TDCP328(Tiny Device Control Program for ATmega328)

TDCP328(for XBee 802.15.4 Series1)

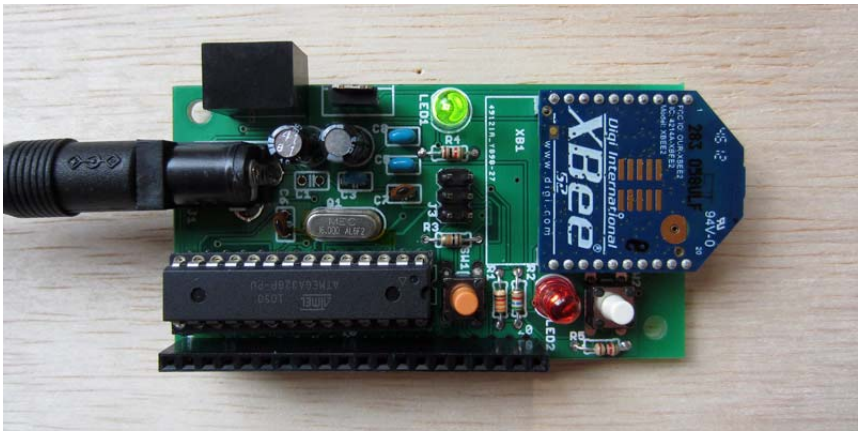
TDCPZB328(for XBee-ZB Series2)

ユーザーマニュアル

TDCP328 User Manual Rev A.2.7　　2015/8/8

TDCP328 firmware version: 2.57

TDCPZB328 firmware version: 1.10



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 5
1.1 著作権および登録商標 5
1.2 連絡先 5
2 使用条件およびライセンス 5
3 イントロダクション 6
4 TDCP動作仕様 10
4.1 マイクロコントローラ(ATmega328P) 10
4.1.1 TDCP 使用時 for XBee 802.15.4 (Series1) 10
4.1.2 TDCPZB使用時 for XBee-ZB (Series2) 11
4.2 Arduino互換ボード使用時のTDCPピン配置図 13
4.2.1 TDCP 使用時 for XBee 802.15.4 (Series1) 13
4.2.2 TDCPZB使用時 for XBee-ZB (Series2) 14
4.3 TDCP,TDCPZBファームウエア書き込み時の Fuse Bits 設定値 15
4.4 XBee モジュール設定値 16
4.4.1 XBee 802.15.4 RF Module Series1 (TDCP) 16
4.4.2 XBee-ZB ZigBee対応 RF Module Series2 (TDCPZB) 16
5 TDCPデータパケットフォーマット 17
5.1 XBee RF データパケット 18
5.1.1 TDCP XBee 802.15.4 Series1 の場合 18
5.1.2 TDCPZB XBee-ZB Series2 の場合 18
5.2 リクエストコマンド 19
5.3 リプライデータ 19
5.4 イベントデータ 20
6 TDCP 動作モード（設定変更はTDCPZBのみ） 20
6.1 app_mode 31 (TDCP 固定設定値) 21
6.2 app_mode 32 (TDCPZB デフォルト設定値) 21
6.3 app_mode 34 (TDCPZB 気圧・湿度センサ計測用) 21
7 TDCPコマンドリファレンス 22
7.1 version 23
7.2 reset 24
7.3 server_addr (TDCP専用) 24
7.4 config_save 25
7.5 sampling_rate 26
7.6 force_sample 27

7.7 sample_once 28
7.8 second_adjust 29
7.9 dio_config 30
7.10 pullup,dio_pullup 31
7.11 change_detect 32
7.12 dio_read,port_read 33
7.13 dio_bit,port_bit 34
7.14 dio_write,port_write 35
7.15 pulse 36
7.16 ad_read,adc_read 37
7.17 ad_vref,adc_vref 37
7.18 serial_number (TDCP専用) 38
7.19 tx_data (TDCP専用) 39
7.20 tx_ascii,tx (TDCP専用) 40
7.21 router (TDCP専用) 41
7.22 reply_prefix,rp (TDCP専用) 44
7.23 ad_margin 45
7.24 ad_margin_reset 46
7.25 i2c_use 47
7.26 i2c_read 49
7.27 i2c_write 50
7.28 i2c_write2 (TDCPZB 専用) 51
7.29 i2c_slave_addr 52
7.30 spi_config 53
7.31 spi_read 55
7.32 spi_write 56
7.33 counter_margin 57
7.34 heartbeat_rate (TDCPZB 専用) 58
7.35 app_mode (TDCPZB専用) 59

8 TDCPイベントリファレンス 60

8.1 CHANGE_DETECTイベント 61
8.2 SAMPLINGイベント(デフォルトapp_mode 31,32 の場合) 61
8.3 SAMPLINGイベント(app_mode 34 の場合) (TDCPZB専用) 63
8.4 ADVAL_UPDATEイベント 64
8.5 I2C_SLAVE_EVENTイベント 65
8.6 I2C_SLAVE_OVERFLOWイベント 65
8.7 COUNTER_UPDATEイベント 66
8.8 LIVEイベント (TDCPZB 専用) 67

9 I2C接続機能 68
9.1 I2C マスター機能 68
9.2 I2C スレーブ機能 69
9.2.1 TDCP 動作モード取得コマンド(0x01) 69
9.2.2 イベント送信コマンド(0x11) 70
9.2.3 ワークエリアデータ書き込みコマンド(0x21) 70
9.2.4 ワークエリアデータ読み出しコマンド(0x22) 70
10 SPI接続機能 71
11 TDCP 動作テスト用回路図 72
11.1 TDCP XBee 802.15.4 (Series1) 16MHz 5V 動作用回路図 72
11.2 TDCP XBee 802.15.4 (Series1) 8MHz 3.3V 動作用回路図 73
11.3 TDCPZB XBee-ZB (Series2) 16MHz 5V 動作用回路図 74
11.4 TDCPZB XBee-ZB (Series2) 8MHz 3.3V 動作用回路図 75
12 Arduino 互換ボード(Arduino FIO) へのTDCPインストール 76
12.1 TDCP XBee 802.15.4 Series1 初期設定 76
12.2 TDCPZB XBee-ZB ZigBee対応 Module Series2 初期設定 78
12.3 TDCP プログラム書き込み 85
12.4 動作確認 86
13 本製品に使用したソフトウェアライセンス表記 87
14 サポートについて 91
15 更新履歴 91

1 このマニュアルについて

1.1 著作権および登録商標

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

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

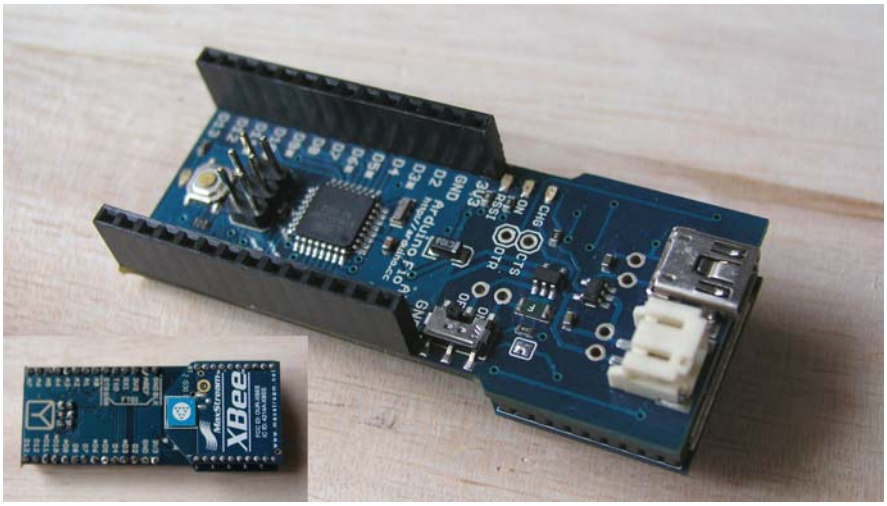
TDCP_328 リモートコントロールモニタプログラム(以降 “TDCP” と略す) の機能について説明します。

TDCP は、Atmel Atmega328P[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 (XBee 802.15.4 RF Module)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。



市販のAtmega328P CPU ボード(with XBee I/F)にTDCP を組み込んだ例

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。

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。

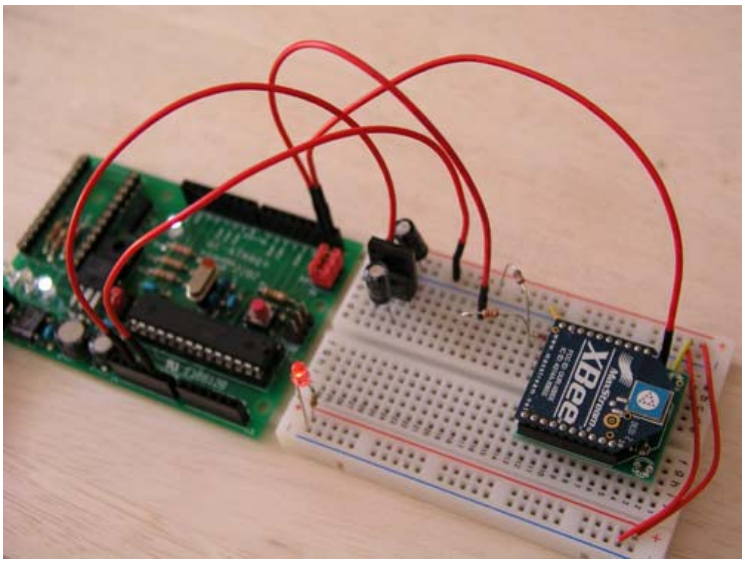
---

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。
- I/O ポート入力変化時にイベントデータパケット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 CPU ボードの 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パケット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 CPU ボードの 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。
- 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
- リモートコマンドやイベント送信時に、TDCP を搭載した CPU ボードを途中に複数経由して送信することができます。
- TDCPZB XBee-ZB のスリープモードに対応していて、リモートデバイスの消費電力を抑えることができます。
- TDCPZB TDCPZB プログラムでは ATmega328P プロセッサの省電力モードを積極的に使用していますので、リモートデバイスの消費電力を抑えることができます。

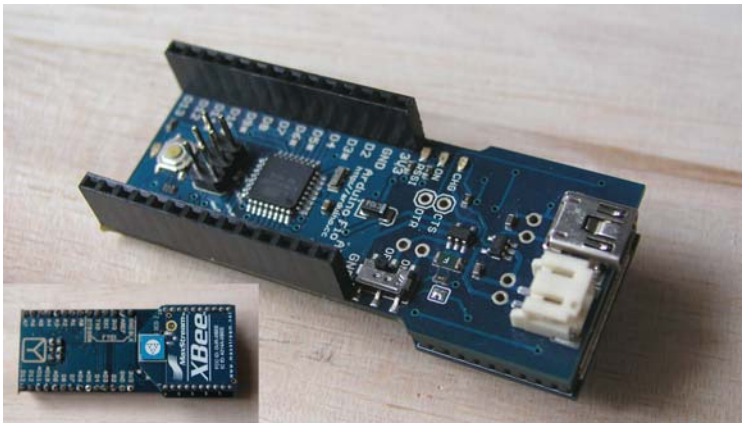
TDCP は、市販の Arduino[3] 互換ボード(ATmega328P プロセッサを搭載したもの)と XBee シールド[4]を組み合わせたハードウエア上で動作させることができます。Arduino は動作クロックが 16MHz で 5V 動作のものと、 8MHz で 3.3V 動作の２種類がありますが、TDCP はどちらの環境でも動作させることが出来ます。回路例は “TDCP動作テスト用回路図” の章を参照してください。



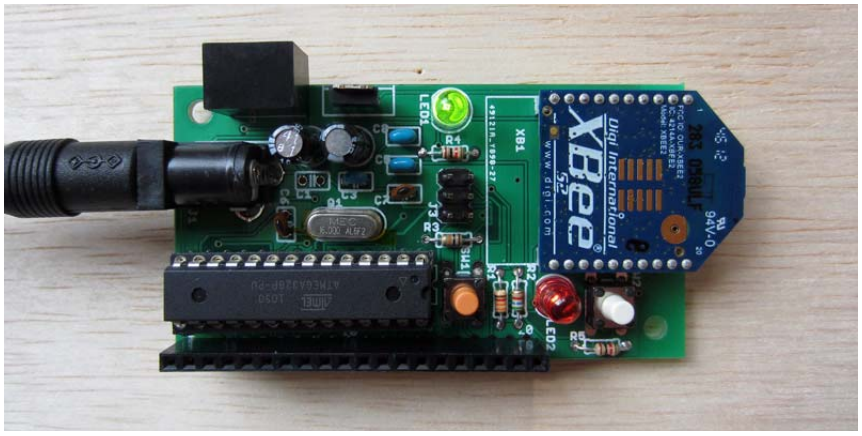
3 http://arduino.cc/en/

4 http://www.sparkfun.com/products/9976

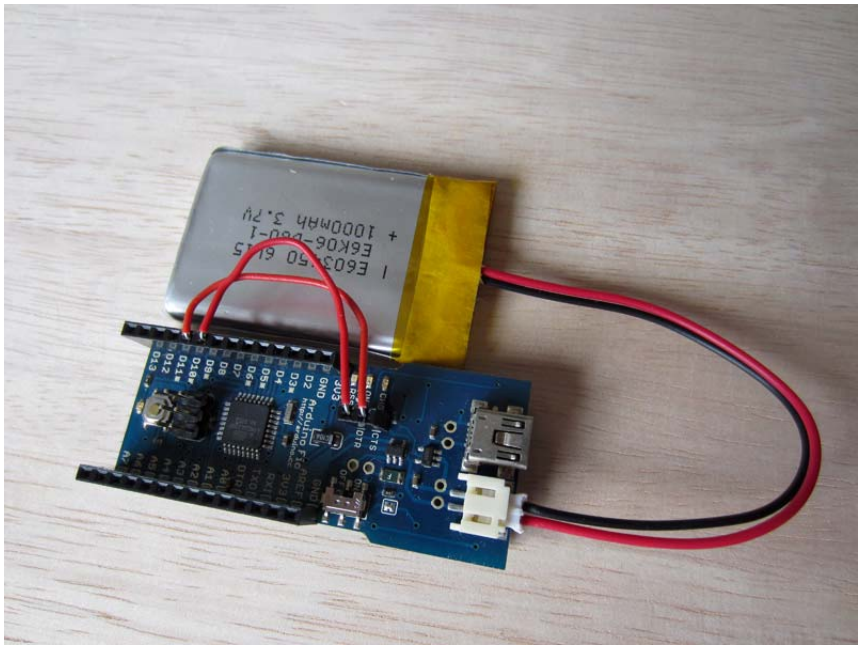
Arduino 互換ボード(16MHz 5V 動作) と XBee 電源用の3.3V レギュレータIC とXBee を組み合わせた例



Arduino FIO[5](8MHz 3.3 動作) に TDCP を組み込んだ例



TDCPZB 専用の CPUボード(ATmega328P 16MHz 5V or 8MHz 3.3V動作)に TDCPZB を組み込んだ例



5 http://www.sparkfun.com/products/10116

Arduino FIO(8MHz 3.3 動作) に TDCPZB を組み込んだ例。XBee-ZB のスリープ動作に対応させるためCTS,DTRピンにジャンパーを接続しています。先に紹介した TDCPZB 専用のCPU ボードでは基板内で配線済みです。

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 ATmega328P 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
- 電源 5Vまたは3.3V(ATmega328P 用) 、3.3V(XBee用)

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(Arduino 互換ボードや XBee シールド等)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(ATmega328P)

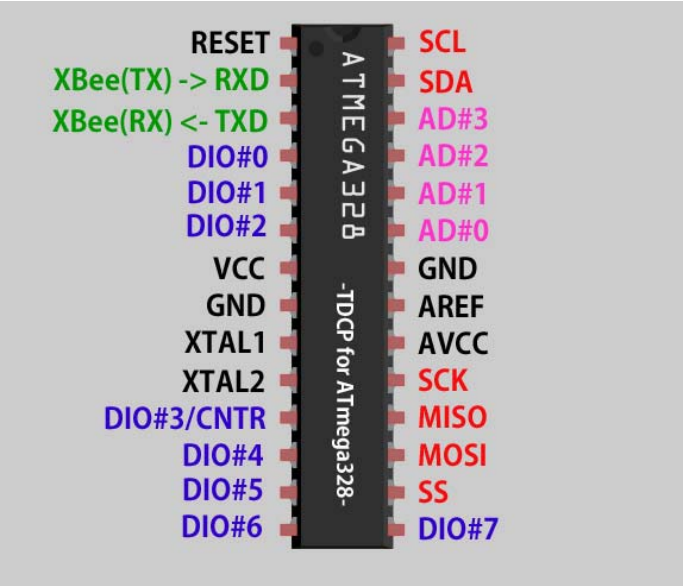
4.1.1 TDCP 使用時 for XBee 802.15.4 (Series1)

		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
PORTB	bit 0	DIO#6 “dio_config” コマンドで入力または出力ポートとして設定する。
	bit 1	DIO#7 “dio_config” コマンドで入力または出力ポートとして設定する。
	bit 2	SPI 通信時の SS(Slave Select) 信号を出力する
	bit 3	SPI 通信時の MOSI 信号を出力する リセット直後のTDCP 起動時に、LOW(GND 接続)しておくことでEEPROM データを初期化することができます。
	bit 4	SPI 通信時の MISO 信号を入力する
	bit 5	(1) SPI 通信時の SCK 信号を出力する。“spi_config”コマンドで SPI を有効に設定した場合 (2) TASK_RUN LED: TDCP 内のタイマータスク処理間隔(約10ms)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。"spi_config”コマンドで SPI を無効に設定した場合(デフォルト値)
PORTC	bit 0	A/D #0
	bit 1	A/D #1

	bit 2	A/D #2
	bit 3	A/D #3
	bit 4	I2C 通信時の SDA
	bit 5	I2C 通信時の SCL
PORTD	bit 0	RXD XBee のデータパケット送受信用。通信条件は 9600bps 8bit no-parity 固定。
	bit 1	TXD XBee のデータパケット送受信用。通信条件は 9600bps 8bit no-parity 固定。
	bit 2	DIO#0 “dio_config” コマンドで入力または出力ポートとして設定する。
	bit 3	DIO#1 “dio_config” コマンドで入力または出力ポートとして設定する。
	bit 4	DIO#2 “dio_config” コマンドで入力または出力ポートとして設定する。
	bit 5	DIO#3 “dio_config” コマンドで入力または出力ポートとして設定する。 このポートは常にカウンタ入力として取り込まれて、サンプリングデータに出力されます
	bit 6	DIO#4 “dio_config” コマンドで入力または出力ポートとして設定する。
	bit 7	DIO#5 “dio_config” コマンドで入力または出力ポートとして設定する。

(*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。

CPU ピン配置図



4.1.2 TDCPZB使用時 for XBee-ZB (Series2)

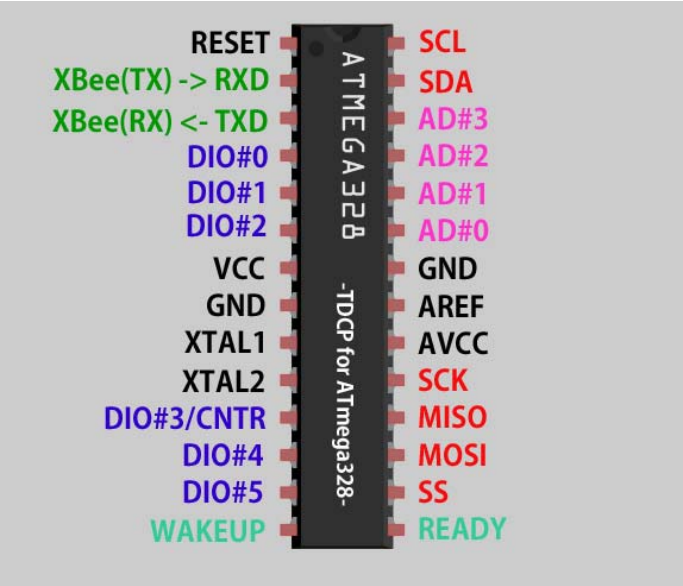
		設定値
CPU クロック		16MHz または 8MHz
電源電圧		5V または 3.3V(*1)
Flash memory		TDCPZB プログラム格納用
EEPROM		TDCPZB コンフィギュレーション保存用
PORTB	bit 0	WAKEUP 信号出力

		XBee-ZB デバイスの DTR/SLEEP_RQ/DIO8(9pin)に接続してください。 このときXBee-ZB デバイスの DTR/SLEEP_RQ/DIO8 ピンのプルアップ(3.3V)を有効にしてください。XBee-ZB デバイスを End Device ファームウエアで使用する場合には、スリープモードを 5 (Cyclic Sleep with pin wake) に設定します。 このピンは通常ハイインピーダンス状態になっています。TDCPZB が XBee-ZB デバイスにシリアルデータを送信する直前に Low アクティブ信号(Sink)を出力します。その後、PORTB bit1(後述)が Low になるまでウェイトした後、シリアルデータを送信します。シリアルデータの送信が終了したときにハイインピーダンス状態に戻します。 このピンを XBee-ZB に接続しない場合や、XBee-ZB のスリープモードを 5 以外に設定することもできますが、この場合には XBee-ZB デバイスがスリープ状態から抜けるまでシリアルデータは送信されません。
	bit 1	READY 信号入力 XBee-ZB デバイスの CTS/DIO7(12pin)に接続してください。XBee-ZB デバイスを End Device ファームウエアで使用するときには、CTS/DIO7ピンの設定を "CTS" モードに設定してください。 Router ファームウエアで使用する場合にも同様に"CTS" モードで使用します。このピンを GND に接続することで、常に ATmega328P からのデータ送信を可能にすることもできます。このピンは常に入力モードでハイインピーダンス状態になっています。XBee-ZB デバイス側で CTS/DIO7 ピンのプルアップ(3.3V)を有効にしてください。 TDCPZB は XBee-ZB デバイスのシリアルデータを送信するときに、このピンが High の場合には内部でウェイトします。Low の時にシリアルデータをXBee-ZB デバイスに送信します。
	bit 2	SPI 通信時の SS(Slave Select) 信号を出力する
	bit 3	SPI 通信時の MOSI 信号を出力する リセット直後のTDCP 起動時に、Low(GND 接続)しておくことでEEPROM データを初期化することができます。
	bit 4	SPI 通信時の MISO 信号を入力する
	bit 5	(1) SPI 通信時の SCK 信号を出力する。“spi_config”コマンドで SPI を有効に設定した場合 (2) TASK_RUN LED: TDCP 内のタイマータスク処理間隔(約10ms)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。"spi_config"コマンドで SPI を無効に設定した場合(デフォルト値)
PORTC	bit 0	A/D #0
	bit 1	A/D #1
	bit 2	A/D #2
	bit 3	A/D #3
	bit 4	I2C 通信時の SDA
	bit 5	I2C 通信時の SCL

PORTD	bit 0	RXD XBee のデータパケット送受信用。通信条件は 9600bps 8bit no-parity 固定。
	bit 1	TXD XBee のデータパケット送受信用。通信条件は 9600bps 8bit no-parity 固定。
	bit 2	DIO#0 “dio_config” コマンドで入力または出力ポートとして設定する。
	bit 3	DIO#1 “dio_config” コマンドで入力または出力ポートとして設定する。
	bit 4	DIO#2 “dio_config” コマンドで入力または出力ポートとして設定する。
	bit 5	DIO#3 “dio_config” コマンドで入力または出力ポートとして設定する。 このポートは常にカウンタ入力として取り込まれて、サンプリングデータに出力されます
	bit 6	DIO#4 “dio_config” コマンドで入力または出力ポートとして設定する。
	bit 7	DIO#5 “dio_config” コマンドで入力または出力ポートとして設定する。

(*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。

CPU ピン配置図



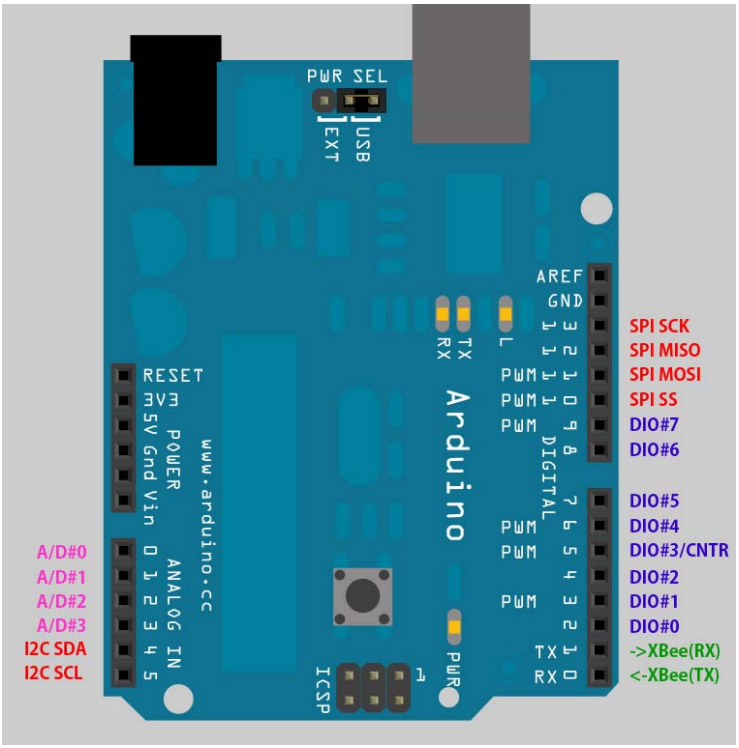
4.2 Arduino互換ボード使用時のTDCPピン配置図

4.2.1 TDCP 使用時 for XBee 802.15.4 (Series1)

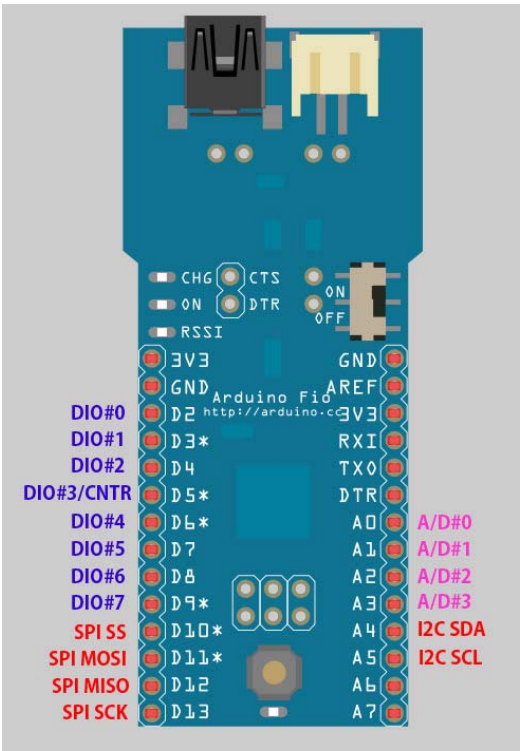
Arduino UNO[6] とその互換ボード用の TDCP ピン配置図

XBee との接続は XBee シールド等を使用して、XBee 用電源とシリアル信号のレベル変換を行ってください。

6 http://arduino.cc/en/Main/Hardware 参照



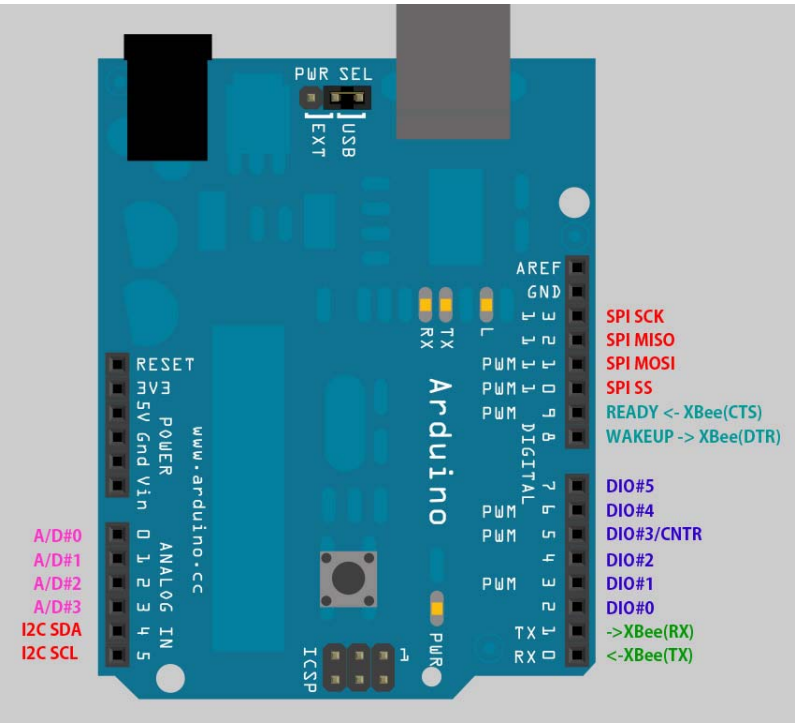
Arduino FIO ボード用の TDCP ピン配置図(XBee と ATmega328Pはボード内で結線されています)



4.2.2 TDCPZB使用時 for XBee-ZB (Series2)

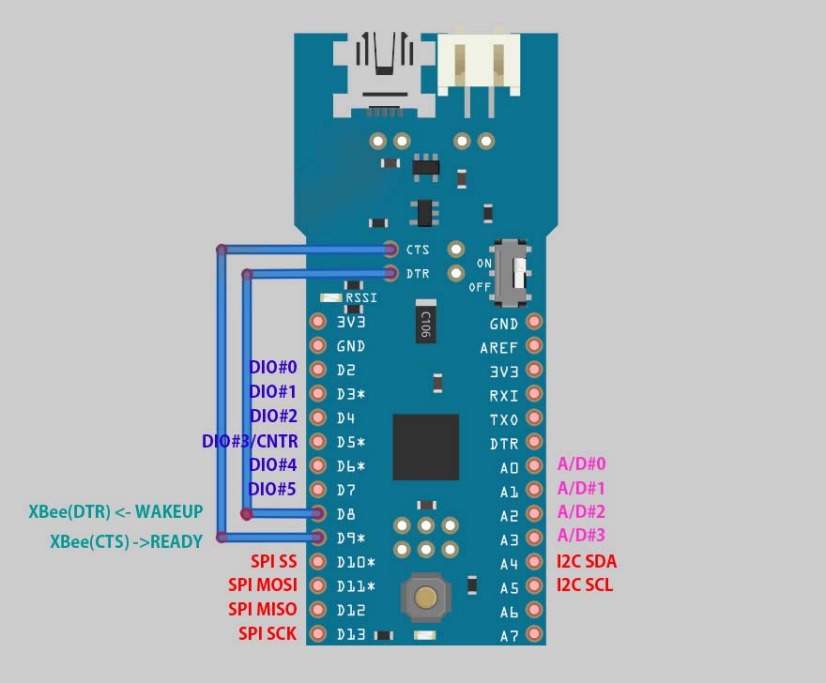
Arduino UNO とその互換ボード用の TDCP ピン配置図

XBee との接続は XBee シールド等を使用して、XBee 用電源とシリアル信号のレベル変換を行ってください。



Arduino FIO ボード用の TDCP ピン配置図(XBee と ATmega328Pはボード内で結線されています)

TDCPZB の WAKEUP信号ピン(D8)は Arduino FIO の DTRパターンに接続します。また同様に READY 信号ピン(D9)は Arduno FIO の CTSパターンに接続します。



4.3 TDCP,TDCPZBファームウエア書き込み時の Fuse Bits 設定値

電源電圧 5V で使用する場合、16MHz,8MHz 用ファームウエア共通 (Arduino UNO 等)

Fuse Bits	設定値

Low Byte	0b11010111
High Byte	0b11011111
Extended Byte(*1)	0b100 (BOD=4.3V)

電源電圧 3.3V で使用する場合、8MHz 用ファームウエア用 (Arduino FIO 等)

Fuse Bits	設定値
Low Byte	0b11010111
High Byte	0b11011111
Extended Byte(*1)	0x110 (BOD=1.8V)
	0x101 (BOD=2.7V) app_mode=34で動作させる時に指定します。この BOD 設定により、外付 I2Cセンサの動作電圧を保障します。

(*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。

4.4 XBee モジュール設定値

4.4.1 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Ox3332) 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。

4.4.2 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 (for ATmega328) で使用可能なのは Routerまたは End device タイプです。

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 または ZIGBEE END DEVICE API を使用してください。
API モード	1 ZIGBEE ROUTER API または ZIGBEE END DEVICE 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
Sleep モード (XBee-ZB ファームウエアに END DEVICE タイプを選択した場合に設定します)	5 ファームウエアに ZIGBEE END DEVICE API を使用するときには Sleepモードを "Cyclic Sleep with pin wake" に設定します。これによってTDCPZB プログラムからデータ送信するときに、XBee-ZB デバイスに設定されたスリープパラメータに影響されないで、確実にデータを送信できます。
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 入力が予め設定した条件になった場合に、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。イベントデ

ータ送信先の XBee デバイスは予め設定しておくことが出来ます。

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します。

5.1.1 TDCP XBee 802.15.4 Series1 の場合

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ります。

```
<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。

5.1.2 TDCPZB XBee-ZB Series2 の場合

XBee-ZB 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ります。

```
<StartDelimiter><Length><FrameType><FrameID><64bit DestinationAddress>
    <16bit DestinationNetworkAddress><BroadcastRadius><Options><RFData><CheckSum>
```

リクエストコマンドデータ送信時は、上記の〈FrameType〉に 0x10 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-ZB 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 = 0x8B）を XBee-ZB から受信することになりま

す。

XBee-ZB 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><FrameType><64bit SourceAddress>
    <16bit SourceNetworkAddress><Options><RFData><CheckSum>
```

リプライデータは、上記の〈FrameType〉に 0x90 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-ZB 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-ZB モジュールは、Coordinator デバイスのみで変更することはできません。

XBee-ZB データパケットの詳細については、"XBeeZB 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,0A01,31,FF,0,0,100,120,130,140”

“$$$,CHANGE_DETECT,0D04,31,01,FF”

6 TDCP 動作モード (設定変更はTDCPZBのみ)

TDCPZB は動作モードを変えることで、自動サンプリング機能で送信するイベントデータ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。

TDCP では動作モードを変更することはできません。常にデフォルト app_mode = 31 で動作します。

以降で動作モード毎の機能を説明します。

6.1 app_mode 31 (TDCP 固定設定値)

- **機能概要**

  TDCP モニタプログラムで使用するデフォルト動作モード。TDCP ではこの値を変更することはできません。

  サンプリングイベント発生時に送信されるイベントデータの詳細は、"TDCPイベントリファレンス"の章中の "SAMPLING" イベントの項を参照してください。

- **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 32 (TDCPZB デフォルト設定値)

- **機能概要**

  TDCPZB モニタプログラムで使用するデフォルト動作モード。

  サンプリングイベント発生時に送信されるイベントデータの詳細は、"TDCPイベントリファレンス"の章中の "SAMPLING" イベントの項を参照してください。

6.3 app_mode 34 (TDCPZB 気圧・湿度センサ計測用)

- **機能概要**

  TDCPZB モニタプログラムで外部 I2C バスに気圧センサ(LPS331) と 湿度センサ(AM2321) を接続したときに使用可能な動作モード。

  この app_mode を使用する時には I2Cバスにこれらのセンサデバイスを接続してください。接続していない 場合には正常にモニタプログラムが動作しません。

  このとき、LPS331 の SAO ピンは Vcc に接続して 7bit I2C スレーブアドレスを 0x5D に設定してください。

  サンプリングイベント発生時には、これらの外部センサから現在の計測値を自動で取得してイベントメッセー ジ中に格納して送信されます。サンプリングイベント発生時に送信されるイベントデータの詳細は、"TDCPイベントリファレンス"の章中の"SAMPLING" イベントの項を参照してください。

- **設定例**

  TDCPZB が動作している CPU ボードの I2C バスに気圧センサ(LPS331) と 湿度センサ(AM2321) を接続した後、動作モードと自動サンプリングの設定をします。

  (設定例)

  I2C バスのプルアップは ATmega328P 内蔵のプルアップ抵抗を使用

  自動サンプリング間隔は 10分(600秒)

```
i2c_use,2
app_mode,34
samling_rate,600
config_save
reset
```

- **SAMPLING イベント受信例**

```
$$$,SAMPLING,34,141,685,679,674,669,0361653FE7DE,030402E000F3B1E3
```

  上記のイベントデータ中に含まれる 気圧センサ(LPS331) と 湿度センサ(AM2321)のセンサデータをサーバー側でデコードすることで、温度(AM2321):24.2(℃)、気圧(LPS331):1014.2(hPa),湿度(AM2321):73.8(%) が得られます。

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。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。

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。

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

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やコマンドの実行結果によっては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バイトまで送受信可能ですが、TDCPZB ファームウエア上では 100bytes までに制限しています。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。

"pullup","i2c_use","dio_config"

7.1 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**

XBee	<TDCPPrefix>,version

- **リプライデータフォーマット**

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

  <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,1,2.50"

7.2 reset

- **機能概要**

  CPU のリセット

- **リクエストフォーマット**

XBee	<TDCPPrefix>,reset

- **リプライデータフォーマット**

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 **

7.3 server_addr (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

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

- **リプライデータフォーマット**

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”)

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 の場合は 常に Coordinator デバイス”0000000000000000” になります

<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 Coordinator を示す ″0000000000000000″ が設定されていて変更することはできません。

- **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 **

7.4　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**

XBee	<TDCPPrefix>,config_save

- **リプライデータフォーマット**

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”

PORTB bit#3(MOSI) をリセット時にGND に接続しておくことで、プロセッサ内部に保存された EEPROM の内容を強制的に初期化して各設定をデフォルト設定値に戻すことができます。

- **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”)**

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

7.5 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**

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

- **リプライデータフォーマット**

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。

<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”

7.6 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**

XBee	<TDCPPrefix>,force_sample

- **リプライデータフォーマット**

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>　　サンプリングデータ項目が入る。(“イベントリファレンス” 章の “SAMPLING” イベントの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 **

7.7　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**

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

- **リプライデータフォーマット**

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。

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”

7.8 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**

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

- **リプライデータフォーマット**

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 では CPUクロックを元に１ミリ秒毎のタイマー割り込みが発生して、<radio> で指定された回数の割り込みが発生することで１秒を計算しています。このため、１秒の時間精度は 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。

  <ratio> の値を少なくすると1 秒をカウントするまでの時間が短くなりますので、クロックのスピードが早くなったのと同様になります。<ratio> の値を変更することでCPUクロックスピードを補正して、サンプリング期間やサンプリングイベントデータ中のカウンタ値の計測時間を正確にすることができます。

  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。

  ミリ秒単位のパラメータを扱うコマンド(“pulse” コマンド” 等) では、<ratio> を変更しても時間補正することはできません。CPUクロックの精度のみに依存します。

- **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”)**

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

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

7.9　dio_config

- **機能概要**

  DIO ポートの入力または出力モード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**

XBee	<TDCPPrefix>,dio_config[,<8BitHex>]

- **リプライデータフォーマット**

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>　出力モードに設定するビットを “1” に、入力モードに設定するビットを “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)

TDCPZB では bit6, bit7 の設定値は無視されて常に 0 が設定されます。

<status>　“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

DIO ポートの入出力モード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 “dio_config” 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。

“dio_config”,コマンド実行例 (全ポートを入力に設定、プルアップを有効にする)

```
$$$,dio_config,00
$$$,pullup,FF
$$$,config_save
$$$,reset
```

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”)**

```
“$$$abc,dio_config,FF” -> “$$$abc,1”
“$$$12345,dio_config” -> “$$$12345,1,00”
```

7.10　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**

XBee	<TDCPPrefix>,pullup[,<8BitHex>]

- **リプライデータフォーマット**

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>　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:00)
TDCPZB では bit6, bit7 の設定値は無視されて常に 0 が設定されます。

<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。
“dio_config” コマンドで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。

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”,コマンド実行例 (入力ポートの全プルアップ)

```
$$$,dio_config,00
$$$,pullup,FF
$$$,config_save
$$$,reset
```

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”)**

```
“$$$abc,pullup,FF” -> “$$$abc,1”
“$$$12345,pullup” -> “$$$12345,1,00”
```

7.11　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**

XBee	<TDCPPrefix>,change_detect[,<8BitHex>]

- **リプライデータフォーマット**

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> 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:00)
TDCPZB では bit6, bit7 の設定値は無視されて常に 0 が設定されます。
<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。”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。

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”)**

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

7.12 dio_read,port_read

- **機能概要**

デジタル入出力ポート値の取得

- **リクエストフォーマット**

XBee	<TDCPPrefix>,port_read

- **リプライデータフォーマット**

XBee	<TDCPPrefix>,<status>,<port_value>,<dio_config>

- **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。
TDCPZB では bit6, bit7 の値は常に 0 になります。

<dio_config>　“dio_config” コマンドで設定した、現在のデジタルポート入出力モード設定値を8ビット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**

“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。

<dio_config> に格納される入出力モードは、”dio_config” コマンドで設定した値が入ります。

- **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”)**

“$$$abc,port_read” -> “$$$abc,1,FF,00”

7.13　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**

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

- **リプライデータフォーマット**

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。
	TDCP の場合には 0 から 7 までの整数値が設定できます。
	TDCPZB の場合には 0 から 5 までの整数値が設定できます。
<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**

- **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”

7.14 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**

XBee	<TDCPPrefix>,port_write[,<8BitHex>]

- **リプライデータフォーマット**

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
<8BitHex>	出力ポートに設定する値を16進数表記で指定する。
	TDCPZB では bit6, bit7 に指定した値は無視されます。
<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”)**

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

7.15 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**

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

- **リプライデータフォーマット**

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。 “dio_config”で出力モードに設定されていて、かつ DIO#0 から DIO#3 までの 4ビットのみが指定できます。(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。

- **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”

7.16 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**

XBee	<TDCPPrefix>,adc_read

- **リプライデータフォーマット**

XBee	<TDCPPrefix>,<status>,<ad#0>,<ad#1>,<ad#2>,<ad#3>

- **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
<ad#0>..<ad#3>	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**

- **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”

7.17 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**

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

- **リプライデータフォーマット**

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 Reserved

3: REFS1:REFS0 = 1:1 (Internal 1.1V 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”

7.18 serial_number (TDCP専用)

- **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**

XBee	<TDCPPrefix>,serial_number

- **リプライデータフォーマット**

TDCP

XBee	<TDCPPrefix>,<status>,<xbee_serial>,<my_addr16>

- **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”)**

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

7.19　tx_data (TDCP専用)

- **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

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

- **リプライデータフォーマット**

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。

<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”

7.20　tx_ascii,tx (TDCP専用)

- **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

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

- **リプライデータフォーマット**

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。

<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”

7.21 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**

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

- **リプライデータフォーマット**

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。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。

7.22 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**

XBee	<TDCPPrefix>,reply_prefix,<insert_prefix>

- **リプライデータフォーマット**

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。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 **

7.23 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**

XBee	<TDCPPrefix>,ad_margin,<channel>[,<ad_margin>]

- **リプライデータフォーマット**

XBee	<TDCPPrefix>,<status>[,<ad_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
<chennel> A/D チャンネルを指定する。0 から 3 までの整数値
<ad_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**

<ad_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。

<ad_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 <ad_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。<ad_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”

7.24 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**

XBee	<TDCPPrefix>,ad_margin_reset

- **リプライデータフォーマット**

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”

7.25　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**

XBee	<TDCPPrefix>,i2c_use[,<flag>]

- **リプライデータフォーマット**

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。

<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。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。

“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”
```

7.26 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**

XBee	<TDCPPrefix>,i2c_read,<SlaveAddrHexStr>,<length>

- **リプライデータフォーマット**

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”

7.27 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**

XBee	<TDCPPrefix>,i2c_write,<SlaveAddrHexStr>,<DataHexStr>[,<length>]

- **リプライデータフォーマット**

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”
```

7.28 i2c_write2 (TDCPZB 専用)

- **機能概要**

I2C に接続されたスレーブデバイスに複数のトランザクションデータを書き込む。

- **リクエストフォーマット**

XBee	<TDCPPrefix>,i2c_write2,<SlaveAddrHexStr>,<DataHexStr#1>[,..,<DataHexStr#n>]

- **リプライデータフォーマット**

XBee	<TDCPPrefix>,<status>,<transaction_count>,<complete_count>

- **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#n>	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。 カンマで区切って複数のデータ列を指定することができる。 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
<transaction_count>	スレーブデバイスへI2C 書き込み操作を行った回数

<complete_count>　　上記書き込み操作で、I2C バス操作が正常に終了した回数

<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 デバイスに複数の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。

カンマ区切りで指定した複数の書き込みデータに対して、各データ毎にI2C の書き込みトランザクションの開始と終了処理が実行されます。複数の書き込みトランザクションを実行するときに、約20ms のウェイトが各トランザクション間に挿入されます。

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”)**

```
// I2C LCD デバイスの初期化コマンド実行
“$$$abc,i2c_write2,3E,0038,0039,0004,0014,0070,005E,006C” -> “$$$abc,1,7,6”
“$$$abc,i2c_write2,3E,0038,000C,0001” -> “$$$abc,1,3,2”
```

7.29　　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**

XBee	<TDCPPrefix>,i2c_slave_addr[,<8BitAddrHexStr>]

- **リプライデータフォーマット**

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 **

7.30　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**

XBee	<TDCPPrefix>,spi_config[,<spi_use>[,<clock_div>,<spi_mode>,<LSB_first>]]

- **リプライデータフォーマット**

XBee	<TDCPPrefix>,<status>[,<spi_use>,<clock_div>,<spi_mode>,<LSB_first>]

- **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 ATmega328P 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)

<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”

7.31 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**

XBee	<TDCPPrefix>,spi_read,<length>

- **リプライデータフォーマット**

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"

7.32 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**

XBee	<TDCPPrefix>,spi_write,<DataHexStr>

- **リプライデータフォーマット**

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。
<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。

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
“$$$abc,spi_write,1111” -> “$$$abc,1,0000”
```

7.33 counter_margin

- **機能概要**

  カウンタ入力変化イベント(COUNTER_UPDATE)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**

XBee	<TDCPPrefix>,counter_margin,[,<cntr_margin>]

- **リプライデータフォーマット**

XBee	<TDCPPrefix>,<status>[,<cntr_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
<cntr_margin>	カウンタ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 32767 までの整数を指定する。 0 を指定した場合は カウンタ値変化のイベントを検出しない。
<status>	“1” でコマンド実行成功、”0” で失敗を表す

- **備考**

<cntr_margin> に 0 を設定した場合はカウンタ値変化のイベント(COUNTER_UPDATE)検出を行いません。この場合でもカウンタ自身の機能は常に有効で、サンプリングイベント(SAMPLING)には現在のカウンタ値や周波数値が設定されます。

<cntr_margin>で指定した値以上にカウンタ値が変化した場合には COUNTER_UPDATE イベントが送信されます。カウンタ変化値は、前回 COUNTER_UPDATE イベントを送信した時からのカウント数を計測しています。

カウンタ値変化(COUNTER_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 <cntr_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 <cntr_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。<cntr_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”)**

“$$$abc,counter_margin” -> “$$$abc,1,5”
“$$$12345,counter_margin,20” -> “$$$12345,1”

7.34 heartbeat_rate (TDCPZB 専用)

- **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**

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

- **リプライデータフォーマット**

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。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”

7.35　app_mode (TDCPZB専用)

- **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**

XBee	〈TDCPPrefix〉,app_mode[,〈app_mode〉]

- **リプライデータフォーマット**

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。
(TDCPZB デフォルト設定値:32)

〈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。

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”)**

“$$$abc, app_mode, 32” -> “$$$abc, 1”
“$$$12345, app_mode” -> “$$$12345, 1, 32”

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。

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・設定しますので、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。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、10ms 間隔(以降 “インターバル”)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 インターバル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、次のインターバルまで遅延させて実行します。

連続してイベントが発生している場合にサーバー側に、イベントパケットが送信されずに取りこぼす場合があります。この場合にはイベント発生条件を緩和してデータ送信の頻度を下げてください。(“ad_margin”, “counter_margin”,”change_detect”,”sampling_rate” コマンドのパラメータを調整する)

8.1 CHANGE_DETECTイベント

- **イベント発生条件**

I/O ポートの値が変化した場合に発生します。

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

- **イベントデータフォーマット**

TDCP

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

TDCPZB

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

<my_addr16> 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> TDCP モニタプログラムのアプリケーションモードを示します。

<diff_bits> 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、8bit幅の16進数表記で設定されます。

<dio> 現在のポート値が 8bit 幅の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)

“change_detect” 検出対象のポートとそのビットを指定する

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

8.2 SAMPLINGイベント(デフォルトapp_mode 31,32 の場合)

- **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**

TDCP

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

TDCPZB

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

<my_addr16>	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>	TDCP モニタプログラムのアプリケーションモードを示します。
<dio>	現在のポート値が8bit 幅の16進数表記で設定されます。
<freq>	DIO#3(PORTD bit5) のポート値が１秒間に何回変化したかを示す周波数値が、10 進数表記で設定されます。周波数値は 1 秒毎に更新されて”SAMPLING”イベント発生時には最新の値が入ります。カウント可能な周波数の最大値は 65535 (2^16 - 1)で、これ以上の値になった場合には オーバーフローを表す –1 が設定されます。
<counter>	カウント期間(前回の “SAMPLING” イベント発生後から今回の “SAMPLING” イベント発生までの間) に、DIO#3(PORTD bit5) のポート値が何回変化したかを示すカウンタ値が、10 進数表記で設定されます。カウント可能な最大数は 2147483647 (2^31 - 1)で、これ以上の値になった場合には オーバーフローを表す –1 が設定されます。カウント期間中に一度でも １秒間に 65535 回以上ポートが変化した場合には、同様にオーバーフローを表す –1 が設定されます。
<adc0>.. <adc3>	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。
“sample_once”	手動でサンプリングイベントを１回だけ強制的に発生させる

- **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)

<freq>、<counter> に記録されるカウンタ値と、“sampling_rate” コマンドで指定した秒間隔の時間精度は 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。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。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。

8.3 SAMPLINGイベント(app_mode 34 の場合) (TDCPZB専用)

- **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**

TDCPZB

```
$$$,SAMPLING,<app_mode>,<counter>,<adc0>,<adc1>,<adc2>,<adc3>,<LPS331_data>,<AM2321_data>
```

<my_addr16> 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> TDCP モニタプログラムのアプリケーションモードを示します。

<counter> カウント期間(前回の “SAMPLING” イベント発生後から今回の “SAMPLING” イベント発生までの間) に、DIO#3(PORTD bit5) のポート値が何回変化したかを示すカウンタ値が、10 進数表記で設定されます。カウント可能な最大数は 2147483647 (2^31 - 1)で、これ以上の値になった場合には オーバーフローを表す –1 が設定されます。カウント期間中に一度でも 1秒間に 65535 回以上ポートが変化した場合には、同様にオーバーフローを表す –1 が設定されます。

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

<LPS331_data> I2C バスに接続された LPS331 気圧センサの計測値。以下の LPS331 内部レジスタの値が順に6バイトの16進数表記で格納されています。

STATUS_REG (レジスタアドレス 0x27)

PRESS_POUT_XL_REH (レジスタアドレス 0x28)

PRESS_OUT_L (レジスタアドレス 0x29)

PRESS_OUT_H (レジスタアドレス 0x2A)

TEMP_OUT_L (レジスタアドレス 0x2B)

TEMP_OUT_H (レジスタアドレス 0x2C)

<AM2321_data> I2C バスに接続された AM2321湿度センサの計測値。以下の AM2321 計測データ値が順に8バイトの16進数表記で格納されています。

Function Code (0x03 固定)

Returns the number of bytes (0x04 固定, 計測エラー発生時には 0x05が入る)

low humidity byte

high humidity byte

low temperature byte

high temperature byte

low CRC code

high CRC code

- 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。

“sample_once”　　手動でサンプリングイベントを１回だけ強制的に発生させる

- **備考**

LPS331 の SAO ピンは Vcc に接続してスレーブアドレスは 0x5D に設定してください。

LPS331 と AM2321 デバイスは I2C バスに接続した状態で使用します。″i2c_use″ コマンドを使用して I2C バスを有効にしてください。I2C バスのプルアップ設定は ″i2c_use″コマンドで内蔵または外部でのプルアップを設定できます。センサ回路に合わせて設定してください。

“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、LPS331 と AM2321 の計測が開始され、計測結果と A/D 変換値、カウンタ値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)

<counter> に記録されるカウンタ値と、“sampling_rate” コマンドで指定した秒間隔の時間精度は 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。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。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。

8.4　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。

- **イベントデータフォーマット**

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>　　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>　　TDCP モニタプログラムのアプリケーションモードを示します。

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

例えば、<ad_update_bits> が 03 の場合には 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>　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。

8.5　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>　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>　TDCP モニタプログラムのアプリケーションモードを示します。

<event_data>　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。

8.6　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>　　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>　　TDCP モニタプログラムのアプリケーションモードを示します。

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

“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。

8.7　COUNTER_UPDATEイベント

- **イベント発生条件**

カウンタ値が “counter_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。

- **イベントデータフォーマット**

TDCP

```
$$$,COUNTER_UPDATE,<my_addr16>,<app_mode>,<upd_counter>
```

TDCPZB

```
$$$,COUNTER_UPDATE,<app_mode>,<upd_counter>
```

<my_addr16>　　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>　　TDCP モニタプログラムのアプリケーションモードを示します。

<upd_counter>　　前回の “COUNTER_UPDATE” イベント発生後から今回の “COUNTER_UPDATE” イベント発生までの間) に、DIO#3(PORTD bit5) のポート値が何回変化したかを示すカウンタ値が10 進数表記で設定されます。カウント可能な最大数は 32767 (2^15－1)です。これ以上の値になった場合には オーバーフローを表す –1 が設定されます。カウント期間中に一度でも１秒間に 65535 回以上ポートが変化した場合には、同様にオーバーフローを表す –1 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)

“counter_margin”　カウンタ変化値を比較するときに使用するマージン値の設定または取得

- **備考**

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。

TDCP はcounter_margin で設定したマージン値以上にカウンタ値が変化しているかを定期的にチェックしてイベントを送信します。このため <upd_counter> の値は常に、counter_magin で設定した値以上の値になります。(オーバーフロー時を除く)

8.8　LIVEイベント (TDCPZB 専用)

- **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**

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>　　TDCP モニタプログラムのアプリケーションモードを示します。

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

“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。

9 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。

”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。”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ので、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。

9.1 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。

9.2 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 マスターデバイスが I2C バスを占有するようにして下さい。

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32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。

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。

9.2.1 TDCP 動作モード取得コマンド(0x01)

TDCP の現在の動作モードを取得します。

TDCP TDCPの場合は固定値 31 が返ります。

TDCPZB TDCPZBの場合は固定値 32 が返ります。

(write) 0x01	(read) <app_mode> 1 byte

9.2.2 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。

9.2.3 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 32 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。

9.2.4 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 32 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。

10 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。

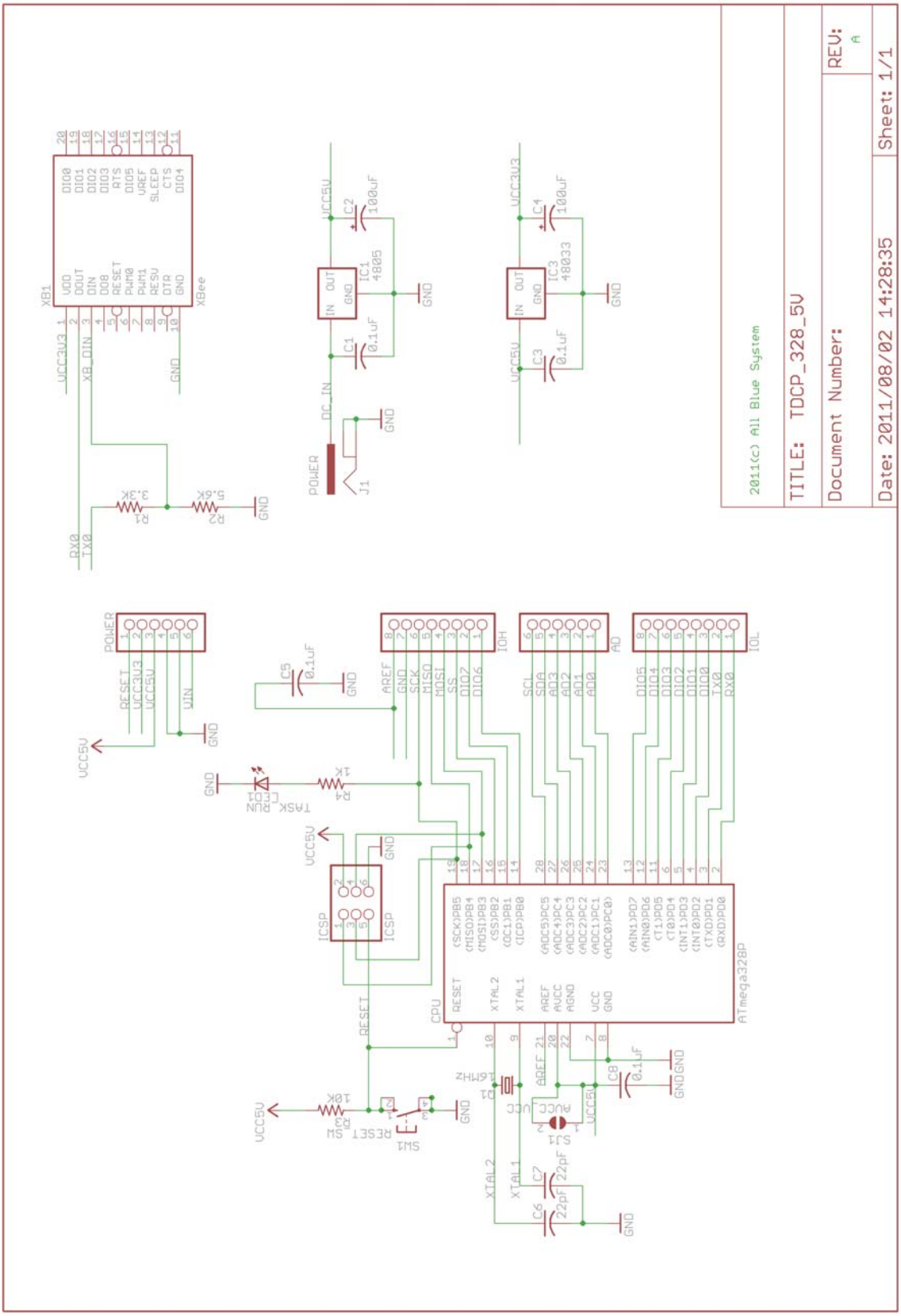
TDCP の SPI 機能は、 マイクロプロセッサ内部の SPIモジュール機能を使用して、データ通信速度やSPI 動作モード、送信オーダなどは “spi_config” コマンドで設定できます。

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。

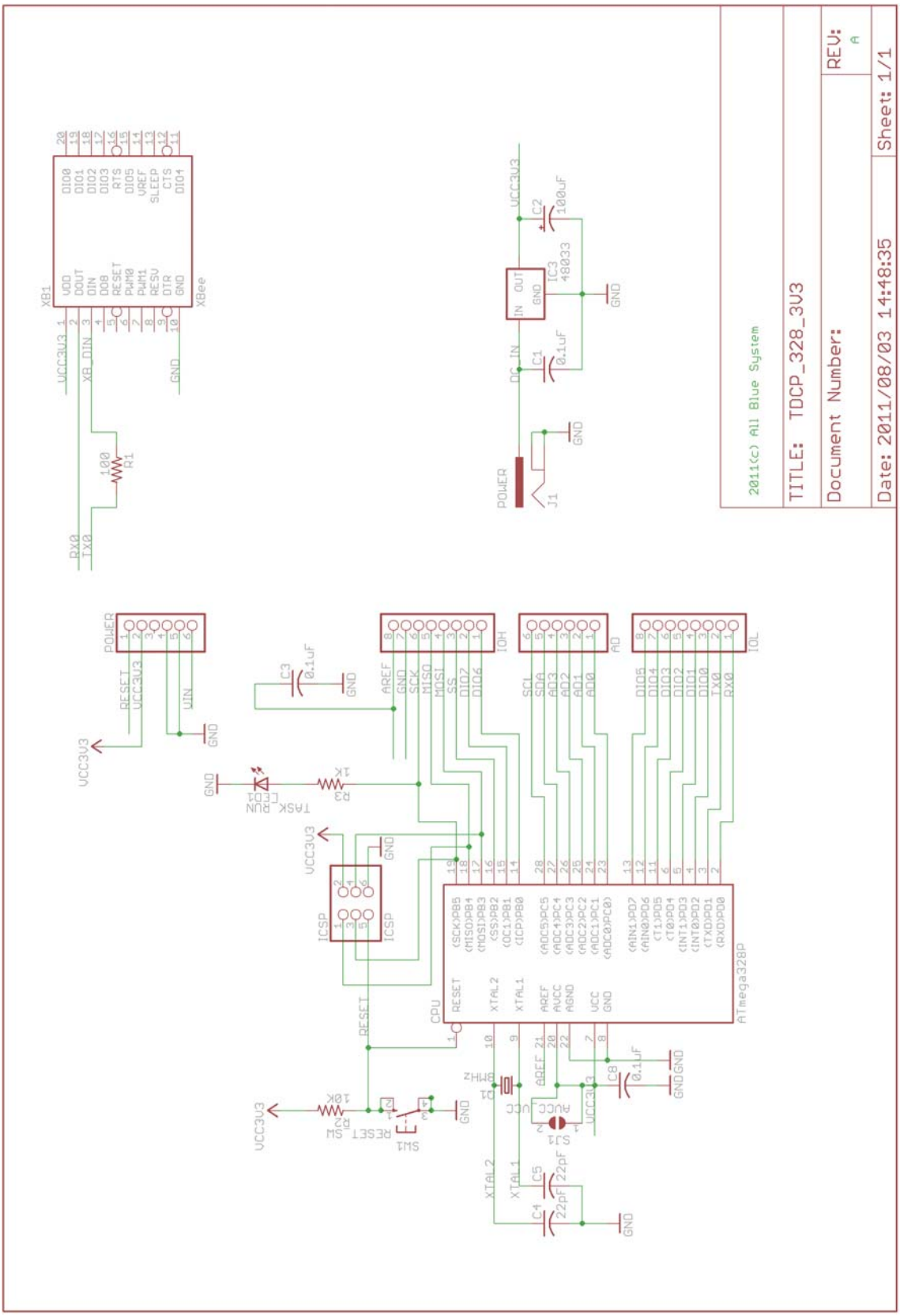
コマンドの機能詳細は、それぞれのコマンドリファレンスを参照して下さい。

11 TDCP 動作テスト用回路図

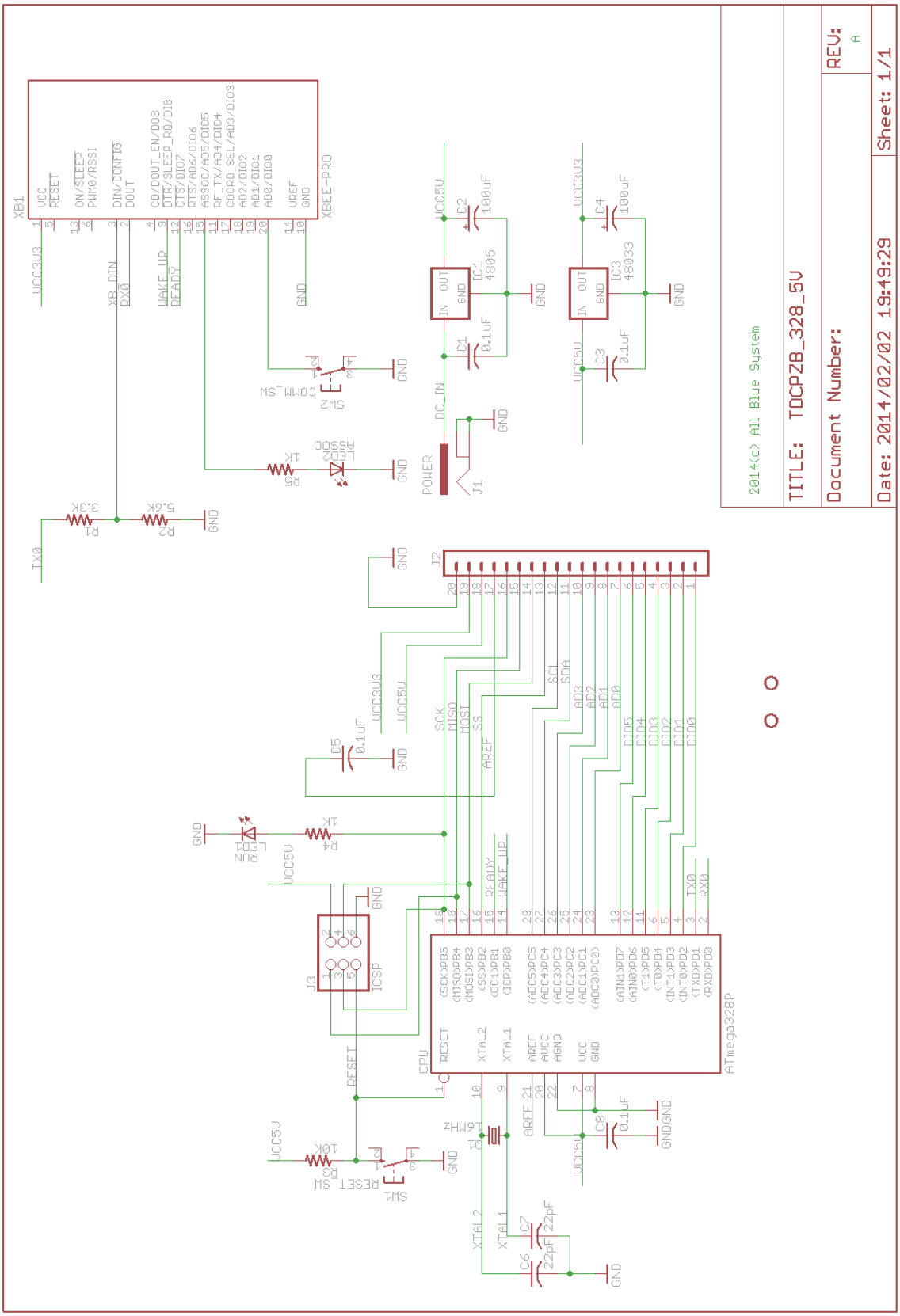
11.1 TDCP XBee 802.15.4 (Series1) 16MHz 5V 動作用回路図



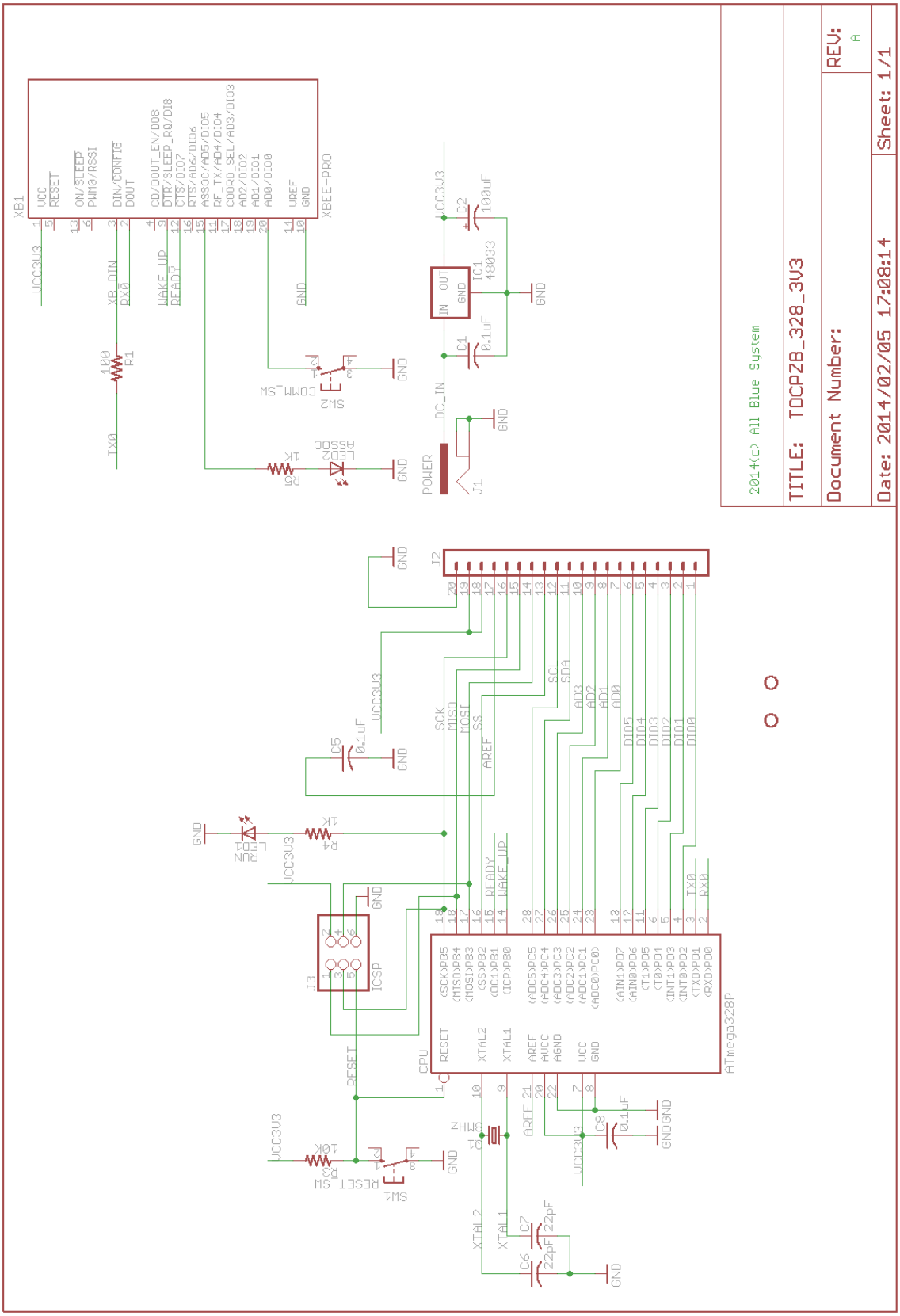
11.2 TDCP XBee 802.15.4 (Series1) 8MHz 3.3V 動作用回路図



11.3 TDCPZB XBee-ZB (Series2) 16MHz 5V 動作用回路図



11.4 TDCPZB XBee-ZB (Series2) 8MHz 3.3V 動作用回路図



12 Arduino 互換ボード(Arduino FIO) へのTDCPインストール

この章では TDCP , TDCPZP モニタプログラムを XBee I/F 付きの Arduino FIO ボードにTDCP プログラムインストールする手順を説明します。他の Arduino 互換ボードにインストールする場合もファームウエア書き込み時の Fuse bits の設定が違うだけで手順は同一です。

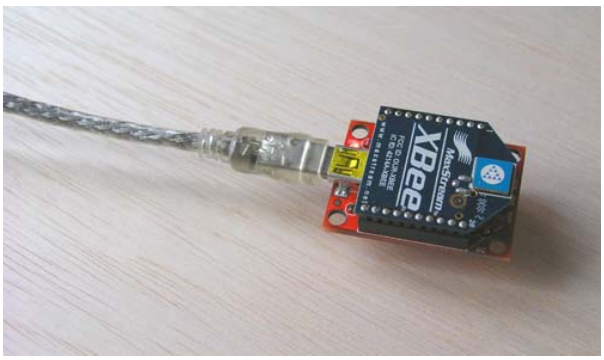
必要な部品

- Arduino FIO x1
- Arduino FIO 用バッテリまたは、USB ケーブル(給電用に使用)
- 3x2 ピンヘッダ(6pin ISP用) x1
- XBee 802.15.4 Series1 RF Module x1 または XBee-ZB Series2 RF Module x1
  サーバー側 PC の XBee をセットアップする場合には、もう一台同じタイプの XBee デバイスが必要です。
- XBee Explorer と PC接続用 USB ケーブル x1
  使用する全てのXBee デバイスの初期設定に使用した後、サーバー側 PC の XBee 接続に継続的に使用します
- ICSP 用 シリアルプログラムライタ x1 (ATmega328P ファームウエア書き込み用)

12.1 TDCP XBee 802.15.4 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。

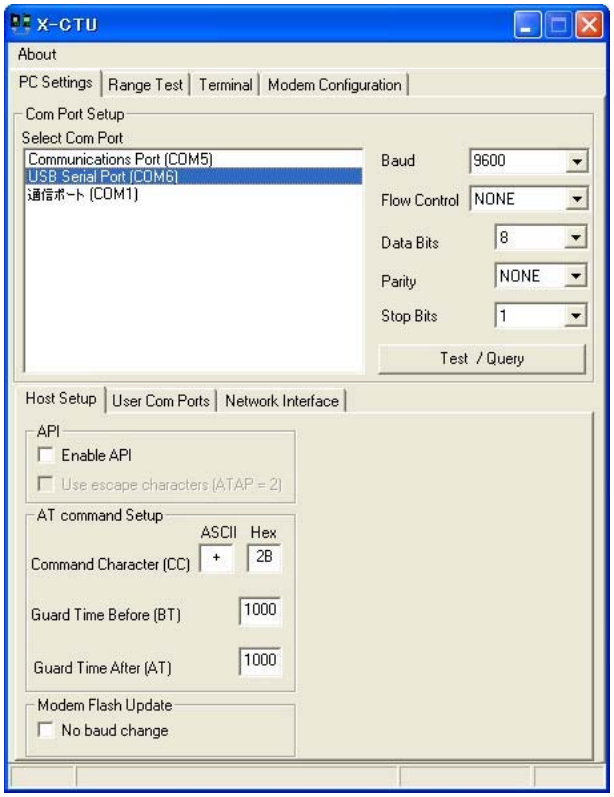
最初に、使用するXBee モジュールを XBee Explorer のソケットに搭載します。XBee Explorer を USB ケーブルで PC に接続すると、仮想 COM ポート経由で XBee デバイスにアクセスすることが出来るようになります。このとき、PC のコントロールパネルから 仮想COM ポート名を確認しておいてください。



XBee 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

XBee デバイス初期設定のコマンドを送信するために、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。仮想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。

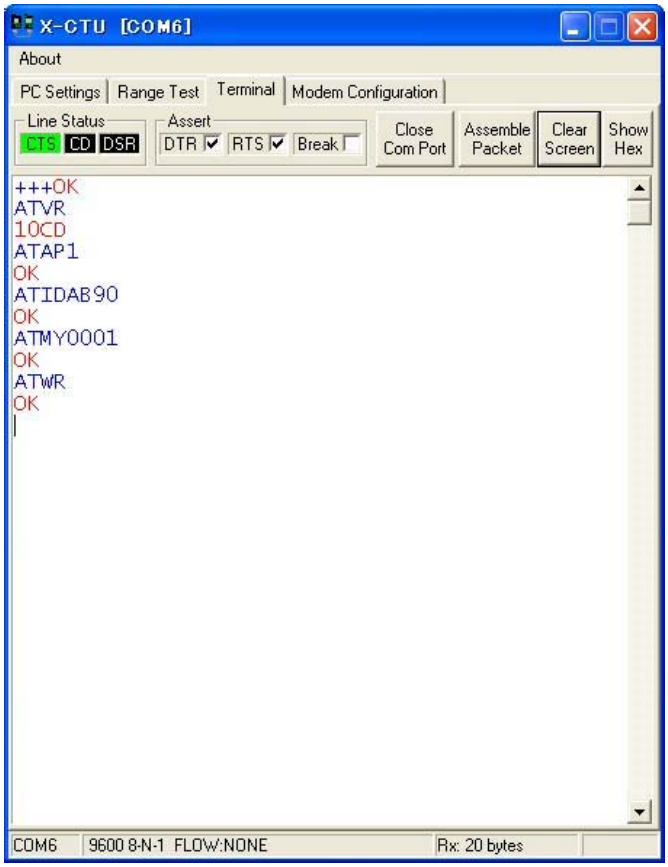


Terminal タブを選択してターミナル画面を表示します。キーボードから、"+++" を入力して、コマンドモードに入ります。コマンドモードに入ると "OK" が表示されますので、続けて以下のコマンド文字列を入力してください。コマンド入力の時間がかかりすぎると、自動的にコマンドモードから抜けてしまいますので、その場合は、"+++" を入力して最初からコマンドを入力し直して下さい。

```
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。

サーバーPC 用の XBee デバイスを使用する場合や、複数の Arduino FIO を使用する場合には、同様の手順で全ての XBee デバイスの初期設定を行ってください。XBee Explorerを PC から外した後、XBee デバイスを入れ替えて設定します。

XBee デバイスに設定する PAN ID は必ず全ての XBee で同一の値を指定して、16 bit Source Address は全てのデバイスで違う値を指定するようにしてください。

12.2 TDCPZB XBee-ZB ZigBee対応 Module Series2 初期設定

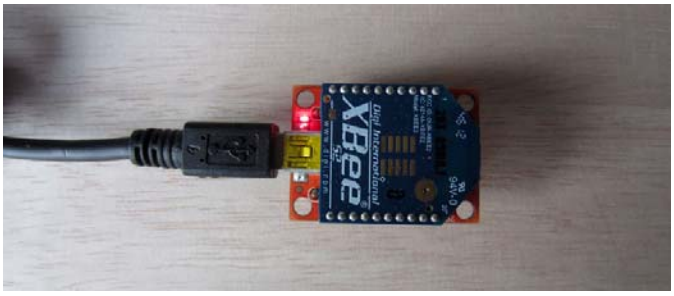
XBee-ZB デバイスの初期設定する項目は以下の内容です。

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 (for ATmega328P) では Router と End deviceタイプが使用できますが、ここではRouter タイプを選択して書き込む例を説明します。

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 または ZIGBEE END DEVICE API を使用してください。
API モード	1 ZIGBEE ROUTER API または ZIGBEE END DEVICE 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
Sleep モード (XBee-ZB ファームウエアに END DEVICE タイプを選択した場合に設定します)	5 ファームウエアに ZIGBEE END DEVICE API を使用するときには Sleepモードを "Cyclic Sleep with pin wake" に設定します。これによってTDCPZB プログラムからデータ送信するときに、XBee-ZB デバイスに設定されたスリープパラメータに影響されないで、確実にデータを送信できます。
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。

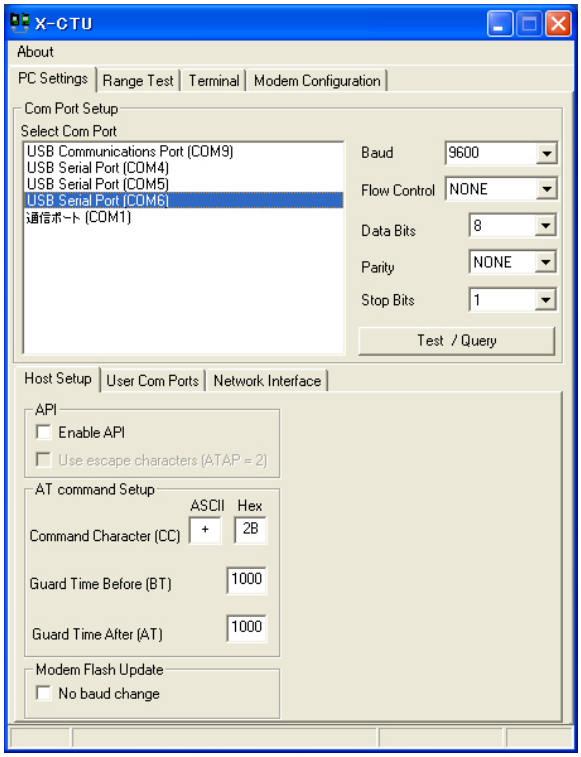
初期設定のコマンドを XBee-ZB に送信するために、XBee-ZB デバイスを PC のCOM ポートに接続して Digi international Inc. 社製の X-CTU プログラムを使用します。XBee-ZB と COM ポートのボーレートは初期値の 9600bps を使用します。Sparkfun Electoronics 社製の XBee Explorer USB などを使用して、仮想 USB ポート経由で接続して設定してください。(これ以外の方法で COM ポート接続する場合も手順は同じです)このマニュアルでは既に、XBee Explorer USBのドライバ等がインストール済みで仮想 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

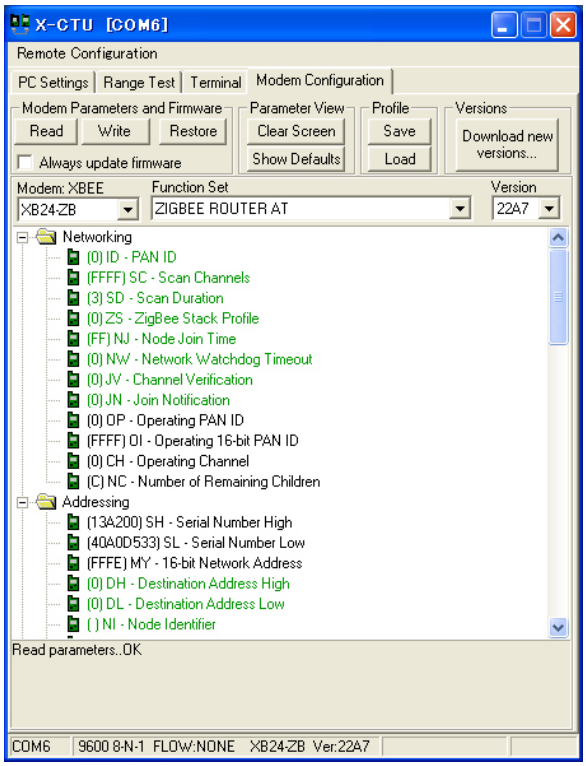
X-CTU プログラムを起動して、COM ポートを選択します。ここでは、USB Serial Port(COM6) を選択しています。

"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)

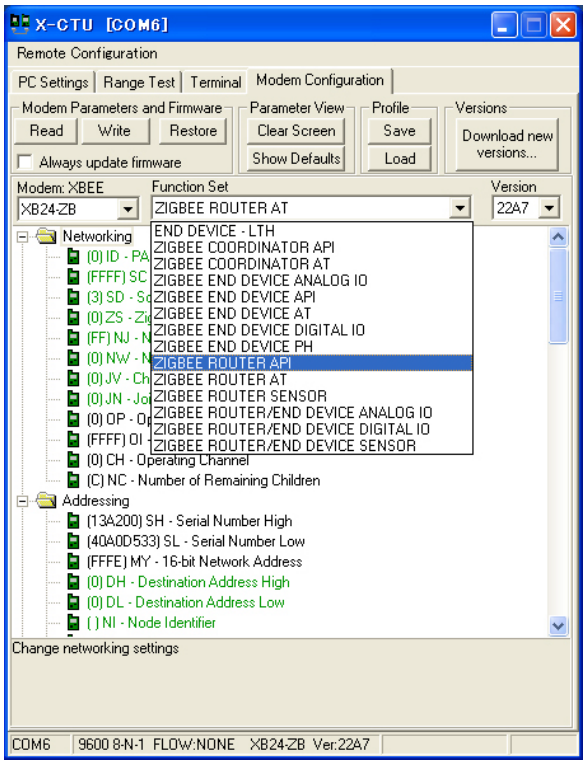
"Modem Configuration" タブを選択して、現在の XBee-ZB デバイスの設定をリードします。これは COM ポートの通信が正常に行えるかの確認も兼ねています。



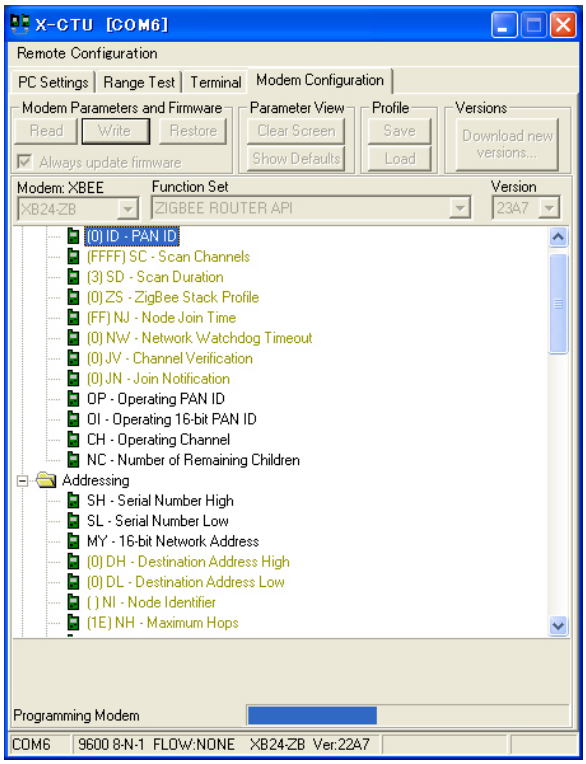
(XBee-ZB デバイスの工場出荷時のデフォルト設定を読み込んだ様子)

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。

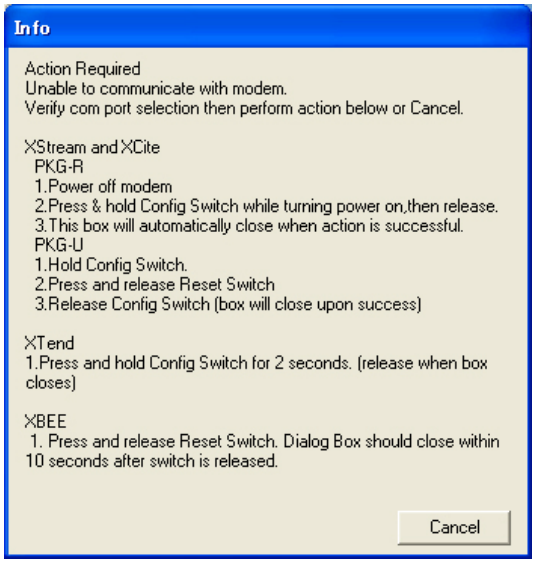


"Always update firmware" チェックボックスにチェックをつけて、"Write" ボタンを押してファームウエアの書き込みをします。

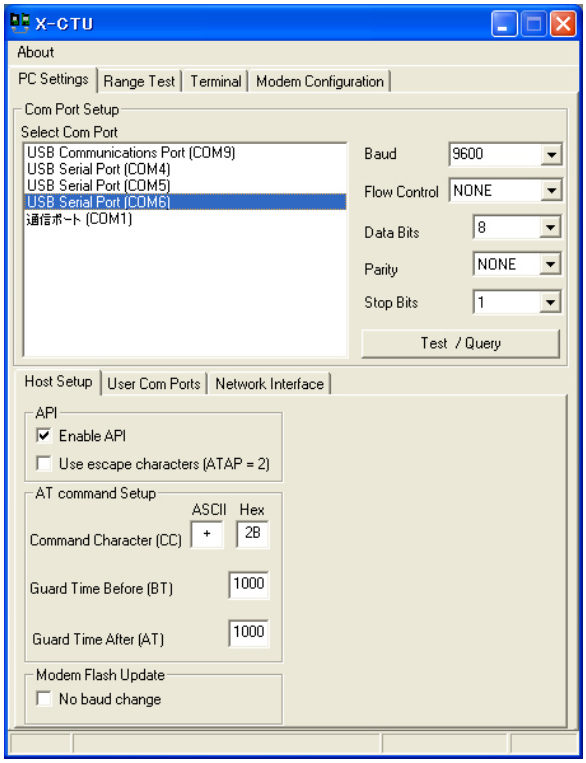


（ファームエア書き込み中の X-CTU の画面）

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。

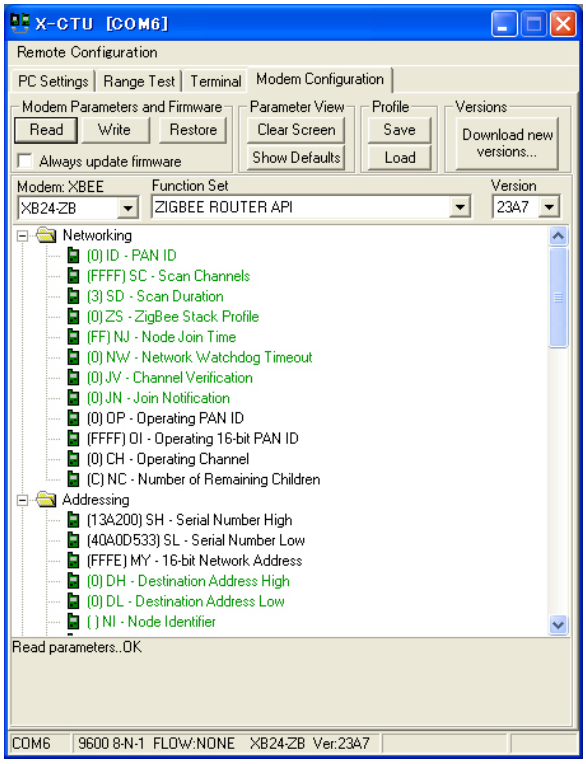


これは、ファームエアが API モード用のものに変わったために、最初にX-CTU プログラムの通信条件設定画面で "Enable API" のチェックを外していた状態と矛盾するようになったのが原因ですので問題ありません。ここでは "Cancel" ボタンを押してダイアログを消してください。ここで 一旦 X-CTU プログラムを終了して、もう一度同じ X-CTU プログラムを起動してください。



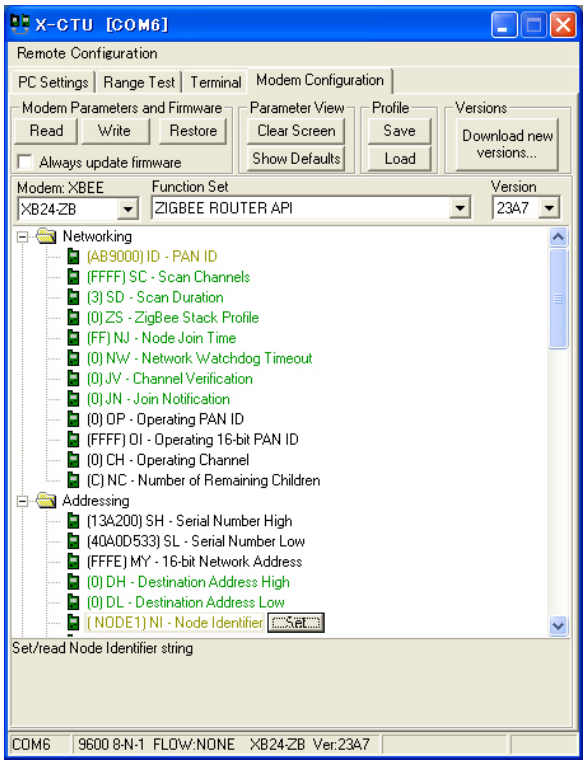
XBee-ZB が接続されたCOM ポートを選択して、今度は "Enable API" にチェックを付けます。

"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。



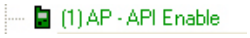
ここから、XBee-ZB のパラメータを設定します。最初に "PAN ID" 項目を選択して、任意の 最大64bit幅の 16進数値を設定します。ここでは "0xAB9000" を設定しています。PAN ID は全ての XBee-ZB デバイスで同一の値を設定してください。次に "Node Identifier" に任意のアルファベット文字列を設定します。ここでは "NODE3" を設定しま

す。XBee-ZB デバイス毎に異なった名前をつけてください。

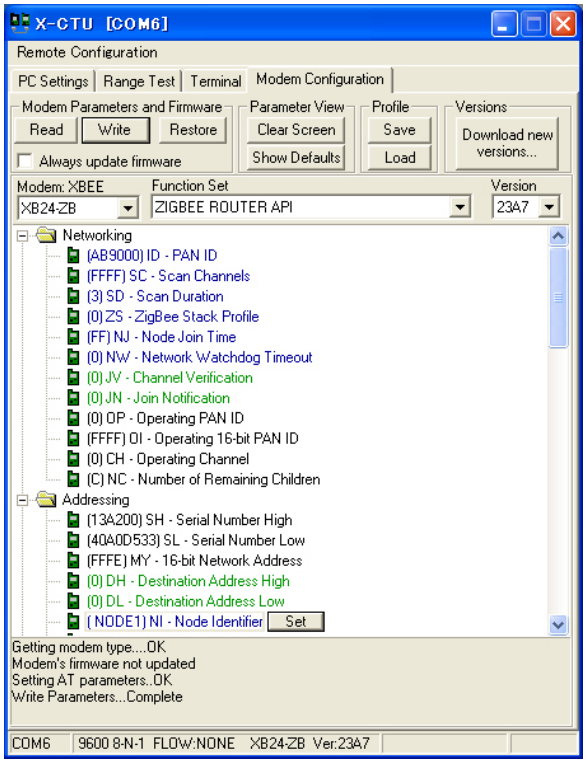


(PAN ID と Node Identifier を設定している様子、画面では "NODE1" が表示されていますが、このマニュアルでは "NODE3" に設定します)

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。

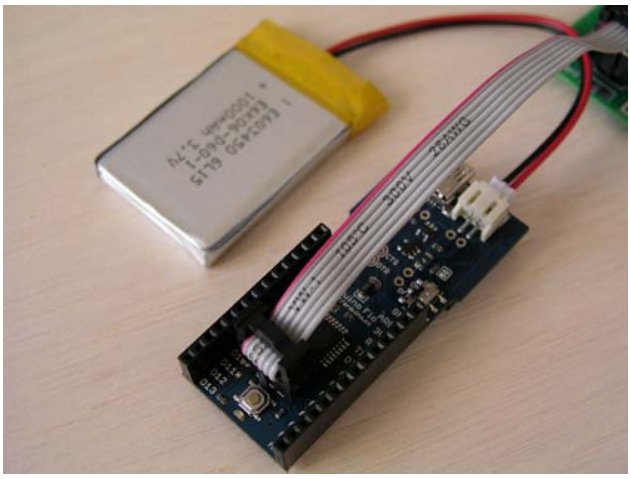
X-CTU で設定する Node Identifier 文字列について

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。

12.3 TDCP プログラム書き込み

殆どの Arduino互換機には ISP 6ピンヘッダを搭載するための配線がされています。ピンヘッダ自身は装備されていない場合がありますので、市販のピンヘッダを半田付けしてください。

TDCP プログラムは、Arduino FIO に電源を接続（USB ケーブル経由で給電またはバッテリを接続）して、POWER SW が ON の状態で、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 ピンヘッダに接続します。この時、ヘッダピンへのISPケーブルの接続方向が正しくなっていることを必ず確認して下さい。



ISP ケーブルを Arduino FIO の ICSP 6ピンヘッダに接続した状態

プログラム書き込み方法については、使用す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。

TDCP プログラムのアーカイブファイルには、XBee Series1 で使用する TDCPと XBee-ZB Series2で使用する TDCPZB の２種類のファームウエアに分かれています。それぞれ、CPU クロックが 8MHz 用と 16MHz 用の合計4つの HEX ファイルが入っています。使用するXBee デバイスタイプと CPU ボードのクロックに合わせてファイルを選択して下さい。

シリアルプログラムライタで書き込む時に指定する ATmega328P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。

⚠ 注意

TDCP プログラムを ISPシリアルプログラムライタでArduino ボードに書き込んだ場合には、Arduino ブートローダプログラムは削除されます。もし、Arduino ブートローダプログラムを再び Arduino ボードに導入する場合には、ISPシリアルプログラムライタを使用して書き込む必要があります。Arduino ブートローダプログラムファイルと Fuse Bits 設定値は Arduino IDE に同梱されているファイルやドキュメントを参照して下さい。

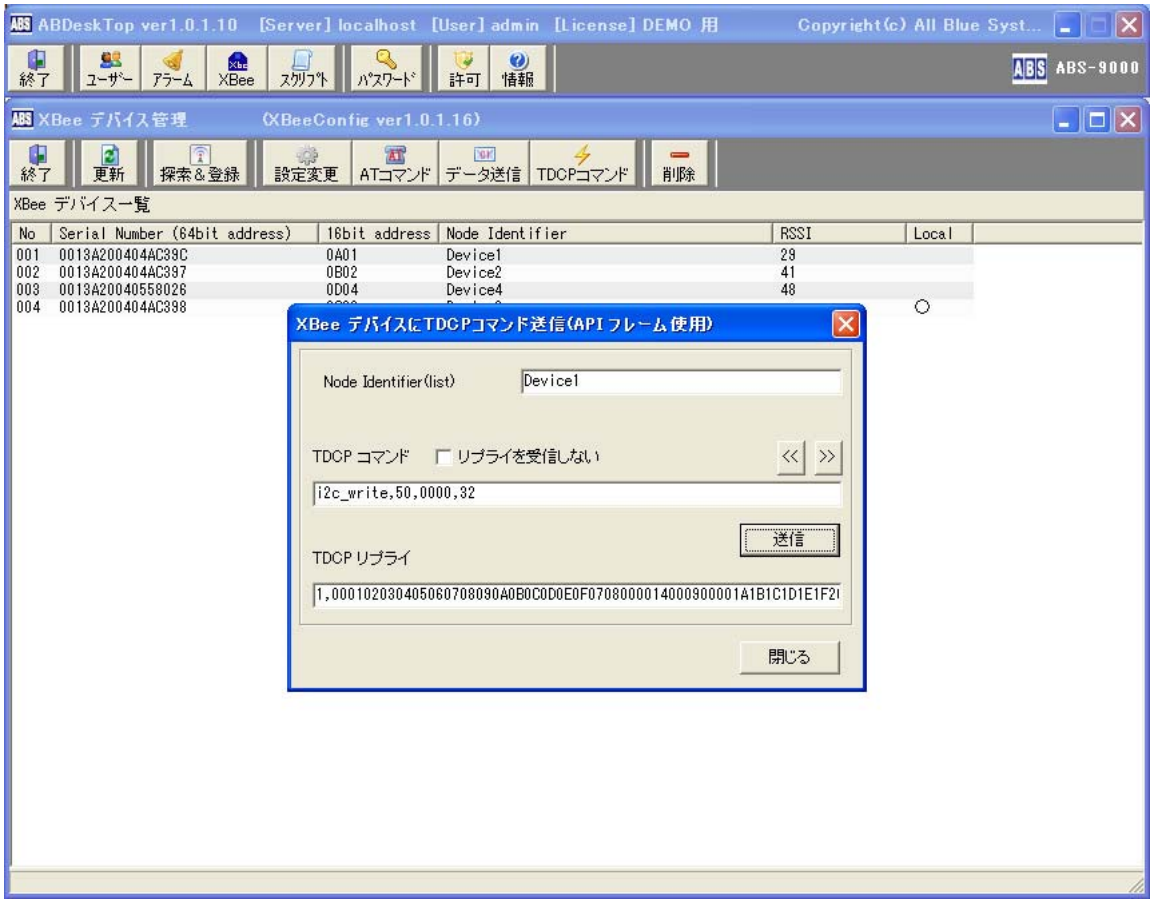
12.4 動作確認

TDCP が正常に動作すると、Arduino ボード上の D13(PORTB bit5) または TDCP ピン配置図中の SPI SCK に接続された LED が点灯します。この状態で他の XBee デバイスからのパケット経由で TDCP コマンドを実行することができます。ここでは TDCP (XBee 802.15.4 Series1) プログラムの動作確認を行う例で説明します。

XBee デバイスにデータパケットを送信するときに、オールブルーシステムの DeviceServer を使用する場合には、XBeeConfig クライアントプログラムから “TDCP コマンド送信” を選択することで簡単にパケットの送信と、リプライパケットを受信することができます。DeviceServer の XbeeConfig やスクリプトプログラム中では、TDCPPrefix 文字列 “$$$xxxxx” の処理を自動的に行いますので、ユーザーはコマンド文字列とリプライ文字列のみを扱います。

DeviceServer を利用しなくても、プログラム中から XBee APIフレーム (API Identifier 0x00 または 0x01 のデー

タパケット）を作成して、RF Data 中に TDCP リクエストコマンド文字列を組み込んで送信することができます。この場合には、コマンドを受信した TDCP デバイスはコマンド実行後、リクエスト送信元の XBee デバイスに対してリプライパケットを XBee API フレームで送信しますので、プログラム中から API フレームを受信して処理を行ってください。



オールブルーシステムの DeviceServer を利用して、TDCP コマンドを Arduino FIO(TDCPプログラムインストール済)デバイスに送信して、リプライパケットを受信した例

動作例として、下記のコマンドを実行します。(”リクエストパケット文字列” -> “リプライパケット文字列”)

```
// DIO#0, DIO#1, DIO#2を出力モードに設定
“$$$123,dio_config,07” -> “$$$123,1”
// DIO の入力ポートにアサインされている全てのビットの内部プルアップ有効
$$$123,pullup,FF -> $$$123,1
// ADC リファレンス電圧をATmega328プロセッサ内部の 1.1V リファレンス電源に設定
$$$123,adc_vref,3 -> $$$123,1
// コンフィギュレーションをプロセッサ内部の EEPROM に保存
$$$123,config_save -> $$$123,1
// プロセッサリセット、DIO 入出力モード変更内容を有効にする
```

```
$$$,reset
// DIO#0 を High にする
$$$123,port_bit,0,1 -> $$$123,1
// DIO#1 に 100ms 幅のパルス信号を出力する
$$$123,pulse,1,0,100-> $$$123,1
```

13 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,

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.

14　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。

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

オールブルーシステムの 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。

15　更新履歴

REV A.2.7 2015/8/3

TDCPZB の I2Cバスに 気圧センサ(LPS331),温湿度センサ(AM2321) を I2Cバスで接続した状態で使用可能な専用のアプリケーションモード(app_mode 34) を追加した。このモードを使用するとリモート側からI2C操作を行わなくても、最新のセンサ値が SAMPLINGイベントに含まれるようになります。

ファームウエア書き込み時の Fuse bit 設定値が間違っていたのを修正した

serial_number, server_addr, tx_data, tx_ascii(tx) コマンドを TDCPZB から除いた

REV A.2.5 2014/4/27

dio_read(port_read)コマンドが現在のポート値と "dio_config" の設定値の２つを返すようにした

REV A.2.4 2014/4/15

i2c_write2 コマンドの複数トランザクション間のウェイト時間を 10ms から 20ms に変更した

REV A.2.3 2014/4/12

I2C slave 機能の TDCP動作モード取得コマンドの記述を修正した

REV A.2.2 2014/3/28

TDCPコマンド文字列長の制限事項を記述した

REV A.2.1 2014/3/25

i2c_write2 コマンドを新規追加した

REV A.2.0 2014/2/5

dio_read,dio_write,dio_pullup,dio_bit,ad_read,ad_vref コマンドエイリアスを作成した

REV A.1.5 2013/1/24

イベントが連続して発生する場合の注意事項を追記した

REV A.1.4 2012/8/15

ATmega328P ピン配置図を追加した

REV A.1.3 2012/7/31

“pulse” コマンドに連続出力モードフラグ〈continuous〉を追加した。

REV A.1.2 2012/7/27

COUNTER_UPDATE イベント送信機能と counter_margin コマンドを追加

REV A.1.1 2012/7/25

リセット時にEEPROM 内容を強制的に初期化する機能を追加

REV A.1.0 2011/8/7

初版作成