TDCP(Tiny Device Control Program for XBee)

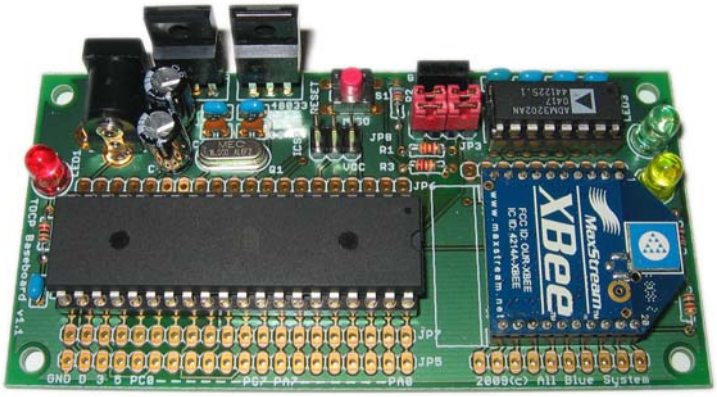
TDCPZB(for XBee-ZB)

ユーザーマニュアル

TDCP User Manual Rev A.3.2　2015/3/28

TDCP Firmware version: 2.10

TDCPZB Firmware version: 1.01



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 6
1.1 著作権および登録商標 6
1.2 連絡先 6
2 使用条件およびライセンス 6
3 イントロダクション 7
4 TDCP動作仕様 9
4.1 マイクロコントローラ(Atmega1284P) 10
4.2 TDCPファームウエア書き込み時の Fuse Bits 設定値 11
4.3 XBee モジュール 11
4.3.1 XBee 802.15.4 RF Module Series1 (TDCP) 11
4.3.2 XBee-ZB ZigBee対応 RF Module Series2 (TDCPZB) 12
5 TDCPデータパケットフォーマット 12
5.1 XBee RF データパケット 13
5.2 リクエストコマンド 13
5.3 リプライデータ 14
5.4 イベントデータ 14
6 TDCP 動作モード(app_mode) 15
6.1 app_mode 0 15
6.2 app_mode 1 16
6.3 app_mode 2 18
6.4 app_mode 3 20
6.5 app_mode 4 22
6.6 app_mode 5 23
6.7 app_mode 6 26
6.8 app_mode 7 28
6.9 app_mode 8 31
6.10 app_mode 9 34
6.11 app_mode 10 36
7 TDCPコマンドリファレンス 39
7.1 help 40
7.2 version 40
7.3 reset 41
7.4 server_addr 42
7.5 interval 43

7.6 config_save 44
7.7 config_load 45
7.8 config_clear 45
7.9 uart0_baud 46
7.10 uart0_echo 47
7.11 sampling_rate 48
7.12 heartbeat_rate 50
7.13 force_sample 51
7.14 sample_once 52
7.15 second_adjust 53
7.16 pullup,dio_pullup 54
7.17 change_detect 55
7.18 app_mode 56
7.19 dio_read,port_read 57
7.20 dio_bit,port_bit 58
7.21 dio_write,port_write 59
7.22 pulse 60
7.23 ad_read,adc_read 61
7.24 ad_vref,adc_vref 62
7.25 range_check_high 63
7.26 range_check_low 64
7.27 range_high 65
7.28 range_low 66
7.29 change_count_check 67
7.30 change_count_high 69
7.31 change_count_reset 70
7.32 signal_read 70
7.33 signal_bit 71
7.34 signal_write 72
7.35 gps_gga 73
7.36 gps_rmc 74
7.37 gps_report,position_report 75
7.38 gps_once,position_once 76
7.39 gps_rate,position_report_rate 77
7.40 serial_number 78
7.41 echo 79
7.42 tx_data 79
7.43 tx_ascii,tx 80
7.44 eeprom_read 82

7.45 eeprom_write 82
7.46 lcd_clear 83
7.47 lcd_wrap 84
7.48 lcd_disp 85
7.49 lcd_lines 86
7.50 lcd_length 88
7.51 router (TDCP専用) 89
7.52 reply_prefix,rp (TDCP専用) 91
7.53 timezone 93
7.54 datetime 94
7.55 signal_timer 95
7.56 signal_event 96
7.57 ad_margin 97
7.58 ad_margin_reset 99
7.59 i2c_use 100
7.60 i2c_read 101
7.61 i2c_write 102
7.62 i2c_slave_addr 104
7.63 i2c_work_read 105
7.64 i2c_work_write 106
7.65 spi_config 107
7.66 spi_read 109
7.67 spi_write 110
7.68 network_address (TDCPZB専用) 111
7.69 association_indication (TDCPZB専用) 112
7.70 network_reset (TDCPZB専用) 112
7.71 disassociation (TDCPZB専用) 113

8 TDCPイベントリファレンス 114

8.1 CHANGE_DETECTイベント 115
8.2 RANGE_EXCEEDイベント 115
8.3 COUNT_EXCEEDイベント 116
8.4 SAMPLINGイベント 118
8.5 $GPRMCイベント 119
8.6 GPSイベント 120
8.7 LIVEイベント 121
8.8 ADVAL_UPDATEイベント 122
8.9 I2C_SLAVE_EVENTイベント 123
8.10 I2C_SLAVE_OVERFLOWイベント 124

9 シグナル出力機能 124
9.1 タイマーによるシグナルの自動リセット 125
9.2 イベントと連動したシグナル設定 126
10 GPSレシーバ接続 126
10.1 GPS レシーバ接続時の通信条件 126
10.2 TDCP コンフィギュレーション 127
10.3 測位データの取得 127
11 I2C接続機能 128
11.1 I2C マスター機能 128
11.2 I2C スレーブ機能 129
11.2.1 TDCP 動作モード取得コマンド(0x01) 129
11.2.2 イベント送信コマンド(0x11) 130
11.2.3 ワークエリアデータ書き込みコマンド(0x21) 130
11.2.4 ワークエリアデータ読み出しコマンド(0x22) 130
12 SPI接続機能 131
13 TDCP リモートコントローラボード作成例 131
13.1 CPU ボード回路図 132
13.2 基板パターン図 133
13.3 部品リスト 134
13.4 シリアルポート設定用ジャンパー(JP3) 135
13.4.1 通常運用時ジャンパー設定 136
13.4.2 XBee ATコマンド操作用ジャンパー設定 136
13.5 TDCP XBee 802.15.4 RF Module Series1 初期設定 136
13.6 TDCPZB XBee-ZB ZigBee対応 Module Series2 初期設定 138
13.7 TDCP,TDCPZB プログラム書き込み 144
13.8 動作確認 145
14 本製品に使用したソフトウェアライセンス表記 145
15 サポートについて 149
16 更新履歴 149

1 このマニュアルについて

1.1 著作権および登録商標

Copyright© 2009-2014 オールブルーシステム

このマニュアルの権利はすべてオールブルーシステムにあります。無断でこのマニュアルの一部を複製、もしくは再利用することを禁じます。

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

XBee® and XBee-PRO® are registered trademarks of Digi, Inc.

Atmel®, logo and combinations thereof, AVR® and others are registered trademarks or trademarks of Atmel Corporation or its subsidiaries.

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。

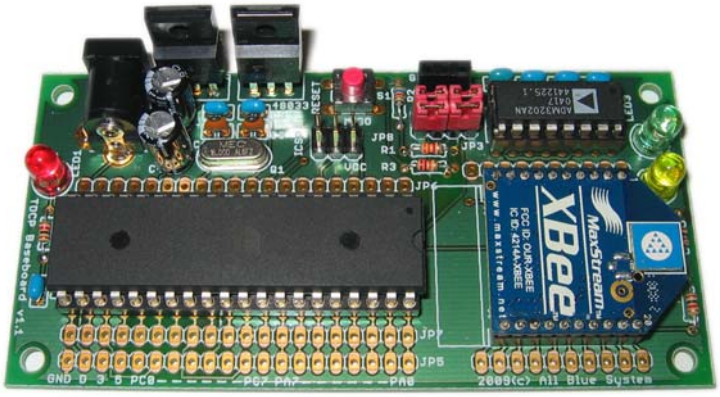
3　　イントロダクション

TDCP リモートコントロールモニタプログラム(以降 “TDCP” と略す) の機能について説明します。
TDCP は、Atmel Atmega1284P[1] マイクロプロセッサ用のファームウエアです。XBee[2] (XBee 802.15.4 RF Module) デバイスまたは、XBee-ZB デバイスと組み合わせて、リモートからマイクロプロセッサ上の I/O ポートや A/D 機能を操作することができます。

XBee 802.15.4 RF Module (Series1) デバイスと組み合わせる場合には TDCP ファームウエア(このマニュアルでは TDCP と表します)を使用します。

XBee-ZB ZigBee 対応 Module (Series2) デバイスと組み合わせる場合には TDCPZB ファームウエア(このマニュアルでは TDCPZB と表します)を使用します。

このマニュアルでは TDCP と TDCPZB の区別が必要な部分以外では、TDCP という共通の名前で説明しています。双方で説明が異なる場合や機能に違いがある場合には、見出しや項目名に TDCP または TDCPZB を付けています。



(TDCP を組み込んだCPU ボード例)

TDCP は全ての操作を XBee デバイスの RF データパケット中に埋め込んだコマンド(TDCPコマンド)で行います。
離れた場所から、I/O ポート操作や設定値の変更など全ての操作が行えます。シリアルポートから直接コマンド(TDCPコマンド)実行を行うことも可能です。 TDCP の主な機能は以下のものがあります。

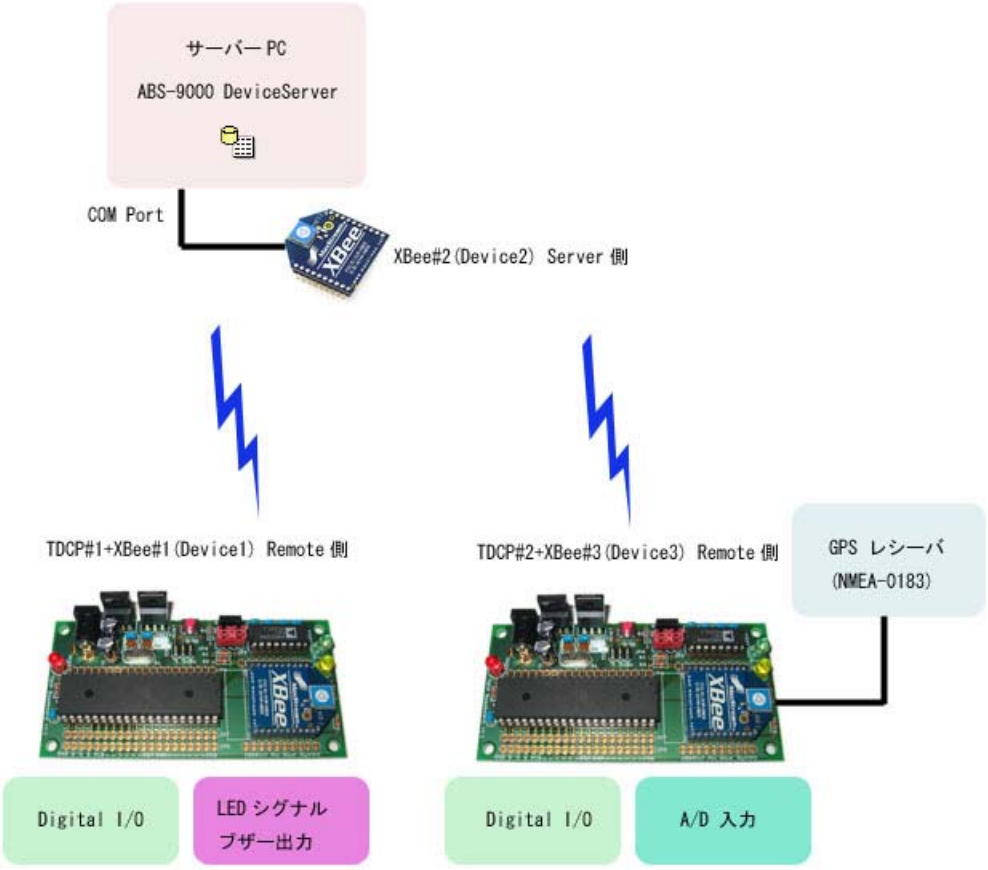
- リモートから指定した値を I/O ポートに出力することができます。
- I/O ポート入力値をリモートで取得することができます。
- A/D 変換値をリモートで取得することができます。

---

1 Atmel®, logo and combinations thereof, AVR® and others are registered trademarks or trademarks of Atmel Corporation or its subsidiaries.
2 XBee® and XBee□PRO® are registered trademarks of Digi, Inc.

- I/O ポート入力変化時にイベントデータをリモートに送信することができます。
- 予め設定した A/D 変換値の上限または下限値を超えた場合に、イベントデータをリモートに送信することができます。
- 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、イベントデータをリモートに送信することができます。
- 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)等をリモートに送信することができます。
- TDCP を搭載したマイクロプロセッサの CPU ボードに I2C インターフェイスで接続されたデバイスに対して、リモートからデータの取得や更新を行えます。また、TDCP デバイス自身を I2C スレーブデバイスとして使用することで、他のコントローラ(I2C マスター)から TDCP デバイス経由でイベントデータをリモートに送信したり、TDCPデバイス中の共有データ領域をサーバや他のコントローラデバイス間で利用することができます。
- TDCP を搭載したマイクロプロセッサの CPU ボードに SPI インターフェイスで接続されたデバイスに対して、リモートからデータの取得や更新を行うことができます。
- シグナル(３種類の LED)の点滅・点灯、ブザー出力(２種類のパターン)をリモートから設定することができます。
- TDCP を搭載したマイクロプロセッサのシリアルポートに 市販のGPS モジュールを接続して、任意のタイミングで測位情報をリモートに送信することができます。また NMEA-0183 センテンス受信毎に測位情報を連続送信することができます。。
- LCD 表示モジュールを接続してリモートから任意の文字列を表示させたり、自動的にスクロール表示させることができます。
- TDCP 設定値をマイクロプロセッサ内の EEPROM に保存、リセット時に自動的に設定値をロードすることができます。
- TDCP 設定値変更、EEPROM への保存、プロセッサリセット等の作業を、リモートから全て操作可能です(シリアルポート経由でのコマンド操作も可能)

TDCPを組み込んだデバイスを使用したアプリケーション例を示します。



(上記アプリケーション例の説明)

TDCP デバイス(TDCP#1, TDCP#2) にそれぞれ I/O ポートにセンサーやコントロール対象のデバイスが接続されます。また、TDCP#1 には、LEDx3(赤、黄、緑)とブザーを接続して簡単にアラーム出力機能を持たせています。これらのコントロールは サーバー PCに接続した XBee デバイス(XBee#2) のデータパケットに記述したTDCP コマンド経由でコントロールされます。TDCP#2 には GPS レシーバをシリアルポート経由で接続して、測位情報を取得できるようになっています。この測位情報をサーバーPC からいつでも利用することが可能です。

DeviceServer を使用することで、TDCP コマンドの発行とレスポンスの受信や解析を簡単にスクリプトで記述することができます。また、TDCP で発生したイベントを DeviceServer のイベントハンドラで処理することも簡単にできます。DeviceServer を使用しない場合でも、TDCP コマンドを記述した RFData パケットを TDCP デバイスに送信したり、レスポンスやイベントデータの RFData パケットの受信を行うことが可能です。

4 TDCP動作仕様

TDCP 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。

- CPU　　Atmel社製 Atmega1284P マイクロコントローラ
- XBee　　TDCP Digi international社製 XBee 802.15.4 RF Module(Series1)
　　TDCPZB Digi international社製 XBee-ZB ZigBee対応 RF Module(Series2)
- CPU クロック　16MHzまたは8MHz
- シリアルI/F　RS232C用レベルコンバータ(ADM2303等) シリアルコンソールまたは GSP接続を行う場合のみ
- 電源　　5Vまたは3.3V(Atmega1284P 用) 、3.3V(XBee用)

詳しくは、”TDCP リモートコントローラボード作成例”の章を参照してください。

⚠ 注意

オールブルーシステムは、ファームウエア以外のハードウエア部分(添付のリモートコントロールボード回路図例を含む)についてのサポートは行いません。 また常にお客さまのハードウエア環境で動作することは保証できません。

4.1　マイクロコントローラ(Atmega1284P)

		設定値
CPU クロック		16MHz または 8MHz
電源電圧		5V または 3.3V(*1)
Flash memory		TDCP プログラム格納用
EEPROM		TDCP コンフィギュレーション保存用
PORTA	bit 0-7	“app_mode” の設定で、I/O ポート、A/D 変換入力、シグナル出力に使用する。 詳しくは、”TDCP 動作モード” の章を参照してください。
PORTB	bit 0	(RECV) XBee 経由でデータフレームを受信した時に点滅(high->low)
	bit 1	(RUN) TDCP 内のタスク処理間隔で high->low を繰り返す。 ノンプリエンプティブでタスク処理を行うため、重いタスク実行中は繰り返し間隔が長くなる。
	bit 2	(SAMPLING) サンプリング間隔毎にhigh->lowを繰り返す “sampling_rate” コマンドを参照の事
	bit 4	SPI 通信時の SS(Slave Select) 信号を出力する
	bit 5	SPI 通信時の MOSI 信号を出力する
	bit 6	SPI 通信時の MISO 信号を入力する
	bit 7	SPI 通信時の SCK 信号を出力する
PORTC	bit 0-7	“app_mode” の設定で、I/O ポート、シグナル出力に使用する。 詳しくは、”TDCP 動作モード” の章を参照してください。 “i2c_use” の設定で、I2C機能を使用する場合には PORTC bit0 は “SCL”,bit1 は “SDA” にアサインされます。
PORTD	bit 5	リセット直後はハイインピーダンス状態で、TDCP起動時に約2 マイクロ秒の LOW 信号を出力した後、ハイインピーダンス状態に戻る。XBee デバイスのリセット用に使用可能

	bit 6	“spi_write” コマンドで〈latch_pulse〉を指定した場合に、シリアルデータ出力後にパルス信号が出力されます。
	bit 7	リセット直後のTDCP 起動時に、LOW しておくことでEEPROM データを初期化することができます。
シリアルポート#0 (RXD0/TXD0)		TDCP コマンド入力(シリアルポート経由)、メッセージ出力、GPS NMEA-0183 センテンス受信に使用する。ボーレートや GPS 受信時のエコーバック抑止の設定については、“uart0_baud”, “uart0_echo” コマンドを参照して下さい。
シリアルポート#1 (RXD1/TXD1)		XBee のデータパケット送受信用。TDCP コマンド入力、コマンドレスポンス出力、イベントパケット送信をXBeeデータパケット経由で行う 通信条件は 9600bps 8bit no-parity 固定。

(*1) 電源電圧を 3.3V で使用する場合には CPU クロックは 8MHz で使用してください。

4.2 TDCPファームウエア書き込み時の Fuse Bits 設定値

Fuse Bits	設定値
Low Byte	0b11100110
High Byte	0b11011111
Extended Byte(*1)	0b100 電源電圧 5V で使用する場合 0b101 電源電圧 3.3V で使用する場合

(*1) EEPROM 動作に必要な電源電圧を保障するために、必ず Extended Byte の “BODLEVL2:0” を上記の値にセットして下さい。

4.3 XBee モジュール

4.3.1 XBee 802.15.4 RF Module Series1 (TDCP)

XBee デバイスを TDCP に接続する前に、XBee デバイス自身の初期設定を行う必要があります。設定する項目は以下の内容です。

XBee モジュール デフォルト値から変更が必要な設定値	
API モード	1 (default は 0)
PAN(Personal Area Network) ID	任意の値 (default は0x3332) デフォルトの値のままだと、予期しないデバイスからのフレームを受信する恐れがありますので、適当な任意の値を設定するようにしてください。このマニュアルでは 0xAB90 を使用しています。

16bit Source Address	同一PAN ID 内でユニークな値 (default は 0x0000) この値は全てのデバイス間で違った値を設定してください。(0x0000,0xFFFF,0xFFFE を除く) 例えば、0x0001, 0x0002,0x0003 等。

4.3.2 XBee-ZB ZigBee対応 RF Module Series2 (TDCPZB)

XBee-ZB デバイスを TDCP に接続する前に、XBee-ZB デバイス自身の初期設定を行ってください。

XBee-ZB には ZigBee デバイスタイプに対応して、Coordinator, Router, End device の３種類のファームウエアがありますが、TDCPZB で使用可能なのは Router タイプのみです。

XBee-ZB デバイス デフォルト値から変更が必要な設定値	
ファームウエアの書き換え	ZIGBEE ROUTER API を使用してください。
API モード	1 ZIGBEE ROUTER API タイプのファームエア書き換え直後は 1 に設定されていますが、念のため確認してください
PAN(Personal Area Network) ID	任意の値 (default は0) デフォルトの値のままだと、予期しないデバイスからのフレームを受信したり、間違ってデバイスを操作する恐れがありますので、適当な任意の値を設定するようにしてください。このマニュアルでは 0xAB9000 を使用しています。
Node Identifier	同一PAN ID 内でユニークな文字列 (default は "") この値は、後から ZB管理プログラムで設定・変更することも可能ですが、デバイス一覧から選択したデバイスがどのデバイスであるかを見分けることが容易になるように便宜的にここで設定します。デバイスを選択するときにわかり易いように全てのデバイスで異なった名前を設定してください。例えば、"NODE01", "NODE02" 等。

5 TDCPデータパケットフォーマット

TDCP は XBee デバイスの RF データパケット中に埋め込んだコマンド(TDCPコマンド)を受信して、コマンドに指定された動作を行います。その後、オプションでTDCP コマンド実行結果を XBee デバイスの RF データパケットに埋め込んで、コマンド送信元の XBee デバイスに送信します。

TDCP の 自動サンプリング間隔を設定した場合や、TDCP の I/O ポートや A/D 入力が予め決められた条件になった場合に、予め決められた XBee デバイス(server_addr) にイベントデータを RF データパケットに埋め込んで送信します。

5.1 XBee RF データパケット

TDCP のリクエストコマンドとリプライデータ、イベントデータは全て XBee モジュールのデータパケット中で送信または受信されます。リクエストコマンドとリプライデータは、TDCP のシリアルポートに接続した RS232C 端末からも操作可能です。

XBee モジュールからデータパケットを送信する時のシリアルデータは下記のデータ構造になっています。

```
<StartDelimiter><Length><API Identifier><FrameID>DestinationAddress><Options><RFData><CheckSum>
```

リクエストコマンドデータ送信時は、上記の <API Identifier> に 0x00 または 0x01 を使用して、<RFData> 部分に TDCP コマンド文字列とパラメータを格納します。リクエストコマンドデータ送信時を行う(コマンド送信元の) XBee モジュールに対して <FrameID> を 0x00 以外に指定する場合は、TDCP からレスポンスデータパケットを受信する前に、リクエストコマンドデータ送信に対するステータスデータパケット (API = 0x89) を XBee から受信することになります。

XBee モジュールでデータパケットを受信する時のシリアルデータは下記の構造になっています。

```
<StartDelimiter><Length><API Identifier><SourceAddress><RSSI><Options><RFData><CheckSum>
```

リプライデータは、上記の <API Identifier> に 0x80,0x81 を使用して、<RFData> 部分に TDCP コマンドレスポンス文字列が格納されてリクエストコマンド送信元の XBee モジュールで受信されます。イベントデータもリプライデータと同様に <RFData> 部分にイベントデータ文字列を受信しますが、受信対象となる XBee モジュールは、TDCP コマンド “server_addr” で予め設定したアドレスを持つデバイス(ブロードキャストアドレスを設定した場合は複数のデバイスが受信対象)になります。

XBee データパケットの詳細については、”XBee/XBee-PRO OEM RF Modules Product Manual”を参照して下さい。

5.2 リクエストコマンド

リクエストコマンドを送信するときの XBee データパケット中の <RFData> の構造は下記の様になっています。

```
<TDCPPrefix>,<Command>[,<Param#1>,<Param2>,...]
```

- 全てのフィールドはカンマ “,” で区切ります。カンマの前後に空白を入れてはいけません。
- 各項目フィールドに使用可能な文字は英数字のみです。
- <TDCPPrefix> は “$$$” 文字列とその後にオプションで任意の英数字を最大 5 文字までつけたものです。 “$$$” は、XBee のデータパケット中に TDCP データ(リクエストコマンド、リプライデータ、イベントデータ)が入っていることを示します。任意の文字列が “$$$” の後に付く場合には、コマンドを受信した TDCP デバイスに対して、リプライデータを返信するように指示します。この時に、リプライデータ中に含まれる <TDCPPrefix> はリクエストコマンドで指定した<TDCPPrefix>と同様の文字列が格納されます。
- <Command> は TDCP コマンド文字列を表します。

- <Param#1>... はオプションでTDCPコマンドパラメータを表します。コマンドパラメータが無い場合は省略されます。複数のコマンドパラメータを入れる場合にはパラメータ間をカンマ “,” で区切ります

リクエストコマンド例：(両端のダブルコートは、実際のデータ中には含めません)

```
“$$$,port_write,FF”
“$$$,reset”
“$$$12345,port_read”
“$$$abc,force_sample”
```

5.3 リプライデータ

リプライデータを受信したときの XBee データパケット中の <RFData> の構造は下記の様になっています。

```
<TDCPPrefix>,<status>[,<Reply#1>,<Reply#2>,...]
```

- 全てのフィールドはカンマ “,” で区切られます。
- <TDCPPrefix> は “$$$” 文字列とその後にオプションで任意の英数字を最大 5 文字までつけたものです。“$$$” は、XBee のデータパケット中に TDCP データ(リクエストコマンド、リプライデータ、イベントデータ)が入っていることを示します。任意の文字列が “$$$” の後に付く場合には、コマンドを受信した TDCP デバイスに対して、リプライデータを返信するように指示します。この時に、リプライデータ中に含まれる <TDCPPrefix> はリクエストコマンドで指定した<TDCPPrefix>と同様の文字列が格納されます。
- <status> は TDCP コマンド実行が成功した場合に “1”,失敗した場合に “0” が返ります。
- <Reply#1>... はTDCPリプライデータに<Status>以外に、データを取得するコマンドを実行した場合に格納されます。リクエストコマンドと対応するリプライデータの詳細は “TDCPコマンドリファレンス”の章を参照して下さい（例：”port_read”,”version” コマンド等）

リプライデータ例：(両端のダブルコートは、実際のデータ中には含めません)

```
“$$$,1”
“$$$,0”
“$$$12345,1,FF”
“$$$abc,1,100,121,22,334”
```

5.4 イベントデータ

イベントデータ送信時の XBee データパケット中の <RFData> の構造は下記の様になっています。

```
<TDCPPrefix>,<EventName>[,<EventData#1>,<EventData#2>,...]
```

- 全てのフィールドはカンマ “,” で区切られます。
- <TDCPPrefix> は 常に“$$$” 文字列になります。
- <EventType> は TDCP イベントタイプ文字列が入ります。

- <EventData#1>... はTDCPイベントに含まれるデータが格納されます。（例：SAMPLINGイベントのサンプリングデータやデバイスアドレス等）

イベントタイプとイベントデータの詳細は “TDCPイベントリファレンス”の章を参照して下さい

イベントデータ例：(両端のダブルコートは、実際のデータ中には含めません)

```
“$$$,SAMPLING,0A01,8,FF,0,100,120,130,140”
“$$$,CHANGE_DETECT,0D04,8,01,FF”
“$$$,$GPRMC,084954,A,4254.1841,N,14135.6412,E,0.0,0.0,211009,9.3,W,A*0C”
“$$$,GPS,0A01,9,A,4254.1627,N,14135.6058,E,000.0,1,28.8,M”
```

6 TDCP 動作モード(app_mode)

TDCP は動作モードを変えることで、PORTAとPORTC にアサインする機能を変更することができます。
動作モード変更するときは、”app_mode” コマンドを使用します。動作モードを変更した場合には、リセット後に TDCP は新しい動作モードになります。

app_mode 設定コマンド実行例 (app_mode 8,入力ポートの全プルアップ)

```
app_mode,8
pullup,FF
config_save
reset
```

注意

“app_mode” コマンドでPORTA, PORTC の入出力モードや A/D 機能を切り替える場合には、回路が適切に接続されていることを十分確認してください

注意

”app_mode”,“pullup” コマンド設定値は、コマンド実行後に config_saveコマンドを実行してEEPROM に最新のコンフィギュレーションを保存した後に、”reset” コマンド実行などで TDCPプログラムが再起動してから有効になります。

以降で動作モード毎の機能を説明します。

6.1 app_mode 0

- **機能概要**

  PORTA,PORTC は TDCP で使用しない。

- **PORT機能**

PORTA	bit 7-0	未使用
PORTC	bit 7-0	未使用

- **イベントデータ(LIVE)**

TDCP

```
$$$,LIVE,<my_addr16>,<app_mode>
```

TDCPZB

```
$$$,LIVE,<app_mode>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “0” が設定されます。

- **サンプリングイベントデータ**

無し

- **備考**

TDCP は、初期状態でこのモードになっています。

6.2 app_mode 1

- **機能概要**

PORTAとPORTC を合わせて 16bit幅 デジタル入力ポートとして使用する。

- **PORT機能**

PORTA	bit 7-0	デジタル入力ポート (High Byte)
PORTC	bit 7-0	デジタル入力ポート (Low Byte)
	(bit 0)	SCL (i2c_use > 0 の場合)
	(bit 1)	SDA (i2c_use > 0 の場合)

- **イベントデータ(CHANGE_DETECT)**

TDCP

```
$$$,CHANGE_DETECT,<my_addr16>,<app_mode>,<diff_bits>,<dio>
```

TDCPZB

```
$$$,CHANGE_DETECT,<app_mode>,<diff_bits>,<dio>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “1” が設定されます。

<diff_bits> には、PORTA とPORTC を合わせた 16bit 幅で、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進数表記で設定されます。

<dio> には PORTA とPORTC を合わせた 16bit 幅の値が 16進数表記で設定されます。

- **サンプリングイベントデータ(SAMPLING)**

TDCP

```
$$$,SAMPLING,<my_addr16>,<app_mode>,<dio>
```

TDCPZB

```
$$$,SAMPLING,<app_mode>,<dio>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “1” が設定されます。

<dio> には PORTA とPORTC を合わせた 16bit 幅の値が 16進数表記で設定されます。

- **イベントデータ(LIVE)**

TDCP

```
$$$,LIVE,<my_addr16>,<app_mode>
```

TDCPZB

```
$$$,LIVE,<app_mode>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “1” が設定されます。

- **force_sample リプライデータ**

```
<TDCPPrefix>,<status>[,<app_mode>,<dio>]
```

<TDCPPrefix>にはリクエストコマンドで指定した <TDCPPrefix> 文字列が入ります。

<status> には “force_sample コマンド実行が成功した場合には、”1” が入り、その後にサンプリングデータが続きます。コマンド実行失敗時には “0” が入ります。サンプリングデータの各フィールドの説明は、サンプリングイベントデータと同一です。

- **備考**

“port_read” コマンドは 16bit 幅の値を返します。

“change_detect” コマンドは 16bit 幅の設定値をとります。

“pullup” コマンドは 16bit 幅の設定値をとります。

6.3 app_mode 2

- **機能概要**

  PORTA を A/D 入力、PORTC をデジタル入力ポートとして使用する。

- **PORT機能**

PORTA	bit 7-0	A/D 入力
PORTC	bit 7-0	デジタル入力ポート
	(bit 0)	SCL (i2c_use > 0 の場合)
	(bit 1)	SDA (i2c_use > 0 の場合)

- **イベントデータ(CHANGE_DETECT)**

  TDCP

  ```
  $$$,CHANGE_DETECT,<my_addr16>,<app_mode>,<diff_bits>,<dio>
  ```

  TDCPZB

  ```
  $$$,CHANGE_DETECT,<app_mode>,<diff_bits>,<dio>
  ```

  <my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

  <app_mode> は “2” が設定されます。

  <diff_bits> には、8bit 幅で、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進数表記で設定されます。

  <dio> には 8bit 幅の値が 16進数表記で設定されます。

- **イベントデータ(RANGE_EXCEED)**

  TDCP

  ```
  $$$,RANGE_EXCEED,<my_addr16>,<app_mode>,<high_exceed_bits>,<low_exceed_bits>
  ```

  TDCPZB

  ```
  $$$,RANGE_EXCEED,<app_mode>,<high_exceed_bits>,<low_exceed_bits>
  ```

  <my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

  <app_mode> は “2” が設定されます。

  <high_exceed_bits> には、8bit 幅で、”range_check_high” コマンドで指定したチェック対象の中で、”range_high” コマンドで指定した上限値を超えた A/D ポートのビットを 1、制限値内のビットを 0にした値が、16進数表記で設定されます。

  <low_exceed_bits> には、8bit 幅で、”range_check_low” コマンドで指定したチェック対象の中

で、”range_low” コマンドで指定した下限値を超えた A/D ポートのビットを 1、制限値内のビットを 0にした値が、16進数表記で設定されます。

- **サンプリングイベントデータ(SAMPLING)**

TDCP

```
$$$,SAMPLING,<my_addr16>,<app_mode>,<dio>,<adc0>,<adc1>,<adc2>,<adc3>,<adc4>,<adc5>,<adc6>,<adc7>
```

TDCPZB

```
$$$,SAMPLING,<app_mode>,<dio>,<adc0>,<adc1>,<adc2>,<adc3>,<adc4>,<adc5>,<adc6>,<adc7>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “2” が設定されます。

<dio> には 8bit 幅の値が 16進数表記で設定されます。

<adc0>.. <adc7> には、A/D 入力変換値が 10 進数表記で設定されます。

- **イベントデータ(LIVE)**

TDCP

```
$$$,LIVE,<my_addr16>,<app_mode>
```

TDCPZB

```
$$$,LIVE,<app_mode>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “2” が設定されます。

- **イベントデータ(ADVAL_UPDATE)**

TDCP

```
$$$,ADVAL_UPDATE,<my_addr16>,<app_mode>,<ad_update_bits>,<adc0>,<adc1>,<adc2>,<adc3>,
<adc4>,<adc5>,<adc6>,<adc7>
```

TDCPZB

```
$$$,ADVAL_UPDATE,<app_mode>,<ad_update_bits>,<adc0>,<adc1>,<adc2>,<adc3>,<adc4>,<adc5>,<adc6>,<adc7>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “2” が設定されます。

<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 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。

例えば、<ad_update_bits> が 03 の場合には adc0, adc1 の両方が、前回 adc0, adc1 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_margin,0,xx”,”ad_margin,1,xx” で設定した値以上に変化したことを示し

ます。

<adc0>.. <adc7> 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。

- **force_sample リプライデータ**

```
<TDCPPrefix>,<status>[,<app_mode>,<dio>,<adc0>,<adc1>,<adc2>,<adc3>,<adc4>,<adc5>,<adc6>,<adc7>]
```

<TDCPPrefix>にはリクエストコマンドで指定した <TDCPPrefix> 文字列が入ります。

<status> には “force_sample コマンド実行が成功した場合には、”1” が入り、その後にサンプリングデータが続きます。コマンド実行失敗時には “0” が入ります。サンプリングデータの各フィールドの説明は、サンプリングイベントデータと同一です。

- **備考**

“port_read” コマンドは 8bit 幅の値を返します。

“change_detect” コマンドは 8bit 幅の設定値をとります。

“pullup” コマンドは 8bit 幅の設定値をとります。

“adc_read” コマンドは 8 つの A/D 変換値を返します

6.4 app_mode 3

- **機能概要**

PORTA を8bit幅 デジタル出力ポートとして使用する。

PORTC を8bit幅 デジタル入力ポートとして使用する。

- **PORT機能**

PORTA	bit 7-0	デジタル出力ポート
PORTC	bit 7-0	デジタル入力ポート
	(bit 0)	SCL (i2c_use > 0 の場合)
	(bit 1)	SDA (i2c_use > 0 の場合)

- **イベントデータ(CHANGE_DETECT)**

TDCP

```
$$$,CHANGE_DETECT,<my_addr16>,<app_mode>,<diff_bits>,<dio>
```

TDCPZB

```
$$$,CHANGE_DETECT,<app_mode>,<diff_bits>,<dio>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “3” が設定されます。

<diff_bits> には、PORTA とPORTC を合わせた 16bit 幅で、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進数表記で設定されます。

<dio> には PORTA とPORTC を合わせた 16bit 幅の値が 16進数表記で設定されます。

- **サンプリングイベントデータ(SAMPLING)**

TDCP

```
$$$,SAMPLING,<my_addr16>,<app_mode>,<dio>
```

TDCPZB

```
$$$,SAMPLING,<app_mode>,<dio>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “3” が設定されます。

<dio> には PORTA とPORTC を合わせた 16bit 幅の値が 16進数表記で設定されます。

- **イベントデータ(LIVE)**

TDCP

```
$$$,LIVE,<my_addr16>,<app_mode>
```

TDCPZB

```
$$$,LIVE,<app_mode>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “3” が設定されます。

- **force_sample リプライデータ**

```
<TDCPPrefix>,<status>[,<app_mode>,<dio>]
```

<TDCPPrefix>にはリクエストコマンドで指定した <TDCPPrefix> 文字列が入ります。

<status> には “force_sample コマンド実行が成功した場合には、”1” が入り、その後にサンプリングデータが続きます。コマンド実行失敗時には “0” が入ります。サンプリングデータの各フィールドの説明は、サンプリングイベントデータと同一です。

- **備考**

“port_read” コマンドは 16bit 幅の値を返します。

”port_write” コマンドは 8bit 幅の設定値をとります。

“change_detect” コマンドは 16bit 幅の設定値をとります。

“pullup” コマンドは 8bit 幅の設定値をとります。

“port_bit” コマンドのビット位置は 0 から 7 の値を指定します。

6.5 app_mode 4

- **機能概要**

PORTAとPORTC を合わせて 16bit幅 デジタル出力ポートとして使用する。

- **PORT機能**

PORTA	bit 7-0	デジタル出力ポート (High Byte)
PORTC	bit 7-0	デジタル出力ポート (Low Byte)
	(bit 0)	SCL (i2c_use > 0 の場合)
	(bit 1)	SDA (i2c_use > 0 の場合)

- **イベントデータ(CHANGE_DETECT)**

TDCP

```
$$$,CHANGE_DETECT,<my_addr16>,<app_mode>,<diff_bits>,<dio>
```

TDCPZB

```
$$$,CHANGE_DETECT,<app_mode>,<diff_bits>,<dio>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “4” が設定されます。

<diff_bits> には、PORTA とPORTC を合わせた 16bit 幅で、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進数表記で設定されます。

<dio> には PORTA とPORTC を合わせた 16bit 幅の値が 16進数表記で設定されます。

- **サンプリングイベントデータ(SAMPLING)**

TDCP

```
$$$,SAMPLING,<my_addr16>,<app_mode>,<dio>
```

TDCPZB

```
$$$,SAMPLING,<app_mode>,<dio>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “4” が設定されます。

<dio> には PORTA とPORTC を合わせた 16bit 幅の値が 16進数表記で設定されます。

- **イベントデータ(LIVE)**

TDCP

```
$$$,LIVE,<my_addr16>,<app_mode>
```

TDCPZB

```
$$$,LIVE,<app_mode>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “4” が設定されます。

- **force_sample リプライデータ**

```
<TDCPPrefix>,<status>[,<app_mode>,<dio>]
```

<TDCPPrefix>にはリクエストコマンドで指定した <TDCPPrefix> 文字列が入ります。

<status> には “force_sample コマンド実行が成功した場合には、”1” が入り、その後にサンプリングデータが続きます。コマンド実行失敗時には “0” が入ります。サンプリングデータの各フィールドの説明は、サンプリングイベントデータと同一です。

- **備考**

“port_read” コマンドは 16bit 幅の値を返します。

”port_write” コマンドは 16bit 幅の設定値をとります。

“change_detect” コマンドは 16bit 幅の設定値をとります。

“port_bit” コマンドのビット位置は 0 から 15 の値を指定します。

6.6 app_mode 5

- **機能概要**

PORTA を A/D 入力、PORTC をデジタル出力ポートとして使用する。

- **PORT機能**

PORTA	bit 7-0	A/D 入力
PORTC	bit 7-0	デジタル出力ポート
	(bit 0)	SCL (i2c_use > 0 の場合)
	(bit 1)	SDA (i2c_use > 0 の場合)

- **イベントデータ(CHANGE_DETECT)**

TDCP

```
$$$,CHANGE_DETECT,<my_addr16>,<app_mode>,<diff_bits>,<dio>
```

TDCPZB

```
$$$,CHANGE_DETECT,<app_mode>,<diff_bits>,<dio>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “5” が設定されます。

<diff_bits> には、8bit 幅で、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進数表記で設定されます。

<dio> には 8bit 幅の値が 16進数表記で設定されます。

- **イベントデータ(RANGE_EXCEED)**

TDCP

```
$$$,RANGE_EXCEED,<my_addr16>,<app_mode>,<high_exceed_bits>,<low_exceed_bits>
```

TDCPZB

```
$$$,RANGE_EXCEED,<app_mode>,<high_exceed_bits>,<low_exceed_bits>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “5” が設定されます。

<high_exceed_bits> には、8bit 幅で、”range_check_high” コマンドで指定したチェック対象の中で、”range_high” コマンドで指定した上限値を超えた A/D ポートのビットを 1、制限値内のビットを 0にした値が、16進数表記で設定されます。

<low_exceed_bits> には、8bit 幅で、”range_check_low” コマンドで指定したチェック対象の中で、”range_low” コマンドで指定した下限値を超えた A/D ポートのビットを 1、制限値内のビットを 0にした値が、16進数表記で設定されます。

- **サンプリングイベントデータ(SAMPLING)**

TDCP

```
$$$,SAMPLING,<my_addr16>,<app_mode>,<dio>,<adc0>,<adc1>,<adc2>,<adc3>,<adc4>,<adc5>,<adc6>,<adc7>
```

TDCPZB

```
$$$,SAMPLING,<app_mode>,<dio>,<adc0>,<adc1>,<adc2>,<adc3>,<adc4>,<adc5>,<adc6>,<adc7>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “5” が設定されます。

<dio> には 8bit 幅の値が 16進数表記で設定されます。

<adc0>.. <adc7> には、A/D 入力変換値が 10進数表記で設定されます。

- **イベントデータ(LIVE)**

TDCP

```
$$$,LIVE,<my_addr16>,<app_mode>
```

TDCPZB

```
$$$,LIVE,<app_mode>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “5” が設定されます。

- **イベントデータ(ADVAL_UPDATE)**

TDCP

```
$$$,ADVAL_UPDATE,<my_addr16>,<app_mode>,<ad_update_bits>,<adc0>,<adc1>,<adc2>,<adc3>,
<adc4>,<adc5>,<adc6>,<adc7>
```

TDCPZB

```
$$$,ADVAL_UPDATE,<app_mode>,<ad_update_bits>,<adc0>,<adc1>,<adc2>,<adc3>,<adc4>,<adc5>,<adc6>,<adc7>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “5” が設定されます。

<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 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。

例えば、<ad_update_bits> が 03 の場合には adc0, adc1 の両方が、前回 adc0, adc1 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_margin,0,xx”,”ad_margin,1,xx” で設定した値以上に変化したことを示します。

<adc0>.. <adc7> 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。

- **force_sample リプライデータ**

```
<TDCPPrefix>,<status>[,<app_mode>,<dio>,<adc0>,<adc1>,<adc2>,<adc3>,<adc4>,<adc5>,<adc6>,<adc7>]
```

<TDCPPrefix>にはリクエストコマンドで指定した <TDCPPrefix> 文字列が入ります。

<status> には “force_sample コマンド実行が成功した場合には、”1” が入り、その後にサンプリングデータが続きます。コマンド実行失敗時には “0” が入ります。サンプリングデータの各フィールドの説明は、サンプリングイベントデータと同一です。

- **備考**

“port_read” コマンドは 8bit 幅の値を返します。

”port_write” コマンドは 8bit 幅の設定値をとります。

“change_detect” コマンドは 8bit 幅の設定値をとります。

“adc_read” コマンドは 8 つの A/D 変換値を返します

“port_bit” コマンドのビット位置は 0 から 7 の値を指定します。

6.7 app_mode 6

- **機能概要**

  PORTA を A/D 入力で使用する。

  PORTC の上位 4 bitをデジタル出力ポート、下位 4bit をデジタル入力ポートとして使用する。

- **PORT機能**

PORTA	bit 7-0	A/D 入力
PORTC	bit 7-4	デジタル出力ポート
	bit 3-0	デジタル入力ポート
	(bit 0)	SCL (i2c_use > 0 の場合)
	(bit 1)	SDA (i2c_use > 0 の場合)

- **イベントデータ(CHANGE_DETECT)**

TDCP

```
$$$,CHANGE_DETECT,<my_addr16>,<app_mode>,<diff_bits>,<dio>
```

TDCPZB

```
$$$,CHANGE_DETECT,<app_mode>,<diff_bits>,<dio>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “6” が設定されます。

<diff_bits> には、8bit 幅で、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進数表記で設定されます。

<dio> には 8bit 幅の値が 16進数表記で設定されます。

- **イベントデータ(RANGE_EXCEED)**

TDCP

```
$$$,RANGE_EXCEED,<my_addr16>,<app_mode>,<high_exceed_bits>,<low_exceed_bits>
```

TDCPZB

```
$$$,RANGE_EXCEED,<app_mode>,<high_exceed_bits>,<low_exceed_bits>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “6” が設定されます。

<high_exceed_bits> には、8bit 幅で、”range_check_high” コマンドで指定したチェック対象の中で、”range_high” コマンドで指定した上限値を超えた A/D ポートのビットを 1、制限値内のビットを 0にし

た値が、16進数表記で設定されます。

<low_exceed_bits> には、8bit 幅で、”range_check_low” コマンドで指定したチェック対象の中で、”range_low” コマンドで指定した下限値を超えた A/D ポートのビットを 1、制限値内のビットを 0にした値が、16進数表記で設定されます。

- **サンプリングイベントデータ(SAMPLING)**

TDCP

```
$$$,SAMPLING,<my_addr16>,<app_mode>,<dio>,<adc0>,<adc1>,<adc2>,<adc3>,<adc4>,<adc5>,<adc6>,<adc7>
```

TDCPZB

```
$$$,SAMPLING,<app_mode>,<dio>,<adc0>,<adc1>,<adc2>,<adc3>,<adc4>,<adc5>,<adc6>,<adc7>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “6” が設定されます。

<dio> には 8bit 幅の値が 16進数表記で設定されます。

<adc0>.. <adc7> には、A/D 入力変換値が 10 進数表記で設定されます。

- **イベントデータ(LIVE)**

TDCP

```
$$$,LIVE,<my_addr16>,<app_mode>
```

TDCPZB

```
$$$,LIVE,<app_mode>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “6” が設定されます。

- **イベントデータ(ADVAL_UPDATE)**

TDCP

```
$$$,ADVAL_UPDATE,<my_addr16>,<app_mode>,<ad_update_bits>,<adc0>,<adc1>,<adc2>,<adc3>,
<adc4>,<adc5>,<adc6>,<adc7>
```

TDCPZB

```
$$$,ADVAL_UPDATE,<app_mode>,<ad_update_bits>,<adc0>,<adc1>,<adc2>,<adc3>,<adc4>,<adc5>,<adc6>,<adc7>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “6” が設定されます。

<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 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。

例えば、<ad_update_bits> が 03 の場合には adc0, adc1 の両方が、前回 adc0, adc1 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_margin,0,xx”,”ad_margin,1,xx” で設定した値以上に変化したことを示します。

<adc0>.. <adc7> 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。

- **force_sample リプライデータ**

<TDCPPrefix>,<status>[,<app_mode>,<dio>,<adc0>,<adc1>,<adc2>,<adc3>,<adc4>,<adc5>,<adc6>,<adc7>]

<TDCPPrefix>にはリクエストコマンドで指定した <TDCPPrefix> 文字列が入ります。

<status> には “force_sample コマンド実行が成功した場合には、”1” が入り、その後にサンプリングデータが続きます。コマンド実行失敗時には “0” が入ります。サンプリングデータの各フィールドの説明は、サンプリングイベントデータと同一です。

- **備考**

“port_read” コマンドは 8bit 幅の値を返します。

”port_write” コマンドは 4bit 幅の設定値をとります。

“change_detect” コマンドは 8bit 幅の設定値をとります。

“pullup” コマンドは 4bit 幅の設定値をとります。

“adc_read” コマンドは 8 つの A/D 変換値を返します。

“port_bit” コマンドのビット位置は 0 から 3 の値を指定します。(注意：4 から 7 ではありません)

6.8 app_mode 7

- **機能概要**

PORTA の上位 4 bitをシグナル出力(ブザー、LED 赤、LED 黄、LED 緑)として使用する。

PORTA の下位 4 bitをデジタル出力ポートとして使用する。

PORTC を8bit幅 デジタル入力ポートとして使用する。その中で 上位 4 bit をカウンタ入力として使用して、下位 4 bit は通常のデジタル入力ポートとして使用する。

- **PORT機能**

PORTA	bit 7	シグナル ブザー出力
	bit 6	シグナル LED 緑 出力
	bit 5	シグナル LED 黄 出力
	bit 4	シグナル LED 赤 出力
	bit 3-0	デジタル出力ポート
PORTC	bit 7-4	カウンタ入力ポート

	bit 3-0	デジタル入力ポート
	(bit 0)	SCL (i2c_use > 0 の場合)
	(bit 1)	SDA (i2c_use > 0 の場合)

- **イベントデータ(CHANGE_DETECT)**

TDCP

```
$$$,CHANGE_DETECT,<my_addr16>,<app_mode>,<diff_bits>,<dio>
```

TDCPZB

```
$$$,CHANGE_DETECT,<app_mode>,<diff_bits>,<dio>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “7” が設定されます。

<diff_bits> には、PORTA とPORTC を合わせた 16bit 幅で、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進数表記で設定されます。ただし、PORTA の上位4bit と PORTC の上位 4 ビットは CHANGE_DETECT イベントの検出から除外されるため、常に “0” が設定されます

<dio> には PORTA とPORTC を合わせた 16bit 幅の値が 16進数表記で設定されます。ただし、PORTA の上位 4 bitは常に “0” が読みこまれます。

- **イベントデータ(COUNT_EXCEED)**

TDCP

```
$$$,COUNT_EXCEED,<my_addr16>,<app_mode>,<change_count>
```

TDCPZB

```
$$$,COUNT_EXCEED,<app_mode>,<change_count>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “7” が設定されます。

<change_count> には、現在のカウンタ値が10進数表記で設定されます。

- **サンプリングイベントデータ(SAMPLING)**

TDCP

```
$$$,SAMPLING,<my_addr16>,<app_mode>,<dio>,<change_count>
```

TDCPZB

```
$$$,SAMPLING,<app_mode>,<dio>,<change_count>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “7” が設定されます。

<dio> には PORTA とPORTC を合わせた 16bit 幅の値が 16進数表記で設定されます。ただし、PORTA の上位 4 bitは常に “0” が読みこまれます。

<change_count> には、イベント発生時のカウンタ値が10進数表記で設定されます。イベント発生直後にこのカウンタ値はクリア (0 を設定)されます。

- **イベントデータ(LIVE)**

TDCP

```
$$$,LIVE,<my_addr16>,<app_mode>
```

TDCPZB

```
$$$,LIVE,<app_mode>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “7” が設定されます。

- **force_sample リプライデータ**

```
<TDCPPrefix>,<status>[,<app_mode>,<dio>,<change_count>]
```

<TDCPPrefix>にはリクエストコマンドで指定した <TDCPPrefix> 文字列が入ります。

<status> には “force_sample コマンド実行が成功した場合には、”1” が入り、その後にサンプリングデータが続きます。コマンド実行失敗時には “0” が入ります。サンプリングデータの各フィールドの説明は、サンプリングイベントデータと同一です。

- **備考**

シグナル状態は PORTA の bit4-6(シグナル LED) に以下の値を出力します。

ON(High) または OFF(Low)

点滅(High-Low 繰り返し)

また、PORTA の bit7(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-....

これらのシグナル状態の取得や設定には “signal_read”,”signal_write” コマンドを使用します。

“port_read” コマンドは 16bit 幅の値を返します。

”port_write” コマンドは 4bit 幅の設定値をとります。

“change_detect” コマンドは 12bit 幅の設定値をとります。

“pullup” コマンドは 8bit 幅の設定値をとります。

“port_bit” コマンドのビット位置は 0 から 3 の値を指定します。

6.9 app_mode 8

- **機能概要**

  PORTA の上位 4 bitをシグナル出力(ブザー、LED 赤、LED 黄、LED 緑)として使用する。

  PORTA の下位 4 bitをA/D 入力で使用する。

  PORTC を8bit幅 デジタル入力ポートとして使用する。その中で 上位 4 bit をカウンタ入力として使用して、下位 4 bit は通常のデジタル入力ポートとして使用する。

- **PORT機能**

PORTA	bit 7	シグナル ブザー出力
	bit 6	シグナル LED 緑 出力
	bit 5	シグナル LED 黄 出力
	bit 4	シグナル LED 赤 出力
	bit 3-0	A/D 入力
PORTC	bit 7-4	カウンタ入力ポート
	bit 3-0 (bit 0) (bit 1)	デジタル入力ポート SCL (i2c_use > 0 の場合) SDA (i2c_use > 0 の場合)

- **イベントデータ(CHANGE_DETECT)**

  TDCP

  ```
  $$$,CHANGE_DETECT,<my_addr16>,<app_mode>,<diff_bits>,<dio>
  ```

  TDCPZB

  ```
  $$$,CHANGE_DETECT,<app_mode>,<diff_bits>,<dio>
  ```

  <my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

  <app_mode> は “8” が設定されます。

  <diff_bits> には、PORTC の 8bit 幅で、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進数表記で設定されます。ただし、PORTC の上位 4 ビットは CHANGE_DETECT イベントの検出から除外されるため、常に “0” が設定されます

  <dio> には PORTC の 8bit 幅の値が 16進数表記で設定されます。

- **イベントデータ(COUNT_EXCEED)**

  TDCP

  ```
  $$$,COUNT_EXCEED,<my_addr16>,<app_mode>,<change_count>
  ```

TDCPZB

```
$$$,COUNT_EXCEED,<app_mode>,<change_count>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “8” が設定されます。

<change_count> には、現在のカウンタ値が10進数表記で設定されます。

- **イベントデータ(RANGE_EXCEED)**

TDCP

```
$$$,RANGE_EXCEED,<my_addr16>,<app_mode>,<high_exceed_bits>,<low_exceed_bits>
```

TDCPZB

```
$$$,RANGE_EXCEED,<app_mode>,<high_exceed_bits>,<low_exceed_bits>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “8” が設定されます。

<high_exceed_bits> には、8bit 幅で、”range_check_high” コマンドで指定したチェック対象の中で、”range_high” コマンドで指定した上限値を超えた A/D ポートのビットを 1、制限値内のビットを 0にした値が、16進数表記で設定されます。ただし、上位 4 ビットは 常に “0” が設定されます。

<low_exceed_bits> には、8bit 幅で、”range_check_low” コマンドで指定したチェック対象の中で、”range_low” コマンドで指定した下限値を超えた A/D ポートのビットを 1、制限値内のビットを 0にした値が、16進数表記で設定されます。ただし、上位 4 ビットは 常に “0” が設定されます。

- **サンプリングイベントデータ(SAMPLING)**

TDCP

```
$$$,SAMPLING,<my_addr16>,<app_mode>,<dio>,<change_count>,<adc0>,<adc1>,<adc2>,<adc3>
```

TDCPZB

```
$$$,SAMPLING,<app_mode>,<dio>,<change_count>,<adc0>,<adc1>,<adc2>,<adc3>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “8” が設定されます。

<dio> には PORTA とPORTC を合わせた 16bit 幅の値が 16進数表記で設定されます。ただし、PORTA の上位 4 bitは常に “0” が読みこまれます。

<change_count> には、イベント発生時のカウンタ値が10進数表記で設定されます。イベント発生直後にこのカウンタ値はクリア (0 を設定)されます。

<adc0>.. <adc3> には、A/D 入力変換値が 10 進数表記で設定されます。

- **イベントデータ(LIVE)**

TDCP

```
$$$,LIVE,<my_addr16>,<app_mode>
```

TDCPZB

```
$$$,LIVE,<app_mode>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “8” が設定されます。

- **イベントデータ(ADVAL_UPDATE)**

TDCP

```
$$$,ADVAL_UPDATE,<my_addr16>,<app_mode>,<ad_update_bits>,<adc0>,<adc1>,<adc2>,<adc3>
```

TDCPZB

```
$$$,ADVAL_UPDATE,<app_mode>,<ad_update_bits>,<adc0>,<adc1>,<adc2>,<adc3>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “8” が設定されます。

<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。上位 4 ビットは 常に “0” が設定されます。

例えば、<ad_update_bits> が 03 の場合には adc0, adc1 の両方が、前回 adc0, adc1 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_margin,0,xx”,”ad_margin,1,xx” で設定した値以上に変化したことを示します。

<adc0>.. <adc3> 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。

- **force_sample リプライデータ**

```
<TDCPPrefix>,<status>[,<app_mode>,<dio>,<change_count>,<adc0>,<adc1>,<adc2>,<adc3>]
```

<TDCPPrefix>にはリクエストコマンドで指定した <TDCPPrefix> 文字列が入ります。

<status> には “force_sample コマンド実行が成功した場合には、”1” が入り、その後にサンプリングデータが続きます。コマンド実行失敗時には “0” が入ります。サンプリングデータの各フィールドの説明は、サンプリングイベントデータと同一です。

- **備考**

シグナル状態は PORTA の bit4-6(シグナル LED) に以下の値を出力します。

ON(High) または OFF(Low)

点滅(High-Low 繰り返し)

また、PORTA の bit7(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-....

これらのシグナル状態の取得や設定には “signal_read”,”signal_write” コマンドを使用します。

“port_read” コマンドは 8bit 幅の値を返します。

“change_detect” コマンドは 8bit 幅の設定値をとります。

“pullup” コマンドは 8bit 幅の設定値をとります。

“adc_read” コマンドは 4 つの A/D 変換値を返します。

6.10 app_mode 9

- **機能概要**

PORTA の上位 4 bitをシグナル出力(ブザー、LED 赤、LED 黄、LED 緑)として使用する。

PORTA の下位 4 bitをデジタル入力ポートとして使用する。

PORTC を HD44780互換チップを使用したLCD 表示モジュール接続用に使用する。LCD 表示モジュールは、4 ビットインターフェースモードで動作させる。

- **PORT機能**

PORTA	bit 7	シグナル ブザー出力
	bit 6	シグナル LED 緑 出力
	bit 5	シグナル LED 黄 出力
	bit 4	シグナル LED 赤 出力
	bit 3-0	デジタル入力ポート
PORTC	bit 7	無接続
	bit 6	E (LCD表示モジュール ピン名)
	bit 5	R/W (LCD表示モジュール ピン名)
	bit 4	RS (LCD表示モジュール ピン名)
	bit 3	DB7 (LCD表示モジュール ピン名)
	bit 2	DB6 (LCD表示モジュール ピン名)
	bit 1	DB5 (LCD表示モジュール ピン名)
	bit 0	DB4 (LCD表示モジュール ピン名)

注意

この app_mode を使用する場合には、″i2c_use,0″ コマンドを使用して PORTC bit0, bit1 を I2Cバスで使用しないように設定してください。

- **イベントデータ(CHANGE_DETECT)**

TDCP

```
$$$,CHANGE_DETECT,<my_addr16>,<app_mode>,<diff_bits>,<dio>
```

TDCPZB

```
$$$,CHANGE_DETECT,<app_mode>,<diff_bits>,<dio>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “9” が設定されます。

<diff_bits> には、PORTA の 下位 4bit 幅で、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進数表記で設定されます。

<dio> には PORTA の 下位 4bit 幅の値が 16進数表記で設定されます。

- **サンプリングイベントデータ(SAMPLING)**

TDCP

```
$$$,SAMPLING,<my_addr16>,<app_mode>,<dio>
```

TDCPZB

```
$$$,SAMPLING,<app_mode>,<dio>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “9” が設定されます。

<dio> には PORTA の 下位 4bit 幅の値が 16進数表記で設定されます。

- **イベントデータ(LIVE)**

TDCP

```
$$$,LIVE,<my_addr16>,<app_mode>
```

TDCPZB

```
$$$,LIVE,<app_mode>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “9” が設定されます。

- **force_sample リプライデータ**

```
<TDCPPrefix>,<status>[,<app_mode>,<dio>]
```

<TDCPPrefix>にはリクエストコマンドで指定した <TDCPPrefix> 文字列が入ります。

<status> には “force_sample コマンド実行が成功した場合には、”1” が入り、その後にサンプリングデータが続きます。コマンド実行失敗時には “0” が入ります。サンプリングデータの各フィールドの説明は、サンプリングイベントデータと同一です。

- **備考**

シグナル状態は PORTA の bit4-6(シグナル LED) に以下の値を出力します。

ON(High) または OFF(Low)

点滅(High-Low 繰り返し)

また、PORTA の bit7(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-....

これらのシグナル状態の取得や設定には “signal_read”,”signal_write” コマンドを使用します。

“port_read” コマンドは 4bit 幅の値を返します。

“change_detect” コマンドは 4bit 幅の設定値をとります。

“pullup” コマンドは 4bit 幅の設定値をとります。

6.11 app_mode 10

- **機能概要**

PORTA の上位 4 bitをシグナル出力(ブザー、LED 赤、LED 黄、LED 緑)として使用する。

PORTA の下位 4 bitをA/D 入力で使用する。

PORTC を HD44780互換チップを使用したLCD 表示モジュール接続用に使用する。LCD 表示モジュールは、4 ビットインターフェースモードで動作させる。

- **PORT機能**

PORTA	bit 7	シグナル ブザー出力
	bit 6	シグナル LED 緑 出力
	bit 5	シグナル LED 黄 出力
	bit 4	シグナル LED 赤 出力
	bit 3-0	A/D 入力
PORTC	bit 7	無接続
	bit 6	E (LCD表示モジュール ピン名)
	bit 5	R/W (LCD表示モジュール ピン名)
	bit 4	RS (LCD表示モジュール ピン名)
	bit 3	DB7 (LCD表示モジュール ピン名)
	bit 2	DB6 (LCD表示モジュール ピン名)

	bit 1	DB5　(LCD表示モジュール ピン名)
	bit 0	DB4　(LCD表示モジュール ピン名)

注意

この app_mode を使用する場合には、"i2c_use,0" コマンドを使用して PORTC bit0, bit1 を I2Cバスで使用しないように設定してください。

- **イベントデータ(RANGE_EXCEED)**

TDCP

```
$$$,RANGE_EXCEED,<my_addr16>,<app_mode>,<high_exceed_bits>,<low_exceed_bits>
```

TDCPZB

```
$$$,RANGE_EXCEED,<app_mode>,<high_exceed_bits>,<low_exceed_bits>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。
<app_mode> は “10” が設定されます。
<high_exceed_bits> には、8bit 幅で、”range_check_high” コマンドで指定したチェック対象の中で、”range_high” コマンドで指定した上限値を超えた A/D ポートのビットを 1、制限値内のビットを 0にした値が、16進数表記で設定されます。ただし、上位 4 ビットは 常に “0” が設定されます。
<low_exceed_bits> には、8bit 幅で、”range_check_low” コマンドで指定したチェック対象の中で、”range_low” コマンドで指定した下限値を超えた A/D ポートのビットを 1、制限値内のビットを 0にした値が、16進数表記で設定されます。ただし、上位 4 ビットは 常に “0” が設定されます。

- **サンプリングイベントデータ(SAMPLING)**

TDCP

```
$$$,SAMPLING,<my_addr16>,<app_mode>,<adc0>,<adc1>,<adc2>,<adc3>
```

TDCPZB

```
$$$,SAMPLING,<app_mode>,<adc0>,<adc1>,<adc2>,<adc3>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。
<app_mode> は “10” が設定されます。
<adc0>.. <adc3> には、A/D 入力変換値が 10 進数表記で設定されます。

- **イベントデータ(LIVE)**

TDCP

```
$$$,LIVE,<my_addr16>,<app_mode>
```

TDCPZB

```
$$$,LIVE,<app_mode>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “10” が設定されます。

- **イベントデータ(ADVAL_UPDATE)**

TDCP

```
$$$,ADVAL_UPDATE,<my_addr16>,<app_mode>,<ad_update_bits>,<adc0>,<adc1>,<adc2>,<adc3>
```

TDCPZB

```
$$$,ADVAL_UPDATE,<app_mode>,<ad_update_bits>,<adc0>,<adc1>,<adc2>,<adc3>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は “10” が設定されます。

<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。上位 4 ビットは 常に “0” が設定されます。

例えば、<ad_update_bits> が 03 の場合には adc0, adc1 の両方が、前回 adc0, adc1 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_margin,0,xx”,”ad_margin,1,xx” で設定した値以上に変化したことを示します。

<adc0>.. <adc3> 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。

- **force_sample リプライデータ**

```
<TDCPPrefix>,<status>[,<app_mode>,<adc0>,<adc1>,<adc2>,<adc3>]
```

<TDCPPrefix>にはリクエストコマンドで指定した <TDCPPrefix> 文字列が入ります。

<status> には “force_sample コマンド実行が成功した場合には、”1” が入り、その後にサンプリングデータが続きます。コマンド実行失敗時には “0” が入ります。サンプリングデータの各フィールドの説明は、サンプリングイベントデータと同一です。

- **備考**

シグナル状態は PORTA の bit4-6(シグナル LED) に以下の値を出力します。

ON(High) または OFF(Low)

点滅(High-Low 繰り返し)

また、PORTA の bit7(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-....

これらのシグナル状態の取得や設定には “signal_read”,”signal_write” コマンドを使用します。

“adc_read” コマンドは 4 つの A/D 変換値を返します。

7 TDCPコマンドリファレンス

TDCP で使用可能なコマンド一覧です。一部のコマンドを除いて殆どのコマンドが、リモートから XBeeデータパケットに埋め込みで実行させる方法と、TDCP シリアルコンソールポートで実行する方法の、２種類の方法で実行可能です。リモートからコマンドを実行した場合に、コマンド実行結果の値はリモートのコマンド送信元 XBee デバイスにデータパケット経由で送信されます。

現在 動作中の “app_mode” によって一部のコマンドが使用できない場合があります。例えば、全てのポートがデジタル入力の “app_mode=1” の時の “adc_read”(A/D 変換入力コマンド)等。また、”app_mode” によってコマンドパラメータの値(ビット数幅) が変化する場合がありますので、”TDCP動作モード” の章も併せて参照して下さい。”pullup” コマンドや “port_write”,”port_read” 等。

コマンドリファレンス中のパラメータ <TDCPPrefix> とリプライデータ中の<status> パラメータについては、”TDCP通信仕様” の章も併せて参照して下さい。

リクエストフォーマットで記述した、下記の表の内容の意味は下記になります。

Console	シリアルコンソール経由でコマンドを送信、リプライを受信する場合のフォーマットを表します
XBee	リモートから XBee データパケットに入れたリクエストコマンドデータを送信、同じく XBee データパケットに入ったリプライデータを受信するときのフォーマットを表します

また、上記の表中のリクエストフォーマットに記述した [ ] で囲んだ部分はオプション文字列で省略可能であることを示します。リプライデータフォーマットに記述した [ ] で囲んだ部分はリクエストコマンドまたは実行結果によってリプライデータに含まれない場合があることを示します。

注意

TDCP ファームウエアを使用する場合には、XBee データパケットで送信する場合の最大データサイズは XBee モジュールの仕様によって、100 bytes までに制限されています。<TDCPPrefix>とコマンド、パラメータの全てを合わせた長さがこの値を越えないように注意してください。

TDCPZB ファームウエアを使用する場合には、XBee-ZB デバイスの仕様によって255バイトまでに制限されます。フラグメンテーションによって複数回パケットが送信されるのを防ぐためには 84バイト、"Encryption enabled" 指定時には66バイト以内にしてください。

注意

下記のコマンドは、コマンド実行後に config_saveコマンドを実行してEEPROM に最新のコンフィギュレーションを保存した後、”reset” コマンド実行後に有効になります。

7.1 help

- **機能概要**

  TDCP コマンド一覧をシリアルコンソールに表示する

- **リクエストフォーマット**

Console	Help
XBee	** Not available **

- **リプライデータフォーマット**

Console	<CommandList>
XBee	** Not available **

- **パラメータとリターン値**

  <CommandList>　　TDCP コマンド一覧

- **備考**

  このコマンドは XBee リモートデータパケット経由では使用できません。

7.2 version

- **機能概要**

  TDCP プログラムバージョン番号を取得

- **リクエストフォーマット**

Console	version
XBee	<TDCPPrefix>,version

- **リプライデータフォーマット**

Console	<TDCPVersionLongText>
XBee	<TDCPPrefix>,<status>,<TDCPVersion>

- **パラメータとリターン値**

  <TDCPPrefix>　　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<TDCPVersionLongText>　TDCP プログラムバージョン、シリアルコンソール表示用。

<TDCPVersion>　TDCP プログラムバージョン(XX.XX形式で表示)

<status>　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$aaa,version” -> “$$$aaa,1,1.00”

7.3 reset

- **機能概要**

CPU のリセット

- **リクエストフォーマット**

Console	reset
XBee	<TDCPPrefix>,reset

- **リプライデータフォーマット**

Console	** Not available **
XBee	** Not available **

- **パラメータとリターン値**

<TDCPPrefix>　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

- **備考**

リセットコマンドを実行すると CPU がリセットされます。TDCP プログラムは再起動時に、EEPROM に最後に保存したコンフィギュレーションを自動的にロードします。

<TDCPPrefix>の内容に関わらず常に、reset コマンドのリプライは返りません。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$,reset” -> ** No reply data **
“$$$abc,reset” -> ** No reply data **

7.4 server_addr

- **機能概要**

  TDCP イベントデータとサンプリングデータ送信先の XBee アドレス(16bit または64bit幅)の設定または取得

  “config_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**

  TDCP

Console	server_addr[,<16BitAddrHexStr|64BitAddrHexStr>]
XBee	<TDCPPrefix>,server_addr[,<16BitAddrHexStr|64BitAddrHexStr>]

  TDCPZB

Console	server_addr[,<64BitAddrHexStr>]
XBee	<TDCPPrefix>,server_addr[,<64BitAddrHexStr>]

- **リプライデータフォーマット**

  TDCP

Console	server_addr=<16BitAddrHexStr|64BitAddrHexStr>
XBee	<TDCPPrefix>,<status>[,<16BitAddrHexStr|64BitAddrHexStr>]

  TDCP

Console	server_addr=<64BitAddrHexStr>
XBee	<TDCPPrefix>,<status>[,<64BitAddrHexStr>]

- **パラメータとリターン値**

  <TDCPPrefix>　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

  <16BitAddrHexStr|64BitAddrHexStr>

  TDCP イベントデータとサンプリングデータ送信先の XBeeアドレスを16bitまたは64bit幅の16進数形式の文字列で記述する(例: “0013A200404AC398”, “0A01”)

  TDCPZB の場合は 64bit幅のみ指定できます。

  ブロードキャストアドレスを設定する場合には ”000000000000FFFF”を指定する

  TDCP (デフォルト設定値:””)

  TDCPZB (デフォルト設定値:”0000000000000000”)

  <status>　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

  リクエストコマンド(XBee データパケット)で <16BitAddrHexStr|64BitAddrHexStr>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。この時、アドレス値が未設定の場合にはリプライデータの<16BitAddrHexStr|64BitAddrHexStr >に “0” <status> に “1” が返ります。

<16BitAddrHexStr|64BitAddrHexStr> パラメータを指定した場合にはリプライデータには <TDCPPrefix>と <status> だけが入ります。

<16BitAddrHexStr|64BitAddrHexStr>パラメータに ‘0’ を指定した場合(“server_addr,0” と入力) には、データ送信先アドレスが未設定の状態になります。

“server_addr” コマンドを使用して、データ送信先アドレスが設定されていない状態では、TDCP イベントデータやサンプリングデータの送信機能は使用できません。

TDCPZB の場合には、デフォルト設定値は Coordinator を示す ″0000000000000000″ が設定されていますので、通常この値を変更する必要はありません。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,server_addr,0013A200404AC398” -> “$$$abc,1”
“$$$,server_addr,0013A200404AC398” -> ** No reply data **
“$$$12345,server_addr” -> “$$$12345,1,0013A200404AC398”
“$$$,server_addr” -> ** No reply data **

7.5 interval

- **機能概要**

TDCP の内部タスク(I/O ポートや A/D 入力値の監視等)を実行する時間間隔の設定または取得

“config_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**

Console	interval[,<timer_interval>]
XBee	<TDCPPrefix>,interval[,<timer_interval>]

- **リプライデータフォーマット**

Console	timer_interval=<timer_interval>
XBee	<TDCPPrefix>,<status>[,<timer_interval>]

- **パラメータとリターン値**

<TDCPPrefix> “$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<timer_interval> インターバル時間を設定する(ms)。0 以上の整数を指定する。

(デフォルト設定値:10)

<status>　　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

ここで設定した値は、イベント発生条件の監視や内部のタスク処理タイミングに使用されます。 通常は default 値 10(ms) から変更する必要はありません。

リクエストコマンド(XBee データパケット)で <timer_interval>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。<timer_interval> パラメータを指定した場合にはリプライデータには <TDCPPrefix>と<status> だけが入ります。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,inteval,10” -> “$$$abc,1”
“$$$12345,interval” -> “$$$12345,1,10”

7.6 config_save

- **機能概要**

現在のコンフィギュレーション設定値を プロセッサの EEPROM に保存する

- **リクエストフォーマット**

Console	config_save
XBee	<TDCPPrefix>,config_save

- **リプライデータフォーマット**

Console	** No reply data **
XBee	<TDCPPrefix>,<status>

- **パラメータとリターン値**

<TDCPPrefix>　　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<status>　　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

保存対象となる設定項目については、コマンド毎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”

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,config_save” -> “$$$abc,1”
“$$$,config_save” -> ** No reply data **

7.7 config_load

- **機能概要**

  プロセッサの EEPROM に保存されているコンフィギュレーション値をロードする

- **リクエストフォーマット**

Console	config_load
XBee	<TDCPPrefix>,config_load

- **リプライデータフォーマット**

Console	** No reply data **
XBee	<TDCPPrefix>,<status>

- **パラメータとリターン値**

  <TDCPPrefix>　　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

  <status>　　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

  現在のコンフィギュレーション設定値を破棄して、最後に “config_save” コマンドで保存した設定値をロードする。

  ロード対象となる設定項目については、各コマンド中の説明を参照してください。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,config_load” -> “$$$abc,1”
“$$$,config_load” -> ** No reply data **

7.8 config_clear

- **機能概要**

プロセッサの EEPROM に保存されているコンフィギュレーション値を初期化する

- **リクエストフォーマット**

Console	config_clear
XBee	<TDCPPrefix>,config_clear

- **リプライデータフォーマット**

Console	** No reply data **
XBee	<TDCPPrefix>,<status>

- **パラメータとリターン値**

<TDCPPrefix>　　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<status>　　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

“config_save” コマンドで保存した設定値をクリアするのみで現在の設定値は変更しません。デフォルト設定値に戻すためには、この後に続けて “reset” コマンドを実行してTDCP プログラムをリセットして下さい。

TDCP プログラム起動時(CPU リセット直後) に PORTD bit#7を GND に接続しておくことで、”config_clear” コマンドと同様に設定値をクリアすることができます。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,config_clear” -> “$$$abc,1”

“$$$,config_clear” -> ** No reply data **

7.9 uart0_baud

- **機能概要**

TDCP のシリアルポート#0(コンソールポート) ボーレートの設定または取得

“config_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**

Console	uart0_baud[,<4800|9600|19200>]
XBee	<TDCPPrefix>,uart0_baud[,<4800|9600|19200>]

- **リプライデータフォーマット**

Console	uart0_baud=<4800|9600|19200>
XBee	<TDCPPrefix>,<status>[,<4800|9600|19200>]

- **パラメータとリターン値**

<TDCPPrefix>　　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<4800|9600|19200>　　ボーレートを指定する(4800,9600,19200 のいずれかを指定する)

（デフォルト設定値:9600)

<status>　　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

コマンド実行直後に設定したボーレートに変更されますので、シリアルコンソールから実行する場合は端末側のボーレートの設定変更も必要になります。

GPS レシーバ接続機能を使用する場合には、レシーバの NMEA-0183 センテンス送信に使用されるボーレートをここで設定します。ただし、GPS 接続に使用する場合には 4800 または 9600 のみ指定できます(19200は使用できません)

リクエストコマンド(XBee データパケット)で <4800|9600|19200>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。<4800|9600|19200> パラメータを指定した場合にはリプライデータには <TDCPPrefix>と<status> だけが入ります。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,uart0_baud,4800” -> “$$$abc,1”
“$$$12345,uart0_baud” -> “$$$12345,1,9600”

7.10　uart0_echo

- **機能概要**

TDCP のシリアルポート#0(コンソールポート) ローカルエコーモードの設定または取得

“config_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**

Console	uart0_echo[,<flag>]
XBee	<TDCPPrefix>,uart0_echo[,<flag]

- **リプライデータフォーマット**

Console	uart0_echo=<flag>
XBee	<TDCPPrefix>,<status>[,<flag>]

- **パラメータとリターン値**

<TDCPPrefix>　　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<flag>　　“0” または “1” の値を指定する。

“0” でローカルエコー無し、 1 でローカルエコー有。

（デフォルト設定値:1）

<status>　　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

コマンド実行直後に設定したローカルエコーモードに変更されますので、シリアルコンソールから実行する場合は端末側の設定変更も必要になります。

ローカルエコー無しに設定すると、端末編集文字の処理や、コマンドプロンプト出力、エラーメッセージのシリアルコンソールへの表示も停止します。

GPS レシーバ接続機能を使用する場合には、必ずローカルエコー ”無し” に設定してください。

リクエストコマンド(XBee データパケット)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。<flag> パラメータを指定した場合にはリプライデータには <TDCPPrefix>と<status> だけが入ります。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,uart0_echo,0” -> “$$$abc,1”

“$$$12345,uart0_echo” -> “$$$12345,1,0”

7.11　sampling_rate

- **機能概要**

自動サンプリング間隔の設定または取得

“config_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**

Console	sampling_rate[,<rate>]

XBee	<TDCPPrefix>,sampling_rate[,<rate>]

- **リプライデータフォーマット**

Console	sampling_rate=<rate>
XBee	<TDCPPrefix>,<status>[,<rate>]

- **パラメータとリターン値**

<TDCPPrefix>　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<rate>　自動サンプリング間隔(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 自動サンプリングを行わない。
(デフォルト設定値:0)

<status>　“1” でコマンド実行成功、”0” で失敗を表す

- **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。

設定したサンプリング間隔毎に “SAMPLING” イベントが発生して、”server_addr” コマンドで設定したデバイスに対してサンプリングイベントデータを送信します。サンプリングイベントデータの内容は、TDCP の現在の “app_mode” によって変わります。

<rate> で設定した秒間隔で “SAMPLING” 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。また、TDCP ではある程度(1/1000秒)までの秒補正を “second_adjust” コマンドで設定することが可能ですので、詳しくは ”コマンドリファレンス” の章の “second_adjust” コマンドの項を参照してください。ただしこの場合でも TDCP で実行中のタスクの状況によっては、”SAMPLING”イベント送信が遅れる(10ms .. 数秒程度)場合があります。

リクエストコマンド(XBee データパケット)で <rate>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。<rate> パラメータを指定した場合にはリプライデータには <TDCPPrefix>と<status> だけが入ります。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,sampling_rate,3600” -> “$$$abc,1”

“$$$12345,sampling_rate” -> “$$$12345,1,0”

7.12 heartbeat_rate

- **機能概要**

  LIVE イベント発生間隔の設定または取得

  “config_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**

Console	heartbeat_rate[,<rate>]
XBee	<TDCPPrefix>,heartbeat_rate[,<rate>]

- **リプライデータフォーマット**

Console	heartbeat_rate=<rate>
XBee	<TDCPPrefix>,<status>[,<rate>]

- **パラメータとリターン値**

<TDCPPrefix>	“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列
<rate>	LIVE イベント発生間隔(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 LIVE イベントは発生しない。 (デフォルト設定値:0)
<status>	“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

  <rate> に 0 を設定した場合は、LIVE イベント送信を行いません。

  設定したイベント発生間隔毎に ”LIVE”イベントが発生して、”server_addr” コマンドで設定したデバイスに対してイベントデータを送信します。

  heartbeat_rate コマンド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。また、TDCP ではある程度(1/1000 秒)までの秒補正を “second_adjust” コマンドで設定することが可能ですので、詳しくは ”コマンドリファレンス” の章の “second_adjust” コマンドの項を参照してください。ただしこの場合でも TDCP で実行中のタスクの状況によっては、LIVEイベント送信が遅れる(10ms .. 数秒程度)場合があります。

  リクエストコマンド(XBee データパケット)で <rate>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。<rate> パラメータを指定した場合にはリプライデータには <TDCPPrefix>と<status> だけが入ります。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,heartbeat_rate,60” -> “$$$abc,1”

7.13 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**

Console	force_sample
XBee	<TDCPPrefix>,force_sample

- **リプライデータフォーマット**

Console	sample=<sampling_data>
XBee	<TDCPPrefix>,<status>[,<sampling_data>]

- **パラメータとリターン値**

<TDCPPrefix>	“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列
<sampling_data>	TDCP の現在の app_mode によって、予め決められたサンプリングデータ項目が入る。(“TDCP動作モード” の章中の “force_sample リプライデータ”の項目を参照して下さい)
<status>	“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

  XBee データパケットからコマンドを実行する場合に、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。リプライデータパケット中に、コマンド実行ステータスとサンプリングデータが含まれますので <TDCPPrefix>には、必ず “$$$” に続けて任意の文字列を指定してください。

  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(“change_count_check”,”change_count_reset”,”change_count_high” コマンド参照の事) はクリアされません。(sampling_rate コマンドで設定した間隔毎に発生するサンプリングイベント時には、カウンタは 0 に戻されます)

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,force_sample” -> “$$$abc,1,8,FF,58,150,118,86,541”
“$$$,force_sample” -> ** No reply data **

7.14 sample_once

- **機能概要**

  サンプリングイベントを強制的に一回だけ発生させてサンプリングイベントパケットの送信を指示するフラグの設定または取得。

- **リクエストフォーマット**

Console	sample_once[,<flag>]
XBee	<TDCPPrefix>, sample_once[,<flag>]

- **リプライデータフォーマット**

Console	sample_once=<flag>
XBee	<TDCPPrefix>,<status>[,<flag>]

- **パラメータとリターン値**

<TDCPPrefix>	“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列
<flag>	イベント送信を行う場合に “1” 、行わない場合に“0”を指定する。 (デフォルト設定値:0)
<status>	“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

  “sample_once” の設定を ”1” にすると、sampling_rate コマンドで設定した定期的なサンプリングイベントとは別に、強制的にサンプリングイベントを発生させて、”server_addr” コマンドで設定したデバイスに対してイベントデータを送信します。

  サンプリングイベント発生と同時に、”sample_once” は “0” にセットされます。

  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のタイミングには影響しませんが、サンプリングデータ中のカウンタ値はクリアされますので注意してください。
  (“change_count_check”,”change_count_reset”,”change_count_high” コマンド参照の事)

  リクエストコマンド(XBee データパケット)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。<flag> パラメータを指定した場合にはリプライデータには <TDCPPrefix>と<status> だけが入ります。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,sample_once,1” -> “$$$abc,1”
“$$$12345,sample_once” -> “$$$12345,1,1”

7.15 second_adjust

- **機能概要**

  CPU クロックからTDCP 内部の ”秒” をカウントする時の ”ミリ秒数” の設定または取得

  “config_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**

Console	second_adjust[,<ratio>]
XBee	<TDCPPrefix>,second_adjust[,<ratio>]

- **リプライデータフォーマット**

Console	second_adjust=<ratio>
XBee	<TDCPPrefix>,<status>[,<ratio>]

- **パラメータとリターン値**

<TDCPPrefix>	“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列
<ratio>	秒あたりのミリ秒数を設定する （デフォルト設定値:1000)
<status>	“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

  TDCP 内部のクロックを元にミリ秒をカウントしていますが、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。

  リクエストコマンド(XBee データパケット)で <ratio>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。<ratio> パラメータを指定した場合にはリプライデータには <TDCPPrefix>と<status>だけが入ります。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,second_adjust,985” -> “$$$abc,1”
“$$$12345,second_adjust” -> “$$$12345,1,1000”

7.16 pullup,dio_pullup

- **機能概要**

  入力ポートのプルアップ設定または取得

  “config_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**

Console	pullup[,<16BitHex|8BitHex|4BitHex>]
XBee	<TDCPPrefix>,pullup[,<16BitHex|8BitHex|4BitHex>]

- **リプライデータフォーマット**

Console	pullup=<16BitHex|8BitHex|4BitHex>
XBee	<TDCPPrefix>,<status>[,<16BitHex|8BitHex|4BitHex>]

- **パラメータとリターン値**

  <TDCPPrefix> “$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

  <16BitHex|8BitHex|4BitHex> プルアップを行うビットを “1” に、プルアップ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:0000)

  <status> “1” でコマンド実行成功、”0” で失敗を表す

- **備考**

  プルアップは、マイクロプロセッサ内部のプルアップ機能を利用しています。

  プルアップ対象となるポートは、動作中の TDCP の “app_mode” によって変わります。プルアップ設定を変更した場合には、リセット後に有効になります。そのため、“pullup” コマンドに続けて、”config_save” コマンドを実行して、EEPROM に設定を保存した後 “reset” コマンドを実行してください。

  “pullup”,”app_mode” 設定コマンド実行例 (app_mode 8,入力ポートの全プルアップ)

app_mode,8
pullup,FF
config_save
reset

“pullup” コマンド設定値は常に 16bit幅で保存されています。TDCP 起動時に設定された、”app_mode” によって LSB 側のビット位置の ”pullup” 設定値から使用され、入力ポートにアサインされていないビット位置の “pullup” 設定値は無視されます。

リクエストコマンド(XBee データパケット)で <16BitHex|8BitHex|4BitHex>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。<16BitHex|8BitHex|4BitHex> パラメータを指定した場合にはリプライデータには <TDCPPrefix>と<status> だけが入ります。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,pullup,FF” -> “$$$abc,1”
“$$$12345,pullup” -> “$$$12345,1,0000”

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

“config_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**

Console	change_detect[,<16BitHex|8BitHex|4BitHex>]
XBee	<TDCPPrefix>,change_detect[,<16BitHex|8BitHex|4BitHex>]

- **リプライデータフォーマット**

Console	change_detect=<16BitHex|8BitHex|4BitHex>
XBee	<TDCPPrefix>,<status>[,<16BitHex|8BitHex|4BitHex>]

- **パラメータとリターン値**

<TDCPPrefix> “$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<16BitHex|8BitHex|4BitHex> ポート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:0000)

<status> “1” でコマンド実行成功、”0” で失敗を表す

- **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、”server_addr” コマンドで設定したデバイスに対してイベントデータを送信します。イベントデータの内容は、TDCP の現在の “app_mode” によって変わります。”CHANGE_DETECT” イベントの詳しい説明は、”TDCP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 16bit幅で保存されています。TDCP 起動時に設定された、”app_mode” によって LSB 側のビット位置の ”pullup” 設定値から使用され、入出力ポートにアサインされていないビット位置の “change_detect” 設定値は無視されます。

“change_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(XBee データパケット)で <16BitHex|8BitHex|4BitHex>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。<16BitHex|8BitHex|4BitHex> パラメータを指定した場合にはリプライデータには <TDCPPrefix>と<status> だけが入ります。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,change_detect,FF” -> “$$$abc,1”
“$$$12345,change_detect” -> “$$$12345,1,FF”

7.18 app_mode

- **機能概要**

TDCP動作モードの設定または取得

“config_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**

Console	app_mode[,<app_mode>]
XBee	<TDCPPrefix>,app_mode[,<app_mode>]

- **リプライデータフォーマット**

Console	app_mode=<app_mode>
XBee	<TDCPPrefix>,<status>[,<app_mode>]

- **パラメータとリターン値**

<TDCPPrefix>　　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<app_mode>　　TDCP 動作モードを指定する。
指定可能なモード番号と機能詳細は、”TDCP 動作モード” の章を参照して下さい。
（デフォルト設定値:0）

<status>　　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

“app_mode” コマンドで TDCP 動作モードを変更した場合には、リセット後に有効になります。そのため、“app_mode” コマンドに続けて、”config_save” コマンドを実行して、EEPROM に設定を保存した後 “reset” コマンドを実行してください。

”app_mode” 設定コマンド実行例 (app_mode 8,入力ポートの全プルアップ)

```
app_mode,8
config_save
reset
```

リクエストコマンド(XBee データパケット)で <app_mode>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。<app_mode> パラメータを指定した場合にはリプライデータには <TDCPPrefix>と <status> だけが入ります。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

```
“$$$abc,app_mode,8” -> “$$$abc,1”
“$$$12345,app_mode” -> “$$$12345,1,0”
```

7.19　dio_read,port_read

- **機能概要**

入出力ポート値の取得

- **リクエストフォーマット**

Console	port_read
XBee	<TDCPPrefix>,port_read

- **リプライデータフォーマット**

Console	app_mode=<port_value>
XBee	<TDCPPrefix>,<status>,<port_value>

- **パラメータとリターン値**

<TDCPPrefix>　　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<port_value>　　現在の入出力ポート値を16進数表記で表します。

<status>　　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

“port_read” では入力ポートに加えて、出力ポートに対しても読み込み対象となります。

現在の “app_mode” によって、対象となる I/O ポートが変化します。詳しいポートの設定については、”TDCP 動作モード” の章を参照して下さい。

“i2c_use” コマンドでフラグ値が 0 以外の値に設定されて、TDCP の I2C 機能が有効になっている場合には、PORTC のビット0とビット1 は “SCL” と “SDA” 機能にアサインされています。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,port_read” -> “$$$abc,1,FFFF”

7.20　dio_bit,port_bit

- **機能概要**

出力ポートの指定ビット位置の値を設定

- **リクエストフォーマット**

Console	port_bit,<bit_number>,<1|0>
XBee	<TDCPPrefix>,port_bit,<bit_number>,<1|0>

- **リプライデータフォーマット**

Console	** No reply data **
XBee	<TDCPPrefix>,<status>

- **パラメータとリターン値**

<TDCPPrefix>　　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<bit_number>　　ポートの値を設定する対象のビット位置を指定する。

<1|0>　　“1” で指定したポートのビット位置に High, “0” で Low を出力します。

<status>　　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

  現在の “app_mode” によって、対象となるポートとそのビット位置が変化します。詳しいポートの設定については、”TDCP 動作モード” の章を参照して下さい。

  “i2c_use” コマンドでフラグ値が 0 以外の値に設定されて、TDCP の I2C 機能が有効になっている場合には、PORTC のビット0とビット1 は “SCL” と “SDA” 機能にアサインされています。これらのビットに対して “port_bit” コマンドで値を設定すると、内部プルアップの有効(1 を設定した場合)と無効(0 を設定した場合)を設定することになります。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,port_bit,0,1” -> “$$$abc,1”

7.21 dio_write,port_write

- **機能概要**

  出力ポートの値を設定

- **リクエストフォーマット**

Console	port_write,<16BitHex|8BitHex|4BitHex>
XBee	<TDCPPrefix>,port_write[,<16BitHex|8BitHex|4BitHex>]

- **リプライデータフォーマット**

Console	** No reply data **
XBee	<TDCPPrefix>,<status>

- **パラメータとリターン値**

  <TDCPPrefix>　　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

  <16BitHex|8BitHex|4BitHex>　　出力ポートに設定する値を16進数表記で指定する。

  <status>　　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

  現在の “app_mode” によって、対象となるポートと出力ビット幅が変化します。詳しいポートの設定については、”TDCP 動作モード” の章を参照して下さい。

  “i2c_use” コマンドでフラグ値が 0 以外の値に設定されて、TDCP の I2C 機能が有効になっている場合には、PORTC のビット0とビット1 は “SCL” と “SDA” 機能にアサインされています。これらのビットに対して

“port_write” コマンドで値を設定すると、内部プルアップの有効(1 を設定した場合)と無効(0 を設定した場合)を設定することになります。

I2C 機能で使用している PORTC ビット0 とビット1 のプルアップに影響を与えないで他のビットのポート値を変更するためには、”port_write” コマンドを使用しないで ”port_bit” コマンドを使用して下さい。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,port_write,FF” -> “$$$abc,1”
“$$$,port_write,00” -> ** No reply data **

7.22 pulse

- **機能概要**

出力ポートの指定ビットにパルス信号を出力する。パルス出力を連続して出力させることもできる。

- **リクエストフォーマット**

Console	pulse,<bit_number>,<wait>,<width>[,<continuous>]
XBee	<TDCPPrefix>,pulse,<bit_number>,<wait>,<width>[,<continuous>]

- **リプライデータフォーマット**

Console	** No reply data **
XBee	<TDCPPrefix>,<status>

- **パラメータとリターン値**

<TDCPPrefix>	“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列
<bit_number>	ポートの値を設定する対象のビット位置を指定する。 app_mode で出力モードに設定されたポートの中で下４ビット(LSBから４ビット分)だけが指定できます。(0 から 3 までの整数を指定)
<wait>	パルス信号を出力するまでのウェイト時間を指定します。 単位はミリセカンド(ms) で 0 から 65535 までの整数を指定できます。 0 を指定した場合にはパルス信号がすぐに出力されます。
<width>	パルス信号幅を指定します。 単位はミリセカンド(ms) で 1 から 65535 までの整数を指定できます。
<continuous>	0 を指定した場合にはパルス信号は１回だけ出力されます。 1 を指定した場合には連続して繰り返しパルス信号が出力されます。 デフォルト値: 0
<status>	“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

  指定した出力ポートのビットにパルス信号を出力します。

  パルス信号は、現在のポートの値を反転させた値で出力されます。パルス信号出力後には、ビット値は元の値に戻ります。

  パルス信号出力中（waitで指定した間を含む）に該当ポートのビットを “port_write”,”port_bit” コマンドで操作した場合には、パルス信号の出力はその時点で中止されます。

  <continuous> に 1 を指定すると連続してパルス信号が出力されます。
  パルス信号を停止させるには、該当ポートのビットに対して再び “pulse” コマンドで <continuous> に 0 を指定するかまたは、”port_write”,”port_bit” コマンドを使用してポートを High または Low に設定してください。連続したパルス信号を出力する場合には、<wait> と <width>パラメータに 1 以上の値を指定する必要があります。

  指定したウェイト時間とパルス幅の時間精度は マイクロコントローラに接続されたクリスタルの精度とその時の TDCP のタスク処理内容によって、数パーセント程度の誤差があります。

  現在の “app_mode” によって、対象となるポートとそのビット位置が変化します。詳しいポートの設定については、”TDCP 動作モード” の章を参照して下さい。

  “i2c_use” コマンドでフラグ値が 0 以外の値に設定されて、TDCP の I2C 機能が有効になっている場合には、PORTC のビット0とビット1 は “SCL” と “SDA” 機能にアサインされています。これらのビットに対して “pulse” コマンドを実行しないでください。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,pulse,0,10,10” -> “$$$abc,1”

7.23 ad_read,adc_read

- **機能概要**

  A/D 変換入力値の取得

- **リクエストフォーマット**

Console	adc_read
XBee	<TDCPPrefix>,adc_read

- **リプライデータフォーマット**

Console	adc_read=<adc_values>
XBee	<TDCPPrefix>,<status>,<adc_values>

- **パラメータとリターン値**

<TDCPPrefix>　　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<adc_values>　　現在の A/D 変換入力値が10進数表記で表される。複数の A/D 変換値はカンマ文字で区切られる。

<status>　　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

現在の “app_mode” によって、対象となる A/D 変換ポートとデータ項目数が変化します。詳しいポートの設定については、”TDCP 動作モード” の章を参照して下さい。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,adc_read” -> “$$$abc,1,149,117,85,53”

7.24　ad_vref,adc_vref

- **機能概要**

A/D 変換リファレンス電圧設定レジスタ中のビット値(REFS1:REFS0)の設定または取得

“config_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**

Console	adc_vref[,<value>]
XBee	<TDCPPrefix>,adc_vref[,<value>]

- **リプライデータフォーマット**

Console	adc_vref=<value>
XBee	<TDCPPrefix>,<status>[,<value>]

- **パラメータとリターン値**

<TDCPPrefix>　　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<value>　　A/D 変換リファレンス電圧設定レジスタ値。0 から3 までの値を指定する。

0: REFS1:REFS0 = 0:0 (AREF, Internal Vref turned off)

1: REFS1:REFS0 = 0:1 (AVCC with external capacitor at AREF pin)

2: REFS1:REFS0 = 1:0 (Internal 1.1V Voltage Reference with external capacitor at AREF pin)

3: REFS1:REFS0 = 1:1 (Internal 2.56V Voltage Reference with external capacitor at AREF pin)

（デフォルト設定値:1）

<status>　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

adc_verf 設定を変更する場合は、回路が適切に接続されていることを確認して下さい。コマンド実行直後から有効になります。

リクエストコマンド(XBee データパケット)で <value>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。<value> パラメータを指定した場合にはリプライデータには <TDCPPrefix>と<status>だけが入ります。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,adc_vref,3” -> “$$$abc,1”

7.25　range_check_high

- **機能概要**

A/D 変換入力の制限値(上限値)オーバーを検出するビット位置の、設定または取得

“config_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**

Console	range_check_high[,<8BitHex>]
XBee	<TDCPPrefix>,range_check_high[,<8BitHex>]

- **リプライデータフォーマット**

Console	range_check_high=<8BitHex>
XBee	<TDCPPrefix>,<status>[,<8BitHex>]

- **パラメータとリターン値**

<TDCPPrefix>　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<8BitHex>	A/D 変換入力の制限値(上限値)オーバーを検出するビットを “1” に、 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:00)
<status>	“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

“range_check_high” で指定した A/D 変換ポートの値が上限値を越えていいた場合には、”RANGE_EXCEED” イベントが発生して、”server_addr” コマンドで設定したデバイスに対してイベントデータを送信します。A/D 変換ポートの上限値を設定するためには、”range_high” コマンドを使用します。

RANGE_EXCEED イベントが発生すると、制限値オーバーを検出したビットは監視対象から外されます。これは “range_check_high” コマンドで対象ビットを “0” にセットすることに相当します。このため、もう一度同じ A/D 入力ポートのビットで、同じ制限値をチェックしてイベントを発生させたい場合は、”range_check_high” コマンドで対象ビットを “1” にセットして下さい。”RANGE_EXCEED” イベントの詳しい説明は、”TDCPイベントリファレンス”の章を参照して下さい。

“range_check_high” コマンド設定値は常に 8bit幅で保存されています。TDCP 起動時に設定された、”app_mode” によって LSB 側のビット位置の ”range_check_high” 設定値から使用され、A/D 変換ポートにアサインされていないビット位置の “range_check_high” 設定値は無視されます。

リクエストコマンド(XBee データパケット)で <8BitHex>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。<8BitHex> パラメータを指定した場合にはリプライデータには <TDCPPrefix>と <status> だけが入ります。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,range_check_high,FF” -> “$$$abc,1”

7.26 range_check_low

- **機能概要**

A/D 変換入力の制限値(下限値)オーバーを検出するビット位置の、設定または取得
“config_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**

Console	range_check_low[,<8BitHex>]
XBee	<TDCPPrefix>,range_check_low[,<8BitHex>]

- **リプライデータフォーマット**

Console	range_check_low=<8BitHex>
XBee	<TDCPPrefix>,<status>[,<8BitHex>]

- **パラメータとリターン値**

<TDCPPrefix>　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<8BitHex>　A/D 変換入力の制限値(下限値)オーバーを検出するビットを “1” に、
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:00)

<status>　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

“range_check_low” で指定した A/D 変換ポートの値が下限値を越えていいた場合には、”RANGE_EXCEED” イベントが発生して、”server_addr” コマンドで設定したデバイスに対してイベントデータを送信します。A/D 変換ポートの下限値を設定するためには、”range_low” コマンドを使用します。

RANGE_EXCEED イベントが発生すると、制限値オーバーを検出したビットは監視対象から外されます。これは “range_check_low” コマンドで対象ビットを “0” にセットすることに相当します。このため、もう一度同じ A/D 入力ポートのビットで、同じ制限値をチェックしてイベントを発生させたい場合は、”range_check_low” コマンドで対象ビットを “1” にセットして下さい。”RANGE_EXCEED” イベントの詳しい説明は、”TDCPイベントリファレンス”の章を参照して下さい。

“range_check_low” コマンド設定値は常に 8bit幅で保存されています。TDCP 起動時に設定された、”app_mode” によって LSB 側のビット位置の ”range_check_low” 設定値から使用され、A/D 変換ポートにアサインされていないビット位置の “range_check_low” 設定値は無視されます。

リクエストコマンド(XBee データパケット)で <8BitHex>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。<8BitHex> パラメータを指定した場合にはリプライデータには <TDCPPrefix>と <status> だけが入ります。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,range_check_low,FF” -> “$$$abc,1”

7.27　range_high

- **機能概要**

A/D 変換入力の制限値(上限値)の、設定または取得

“config_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**

Console	range_high,<bit_number>[,<limit_value>]
XBee	<TDCPPrefix>,range_high,<bit_number>[,<limit_value>]

- **リプライデータフォーマット**

Console	range_high_<bit_number>=<limit_value>
XBee	<TDCPPrefix>,<status>[,<limit_value>]

- **パラメータとリターン値**

<TDCPPrefix>	“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列
<bit_number>	ポートの値を設定する対象のビット位置を指定する。
<limit_value>	A/D 変換値の上限値を10進数表記で指定する。(0 から1023 の値を指定する) (デフォルト設定値:700)
<status>	“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

“range_high” コマンドで、A/D 変換ポートの上限値を設定します。上限値の検出を行うためには、”range_high” コマンドに加えて、“range_check_high” コマンドで該当ビットの検出を有効にして下さい。

A/D 変換ポートの値が上限値を越えていいた場合には、”RANGE_EXCEED” イベントが発生して、”server_addr” コマンドで設定したデバイスに対してイベントデータを送信します。

リクエストコマンド(XBee データパケット)で <limit_value>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。<limit_value> パラメータを指定した場合にはリプライデータには <TDCPPrefix>と<status> だけが入ります。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,range_high,0,450” -> “$$$abc,1”
“$$$12345,range_high,0” -> “$$$12345,1,700”

7.28 range_low

- **機能概要**

A/D 変換入力の制限値(下限値)の、設定または取得

“config_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**

Console	range_low,<bit_number>[,<limit_value>]
XBee	<TDCPPrefix>,range_low,<bit_number>[,<limit_value>]

- **リプライデータフォーマット**

Console	range_low_<bit_number>=<limit_value>
XBee	<TDCPPrefix>,<status>[,<limit_value>]

- **パラメータとリターン値**

<TDCPPrefix>　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<bit_number>　ポートの値を設定する対象のビット位置を指定する。

<limit_value>　A/D 変換値の下限値を10進数表記で指定する。(0 から1023 の値を指定する)
(デフォルト設定値:300)

<status>　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

“range_low” コマンドで、A/D 変換ポートの下限値を設定します。下限値の検出を行うためには、”range_low” コマンドに加えて、“range_check_low” コマンドで該当ビットの検出を有効にして下さい。

A/D 変換ポートの値が下限値を越えていいた場合には、”RANGE_EXCEED” イベントが発生して、”server_addr” コマンドで設定したデバイスに対してイベントデータを送信します。

リクエストコマンド(XBee データパケット)で <limit_value>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。<limit_value> パラメータを指定した場合にはリプライデータには <TDCPPrefix>と<status> だけが入ります。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,range_low,0,450” -> “$$$abc,1”
“$$$12345,range_low,0” -> “$$$12345,1,300”

7.29　change_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

“config_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**

Console	change_count_check[,<flag>]
XBee	<TDCPPrefix>,change_count_check[,<flag>]

- **リプライデータフォーマット**

Console	change_count_check=<flag>
XBee	<TDCPPrefix>,<status>[,<flag>,<limit_value>]

- **パラメータとリターン値**

<TDCPPrefix> “$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<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

（デフォルト設定値:”0”）

<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)

change_count_high コマンドで設定した値が表示されます

<status> “1” でコマンド実行成功、”0” で失敗を表す

- **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、”TDCP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(XBee データパケット)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。この時ポート入力カウンタの制限値(上限値)の設定値も同時にリプライデータに含まれます。<flag> パラメータを指定した場合にはリプライデータには <TDCPPrefix>と<status> だけが入ります。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,change_count_check,1” -> “$$$abc,1”
“$$$12345,change_count_check” -> “$$$12345,1,0,10”

7.30 change_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

  “config_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**

Console	change_count_high[,<limit_value>]
XBee	<TDCPPrefix>,change_count_high[,<limit_value>]

- **リプライデータフォーマット**

Console	change_count_high=<limit_value>
XBee	<TDCPPrefix>,<status>[,<limit_value>]

- **パラメータとリターン値**

<TDCPPrefix>	“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列
<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)
<status>	“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

  入力ポートカウンタ値のチェックについては、”TDCP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(XBee データパケット)で <limit_value>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。<limit_value> パラメータを指定した場合にはリプライデータには <TDCPPrefix>と<status> だけが入ります。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,change_count_high,200” -> “$$$abc,1”
“$$$12345,change_count_high” -> “$$$12345,1,10”

7.31 change_count_reset

- **機能概要**

  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**

Console	change_count_reset
XBee	<TDCPPrefix>,change_count_reset

- **リプライデータフォーマット**

Console	** No reply data **
XBee	<TDCPPrefix>,<status>

- **パラメータとリターン値**

  <TDCPPrefix>　　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

  <status>　　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

  入力ポートカウンタ値のチェックについては、”TDCP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。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc, change_count_reset” -> “$$$abc,1”

7.32 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**

Console	signal_read
XBee	<TDCPPrefix>,signal_read

- **リプライデータフォーマット**

Console	signal_read =<signal_state>
XBee	<TDCPPrefix>,<status>[,<signal_state>]

- **パラメータとリターン値**

<TDCPPrefix>　　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<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。

<status>　　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

シグナル出力機能は、TDCP の “app_mode” を設定することで有効になります。詳しくは ”TDCP動作モード”の章を参照して下さい。

現在のシグナルステータスの値を取得する。

シグナルステータスとTDCP のポート出力値の説明は、”シグナル出力機能” の章を参照して下さい。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,signal_read” -> “$$$abc,1,58”

7.33　signal_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**

Console	signal_bit,<bit_number>,<flag>
XBee	<TDCPPrefix>,signal_bit,<bit_number>,<flag>

- **リプライデータフォーマット**

Console	** No reply data **
XBee	<TDCPPrefix>,<status>

- **パラメータとリターン値**

<TDCPPrefix>　　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<bit_number>　　シグナルステータスを設定する対象のビット位置を指定する。

<flag>　　“1” を設定すると該当シグナルが “ON” 、“0” を設定すると “OFF” になる。

<status>　　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

シグナル出力機能は、TDCP の “app_mode” を設定することで有効になります。詳しくは ”TDCP動作モード”の章を参照して下さい。

シグナルステータス中の指定したビットを 1 または 0 に設定する。

シグナルステータスとTDCP のポート出力値の説明は、”シグナル出力機能” の章を参照して下さい。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,signal_bit,7,1” -> “$$$abc,1”

7.34　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**

Console	signal_write,<signal_state>
XBee	<TDCPPrefix>, signal_write,<signal_state>

- **リプライデータフォーマット**

Console	** No reply data **
XBee	<TDCPPrefix>,<status>

- **パラメータとリターン値**

<TDCPPrefix>　　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<signal_state>　　シグナルステータス(8bit)を16進数表記で設定する。

<status>　　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

シグナル出力機能は、TDCP の “app_mode” を設定することで有効になります。詳しくは ”TDCP動作モード”の章を参照して下さい。

シグナルステータスを指定した値に設定する。

シグナルステータスとTDCP のポート出力値の説明は、”シグナル出力機能” の章を参照して下さい。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,signal_write,58” -> “$$$abc,1”

7.35 gps_gga

- **機能概要**

GPS レシーバから受信した NMEA-0183 センテンス($GPGGA) の取得

- **リクエストフォーマット**

Console	** コンソールからは使用できません **
XBee	<TDCPPrefix>,gps_gga

- **リプライデータフォーマット**

Console	** コンソールからは使用できません **
XBee	<TDCPPrefix>,<status>[,<NMEA-0183_GGA_sentence>]

- **パラメータとリターン値**

<TDCPPrefix> “$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<NMEA-0183_GGA_sentence> GPSレシーバから受信した最後のNMEAセンテンス($GPGGA) の内容が入る。一度もセンテンスを受信していな場合には空文字列になる。

<status> “1” でコマンド実行成功、”0” で失敗を表す

- **備考**

このコマンドは XBee データパケット経由からのコマンドのみ利用可能です。(コンソールポートは GPS レシーバに接続されている為)

コマンド実行結果は、常にGPS レシーバから受信した最後のセンテンスを返します。GPS レシーバから最新のデータを取得したい場合には、”position_report”,”position_once” コマンドを使用して、”$GPRMC” イベントハンドラまたは “GPS” イベントハンドラ中からこのコマンドを使用する事を検討してください。

GPS レシーバの詳しい接続方法とデータ取得方法については、”GPSレシーバ接続”の章を参照してください。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,gps_gga” ->“$$$abc,1,$GPGGA,090058,4253.1660,N,14132.6111,E,1,04,2.9,15.0,M,30.7,M,,*7A”

7.36 gps_rmc

- **機能概要**

  GPS レシーバから受信した NMEA-0183 センテンス($GPRMC) の取得

- **リクエストフォーマット**

Console	** コンソールからは使用できません **
XBee	<TDCPPrefix>,gps_rmc

- **リプライデータフォーマット**

Console	** コンソールからは使用できません **
XBee	<TDCPPrefix>,<status>[,<NMEA-0183_RMC_sentence>]

- **パラメータとリターン値**

  <TDCPPrefix>　　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

  <NMEA-0183_RMC_sentence>　　GPSレシーバから受信した最後のNMEAセンテンス($GPRMC) の内容が入る。一度もセンテンスを受信していな場合には空文字列になる。

  <status>　　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

  このコマンドは XBee データパケット経由からのコマンドのみ利用可能です。（コンソールポートは GPS レシーバに接続されている為）

  コマンド実行結果は、常にGPS レシーバから受信した最後のセンテンスを返します。GPS レシーバから最新のデータを取得したい場合には、”position_report”,”position_once” コマンドを使用して、”$GPRMC” イベントハンドラまたは “GPS” イベントハンドラ中からこのコマンドを使用する事を検討してください。

  GPS レシーバの詳しい接続方法とデータ取得方法については、”GPSレシーバ接続”の章を参照してください。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,gps_rmc” ->“$$$abc,1,$GPRMC,090100,A,4253.1660,N,14133.6111,E,0.0,225.8,071109,9.3,W,A*0B”

7.37 gps_report,position_report

- **機能概要**

GPSレシーバからNMEA-0183センテンス受信毎に、繰り返し"$GPRMC"イベントデータまたは "GPS" イベントデータの送信を行うフラグの設定または取得。

"config_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**

Console	position_report[,<flag>]
XBee	<TDCPPrefix>, position_report[,<flag>]

- **リプライデータフォーマット**

Console	position_report=<flag>
XBee	<TDCPPrefix>,<status>[,<flag>]

- **パラメータとリターン値**

<TDCPPrefix> "$$$" または $$$*****、"*****" は省略可能で、指定時は5文字以内の英数字文字列

<flag> 0: イベント送信を行わない。

1: "$GPRMC" イベント送信を行う

2: "GPS" イベント送信を行う

(デフォルト設定値:0)

<status> "1" でコマンド実行成功、"0" で失敗を表す

- **備考**

"position_report" の設定を有効にすると、GPS レシーバから NMEA-0183 センテンスの "$GPRMC", "$GPGGA" の両方を受信したときに、"$GPRMC" イベントまたは "GPS" イベントが発生して、"server_addr" コマンドで設定したデバイスに対してイベントデータを送信します。"$GPRMC" イベントと"GPS" イベントの詳しい説明は、"TDCPイベントリファレンス"の章を参照して下さい。GPS レシーバの詳しい接続方法とデータ取得方法については、"GPSレシーバ接続"の章を参照してください。

リクエストコマンド(XBee データパケット)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。<flag> パラメータを指定した場合にはリプライデータには <TDCPPrefix>と<status> だけが入ります。

- **XBee データパケットの使用例("リクエストコマンドデータ" -> "リプライデータ")**

"$$$abc,position_report,1" -> "$$$abc,1"

"$$$12345,position_report" -> "$$$12345,1,1"

7.38 gps_once,position_once

- **機能概要**

GPSレシーバからNMEA-0183センテンス受信時に、一回だけ ”$GPRMC”イベントデータまたは”GPS” イベントデータの送信を行うフラグの設定または取得。

- **リクエストフォーマット**

Console	position_once[,<flag>]
XBee	<TDCPPrefix>, position_once[,<flag>]

- **リプライデータフォーマット**

Console	position_once=<flag>
XBee	<TDCPPrefix>,<status>[,<flag>]

- **パラメータとリターン値**

<TDCPPrefix> “$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<flag> 0: イベント送信を行わない。

1: “$GPRMC” イベント送信を行う

2: “GPS” イベント送信を行う

<status> “1” でコマンド実行成功、”0” で失敗を表す

- **備考**

“position_once” の設定を ”1” にすると、GPS レシーバから NMEA-0183 センテンスの “$GPRMC”, “$GPGGA” の両方を受信したときに、”$GPRMC” イベントが１回だけ発生して、”server_addr” コマンドで設定したデバイスに対してイベントデータを送信します。

”$GPRMC” イベントまたは “GPS” イベント発生と同時に、”position_once” は “0” にセットされます。”$GPRMC” イベントと “GPS”イベントの詳しい説明は、”TDCPイベントリファレンス”の章を参照して下さい。GPS レシーバの詳しい接続方法とデータ取得方法については、”GPSレシーバ接続”の章を参照してください。

リクエストコマンド(XBee データパケット)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。<flag> パラメータを指定した場合にはリプライデータには <TDCPPrefix>と<status> だけが入ります。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,position_once,1” -> “$$$abc,1”
“$$$12345,position_once” -> “$$$12345,1,1”

7.39 gps_rate,position_report_rate

- **機能概要**

  “position_report”コマンドで送信される “$GPRMC”,”GPS” イベントの送信間隔の設定または取得

  “config_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**

Console	position_report_rate[,<rate>]
XBee	<TDCPPrefix>,position_report_rate[,<rate>]

- **リプライデータフォーマット**

Console	position_report_rate=<rate>
XBee	<TDCPPrefix>,<status>[,<rate>]

- **パラメータとリターン値**

<TDCPPrefix>	“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列
<rate>	“$GPRMC”,”GPS” イベントの送信間隔を指定する (1 から 2^31 までの整数) イベント発生時に、実際にサーバーに送信する動作をこのコマンドで設定した回数分の１にまで間引く。1 を設定した場合には間引かずに送信される。 (デフォルト設定値:1)
<status>	“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

  “position_report” コマンドによって設定された GPS イベントデータ送信時に、GPS レシーバーから送信されるセンテンスデータの頻度よりも、送信するイベントデータ数を少なくしたい場合に使用します。

  例えば、”position_report_rate” に 3 を設定した場合には、サーバーに”GPS” イベント送信を送信する頻度は、GPS レシーバーから送信される GPSセンテンス送信頻度の 1/3 になります。

  リクエストコマンド(XBee データパケット)で <rate>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。<rate> パラメータを指定した場合にはリプライデータには <TDCPPrefix>と<status> だけが入ります。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,position_report_rate,10” -> “$$$abc,1”
“$$$12345,position_report_rate” -> “$$$12345,1,10”

7.40 serial_number

- **機能概要**

  TDCP に接続されている XBee デバイスのシリアル番号(64bitアドレス)と16bitアドレスの取得

- **リクエストフォーマット**

Console	serial_number
XBee	<TDCPPrefix>,serial_number

- **リプライデータフォーマット**

  TDCP

Console	serial_number=<xbee_serial>,<my_addr16>
XBee	<TDCPPrefix>,<status>,<xbee_serial>,<my_addr16>

  TDCPZB

Console	serial_number=<xbee_serial>
XBee	<TDCPPrefix>,<status>,<xbee_serial>

- **パラメータとリターン値**

<TDCPPrefix>	“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列
<xbee_serial>	XBee デバイスのシリアル番号（64bitアドレス）を16進数表記で表示する。
<my_addr16>	XBee デバイスの 16bitアドレスを16進数表記で表示する。
<status>	“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

  TDCP プログラム起動時に、XBee デバイスから読み込んだシリアル番号と16ビットアドレスを取得する。

  TDCPZB の場合にはシリアル番号のみで 16ビットアドレスは取得しません。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,serial_number” -> “$$$abc,1,0013A200404AC39C,0A01”

7.41 echo

- **機能概要**

  リクエストコマンドで指定した文字列をリプライデータにエコーバックする

- **リクエストフォーマット**

Console	echo,<string>
XBee	<TDCPPrefix>,echo,<string>

- **リプライデータフォーマット**

Console	echo=<string>
XBee	<TDCPPrefix>,<status>,<string>

- **パラメータとリターン値**

<TDCPPrefix>　　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<string>　　エコーバックしたい任意の文字列を指定する。

<status>　　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,echo,Hello_World!!” -> “$$$abc,1,Hello World !!”
“$$$12345,echo,123,456,ABC” -> “$$$12345,1,123,456,ABC”

7.42 tx_data

- **機能概要**

  TDCP に接続したリモートの XBee デバイスから、別の XBee デバイスにバイナリデータを送信

- **リクエストフォーマット**

TDCP

Console	tx_data[,<16BitAddrHexStr|64BitAddrHexStr>],<RFDataHexStr>
XBee	<TDCPPrefix>,tx_data[,<16BitAddrHexStr|64BitAddrHexStr>],<RFDataHexStr>

TDCPZB

Console	tx_data[,<64BitAddrHexStr>],<RFDataHexStr>

XBee	<TDCPPrefix>,tx_data[,<64BitAddrHexStr>],<RFDataHexStr>

- **リプライデータフォーマット**

Console	** No reply data **
XBee	<TDCPPrefix>,<status>

- **パラメータとリターン値**

<TDCPPrefix>　　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<16BitAddrHexStr|64BitAddrHexStr>

送信対象の XBee デバイスアドレス。16bitまたは、64bit アドレスを16進数表記で指定する。このパラメータを省略した場合には “server_addr” コマンドで設定されたアドレスが送信対象になる。TDCPZB の場合には64bitアドレスのみが指定できます。

<RFDataHexStr>　　送信するバイナリデータを16進数表記で指定する。

<status>　　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

リモートのXBee デバイスから送信する時にエラーフレームのチェックは行いません。送信先の XBee デバイスに対してのリプライデータ（もし送信先が TDCP デバイスであった場合）の処理は行いません。

指定した送信先アドレスにデータ送信を行った後に、リクエストコマンド元にリプライデータを続けて送信します。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,tx_data,0013A200404AC397,414243” -> “$$$abc,1”

7.43　tx_ascii,tx

- **機能概要**

TDCP に接続したリモートの XBee デバイスから、別の XBee デバイスにASCII文字列を送信

- **リクエストフォーマット**

TDCP

Console	tx_ascii[,<16BitAddrHexStr|64BitAddrHexStr>],<string>
XBee	<TDCPPrefix>,tx_ascii[,<16BitAddrHexStr|64BitAddrHexStr>],<string>

TDCPZB

Console	tx_ascii[,<64BitAddrHexStr>],<string>

XBee	<TDCPPrefix>,tx_ascii[,<64BitAddrHexStr>],<string>

- **リプライデータフォーマット**

Console	** No reply data **
XBee	<TDCPPrefix>,<status>

- **パラメータとリターン値**

<TDCPPrefix>　　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

“tx_ascii”　　コマンド文字列。省略形式として “tx” を使用することもできる

<16BitAddrHexStr|64BitAddrHexStr>

送信対象の XBee デバイスアドレス。16bitまたは、64bit アドレスを16進数表記で指定する。このパラメータを省略した場合には “server_addr” コマンドで設定されたアドレスが送信対象になる。TDCPZB の場合には64bitアドレスのみが指定できます。

<string>　　送信する任意の文字列を指定する。

<status>　　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

リモートのXBee デバイスから送信する時にエラーフレームのチェックは行いません。送信先の XBee デバイスに対してのリプライデータ（もし送信先が TDCP デバイスであった場合）の処理は行いません。

送信可能な任意の文字列 <string> 中に “,” コンマを含めることができます。コマンドパラメータの区切り文字もコンマのため、コマンド行全体でカンマ文字がリモートコマンドで 3 つ以上、コンソールコマンドで 2 つ以上の場合は、必ず送信対象の XBee デバイスアドレスパラメータ<16BitAddrHexStr|64BitAddrHexStr>を指定する必要があります。このときは server_addr コマンドで設定したデフォルトアドレスへ送信する場合でも 、<16BitAddrHexStr|64BitAddrHexStr> コマンドパラメータのアドレス指定が必要になります。

指定した送信先アドレスにデータ送信を行った後に、リクエストコマンド元にリプライデータを続けて送信します。

省略形式 “tx” は、送信データ内にtdcp コマンドを含める場合など、全体の送信データ文字列を短くしたい時に便利です。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,tx_ascii,0013A200404AC397,Hello World!!” -> “$$$abc,1” “$$$abc,tx,0013A200404AC39C,$$$,lcd_disp,0,*** ｴﾗｰﾊｯｾｲ ***” -> “$$$abc,1”

7.44 eeprom_read

- **機能概要**

  マイクロプロセッサ内部のEEPROM に書き込まれているデータを取得する。

- **リクエストフォーマット**

Console	eeprom_read,<RomAddrHexStr>[,<length>]
XBee	<TDCPPrefix>,eeprom_read,<RomAddrHexStr>[,<length>]

- **リプライデータフォーマット**

Console	** EEPROM メモリ内容がダンプされます **
XBee	<TDCPPrefix>,<status>,<DataHexStr>

- **パラメータとリターン値**

<TDCPPrefix>	“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列
<RomAddrHexStr>	EEPROM アドレスを16進数表記で指定する 例: FF,0x0F
<length>	読み込むバイト数を 10進数表記で指定する。このパラメータを省略した場合は、デフォルトで 32 バイト読み込まれる。 例: 10,32
<DataHexStr>	読み込んだ EEPROM データを16進数表記で表示される。
<status>	“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

  TDCP 内部の処理で、EEPROM メモリ操作中は全てのインターラプトが禁止されます。そのため、その間のシリアルコンソールポートや XBee デバイスからのシリアルデータを取りこぼす場合があります。

  I2C インターフェイスで接続した、外部の EEPROM デバイスを扱うときは I2C 機能のコマンドを使用してください。”i2c_use”,”i2c_read”,”i2c_write” コマンドを参照して下さい。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,eeprom_read,10,16” -> “$$$abc,1,010A00000000010013A200404AC398C0”

7.45 eeprom_write

- **機能概要**

  マイクロプロセッサ内部のEEPROM に、指定したバイナリデータを書き込む。

- **リクエストフォーマット**

Console	eeprom_write,<RomAddrHexStr>,<DataHexStr>
XBee	<TDCPPrefix>,eeprom_write,<RomAddrHexStr>,<DataHexStr>

- **リプライデータフォーマット**

Console	** No reply data **
XBee	<TDCPPrefix>,<status>

- **パラメータとリターン値**

<TDCPPrefix>　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<RomAddrHexStr>　EEPROM アドレスを16進数表記で指定する

例: FF,0x0F

<DataHexStr>　EEPROM に書き込むバイナリデータを16進数表記で指定する。

16進数表記を表す ”0x” 部分は必ず除いて、バイト単位(2文字)で指定する。

例: F1F2F3,0000,01020304,00

<status>　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

TDCP 内部の処理で、EEPROM メモリ操作中は全てのインターラプトが禁止されます。そのため、その間のシリアルコンソールポートや XBee デバイスからのシリアルデータを取りこぼす場合があります。

I2C インターフェイスで接続した、外部の EEPROM デバイスを扱うときは I2C 機能のコマンドを使用してください。”i2c_use”,”i2c_read”,”i2c_write” コマンドを参照して下さい。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,eeprom_write,200,01020304” -> “$$$abc,1”

7.46　lcd_clear

- **機能概要**

LCD 表示モジュールの表示内容消去(全てスペース文字が表示される)

- **リクエストフォーマット**

Console	lcd_clear

XBee	<TDCPPrefix>,lcd_clear

- **リプライデータフォーマット**

Console	** No reply data **
XBee	<TDCPPrefix>,<status>

- **パラメータとリターン値**

<TDCPPrefix>　　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<status>　　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

LCD 表示モジュールの表示をクリアします。

“lcd_disp” コマンドで設定した文字列バッファの内容も全てクリアします。

LCD 表示モジュール接続をサポートした app_mode でのみ動作します。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,lcd_clear” -> “$$$abc,1”

7.47　lcd_wrap

- **機能概要**

複数行表示可能な LCD 表示モジュールに、文字列バッファ[0] の内容を行末尾で折り返して複数行にまたがって表示するフラグの設定または取得

“config_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**

Console	lcd_wrap[,<flag>]
XBee	<TDCPPrefix>,lcd_wrap[,<flag>]

- **リプライデータフォーマット**

Console	lcd_wrap=<flag>
XBee	<TDCPPrefix>,<status>[,<flag>]

- **パラメータとリターン値**

<TDCPPrefix>	“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列
<flag>	LCD 表示モジュールの各行を折り返して文字列バッファ[0]の内容を表示する場合は “1” を設定する。 LCD 表示モジュール各行の表示内容を文字列バッファの各行ごとの文字列バッファの内容にする場合は “0”を設定する。 (デフォルト設定値:0)
<status>	“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

“1” に設定した場合で、文字列バッファ[0]の内容が LCD 表示モジュール全体の表示可能な文字数を超えていたときには最初の部分のみが表示されます。この場合にはスクロール表示されません。

“0” に設定した場合には、文字列バッファ[N] (N は 0 から始まる行番号) の内容が、LCD 表示モジュールの対応する行にそれぞれ表示されます。文字列バッファ[N]の内容が LCD 表示モジュール1 行分の表示可能な文字数を超えていたときには、文字列の最初の部分のみが始めに表示され、残りの部分はスクロール表示されます。

LCD表示用の文字列バッファの内容を設定する場合は、“lcd_disp” コマンドの項目を参照してください。

LCD 表示モジュール接続をサポートした app_mode でのみ動作します。

リクエストコマンド(XBee データパケット)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。<flag> パラメータを指定した場合にはリプライデータには <TDCPPrefix>と<status>だけが入ります。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,lcd_wrap,1” -> “$$$abc,1”

7.48 lcd_disp

- **機能概要**

LCD 表示モジュールに表示する、文字列バッファの内容を設定する

- **リクエストフォーマット**

Console	lcd_disp,<line_no>,<message>
XBee	<TDCPPrefix>,lcd_disp,<line_no>,<message>

- **リプライデータフォーマット**

Console	** No reply data **
XBee	<TDCPPrefix>,<status>

- **パラメータとリターン値**

<TDCPPrefix>　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<line_no>　LCD 表示モジュールの各行に対応する文字列バッファの行を 0 から始まる数で指定する。 0 以上で ”lcd_lines” で設定した行数よりも少ない整数を指定する

<message>　LCD 表示モジュールの各行に対応する文字列バッファの内容を指定する
各行のバッファに設定可能な文字列は最大 80 文字(ASCII) で、それ以上指定した場合は切り捨てられます。

<status>　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

文字列バッファ[N] (N は 0 から始まる行番号) に設定した内容が、LCD 表示モジュールの各行にそれぞれ表示されます。文字列バッファ[N]の内容が LCD 表示モジュール1 行分の表示可能な文字数を超えていたときには、文字列の最初の部分のみが始めに表示され、残りの部分はスクロール表示されます。

LCD 表示モジュールの各行を折り返して文字列バッファ[0]の内容のみを表示する場合には “lcd_wrap” コマンドを使用します。詳しくは “lcd_wrap” コマンドを参照してください。

<message> にスペース文字(0x20) を１つだけ指定することで、LCD表示モジュールの指定した行は空白になります。全行をクリアする場合には”lcd_clear” コマンドを使用するほうが便利です。

LCD 表示モジュール接続をサポートした app_mode でのみ動作します。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,lcd_wrap,1” -> “$$$abc,1”

“$$$abc,lcd_disp,0,long long long message !!” -> “$$$abc,1”

“$$$abc,lcd_wrap,0” -> “$$$abc,1”

“$$$abc,lcd_disp,0, ” -> “$$$abc,1”

“$$$abc,lcd_disp,1,long long long message !!” -> “$$$abc,1”

7.49　lcd_lines

- **機能概要**

LCD 表示モジュールの行数の設定または取得

“config_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**

Console	lcd_lines[,<lines>]
XBee	<TDCPPrefix>,lcd_lines[,<lines>]

- **リプライデータフォーマット**

Console	lcd_lines=<lines>
XBee	<TDCPPrefix>,<status>[,<lines>]

- **パラメータとリターン値**

<TDCPPrefix>　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<lines>　LCD 表示モジュールの行数を指定する。1,2,4 のいずれかを指定可能。
（デフォルト設定値:2)

<status>　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

<lines> 設定を変更した場合には、リセット後に有効になります。そのため、“lcd_lines” コマンドに続けて、”config_save” コマンドを実行して、EEPROM に設定を保存した後 “reset” コマンドを実行してください。

“lcd_lines”,”lcd_length”,”app_mode” 設定コマンド実行例

```
app_mode,9
lcd_lines,2
lcd_length,16
config_save
reset
```

リクエストコマンド(XBee データパケット)で <lines>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。<lines> パラメータを指定した場合にはリプライデータには <TDCPPrefix>と<status> だけが入ります。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

```
“$$$abc,lcd_lines,2” -> “$$$abc,1”
“$$$12345,lines” -> “$$$12345,1,2”
```

7.50 lcd_length

- **機能概要**

  LCD 表示モジュールの文字カラム数の設定または取得

  “config_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**

Console	lcd_length[,<columns>]
XBee	<TDCPPrefix>,lcd_length[,<columns>]

- **リプライデータフォーマット**

Console	lcd_length=<columns>
XBee	<TDCPPrefix>,<status>[,<columns>]

- **パラメータとリターン値**

<TDCPPrefix>	“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列
<columns>	LCD 表示モジュールの文字カラム数を指定する。1 から 20 までの整数を指定可能。（デフォルト設定値:16）
<status>	“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

  < columns> 設定を変更した場合には、リセット後に有効になります。そのため、“lcd_length” コマンドに続けて、”config_save” コマンドを実行して、EEPROM に設定を保存した後 “reset” コマンドを実行してください。

  “lcd_lines”,”lcd_length”,”app_mode” 設定コマンド実行例

```
app_mode,9
lcd_lines,2
lcd_length,16
config_save
reset
```

  リクエストコマンド(XBee データパケット)で <columns>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。<columns> パラメータを指定した場合にはリプライデータには <TDCPPrefix>と <status> だけが入ります。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,lcd_length,16” -> “$$$abc,1”
“$$$12345,length” -> “$$$12345,1,16”

7.51 router (TDCP専用)

- **機能概要**

  TDCP イベントデータとサンプリングデータ送信時に、途中に複数の TDCP デバイスを経由して送信するためのアドレス(ルータアドレス)の設定または取得

  “config_save” コマンドによる EEPROM への設定値保存対象

  このコマンドは TDCPZB では使用できません

- **リクエストフォーマット**

Console	router,<index>[,<16BitAddrHexStr|64BitAddrHexStr>]
XBee	<TDCPPrefix>,router,<index>[,<16BitAddrHexStr|64BitAddrHexStr>]

- **リプライデータフォーマット**

Console	router<index>=<16BitAddrHexStr|64BitAddrHexStr>
XBee	<TDCPPrefix>,<status>[,<16BitAddrHexStr|64BitAddrHexStr>]

- **パラメータとリターン値**

  <TDCPPrefix>　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

  <index>　ルータアドレスのインデックスで 0 または 1 を指定する。

  <16BitAddrHexStr|64BitAddrHexStr>

  TDCP イベントデータとサンプリングデータ送信時に経由するTDCP デバイスの XBeeアドレス(16bit または64bit) 16進数形式の文字列で記述する(例: “0A01”)

  (デフォルト設定値:””)

  <status>　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

  ルータアドレスを設定することで、TDCP イベントデータとサンプリングデータ送信時に server_addr コマンドで設定した XBee デバイスに直接送信しないで、ルータ経由でイベントデータパケットを送信することができます。ルータとして使用される TDCP デバイスでは特別な設定をする必要はありません。

  ルータアドレスは最大 2 個まで指定可能で <index> パラメータで指定します。 0 は “NEAR ROUTER”、1 は “FAR ROUTER” と呼び、自身のデバイスに近い側と遠い側のルータをそれぞれ意味しています。

  デフォルトでは <index> = 0 の “NEAR ROUTER” と <index> = 1 の“FAR ROUTER” 両方が未設定の状態になっています。この場合にはTDCP イベントデータとサンプリングデータは直接 server_addr コマンドで設定した

XBee デバイスに送信されます。
（自身のデバイス）=>（server_addr で指定されたXBee デバイス）

<index> = 0 の “NEAR ROUTER” のみが設定されていた場合には、設定されたルータ(TDCP デバイス）経由でイベントデータが送信されます。
（自身のデバイス）=>（NEAR ROUTER）=>（server_addr で指定されたXBee デバイス）

<index> = 1 の “FAR ROUTER” のみが設定されていた場合には、設定されたルータ(TDCP デバイス）経由でイベントデータが送信されます。
（自身のデバイス）=>（FAR ROUTER）=>（server_addr で指定されたXBee デバイス）

<index> = 0 の “NEAR ROUTER” と <index> = 1 の “FAR ROUTER” の両方が設定されていた場合には、設定された複数のルータ(TDCP デバイス）経由でイベントデータが送信されます。
（自身のデバイス）=>（NEAR ROUTER）=>（FAR ROUTER）=>（server_addr で指定されたXBee デバイス）

リクエストコマンド(XBee データパケット)で <16BitAddrHexStr|64BitAddrHexStr>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。この時、アドレス値が未設定の場合にはリプライデータの<16BitAddrHexStr|64BitAddrHexStr >に “0” <status> に “1” が返ります。
<16BitAddrHexStr|64BitAddrHexStr> パラメータを指定した場合にはリプライデータには <TDCPPrefix>と <status> だけが入ります。

<16BitAddrHexStr|64BitAddrHexStr>パラメータに ‘0’ を指定した場合(“router,1,0” または “router,0,0”と入力）には、<index>で指定したルータアドレスが未設定の状態になります。

- **ルータアドレス幅とserver_addr のアドレス幅について**

  ルータアドレスを 2 つ使用してルーティングする場合には、router コマンドで指定するルータアドレスは 16bit 幅のアドレスを使用してください。また、この時の server_addr コマンドで設定するアドレスもできるだけ 16bit 幅で指定して下さい。これは、A/D 変換値を含むイベント情報などデータサイズが大きいパケットを転送するときに、ルーティングアドレス情報もデータパケット中に含めて送信するために、XBee データパケットサイズが制限値（100bytes）を超える場合があるためです。

- **ルーティング可能なデータパケットについて**

  router コマンドで設定したルータ経由で自動的に転送されるパケットデータは TDCP イベントパケットのみです。リクエストデータパケットやリプライデータパケットを、自動的にルータ経由で送信または受信することはできません。

リクエストコマンドについては、router コマンドを使用しなくても下記の様なコマンドデータを作成することで、途中に複数のルータを経由させてコマンドを実行することができます。
ルータとして指定する TDCP デバイスアドレスを “0A01”,”0B02”、最終的なコマンド実行先の TDCP アドレス

が “0C03” の場合のリクエストコマンドパケットは下記のようになります。

“$$$,tx,0B02,$$$,tx,0C03,$$$,port_write,FF”

上記のリクエストコマンドパケットを “0A01” のアドレスを持つ TDCP デバイスに送信します。”リクエスト元” -> ”0A01” -> “0B02” -> “0C03” の順にパケットが転送されて、”0C03” のアドレスを持つ TDCP デバイスで “port_write,FF” コマンドが実行されます。この時、”0A01”, “0B02” はそれぞれ “NEAR ROUTER”, “FAR ROUTER” の役割を持つことになります。

“port_write” コマンドが実行された “0C03” アドレスを持つTDCP デバイスのリプライデータパケットを、リクエスト元で受信することはできませんので注意して下さい。これは、”$$$,port_write,FF” 部分を ”$$$abc,port_write,FF” の様にしても、リプライパケットデータは “0C03” に送信されてリクエスト元には転送されない為です。ルータを経由してリプライパケットを受信する場合には “reply_prefix” コマンドを使用します。詳しくは “reply_prefix” の項を参照してください。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,router,0,0013A200404AC398” -> “$$$abc,1” “$$$12345,router,0” -> “$$$12345,1,0013A200404AC398”

制限事項

“router” コマンドで 2 つのルータを設定した場合に、送信間隔が 1秒未満の連続したイベント送信が失敗する場合があります。この場合には送信間隔を大きく(1秒以上)するか、使用するルータを 1 つにしてください。

7.52 reply_prefix,rp (TDCP専用)

- **機能概要**

  リモートTDCP コマンドのリプライパケット送信時に使用する<TDCPPrefix> の先頭に任意の文字列を挿入する。

  このコマンドは TDCPZB では使用できません

- **リクエストフォーマット**

Console	reply_prefix,<insert_prefix>
XBee	<TDCPPrefix>,reply_prefix,<insert_prefix>

- **リプライデータフォーマット**

Console	reply_prefix=<insert_prefix>
XBee	<TDCPPrefix>,<status>

- **パラメータとリターン値**

<TDCPPrefix>	“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列
“reply_prefix”	コマンド文字列。省略形式として “rp” を使用することもできる
<insert_prefix>	リモートTDCP コマンド実行時のリプライ送信時に使用する <TDCPPrefix> 文字列を、<insert_prefx> + “,” + <TDCPPrefix> に置き換える。 “0” を指定すると設定内容をクリアします。
<status>	“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

このコマンドで、<insert_prefix> を指定すると、リモートから TDCP コマンドを実行する時に使用する <TDCPPrefix> が自動的に全て置き換えられます。

例えば、<insert_prefx> に “$$$,tx,0002,$$$,tx,0001” を設定した場合には、
続くリモートからのTDCP コマンドが “$$$123,port_read” であった場合に、コマンド実行に成功した場合(ポート値が FFFF の場合)のリプライパケットは “$$$,tx,0002,$$$,tx,0001,$$$123,1,FFFF” になります。
この場合に <insert_prefix> が設定されていない場合のリプライパケットは “$$$123,1,FFFF”です。

このコマンドで設定した <insert_prefix> の内容は、設定後 10秒経つと自動的にクリアされます。
<insert_prefix> の内容を使用したリプライパケットを送信した場合には、最後に使用してから10秒後に自動的にクリアされます。”reply_prefix,0” を実行した場合には直ぐに設定がクリアされます。

注意

“reply_prefix” を使用するとTDCPコマンドリプライの送信先が一時的に固定されるため、別デバイスからコマンド送信を行う場合にリプライパケットの受信ができなくなります。このため、”reply_prefix” コマンドの設定内容は自動的にクリアされる仕様になっています。またこのコマンドを使用した後、目的となるルータ経由のコマンド実行を行ったら、できるだけすみやかに設定内容をマニュアルでクリアしてください。(”reply_prefix,0” を実行する)

注意

<insert_prefix> を設定するとその直後から有効になります。このため、リモートからの reply_prefix コマンド実行時の自身のリプライパケットも置き換えてしまうため、この時の<TDCPPrefix> は “$$$” を指定してください。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

リクエスト先のTDCP デバイスの途中に2つのルータ(TDCPデバイス)を経由してリモートコマンドを実行してリプライ値を取得する。
ルータのアドレスはリクエスト元に近い順からそれぞれ”0002”,”0003で、リクエスト元の アドレスが “0001”, リクエスト先のアドレスが “0004” の場合

“$$$,tx,0003,$$$,tx,0004,$$$,rp,$$$,tx,0002,$$$,tx,0001” -> ** No reply data **

“$$$,tx,0003,$$$,tx,0004,$$$abc,port_read” -> “$$$abc,1,FFFF”

（上記のリプライ受信時には “0004” から “0003” に対して下記のパケットが送信される）

“$$$,tx,0002,$$$,tx,0001,$$$abc,1,FFFF”

“$$$,tx,0003,$$$,tx,0004,$$$,rp,0” -> ** No reply data **

7.53 timezone

- **機能概要**

TDCP 内部時計(GMT) とローカル日時との時間差（分単位）の設定または取得

“config_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**

Console	timezone[,<timezone_min>]
XBee	<TDCPPrefix>,timezone[,<timezone_min>]

- **リプライデータフォーマット**

Console	timezone=<timezone_min>
XBee	<TDCPPrefix>,<status>[,<timezone_min>]

- **パラメータとリターン値**

<TDCPPrefix> “$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<timezone_min> GMT からの時間差を分単位で指定する。

日本(JST:GMT- 9時間) の場合には-540 になる(デフォルト設定値:-540)

<status> “1” でコマンド実行成功、”0” で失敗を表す

- **備考**

TDCP の “datetime” コマンドで使用する日付時刻の GMT からの時間差を設定または取得します。

TDCP 内部で保持している時計が GMT のため “datetime” コマンドで日付時刻を設定する場合には、”timezone” コマンドで設定した時間差を考慮して内部の時計が設定されます。また、”datetime” コマンドで現在の日付時刻を取得するときにも、内部時計から “timezone” で設定された時間差を考慮してローカル日時が計算されます。

リクエストコマンド(XBee データパケット)で <timezone_min>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。<timezone_min> パラメータを指定した場合にはリプライデータには

<TDCPPrefix>と<status> だけが入ります。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,timezone” -> “$$$abc,1,-540”
“$$$12345,timezone,-540” -> “$$$12345,1”

7.54 datetime

- **機能概要**

TDCP のローカル日時の設定または取得

- **リクエストフォーマット**

Console	datetime[,<year,month,day,hour,minute,second>]
XBee	<TDCPPrefix>,timezone[,<year,month,day,hour,minute,second>]

- **リプライデータフォーマット**

Console	datetime=<year,month,day,hour,minute,second>
XBee	<TDCPPrefix>,<status>[,<year,month,day,hour,minute,second>]

- **パラメータとリターン値**

<TDCPPrefix> “$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<year,month,day,hour,minute,second>

ローカル日時の年、月、日、時、分、秒を指定する。

year は ４桁の西暦年を入れる

month は 1 から 12 までの整数

day は 1から 31 までの整数

hour は 0 から 23 までの整数

minute は 0 から 59 までの整数

second は 0 から 59 までの整数

<status> “1” でコマンド実行成功、”0” で失敗を表す

- **備考**

TDCP の内部のローカル日時を設定または取得します。

TDCP 内部で保持している時計はGMTでEpoch 秒を内部でカウントしています。この時計を元に ”timezone ”コマンドで設定した時間差を考慮して、ローカル日時の設定または表示を行います。

“datetime” コマンドで時刻を設定すると、内部時計がその値を元に設定されます。その後、内部時計は自動的に１秒ごとにインクリメントされて現在時刻が更新されます。

内部時計の時間精度は 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。クリスタルの発振周波数は数パーセントの誤差がありますので、内部時計はあまり正確な時間を刻みません。TDCP ではある程度(1/1000 秒)までの秒補正を “second_adjust” コマンドで設定することが可能ですので、詳しくは ”コマンドリファレンス” の章の “second_adjust” コマンドの項を参照してください。

実用的に内部時計を使用するためには、定期的に “datetime” コマンドで正確な時刻をサーバーなどから設定する必要があります。

リクエストコマンド(XBee データパケット)で <year,month,day,hour,minute,second>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。<year,month,day,hour,minute,second>パラメータを指定した場合にはリプライデータには <TDCPPrefix>と<status> だけが入ります。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,datetime” -> “$$$abc,1,2011,1,2,22,33,0”
“$$$12345,datetime,2011,4,22,11,22,4” -> “$$$12345,1”

7.55 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

  “config_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**

Console	signal_timer[,<timer>]
XBee	<TDCPPrefix>,signal_timer[,<timer>]

- **リプライデータフォーマット**

Console	signal_timer=<timer>
XBee	<TDCPPrefix>,<status>[,<timer>]

- **パラメータとリターン値**

  <TDCPPrefix>　　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<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。 (デフォルト設定値:"0")
<status>	"1" でコマンド実行成功、"0" で失敗を表す

- **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。

シグナルステータスとTDCP のポート出力値の説明は、"シグナル出力機能" の章を参照して下さい。

リクエストコマンド(XBee データパケット)で <timer>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。<timer>パラメータを指定した場合にはリプライデータには <TDCPPrefix>と<status>だけが入ります。

- **XBee データパケットの使用例("リクエストコマンドデータ" -> "リプライデータ")**

"$$$abc,signal_timer" -> "$$$abc,1,0"
"$$$12345,signal_timer,60" -> "$$$12345,1"

7.56 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

"config_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**

Console	signal_event[,<signal_state>]
XBee	<TDCPPrefix>,signal_event[,<signal_state>]

- **リプライデータフォーマット**

Console	signal_event=<signal_state>

XBee	<TDCPPrefix>,<status>[,<signal_state>]

- **パラメータとリターン値**

<TDCPPrefix>　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<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。

(デフォルト設定値:”0”)

<status>　“1” でコマンド実行成功、”0” で失敗を表す

- **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。

”TDCP 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 0x0 に設定されています。

シグナルステータスとTDCP のポート出力値の説明は、”シグナル出力機能” の章を参照して下さい。

リクエストコマンド(XBee データパケット)で <signal_state>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。<signal_state>パラメータを指定した場合にはリプライデータには <TDCPPrefix>と<status> だけが入ります。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,signal_event” -> “$$$abc,1,00”
“$$$12345,signal_event,0x18” -> “$$$12345,1”

7.57　ad_margin

- **機能概要**

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

“config_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**

Console	ad_margin,<channel>[,<margin>]
XBee	<TDCPPrefix>,ad_margin,<channel>[,<margin>]

- **リプライデータフォーマット**

Console	ad_margin,<channel>=<margin>
XBee	<TDCPPrefix>,<status>[,<margin>]

- **パラメータとリターン値**

<TDCPPrefix>　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<channel>　A/D チャンネルを指定する。0 から 7 までの整数値
app_mode によって、指定可能なチャンネルが異なります。

<margin>　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A/D チャンネルを指定する。
0 を指定した場合は A/D変換値の変化を検出しない。

<status>　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

<margin> 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>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 <ad_margin> 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 <ad_margin> 値を小さくしていきます。最初から <ad_margin> に小さい値を指定すると、イベントが連続発生して XBee のパケット転送や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。

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

リセット直後の全てのA/D チャンネルの <ad_margin>の初期値は 0 に設定されています。

リクエストコマンド(XBee データパケット)で <margin>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。<margin>パラメータを指定した場合にはリプライデータには <TDCPPrefix>と <status> だけが入ります。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,ad_margin,0” -> “$$$abc,1,10”
“$$$12345,ad_margin,0,5” -> “$$$12345,1”

7.58 ad_margin_reset

- **機能概要**

  A/D 変換値変化(ADVAL_UPDATE)イベント検出時に比較する前回のイベント送信時のA/D値を、最新のA/D値で上書きする

- **リクエストフォーマット**

Console	ad_margin_reset
XBee	<TDCPPrefix>,ad_margin_reset

- **リプライデータフォーマット**

Console	** No reply data **
XBee	<TDCPPrefix>,<status>

- **パラメータとリターン値**

  <TDCPPrefix>　　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

  <status>　　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

  このコマンドを実行すると、A/D 変換値変化(ADVAL_UPDATE)イベント検出時に比較する前回のイベント送信時のA/D値を、最新のA/D値で上書きして累積の変化量をリセットします。このコマンドは 全 A/D チャンネルの比較値をリセットします。

  日の出や日の入りの日照変化など、ゆっくりと変化する対象を ADVAL_UPDATE イベントの検出対象としたくない場合に使用します。定期的にサーバーからこのコマンドを実行すると、累積した変化量をリセットすることができ、急激なA/D 値の変化が ″ad_margin″ コマンドで設定した値をを超えたときにのみイベントを検出することができます。

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

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,ad_margin_reset” -> “$$$abc,1”

7.59 i2c_use

- **機能概要**

TDCP の I2C 機能を有効にするフラグ値の設定または取得

“config_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**

Console	i2c_use[,<flag>]
XBee	<TDCPPrefix>,i2c_use[,<flag>]

- **リプライデータフォーマット**

Console	i2c_use=<flag>
XBee	<TDCPPrefix>,<status>[,<flag>]

- **パラメータとリターン値**

<TDCPPrefix>	“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列
<flag>	TDCP のI2C 機能を使用するかどうかの設定値を指定する。 0: I2C 機能を使用しない 1: I2C 機能を使用する。SCL, SDA ピンのプロセッサ内部のプルアップ無効 2: I2C 機能を使用する。SCL, SDA ピンのプロセッサ内部のプルアップ有効 (デフォルト設定値:0)
<status>	“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

TDCP の I2C 機能は、 マイクロプロセッサ内部の TWI モジュール機能を使用します。

データ通信速度は 約 100Kbps (Standard Mode) で固定です。<flag> を 0以外に設定すると、TDCP の I2C 機能が有効になり、I2C マスター機能のコマンド “i2c_write”, “i2c_read” コマンドが使用できます。

TDCP をI2C スレーブデバイスとしても動作させる場合には ”i2c_slave_addr” コマンドで、スレーブアドレスを設定しておく必要があります。

<flag> 値を変更した場合には、リセット後に有効になります。そのため、“i2c_use” コマンドに続けて”config_save” コマンドを実行してEEPROM に設定を保存した後、“reset” コマンドを実行してください。

i2c_use の <flag> 値を 0 以外の値に設定した場合にはリセット後 I2C 機能が有効になり、PORTC のビット0とビット1のピンが、それぞれ I2C コントロール用の “SCL” と ”SDA” に割り当てられます。また現在の

app_mode でそのピンに割り当てられている機能は使用できなくなります。

<flag> を 1 に設定した場合には、"SCL" と "SDA" ピン内部のプルアップが無効になります。この時、PORTC のビット0とビット1 に設定された "pullup" コマンドの設定値は無視されます。外部の回路で適切なプルアップ抵抗を接続して下さい。

<flag> を 2 に設定した場合には、"SCL" と "SDA" ピン内部のプルアップが有効になります。この時、PORTC のビット0とビット1 に設定された "pullup" コマンドの設定値は無視されます。

"i2c_use" 設定コマンド実行例

```
i2c_use,1
config_save
reset
```

リクエストコマンド(XBee データパケット)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。<flag> パラメータを指定した場合にはリプライデータには <TDCPPrefix>と<status> だけが入ります。

- **XBee データパケットの使用例("リクエストコマンドデータ" -> "リプライデータ")**

```
"$$$abc,i2c_use,1" -> "$$$abc,1"
"$$$12345,i2c_use" -> "$$$12345,1,1"
```

7.60 i2c_read

- **機能概要**

I2C に接続されたスレーブデバイスからデータを取得する。

- **リクエストフォーマット**

Console	i2c_read,<SlaveAddrHexStr>,<length>
XBee	<TDCPPrefix>,i2c_read,<SlaveAddrHexStr>,<length>

- **リプライデータフォーマット**

Console	** I2C スレーブデバイスから読み込まれたデータ内容がダンプされます **
XBee	<TDCPPrefix>,<status>,<ReadDataHexStr>

- **パラメータとリターン値**

<TDCPPrefix>	“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列
<SlaveAddrHexStr>	I2C スレーブデバイスアドレスを16進数表記で指定する。 7 bit アドレスが指定可能で R/W ビットフィールドは含めないでアドレス部分だけを指定する。 例:50,0x07
<length>	読み込むバイト数を 10進数表記で指定する。このパラメータで指定可能な最大バイト数は 32 まで。 例:10,2,32
<ReadDataHexStr>	スレーブデバイスから読み込んだデータが16進数表記で出力される。
<status>	“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

スレーブデバイスアドレスで指定した I2C デバイスから <length>で指定したバイト数だけデータを読み込みます。

I2C の “MASTER RECEIVER MODE” でデータを取得します。

このコマンドを使用する場合は、予め “i2c_use” コマンドでTDCPの I2C 機能を有効にしておく必要があります。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,i2c_read,05,16” -> “$$$abc,1,010A00000000010013A200404AC398C0”

7.61 i2c_write

- **機能概要**

I2C に接続されたスレーブデバイスにデータを書き込む。

データ書き込み後、指定したバイト数分のデータを同一スレーブデバイスから読み込むことができる

- **リクエストフォーマット**

Console	i2c_write,<SlaveAddrHexStr>,<DataHexStr>[,<length>]
XBee	<TDCPPrefix>,i2c_write,<SlaveAddrHexStr>,<DataHexStr>[,<length>]

- **リプライデータフォーマット**

Console	** <length>パラメータを指定した場合には、I2C スレーブデバイスから読み込まれたデータ内容がダンプされます **
XBee	<TDCPPrefix>,<status>[,<ReadDataHexStr>]

- **パラメータとリターン値**

<TDCPPrefix>	“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列
<SlaveAddrHexStr>	I2C スレーブデバイスアドレスを16進数表記で指定する。 7 bit アドレスが指定可能で R/W ビットフィールドは含めないでアドレス 部分だけを指定する。 例:50,0x07
<DataHexStr>	スレーブデバイスに書き込むデータを16進数表記で指定する。 このパラメータで指定可能な最大バイト数は 32(16進文字列で64文字)まで 16進数表記を表す ”0x” 部分は必ず除いて、バイト単位(2文字)で指定する。 例: F1F2F3,0000,01020304,00
<length>	データ書き込み後に、デバイスから読み込むバイト数を 10進数表記で指定する。 このパラメータで指定可能な最大バイト数は 32 まで。 このパラメータを省略した場合には書き込み動作のみが実行される。 例:10,2,32
<ReadDataHexStr>	スレーブデバイスから読み込んだデータが16進数表記で出力される。
<status>	“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

スレーブデバイスアドレスで指定したI2C デバイスに指定したデータを書き込みます。

I2C の “MASTER TRANSMITTER MODE” でデータを送信します。

<length> パラメータを指定した場合には、データ書き込み後に同一スレーブデバイスからのデータ取得を行います。このときは、”REPEATED START MODE” で I2C バス操作を行います。

このコマンドを使用する場合は、予め “i2c_use” コマンドでTDCPの I2C機能を有効にしておく必要があります。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

```
// EEPROM Atmel24LC256 デバイスへのデータ書き込み(デバイスの A0,A1,A2 ピンは “low”)
“$$$abc,i2c_write,50,000101” -> “$$$abc,1”
“$$$abc,i2c_write,50,000202” -> “$$$abc,1”
“$$$abc,i2c_write,50,000303” -> “$$$abc,1”

// EEPROM Atmel24LC256 デバイスからのデータ読み込み(1)
“$$$abc,i2c_write,50,0001” -> “$$$abc,1”
“$$$abc,i2c_read,50,3” -> “$$$abc,1,010203”

// EEPROM Atmel24LC256 デバイスからのデータ読み込み(2)
```

“$$$abc,i2c_write,50,0001,3” -> “$$$abc,1,010203”

7.62 i2c_slave_addr

- **機能概要**

TDCP の I2C スレーブアドレスの設定または取得。

“config_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**

Console	i2c_slave_addr[,<8BitAddrHexStr>]
XBee	<TDCPPrefix>,i2c_slave_addr[,<8BitAddrHexStr>]

- **リプライデータフォーマット**

Console	i2c_slave_addr=<8BitAddrHexStr>
XBee	<TDCPPrefix>,<status>[,<8BitAddrHexStr>]

- **パラメータとリターン値**

<TDCPPrefix> “$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<8BitAddrHexStr>

I2C スレーブアドレスを8bit幅の16進数形式の文字列で記述する。LSB側の 7 ビットのみが有効で、ここで指定したスレーブアドレスは 1 ビット左シフトされてI2Cバスに出力される。(例: “30”, “28”)

“0” を指定することで、TDCP の I2Cスレーブ機能を無効にします。

(デフォルト設定値:”0”)

<status> “1” でコマンド実行成功、”0” で失敗を表す

- **備考**

I2C スレーブ機能を使用する場合に、TDCP が動作するCPU のスレーブアドレスを設定します。”i2c_use” コマンドと共に使用して、I2C 機能を有効に設定したときにスレーブ機能も使用可能になります。

リクエストコマンド(XBee データパケット)で <8BitAddrHexStr>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。この時、アドレス値が未設定の場合にはリプライデータの<8BitAddrHexStr>に “0” <status> に “1” が返ります。<8BitAddrHexStr> パラメータを指定した場合にはリプライデータには <TDCPPrefix>と<status> だけが入ります。

<8BitAddrHexStr>パラメータに ‘0’ を指定した場合(“i2c_slave_addr,0” と入力) には、I2C スレーブ機能は無効になります。

<8BitAddrHexStr> 値を変更した場合には、リセット後に有効になります。”i2c_slave_addr” コマンドでスレーブアドレスを設定した場合には、“i2c_use” コマンドでI2C 機能も有効に設定します。その後、”config_save” コマンドを実行してEEPROM に設定を保存した後、“reset” コマンドを実行して TDCP を再起動してください。

TDCP の I2Cスレーブ機能の詳細は “I2C 接続機能” の章を参照してください。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,i2c_use,2” -> “$$$abc,1”
“$$$abc,i2c_slave_addr,0x30” -> “$$$abc,1”
“$$$abc,config_save” -> “$$$abc,1”
“$$$,reset” -> ** No reply data **

7.63 i2c_work_read

- **機能概要**

I2C スレーブ機能で操作可能な I2C ワークエリアのデータを取得する。

I2C ワークエリアは TDCP プログラムでマイクロプロセッサ内のメモリ中に作成されています。

- **リクエストフォーマット**

Console	i2c_work_read,<WorkAddrHexStr>,<length>
XBee	<TDCPPrefix>,i2c_work_read,<WorkAddrHexStr>,<length>

- **リプライデータフォーマット**

Console	** I2C スレーブ機能のワークエリア内容がダンプされます **
XBee	<TDCPPrefix>,<status>,<DataHexStr>

- **パラメータとリターン値**

<TDCPPrefix>	“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列
<WorkAddrHexStr>	ワークエリア内のアドレスを16進数表記で指定する。ワークエリアは 256 バイト長で 0x00 から 0xFF までのアドレスを指定可能。 例: 0A,0x0F
<length>	読み込むバイト数を 10進数表記で指定する。指定可能な最大バイト数は 32。 例: 10,32
<DataHexStr>	読み込んだ ワークエリアデータを16進数表記で表示される。
<status>	“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

  I2C スレーブ機能でアクセス可能なワークエリアの内容を TDCP コマンドで取得します。

  ワークエリアは 256 バイト分操作可能で、<WorkAddrHexStr> で指定したアドレスから連続して<length>バイト分のデータを取得します。データ取得中に、ワークエリアの終端に達した場合には 0 番地から引き続き読み込まれます。

  TDCP の I2Cスレーブ機能の詳細は “I2C 接続機能” の章を参照してください。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,i2c_work_read,10,16” -> “$$$abc,1,010A00000000010013A200404AC398C0”

7.64 i2c_work_write

- **機能概要**

  I2C スレーブ機能で操作可能な I2Cワークエリアにデータを書き込む。

  I2C ワークエリアは TDCP プログラムでマイクロプロセッサ内のメモリ中に作成されています。

- **リクエストフォーマット**

Console	i2c_work_write,<WorkAddrHexStr>,<DataHexStr>
XBee	<TDCPPrefix>,i2c_work_write,<WorkAddrHexStr>,<DataHexStr>

- **リプライデータフォーマット**

Console	** No reply data **
XBee	<TDCPPrefix>,<status>

- **パラメータとリターン値**

<TDCPPrefix>	“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列
<WorkAddrHexStr>	ワークエリア内のアドレスを16進数表記で指定する。ワークエリアは 256 バイト長で 0x00 から 0xFF までのアドレスを指定可能。 例: 0A,0x0F
<DataHexStr>	ワークエリアに書き込むバイナリデータを16進数表記で指定する。 16進数表記を表す ”0x” 部分は必ず除いて、バイト単位(2文字)で指定する。 例: F1F2F3,0000,01020304,00
<status>	“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

  I2C スレーブ機能でアクセス可能なワークエリアの内容を TDCP コマンドで設定します。

  ワークエリアは 256 バイト分操作可能で、<WorkAddrHexStr> で指定したアドレスから <DataHexStr> で指定したデータを連続して書き込みます。データ書き込み中に、ワークエリアの終端に達した場合には 0 番地から引き続き書き込まれます。

  TDCP の I2Cスレーブ機能の詳細は “I2C 接続機能” の章を参照してください。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,i2c_work_write,20,01020304” -> “$$$abc,1”

7.65 spi_config

- **機能概要**

  TDCP の SPI 機能の動作条件の設定または取得

  “config_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**

Console	spi_config[,<spi_use>[,<clock_div>,<spi_mode>,<LSB_first>,<pulse_polarity>]]
XBee	<TDCPPrefix>,spi_config[,<spi_use>[,<clock_div>,<spi_mode>,<LSB_first>,<pulse_polarity>]]

- **リプライデータフォーマット**

Console	spi_config=<spi_use>,<clock_div>,<spi_mode>,<LSB_first>,<pulse_polarity>
XBee	<TDCPPrefix>,<status>[,<spi_use>,<clock_div>,<spi_mode>,<LSB_first>,<pulse_polarity>]

- **パラメータとリターン値**

  <TDCPPrefix> “$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

  <spi_use> TDCP のSPI 機能を使用するかどうかの設定値を指定する。

  0: SPI 機能を使用しない

  1: SPI 機能を使用する。

  (デフォルト設定値: 0)

  <clock_div> SPI動作に使用するクロックは マイクロプロセッサのCPU クロックを、<clock_div> で分周した値が使用される。指定可能な値(2, 4, 8, 16, 32, 64, 128)

  (デフォルト設定値: CPUクロック周波数が16MHz の場合には 16, CPUクロック周波

	数が8MHz の場合には 8に設定されて、1MHz 動作に初期設定されています)
<spi_mode>	SPI のクロック位相と極性のモードを指定する。
	(各モード詳細はAtmel社の ATMega1284P の仕様書をご覧下さい)
	0: Clock Polarity = 0, Clock Phase = 0
	1: Clock Polarity = 0, Clock Phase = 1
	2: Clock Polarity = 1, Clock Phase = 0
	3: Clock Polarity = 1, Clock Phase = 1
	(デフォルト設定値: 0)
<LSB_first>	シリアルデータのビット送信順序を指定する。
	0: MSB を最初に送信
	1: LSB を最初に送信
	(デフォルト設定値: 0)
<pulse_polarity>	“spi_write” コマンドでパルス出力を行う場合の極性を指定する。
	0: Low パルス信号を出力。常時 PORTC bit#6のレベルは High。
	1: High パルス信号を出力。常時 PORTC bit#6のレベルは Low。
	(デフォルト設定値: 0)
<status>	“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

TDCP の SPI 機能は マイクロプロセッサの SPIモジュールを使用しています。TDCP では SPI のマスターモードだけが使用できます。

“spi_config” コマンドで設定値を変更した場合には、コマンド実行直後に設定値が有効になります。

<spi_use> を 1 に設定すると、マイクロプロセッサの “SS” (Slave Select) 端子には常に High が出力されて、SPI 通信中には Low が出力されます。

リクエストコマンド(XBee データパケット)で、全ての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。パラメータを指定した場合にはリプライデータには <TDCPPrefix>と<status> だけが入ります。

リクエストパラメータで <spi_use> だけを指定した場合には、他のパラメータ値は現在の設定値を使用して、<spi_use> のみが更新されます。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,spi_confg,1,64,0,0,1” -> “$$$abc,1”
“$$$12345,spi_config” -> “$$$12345,1,1,64,0,0,1”

7.66 spi_read

- **機能概要**

SPI に接続されたスレーブデバイスからデータを取得する。

- **リクエストフォーマット**

Console	spi_read,<length>
XBee	<TDCPPrefix>,spi_read,<length>

- **リプライデータフォーマット**

Console	** SPI スレーブデバイスから読み込まれたデータ内容がダンプされます **
XBee	<TDCPPrefix>,<status>,<ReadDataHexStr>

- **パラメータとリターン値**

<TDCPPrefix>	“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列
<length>	読み込むバイト数を 10進数表記で指定する。このパラメータで指定可能な最大バイト数は 32 まで。 例:10,2,32
<ReadDataHexStr>	スレーブデバイスから読み込んだデータが16進数表記で出力される。
<status>	“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

SPI スレーブデバイスから <length>で指定したバイト分のデータを読み込みます。

予め、対象となるSPI スレーブデバイスの “SS” (Slave Select) 端子をアクティブに設定しておく必要があります。TDCP では、SPI 転送開始前にマイクロプロセッサの “SS” 端子に Low が出力されます。全ての転送終了後に “SS” 端子に High が出力されます。１つのスレーブデバイスを接続している場合にはこのマイクロプロセッサの “SS” 信号を利用できます。複数のスレーブデバイスを使用するときは、”port_bit”コマンドを使用して出力したポート信号を スレーブデバイスの “Slave Select” 信号に利用できます。

データを取得するときに、スレーブデバイスには <length> で指定されたバイト数分のゼロデータ(0x00) がマスターから送信されます。マスターから送信するデータを 0x00 以外に指定したい場合は、”spi_write” コマンドを使用してください。

このコマンドを使用する場合は、予め “spi_confg” コマンドで <spi_use> を1 に設定して、TDCPの SPI機能を有効にしておく必要があります。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$abc,i2c_read,6” -> “$$$abc,1,010203040506”

7.67 spi_write

- **機能概要**

SPI に接続されたスレーブデバイスにデータを書き込む。書き込みと同時に、スレーブデバイスから送信されたデータを取得する。

- **リクエストフォーマット**

Console	spi_write,<DataHexStr>[,<latch_pulse>]
XBee	<TDCPPrefix>,spi_write,<DataHexStr>[,<latch_pulse>]

- **リプライデータフォーマット**

Console	** SPI スレーブデバイスから読み込まれたデータ内容がダンプされます **
XBee	<TDCPPrefix>,<status>,<ReadDataHexStr>

- **パラメータとリターン値**

<TDCPPrefix>	“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列
<DataHexStr>	スレーブデバイスに書き込むデータを16進数表記で指定する。 このパラメータで指定可能な最大バイト数は 32(16進文字列で64文字)まで 16進数表記を表す ”0x” 部分は必ず除いて、バイト単位(2文字)で指定する。 例: F1F2F3,0000,01020304,00
<ReadDataHexStr>	スレーブデバイスから読み込んだデータが16進数表記で出力される。
<latch_pulse>	データ転送後に PORTC bit#6 にパルス信号を出力する。 0: パルス信号を出力しない 1: パルス信号を出力する。“SS” 信号アクティブ中に出力 2: パルス信号を出力する。“SS” 信号インアクティブ後に出力 (デフォルト設定値: 0)
<status>	“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

SPI スレーブデバイスに指定したデータを書き込みます。

TDCP では、SPI 転送開始前にマイクロプロセッサの “SS” 端子に Low が出力されます。全ての転送終了後に “SS” 端子に High が出力されます。１つのスレーブデバイスを接続している場合にはこのマイクロプロセッサ

の “SS” 信号を利用できます。複数のスレーブデバイスを使用するときは、”port_bit”コマンドを使用して出力したポート信号を スレーブデバイスの “Slave Select” 信号に利用できます。

〈latch_pulse〉 を 1 に指定すると、シリアルデータ出力後に PORTC bit#6ピンに 約2マイクロ秒幅のパルス信号が出力されます。信号は “spi_config” コマンドの 〈pulse_polarity〉 で設定した極性のパルス信号が出力されます。シリアルデータ出力とパルス信号出力中はマイクロプロセッサの “SS” には常に Lowが出力されています。〈latch_pulse〉 を 2 に指定した場合には、全てのシリアルデータ出力後に “SS”信号が High になった後でパルス信号が出力されます。

書き込みと同時に、スレーブデバイスからデータを取得します。スレーブデバイスに書き込んだデータと同じバイト数のデータをスレーブデバイスから取得します。

このコマンドを使用する場合は、予め “spi_config” コマンドで 〈spi_use〉 を1 に設定して、TDCPの SPI機能を有効にしておく必要があります。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

```
// MCP4922 D/A デバイスへのデータ書き込み
“$$$abc,spi_write,3FFF” -> “$$$abc,1,0000”

// 74HC595AP x2 (16bit) シリアルパラレル変換デバイスにデータ書き込み
// ラッチ出力パルスを 74HC595AP のRCK 入力に接続。
// Atmega1284p のSS出力を 74HC595AP のRCK 入力に使用する場合には、
// spi_write コマンドのラッチパルス出力パラメータは不用
“$$$abc,spi_write,1111,2” -> “$$$abc,1,0000”
```

7.68 network_address (TDCPZB専用)

- **機能概要**

  XBee-ZB モジュールの現在のネットワークアドレスを取得

- **リクエストフォーマット**

Console	network_address
XBee	** Not available **

- **リプライデータフォーマット**

Console	〈XBee-ZB network address〉

XBee	** Not available **

- **パラメータとリターン値**

  <XBee-ZB network address> XBee-ZB モジュールにアサインされている16ビットネットワークアドレス。

- **備考**

  このコマンドはシリアルコンソール専用です。

  リモートから XBee-ZB デバイスのネットワークアドレスを取得する場合には、XBee-ZB から送信された API イベントフレームや AT コマンドのレスポンスフレーム中に格納されているネットワークアドレス情報を利用してください。DeviceServer では zb_find_device(), zb_all_list() ライブラリ関数でこれらのネットワークアドレス情報を取得できます。

7.69 association_indication (TDCPZB専用)

- **機能概要**

  XBee-ZB モジュールのアソシエーション情報を取得

- **リクエストフォーマット**

Console	association_indication
XBee	** Not available **

- **リプライデータフォーマット**

Console	<XBee-ZB association_indication>
XBee	** Not available **

- **パラメータとリターン値**

  <XBee-ZB association_indication> XBee-ZB モジュールの association indication 情報。

- **備考**

  このコマンドはシリアルコンソール専用です。

  リモートから XBee-ZB デバイスのassociation indication情報を取得する場合には、リモート AT コマンド "AI" を実行して結果を取得できます。DeviceServer では zb_at_command()ライブラリ関数で "AI" リモートコマンドを実行して結果を取得できます。

7.70 network_reset (TDCPZB専用)

- **機能概要**

XBee-ZB モジュールのネットワーク情報をリセットする

- **リクエストフォーマット**

Console	network_reset
XBee	<TDCPPrefix>,network_reset

- **リプライデータフォーマット**

Console	
XBee	<TDCPPrefix>,<status>

- **パラメータとリターン値**

<TDCPPrefix>　　“$$$” または $$$*****、”*****” は省略可能で、指定時は5文字以内の英数字文字列

<status>　　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

このコマンドは、XBee-ZB モジュールで ″NR″ ATコマンドを実行するのと同様です。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$aaa,network_reset” -> “$$$aaa,1”

7.71　　disassociation (TDCPZB専用)

- **機能概要**

XBee-ZB モジュールを ZigBee ネットワークから切り離す

- **リクエストフォーマット**

Console	disassociation
XBee	<TDCPPrefix>,disassociation

- **リプライデータフォーマット**

Console	
XBee	** Not available **

- **パラメータとリターン値**

<TDCPPrefix>　　“$$$”

<status>　　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

  このコマンドは XBee-ZB モジュールで ″DA″ ATコマンドを実行するのと同様です。

  このコマンドを実行すると、デバイスがZigBee ネットワークから切り離されますのでリモートからはリプライメッセージを取得することはできません。このため、必ず <TDCOPrefix> は ″$$$″ のみを指定して下さい。

- **XBee データパケットの使用例(“リクエストコマンドデータ” -> “リプライデータ”)**

“$$$,disassociation” -> ** No reply data **

8 TDCPイベントリファレンス

TDCP では、予め決められた I/O ポート値が変化した場合や、A/D 変換値が制限値を超えた場合などにイベントデータを XBee データパケットで送信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。

予め設定した時間間隔で自動的にサンプリング行って、イベントデータを送信することも可能です。

GPS レシーバを接続した場合には、NMEA-0183 センテンス受信時にも同様に、イベントデータを送信することができますので、リモートから TDCP デバイスを組み込んだ装置の測位情報を随時取得することができます。

イベント発生条件や、イベント送信先等はすべてリモートから操作・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

“server_addr” コマンドでデータ送信先アドレスが未設定の場合は、TDCP イベントデータの送信機能は使用できません。また、サンプリングイベントを使用する場合には、”sampling_rate” コマンドでサンプリング間隔を秒単位で設定して下さい。”sampling_rate” が 0 の場合 (default) にはサンプリングイベントは発生しません。

注意

TDCP 内部では、”interval” コマンドで設定した間隔(default:10ms) 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 “server_addr” コマンド設定した XBee デバイス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。

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。

“signal_event” で設定したシグナル更新の対象イベントです。

- **イベントデータフォーマット**

TDCP

```
$$$,CHANGE_DETECT,<my_addr16>,<app_mode>,<diff_bits>,<dio>
```

TDCPZB

```
$$$,CHANGE_DETECT,<app_mode>,<diff_bits>,<dio>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は 現在の “app_mode” 値が設定されます。

<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進数表記で設定されます。”app_mode” によってデータビット数が違いますので詳しくは該当する “app_mode” の “TDCP動作モード” の章を参照して下さい。

<dio> には 現在のポート値が16進数表記で設定されます。”app_mode” によってデータビット数が違いますので詳しくは該当する “app_mode” の “TDCP動作モード” の章を参照して下さい。

- 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
“interval”	イベント検出間隔を設定

- **備考**

イベントの発生条件になる毎にに、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、”interval” コマンドで設定した間隔(default: 10ms) 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。

8.2 RANGE_EXCEEDイベント

- **イベント発生条件**

A/D 入力ポートの値が予め決められた制限値(上限または下限)を越えた場合に発生します。

監視対象のA/D 入力ポートのビット位置は “range_check_high”,”range_check_low” コマンドで設定します。

監視対象の各 A/D 入力ポートの制限値は、”range_high”,”range_low” コマンドで設定します。

“signal_event” で設定したシグナル更新の対象イベントです。

- **イベントデータフォーマット**

TDCP

```
$$$,RANGE_EXCEED,<my_addr16>,<app_mode>,<high_exceed_bits>,<low_exceed_bits>
```

TDCPZB

```
$$$,RANGE_EXCEED,<app_mode>,<high_exceed_bits>,<low_exceed_bits>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は 現在の “app_mode” 値が設定されます。

<high_exceed_bits> には、”range_check_high” コマンドで指定したチェック対象の中で、”range_high” コマンドで指定した上限値を超えた A/D ポートのビットを 1、制限値内のビットを 0にした値が、16進数表記で設定されます。

<low_exceed_bits> には、”range_check_low” コマンドで指定したチェック対象の中で、”range_low” コマンドで指定した下限値を超えた A/D ポートのビットを 1、制限値内のビットを 0にした値が、16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 “TDCPコマンドリファレンス” の章を参照してください)

“range_check_high”	A/D 制限値上限を設定する
“range_check_low”	A/D 制限値下限を設定する
“range_high”	A/D 制限値上限の検出対象ビットを設定する
“range_low”	A/D 制限値下限の検出対象ビットを設定する
“interval”	イベント検出間隔を設定

- **備考**

イベントの発生条件になる毎にに、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。

RANGE_EXCEED イベントが発生すると、<high_exceed_bits>と<low_exceed_bits>で、”1” がセットされたビットは監視対象から外されます。これは “range_check_high”,”range_check_low” コマンドで対象ビットを “0” にセットすることに相当します。このため、もう一度同じ A/D 入力ポートのビットで、同じ制限(上限または下限)をチェックしてイベントを発生させたい場合は、”range_check_high”,”range_check_low” コマンドで対象ビットを “1” にセットして下さい。

8.3 COUNT_EXCEEDイベント

- **イベント発生条件**

TDCP の ”app_mode” で入力ポート（カウント入力）に指定された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。

監視対象の入力ポート(カウント入力)は “app_mode” によって予め決められています。詳しくは “TDCP動作モード” の章を参照してください。

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

“signal_event” で設定したシグナル更新の対象イベントです。

- **イベントデータフォーマット**

TDCP

```
$$$,COUNT_EXCEED,<my_addr16>,<app_mode>,<change_count>
```

TDCPZB

```
$$$,COUNT_EXCEED,<my_addr16>,<app_mode>,<change_count>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は 現在の app_mode 値が設定されます。

<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 “TDCPコマンドリファレンス” の章を参照してください)

“change_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
“change_count_high”	カウンタ制限値上限を設定する
“change_count_reset”	現在のカウンタ値をクリアする(0 を設定)
“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
“interval”	イベント検出間隔を設定

- **備考**

CHANGE_COUNT イベントを使用する場合には、”change_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。TDCP リセット後の 初期値は 0 で、ディセーブル状態です。

change_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。

- **change_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 (change_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。(change_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、”change_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。

- **change_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、”change_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。

8.4 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(秒)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**

イベントデータフォーマットはイベント発生時の “app_mode” によって格納されるサンプリングデータが異なります。代表的なサンプリングデータフォーマットを下記に列挙します。ここで記載した以外のデータフォーマットもありますので、”TDCP動作モード” の章と、その中の “app_mode” 毎の サンプリングイベントデータの項目を参照してください。

TDCP

$$$,SAMPLING,<my_addr16>,<app_mode>,<dio>
$$$,SAMPLING,<my_addr16>,<app_mode>,<dio>,<adc0>,<adc1>,<adc2>,<adc3>,<adc4>,<adc5>,<adc6>,<adc7>

TDCPZB

$$$,SAMPLING,<app_mode>,<dio>
$$$,SAMPLING,<app_mode>,<dio>,<adc0>,<adc1>,<adc2>,<adc3>,<adc4>,<adc5>,<adc6>,<adc7>

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。
<app_mode> は 現在の “app_mode” 値が設定されます。
<dio> には 現在のポート値が16進数表記で設定されます。”app_mode” によってデータビット数が違いますので詳しくは該当する “app_mode” の “TDCP動作モード” の章を参照して下さい。
<adc0>.. <adc7> 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 “TDCP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(“change_count_check”,”change_count_reset”,”change_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。また、TDCP ではある程度(1/1000 秒)までの秒補正を “second_adjust” コマンドで設定することが可能ですので、詳しくは ”コマンドリファレンス” の章の “second_adjust” コマンドの項を参照してください。ただしこの場合でも TDCP 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。

8.5 $GPRMCイベント

- **イベント発生条件**

シリアルコンソールに接続した GPS レシーバから、NMEA-0183 センテンスを受信した時に発生します。
position_report または position_once コマンドでフラグ値 “1” が指定されていた場合

- **イベントデータフォーマット**

```
$$$,$GPRMC,<NMEA-0183_RMC_sentence>
```

<NMEA-0183_RMC_sentence> には GPS レシーバより取得した NMEA-0183 “RMC” センテンスの内容がそのまま入ります。NMEA-0183 “RMC” センテンス先頭の “$GPRMC” はそのまま TDCP イベントデータの第２カラム(イベント名)として使用します。その他の NMEA-0183 “RMC”センテンス中のデータフィールドもカンマ区切りの英数字で構成されていますので、そのままイベントデータに格納されて送信されます。

イベントデータ例：

```
"$$$,$GPRMC,084954,A,4254.1841,N,14132.6312,E,0.0,0.0,211009,9.3,W,A*0C"
“$$$,$GPRMC,,V,,,,,,,261009,9.3,W,N*2C”
```

- 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)

“uart0_baud”	コンソールポート(シリアルポート#0) のボーレートを設定する。
“uart0_echo”	コンソールポートの動作モードを変更する
“position_report”	GPS レシーバから NMEA-0183 センテンス受信毎に測位データを含んだイベントデータを送信する。
“position_once”	コマンド実行後にGPS レシーバから NMEA-0183 センテンス受信した時に、一回だけ測位データを含んだイベントデータを送信する。
“position_report_rate”	GPSレシーバから受信したNMEA-0183 センテンスを、このコマンドで指定した回数に

一回だけTDCPのイベント送信に使用する。

- **備考**

$GPRMC イベント発生を有効にするために、”position_report” コマンドで自動的に繰り返し送信する設定にするか、”position_once” コマンドで一回だけのイベントデータ送信に設定してください。

シリアルコンソールから 有効な NMEA-0183 センテンス($GPRMC,$GPGGA) を受信したときに、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。

イベント発生時には、直近で $GPRMC,$GPGGA 両方の NMEA-0183センテンスを受信していますので、このイベント発生で処理を行うイベントハンドラ中から “gps_rmc”“gps_gga” コマンドを実行することで、少なくともこのイベント発生時または、それよりも新しい測位データを確実に取得することができます。

注意

$GPRMC イベントデータには、<my_addr16>,<app_mode> データは含まれませんので注意して下さい。

8.6 GPSイベント

- **イベント発生条件**

シリアルコンソールに接続した GPS レシーバから、NMEA-0183 センテンスを受信した時に発生します。

position_report または position_once コマンドでフラグ値 “2” が指定されていた場合

- **イベントデータフォーマット**

TDCP

```
$$$,GPS,<my_addr16>,<app_mode>,<RMCStatus>,<RMCLatitude>,<RMCLatitudeCompass>,<RMCLongitude>,
<RMCLongitudeCompass>,<RMCSpeed>,<GGAQuality>,<GGAAntennaAltitude>,<GGAAntennaAltitudeUnit>
```

TDCPZB

```
$$$,GPS,<app_mode>,<RMCStatus>,<RMCLatitude>,<RMCLatitudeCompass>,<RMCLongitude>,
<RMCLongitudeCompass>,<RMCSpeed>,<GGAQuality>,<GGAAntennaAltitude>,<GGAAntennaAltitudeUnit>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は 現在の “app_mode” 値が設定されます。

<RMCStatus> NMEA-0183 “RMC” センテンス中の “Status” カラムの内容が入ります。

<RMCLatitude> NMEA-0183 “RMC” センテンス中の “Latitude” カラムの内容が入ります。

<RMCLatitudeCompass> NMEA-0183 “RMC” センテンス中の “N or S” カラムの内容が入ります。

<RMCLongitude> NMEA-0183 “RMC” センテンス中の “Longitude” カラムの内容が入ります。

<RMCLongitudeCompass> NMEA-0183 “RMC” センテンス中の “E or W” カラムの内容が入ります。

<RMCSpeed> NMEA-0183 “RMC” センテンス中の “Speed over ground, knots” カラムの内容が入ります。

<GGAQuality> NMEA-0183 “GGA” センテンス中の “GPS Quality indicator” カラムの内容が入ります。

<GGAAntennaAltitude> NMEA-0183 “GGA” センテンス中の “Antenna Altitude above/below mean-sea-level” カラムの内容が入ります。

<GGAAntennaAltitudeUnit> NMEA-0183 “GGA” センテンス中の “Units of antenna altitude, meters” カラムの内容が入ります。

イベントデータ例：

```
“$$$,GPS,0A01,9,A,4254.1640,N,14135.6099,E,000.0,1,37.3,M”
```

- 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)

“uart0_baud”	コンソールポート(シリアルポート#0) のボーレートを設定する。
“uart0_echo”	コンソールポートの動作モードを変更する
“position_report”	GPS レシーバから NMEA-0183 センテンス受信毎に測位データを含んだイベントデータを送信する。
“position_once”	コマンド実行後にGPS レシーバから NMEA-0183 センテンス受信した時に、一回だけ測位データを含んだイベントデータを送信する。
“position_report_rate”	GPSレシーバから受信したNMEA-0183 センテンスを、このコマンドで指定した回数に一回だけTDCPのイベント送信に使用する。

- **備考**

“GPS” イベント発生を有効にするために、”position_report” コマンドで自動的に繰り返し送信する設定にするか、”position_once” コマンドで一回だけのイベントデータ送信に設定してください。

シリアルコンソールから 有効な NMEA-0183 センテンス(\$GPRMC,\$GPGGA) を受信したときに、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。

イベント発生時には、直近で \$GPRMC,\$GPGGA 両方の NMEA-0183センテンスを受信していますので、このイベント発生で処理を行うイベントハンドラ中から “gps_rmc”“gps_gga” コマンドを実行することで、少なくともこのイベント発生時または、それよりも新しい測位データを確実に取得することができます。

8.7 LIVEイベント

- **イベント発生条件**

“heartbeat_rate” コマンドを使用して、LIVEイベント発生間隔(秒)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**

全ての “app_mode” で使用可能なイベントです。イベントデータフォーマットは共通で下記になります。

TDCP

```
$$$,LIVE,<my_addr16>,<app_mode>
```

TDCPZB

$$$,LIVE,<app_mode>

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。
<app_mode> は 現在の “app_mode” 値が設定されます。

- 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)

“heartbeat_rate” イベント発生間隔を設定する。

- **備考**

デバイス自身が正常に動作しているかまたは、デバイスとサーバー間の通信が可能かどうかを定期的に監視するために使用します。“heartbeat_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。

“heartbeat_rate” コマンドで指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。TDCP ではある程度(1/1000 秒)までの秒補正を “second_adjust” コマンドで設定することが可能ですので、詳しくは ”コマンドリファレンス” の章の “second_adjust” コマンドの項を参照してください。ただしこの場合でも TDCP で実行中のタスクの状況によっては、イベント送信が遅れる(10ms.. 数秒程度)場合があります。

8.8 ADVAL_UPDATEイベント

- **イベント発生条件**

A/D 変換値が “ad_margin” コマンドで設定した値以上に変化した場合に発生します。

“signal_event” で設定したシグナル更新の対象イベントです。

- **イベントデータフォーマット**

イベントデータフォーマットはイベント発生時の “app_mode” によって格納されるサンプリングデータが異なります。代表的なサンプリングデータフォーマットを下記に列挙します。ここで記載した以外のデータフォーマットもありますので、”TDCP動作モード” の章と、その中の “app_mode” 毎の サンプリングイベントデータの項目を参照してください。

TDCP

$$$,ADVAL_UPDATE,<my_addr16>,<app_mode>,<ad_update_bits>,<adc0>,<adc1>,<adc2>,<adc3>
$$$,ADVAL_UPDATE,<my_addr16>,<app_mode>,<ad_update_bits>,<adc0>,<adc1>,<adc2>,<adc3>,<adc4>,<adc5>,<adc6>,<adc7>

TDCPZB

$$$,ADVAL_UPDATE,<app_mode>,<ad_update_bits>,<adc0>,<adc1>,<adc2>,<adc3>
$$$,ADVAL_UPDATE,<app_mode>,<ad_update_bits>,<adc0>,<adc1>,<adc2>,<adc3>,<adc4>,<adc5>,<adc6>,<adc7>

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は 現在の “app_mode” 値が設定されます。

<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 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。

例えば、<ad_update_bits> が 03 の場合には adc0, adc1 の両方が、前回 adc0, adc1 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_margin,0,xx”,”ad_margin,1,xx” で設定した値以上に変化したことを示します。

<adc0>.. <adc7> 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。

- 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” 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

“ad_margin_reset” A/D 変化値を比較するときの前回のイベント送信時の比較値をリセットする

- **備考**

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。

8.9 I2C_SLAVE_EVENTイベント

- **イベント発生条件**

TDCP I2C スレーブ機能で、I2C マスターデバイスからイベント送信コマンド(0x11) を受信した時に発生します。

- **イベントデータフォーマット**

TDCP

```
$$$,I2C_SLAVE_EVENT,<my_addr16>,<app_mode>,<event_data>
```

TDCPZB

```
$$$,I2C_SLAVE_EVENT,<app_mode>,<event_data>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は 現在の “app_mode” 値が設定されます。

<event_data> には、I2C マスターデバイスから I2C スレーブコマンド(0x11) で指定されたイベントデータが入ります。16進数表記で１バイト毎に 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、文字列形式で入ります。

例えば、イベントデータが 3 バイトで 0x01,0x10,0xFF の場合には、<event_data> には “0110FF” 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)

“i2c_use” I2C 機能を使用するかどうかを指定する。

“i2c_slave_addr” TDCPの I2C スレーブ機能を有効にして、そのときのスレーブアドレスを指定する。

- **備考**

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。

8.10 I2C_SLAVE_OVERFLOWイベント

- **イベント発生条件**

TDCP I2C スレーブ機能で、I2C マスターデバイスからイベント送信コマンド(0x11) を受信して、TDCP 内部のイベントキューがオーバーフローした時に発生します。

- **イベントデータフォーマット**

TDCP

```
$$$,I2C_SLAVE_OVERFLOW,<my_addr16>,<app_mode>
```

TDCPZB

```
$$$,I2C_SLAVE_OVERFLOW,<app_mode>
```

<my_addr16> には TDCP に接続した XBee デバイスの16ビットアドレスが16進数表記で設定されます。

<app_mode> は 現在の “app_mode” 値が設定されます

- 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)

“i2c_use” I2C 機能を使用するかどうかを指定する。

“i2c_slave_addr” TDCPの I2C スレーブ機能を有効にして、そのときのスレーブアドレスを指定する。

- **備考**

TDCP スレーブ機能では、I2C マスターから送信されたイベント送信コマンドは内部でキューイングされて、TDCPのタスク処理中にイベントキューから取り出されて順次送信されます。イベントキューは最大 5イベント分のサイズが確保されています。これ以上のイベント送信が連続して送信コマンドで指定されるとオーバーフローが発生して、DeviceServer には “I2C_SLAVE_OVERFLOW” イベントが送信されます。オーバーフロー発生直後から、次にイベントキューに空きができるまでの間のイベント送信コマンドは全て破棄されます。

オーバーフロー発生直後から、“I2C_SLAVE_OVERFLOW” 送信までの間に複数回 I2C イベント送信コマンドを受信した場合でも、“I2C_SLAVE_OVERFLOW” イベントデータは１回だけ送信されます。

“I2C_SLAVE_OVERFLOW” イベントが送信された後に、まだイベントキューに空きがない状態で I2C イベント送信コマンドを受信すると、再び “I2C_SLAVE_OVERFLOW”イベントが発生してイベントデータが送信されます。

9 シグナル出力機能

TDCP 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。app_mode によって決められたシグナルデータ出力ポートにこれらの出力を行っています。

TDCP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	緑LED 点滅	黄LED 点滅	赤LED 点滅	BEEP1	緑LED 点灯	黄LED 点灯	赤LED 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。
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値に従って、TDCP の PORTA に下記の出力を行います。

PORTA	Bit 7	シグナル ブザー出力
	Bit 6	シグナル LED 緑 出力
	Bit 5	シグナル LED 黄 出力
	Bit 4	シグナル LED 赤 出力

緑LED, 黄色LED, 赤LEDのシグナルは PORTA の bit4-6(シグナル LED) 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 bit7(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 TDCP 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TDCP 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 GPSレシーバ接続

TDCP は、コンソールポートにGPS レシーバを RS-232C で接続して測位情報を取得することができます。

全ての app_mode で GPS 接続機能が使用可能です。GPS 接続には、コンソールポートを使用しますので、接続中のTDCPコマンドを実行は全て XBee 経由でリモートから行います。

10.1 GPS レシーバ接続時の通信条件

シリアルコンソールポートに接続するGPS レシーバ通信条件	
通信フォーマット	NMEA-0183
TDCP で使用するNMEA-0183センテンス(*1)	$GPRMC $GPGGA
通信インターフェース	RS-232
ボーレート	9600 または 4800
ビット長	8
ストップビット	1
パリティ	無し
フロー制御	無し
測位情報更新スピード(回/秒) (*2)	1 以下

(*1) 接続するGPS レシーバの設定で、これら以外のセンテンスが送信された場合は、TDCP はそれらの内容は無視します。これらの TDCP で不用なセンテンスをGPS レシーバ側で抑止することが可能な場合は、できるだけ送信しないようにしてください。

(*2) GPS レシーバの 測位情報更新スピードが 1 Hz よりも大きい場合には、必ず 1 Hz 以下の更新スピードになる

ように GPS レシーバの設定を変更して下さい。

10.2 TDCP コンフィギュレーション

TDCPの全ての app_mode で、GPS レシーバを接続することができます。

- ケーブル接続

TDCP に GPS レシーバを接続する場合には、シリアルコンソールポートに RS-232 インターフェースで接続します。マイクロプロセッサのシリアルポート#0 の RX GND を GPSレシーバ側の TX,GND に接続してください。TDCP からGPS レシーバにデータ送信することはありませんので、シリアルポート#0 の TX 側の接続は不要です。(接続しても構いません) TDCP では H/W フロー制御を使用しませんので、必要に応じてGPSレシーバ側の RTS,CTS の処理を行ってください。

- ボーレート設定

シリアルポート#0 のボーレートを GPS レシーバに合わせて設定します。シリアルコンソールポートに接続した端末からも操作可能ですが、コマンド実行直後にボーレートが変化しますので端末側のボーレートの変更が必要になります。以降の操作は、リモートから操作することをお勧めします。

コマンド実行例： “uart0_baud,9600”

- エコーバック抑止

TDCP のシリアルポート#0 では、シリアルポート経由でコマンド入力を容易にするために、ローカルエコーや端末編集文字の処理を行っています。GPS レシーバ接続時にはこれらを停止させます。

コマンド実行例： “uart0_echo,0”

- コンフィギュレーションの保存

設定したコンフィギュレーションを EEPROM に保存します。

コマンド実行例： “config_save”

10.3 測位データの取得

GPS レシーバから、NMEA-0183 センテンスが送信されると TDCP では自動的に最新の測位情報を内部に保存・更新しています。リモートから TDCP コマンドを実行することで、何時でもこれらの測位情報を利用可能です。

- 測位データの手動取得

“gps_rmc”,”gps_gga” コマンドを使用すると、TDCP 内に保存されている最新の即位情報を常にリモートから取得することができます。GPS レシーバが停止していた場合でもこのコマンドは最後にTDCP が受信した測位データを取得します。

- 測位データの最新情報を取得

“position_once,1” または “position_once,2” コマンドを実行することで、コマンド実行以降に GPS レシーバから受信した最新の測位情報をイベントデータ(*1)で送信できます。

- 測位データの自動繰り返し受信

“position_report,1” または “position_report,2”コマンドを実行することで、GPS レシーバから測位情報を受信する毎に、その測位情報をイベントデータ(*1)で繰り返し送信することができます。

(*1) GPS の測位情報のイベントについての詳しい内容は、”TDCPイベントリファレンス”の章内の “$GPRMCイベント” と “GPSイベント” の項目を参照して下さい。GPS レシーバが停止していた場合には、再びレシーバから測位情報(NMEA-0183 センテンス)を取得するまでイベントは発生しません。

11 I2C接続機能

TDCP は I2C バスで接続した複数のデバイスと通信することができます。”i2c_use” コマンドでTDCP のI2C 機能を有効または無効に設定できます。（初期値は無効に設定されています）

TDCP の I2C 機能は、マイクロプロセッサ内部の TWI モジュール機能を使用して、データ通信速度は 約 100Kbps (Standard Mode) で固定です。

TDCP の I2C 機能を有効にした場合には、PORTC のビット0とビット1のピンが、それぞれ I2C コントロール用の ″SCL″ と ″SDA″ に割り当てられます。また現在の app_mode でそのピンに割り当てられている機能は使用できなくなります。

”i2c_use” コマンドの設定値でプロセッサ内部のプルアップを有効に設定した場合には、″SCL″ と ″SDA″ ピン内部のプルアップが有効になります。この時、PORTC のビット0とビット1 に設定された ″pullup″ コマンドの設定値は無視されます。

”i2c_use” コマンドの設定値でプロセッサ内部のプルアップを無効に設定した場合には、″SCL″ と ″SDA″ ピン内部のプルアップが無効になります。この時、PORTC のビット0とビット1 に設定された ″pullup″ コマンドの設定値は無視されます。外部の回路で適切なプルアップ抵抗を接続して下さい。

TDCP の I2C 機能はデフォルトでマスターデバイスとして動作します。TDCP をスレーブデバイスとしても機能させる場合には “i2c_slave_addr” コマンドで、予めTDCP 自身のスレーブアドレスを設定してください。

“i2c_use” コマンドや “i2c_slave_addr” コマンドの設定値を変更した場合には、リセット後に有効になります。そのため、″i2c_use″、“i2c_slave_addr” コマンドに続けて″config_save″ コマンドを実行してEEPROM に設定を保存した後、″reset″ コマンドを実行してください。

コマンドの機能詳細は、それぞれのコマンドリファレンスを参照して下さい。

11.1 I2C マスター機能

“i2c_use” コマンドで、I2C 機能を有効にした場合に TDCP は I2C マスターデバイスとして動作できます。

TDCP を I2C マスターデバイスとして動作させるためのコマンドは “i2c_read” コマンドと “i2c_write” コマンドです。それぞれ、指定した I2C スレーブデバイスに対してデータの受信や送信を行います。

コマンドの機能詳細は、それぞれのコマンドリファレンスを参照して下さい。

11.2 I2C スレーブ機能

“i2c_slave_addr” コマンドで、TDCP 自身に I2C スレーブアドレスを設定して、且つ “i2c_use” コマンドでI2C 機能を有効にした場合に TDCP は I2C スレーブデバイスとしても動作します。

TDCP の I2C スレーブデバイス機能は、I2C マスターデバイスからのコマンドデータを受信してコマンド処理を行った後、コマンドの種類によってはリターン値をマスターデバイスに送信します。

TDCP の I2C スレーブデバイスは必ず最初にマスターデバイスから1 byte のコマンドデータと必要に応じてコマンドパラメータを受信します。コマンドデータとパラメータを合わせて最大 32 バイト分のデータをTDCP の I2Cスレーブデバイスに送信することができます。

```
<command>,<param#0>,<param#1> ... ,<param#30>
```

送信されたコマンドデータごとに予め決められた処理を TDCP 内で実行して、必要に応じてI2C マスターデバイスに最大 32 バイト分のリターンデータを返します。

```
<return#0>,< return#1>,< return#2> ... < return#31>,
```

コマンドデータ送信後にリターンデータをマスターデバイスが受信する場合には、I2C バスを “Master Transmitter” から “Master Receiver” に切り替えて操作して下さい。このとき、先に送信済みのコマンドデータの実行結果をリターン値として取得することができます。

I2C バス上にある１つのTDCP スレーブデバイスに対して、複数のマスターデバイスがアクセスする場合には、それぞれのマスターデバイスは、”Repeated start mode” を使用して上記のモード切り替え操作を行って、コマンド送信からリターン値取得まで、同一のI2C マスタデバイスが I2C バスを占有するようにして下さい。

また、TDCP には I2C スレーブ機能としてユーザーが自由に読み書き可能な I2Cワークエリアがあります。ワークエリアは 256 バイトのサイズで、TDCP のI2C スレーブコマンドを使用して、I2C バスに接続された I2C マスターデバイスから操作できます。

I2C ワークエリアは、TDCP コマンド”i2c_work_read”, “i2c_work_write” を使用しても読み書きができますので、I2Cデバイスと他のシステム間のデータ通信にも利用できます。

以下にTDCP のスレーブ機能で提供するコマンドを説明します。

11.2.1 TDCP 動作モード取得コマンド(0x01)

TDCP の現在の動作モード(“app_mode” コマンドで設定した値) を取得します。

(write) 0x01	(read) <app_mode> 1 byte

11.2.2 イベント送信コマンド(0x11)

コマンドパラメータで指定したバイト列のデータを "I2C_SLAVE_EVENT" イベントパケットとしてマスター(TDCP)からDeviceServer に送信させる。

(write) 0x11	(write) <event_data> max 31 bytes

<event_data> の内容は、31 バイト以内のバイナリデータで、アプリケーション毎に自由に設定できます。
ここで設定したバイナリデータは、I2C スレーブデバイス(TDCP デバイス) を経由して、DeviceServerにイベントデータとして送信されます。

イベント送信コマンドは内部でキューイングされて、TDCPのタスク処理中にイベントキューから取り出されて順次送信されます。イベントキューは最大 5イベント分のサイズが確保されています。これ以上のイベント送信が連続して送信コマンドで指定されるとオーバーフローが発生して、DeviceServer には "I2C_SLAVE_OVERFLOW" イベントが送信されます。オーバーフロー発生直後から、次にイベントキューに空きができるまでの間のイベント送信コマンドは全て破棄されます。

event_data[0] から event_data[30] の間に任意のイベントデータを格納して下さい。

11.2.3 ワークエリアデータ書き込みコマンド(0x21)

コマンドパラメータで指定したバイト列のデータを I2C ワークエリアに書き込む。

(write) 0x21	(write) <work_area_address> 1 byte	(write) <work_data> max 30 bytes

ワークエリアは 256 バイト分操作可能で、<work_area_address> で指定したアドレスから連続して<work_data> で指定したデータを書き込みます。データ書き込み中に、ワークエリアの終端に達した場合には 0 番地から引き続き書き込まれます。

11.2.4 ワークエリアデータ読み出しコマンド(0x22)

コマンドパラメータで指定したワークエリアアドレスから I2C ワークエリアデータを読み出す。

(write) 0x22	(write) <work_area_address> 1 byte	(read) <work_data> max 32 bytes

ワークエリアは 256 バイト分操作可能で、<work_area_address> で指定したアドレスから連続してデータを読み出します。データ読み出し中に、ワークエリアの終端に達した場合には 0 番地から引き続き読み出されます。

12 SPI接続機能

TDCP は SPI バスで接続したデバイスと通信することができます。”i2c_config” コマンドでTDCP のSPI 機能を有効または無効に設定できます。（初期値は無効に設定されています）
TDCP のSPI 機能はマスターデバイスとしてのみ使用可能で、TDCP を SPI スレーブデバイスとして動作させることはできません。

TDCP の SPI 機能は、 マイクロプロセッサ内部の SPIモジュール機能を使用して、データ通信速度やSPI 動作モード、送信オーダなどは “spi_config” コマンドで設定できます。

TDCP の SPI マスターデバイスからスレーブデバイスのデータの読み書きには “spi_read”,”spi_write” コマンドを使用します。

SPI 機能を有効にすると、PORTB bit4 は “Slave Select”, PORTB bit5 は ”MOSI”、 PORTB bit6 は ”MISO”、 PORTB bit7 は ”SCK” 信号に使用されます。 また、”spi_write” コマンドでラッチパルスを出力した場合には、PORTD bit6 にパルス信号が出力されます。

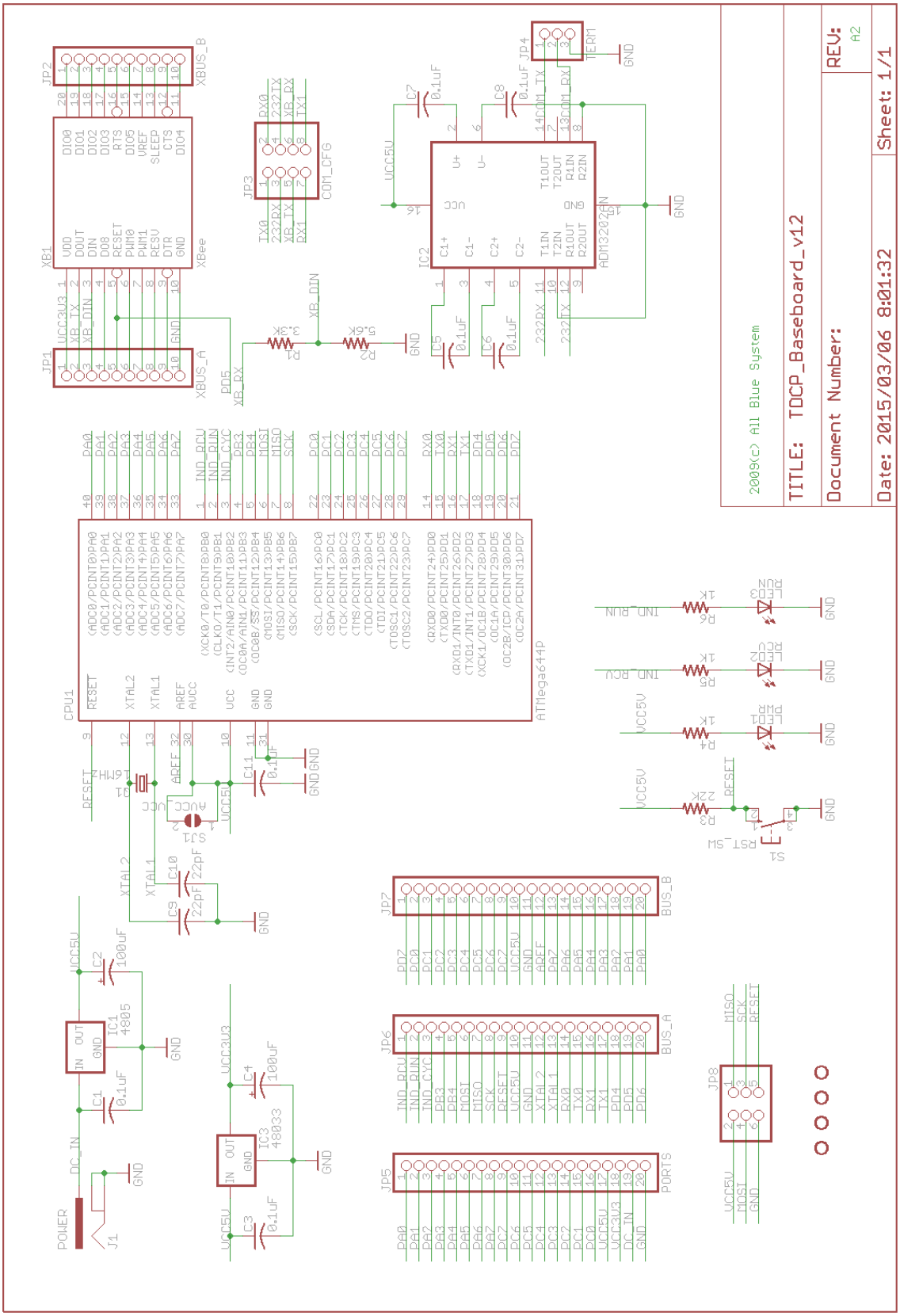
コマンドの機能詳細は、それぞれのコマンドリファレンスを参照して下さい。

13 TDCP リモートコントローラボード作成例

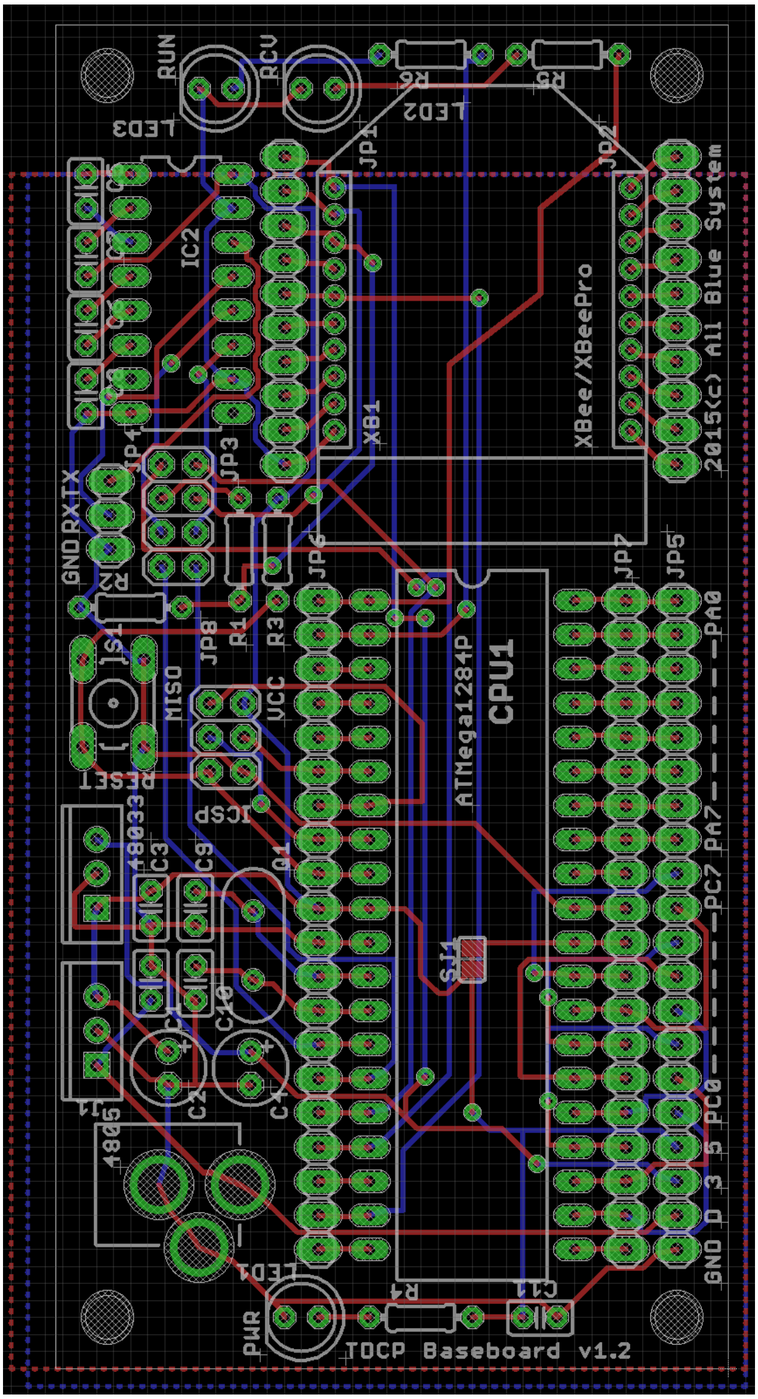
TDCP プログラムを動作させるための CPU ボードの作成例を説明します。ここで説明するボードの回路は基本的な動作確認を目的にしていますので、実際のアプリケーションで TDCP を使用する場合には動作環境に合わせて回路を設計して下さい。

13.1 CPU ボード回路図

（回路図中の CPU は ATMega644P となっていますが、ATMega1284P で実装しています）



13.2 基板パターン図


TDCP Baseboard v1.2
2015(c) All Blue System
XBee/XBeePro
CPU1
ATMega1284P
RUN
RCV
PWR
RESET
ICSP
MISO
VCC
GND RX TX

13.3 部品リスト

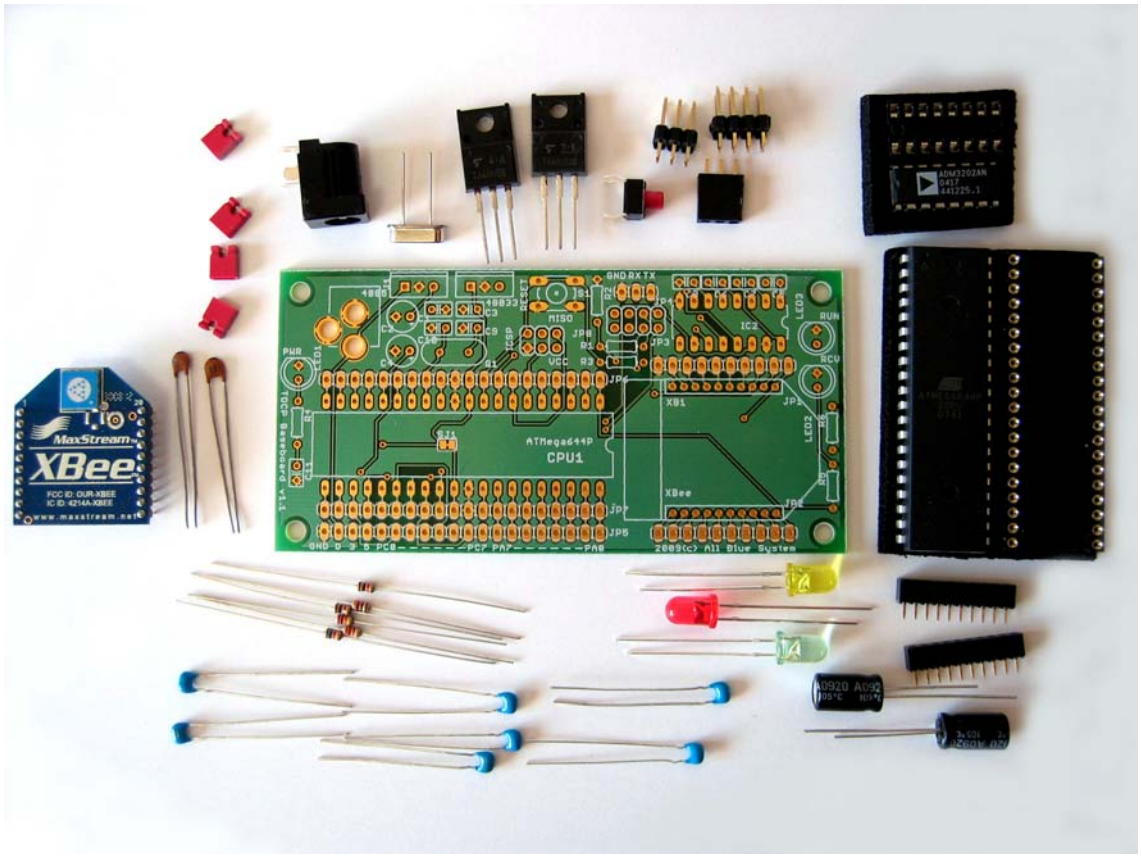
[] 内は 3.3V 8MHz 動作の場合

部品番号(回路図)	値
C1	0.1uF
C2	100uF
C3	0.1uF [実装しない]
C4	100uF [実装しない]
C5	0.1uF
C6	0.1uF
C7	0.1uF
C8	0.1uF
C9	22pF
C10	22pF
C11	0.1uF
CPU1	Atmega1284P
IC1	4805 (5V レギュレータ) [48033 (3.3V レギュレータ)]
IC2	ADM3202AN (RS232 レベルコンバータ)
IC3	48033 (3.3V レギュレータ) [IN-OUT 端子間を接続]
J1	電源ジャック
JP1	10ピンソケット(2mm幅)
JP2	10ピンソケット(2mm幅)
JP3	2X4 ピンヘッダ
JP4	3 ピンソケット
JP8	2X3 ピンヘッダ
LED1	赤 LED (POWER)
LED2	黄 LED (RECEIVE PACKET)
LED3	緑 LED (RUN TASK)
Q1	クリスタル 16MHz [クリスタル 8MHz]
R1	3.3KΩ [100Ω]
R2	5.6KΩ [実装しない]

R3	22KΩ
R4	1KΩ
R5	1KΩ
R6	1KΩ
S1	タクトスイッチ (RESET)
XB1	XBee 802.15.4 OEM RF モジュール または XBee-ZB モジュール
その他	IC ソケット(16pin) IC ソケット(40pin) ジャンパーピン x4

CPU ボードに使用する XBee デバイスについて、API モードとアドレス等が既に設定済みで、TDCP のシリアルコンソール経由で ATコマンドを使用しない場合や、GPS モジュールを接続しない場合には ADM3202ANデバイスとその周辺コンデンサは省略できます。

(TDCP リモートコントローラボード 部品例)



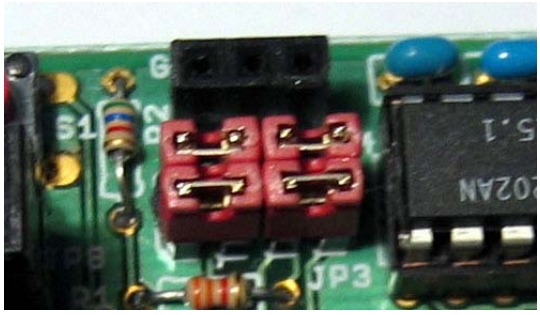
13.4 シリアルポート設定用ジャンパー(JP3)

シリアルポート設定用 JP3 (2x4 ピンヘッダ) にジャンパーピンを設定して、XBee と Atmega1284P プロセッサのシリアルポート、コンソールポート JP4(1x3ピンヘッダ)のコンフィギュレーションを変更することができます。

ジャンパーピンの接続を変更する場合は、必ず電源を OFF にして行ってください。

13.4.1 通常運用時ジャンパー設定

通常運用時は 4 つのジャンパーピンで、下記の様に接続します。

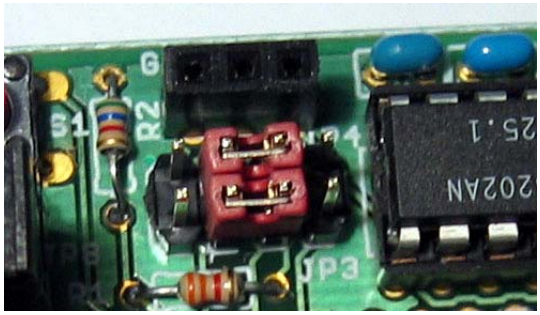


この場合には、下記の接続と信号レベル変換が行われます。

接続対象デバイス・ポート	
コンソールポート JP4 (RS232Cレベル)	Atmega1284P RX0/TX0
XBee シリアルポート (3.3V レベル)	Atmega1284P RX1/TX1

13.4.2 XBee ATコマンド操作用ジャンパー設定

XBee モジュールの API モードの初期設定を行う場合など、モジュールを直接シリアルコンソールに接続して ATコマンド操作を行いたい場合は 2 つのジャンパーピンで下記の様に接続します。



この場合には、下記の接続と信号レベル変換が行われます。

接続対象デバイス・ポート	
コンソールポート JP4 (RS232Cレベル)	XBee シリアルポート

(Atmega1284P のシリアルポート RX0/TX0, RX1/TX1 はどこにも接続されません)

13.5 TDCP XBee 802.15.4 RF Module Series1 初期設定

XBee モジュールの API モードとID,アドレスを設定します。API モードの値は必ず “1” を指定します。

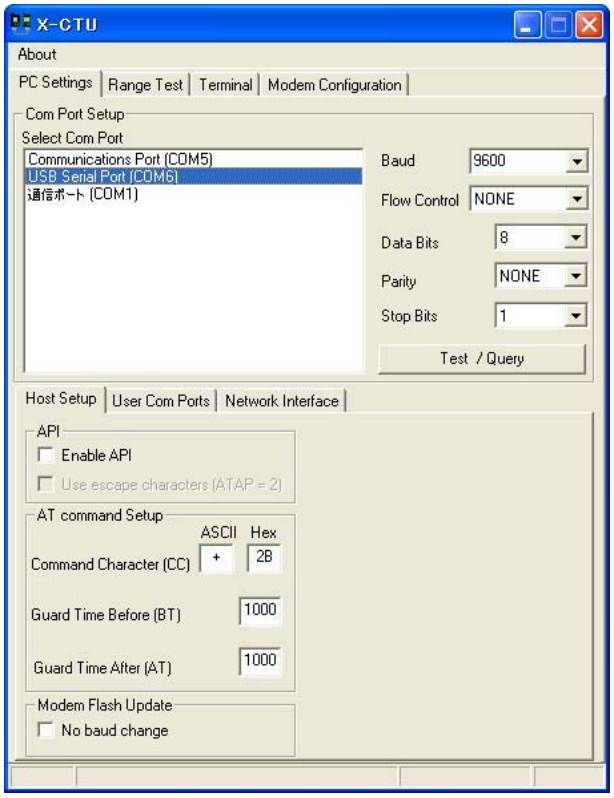
ID とアドレス(16ビットアドレス) をここでは 0xAB90, 0x0001 にそれぞれ設定する例で説明します。

まず、XBee モジュールをTDCP リモートコントローラボードに接続します。シリアルポート設定用 JP3を前述の

“XBee ATコマンド操作用ジャンパー設定” の項目に従ってジャンパー設定します。

TDCP リモートコントローラボードのシリアルコンソールポートを PC の COM ポートに接続します。
デバイス初期設定のコマンドをXBee に送信するために、Digi international Inc. 社製の X-CTU プログラム、または汎用のターミナルエミュレータプログラム等を使用します。COM ポートのボーレートは初期設定の 9600 bps にして下さい。（ターミナルエミュレータを使用する場合は、ローカルエコー ON, 受信時の改行 CR + LF にするとコマンド実行の結果が見やすくなります）

X-CTU プログラムを起動して、COM ポートを選択します。ここでは、USB Serial Port(COM6) を選択しています。



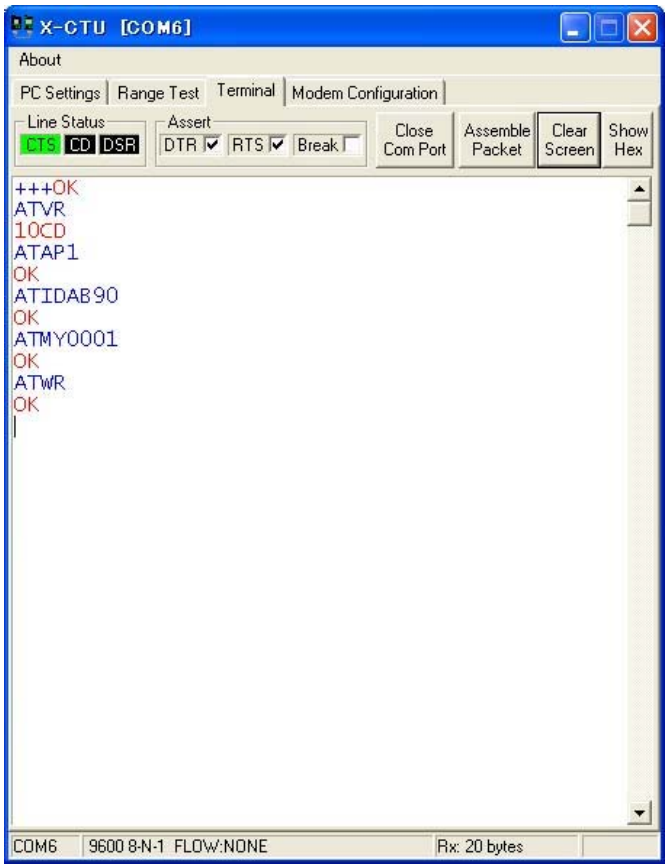
Terminal タブを選択してターミナル画面を表示します。キーボードから、”+++” を入力して、コマンドモードに入ります。コマンドモードに入ると “OK” が表示されますので、続けて以下のコマンド文字列を入力してください。
コマンド入力の時間がかかりすぎると、自動的にコマンドモードから抜けてしまいますので、その場合は、”+++” を入力して最初からコマンドを入力し直して下さい。

```
ATVR
ATAP1
ATIDAB90
ATMY0001
ATWR
```

最初に ATVR でファームウエアバージョンを表示しています。”10CD” 以降になっていることを確認してください。

ATAP1 は、API モードを “1” に設定しています。ATIDAB90 は PAN_ID を 0xAB90 に設定しています。もし別の PAN_ID を使用する場合は適宜変更してください。次に、ATMY0001 で、デバイスの16 bit Source Address を “0x0001” に設定しています。この部分は、デバイスごとにユニークな値になるように変更して下さい。

最後に、ATWR で、設定値を不揮発メモリに書き込みます。入力時の画面表示は以下の様になります。



X-CTU プログラムを終了します。

その後、シリアルポート設定用 JP3を前述の “通常運用時ジャンパー設定” の項目に従ってジャンパー設定します。

注意

シリアルポート設定用 ジャンパーピンの接続を変更する場合は、必ず TDCP リモートコントローラボードの電源を OFF にした状態で行ってください。

13.6 TDCPZB XBee-ZB ZigBee対応 Module Series2 初期設定

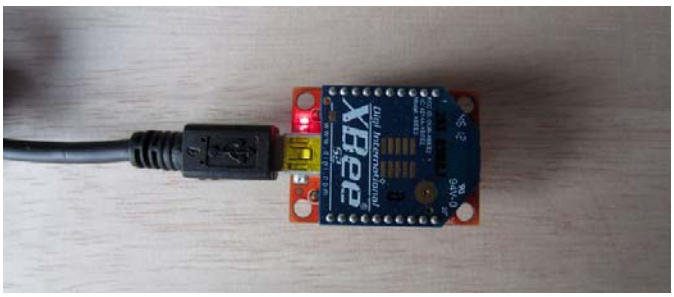
XBee-ZB デバイスの初期設定する項目は以下の内容です。

XBee-ZB には ZigBee デバイスタイプに対応して、Coordinator, Router, End device の３種類のファームウエアがあります。TDCPZB で使用する XBee-ZB は Router タイプを選択して書き込みます。

XBee-ZB デバイス デフォルト値から変更が必要な設定値

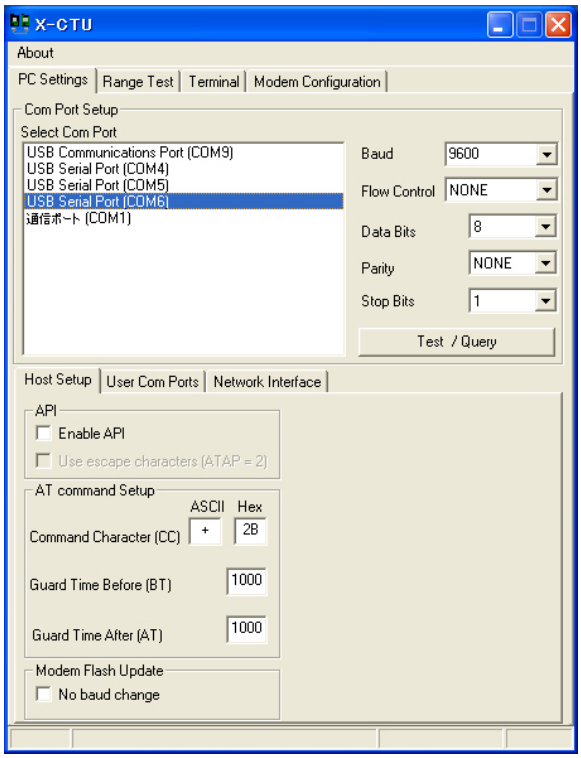
ファームウエアの書き換え	ZIGBEE ROUTER API を使用してください。
API モード	1 ZIGBEE ROUTER API タイプのファームエア書き換え直後は 1 に設定されていますが、念のため確認してください
PAN(Personal Area Network) ID	任意の値 (default は0) デフォルトの値のままだと、予期しないデバイスからのフレームを受信したり、間違ってデバイスを操作する恐れがありますので、適当な任意の値を設定するようにしてください。このマニュアルでは 0xAB9000 を使用しています。
Node Identifier	同一PAN ID 内でユニークな文字列 (default は "") この値は、後から ZB管理プログラムで設定・変更することも可能ですが、デバイス一覧から選択したデバイスがどのデバイスであるかを見分けることが容易になるように便宜的にここで設定します。デバイスを選択するときにわかり易いように全てのデバイスで異なった名前を設定してください。例えば、"NODE01", "NODE02" 等。

初期設定のコマンドを XBee-ZB に送信するために、XBee-ZB デバイスを PC のCOM ポートに接続して Digi international Inc. 社製の X-CTU プログラムを使用します。XBee-ZB と COM ポートのボーレートは初期値の 9600bps を使用します。Sparkfun Electoronics 社製の XBee Explorer USB などを使用して、仮想 USB ポート経由で接続して設定してください。(これ以外の方法で COM ポート接続する場合も手順は同じです)このマニュアルでは既に、XBee Explorer USBのドライバ等がインストール済みで仮想 USB ポートが作成されているものとします。(例では COM6)



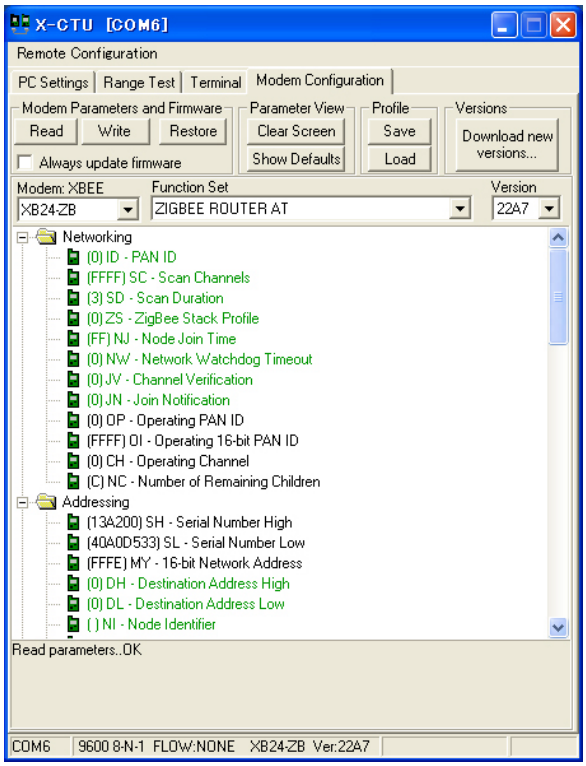
XBee-ZB デバイスを搭載した、XBee Explorer をUSB ケーブルで PC に接続した状態

X-CTU プログラムを起動して、COM ポートを選択します。ここでは、USB Serial Port(COM6) を選択しています。"Host Setup"タブ中の "API 設定" のチェックボックスは、最初にデバイスを接続するときにはチェックを外してください。XBee-ZB 購入直後には ZIGBEE ROUTER AT タイプのファームウエアがロードされています。後で ZIGBEE ROUTER API タイプのファームウエアを書き込んだ後には、 "API 設定" のチェックを入れて設定値の読み込みと書き込み操作をします。



(X-CTU プログラムで XBee-ZB デバイスへの接続条件を設定している画面)

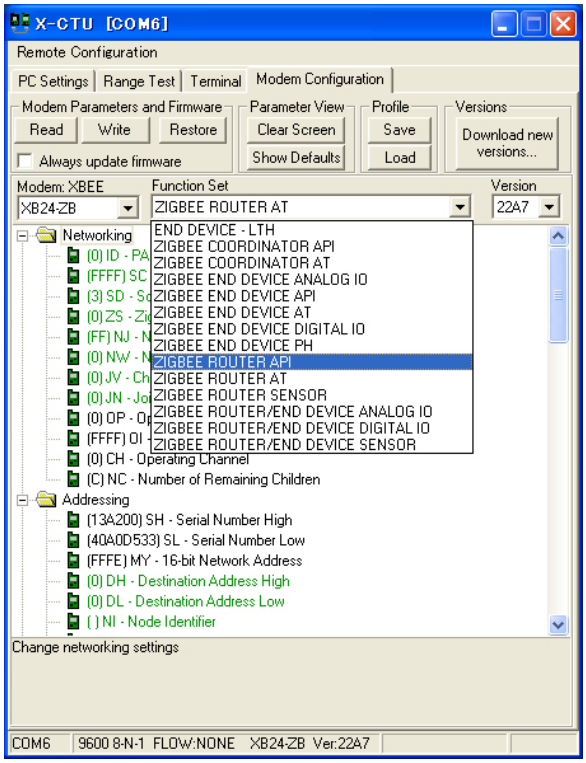
"Modem Configuration" タブを選択して、現在の XBee-ZB デバイスの設定をリードします。これは COM ポートの通信が正常に行えるかの確認も兼ねています。



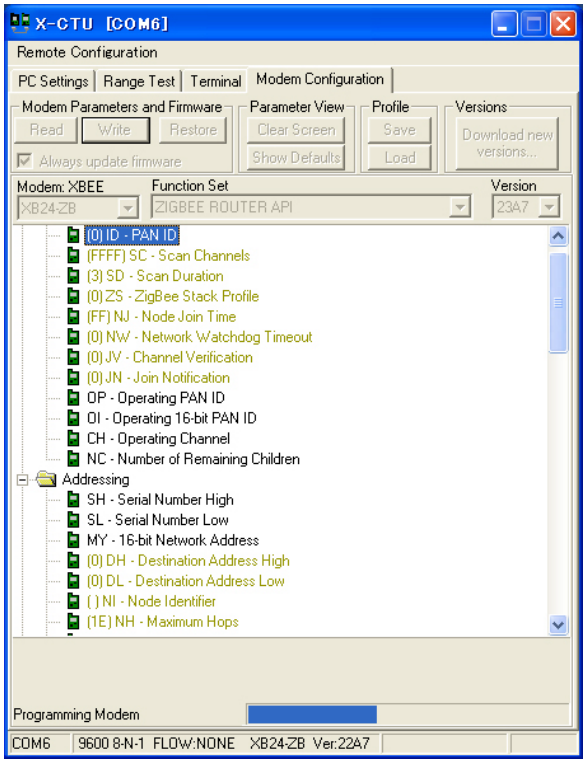
(XBee-ZB デバイスの工場出荷時のデフォルト設定を読み込んだ様子)

XBee-ZB デバイスの初期設定は AT モードのファームエアがロードされていますのでこれを、書き換えます。

"Function Set" プルダウンメニューから ZIGBEE ROUTER API を選択します。

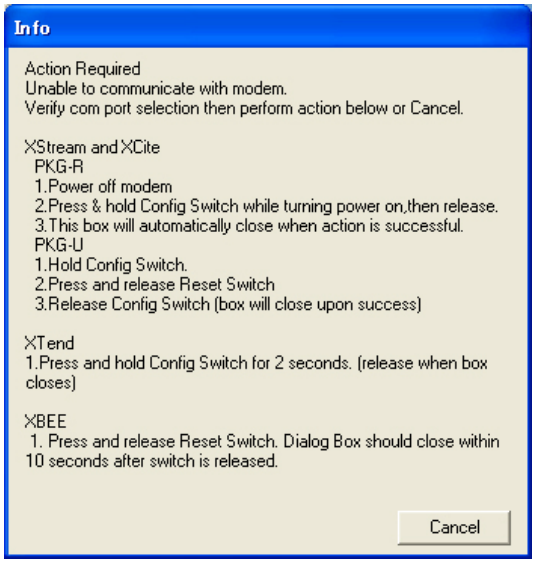


"Always update firmware" チェックボックスにチェックをつけて、"Write" ボタンを押してファームウエアの書き込みをします。

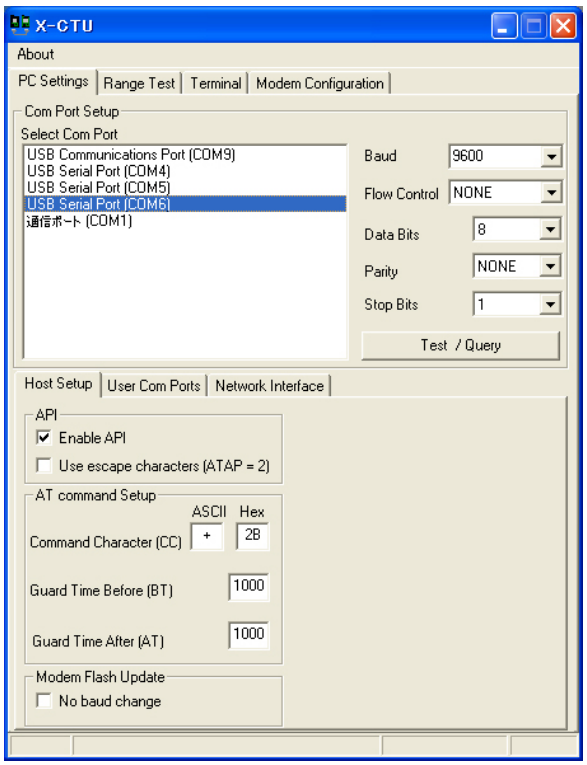


（ファームエア書き込み中の X-CTU の画面）

ここでファームエア書き込み終了後に、下記の様な画面が表示される場合があります。

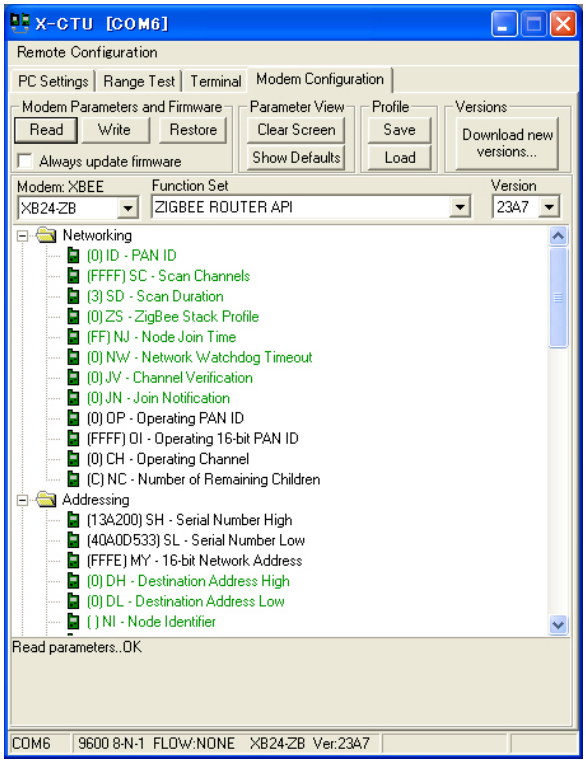


これは、ファームエアが API モード用のものに変わったために、最初にX-CTU プログラムの通信条件設定画面で "Enable API" のチェックを外していた状態と矛盾するようになったのが原因ですので問題ありません。ここでは "Cancel" ボタンを押してダイアログを消してください。ここで 一旦 X-CTU プログラムを終了して、もう一度同じ X-CTU プログラムを起動してください。

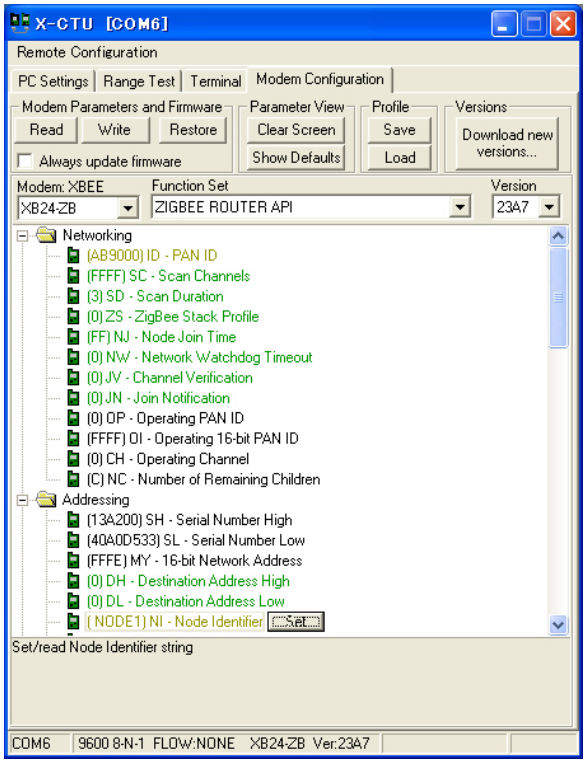


XBee-ZB が接続されたCOM ポートを選択して、今度は "Enable API" にチェックを付けます。

"Modem Configuration" タブを選択して、"Read" ボタンを押してファームウエア書き込み後の XBee-ZB デバイスの設定値をリードします。



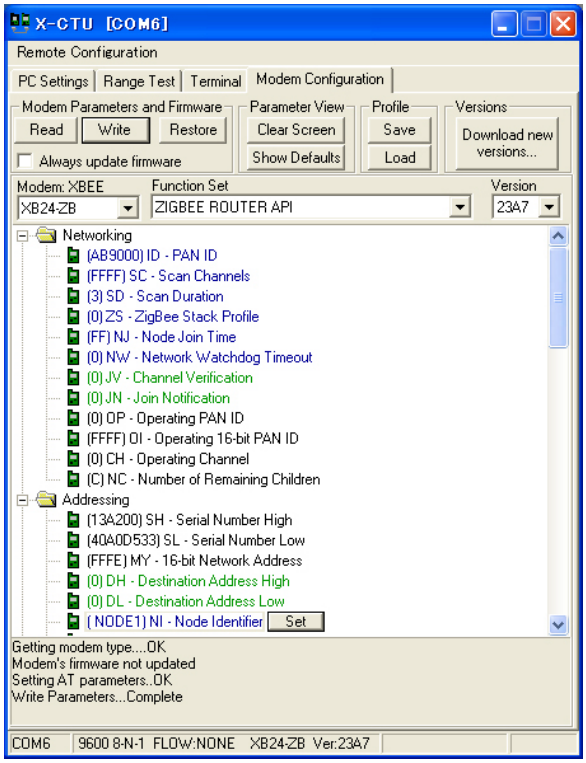
ここから、XBee-ZB のパラメータを設定します。最初に "PAN ID" 項目を選択して、任意の 最大64bit幅の 16進数値を設定します。ここでは "0xAB9000" を設定しています。PAN ID は全ての XBee-ZB デバイスで同一の値を設定してください。次に "Node Identifier" に任意のアルファベット文字列を設定します。ここでは "NODE3" を設定します。XBee-ZB デバイス毎に異なった名前をつけてください。



(PAN ID と Node Identifier を設定している様子、画面では "NODE1" が表示されていますが、このマニュアルでは "NODE3" に設定します)

また、API モード値がデフォルト値の 1 になっていることも確認してください。設定項目をスクロールして設定値を表示できます。

設定値を XBee-ZB デバイスに書き込むために、"Write" ボタンを押します。



(設定値を XBee-ZB デバイスに書き込んでいる様子)

これでリモート側の XBee-ZB デバイスの初期設定が終了しました。

X-CTU プログラムを終了させてから XBee Explorer USB を PC から取り外します。XBee Explorer USB の XBee ソケットから XBee-ZB デバイスを取り外します。このとき "Node Identifier" をメモした紙をデバイスに添付しておくことをお勧めします。

X-CTU で設定する Node Identifier 文字列について

X-CTU プログラムを使用して "Node Identifier" を設定するときに、大文字の ASCII 文字しか入力できなかったり、文字列の先頭にスペースが入ることがあります。この場合でも後からリモート操作で簡単に修正できますので構わずにそのままにしてください。初期設定ではデバイスの区別のみが目的ですので適当な名前をとりあえず付けてください。

13.7 TDCP,TDCPZB プログラム書き込み

TDCP プログラムまたは TDCPZB プログラムは、Atmega1284P マイクロコントローラを TDCP リモートコントローラボードに接続した状態で、市販のシリアルプログラムライタ(ISP 6ピンタイプ)で書き込みます。シリアルプログラムライタの ISP ケーブルを下記の様に ICSP ピンヘッダ(JP8)に接続します。

（ISP ケーブルを TDCP のピンヘッダに接続した図）

プログラム書き込み方法については、使用されるシリアルプログラムライタのマニュアルを参照してください。

シリアルプログラムライタで書き込む時に指定する Atmega1284Pプロセッサのフューズビットは、”TDCP動作仕様”の章に記載されていますので参照下さい。TDCP プログラムは 外部クロックでプロセッサを動作させますので、TDCP リモートコントローラボード上の電源とクロックを供給した状態で、書き込み操作を行って下さい。

13.8 動作確認

TDCP が正常に動作すると、シリアルコンソールに端末を接続していると下記のメッセージが表示されます。

（デフォルト通信条件: 9600bボー、 1ストップビット、パリティ無し、フロー制御無し）

```
initializing configuration..
TDCP -- Tiny Device Control Program --   ver1.00    (c)2009 All Blue System
app_mode=0
XBee's SerialNumber has been acquired
>
```

この状態で、シリアルコンソールからコマンド入力が可能です。

XBee データパケット経由のコマンド実行も同時に実行可能になっていますので、シリアルコンソールが無い状態でもリモートから全てのコマンド実行が可能です。

POARA,PORTC を共に出力ポートにして、全ビットの点灯と消灯を行ってみます。(app_mode 4 に設定)

```
>
>app_mode,4
>config_save
>reset
loading configuration..
TDCP -- Tiny Device Control Program --   ver1.00    (c)2009 All Blue System
app_mode=4
XBee's SerialNumber has been acquired
>
>port_write,FFFF
```

```
>port_read
port_read=FFFF
>port_write,0000
>port_read
port_read=0000
>
>help
TDCP -- Tiny Device Control Program --   ver1.00    (c)2009 All Blue System

available commands:

 help
 version
 reset
 server_addr[,<16BitAddrHexStr|64BitAddrHexStr>]
 frame_dump[,<1|0>]
 interval[,<value>]
(以降の表示省略).....
```

14　本製品に使用したソフトウェアライセンス表記

avr-libc License

avr-libc can be freely used and redistributed, provided the following license conditionsare met.

Portions of avr-libc are Copyright (c) 1999-2008
Werner Boellmann,
Dean Camera,
Pieter Conradie,
Brian Dean,
Keith Gudger,
Wouter van Gulik,
Bjoern Haase,
Steinar Haugen,
Peter Jansen,
Reinhard Jessich,
Magnus Johansson,
Harald Kipp,

Carlos Lamas,

Cliff Lawson,

Artur Lipowski,

Marek Michalkiewicz,

Todd C. Miller,

Rich Neswold,

Colin O'Flynn,

Bob Paddock,

Andrey Pashchenko,

Reiner Patommel,

Florin-Viorel Petrov,

Alexander Popov,

Michael Rickman,

Theodore A. Roth,

Juergen Schilling,

Philip Soeberg,

Anatoly Sokolov,

Nils Kristian Strom,

Michael Stumpf,

Stefan Swanepoel,

Helmut Wallner,

Eric B. Weddington,

Joerg Wunsch,

Dmitry Xmelkov,

Atmel Corporation,

egnite Software GmbH,

The Regents of the University of California.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the

distribution.

* Neither the name of the copyright holders nor the names of contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

15　サポートについて

TDCP のファームウエ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。

オールブルーシステムの TDCP 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、マイクロコントローラ自身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 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。

16　更新履歴

REV A.3.2 2015/03/28

回路図を CPUボード v1.2 に合わせて入れ替え(XBEE_RESET信号接続)

REV A.3.1 2014/03/16

ad_margin_reset コマンド新規追加

TDCPZBの項を追加。XBee-ZB ZigBee Series2 に対応した。

REV A.2.14 2013/06/20

コマンドエイリアスの追加、PORTD bit5 に XBee リセット信号出力を追加した。

REV A.2.13 2013/05/09

“position_report_rate” コマンド新規追加。

REV A.2.12 2012/07/31

“pulse” コマンドに連続出力モードフラグ〈continuous〉を追加した。

REV A.2.11 2011/06/13

I2C スレーブ機能を追加、”i2c_slave_addr”,”i2c_work_read”,”i2c_work_write” コマンドを追加した。

REV A.2.10 2011/06/01

app_mode = 10 動作モードを追加した。

“datetime”, “timezone”,”signal_event”,”signal_timer”,”ad_margin” コマンドを追加した。

“シグナル出力機能”の章を追加

“ADVAL_UPDATE” イベントを追加

NMEA-0183 シーケンス受信時にPORTB bit#3,#4 へのポート出力を行わないようにした

I2C 接続機能を追加、“i2c_use”,”i2c_read,”i2c_write” コマンドを追加した。

SPI 接続機能を追加、”spi_config”,”spi_read”,”spi_write” コマンドを追加した。

REV A.2.03 2011/04/20

動作環境にクロック周波数 8Mhzと電源電圧 3.3V 時の記述を追加した。

書き込み時に指定するFure ビットに電源電圧 3.3V 時の記述を追加した。

REV A.2.02 2011/02/24

server_addr,router コマンドで現在の設定値を取得する場合に、アドレス値が未設定の場合にアドレス値として “0” を返して、コマンドの実行結果ステータス(status) は “1” を返すようにした。

REV A.2.01 2011/01/19

reply_prefix コマンドを新規追加

REV A.2.00 2010/12/13

コードサイズ増加に対応するために、使用するマイクロプロセッサをATMega644P からATMega1284P に変更して、メジャーバージョン番号を 2.X に更新した。

REV A.1.11 2010/09/11

change_count_check コマンドに連続してイベントを送信する設定値 “2” を追加した。

COUNT_EXCEED イベントに、連続してイベントを送信する機能を追加した。

一部のLCD モジュールで発生した表示不具合を修正した。(app_mode=9の場合)

REV A.1.10 2010/05/25

リセット直後に PORTD bit#7 をLOW にしておくことで、EEPROM データを強制的に初期化する機能を追加

REV A.1.9 2010/05/20

新規イベント “GPS” を追加した。これに伴って、position_report, position_once コマンドのパラメータ値に”1: $GPRMC イベント” に加えて、“2: GPS イベント” を指定可能にした。

デバッグ・試験運用時用のコマンド frame_dump を削除した。

REV A.1.8 2010/05/07

イベントデータパケット($GPRMC を除く)中に 16ビットアドレス情報を追加した

serial_number コマンドで 16 ビットアドレスを同時に取得するようにした

イベント名 CHANGE_COUNT_EXCEED を COUNT_EXCEED に変更

heartbeat_rate コマンドと LIVE イベントを新規追加

REV A.1.7 2010/04/28

sample_once,adc_vref,router コマンドを新規追加

force_sampleコマンド説明を修正（サンプリングデータ送信に関する部分）

server_addr コマンドで 16bit幅のアドレスを扱えるようにした

REV A.1.6 2010/04/20

tx_ascii,echo コマンドで送信可能な文字列に空白文字とカンマを含めるようにした

tx_ascii コマンドの省略形式 tx を追加

lcd_disp コマンド説明の誤記を修正

REV A.1.5 2010/04/16

LCD 表示モジュール接続用の app_mode=9 を追加

lcd_clear,lcd_wrap,lcd_disp,lcd_lines,lcd_length コマンドを新規追加

Fuse bits 設定値 High byte(boot size)を変更

REV A.1.4 2010/03/23

change_count_check の説明に change_detect コマンド設定についての記述を追加

REV A.1.3 2010/01/04

change_count_check 実行時のリプライデータに change_count_high で設定した値を含める様にした

NMEA-0183 の表記間違いを修正

REV A.1.2 2009/12/20

force_sample 実行時に、カウンタ値はクリアしない様にした

REV A.1.1 2009/11/15

記述・レイアウト修正

REV A.1.0 2009/11/11

初版作成