変化回数) が10進数表記で設定されます。

- イベント関連コマンド(コマンドの詳細は “コマンドリファレンス” の章を参照してください)

“count_check” カウンタチェックを行うかどうかのフラグを設定する

“count_high” カウンタ制限値上限を設定する

“count_reset” 現在のカウンタ値をクリアする

“sampling_rate” 自動サンプリング間隔を設定して、そのサンプルイベント中にカウンタ値を得る

“force_sample” 手動サンプリングを実施して、そのサンプルイベント中にカウンタ値を得る

- **備考**

**“signal_event” で設定したシグナル更新の対象イベントです。**

イベントの発生条件になる毎に、このイベントが発生してイベントデータが送信されます。

CHANGE_COUNT イベントを使用する場合には、”count_check” コマンドで 1 または 2 を設定して、イベント発生をイネーブル状態にして下さい。UIOUSBリセット後の 初期値は 0 で、ディセーブル状態です。

count_check コマンドで “1” を設定すると１回だけのイベント発生モードになります。”2” を設定すると連続してイベントを発生するモードになります。それぞれのモードの動作は以下の様になります。

- **“count_check 1”（１回だけのイベント検出モード)**

COUNT_EXCEED イベントが発生すると、自動的にイベント発生フラグがディセーブル状態になります。これは “count_check 0” コマンドを実行することに相当します。このため、もう一度 COUNT_EXCEED イベントを発生させたい場合は、イベント発生をイネーブル状態にして下さい。”count_check 1”

COUNT_EXCEED イベントが発生したときには、内部のカウンタ値は自動的にはクリアされません。クリアするためには、”count_reset” コマンドを実行してください。SAMPLING イベント発生時には自動的にカウンタ値はクリアされます。

- **“count_check 2”（連続したイベント検出モード）**

COUNT_EXCEED イベントが発生すると内部のカウンタ値は自動的にはクリアされて、次のイベント発生がイネーブル状態になります。

内部のカウンタ値をイベント発生前に強制的にクリアするためには、”count_reset” コマンドを実行してください。SAMPLING イベント発生時には自動的にカウンタ値はクリアされます。

- **イベントデータ例**

```
$$$,COUNT_EXCEED,10
```

## 8.3 SAMPLING イベント

- **イベント発生条件**

“sampling_rate” コマンドを使用して、自動サンプリング間隔(100ミリ秒単位)を 0 以外に設定した場合に、その設定した時間間隔で繰り返し発生します。

- **イベントデータフォーマット**

```
$$$,SAMPLING,[port_val],[counter_val],[ad0],[ad1],[ad2],[ad3]
```

[port_val] には 現在のポート値が16進数表記で設定されます。

[counter_val] 現在のカウンタ値が10進数で入る

[adc0].. [ad3] には、A/D 入力変換値が 10 進数表記で設定されます。

- イベント関連コマンド(コマンドの詳細は “コマンドリファレンス” の章を参照してください)

“sampling_rate” 自動サンプリング間隔を設定する。

“force_sample” 手動サンプリングを実施してサンプルイベントを１回だけ取得する

- **備考**

“sampling_rate” コマンドで設定したサンプリング間隔毎に、このイベントが発生してイベントデータが送信されます。

イベント発生と同時に、現在のポート値や A/D 変換値、カウンタ値等がイベントデータとして送信されます。

イベント発生と同時にカウンタ値(“count_check”,”count_reset”,”count_high” コマンド参照の事) はクリアされ 0 に戻されます。( ”force_sample” コマンドで手動サンプリングを行った場合にはクリアされません)

“sampling_rate” コマンドで指定した100ミリ秒単位間隔でイベントが発生しますが、時間精度は マイクロコントローラに接続されたクリスタルの精度に依存しています。そのため、正確なサンプリング間隔を必要とする場合には、サーバー等から “force_sample” コマンドを使用する方法を検討してください。UIOUSB で実行中のタスクの状況によっては、サンプリングイベント送信が遅れる(10ms .. 数秒程度)場合があります。

- **イベントデータ例**

```
$$$,SAMPLING,FF,0,817,613,410,205
```

## 8.4 ADVAL_UPDATE イベント

- **イベント発生条件**

A/D 変換値が ad_margin(ch#) で設定した値以上に変化した場合に発生します。

- **イベントデータフォーマット**

```
$$$,ADVAL_UPDATE,[ad_update_bits],[ad0],[ad1],[ad2],[ad3],[diff0],[diff1],[diff2],[diff3]
```

[ad_update_bits] には、変化したビットを 1、変化していないビットを 0にした値が、16進数で 00 から 0F までの値で、先頭の"0x"部分を取り除いて大文字で表記したものが入ります。
例えば、[ad_update_bits] が 03 の場合には ad0, ad1 の両方が、前回 ad0, ad1 のそれぞれに対するA/D 変化イベントを検出してから ad_margin0,ad_margin1 値で設定した値以上に変化したことを示します。
[adc0].. [ad3] には、最新の A/D 入力変換値が 10 進数表記で設定されます。
[diff0].. [diff3]には、各A/D チャンネル毎の変化(差分)値が入ります。値がマイナスの場合は前回と比較して値が減少している事を示します。

- イベント関連コマンド(コマンドの詳細は "TDCPコマンドリファレンス" の章を参照してください)

"ad_margin(ch#)" A/D 変化値を比較するときに使用するマージン値の設定または取得

- **備考**

**"signal_event" で設定したシグナル更新の対象イベントです。**
イベントの発生条件になる毎に、このイベントが発生してイベントデータが送信されます。

- **イベントデータ例**

```
$$$,ADVAL_UPDATE,01,651,766,511,254,-10,0,1,0
```

# 9 シグナル出力機能

UIOUSB には ランプの点滅やブザー出力でアラーム出力の機能があります。PORTA の RA4 と RA5 は常に出力ポートとしてこれらの出力を行っています。
UIOUSB コマンドでランプやブザーのシグナルステータスを変更することで、シグナル出力ポートから "点滅" やブ

ザー用の２種類の出力パターンが連続的に出力されます。

シグナルステータスは 8it幅のバイトデータで、各ビットは下記の意味があります。

| bit 7 | bit 6 | bit 5 | bit 4 | bit 3 | bit 2 | bit 1 | bit 0 |
|---|---|---|---|---|---|---|---|
| BEEP2 | | | LED<br>点滅 | BEEP1 | | | LED<br>点灯 |

上記のビット位置が “1” の時に、該当シグナル状態が ON であることを示します。”0” の時に、該当シグナル状態が OFF であることを示します。(空白の部分のステータスビットは使用していませんので値を設定しても無視されます)
シグナルステータスの値は、”signal_write”, “signal_bit”, “signal_read” コマンドで変更または参照することができます。

現在のシグナルステータスの値によって、UIOUSB は PORTA に下記の出力を行います。

| | | |
|---|---|---|
| PORTA | RA5 | シグナル ブザー出力 |
| | RA4 | シグナル LED 出力 |

シグナル LED は PORTA の RA4 に以下の値を出力することを意味します。

点灯 ON(High) または OFF(Low)

点滅 (High-Low 繰り返し)

また、PORTA の RA5 (シグナル ブザー) には、以下の値を出力します。

OFF(Low)

BEEP1(High-Low の 早い繰り返し) ブザー音 : PiPiPiPi....

BEEP2(High-Low の ゆっくりとした繰り返し) ブザー音 : Pi-,Pi-,Pi-....

## 9.1 タイマーによるシグナルの自動リセット

シグナルステータスを変更してシグナル出力を行った後に、シグナル出力を停止する場合には “signal_write”, “signal_bit” コマンドで、該当するシグナルステータスの値を ‘0’ にしますが、タイマーを設定することで UIOUSB コマンドを送信しないで自動的にシグナルを停止することができます。

“signal_timer” コマンドでタイマー値(秒) を 0 以外に設定すると、シグナルステータスを更新してからタイマー値に設定した秒を超えると、自動的にシグナルステータスが 0x00 にリセットされて全てのシグナル出力が停止します。

シグナルステータスをタイマーで設定した秒以前に連続して更新した場合には、最後に更新したタイミングからタイマーはカウントします。

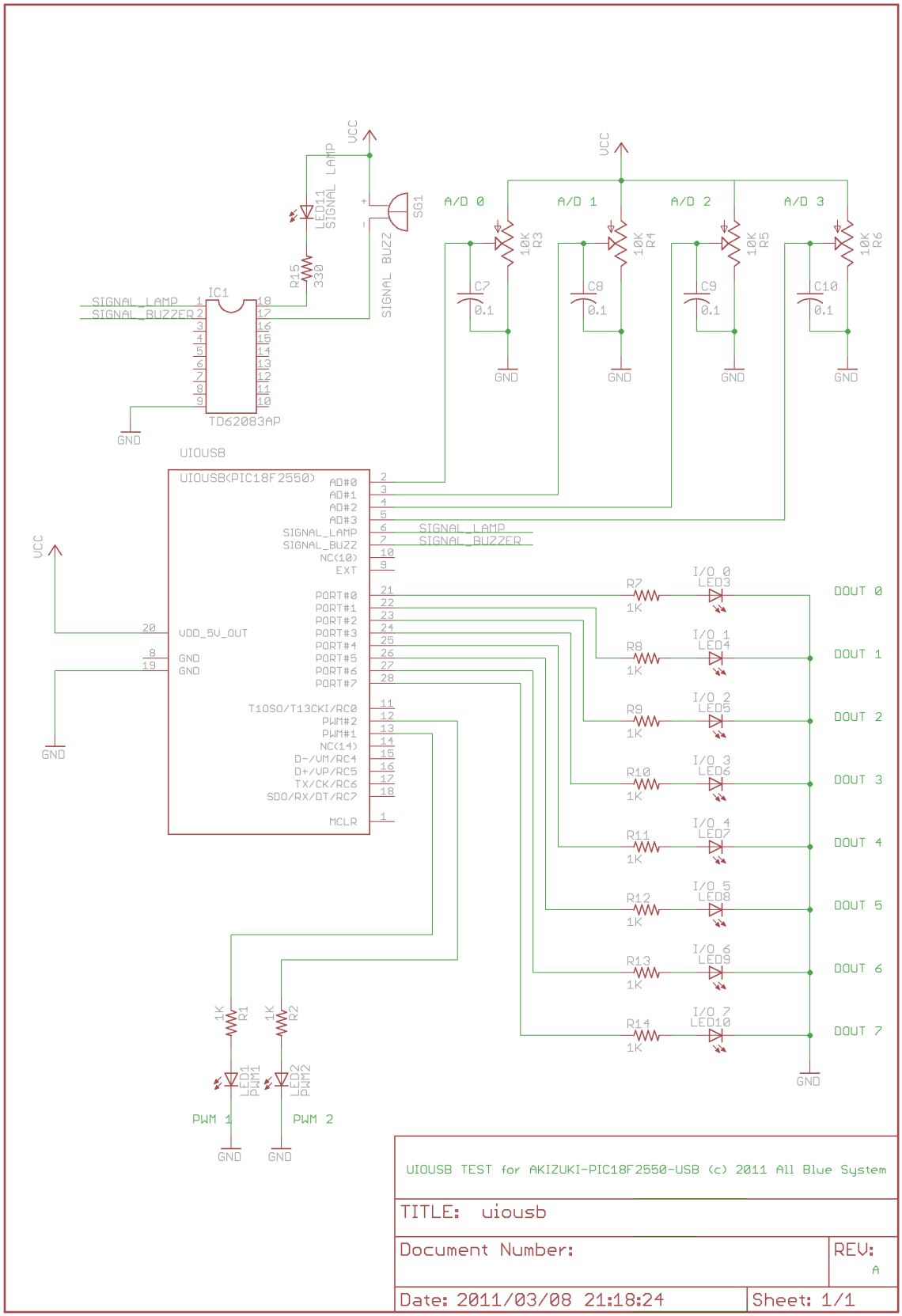
## 9.2 イベントと連動したシグナル設定

イベント送信条件になった場合にUIOUSB はイベントデータを出力しますが、同時にシグナルステータスを変更してアラーム出力をすることができます。この機能を使う場合には、予め “signal_event” コマンドでイベント送信条件になった場合にセットするシグナルステータスを設定しておきます。

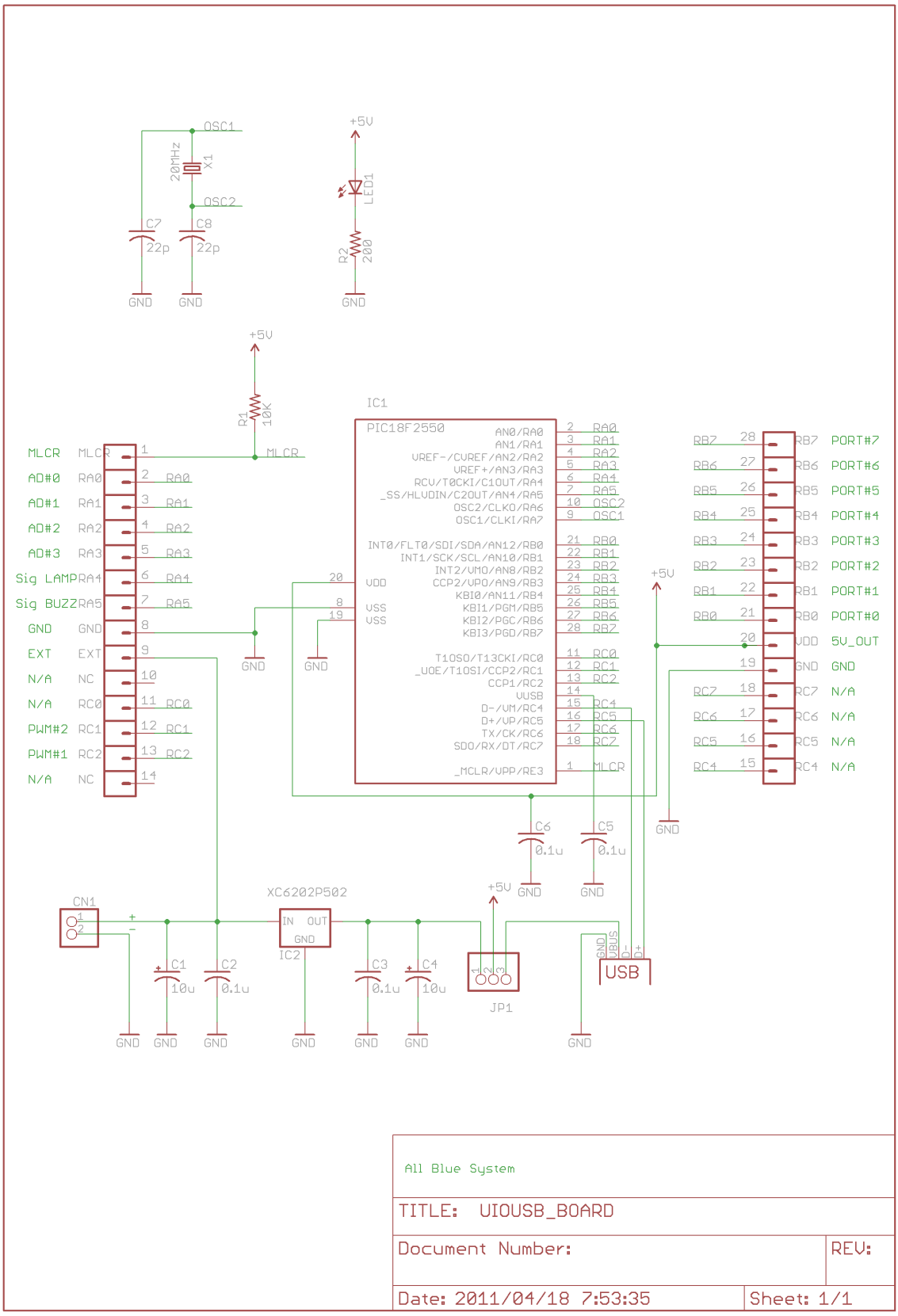
”イベントリファレンス” の章にイベントについての詳細と、どのイベントが “signal_event” の対象になるかが記述してありますの参照して下さい。

“signal_timer” コマンドと組み合わせることで、イベント送信と同時にシグナル出力を行った後、タイマーで設定した時間になるとシグナル出力を止めるような動作を自動でできます。

# 10　テスト用回路図


UCC
LED11
SIGNAL LAMP
SG1
SIGNAL BUZZ
R15
330
SIGNAL_LAMP
SIGNAL_BUZZER
IC1
TD62083AP
GND
A/D 0
A/D 1
A/D 2
A/D 3
10K
R3
R4
R5
R6
C7
C8
C9
C10
0.1
UIOUSB
UIOUSB(PIC18F2550)
AD#0
AD#1
AD#2
AD#3
SIGNAL_LAMP
SIGNAL_BUZZ
NC(10)
EXT
PORT#0
PORT#1
PORT#2
PORT#3
PORT#4
PORT#5
PORT#6
PORT#7
T1OSO/T13CKI/RC0
PWM#2
PWM#1
NC(14)
D-/VM/RC4
D+/VP/RC5
TX/CK/RC6
SDO/RX/DT/RC7
MCLR
VDD_5V_OUT
GND
GND
R1
R2
LED1
PWM1
LED2
PWM2
PWM 1
PWM 2
R7
R8
R9
R10
R11
R12
R13
R14
1K
I/O 0
LED3
I/O 1
LED4
I/O 2
LED5
I/O 3
LED6
I/O 4
LED7
I/O 5
LED8
I/O 6
LED9
I/O 7
LED10
DOUT 0
DOUT 1
DOUT 2
DOUT 3
DOUT 4
DOUT 5
DOUT 6
DOUT 7
UIOUSB TEST for AKIZUKI-PIC18F2550-USB (c) 2011 All Blue System
TITLE: uiousb
Document Number:
REV:
A
Date: 2011/03/08 21:18:24
Sheet: 1/1

# 11 UIOUSB ボード回路図



# 12　サポートについて

本ソフトウェアをオールブルーシステムの DeviceServer と組み合わせて使用する場合には、ABS-9000 DeviceServer のライセンスのサポート期間内において、メールにてサポートを行います。

UIOUSB のコマンドを利用した、お客様のアプリケーションソフトウエアを作成するときのサポートは別途有償対応となります。（詳しくはお問い合わせください）

オールブルーシステムの UIOUSB はファームウエアとそのソフト的な不具合に対してのみサポート致します。お客様のハードウエアへの組み込みに関する内容や、PIC18F2550 USB CPU ボード自身の仕様や不具合に関しましてはサポートできません。本ソフトウェアをオールブルーシステムの ABS-9000 DeviceServer と組み合わせずに、使用する場合のサポートは行いません。

ABS-9000 DeviceServer のスクリプトライブラリ関数(Luaのライブラリ)とEXCEL VBA から利用するための API ライブラリ関数(XASDLCMD.DLL)を利用する場合の、プログラミング方法やサンプルプログラムのソースコードに関する解説など、お客様のソフトウェア開発自身に関するサポートにつきましては、別途有償対応となります、詳しくはお問い合わせください。

動作環境を満たしている場合でも、お客様の環境やハードウエアによっては正常に動作しない場合があります。これらを含めて、こちらで再現できない問題については十分なサポートができない場合があります。ABS-9000 DeviceServer のライセンスをご購入になる前に、目的の機能がお客様の環境で動作することを、事前にデモライセンスをご利用になって、充分確認されることをお勧めします。

# 13　更新履歴

REV A.2.6-7 2013/03/14

　　A/D マージン設定時の注意事項を追記

REV A.2.5 2011/04/18

　　機能一覧にA/D マージン検出機能の記述が抜けていたのを修正

REV A.2.5 2011/04/18

　　UIOUSB ボード回路図追加

REV A.2.4 2011/03/08

　　テスト用回路図中の部品名修正。

REV A.2.3 2010/12/29

　　スペルミス、文章修正

REV A.2.2 2010/12/3

　　ADVAL_UPDATE イベントに diff(ch#) データ項目を追加した。

REV A.2.1 2010/11/27

ADVAL_UPDATE イベントを新規追加した。

ad_margin コマンドを新規追加した。

REV A.2.0 2010/11/24

ファームウエアのバージョンを ver1.0 -> ver2.0 に更新して下記の新規機能を追加した。

（自動・手動サンプリング、イベント送信、カウンタ入力、シグナル出力）