REX-USB61mk2

USB-SPI/I2C Protocol Emulator

High-Grade Model

ユーザーズマニュアル

2021 年 11 月

第 5.0 版

ラトックシステム株式会社

安全にご使用いただくために

第1章 はじめに ---------- 1- 1
(1-1) 製品仕様 1- 1
(1-2) 梱包内容の確認 1- 3
(1-3) ケーブル仕様 1- 4
(1-4) 各モードについて 1- 5
(1-5) SPI デバイスとの接続例について 1- 6
(1-6) I2C デバイスとの接続例について 1- 9

第 2 章 セットアップと UnitID の登録---------- 2- 1
(2-1) Windows でのセットアップ 2- 1
(2-2) REX-USB61mk2 インストールの確認 2- 3
(2-3) Windows でのアンインストール方法 2- 4
(2-4) UnitID の登録方法 2- 5
(2-5) 登録 UnitID の確認 2- 5

第 3 章 スクリプト制御について---------- 3- 1
(3-1) スクリプト実行アプリケーションについて 3- 1
(3-2) スクリプト仕様 3- 5

第4章 API 関数仕様とサンプルプログラム---------- 4- 1
(4-1) VC での使用について 4- 1
(4-2) VB/Visual C#での使用について 4- 1
(4-3) API 関数一覧 4- 7
(4-4) API 関数詳細 4-10
(4-5) サンプルアプリケーションについて 4-48

第 5 章 仮想 COM ポートでのコマンド制御---------- 5- 1
(5-1) 制御コマンドについて 5- 1
(5-2) 制御コマンド仕様と使用例について 5- 3

安全にご使用いただくために

本製品は安全に充分配慮して設計を行っていますが、誤った使い方をすると火災や感電などの事故につながり大変危険です。ご使用の際は、警告/注意事項を必ず守ってください。

表示について

この取扱説明書は、次のような表示をしています。表示の内容をよく理解してから本文をお読みください。

⚠警告	この表示を無視して誤った取扱いをすると、火災や感電などにより、人が死亡または重傷を負う可能性がある内容を示しています。
⚠注意	この表示を無視して誤った取扱いをすると、感電やその他の事故により、人が負傷または物的損害が発生する可能性がある内容を示しています。

⚠警告

- 製品の分解や改造などは、絶対に行わないでください。
- 無理に曲げる、落とす、傷つける、上に重い物を載せることは行わないでください。
- 製品が水・薬品・油などの液体によって濡れた場合、ショートによる火災や感電の恐れがあるため使用しないでください。

⚠注意

- 本製品は電子機器ですので、静電気を与えないでください。
- 高温多湿の場所、温度差の激しい場所、チリやほこりの多い場所、振動や衝撃の加わる場所、スピーカなどの磁気を帯びた物の近くで保管しないでください。
- 煙が出たり異臭がする場合は、直ちにパソコンや周辺機器の電源を切り、USBケーブルをPCから抜いてください。
- 本製品は、医療機器、原子力機器、航空宇宙機器、輸送機器など人命に関わる設備や機器、及び高度な信頼性を必要とする設備や機器での使用は意図されておりません。これらの設備、機器制御システムに本製品を使用し、本製品の故障により人身事故/火災事故/その他の障害が発生した場合、いかなる責任も負いかねます。
- 取り付け時、鋭い部分で手を切らないように、十分注意して作業を行ってください。
- 配線を誤ったことによる損失、逸失利益などが発生した場合でも、いかなる責任も負いかねます。

その他のご注意

- 本書の内容に関して、将来予告なしに変更することがあります。
- 本書の内容につきましては万全を期して作成しておりますが、万一不審な点や誤りなどお気づきになりましたらご連絡お願い申し上げます。
- 本製品の運用を理由とする損失、逸失利益などの請求につきましては、いかなる責任も負いかねますので、予めご了承ください。
- 製品改良のため、将来予告なく外観または仕様の一部を変更する場合があります。
- 本製品は日本国内仕様となっており、海外での保守及びサポートは行っておりません。
- 本製品を廃棄するときは地方自治体の条例に従ってください。条例の内容については各地方自治体にお問い合わせください。
- 本製品の保証や修理に関しましては、添付の保証書に内容を明記しております。必ず内容をご確認の上、大切に保管してください。
- “REX”は株式会社リコーが商標権を所有しておりますが、弊社はその使用許諾契約により本商標の使用が認められています。
- Windowsは米国マイクロソフト社の米国およびその他の国における登録商標です。その他本書に記載されている商品名/社名などは、各社の商標または登録商標です。 なお本書では、TM、Ⓡ マークは明記しておりません。

第1章 はじめに

(1-1) 製品仕様

REX-USB61mk2 を利用することによって SPI または I2C の EEPROM や Flash ROM への書き込みや、各種センサーの初期動作確認などが容易に行うことができます。

[スクリプト制御ユーティリティを提供]

スクリプト制御ユーティリティでは、SPI/I2C および DIO(ポート入出力)の制御をビット演算や分岐機能のあるスクリプトで行うことができます。
(詳細につきましては第 3 章をご参照ください。)

[仮想 COM モードを搭載]

ターミナルソフトから簡単なコマンドで SPI/I2C デバイスにアクセスすることができ、小規模なメモリーデバイスの読み書きや、デバイスへのアクセステストに最適です。(詳細につきましては第 5 章をご参照ください。)

[API ライブラリとサンプルを提供]

API ライブラリを利用したアプリケーションソフトを作成することにより、
使用環境に合わせた制御が可能となります。
また、API ライブラリを呼び出すためのプログラムソースコードも提供しております。(詳細につきましては第 4 章をご参照ください。)

[複数台の接続が可能]

1 台の WindowsPC に本製品を最大で 4 台まで接続することが可能です。

[ファームウェアアップデートプログラムを提供]

本製品は将来的な仕様の追加や変更に対応できる様に、装置のファームウェアを更新することができます。最新ファームウェアおよびアップデートプログラムは弊社ホームページよりダウンロードすることができます。

ハードウェア仕様

項　　目	仕　　様　　内　　容		
インターフェイス	USB2.0 High Speed Device		
接続コネクタ	USB micro B コネクタ		
電源電圧	5V (USB バスパワーから取得)		
消費電流	200mA(デバイスへの供給分は含まず)		
サポート インターフェイス	SPI	マスター	対応周波数：762Hz～50MHz (設定値以下の近似値)
		スレーブ (2ch)	対応周波数：762Hz～16.6MHz
	I2C	マスター	対応周波数：762Hz～5MHz (設定値以下の近似値) 100kHz/400kHz/1MHz/3.4MHz/5MHz
		スレーブ (2ch)	
外部入出力レベル	本製品からの電源供給：1.8V/2.5V/3.3V/5V 外部電源からの電源供給：1.8V～5.0V		
外形寸法	69(L)×115(W)×19.5(H) mm		
重量	約 98g (本体のみ)		
動作環境	温度:0～55℃ 湿度:20～80%(ただし結露しないこと)		

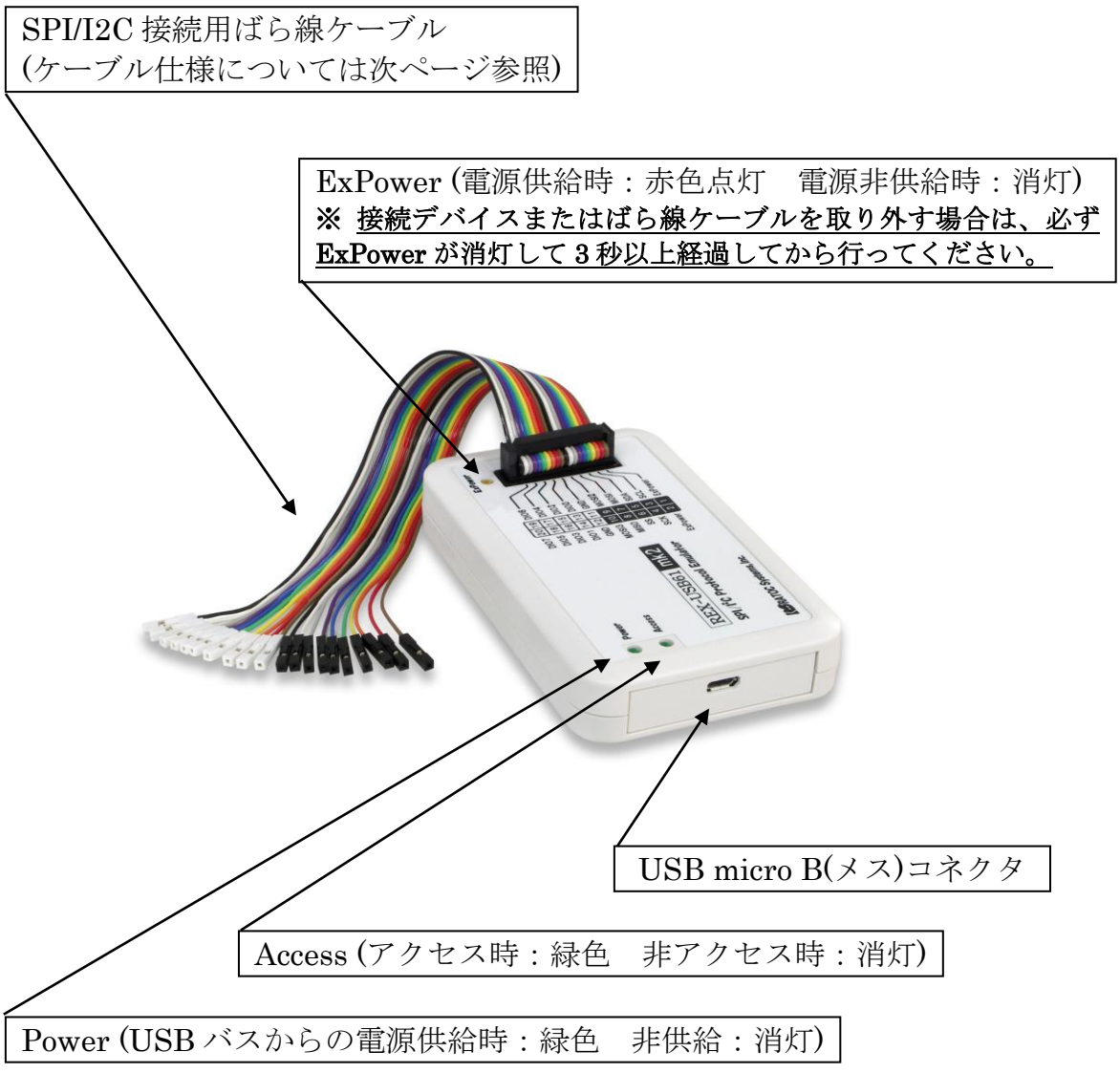
ソフトウェア仕様

項　　目	仕　　様　　内　　容
ドライバー	USB61mk2_Setup.exe
ユーティリティ	スクリプト制御ユーティリティ(USB61mk2_Script.exe)
個別 ID 割当てツール	UnitID セッティングツール(Usb61mk2Setting.exe)
個別 ID 確認ツール	UnitID 確認ツール(Usb61mk2View.exe)
サンプルプログラム	SPI/I2C マスター用 EEPROM Read/Write サンプル SPI/I2C スレーブ用サンプル DIO 入出力/割り込みサンプル (VC++2010/2005/6.0、VB2010/2005/6.0、VC# 2010/2005)
ライブラリ	SPI/I2C 制御用ライブラリ(RexUsb61mk2.dll/REXUSB61MK2x64.dll) VC 用リンクファイル(RexUsb61mk2.lib/REXUSB61MK2x64.lib) VC 用ヘッダファイル (RexUsb61mk2.h/RexUsb61mk2x64lib.h) VB6 用標準モジュール (RexUsb61mk2.bas) VB.net 用コードファイル (RexUsb61mk2.vb/RexUsb61mk2x64.vb) VB.net/C#用 OCX(RexUsb61mk2.ocx) C#用コードファイル(RexUsb61mk2.cs/RexUsb61mk2x64.cs)
対応 OS	Windows 11/10/8.1/7/Vista 対応 (32-bit/64-bit 含む)

(1-2) 梱包内容の確認

ご使用前に添付品のご確認をお願いします。

- ☑ REX-USB61mk2 本体
- ☑ USB A – micro B ケーブル(約 1m)
- ☑ デバイス接続用 20pin ばら線ケーブル(約 15cm)
- ☑ 保証書



(1-3) ケーブル仕様

REX-USB61mk2 に添付されている、ターゲットデバイスとの接続用ケーブルの仕様を説明します。

ピン番号	ハウジング色	ケーブル色	信号名	用途
1	黒	茶	ExPower	ターゲットデバイス電源入出力 (出力時 1.8/2.5/3.3/5.0V @100mA) (入力時 1.8V ～ 5V)
2	黒	赤	ExPower	ターゲットデバイス電源入出力 (出力時 1.8/2.5/3.3/5.0V @100mA) (入力時 1.8V ～ 5V)
3	黒	橙	SCL	I2C 用クロック信号
4	黒	黄	SCK	SPI 用クロック信号
5	黒	緑	SDA	I2C 用データ信号
6	黒	青	SS	SPI 用スレーブセレクト(マスター時)
7	黒	紫	MOSI	SPI 用 MOSI(MOSI 0)
8	黒	灰	MISO	SPI 用 MISO(MOSI 1)
9	黒	白	MOSI 2	SPI 用 MOSI 2
１０	黒	黒	MOSI 3	SPI 用 MOSI 3
１１	白(灰)	茶	GND	グランド
１２	白(灰)	赤	GND	グランド
１３	白(灰)	橙	DIO 0	DIO ポート 0
				SPI スレーブ時：Ch1 の SS
１４	白(灰)	黄	DIO 1	DIO ポート 1
				SPI スレーブ時：Ch2 の SS
１５	白(灰)	緑	DIO 2	DIO ポート 2
				SPI スレーブ時：Ch1 の WP
１６	白(灰)	青	DIO 3	DIO ポート 3
				SPI スレーブ時：Ch2 の WP
１７	白(灰)	紫	DIO 4	DIO ポート 4
１８	白(灰)	灰	DIO 5	DIO ポート 5
１９	白(灰)	白	DIO 6	DIO ポート 6
２０	白(灰)	黒	DIO 7	DIO ポート 7

(1-4) 各モードについて

本製品の SPI /I2C・マスター/スレーブ動作について説明いたします。

バス	動作	
SPI バス	マスターモード	SCK/SS/MOSI/MISO 信号を使用し、スレーブデバイスに対し送受信を行う。 また、Dual モードの場合は MOSI/MISO を入出力信号として使用し、Quad モードの場合は MOSI/MISO/MOSI2/MOSI3 を入出力信号として使用する。
	スレーブモード	スレーブデバイスとして各種設定を行い、動作させることができる。 (詳細につきましては第 4 章の SPI スレーブサンプルをご参照ください。)
I2C バス	マスターモード	SCL/SDA 信号を使用し、スレーブデバイスに対し送受信を行う。
	スレーブモード	スレーブデバイスとして各種設定を行い、動作させることができる。 (詳細につきましては第 4 章の I2C スレーブサンプルをご参照ください。)

※　ExPower/GND もそれぞれデバイスと接続が必要です。

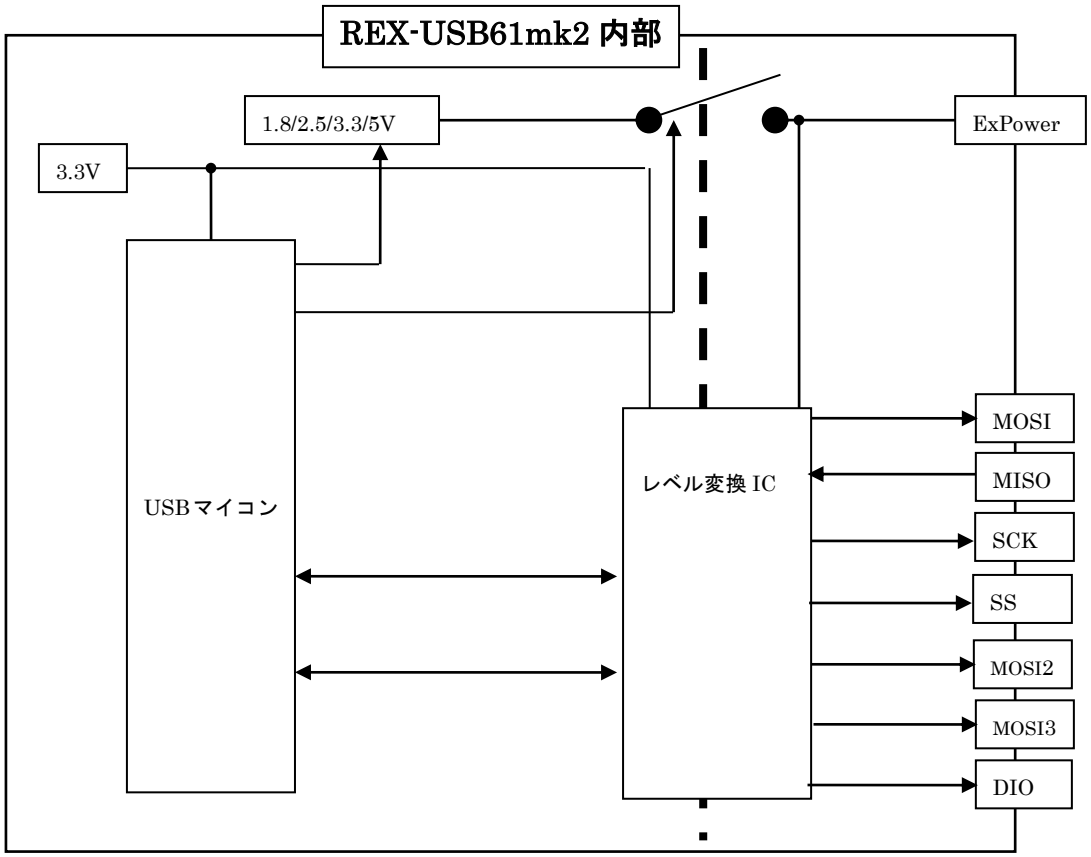
REX-USB61mk2 のマスターモード/スレーブモードは、
本製品用のアプリケーションまたはライブラリ関数により指定いたします。

(1-5) SPI デバイスとの接続例について

SPI のインターフェイスを持った EEPROM との接続例を以下に示します。

- **REX-USB61mk2 の電源部分について**

  REX-USB61mk2 内部のレベル変換 IC へ電源供給をおこなうため、ターゲットデバイスへの電源供給の有無にかかわらず、**必ず REX-USB61mk2 の ExPower ピンをターゲットデバイスの電源と接続してください。**

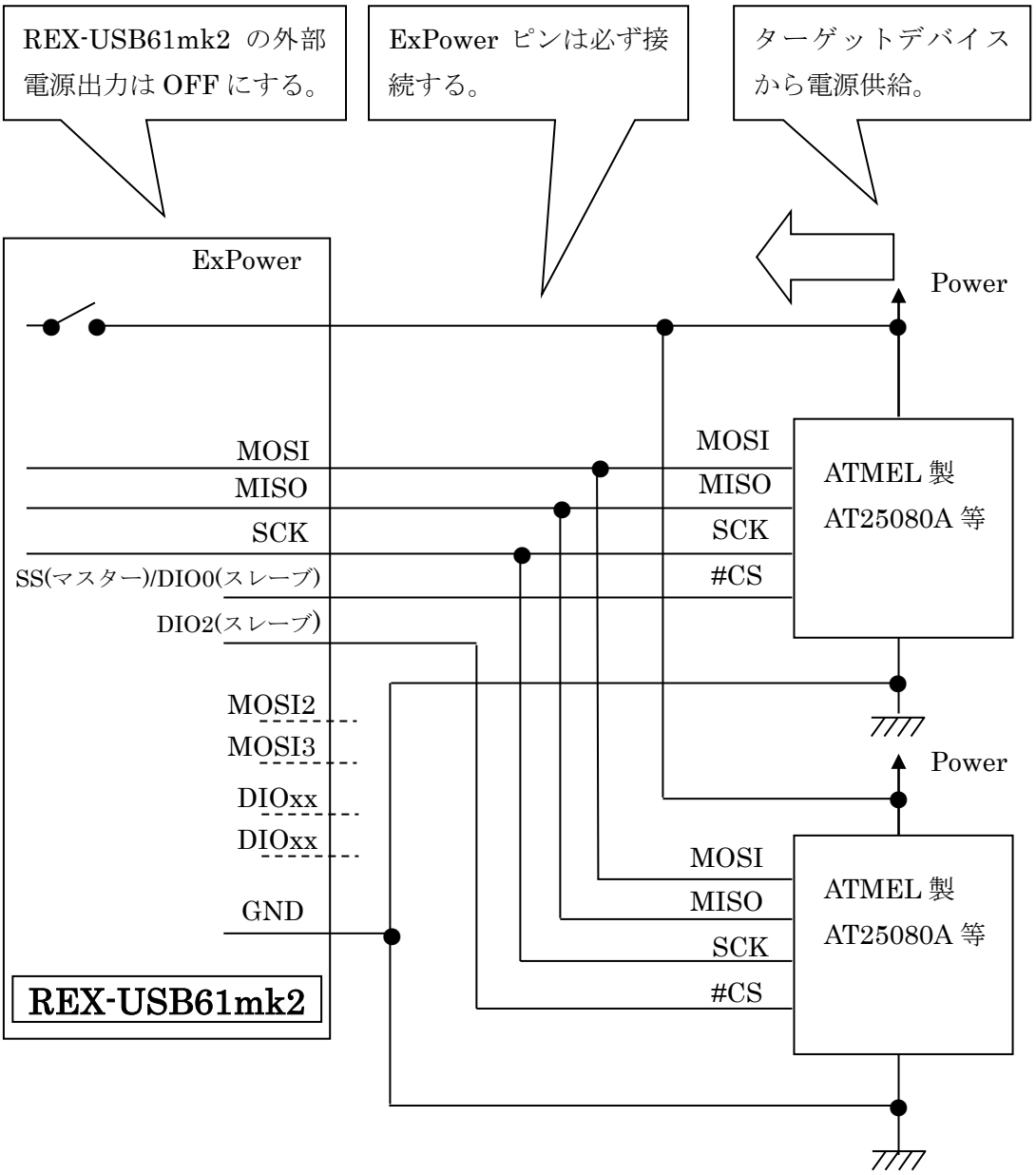


【注意】

接続デバイスの取り付け・取り外しは、製品本体の **ExPower** が消灯して 3 秒以上経過してから行ってください。
(本製品もしくは制御する接続デバイスに電源供給を行った状態で、本製品から接続デバイスの取り付け・取り外しを行うと、故障いたします。)

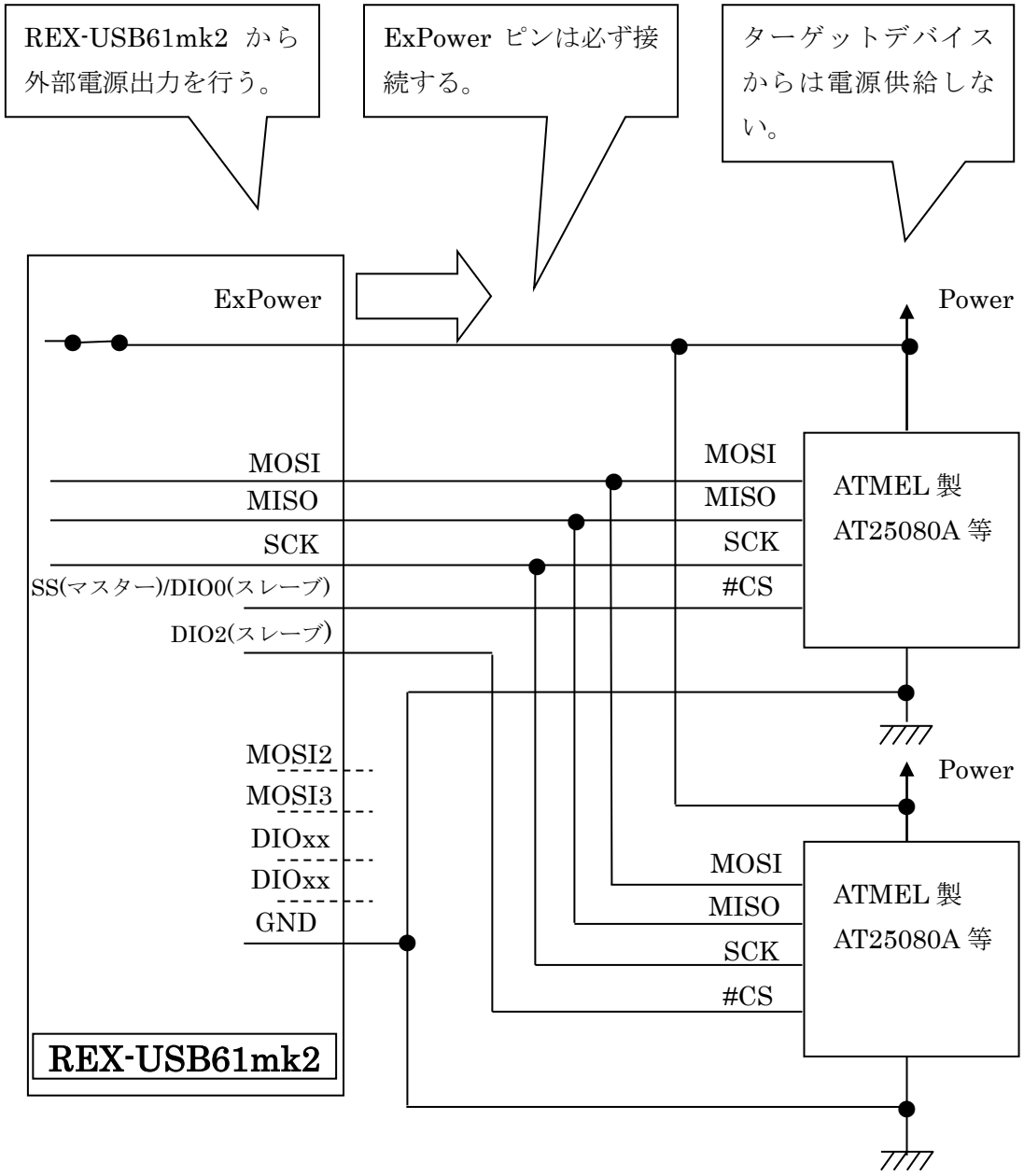
SPI 接続（ターゲットデバイスに電源がある場合）

ターゲットデバイスに電源がある場合は、ユーティリティソフトウェアまたはライブラリ関数を利用したアプリケーションにて電源供給を無効にしてください。
(該当するライブラリ関数は usb61mk2_power_control()となります。
参照：(4-4)API 関数詳細)



SPI 接続（ターゲットデバイスが電源を持たない場合）

本機からターゲットデバイス側へ電源供給(1.8/2.5/3.3/5.0V)する場合は、ユーティリティソフトウェアまたはライブラリ関数を利用したアプリケーションにておこないます。(供給電流は最大 100mA)
(該当するライブラリ関数は usb61mk2_power_control()となります。
参照：(4-4)API 関数詳細)

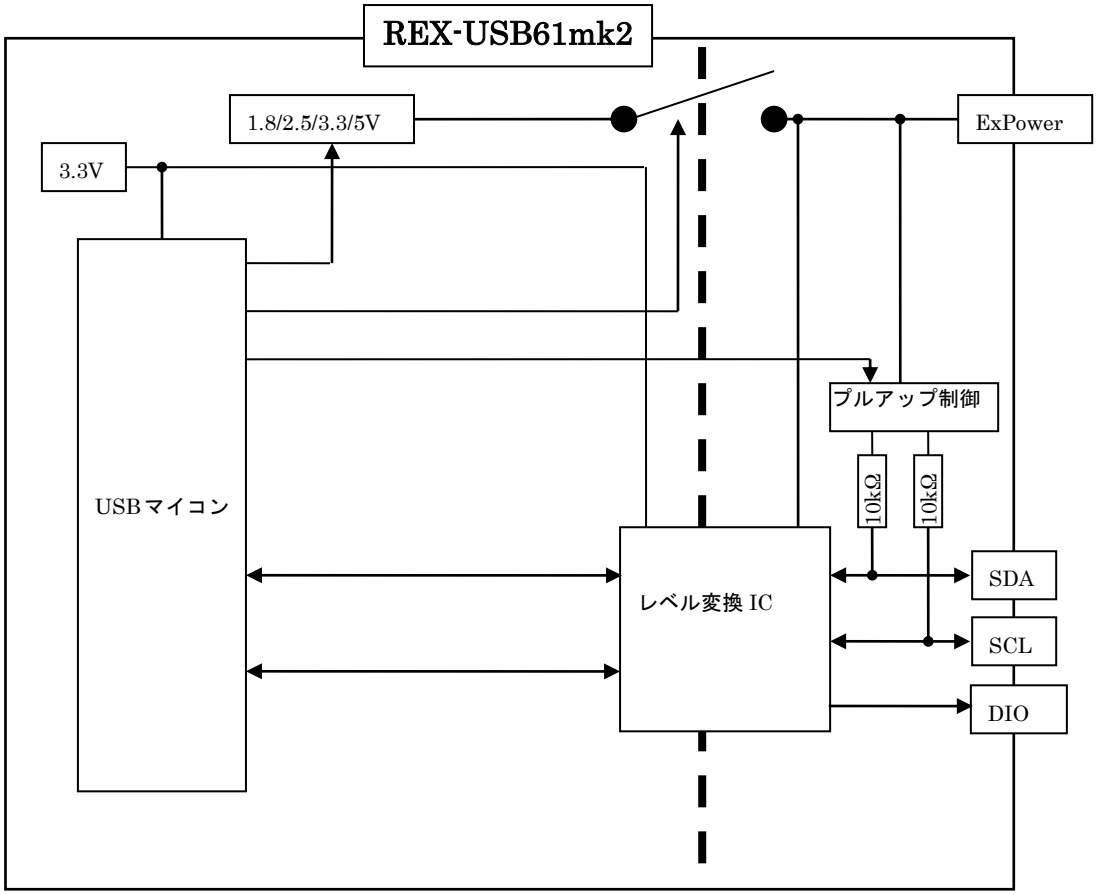


(1-6) I2C デバイスとの接続例について

I2C のインターフェイスを持った EEPROM との接続例を以下に示します。

- **REX-USB61mk2 の電源部分について**

  REX-USB61mk2 内部のレベル変換 IC へ電源供給をおこなうため、ターゲットデバイスへの電源供給の有無にかかわらず、**必ず REX-USB61mk2 の ExPower ピンをターゲットデバイスの電源と接続してください。**

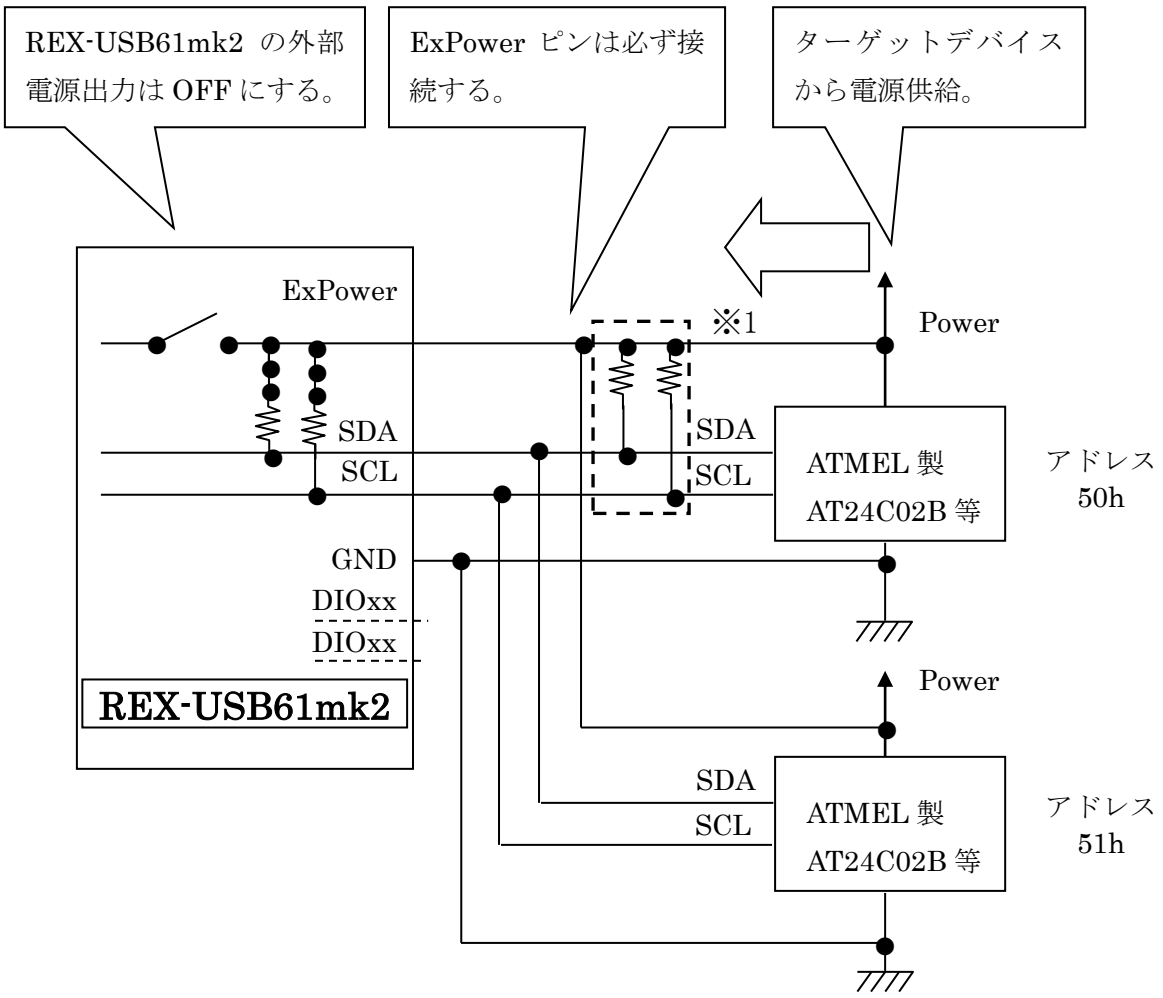


【注意】

接続デバイスの取り付け・取り外しは、製品本体の **ExPower** が消灯して 3 秒以上経過してから行ってください。
(本製品もしくは制御する接続デバイスに電源供給を行った状態で、本製品から接続デバイスの取り付け・取り外しを行うと、故障いたします。)

I2C 接続（ターゲットデバイスに電源がある場合）

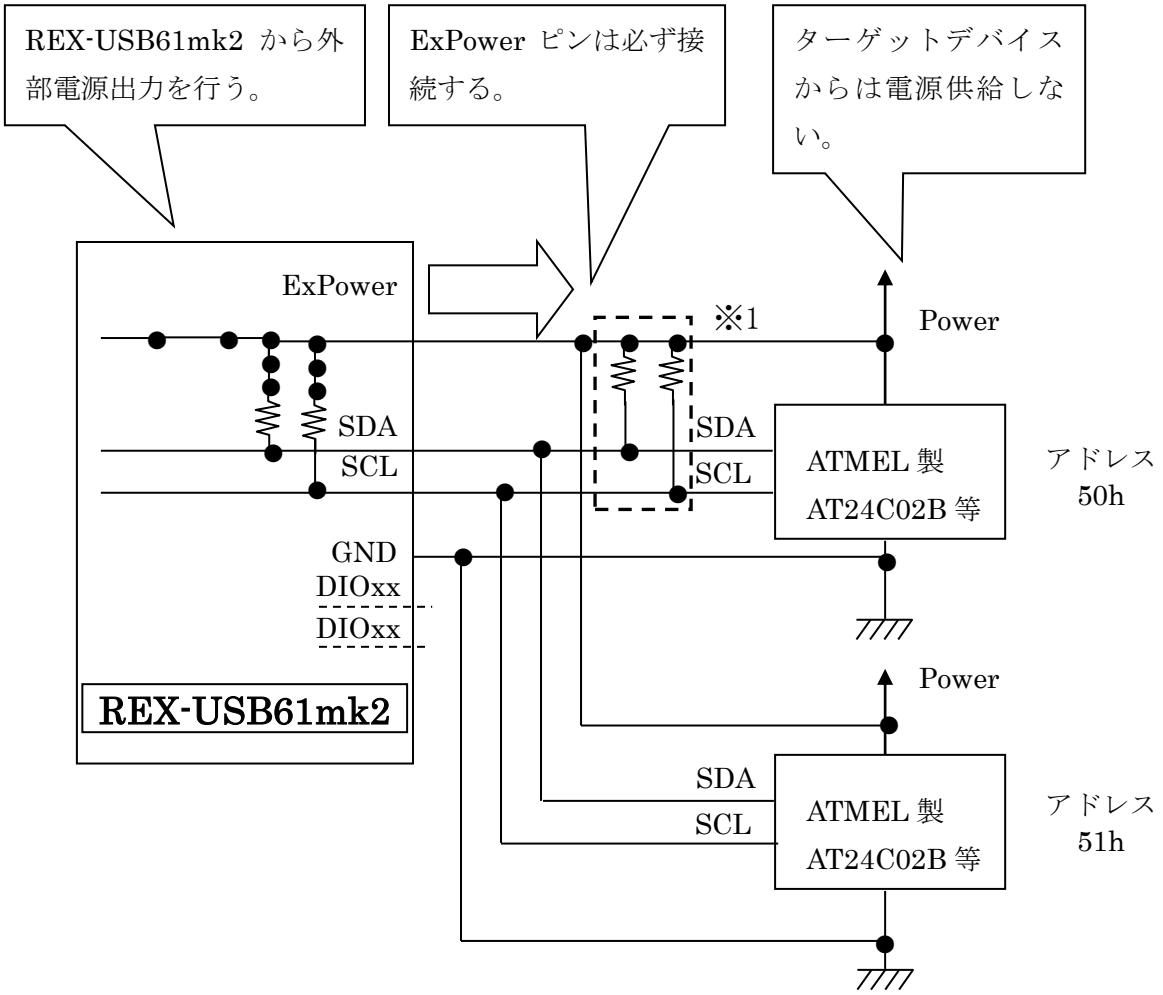
ターゲットデバイスに電源がある場合は、ユーティリティソフトウェアまたはライブラリ関数を利用したアプリケーションにて電源供給を無効にしてください。
(該当するライブラリ関数は usb61mk2_power_control()となります。
参照：(4-4)API 関数詳細)



※1 本機の内蔵プルアップ抵抗は 10kΩ です。
必要に応じてプルアップ抵抗を追加してください。

I2C 接続（ターゲットデバイスが電源を持たない場合）

本機からターゲットデバイス側へ電源供給(1.8/2.5/3.3/5.0V)する場合は、ユーティリティソフトウェアまたはライブラリ関数を利用したアプリケーションにておこないます。(供給電流は最大 100mA)
(該当するライブラリ関数は usb61mk2_power_control()となります。
参照：(4-4)API 関数詳細)



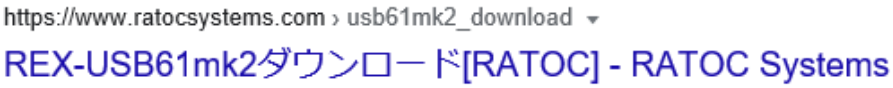
※1 本機の内蔵プルアップ抵抗は 10kΩ です。
必要に応じてプルアップ抵抗を追加してください。

第2章 セットアップとUnitIDの登録

● ドライバー/ユーティリティ/サンプルプログラムのダウンロード

弊社ホームページを開き、画面右上部の検索欄に「USB61mk2 ダウンロード」と入力して検索します。 https://www.ratocsystems.com/

Web 検索エンジンに表示された下記リンクをクリックするとドライバー/ユーティリティ/サンプルプログラムのダウンロードページが表示されます。

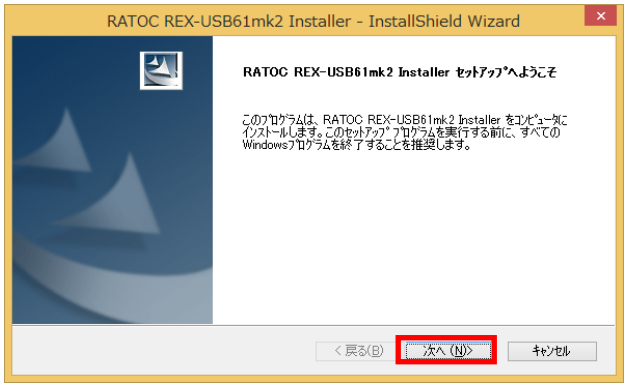


(2-1) Windows でのセットアップ

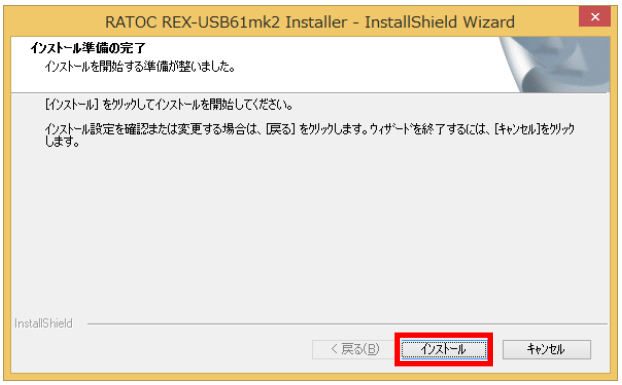
PC の電源を ON にし、本製品を USB ポートへ接続する前に以下の手順にてドライバーのセットアップを行ってください。

ダウンロードした USB61mk2_Setup.exe を実行します。
ユーザーアカウント制御の画面が表示される場合は、「はい(Y)」をクリックします。

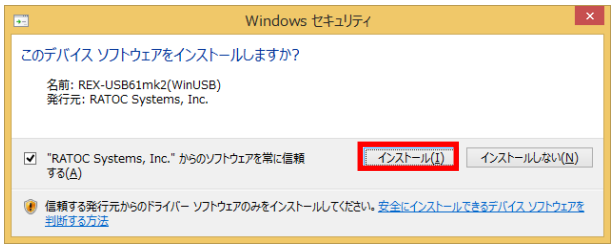
「RATOC REX-USB61mk2 Installer セットアップへようこそ」で「次へ(N)」をクリックします。



「インストール準備の完了」で「インストール」をクリックします。

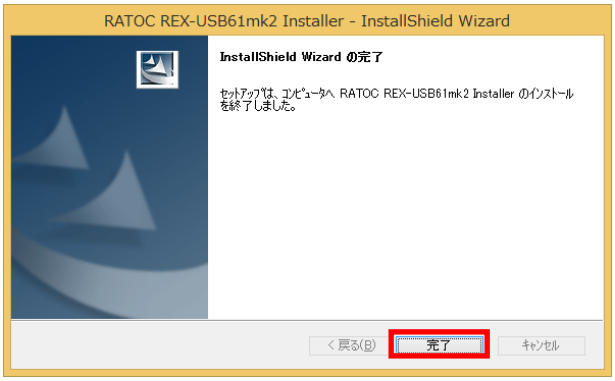


Windows セキュリティ画面が表示される場合は「インストール(I)」をクリックします。



以上でドライバーのセットアップは完了です。

REX-USB61mk2 を PC に接続すると自動的にインストールが完了します。

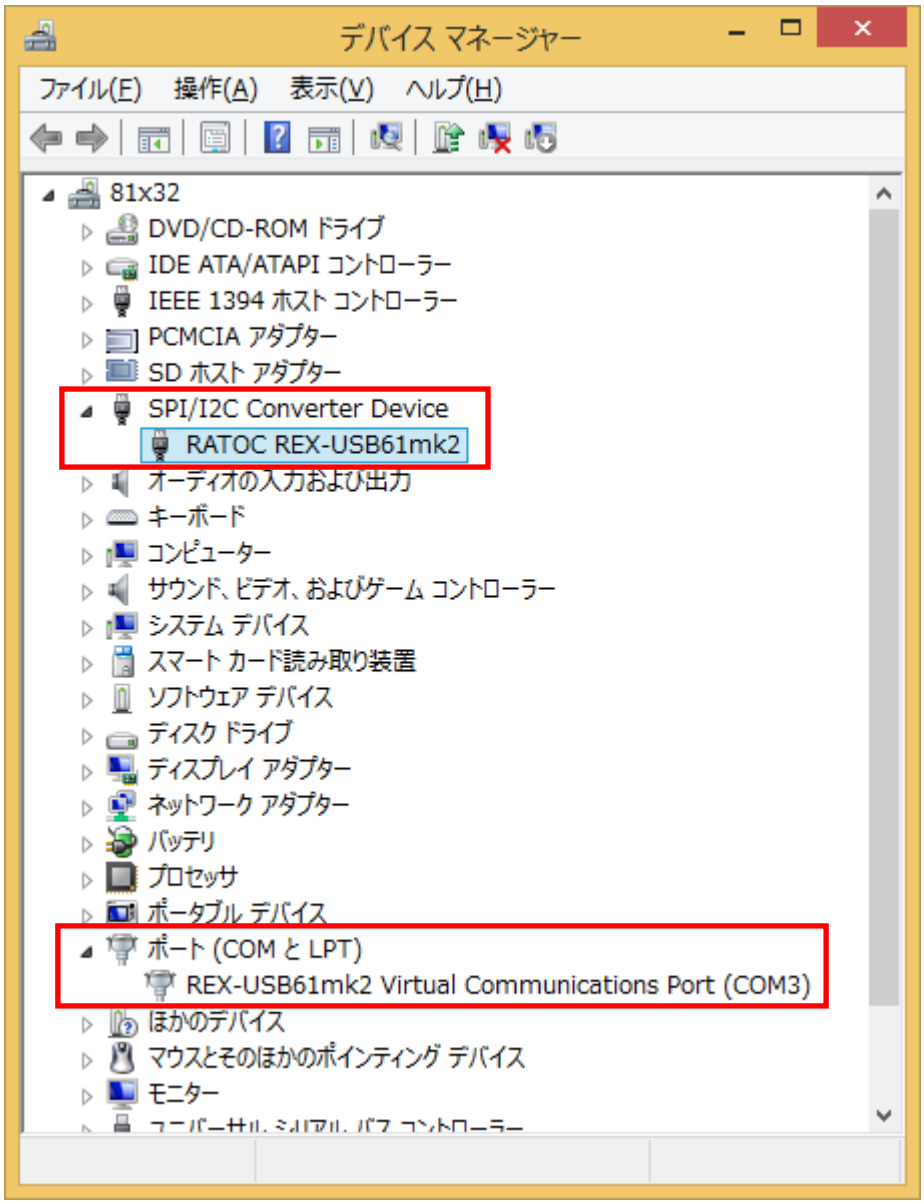


「(2-2) REX- USB61mk2 インストールの確認」へ進み、インストールの確認を行ってください。

(2-2) REX- USB61mk2 インストールの確認

コントロールパネルを開き「デバイスマネージャー」を起動します。

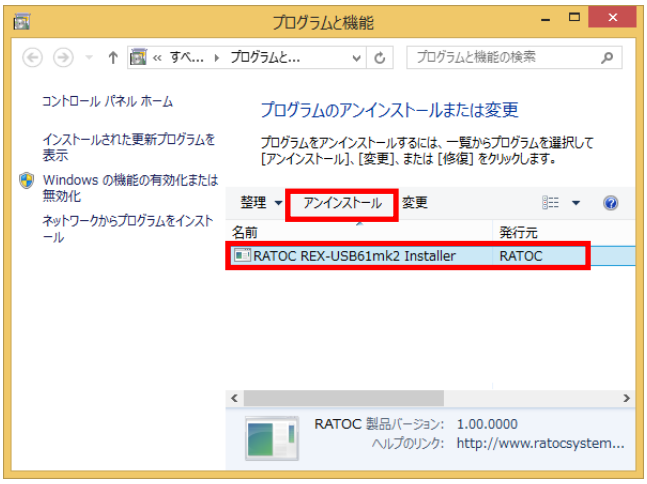
[SPI /I2C Converter Device」クラスの下に「RATOC REX-USB61mk2」と[ポート(COM と LPT)]クラスの下に「REX-USB61mk2 Virtual Communications Port(COMxx)」
が正常に認識されていることを確認してください。
(COMxx の番号は環境によって異なります。)



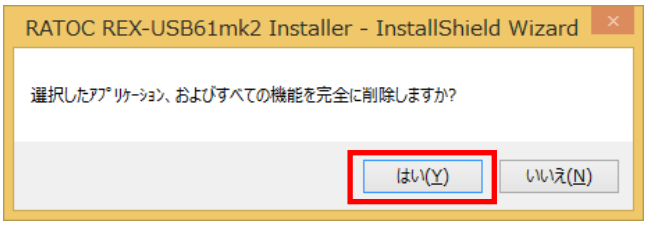
(2-3) Windows でのアンインストール方法

コントロールパネルの「プログラムと機能」を起動します。

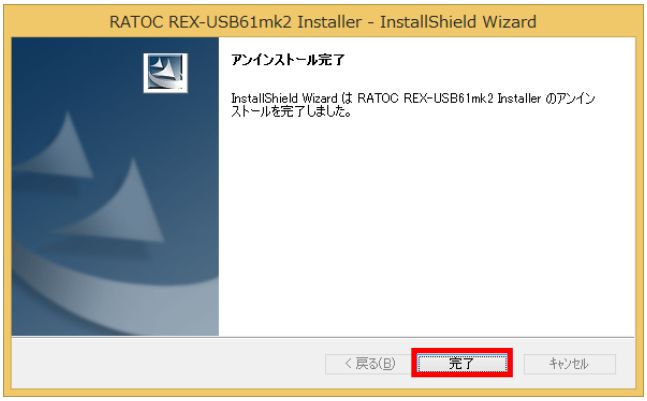
「RATOC REX-USB61mk2 Installer」を選択し、「アンインストール」をクリックします。



アンインストールの確認画面で、「はい(Y)」をクリックします。



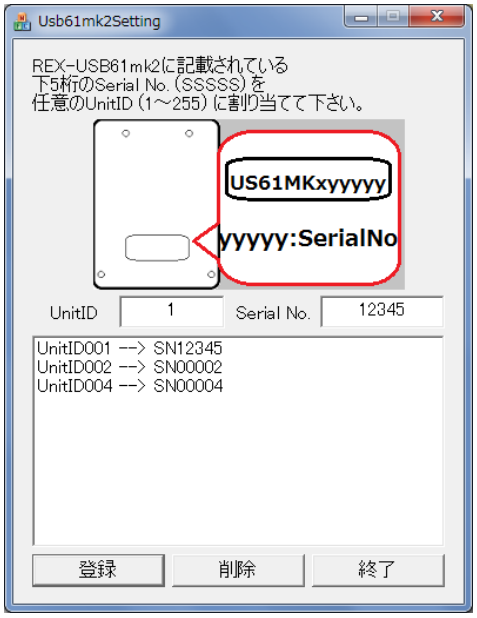
以上で REX-USB61mk2 のアンインストールは完了です。



(2-4) UnitIDの登録方法

複数台のREX-USB61mk2を1台のPCで使用する場合は、アプリケーション上で判別して制御するために、各REX-USB61mk2に固有のUnitIDを登録する必要があります。
(本製品を1台で使用する場合はUnitIDを登録する必要はありません。
UnitIDの登録有無にかかわらず、UnitID「0」としてご使用いただけます。)

ダウンロードした
Usb61mk2Setting.exeを実行します。



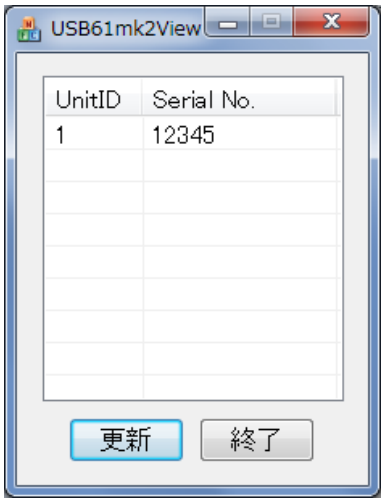
[UnitID]には固有のID(1～255)を入力、
[Serial No.]には製品のシリアルナンバー(下5桁)を入力し「登録」ボタンをクリックします。
登録されていることを確認し、「終了」ボタンをクリックします。

以上でUnitIDの登録は完了です。

※ 設定したUnitIDは、登録したPC上で有効となります。

(2-5) 登録済みUnitIDの確認

PCに接続されているREX-USB61mk2のUnitIDとSerial No.を確認することができます。
ダウンロードしたUSB61mk2View.exeを実行します。



接続されているREX-USB61mk2の[UnitID] [Serial No.]が表示されます。

「更新」ボタンをクリックすると接続情報が更新されます。

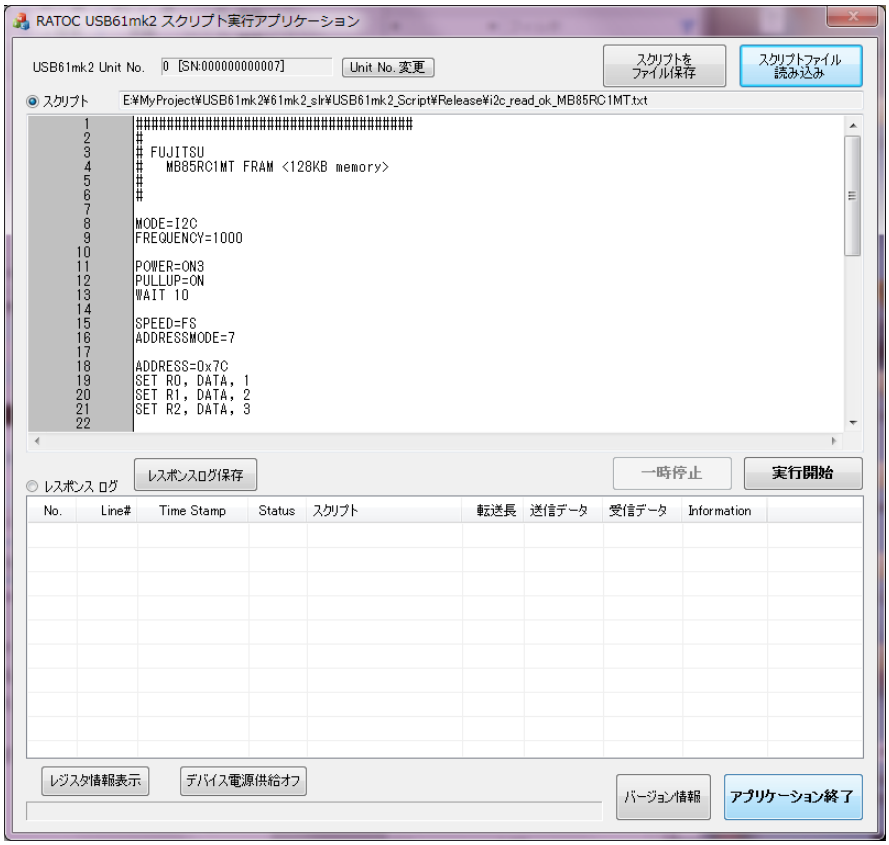
第3章 スクリプト制御について

(3-1) スクリプト実行アプリケーションについて

USB61mk2_Script.exe(ダウンロード提供)では、スクリプト命令を使用して SPI または I2C インターフェイスを持ったターゲット機器に対して制御を行うことができます。
スクリプト処理は、あらかじめ決められた制御で比較的簡単かつ短時間で完了するような処理を行う場合に適しています。
また、スクリプト実行時のブレークポイントを設定することができ、レジスタ情報の表示・変更、レスポンスデータの保存も可能です。
(複雑な処理や判断を伴うもの、長時間の制御やきめ細かな処理を行う場合は、API を使用したアプリケーションでの制御が必要です。)

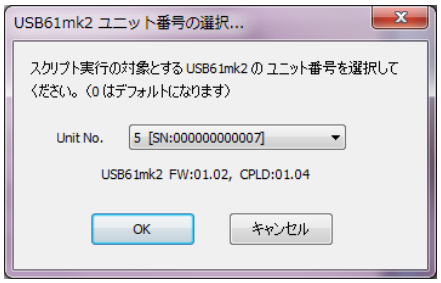
■ スクリプト実行アプリケーションの動作環境

・OS：Windows 10/8.1/8/7Vista(SP2) [32bit/64bit]
・CPU：x86 プロセッサ(Intel,AMD)　2GHz 以上、マルチコアプロセッサ
・メモリー：使用可能な空き物理 RAM 2GB 以上
・HDD：空き容量 2GB 以上



■ スクリプト実行アプリケーション画面の説明

[Unit No.] 制御する REX-USB61mk2 の UnitID と製品シリアル番号を表示します。
(UnitID 登録ツールで設定されている ID です。)
[Unit No. 更新] 本製品が複数台接続されている場合、制御する REX-USB61mk2 を変更することができます。
(本製品が 1 台の場合は変更する画面は表示されません。)



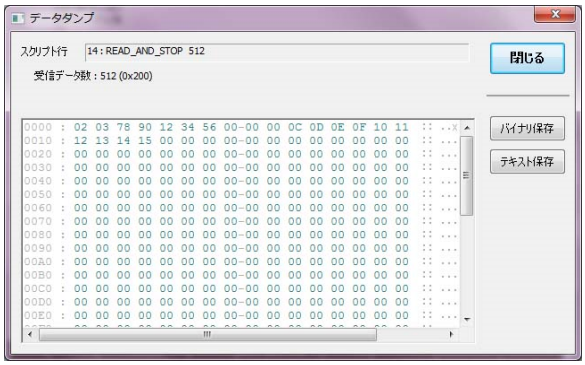
[スクリプトをファイル保存] 現在のスクリプト内容をファイルに保存します。
[スクリプトファイル読み込み] 保存されたスクリプトファイルを読み込みます。
[レスポンスログ保存] レスポンスログ欄の内容を CSV 形式でファイル保存します。

【レスポンスログ各項目の説明】

[No.] 1 から始まるレスポンス番号。
[Line#] このレスポンスに対応するスクリプト行番号。
[Time Stamp] 本製品が返したタイムスタンプ情報。
実行開始時からの時間を [秒.ミリ秒,マイクロ秒]で表示。
[Status] 本製品がエラーを返した場合のエラー内容。
[スクリプト] 対応するスクリプト内容。
[転送長] データ送信・受信時のデータ転送長。
[送信データ] データ送信時の先頭から 10 バイトまでの送信データ内容。
[受信データ] データ受信時の先頭から 10 バイトまでの受信データ内容。
[Information] スクリプト実行時の追加情報。
-- 対象の I2C スレーブアドレス／アドレスモード、
DIOINT/SPI_WAITRDY の完了またはタイムアウト情報、
一時停止情報、FILE 指定による受信データ保存のファイル名、
GET Rn/DIOGET の取得値。

【送受信データのダンプ表示】

レスポンスログ上の送信データまたは受信データ上でダブルクリックすると、データをダンプ表示することができます。また、バイナリ/テキスト保存が可能です。



[実行開始] 現在のスクリプト内容を実行します。
実行開始すると[実行停止]にボタン名が変更となります。(実行停止)

[一時停止] 実行中のスクリプトを一時停止します。
一時停止すると[引き続き実行]にボタン名が変更となります。(実行再開)

[レジスタ情報表示] スクリプトの実行終了後や一時停止状態時に、本製品のレジスタ情報および DIO 入力値(1 バイト)を確認することができます。
また、一時停止状態であればレジスタ内容を変更可能です。

[デバイス電源供給オフ] 接続デバイスへの電源供給がオン状態のとき、電源供給をオフにします。オフにすると、レジスタ情報はリセットされます。
(スクリプト実行が終了している状態のみ有効)

[バージョン情報] アプリケーション/DLL/ファームウェア/CPLD のバージョン情報を表示します。

[アプリケーション終了] アプリケーションを終了します。
(接続デバイスへの電源供給はオフとなります。)

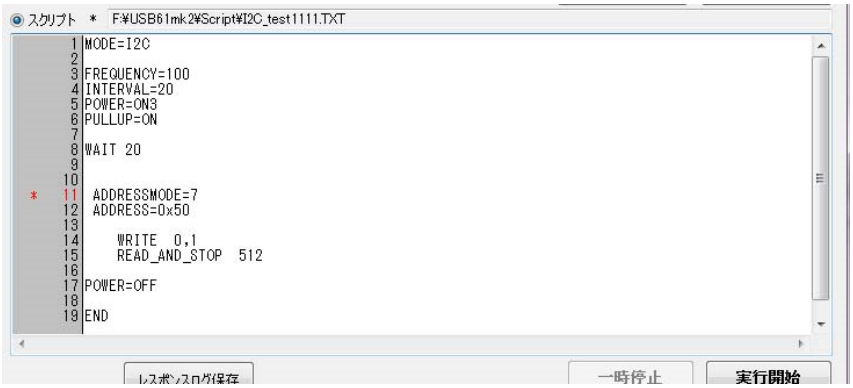
■ ブレークポイント機能について

スクリプト画面にカーソルがある状態で [F9]キーを押すと(または行番号をダブルクリック)、その行にブレークポイントを設定（または設定解除）することができます。

スクリプト実行中、指定したブレークポイント箇所に到達すると、そこで実行を一時停止します。

設定した箇所は、行番号が赤色で表示され先頭に[＊]でマークされます。

(以下は 11 行目にブレークポイントを設定した例です。)



ブレークポイントで一時停止となった箇所は、行番号の先頭に矢印 [=>]でマークされます。(ブレークポイントは 6 箇所まで設定可能です。)

以下の個所に設定されたブレークポイントは無効となり停止しません。

- スクリプトの先頭行
- 空白
- コメント
- ブロック開始（{）
- STOP 行

(3-2) スクリプト仕様

■ SPI/I2C 共通の文法

・ 文字の定義
アルファベット文字の大文字/小文字の区別は行わない。コメントは日本語に対応する。（SHIFT-JIS 形式）

・ 文法
命令文とパラメーターの間にはスペースまたは TAB を入れる。
複数の命令を同一行に記述することは禁止する。ただし例外として、READ/WRITE 系命令の後に STOP 命令を記述することは可能。（I2C の場合）

・ 数値（リテラル値）の定義
数値は 10 進数(READ,WRITE 系は 16 進数、8 進数、2 進数、1 文字キャラクタ）で記述する。
16 進数を表す場合は数値の後ろに[h]または[H]を付ける。
C 言語表記（16 進数 : 0xFF, 8 進数 : 077, 2 進数 0b11111111, キャラクタ ' A ' ）による指定も可能。

・ 分岐、ループ命令、ダイアログ命令
IF、DO-WHILE、REPEAT、MSGYESNO 命令ではブロック {} で囲うこと。

・ レジスタの定義
レジスタは 0～15 までの 16 個あり、1 つに付き 8 ビットサイズの領域がある。
分岐処理やビット演算のオペランドとして使用することができる。
命令文のパラメーターで使用する場合は、「Rn」（n にレジスタ番号を記述:0～15）のように記述する。
また、特別なレジスタとして CR(R16) がある。このレジスタは、直前に行った加算もしくは減算の結果キャリーが発生した場合に、「１」となる。

・ ダイアログ表示
スクリプト途中にダイアログを表示する機能。対応する命令文の行で実行を一時停止しダイアログ表示を行う。ダイアログクローズ後実行を継続する動作となる。

・ ファイル指定
スクリプトの前方で、FILEn “Filename”（n は 10 進数の値）と記述した場合、指定したファイルの使用を宣言する。以降の READ/WRITE 系スクリプトのパラメーターに、FILEn を指定することができる。
また、READ/WRITE 系スクリプトのパラメーターで、FILEn の替わりに FILE:”ファイル名”と記述することで、直接パス・ファイル名を指定することができる。データ受信の場合で本形式を指定した場合、すでにファイルが存在する場合は追記となる。（FILE_CREATE: ”ファイル名”と指定すれば、新規作成となる）
FILE と 数値または コロン（ :)、イコール（=）は連続して記述すること。（空白を入れてはいけない）

■ 共通命令表

<table>
<tr><th>命令</th><th>パラメーター</th><th>意味</th></tr>
<tr><td>#
//
;</td><td></td><td>行中の「#」「//」「;」以降はコメント文として取り扱う。</td></tr>
<tr><td>MODE=SPI
または I2C</td><td>SPI

I2C</td><td>SPI、I2C のどちらのモードであるかを指定する。
MODE 命令にデフォルト設定はない。一度設定すると途中でモードを切り替えることは出来ない。
スクリプトの先頭でモード指定を行うこと。</td></tr>
<tr><td>FREQUENCY=nn</td><td>nn
周波数(KHz 単位)：
下記の設定が可能
<table><tr><th></th><th>nn 設定値</th></tr><tr><td>SPI</td><td>1～50000</td></tr><tr><td>I2C</td><td>1～5000</td></tr></table></td><td>使用する周波数を設定する。周波数は 1kHz 単位で設定が可能だが、実際に動作する周波数は設定値以下の近似値となる。
(本体内部コントローラーで計算される)
例) 16MHz に設定 ⇒ 12.5MHz で動作
17MHz に設定 ⇒ 16.66MHz で動作

周波数設定を行わない場合の初期値は下記とする。
<table><tr><th>モード</th><th>周波数</th></tr><tr><td>SPI</td><td>100kHz</td></tr><tr><td>I2C</td><td>100kHz</td></tr></table>
<u>注意</u>：
SPI の場合、「SAMPLING=」「FB=」「SSPOL=」設定の後に、本命例による周波数設定を行うこと。</td></tr>
<tr><td>INTERVAL=nn</td><td>nn
マイクロ秒単位定
<table><tr><th></th><th>設定値</th></tr><tr><td>SPI</td><td>0、
12～65535</td></tr><tr><td>I2C</td><td>0、
12～65535</td></tr></table></td><td>送信するデータのバイトとバイトの間に入る待ち時間を設定する。単位はマイクロ秒。
待ち時間設定を行わない場合、0 を設定したときと同じとなる。</td></tr>
</table>

<table>
<tr><th>命令</th><th>パラメーター</th><th>意味</th></tr>
<tr><td>POWER=ss</td><td>ss
下記の指定で設定可能
<table><tr><th>パラメーター</th><th>出力</th></tr><tr><td>OFF</td><td>OFF</td></tr><tr><td>ON1</td><td>1.8V</td></tr><tr><td>ON2</td><td>2.5V</td></tr><tr><td>ON3</td><td>3.3V</td></tr><tr><td>ON5</td><td>5.0V</td></tr></table></td><td>パラメーターで設定された電源出力を行う。
電源はいつでも変更することができる。
電源設定を行わない場合の初期値は「出力 OFF」とする。</td></tr>
<tr><td>WAIT=nn</td><td>nn
100 ミリ秒単位の時間
1～600 の数値を指定</td><td>次の命令を実行する前に指定された時間待ちを行う。100 ミリ秒単位で待ち時間を設定する。指定できる時間は 100 ミリ秒～60 秒までとする。</td></tr>
<tr><td>WAIT2=nn</td><td>nn
1 ミリ秒単位の時間
1～60000 の数値を指定</td><td>次の命令を実行する前に指定された時間待ちを行う。1 ミリ秒単位で待ち時間を設定する。指定できる時間は 1 ミリ秒～60 秒までとする。</td></tr>
<tr><td>PULLUP=ON
または OFF</td><td>ON または OFF</td><td>SDA、SCL 信号線を Pull-up するかどうかを指定する。初期値は Pull-up しない(OFF)とする。</td></tr>
<tr><td>FILEn “filename”</td><td>n
番号(1～20)
“filename”
パス・ファイル名を指定</td><td>送信・受信で使用するファイルを定義する。n にはファイル番号が入る。最大 20 ファイルまで定義可能。
”filename”はファイル名を指定する。送信の場合、ファイルが無い場合はエラーとなる。受信の場合、すでに同じファイルが存在す場合は追記となり、ファイルがなければ新たにファイルが作成される。</td></tr>
<tr><td>FILEn_CREATE
“filename”</td><td>n
番号(1～20)
“filename”
パス・ファイル名を指定</td><td>受信で使用するファイルを定義する。n にはファイル番号が入る。最大 20 ファイルまで定義可能。
”filename”はファイル名を指定する。すでに同じファイルが存在す場合、そのファイルは消去され新たにファイルが作成される。</td></tr>
<tr><td>END</td><td>無し</td><td>処理を終了する。（EXIT と同じ）</td></tr>
</table>

■ レジスタ、分岐処理の命令表

<table>
<tr><th>命令</th><th>パラメーター</th><th>意味</th></tr>
<tr><td>DIODIR=x1</td><td>x1
レジスタまたはリテラル値</td><td>DIO の入出力設定を行う。各ビットが DIO0～DIO7 に対応しており、「0」で入力、「1」で出力とする。
設定しない場合の初期値は、「0」ですべて入力とする。
一度設定した後は変更することを禁止する。</td></tr>
<tr><td>DIOINT=x1, x2

DIOINT=DISABLE</td><td>x1
レジスタまたはリテラル値

x2
0～3</td><td>x1 は DIO の入力ポートの割り込み検出設定を行う。
各ビットが DIO0～DIO7 に対応しており、「0」で割込検出なし、「1」で割込検出ありとする。出力設定となっているビットは無視される。
設定しない場合の初期値は、「00h」ですべて割込検出なし（割り込み禁止）とする。(DISABLE を指定したのと同じ)
x2 は割込判定条件を指定する。

割込判定条件
<table>
<tr><th>x2 値</th><th>条件</th></tr>
<tr><td>0</td><td>Low レベル</td></tr>
<tr><td>1</td><td>High レベル</td></tr>
<tr><td>2</td><td>Low ⇒ High の変化</td></tr>
<tr><td>3</td><td>High ⇒ Low の変化</td></tr>
</table>
DIOINT=DISABLE　とした場合は、DIO 全ポートで割り込み検出なし（禁止：x1 で 00h を指定したのと同じ）とする。
DIO 割り込みを使用後は、DISABLE を指定して割り込みを禁止とすること。</td></tr>
<tr><td>DIOINT_DEBOUNCE
=nn</td><td>nn
リテラル値 ミリ秒
0～500</td><td>DIO 割り込み検知時のチャタリング防止の時間をミリ秒単位で指定する。
設定範囲は 0～500 とし、初期値は 0 とする。</td></tr>
<tr><td>WAITINT=x1, Rn</td><td>x1
リテラル値 0～60000
Rn
レジスタ（n:0～15）</td><td>パラメーターには、「タイムアウトの時間（x1)」を設定する。
DIOINT で指定した条件で割り込みを検知するまで待つ。
タイムアウトは 1 ミリ秒単位で指定する。「0」を指定した場合は、タイムアウトなしとする。
Rn で指定したレジスタには、割り込み発生で 0, タイムアウトで 1 がセットされる。
DIO 割り込みを使用後は、DIOINT 設定で、割り込みを禁止(DISABLE)にすること。</td></tr>
<tr><td>DIOSET=x1</td><td>x1
レジスタまたはリテラル値</td><td>DIO 出力ポートに出力を行う。各ビットが DIO0～DIO7 に対応しており、「0」で Low、「1」で High とする。
設定しない場合の初期値は、「0」ですべて Low とする。</td></tr>
<tr><td>DIOGET</td><td>無し</td><td>DIO 入力ポートの状態を入力する(DIO 入力データをレスポンスログ表示欄に表示する)。
各ビットが DIO0～DIO7 に対応しており、「0」で Low、「1」で High とする。</td></tr>
</table>

<table>
<tr><th>命令</th><th>パラメーター</th><th>意味</th></tr>
<tr><td>SET Rn=x1 (, x2)
SET Rn,x1 (, x2)</td><td>Rn
レジスタ（n:0～15）

x1
レジスタ,リテラル値, DIO, DATA

x2
リテラル値（x1 が DATA の時のみ指定可能）</td><td>内部の演算で使用するレジスタの設定を行う。
パラメーターx1 で DATA を指定した時は、読み込み値は、本命令以降に SPI/I2C のデバイスから読み込んだデータが対象となり、パラメーターx2 で指定した、1 から始まるバイト数目の値を入力値としてレジスタに設定する。
<table>
<tr><th>対象</th><th>x1</th><th>x2</th></tr>
<tr><td>DIO</td><td>DIO</td><td>なし</td></tr>
<tr><td>読み込み値</td><td>DATA</td><td>xxxx バイト目（1 以上）</td></tr>
<tr><td>リテラル値</td><td>0-255</td><td>なし</td></tr>
<tr><td>レジスタ</td><td>Rn</td><td>なし</td></tr>
</table>
</td></tr>
<tr><td>GET Rn</td><td>Rn
レジスタ（n:0～16）</td><td>指定したレジスタの値の取得を行う(指定したレジスタ値をレスポンスログ表示欄に表示する)。
CR（キャリーレジスタ）を取得する場合は 16 を指定する。</td></tr>
</table>

命令	パラメーター	意味
AND Rn=x1, x2 AND Rn, x2	Rn レジスタ（n:0～15） x1, x2 レジスタまたはリテラル値	x1 と x2 で指定した値の論理積を指定したレジスタへ代入する。 [Rn,x2] の形式は、[Rn = Rn, x2]を指定したと解釈する。
OR Rn=x1, x2 OR Rn, x2	Rn レジスタ（n:0～15） x1, x2 レジスタまたはリテラル値	x1 と x2 で指定した値の論理和を指定したレジスタへ代入する。 [Rn,x2] の形式は、[Rn = Rn, x2]を指定したと解釈する。
XOR Rn=x1, x2 XOR Rn, x2	Rn レジスタ（n:0～15） x1, x2 レジスタまたはリテラル値	x1 と x2 で指定した値の排他的論理和を指定したレジスタへ代入する。 [Rn,x2] の形式は、[Rn = Rn, x2]を指定したと解釈する。
NOT Rn=x1 NOT Rn, x1	Rn レジスタ（n:0～15） x1 レジスタまたはリテラル値	x1 で指定した値の否定を指定したレジスタへ代入する。
SHIFTR Rn=x1, x2 SHIFTR Rn, x2	Rn レジスタ（n:0～15） x1 レジスタまたはリテラル値 x2 リテラル値(シフト回数)	x1 で指定した値を x2 で指定したビット数右へビットシフト演算を行い指定したレジスタへ代入する。空いたビットには、0 を挿入する。 [Rn,x2] の形式は、[Rn = Rn, x2]を指定したと解釈する。
SHIFTL Rn=x1, x2 SHIFTL Rn, x2	Rn レジスタ（n:0～15） x1, レジスタまたはリテラル値 x2 リテラル値(シフト回数)	x1 で指定した値を x2 で指定したビット数左へビットシフト演算を行い指定したレジスタへ代入する。空いたビットには、0 を挿入する。 [Rn,x2] の形式は、[Rn = Rn, x2]を指定したと解釈する。
ADD Rn=x1, x2 ADD Rn, x2	Rn レジスタ（n:0～15） x1, x2 レジスタまたはリテラル値	x1 に x2 で指定した値を加算した結果を、指定したレジスタへ代入する。 キャリー発生時レジスタ CR(R16)に「1」が設定される。 [Rn,x2] の形式は、[Rn = Rn, x2]を指定したと解釈する。
SUB Rn=x1, x2 SUB Rn, x2	Rn レジスタ（n:0～15） x1, x2 レジスタまたはリテラル値	x1 に x2 で指定した値を減算した結果を、指定したレジスタへ代入する。計算は、「x1 – x2」が実行される。 キャリー発生時レジスタ CR に「1」が設定される。 [Rn,x2] の形式は、[Rn = Rn, x2]を指定したと解釈する。

命令	パラメーター	意味

IF x1 ope x2
{
proc1
}
ELSE
{
proc2
}
ENDIF
	x1, x2
	レジスタまたはリテラル値
	
	ope
	演算子
	
	proc1
	条件成立時の処理スクリプト
	proc2
	条件不成立時の処理スクリプト
		演算した結果、TRUE の場合は proc1 の処理が実行され、FALSE の場合は proc2 を実行する。
		使用できる演算子は、「==」「<>」「<」「>」「<=」「>=」とする。
		(「!=」も可能、この場合 「<>」と解釈する)
		proc2 が不要な場合は、ELSE{ }ブロックを省略可能とする。
		最後の ENDIF 文は必須である。
		IF は最大 5 回までネストして記述可能とする。
		DO~WHILE および REPEAT の中にも記述可能とする。
		(本製品の内部バッファサイズに制限がありますので、エラーとなる場合は命令数を減らしてください。)

DO
{
proc
}
WHILE x1 ope x2
	x1, x2
	レジスタまたはリテラル値
	
	ope
	演算子
	
	proc
	処理スクリプト
		初回 1 回目は必ず proc が実行され、2 回目以降は演算した結果、TRUE の間実行が繰り返されるとする。
		使用できる演算子は、「==」「<>」「<」「>」「<=」「>=」の 6 個とする。
		proc で記述できるコード数は本製品の内部バッファサイズにより制限がある。
		DO～WHILE のネストは不可とする。
		DO-WHILE ブロックに、REPEAT、ダイアログ系命令を入れることはできない。
		(本製品の内部バッファサイズに制限がありますので、エラーとなる場合は命令数を減らしてください。)
		無限ループになるような記述は禁止する。

REPEAT nn
{
proc
}
	nn
	リピート回数：リテラル値(1～65536)
		REPEAT 命令文の次に書かれた「{ }」内の命令を指定回数繰り返す。
		繰り返す範囲は必ず {} で囲うこと。
		「{ }」内に記述できるコード数は本製品の内部バッファサイズにより制限がある。
		REPEAT のネストは不可とする。
		リピートブロックに、Do-WHILE、ダイアログ系命令を入れることはできない。
		(本製品の内部バッファサイズに制限がありますので、エラーとなる場合は命令数を減らしてください。)
		無限ループになるような記述は禁止する。

EXIT		この時点で実行を中断し終了する。 （END と同じ）

■ ダイアログ表示処理の命令表

命令	パラメーター	意味
MSGOK "message", Rn	“message” 表示する文字列 （省略可能） Rn 表示するレジスタ （省略可能）	OK ダイアログを表示する。message で指定した文字列と Rn で指定したレジスタ値を表示する。 OK を押下するまで実行が一時停止する。 メッセージを省略した時は レジスタ値のみ表示する。 レジスタのみ指定した場合、メッセージは表示せずレジスタ値のみ表示する。

MSGYESNO “message”, Rn
{
proc1
}
ELSEMSG
{
proc2
}
ENDMSG
	“message”
	表示する文字列
	（省略可能）
	
	Rn
	表示するレジスタ
	（省略可能）
		YES/NO ダイアログを表示する。message で指定した文字列と Rn で指定したレジスタ値を表示する。
		メッセージを省略した時は レジスタ値のみ表示する。
		レジスタのみ指定した場合、メッセージは表示せずレジスタ値のみ表示する。
		YES または NO を押下するまで実行が一時停止する。
		YES を押した場合は、proc1 を実行する。NO を押した場合は、proc2 を実行する。
		MSGYESNO は最大 5 回までネストして記述可能とする。
		なお、本命令は本製品の内部バッファサイズにより制限がある。

■ I2C 専用の命令表

命令	パラメーター	意味
ADDRESSMODE=n	n リテラル値：I2C スレーブアドレスのモード（7 または 10）	I2C アドレスを 7 ビットモードか 10 ビットモードに設定する。初期値は 7 ビットモードとする。
ADDRESS=nn	nn リテラル値：I2C スレーブアドレス （0～1023）	I2C スレーブアドレスを指定する。スレーブアドレスはいつでも変更することが可能だが、アドレス指定する前に「READ」や「WRITE」がある場合、文法エラーとする。
SPEED=speed, [ad]	speed LOW, HS, UF FS(LOW と同じ) US(UF と同じ) ad 通信速度の調整値. 設定範囲は-15～15 未設定時は 0 とする	speed I2C の通信速度の指定を行う。 LOW 指定で、Standard/Fast-mode、HS 指定で High-speed モード、UF 指定で Ultra Fast-mode での通信を有効にする。この設定はいつでも変更することが可能。 Ultra Fast-mode 有効の状態で READ 命令を実行した場合はエラーとなる。初期値は LOW とする。 ad SPEED に設定しているモードに対して周波数の調整値を反映する。調整値は SPEED モード毎に保持される。デフォルトは 0 とする。 調整値範囲 100kHz : -3.6kHz ～ 2.2kHz 400kHz : -40kHz ～ 50kHz 1MHz : -120kHz ～430kHz 3.4MHz(Hs-mode) : -1.5MHz ～ 18MHz 5.0MHz(UF-mode) : -3MHz ～ 50MHz
MASTER_CODE nn	nn リテラル値：マスターコード (通常 0x08)	Hs-mode 切り替え時の マスターコードを送信する。マスターコードとして任意の値を送信可能だが、I2C 規格上 00001xxx(2 進数表記：x はデバイスにより決められた値) を送信することになる。
READ (LENGTH=)nn READ (LENGTH=)nn, FILEn	nn リテラル値：読み出しバイト数を指定 （1～65536 まで） FILEn 入力データを保存するファイルを指定する	指定されたバイト数分読み出しを行う FILE を指定した場合は、指定ファイルにバイナリデータで保存する。 既存のファイルが指定された場合は追加書き込みされていく。

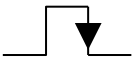
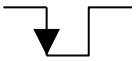
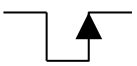
命令	パラメーター	意味
WRITE　xx (, xx …) WRITE　FILEn	xx リテラル値または レジスタ FILEn 出力するデータファ イルを指定する	指定されたデータを送信する。データが複数ある場合カンマで区切る。 レジスタを指定する場合、その WRITE スクリプト行の送信データはすべてレジスタ指定でなければならない。（レジスタとリテラル値の混在は不可）FILE を指定した場合、指定ファイル内容をバイナリデータで送信する。
READ_AND_STOP (LENGTH=)nn READ_AND_STOP (LENGTH=)nn, FILEn	READ と同じ	READ と同じだが、最後に STOP ビットを送信する。 READ と STOP の組み合わせでの命令は REX-USB61 との互換性を保つために使用可能だが、本製品のみで使用する場合は、READ_AND_STOP を使用することを推奨する。
WRITE_AND_STOP　xx (, xx …) WRITE_AND_STOP　FILEn	WRITE と同じ	WRITE と同じだが、最後に STOP ビットを送信する。 WRITE と STOP の組み合わせでの命令は REX-USB61 との互換性を保つために使用可能だが、本製品のみで使用する場合は、WRITE_AND_STOP を使用することを推奨する。
RESET	無し	バスにリセットを発生させる

■ I2C スクリプト例

```
# I2C スクリプト例
MODE=I2C

FILE1  “ReadData.bin”

FREQUENCY=300      ; 300KHz
INTERVAL=20

POWER=ON3
PULLUP=ON

WAIT 10

ADDRESSMODE=7
ADDRESS=0x50            ; I2C Slave address


  WRITE   0x10, 0x02
  READ_AND_STOP   8, FILE1    ; 8 バイトリードしてファイルに保存


POWER=OFF

END
```

■ SPI 専用の文法

<table>
<tr><th>命令</th><th>パラメーター</th><th>意味</th></tr>
<tr><td>SAMPLING=n</td><td>n=0～3</td><td>バスサンプリング方法を指定する。初期値は 0 とする。

<table>
<tr><th>パラメーター</th><th>サンプリングエッジ</th><th>図</th></tr>
<tr><td>0</td><td>立ち上がりエッジ</td><td>  </td></tr>
<tr><td>1</td><td>立ち下がりエッジ</td><td>  </td></tr>
<tr><td>2</td><td>立ち下がりエッジ</td><td>  </td></tr>
<tr><td>3</td><td>立ち上がりエッジ</td><td>  </td></tr>
</table>
一度設定した後は変更することを禁止する。
IF、MSGYESNO 分岐処理内で使用することはできない。
周波数設定は、本命令より後に行うこと。</td></tr>
<tr><td>FB=</td><td>MSB / LSB</td><td>ファーストビットを指定する。初期値は MSB とする。
一度設定した後は変更することを禁止する。
IF、MSGYESNO 分岐処理内で使用することはできない。
周波数設定は、本命令より後に行うこと。</td></tr>
<tr><td>SSPOL=
or
SS_ACTIVE=</td><td>HIGH / LOW</td><td>SS 信号のアクティブレベルを選択する。アクティブ HIGH､アクティブ LOW を設定する。
一度設定した後は変更することを禁止する。
IF、MSGYESNO 分岐処理内で使用することはできない。
周波数設定は、本命令より後に行うこと。</td></tr>
<tr><td>SSSET
or
SS_SET</td><td>無</td><td>SS 信号をアクティブにする。</td></tr>
<tr><td>SSRESET
or
SS_RESET</td><td>無</td><td>SS 信号を非アクティブにする。</td></tr>
<tr><td>BITS=n</td><td>n=1～32</td><td>SPI で転送するビット数の指定を行う。いつでも変更することが可能で､変更した以降の通信に適用される。
設定しない場合の初期値は、「8」とする。</td></tr>
<tr><td>MULTI=SINGLE or DUAL or QUAD</td><td>SINGLE/DUAL/
QUAD</td><td>SPI の高速書込みモード（DUAL-SPI/QUAD-SPI）の設定を行う。いつでも変更することが可能で､変更した以降の通信に適用される。
設定しない場合の初期値は、「SINGLE」とする。</td></tr>
</table>

WRITE xx (, xx …) WRITE FILEn	xx リテラル値または レジスタ FILEn 出力するデータ ファイルを指定する	ライトデータのみ指定する。 指定されたデータを送信する。データが複数ある場合カンマで区切る。 レジスタを指定する場合、本命例の送信データはすべてレジスタ指定でなけらばならない。（レジスタとリテラル値の混在は不可）FILE を指定した場合、指定ファイル内容をバイナリデータで送信する。
READ (LENGTH=)nn READ (LENGTH=)nn,FILEn	nn リテラル値：読み出しバイト数を指定（1～65536 まで） FILEn 入力データを保存するファイルを指定する	データ長を指定する。 ファイル指定を行った場合、指定バイト数のリードデーターをファイルにバイナリデータで保存する。 既存のファイルが指定された場合は追加書き込みする。
SPI_XFER xx, xx, … FILEin SPI_XFER FILEout, FILEin	xx リテラル値またはレジスタ FILEout 出力するデータファイルを指定する FILEin 入力データを保存するファイルを指定する	指定バイトデータ、レジスタ、またはファイル(FILEout)内容を出力しつつ、入力データを最終パラメーターの FILEin に保存する。 レジスタを指定する場合、本命例の送信データはすべてレジスタ指定でなけらばならない。（レジスタとリテラル値の混在は不可）
SPI_WAITRDY x1, x2, x3, x4, x5, Rn	x1 リテラル値：1 バイトのオペコード x2 リテラル値：ビット番号(0～7) x3 リテラル値：0 または 1 x4 リテラル値：タイムアウト値（100 ミリ秒単位）0～65535 x5 リテラル値：ステータスレジスタのセンス間隔（マイクロ秒単位）10-65535 Rn レジスタ（n:0～15）	SPI デバイス内部のステータスレジスタの RDY/BUSY が RDY になるまで待つ。 x1 に Read Status Register のオペコードを指定する。（1 バイト） x2 に RDY/BUSY のビット番号(0～7)を指定する。 x3 に RDY 時の論理を指定する（0 または 1）。 x4 にタイムアウト値を指定する。タイムアウトは 100 ミリ秒単位で指定する。 「0」を指定した場合は、タイムアウトなしとする。 x5 にステータスレジスタをセンスする間隔をマイクロ秒で指定する。最短は 10 マイクロ秒とする。 Rn で指定したレジスタには、完了ステータスがセットされる。RDY で 0, タイムアウト(BUSY)で 1 がセットされる。 例：SPI Flash ROM などで使用。

■ SPI スクリプト例

```
# SPI スクリプト例
MODE=SPI                # SPI モード

FILE1 "file1.bin"

POWER=ON3               # 外部電源出力 3.3V

WAIT 10

SSPOL=LOW
FREQUENCY=1000          # 1MHz
INTERVAL=100            # 送信バイト間隔
SAMPLING＝0
FB=MSB

SS_SET
  WRITE 0x03, 0, 0, 0
  READ   length=512, FILE1      # 512 バイトリードしてファイルに保存
SS_RESET


POWER=OFF

END
```

第4章 API関数仕様とサンプルプログラム

(4-1) VC での使用について

本 API 関数は、REX-USB61mk2 を使用したソフトウェア開発を支援するライブラリソフトウェアです。
API 関数を使用することで、SPI/I2C ターゲットデバイスの制御を自作のアプリケーションプログラムに組み込むことが可能となります。

Visual C++でライブラリ関数を使用するための
ヘッダファイル(RexUsb61mk2lib.h/RexUsb61mk2x64lib.h)、
ライブラリファイル(REXUSB61MK2.lib/REXUSB61MK2.dll, REXUSB61MK2x64.lib/ REXUSB61MK2x64.dll）を用意しています。
プロジェクトに上記のファイルを追加し、ライブラリ関数を呼び出してください。
ライブラリ関数のインポート宣言およびユーザ定義型の記述については、ヘッダファイル RexUsb61mk2lib.h/ RexUsb61mk2x64lib.h を参照してください。
※ LIB¥x64 フォルダー内にあるファイルは 64bit 版アプリケーション用となります。

(4-2) VB / Visual C#での使用について

Visual BASIC および Visual C#のアプリケーションから製品に添付された ActiveX コンポーネントを利用するためには、以下の方法により ActiveX の登録が必要です。

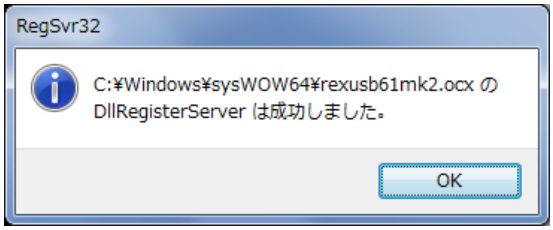
※ ActiveX コンポーネントは 32bit 版アプリケーションのみ対応となります。
64bit 版アプリケーションを作成される場合は LIB¥x64 フォルダー内にある定義ファイルを参照して API を呼び出してください。

（1）ActiveX の登録

第 2 章 Windows セットアップを参照しドライバのインストールを行ってください。
自動的に DLL,ActiveX のコピーが行われます。
RexUsb61mk2.ocx を VB で使用するためには、Visual BASIC に添付されているツール”Regsvr32.exe”を使って登録を行います。
”Regsvr32.exe”は 32 ビットコンソールアプリケーションですのでコマンドプロンプトから実行します。
登録の際にはコマンドプロンプトを管理者権限で起動し、以下のように実行します。

■ 32 ビット版 OS の場合
>regsvr32 rexusb61mk2.ocx
■ 64 ビット版 OS の場合
>regsvr32 C:¥Windows¥sysWOW64¥rexusb61mk2.ocx

登録成功メッセージ(32 ビット版 OS)



登録成功メッセージ(64 ビット版 OS)

(2) ActiveX の削除

登録から削除する際にはコマンドプロンプトを管理者権限で起動し、以下のように実行します。

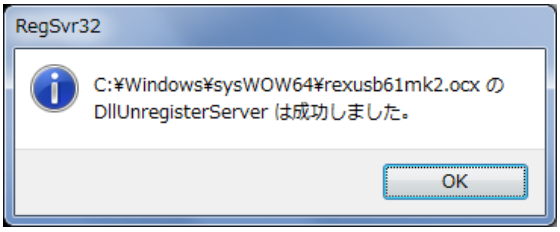
■ 32 ビット版 OS の場合

>regsvr32 /u rexusb61mk2.ocx

■ 64 ビット版 OS の場合

>regsvr32 /u C:¥Windows¥sysWOW64¥rexusb61mk2.ocx

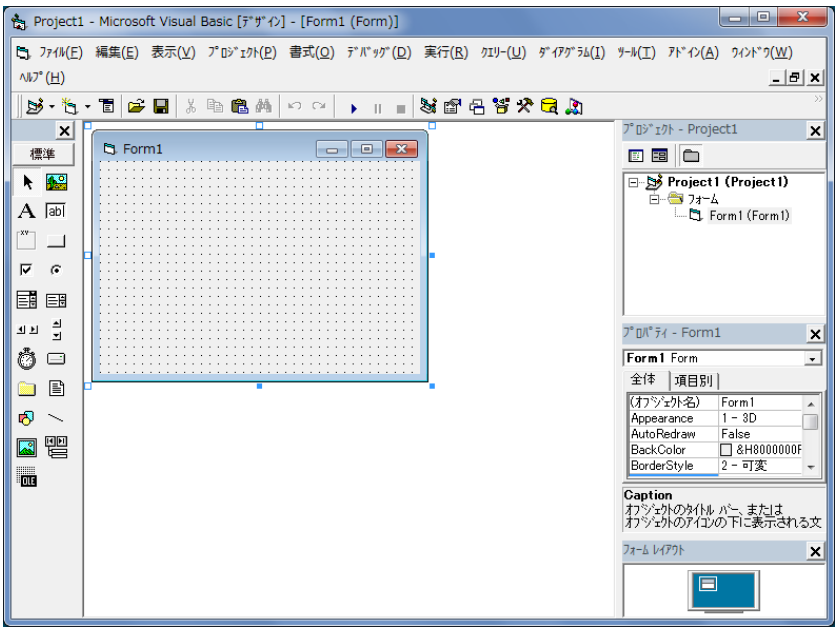
登録削除成功メッセージ(32 ビット版 OS)



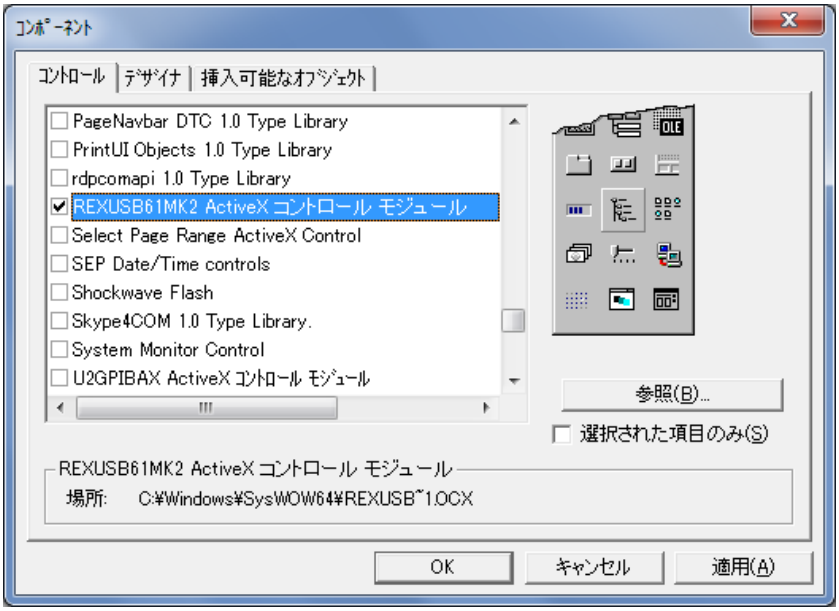
登録削除成功メッセージ(64 ビット版 OS)

(3) VB6 での ActiveX 参照方法

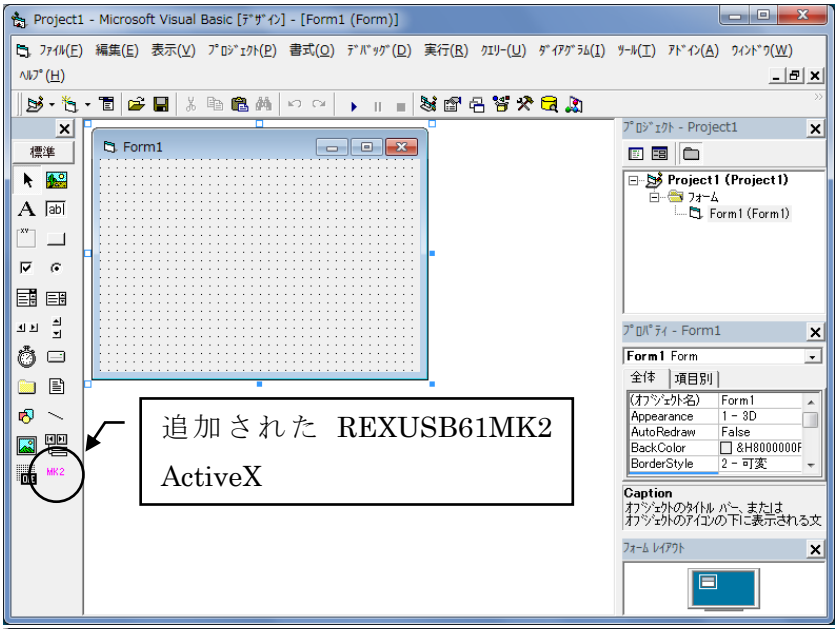
新しいプロジェクトを作成します。



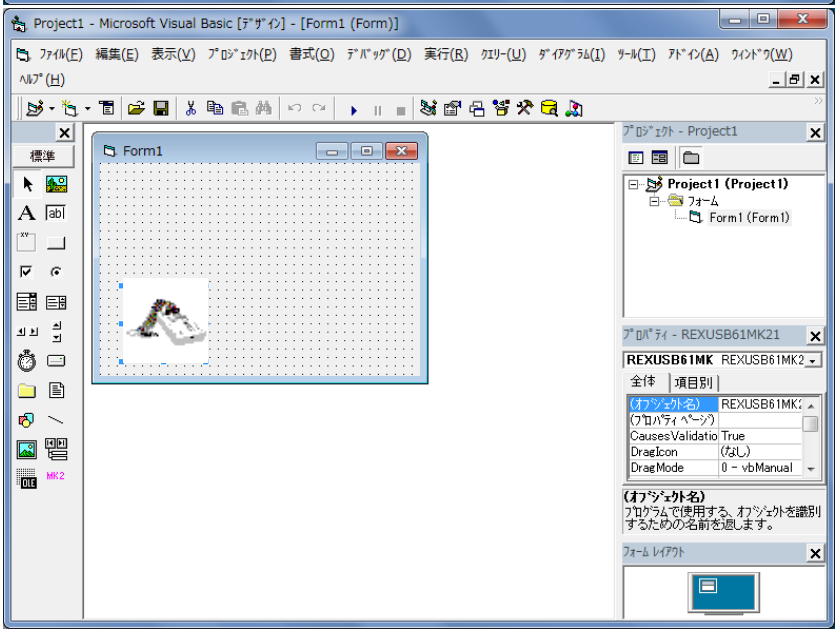
プロジェクトメニューのコンポーネントを選択します。コントロール一覧の、「 REXUSB61MK2 ActiveX コントロールモジュール」にチェックを入れて OK ボタンをクリックします。

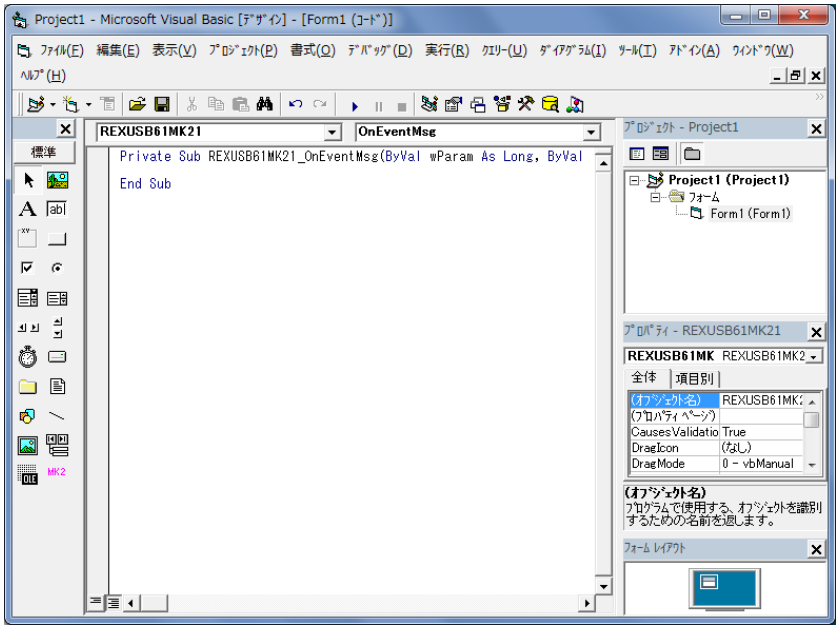


REXUSB61MK2 ActiveX コンポーネントが追加されます。



追加された usb61api ActiveX コンポーネントを選択し、フォームにオブジェクトを貼り付けます。
(追加直後のオブジェクト名は「REXUSB61MK21」となります。)
オブジェクトのプロパティ内の「Visible」を False にして、実行時表示されないようにしておきます。

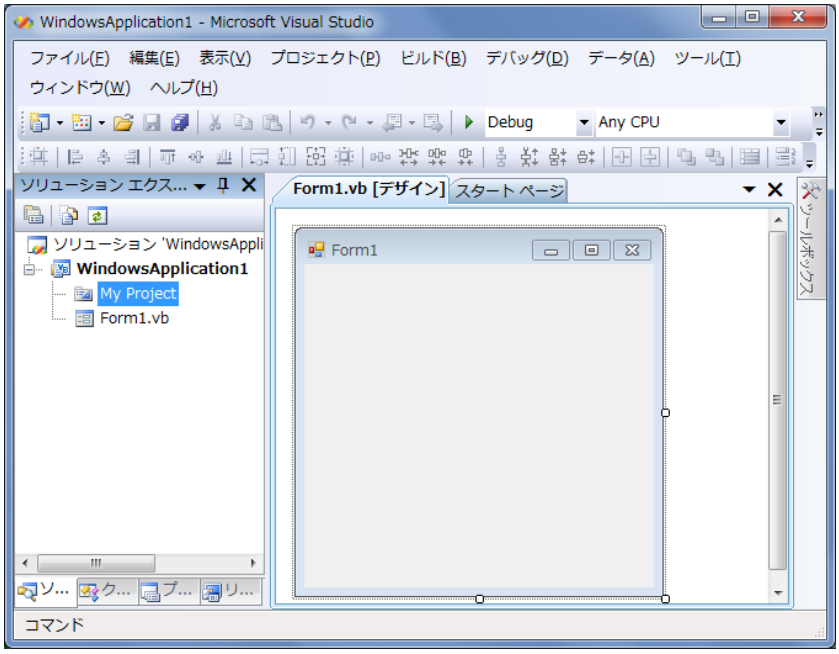




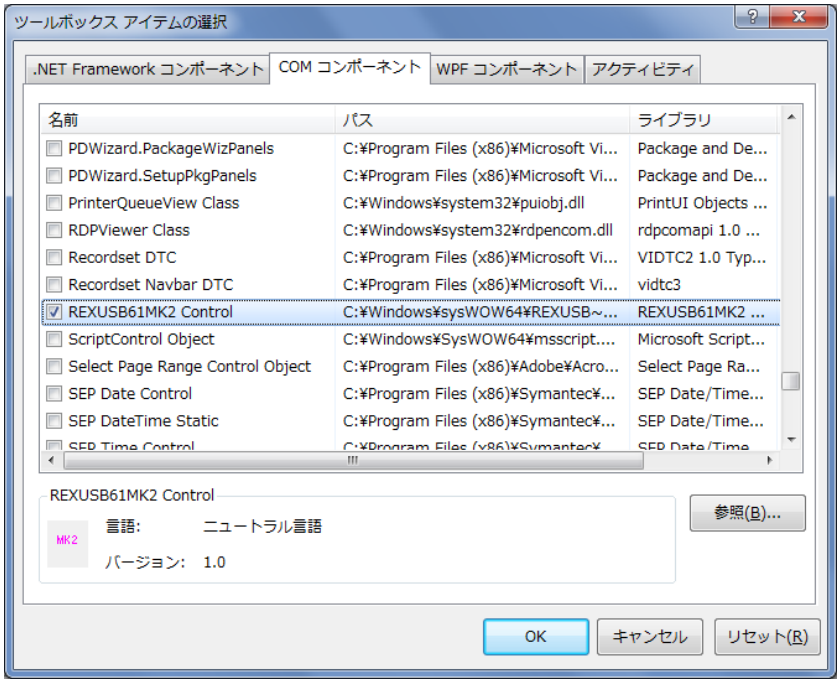
オブジェクトをダブルクリックすると、イベント発生時の呼び出されるサブルーチン
Sub REXUSB61MK21_OnEventMsg（...)が表示されます。
関数仕様の説明を参照願います。

(4) VB.NET / Visual C#での ActiveX 参照方法

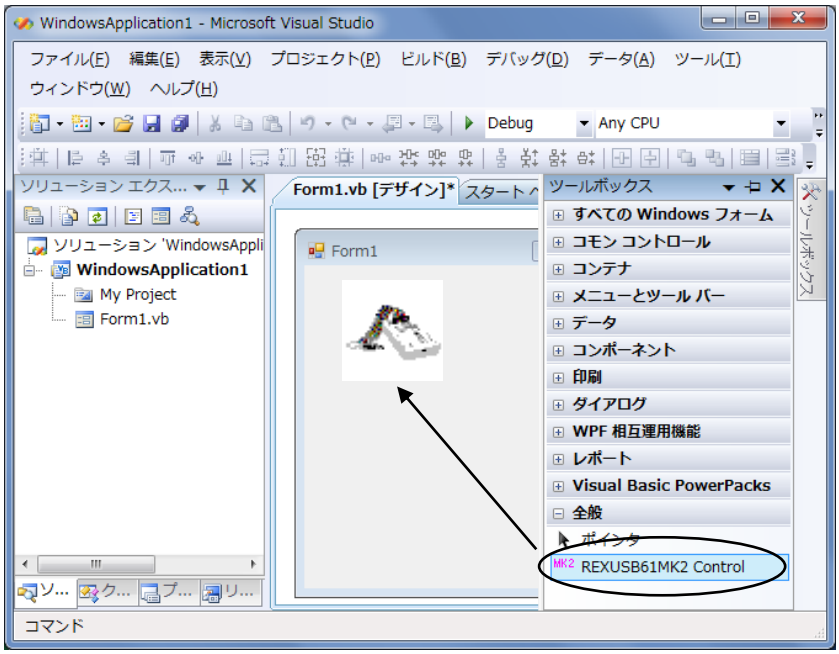
新しいプロジェクトを作成します。



メニューの「ツール」→「ツールボックスアイテムの選択」→「COM コンポーネント」を選択し、「REXUSB61MK2 Control」にチェックを入れOKボタンをクリックします。

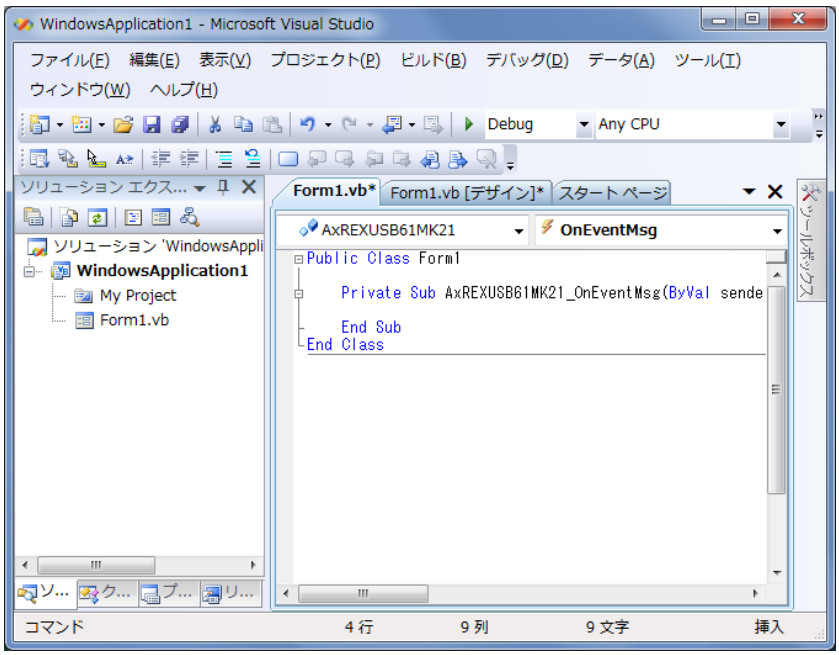


ツールボックスに登録されていることを確認し、フォームへ貼り付けてください。(追加直後のオブジェクト名は「REXUSB61MK21」となります。)



貼り付けたオブジェクトのプロパティ内の「Visible」を False にして、実行時表示されないようにしておきます。

オブジェクトをダブルクリックすると、イベント発生時の呼び出されるサブルーチン
Sub AxREXUSB61MK21_OnEventMsg ()が表示されます。
関数仕様の説明を参照願います。



(4-3) API 関数一覧

以下に，API 関数(VC)の一覧を示します。

表 4-1 API 関数一覧

関数名	機能
一般関数	
usb61mk2_init	デバイスの使用を開始する。
usb61mk2_get_state_device	装置の実行状態取得。
usb61mk2_get_buffer_size	バッファサイズ取得。
usb61mk2_get_regist_value	レジスタ値取得。
usb61mk2_get_state_hw	H/W 状態取得。
usb61mk2_get_version_fw	F/W バージョン取得。
usb61mk2_reset	本製品のリセット。
usb61mk2_get_fw_version	F/W・FPGA のバージョン取得。
usb61mk2_power_control	ターゲットデバイスへの電源供給を行う。
usb61mk2_mode_change	SPI／I2C の切り替えを行う。
usb61mk2_set_interval	データ転送時に送信バイト毎の送信間隔を設定する。
usb61mk2_get_hw_info	ハードウェアの設定を取得する。
usb61mk2_get_serial	シリアル番号を取得する。
usb61mk2_lock	アプリケーションでロックする。
usb61mk2_unlock	アプリケーションでアンロックする。
DIO 関数	
usb61mk2_set_dio_mode	DIO の IN/OUT,割込検知を設定する。
usb61mk2_get_dio_mode	DIO の IN/OUT,割込検知を取得する。
usb61mk2_set_dio	DIO の OUT。
usb61mk2_get_dio	DIO の IN。
usb61mk2_set_dio_chattering	DIO のチャタリング防止時間を設定する。
usb61mk2_get_dio_chattering	DIO のチャタリング防止時間を取得する。
usb61mk2_start_interrupt	割り込み信号の検出を待つ。
usb61mk2_stop_interrupt	割り込み信号の検出待ち状態を停止する。
I2C 関数	
usb61mk2_i2c_pullup	I2C バスの SCL, SDA ラインのプルアップを行う。
usb61mk2_i2c_bus_reset	I2C バスをリセットする。
usb61mk2_i2c_set_freq	I2C バスの動作周波数をセットする。
usb61mk2_i2c_get_freq_ex	I2C バスの設定した動作周波数の値と周波数の計算上の近似値を取得する。
usb61mk2_i2c_reset_device	I2C デバイスをリセットする。
I2C Master 関数	
usb61mk2_i2c_read_master	I2C マスターとしてデータのリードを行う。
usb61mk2_i2c_read_master_ex	I2C マスターとしてデータのリードを行う。
usb61mk2_i2c_write_master	I2C マスターとしてデータのライトを行う。
usb61mk2_i2c_send	I2C デバイスに任意のデータを送信する。
usb61mk2_i2c_read_master_config	I2C マスターの通信設定を取得する。
usb61mk2_i2c_write_master_config	I2C マスターの通信設定を行う。
usb61mk2_i2c_read_master_data	事前の設定でI2Cマスターとしてデータのリードを行う。
usb61mk2_i2c_write_master_data	事前の設定でI2Cマスターとしてデータのライトを行う。
usb61mk2_i2c_acknowledge	I2C デバイスへ Acknowledge を行う。

I2C Slave 関数	
usb61mk2_i2c_slave_start	I2C スレーブ開始。
usb61mk2_i2c_slave_end	I2C スレーブ終了。
usb61mk2_i2c_read_slave_config	I2C スレーブコンフィグ取得。
usb61mk2_i2c_write_slave_config	I2C スレーブコンフィグ設定。
usb61mk2_i2c_read_slave_data	I2C スレーブバッファメモリ取得。
usb61mk2_i2c_write_slave_data	I2C スレーブバッファメモリ設定。
SPI 関数	
usb61mk2_spi_set_freq	SPI バスの動作周波数をセットする。
usb61mk2_spi_get_freq	SPI バスの動作周波数を取得する。
usb61mk2_spi_get_freq_ex	SPI バスの設定した動作周波数の値と 周波数の計算上の近似値を取得する。
SPI Master 関数	
usb61mk2_spi_transmit_master	SPI マスターとしてデバイスへデータを ライトしリードしたデータを返す。
usb61mk2_spi_transmit_master_fast	SPI マスターとしてデバイスへデータを ライトしリードしたデータを返す。 (fast モード)
usb61mk2_spi_transmit_master_quad_fast	SPI マスターとしてデバイスへデータを ライトしリードしたデータを返す。 (Quad 用 fast モード)
usb61mk2_spi_read_master_config	SPI 通信設定取得。
usb61mk2_spi_write_master_config	SPI 通信設定。
usb61mk2_spi_transmit	usb61mk2_spi_write_master_config の 設定で通信。
usb61mk2_wait_readstatus	ステータスレジスタのクリアを待つ。
SPI Slave 関数	
usb61mk2_spi_slave_start	SPI スレーブ開始。
usb61mk2_spi_slave_end	SPI スレーブ終了。
usb61mk2_spi_read_slave_data	SPI スレーブバッファメモリ取得。
usb61mk2_spi_write_slave_data	SPI スレーブバッファメモリ設定。
usb61mk2_spi_read_slave_config	SPI スレーブコンフィグ取得。
usb61mk2_spi_write_slave_config	SPI スレーブコンフィグ設定。
usb61mk2_spi_read_slave_status	SPI スレーブステータス取得。
usb61mk2_spi_write_slave_status	SPI スレーブステータス設定。
usb61mk2_spi_get_slave_select	SS 端子のマニュアル制御を取得。
usb61mk2_spi_set_slave_select	SS 端子のマニュアル制御を設定。
スクリプト関数	
usb61mk2_set_reg	装置内で使用する変数レジスタへ値を設 定する。
usb61mk2_get_reg	装置内で使用する変数レジスタの値を取 得する。
usb61mk2_wait_process	装置内で指定した時間処理待ちを行う。
usb61mk2_wait_interrupt	装置内で割り込み監視を指定した DIO を 対象として, 割り込みを検知するまで待ち を行う。

表 4-2 エラーコード一覧

■ API で返されるエラーコード

エラーコード	値	内容
RS_SUCCESS	0	成功
RS_OTHER_ERROR	-1	その他のエラー
RS_INIT_ERROR	-2	Winusb.sys 初期化エラー
RS_CONTROL_ERROR	-3	コントロール転送エラー
RS_WRITE_ERROR	-4	バルクアウトエラー
RS_READ_ERROR	-5	バルクインエラー
RS_INT_ERROR	-6	インタラプトエラー
RS_OUT_OF_RANGE	-7	デバイス ID エラー
RS_TIMEOUT_ERROR	-8	タイムアウトエラー

■ ファームから返されるエラーコード

値	内容
01h	通信エラー
81h	パラメーターエラー
82h	受信タイムアウトエラー
83h	I2C：NAK 受信エラー
84h	I2C：アービットレーション Fail エラー
85h	I2C：通信エラー
86h	I2C：タイムアウトエラー
87h	I2C：電源供給エラー
91h	呼び出し関数のコマンド番号エラー
92h	呼び出し関数のリクエスト番号エラー
93h	呼び出し関数のシーケンス番号エラー
94h	分岐・繰り返し処理が対になっていない
95h	分岐・繰り返し処理のネスト数が超えている
96h	現在のモードで使用できない中間コード
97h	中間コードに対して送信データが不足している
98h	繰り返しの中にある処理がバッファをオーバーしている
A1h	SPI：ステータスエラー
A2h	SPI：タイムアウトエラー
A3h	SPI：電源供給エラー
A4h	SPI：コマンドパラメーターエラー
E1h	FPGA Update：ベリファイエラー（ファームアップ時)
E2h	FPGA Update：ファイルリードエラー(ファームアップ時)
E3h	FPGA Update：VME ファイルバージョンエラー(ファームアップ時)
E4h	FPGA Update：データファイルエラー(ファームアップ時)
E5h	FPGA Update：CRC エラー(ファームアップ時)

(4-4) API 関数詳細

以下に API 関数の詳細を示します。
構造体メンバの各設定値については、ヘッダファイルをご参照ください。
(VB/C#にて ActiveX を使用せず、DLL から直接ライブラリ関数を呼び出す場合は、関数定義ファイル RexUsb61mk2.bas/RexUsb61mk2.vb/RexUsb61mk2.cs をご使用ください。)

共通関数

関数	VC ➢	INT usb61mk2_init (HWND hwnd);
	VB ➢	Function Usb61mk2Init(ByVal hwnd As Long) As Long
	VB.NET➢ C#➢	Function Usb61mk2Init(ByVal hwnd As Integer) As Integer
機能	デバイスの使用を開始する。	
引数	[IN] hwnd	
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。(表 4-2 参照)	

<table>
<tr><td rowspan="3">関数</td><td>VC ➢</td><td>INT usb61mk2_get_state_device (UCHAR UnitID,
PUCHAR pData);</td></tr>
<tr><td>VB ➢</td><td>Function Usb61mk2GetStateDevice(ByVal UnitID As Byte,
pData) As Long</td></tr>
<tr><td>VB.NET➢
C#➢</td><td>Function Usb61mk2GetStateDevice(ByVal UnitID As Byte,
ByRef pData As Object) As Integer</td></tr>
<tr><td>機能</td><td colspan="2">装置の実行状態取得。</td></tr>
<tr><td>引数</td><td colspan="2">[IN] UnitID ：ユニット番号
[OUT] pData ：データ
8Byte で取得されます。1 バイト目から 8 バイト目までの内容が
下記オフセット 0～7 に対応します。
<table>
<tr><th>オフセット</th><th>意味</th></tr>
<tr><td>0</td><td>Bit0:送信用バッファの状態
0:空きがある 1:空きがない
Bit2:受信用バッファの状態
0:空きがある 1:空きがない
Bit4、Bit5：動作状態
0:停止中 1:一時停止中 2:動作中</td></tr>
<tr><td>1</td><td>Bit0-7:各ポート設定(DIO ポート)
1:出力 0:入力設定</td></tr>
<tr><td>2</td><td>Bit0-7:各ポート変化通知設定</td></tr>
<tr><td>3</td><td>Bit0-7:各ポート極性設定
1:High/Low 反転</td></tr>
<tr><td>4</td><td>Bit0-7:各ポート値
1:High 0:Low</td></tr>
<tr><td>5</td><td>割込通知の種類
0:Low レベル 1:High レベル 2:Low→High 変化
3:High→Low 変化</td></tr>
<tr><td>6</td><td>Reserved</td></tr>
<tr><td>7</td><td>Reserved</td></tr>
</table>
※残り Reserved</td></tr>
<tr><td>戻値</td><td colspan="2">関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）</td></tr>
</table>

<table>
<tr><td rowspan="3">関数</td><td>VC ➢</td><td>INT usb61mk2_get_buffer_size (UCHAR UnitID,
PUCHAR pData);</td></tr>
<tr><td>VB ➢</td><td>Function Usb61mk2GetBufferSize(ByVal UnitID As Byte, pData)
As Long</td></tr>
<tr><td>VB.NET➢
C#➢</td><td>Function Usb61mk2GetBufferSize(ByVal UnitID As Byte,
ByRef pData As Object) As Integer</td></tr>
<tr><td>機能</td><td colspan="2">バッファサイズ取得。</td></tr>
<tr><td>引数</td><td colspan="2">[IN] UnitID ：ユニット番号
[OUT] pData ：データ
8Byte で取得されます。1 バイト目から 4 バイト目までの内容が
下記オフセット 0～3 に対応します。
<table>
<tr><th>オフセット</th><th>意味</th></tr>
<tr><td>0</td><td>受信用バッファの最大サイズ(KB 単位)</td></tr>
<tr><td>1</td><td>受信用バッファの空きサイズ(KB 単位)</td></tr>
<tr><td>2</td><td>送信用バッファの最大サイズ(KB 単位)</td></tr>
<tr><td>3</td><td>送信用バッファの空きサイズ(KB 単位)</td></tr>
</table>
※残り Reserved</td></tr>
<tr><td>戻値</td><td colspan="2">関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）</td></tr>
</table>

<table>
<tr><td rowspan="3">関数</td><td>VC ➢</td><td>INT usb61mk2_get_regist_value (UCHAR UnitID,
PUCHAR pData);</td></tr>
<tr><td>VB ➢</td><td>Function Usb61mk2GetRegistValue(ByVal UnitID As Byte,
pData) As Long</td></tr>
<tr><td>VB.NET➢
C#➢</td><td>Function Usb61mk2GetRegistValue(ByVal UnitID As Byte,
ByRef pData As Object) As Integer</td></tr>
<tr><td>機能</td><td colspan="2">レジスタ値取得。</td></tr>
<tr><td>引数</td><td colspan="2">[IN] UnitID ：ユニット番号
[OUT] pData ：データ
24Byte で取得されます。1 バイト目から 17 バイト目までの内容
が下記オフセット 0～16 に対応します。
<table>
<tr><th>オフセット</th><th>意味</th></tr>
<tr><td>0</td><td>レジスタ No.0 の値</td></tr>
<tr><td>・・・</td><td>・・・</td></tr>
<tr><td>15</td><td>レジスタ No.15 の値</td></tr>
<tr><td>16</td><td>キャリーレジスタの値</td></tr>
</table>
※残り Reserved</td></tr>
<tr><td>戻値</td><td colspan="2">関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）</td></tr>
</table>

<table>
<tr><td rowspan="3">関数</td><td>VC ➢</td><td>INT usb61mk2_get_state_hw (UCHAR UnitID,
PUCHAR pData);</td></tr>
<tr><td>VB ➢</td><td>Function Usb61mk2GetStateHw(ByVal UnitID As Byte, pData)
As Long</td></tr>
<tr><td>VB.NET➢
C#➢</td><td>Function Usb61mk2GetStateHw(ByVal UnitID As Byte,
ByRef pData As Object) As Integer</td></tr>
<tr><td>機能</td><td colspan="2">H/W 状態取得。</td></tr>
<tr><td>引数</td><td colspan="2">[IN] UnitID ：ユニット番号
[OUT] pData ：データ
16Byte で取得されます。1 バイト目から 12 バイト目までの内容が下記オフセット 0～11 に対応します。
<table>
<tr><th>オフセット</th><th>意味</th></tr>
<tr><td>0</td><td>Bit0:モード
0:SPI 1:I2C
Bit1:実行モード
0:一括実行モード 1:順次実行モード</td></tr>
<tr><td>1</td><td>Bit0:電源供給
0:しない 1:する
Bit2、Bit3、Bit4:電源電圧
1:1.8V 2:2.5V 3:3.3V 4:5.0V</td></tr>
<tr><td>2</td><td>ターゲットデバイス電源電圧(MSB)</td></tr>
<tr><td>3</td><td>ターゲットデバイス電源電圧(LSB)</td></tr>
<tr><td>4</td><td>送信間隔(MSB)</td></tr>
<tr><td>5</td><td>送信間隔(LSB)</td></tr>
<tr><td>6</td><td>Reserved</td></tr>
<tr><td>7</td><td>Reserved</td></tr>
<tr><td>8</td><td>Frequency (MSB)</td></tr>
<tr><td>9</td><td>Frequency</td></tr>
<tr><td>10</td><td>Frequency</td></tr>
<tr><td>11</td><td>Frequency (LSB)</td></tr>
</table>
※残り Reserved</td></tr>
<tr><td>戻値</td><td colspan="2">関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）</td></tr>
</table>

<table>
<tr><td rowspan="3">関数</td><td>VC ➢</td><td>INT usb61mk2_get_version_fw (UCHAR UnitID,
PUCHAR pData);</td></tr>
<tr><td>VB ➢</td><td>Function Usb61mk2GetVersionFw(ByVal UnitID As Byte, pData)
As Long</td></tr>
<tr><td>VB.NET➢
C#➢</td><td>Function Usb61mk2GetVersionFw(ByVal UnitID As Byte,
ByRef pData As Object) As Integer</td></tr>
<tr><td>機能</td><td colspan="2">F/W バージョン取得。</td></tr>
<tr><td>引数</td><td colspan="2">[IN] UnitID ：ユニット番号
[OUT] pData ：データ
16Byte で取得されます。1 バイト目から 6 バイト目までの内容が下記オフセット 0～5 に対応します。
<table>
<tr><th>オフセット</th><th>意味</th></tr>
<tr><td>0</td><td>ファームウェアバージョン(Major Version)</td></tr>
<tr><td>1</td><td>ファームウェアバージョン(Minor Version)</td></tr>
<tr><td>2</td><td>ファームウェアバージョン(Debug Version)</td></tr>
<tr><td>4</td><td>FPGA バージョン(Major Version)</td></tr>
<tr><td>5</td><td>FPGA バージョン(Minor Version)</td></tr>
</table>
※残り Reserved</td></tr>
<tr><td>戻値</td><td colspan="2">関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）</td></tr>
</table>

<table>
<tr><td rowspan="3">関数</td><td>VC ➢</td><td>INT usb61mk2_reset (UCHAR UnitID);</td></tr>
<tr><td>VB ➢</td><td>Function Usb61mk2Reset(ByVal UnitID As Byte) As Long</td></tr>
<tr><td>VB.NET➢
C#➢</td><td>Function Usb61mk2Reset(ByVal UnitID As Byte) As Integer</td></tr>
<tr><td>機能</td><td colspan="2">本製品のリセットを行う。</td></tr>
<tr><td>引数</td><td colspan="2">[IN] UnitID ：ユニット番号</td></tr>
<tr><td>戻値</td><td colspan="2">関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）</td></tr>
</table>

関数	VC ➢	INT usb61mk2_get_fw_version (UCHAR UnitID, PUCHAR pFWMajorVer, PUCHAR pFWMinorVer, PUCHAR pFPGAMajorVer, PUCHAR pFPGAMinorVer);
	VB ➢	Function Usb61mk2GetFwVersion(ByVal UnitID As Byte, pFWMajorVer As Byte, pFWMinorVer As Byte, pFPGAMajorVer As Byte, pFPGAMinorVer As Byte) As Long
	VB.NET➢ C#➢	Function Usb61mk2GetFwVersion(ByVal UnitID As Byte, ByRef pFWMajorVer As Byte, ByRef pFWMinorVer As Byte, ByRef pFPGAMajorVer As Byte, ByRef pFPGAMinorVer As Byte) As Integer
機能	ファームウェア/FPGA のバージョン取得を行う。	
引数	[IN] UnitID ：ユニット番号 [OUT] pFWMajorVer: ファームウェアのメジャーバージョン [OUT] pFWMinorVer: ファームウェアのマイナーバージョン [OUT] pFPGAMajorVer: FPGA のメジャーバージョン [OUT] pFPGAMinorVer: FPGA のマイナーバージョン	
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_power_control (UCHAR UnitID, UINT fPowerState);
	VB ➢	Function Usb61mk2PowerControl(ByVal UnitID As Byte, ByVal fPowerState As Long) As Long
	VB.NET➢ C#➢	Function Usb61mk2PowerControl(ByVal UnitID As Byte, ByVal fPowerState As Integer) As Integer
機能	ターゲットデバイスへの電源供給を行う。	
引数	[IN] UnitID ：ユニット番号 [IN] fPowerState ：電源供給およびターゲットの電源電圧を指定する	

スイッチ	電圧	意味
RS_PWRCTRL_OFF		電源供給を行わない
RS_PWRCTRL_ON		電源供給を行う
	RS_PWRCTRL_1_8V	ターゲットの電源電圧 1.8V
	RS_PWRCTRL_2_5V	ターゲットの電源電圧 2.5V
	RS_PWRCTRL_3_3V	ターゲットの電源電圧 3.3V
	RS_PWRCTRL_5_0V	ターゲットの電源電圧 5.0V

戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）

関数	VC ➢	INT usb61mk2_mode_change (UCHAR UnitID, UINT fDeviceMode);
	VB ➢	Function Usb61mk2ModeChange(ByVal UnitID As Byte, ByVal fDeviceMode As Long) As Long
	VB.NET➢ C#➢	Function Usb61mk2ModeChange(ByVal UnitID As Byte, ByVal fDeviceMode As Integer) As Integer
機能	SPI／I2C の切り替えを行う。	
引数	[IN] UnitID ：ユニット番号 [IN] fDeviceMode ：デバイスモード設定フラグ RS_DEVMODE_SPI SPI モード RS_DEVMODE_I2C I2C モード	
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_set_interval (UCHAR UnitID, USHORT IntervalCnt);
	VB ➢	Function Usb61mk2SetInterval(ByVal UnitID As Byte, ByVal IntervalCnt As Long) As Long
	VB.NET➢ C#➢	Function Usb61mk2SetInterval(ByVal UnitID As Byte, ByVal IntervalCnt As Integer) As Integer
機能	データ転送時に送信バイト毎の送信間隔を設定する。	
引数	[IN] UnitID ：ユニット番号 [IN] IntervalCnt ：送信間隔を指定する(usec 指定)	
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_get_hw_info (UCHAR UnitID, PRS_HARDWARE_INFO pHardwareInfo);
	VB ➢	Function Usb61mk2GetHwInfo(ByVal UnitID As Byte, pHardwareInfo) As Long
	VB.NET➢	Function Usb61mk2GetHwInfo(ByVal UnitID As Byte, ByRef pHardwareInfo As Object) As Integer
機能	ハードウェアの設定を取得する。	

引数	[IN] UnitID ：ユニット番号
	[OUT] pHardwareInfo ：ハードウェア情報が格納される
	typedef struct _RS_HARDWARE_INFO {
	UCHAR DeviceMode; // SPI/I2Cデバイス
	UCHAR PowerCtrl; // 電源電圧
	WORD TargetPower; // ターゲット電源電圧
	WORD Interval; // 送信間隔
	LONG Frequency; // 周波数設定(1Hz 単位)
	UCHAR Sign1; // 動作周波数の調整値
	UCHAR Sign2; // 動作周波数の調整値
	UCHAR Sign3; // 動作周波数の調整値
	UCHAR Adjust1; // 動作周波数の調整値
	UCHAR Adjust2; // 動作周波数の調整値
	UCHAR Adjust3; // 動作周波数の調整値
	} RS_HARDWARE_INFO, *PRS_HARDWARE_INFO;	

戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

動作周波数の調整値について

これらの値は I2C 使用時のみ有効となります。

【Sign】-- 符号を設定する。0 がプラスで 1 がマイナス。

【Adjust】-- 0 から 15 の調整値を指定。

【Sign1】【Adjust1】-- Standard-mode/Fast-mode/Fast-mode Plus に対応。

【Sign2】【Adjust2】-- Hs-mode に対応。

【Sign3】【Adjust3】-- Ultra Fast-mode に対応。

-15～15 の調整値は、ベースとなる周波数の設定値に対しての割合となりますので、
設定される範囲の目安は下記になります。

100kHz : -3.6kHz ～ 2.2kHz

400kHz : -40kHz ～ 50kHz

1MHz : -120kHz～430kHz

3.4MHz(Hs-mode) : -1.5MHz ～ 18MHz

5.0MHz(UF-mode) : -3MHz ～ 50MHz

関数	VC ➢	INT usb61mk2_get_serial (UCHAR UnitID, WORD DataLength, UCHAR *pData);
	VB ➢	Function Usb61mk2GetSerial (ByVal UnitID As Byte, ByVal DataLength As Long, pData) As Long
	VB.NET➢	Function Usb61mk2GetSerial (ByVal UnitID As Byte, ByVal DataLength As Integer, ByRef pData As Object) As Integer
機能	製品のシリアル番号を取得する。	
引数	[IN] UnitID ：ユニット番号 [IN] DataLength ：データ長 [OUT] pData ：シリアル番号(データ長に 5Byte 以上を設定しても 5Byte まで)	
戻値	関数成功時は RS_SUCCESS を，失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_lock (UCHAR UnitID);
	VB ➢	Function Usb61mk2Lock (ByVal UnitID As Byte) As Long
	VB.NET➢	Function Usb61mk2Lock (ByVal UnitID As Byte) As Integer
機能	複数台対応のためアプリケーションでこの関数を呼び出しておくと、他のアプリケーションからこの関数を呼び出した場合に使用中であるかを確認することができる。	
引数	[IN] UnitID ：ユニット番号	
戻値	関数成功時は RS_SUCCESS を，失敗時はエラーコード返す。（表 4-2 参照） その他のエラー 1：Lock 済み 2：Lock 失敗 3：デバイスが存在しない 4：デバイスアクセス中	

関数	VC ➢	INT usb61mk2_unlock (UCHAR UnitID);
	VB ➢	Function Usb61mk2UnLock (ByVal UnitID As Byte) As Long
	VB.NET➢	Function Usb61mk2UnLock (ByVal UnitID As Byte) As Integer
機能	Usb61mk2_lock 関数でロックした状態を開放する。	
引数	[IN] UnitID ：ユニット番号	
戻値	関数成功時は RS_SUCCESS を，失敗時はエラーコード返す。（表 4-2 参照） その他のエラー 1：Lock していない	

DIO 関数

関数	VC ➢	INT usb61mk2_set_dio_mode (UCHAR UnitID, UCHAR bDir, UCHAR bInt, UCHAR bPol, UCHAR bTrg);
	VB ➢	Function Usb61mk2SetDioMode(ByVal UnitID As Byte, ByVal bDir As Byte, ByVal bInt As Byte, ByVal bPol As Byte, ByVal bTrg As Byte) As Long
	VB.NET➢ C#➢	Function Usb61mk2SetDioMode(ByVal UnitID As Byte, ByVal bDir As Byte, ByVal bInt As Byte, ByVal bPol As Byte, ByVal bTrg As Byte) As Integer
機能	DIO の IN/OUT、割込検知を設定する。	
引数	[IN] UnitID [IN] bDir [IN] bInt [IN] bPol [IN] bTrg 0:Low レベル 1:High レベル 2:Low→High 変化 3:High→Low 変化	：ユニット番号 ：1:出力 0:入力設定 ：入力検知設定(インターラプト通知) ：1 を設定したポートは High/Low が反対になる ：割込検知の種類
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_get_dio_mode (UCHAR UnitID, PUCHAR pDir, PUCHAR pInt, PUCHAR pPol, PUCHAR pTrg);
	VB ➢	Function Usb61mk2GetDioMode(ByVal UnitID As Byte, pDir As Byte, pInt As Byte, pPol As Byte, pTrg As Byte) As Long
	VB.NET➢ C#➢	Function Usb61mk2GetDioMode(ByVal UnitID As Byte, ByRef pDir As Byte, ByRef pInt As Byte, ByRef pPol As Byte, ByRef pTrg As Byte) As Integer
機能	usb61mk2_set_dio_mode で設定した「モード」を取得する。 (割り込みピンや入出力設定)	
引数	[IN] UnitID [OUT] pDir [OUT] pInt [OUT] pPol [OUT] pTrg 0:Low レベル 1:High レベル 2:Low→High 変化 3:High→Low 変化	：ユニット番号 ：1:出力 0:入力設定 ：入力検知設定(インターラプト通知) ：1 を設定したポートは High/Low が反対になる ：割込検知の種類
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_set_dio (UCHAR UnitID, UCHAR bData);
	VB ➢	Function Usb61mk2SetDio(ByVal UnitID As Byte, ByVal bData As Byte) As Long
	VB.NET➢ C#➢	Function Usb61mk2SetDio(ByVal UnitID As Byte, ByVal bData As Byte) As Integer
機能	DIO の出力データを書き込む。	
引数	[IN] UnitID ：ユニット番号 [IN] bData ：出力データ(8bit で出力 1:High 0:Low)	
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_get_dio (UCHAR UnitID, PUCHAR pData);
	VB ➢	Function Usb61mk2GetDio(ByVal UnitID As Byte, pData As Byte) As Long
	VB.NET➢ C#➢	Function Usb61mk2GetDio(ByVal UnitID As Byte, ByRef pData As Byte) As Integer
機能	DIO の入力データを読み込む。	
引数	[IN] UnitID ：ユニット番号 [OUT] pData ：入力データ(8bit で取得 1:High 0:Low)	
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_set_dio_chattering (UCHAR UnitID, SHORT wTime);
	VB ➢	Function Usb61mk2SetDioChattering(ByVal UnitID As Byte, ByVal wTime As Short) As Long
	VB.NET➢ C#➢	Function Usb61mk2SetDioChattering (ByVal UnitID As Byte, ByVal wTime As Short) As Integer
機能	DIO のチャタリング防止時間を設定する。	
引数	[IN] UnitID ：ユニット番号 [IN] wTime ：チャタリング防止時間(ms)	
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_get_dio_chattering (UCHAR UnitID, PSHORT pTime);
	VB ➢	Function Usb61mk2GetDioChattering (ByVal UnitID As Byte, pTime As Short) As Long
	VB.NET➢ C#➢	Function Usb61mk2GetDioChattering (ByVal UnitID As Byte, ByRef pTime As Short) As Integer
機能	DIO のチャタリング防止時間を取得する。	
引数	[IN] UnitID ：ユニット番号 [IN] pTime ：チャタリング防止時間(ms)	
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_start_interrupt (UCHAR UnitID);
	VB ➢	Function Usb61mk2StartInterrupt(ByVal UnitID As Byte) As Long
	VB.NET➢ C#➢	Function Usb61mk2StartInterrupt(ByVal UnitID As Byte) As Integer
機能	割り込み信号の検出待ちを開始する。 usb61mk2_set_dio_mode( )で設定した割り込み信号が検出されると ユーザー定義メッセージ内の処理(VC)、またはイベント内の処理(VB/C#)が実行され lParam には 0x00010000(DIO 割り込み)、wParam には UnitID が格納されます。	
引数	[IN] UnitID ：ユニット番号	
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INTusb61mk2_stop_interrupt (UCHAR UnitID);
	VB ➢	Function Usb61mk2StopInterrupt(ByVal UnitID As Byte) As Long
	VB.NET➢ C#➢	Function Usb61mk2StopInterrupt(ByVal UnitID As Byte) As Integer
機能	割り込み信号の検出待ち状態を停止する。	
引数	[IN] UnitID ：ユニット番号	
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

I2C 関数

関数	VC ➢	INT usb61mk2_i2c_pullup (UCHAR UnitID, RS_I2C_PULLUP fI2cPullup);
	VB ➢	Function Usb61mk2I2cPullup(ByVal UnitID As Byte, ByVal fI2cPullup As Byte) As Long
	VB.NET➢ C#➢	Function Usb61mk2I2cPullup(ByVal UnitID As Byte, ByVal fI2cPullup As Byte) As Integer
機能	I2C のプルアップ設定を行う。(SDA，SCL の各ピン)	
引数	[IN] UnitID ：ユニット番号 [IN] fI2cPullup ：プルアップ設定を指定する RS_I2C_PULLUP_DISABLE SCL，SDA ピンをプルアップしない RS_I2C_PULLUP_ENABLE SCL，SDA ピンをプルアップする	
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。(表 4-2 参照)	

関数	VC ➢	INT usb61mk2_i2c_bus_reset (UCHAR UnitID);
	VB ➢	Function Usb61mk2I2cBusReset(ByVal UnitID As Byte) As Long
	VB.NET➢ C#➢	Function Usb61mk2I2cBusReset(ByVal UnitID As Byte) As Integer
機能	I2C バスリセットを行う。	
引数	[IN] UnitID ：ユニット番号	
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。(表 4-2 参照)	

関数	VC ➢	INT usb61mk2_i2c_set_freq (UCHAR UnitID, LONG dwI2cFreq);
	VB ➢	Function Usb61mk2I2cSetFreq(ByVal UnitID As Byte, ByVal dwI2cFreq As Long) As Long
	VB.NET➢ C#➢	Function Usb61mk2I2cSetFreq(ByVal UnitID As Byte, ByVal dwI2cFreq As Integer) As Integer
機能	I2C インターフェイスの周波数を設定する。	
引数	[IN] UnitID ：ユニット番号 [IN] dwI2cFreq ：I2C バスの周波数を指定する (1Hz 単位)	
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。(表 4-2 参照)	

関数	VC ➢	INT usb61mk2_i2c_get_freq_ex(UCHAR UnitID, LONG* pI2cFreq, LONG* pI2cApproFreq);
	VB ➢	Function Usb61mk2I2cGetFreqEx(ByVal UnitID As Byte, pI2cFreq As Long,pI2cApproFreq As Long) As Long
	VB.NET➢ C#➢	Function Usb61mk2I2cGetFreqEx(ByVal UnitID As Byte, ByRef pI2cFreq As Integer,ByRef pI2cApproFreq As Integer) As Integer
機能	I2C インターフェイスの設定した周波数と計算上の動作近似値を取得する。	
引数	[IN] UnitID ：ユニット番号 [IN] dwI2cFreq ：I2C バスの設定した周波数を取得する(1Hz 単位) [IN] dwI2cApproFreq：I2C バスの計算上の動作近似値周波数を取得する (1Hz 単位)	
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_i2c_reset_device (UCHAR UnitID, UCHAR mode);
	VB ➢	Function Usb61mk2I2cResetDevice(ByVal UnitID As Byte, ByVal mode As Byte) As Long
	VB.NET➢ C#➢	Function Usb61mk2I2cResetDevice(ByVal UnitID As Byte, ByVal mode As Byte) As Integer
機能	I2C デバイスにリセットを行う。	
引数	[IN] UnitID ：ユニット番号 [IN] mode ：モード 0:動作なし 1: usb61mk2_i2c_bus_reset と同じ 2: ジェネラルバスリセット(00h,06h 送信) 3: 9 クロック後に Start,Stop のみ送信	
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_i2c_acknowledge (UCHAR UnitID, PRS_I2C_ACKNOWLEDGE_INFO pI2CAcknowledgeInfo);
	VB ➢	Function Usb61mk2I2cAcknowledge (ByVal UnitID As Byte, pI2CAcknowledgeInfo) As Long
	VB.NET➢ C#➢	Function Usb61mk2I2cAcknowledge (ByVal UnitID As Byte, ByRef pI2CAcknowledgeInfo As Object) As Integer
機能	I2C デバイスへの Write 実行完了待ちのために Acknowledge を行う。 ターゲットデバイスから ACK を受け取るか、指定した時間経過するまで待ちます。	

引数	[IN] UnitID ：ユニット番号
	[IN] pI2CAcknowledgeInfo ：I2C Acknowledge 情報が格納されている
	typedef struct _RS_I2C_ACKNOWLEDGE_INFO {
	UCHAR Status; // ステータス(0:ACK受信 1:WriteCycle時間満了)
	USHORT DeviceAddress; // デバイスアドレス
	UCHAR Addr10; // I2C デバイスが10 ビットアドレスであるか指定する
	ULONG WriteCycle; // ACK Pollingを行う時間をミリ秒単位で指定する
	USHORT Polling; // ACK Pollingを行う間隔をマイクロ秒単位で指定 最短は約10μ)
	UCHAR REG; // Acknowledge Polling の正常動作時の結果を格納するレジスタ番号
	} RS_I2C_ACKNOWLEDGE_INFO, *PRS_I2C_ACKNOWLEDGE_INFO;	

戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

I2C マスター関数

関数	VC ➢	INT usb61mk2_i2c_read_master (UCHAR UnitID, PRS_I2C_MASTER_INFO pI2CMasterInfo);
	VB ➢	Function Usb61mk2I2cReadMaster(ByVal UnitID As Byte, pI2CMasterInfo) As Long
	VB.NET➢ C#➢	Function Usb61mk2I2cReadMaster(ByVal UnitID As Byte, ByRef pI2CMasterInfo As Object) As Integer
機能	I2C マスターとしてデータのリードを行う。	

引数	[IN] UnitID ：ユニット番号
	[OUT] pI2CMasterInfo ：I2C Master 情報が格納されている
	typedef struct _RS_I2C_MASTER_INFO {
	USHORT DeviceAddress; // デバイスアドレス
	UCHAR Addr10; // I2C デバイスが10ビットアドレスであるか指定する
	UCHAR StopCondition; // ストップコンディションの有無を設定する
	UCHAR Speed; // 通信速度
	UCHAR REG; // データ値の参照元の指定
	UCHAR SubMode; // サブアドレスの送信を指定する
	UCHAR Sign; // 動作周波数の調整値
	UCHAR Adjust; // 動作周波数の調整値
	ULONG SubAddress; // サブアドレス
	ULONG DataBytes; // アクセスするバイト数
	UCHAR* pDataBuf; // アクセスしたしたデータ
	} RS_I2C_MASTER_INFO, *PRS_I2C_MASTER_INFO;	

戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

サブアドレスの指定について

- 1 バイトアドレス送信時のサブアドレス指定　0x01 を送信する場合
  SubMode = RS_I2C_SUBADRS_0
  SubAddress = &H01000000 ← 上位 1 バイトで指定
- 2 バイトアドレス送信時のサブアドレス指定　0x1234 を送信する場合
  SubMode = RS_I2C_SUBADRS_01
  SubAddress = &H12340000 ← 上位から 2 バイトで指定
- 3 バイトアドレス送信時のサブアドレス指定　0x123456 を送信する場合
  SubMode = RS_I2C_SUBADRS_012
  SubAddress = &H12345600 ← 上位から 3 バイトで指定
- 4 バイトアドレス送信時のサブアドレス指定　0x12345678 を送信する場合
  SubMode = RS_I2C_SUBADRS_0123
  SubAddress = &H12345678 ← 上位から 4 バイトで指定

動作周波数の調整値について

これらの値は I2C 使用時のみ有効となります。

【Sign】-- 符号を設定する。0 がプラスで 1 がマイナス。

【Adjust】-- 0 から 15 の調整値を指定。

-15～15 の調整値は、ベースとなる周波数の設定値に対しての割合となりますので、
設定される範囲の目安は下記になります。

100kHz : -3.6kHz ～ 2.2kHz

400kHz : -40kHz ～ 50kHz

1MHz : -120kHz～430kHz

3.4MHz(Hs-mode) : -1.5MHz ～ 18MHz

5.0MHz(UF-mode) : -3MHz ～ 50MHz

関数	VC ➢	INT usb61mk2_i2c_read_master_ex(UCHAR UnitID, PRS_I2C_MASTER_INFO pI2CWriteInfo, PRS_I2C_MASTER_INFO pI2CReadInfo);
	VB ➢	Function Usb61mk2I2cReadMasterEx(ByVal UnitID As Byte, pI2CWriteInfo, pI2CReadinfo) As Long
	VB.NET➢ C#➢	Function Usb61mk2I2cReadMasterEx(ByVal UnitID As Byte, ByRef pI2CWriteInfo As Object, ByRef pI2CReadInfo As Object) As Integer
機能	usb61mk2_i2c_write_master()と usb61mk2_i2c_read_master ()を一度に呼び出す。	

引数	[IN] UnitID : ユニット番号
	[IN] pI2CWriteInfo : I2C Master 情報が格納されている
	[OUT] pI2CReadInfo : I2C Master 情報が格納されている
	typedef struct _RS_I2C_MASTER_INFO {
	USHORT DeviceAddress; // デバイスアドレス
	UCHAR Addr10; // I2C デバイスが10ビットアドレスであるか指定する
	UCHAR StopCondition; // ストップコンディションの有無を設定する
	UCHAR Speed; // 通信速度
	UCHAR REG; // データ値の参照元の指定
	UCHAR SubMode; // サブアドレスの送信を指定する
	UCHAR Sign; // 動作周波数の調整値
	UCHAR Adjust; // 動作周波数の調整値
	ULONG SubAddress; // サブアドレス
	ULONG DataBytes; // アクセスするバイト数
	UCHAR* pDataBuf; // アクセスしたしたデータ
	} RS_I2C_MASTER_INFO, *PRS_I2C_MASTER_INFO;	

戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_i2c_write_master (UCHAR UnitID, PRS_I2C_MASTER_INFO pI2CMasterInfo);
	VB ➢	Function Usb61mk2I2cWriteMaster(ByVal UnitID As Byte, pI2CMasterInfo) As Long
	VB.NET➢ C#➢	Function Usb61mk2I2cWriteMaster(ByVal UnitID As Byte, ByRef pI2CMasterInfo As Object) As Integer
機能	I2C マスターとしてデータのライトを行う。	

引数	[IN] UnitID ：ユニット番号
	[IN] pI2CMasterInfo ：I2C Master 情報が格納されている
	typedef struct _RS_I2C_MASTER_INFO {
	USHORT DeviceAddress; // デバイスアドレス
	UCHAR Addr10; // I2C デバイスが10ビットアドレスであるか指定する
	UCHAR StopCondition; // ストップコンディションの有無を設定する
	UCHAR Speed; // 通信速度
	UCHAR REG; // データ値の参照元の指定
	UCHAR SubMode; // 未使用
	UCHAR Sign; // 動作周波数の調整値
	UCHAR Adjust; // 動作周波数の調整値
	ULONG SubAddress; // サブアドレス
	ULONG DataBytes; // アクセスするバイト数
	UCHAR* pDataBuf; // アクセスしたしたデータ
	} RS_I2C_MASTER_INFO, *PRS_I2C_MASTER_INFO;	

戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

動作周波数の調整値について

これらの値は I2C 使用時のみ有効となります。

【Sign】-- 符号を設定する。0 がプラスで 1 がマイナス。

【Adjust】-- 0 から 15 の調整値を指定。

-15～15 の調整値は、ベースとなる周波数の設定値に対しての割合となりますので、
設定される範囲の目安は下記になります。

100kHz : -3.6kHz ～ 2.2kHz

400kHz : -40kHz ～ 50kHz

1MHz : -120kHz～430kHz

3.4MHz(Hs-mode) : -1.5MHz ～ 18MHz

5.0MHz(UF-mode) : -3MHz ～ 50MHz

関数	VC ➢	INT usb61mk2_i2c_send (UCHAR UnitID, PRS_I2C_SEND_INFO pI2CSendInfo);
	VB ➢	Function Usb61mk2I2cSend(ByVal UnitID As Byte, pI2CSendInfo) As Long
	VB.NET➢ C#➢	Function Usb61mk2I2cSend(ByVal UnitID As Byte, ByRef pI2CSendInfo As Object) As Integer
機能	I2C デバイスに任意のデータを送信する。	

引数	[IN] UnitID ：ユニット番号
	[IN] pI2CSendInfo ：I2CSend 情報が格納されている
	typedef struct _RS_I2C_SEND _INFO {
	UCHAR StopCondition; // ストップコンディションの有無を設定する
	UCHAR Speed; // 通信速度
	UCHAR IgnoreNack; // Nackを発生してもエラーとせずに通信を継続する
	UCHAR REG; // データ値の参照元の指定
	UCHAR Sign; // 動作周波数の調整値
	UCHAR Adjust; // 動作周波数の調整値
	ULONG DataBytes; // アクセスするバイト数
	UCHAR* pDataBuf; // アクセスしたデータ
	} RS_I2C_SEND INFO, *PRS_I2C_SEND_INFO;	

戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

動作周波数の調整値について

これらの値は I2C 使用時のみ有効となります。

【Sign】-- 符号を設定する。0 がプラスで 1 がマイナス。

【Adjust】-- 0 から 15 の調整値を指定。

-15～15 の調整値は、ベースとなる周波数の設定値に対しての割合となりますので、
設定される範囲の目安は下記になります。

100kHz : -3.6kHz ～ 2.2kHz

400kHz : -40kHz ～ 50kHz

1MHz : -120kHz～430kHz

3.4MHz(Hs-mode) : -1.5MHz ～ 18MHz

5.0MHz(UF-mode) : -3MHz ～ 50MHz

関数	VC ➢	INT usb61mk2_i2c_read_master_config (UCHAR UnitID, PRS_I2C_MASTER_INFO pI2CMasterInfo);
	VB ➢	Function Usb61mk2I2cReadMasterConfig(ByVal UnitID As Byte, pI2CMasterInfo) As Long
	VB.NET➢ C#➢	Function Usb61mk2I2cReadMasterConfig(ByVal UnitID As Byte, ByRef pI2CMasterInfo As Object) As Integer
機能	I2C マスターの通信設定を取得する。	
引数	[IN] UnitID ：ユニット番号 [OUT] pI2CMasterInfo ：I2C Master 情報が格納されている	
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_i2c_write_master_config (UCHAR UnitID, UCHAR fDevAdrs, UCHAR fAdrs10,UCHAR fSpeed, PRS_I2C_MASTER_INFO pI2CMasterInfo);
	VB ➢	Function Usb61mk2I2cWriteMasterConfig(ByVal UnitID As Byte, ByVal fDevAdrs As Byte,ByVal fAdrs10 As Byte, ByVal fSpeed As Byte, pI2CMasterInfo) As Long
	VB.NET➢ C#➢	Function Usb61mk2I2cWriteMasterConfig(ByVal UnitID As Byte, ByVal fDevAdrs As Byte,ByVal fAdrs10 As Byte, ByVal fSpeed As Byte, ByRef pI2CMasterInfo As Object) As Integer
機能	I2C マスターの通信設定を取得する。	
引数	[IN] UnitID ：ユニット番号 [IN] fDevAdrs ：I2C Master 情報 DeviceAddress パラメータ変更(0:しない、1：する) [IN] fAdrs10 ：I2C Master 情報 Addr10 パラメータ変更(0:しない、1：する) [IN] fSpeed ：I2C Master 情報 Speed パラメータ変更(0:しない、1：する) [IN] pI2CMasterInfo ：I2C Master 情報が格納されている	
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_i2c_read_master_data (UCHAR UnitID, PRS_I2C_MASTER_INFO pI2CMasterInfo);
	VB ➢	Function Usb61mk2I2cReadMasterData (ByVal UnitID As Byte, pI2CMasterInfo) As Long
	VB.NET➢ C#➢	Function Usb61mk2I2cReadMasterData (ByVal UnitID As Byte, ByRef pI2CMasterInfo As Object) As Integer
機能	事前の設定で I2C マスターとしてデバイスからデータを取得する。	
引数	[IN] UnitID ：ユニット番号 [OUT] pI2CMasterInfo ：I2C Master 情報が格納されている	
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_i2c_write_master_data (UCHAR UnitID, PRS_I2C_MASTER_INFO pI2CMasterInfo);
	VB ➢	Function Usb61mk2I2cWriteMasterData (ByVal UnitID As Byte, pI2CMasterInfo) As Long
	VB.NET➢ C#➢	Function Usb61mk2I2cWriteMasterData (ByVal UnitID As Byte, ByRef pI2CMasterInfo As Object) As Integer
機能	事前の設定で I2C マスターとしてデバイスにデータを設定する。	
引数	[IN] UnitID ：ユニット番号 [OUT] pI2CMasterInfo ：I2C Master 情報が格納されている	
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

I2C スレーブ関数

関数	VC ➢	INT usb61mk2_i2c_slave_start (UCHAR UnitID);
	VB ➢	Function Usb61mk2I2cSlaveStart(ByVal UnitID As Byte) As Long
	VB.NET➢ C#➢	Function Usb61mk2I2cSlaveStart(ByVal UnitID As Byte) As Integer
機能	I2C スレーブ開始。	
引数	[IN] UnitID ：ユニット番号 [IN] I2CSlaveInfo ：I2C Slave 情報が格納されている	
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_i2c_slave_end (UCHAR UnitID);
	VB ➢	Function Usb61mk2I2cSlaveEnd(ByVal UnitID As Byte) As Long
	VB.NET➢ C#➢	Function Usb61mk2I2cSlaveEnd(ByVal UnitID As Byte) As Integer
機能	I2C スレーブ終了。	
引数	[IN] UnitID ：ユニット番号	
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_i2c_read_slave_config(UCHAR UnitID, UCHAR Channel, PRS_I2C_SLAVE_INFO pI2CSlaveInfo);
	VB ➢	Function Usb61mk2I2cReadSlaveConfig(ByVal UnitID As Byte, ByVal Channel As Byte, pI2CSlaveInfo) As Long
	VB.NET➢ C#➢	Function Usb61mk2I2cReadSlaveConfig(ByVal UnitID As Byte, ByVal channel As Byte, ByRef pI2CSlaveInfo As Object) As Integer
機能	I2C スレーブコンフィグ取得。	

引数	[IN] UnitID ：ユニット番号
	[IN] Channel ：指定チャンネル(0-3)
	[OUT] pI2CSlaveInfo ：I2C Slave 情報が格納されている
	typedef struct _RS_I2C_SLAVE _INFO {
	USHORT DeviceAddress; // デバイスアドレス
	UCHAR Addr10; // I2C デバイスが 10 ビットアドレスである
	か指定する
	UCHAR SubAdrsSize; // サブアドレスのバイト数
	UCHAR Nack; // NACK を返すアドレス機能を指定する
	UCHAR Clock; // クロックストレッチ動作を行うアドレス
	機能を指定する
	UCHAR Channel; // チャンネル
	UCHAR Ring; // メモリ領域をリングバッファに使用
	UCHAR Ringp; // ページ領域をリングバッファに使用
	UCHAR EN; // 指定したチャンネルの設定
	UCHAR ClockTime; // クロックストレッチ時間（1～255ms）
	UCHAR PageSize; // ページサイズ
	UCHAR WriteCycle; // ライトサイクル
	ULONG NackAddress; // Nack アドレス
	ULONG NackAddressMask; // Nack アドレスマスク
	ULONG ClockAddress; // Clock Stretch アドレス
	ULONG ClockAddressMask; // Clock Stretch アドレスマスク
	ULONG MemoryAddress; // メモリアドレス
	ULONG MemoryAddressMask; // メモリアドレスマスク
	} RS_I2C_SLAVE INFO, *PRS_I2C_SLAVE_INFO;	

戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_i2c_write_slave_config (UCHAR UnitID, UCHAR Channel, PRS_I2C_SLAVE_INFO pI2CSlaveInfo);
	VB ➢	Function Usb61mk2I2cWriteSlaveConfig(ByVal UnitID As Byte, ByVal Channel As Byte, ByVal pI2CSlaveInfo) As Long
	VB.NET➢ C#➢	Function Usb61mk2I2cWriteSlaveConfig(ByVal UnitID As Byte, ByVal channel As Byte, ByVal pI2CSlaveInfo As Object) As Integer
機能	I2C スレーブコンフィグ設定。	
引数	[IN] UnitID ：ユニット番号 [IN] Channel ：指定チャンネル(0-3) [In] pI2CSlaveInfo ：I2C Slave 情報が格納されている	
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。(表 4-2 参照)	

関数	VC ➢	INT usb61mk2_i2c_read_slave_data (UCHAR UnitID, UCHAR Channel,USHORT Offset, USHORT DataLength, PUCHAR pData);
	VB ➢	Function Usb61mk2I2cReadSlaveData(ByVal UnitID As Byte, ByVal Channel As Byte, ByVal Offset As Long, ByVal DataLength As Long, pData) As Long
	VB.NET➢ C#➢	Function Usb61mk2I2cReadSlaveData(ByVal UnitID As Byte, ByVal channel As Byte, ByVal Offset As Integer, ByVal DataLength As Integer, ByRef pData As Object) As Integer
機能	I2C スレーブバッファメモリ取得。	
引数	[IN] UnitID ：ユニット番号 [IN] Channel ：チャンネル [IN] Offset ：オフセット [IN] DataLength ：データ長 [OUT] pData ：データ	
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。(表 4-2 参照)	

関数	VC ➢	INT usb61mk2_i2c_write_slave_data (UCHAR UnitID, UCHAR Channel,USHORT Offset, USHORT DataLength, PUCHAR pData);
	VB ➢	Function Usb61mk2I2cWriteSlaveData(ByVal UnitID As Byte, ByVal Channel As Byte, ByVal Offset As Long, ByVal DataLength As Long, ByVal pData) As Long
	VB.NET➢ C#➢	Function Usb61mk2I2cWriteSlaveData(ByVal UnitID As Byte, ByVal Channel As Byte, ByVal Offset As Integer, ByVal DataLength As Integer, ByVal pData As Object) As Integer
機能	I2C スレーブバッファメモリ設定。	
引数	[IN] UnitID [IN] Channel [IN] Offset [IN] DataLength [IN] pData	：ユニット番号 ：チャンネル ：オフセット ：データ長 ：データ
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

SPI 関数

関数	VC ➢	INT usb61mk2_spi_set_freq(UCHAR UnitID, PRS_SPI_INFO pSPIInfo);
	VB ➢	Function Usb61mk2SpiSetFreq(ByVal UnitID As Byte, ByVal pSPIInfo) As Long
	VB.NET➢ C#➢	Function Usb61mk2SpiSetFreq(ByVal UnitID As Byte, ByVal pSPIInfo As Object) As Integer
機能	SPI バスの動作周波数をセットする。	

引数	[IN] UnitID ：ユニット番号
	[IN] pSPIInfo ：SPI 情報が格納されている
	typedef struct _RS_SPI_INFO {
	UCHAR SampTiming; // サンプリングタイミングの設定
	UCHAR FB; // データオーダー設定
	UCHAR SS_A; // 信号論理
	UCHAR SS_Dis; // 同期設定
	LONG dwSPIFreq; // 設定周波数(1Hz単位)
	} RS_SPI_INFO, *PRS_SPI_INFO;	

戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_spi_get_freq(UCHAR UnitID, PRS_SPI_INFO pSPIInfo);
	VB ➢	Function Usb61mk2SpiGetFreq(ByVal UnitID As Byte, pSPIInfo) As Long
	VB.NET➢ C#➢	Function Usb61mk2SpiGetFreq(ByVal UnitID As Byte, ByRef pSPIInfo As Object) As Integer
機能	SPI バスの動作周波数を取得する。	

引数	[IN] UnitID ：ユニット番号
	[OUT] pSPIInfo ：SPI 情報が格納されている
	typedef struct _RS_SPI_INFO {
	UCHAR SampTiming; // サンプリングタイミングの設定
	UCHAR FB; // データオーダー設定
	UCHAR SS_A; // 信号論理
	UCHAR SS_Dis; // 同期設定
	LONG dwSPIFreq; // 設定周波数(1Hz単位)
	} RS_SPI_INFO, *PRS_SPI_INFO;	

戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_spi_get_freq_ex(UCHAR UnitID, LONG* pSpiFreq, LONG* pSpiApproFreq);
	VB ➢	Function Usb61mk2SpiGetFreqEx(ByVal UnitID As Byte, pSpiFreq As Long,pSpiApproFreq As Long) As Long
	VB.NET➢ C#➢	Function Usb61mk2SpiGetFreqEx(ByVal UnitID As Byte, ByRef pSpiFreq As Integer,ByRef pSpiApproFreq As Integer) As Integer
機能	SPI バスの設定した周波数と計算上の動作近似値を取得する。	
引数	[IN] UnitID [IN] dwSpiFreq [IN] dwSpiApproFreq	：ユニット番号 ：SPI バスの設定した周波数を取得する(1Hz 単位) ：SPI バスの計算上の動作近似値周波数を取得する (1Hz 単位)
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_wait_readstatus (UCHAR UnitID, BYTE RDSR, BYTE bit, BYTE Value, DWORD dwTime);
	VB ➢	Function Usb61mk2WaitReadStatus(ByVal UnitID As Byte, ByVal RDSR As Byte, ByVal bit As Byte, ByVal Value As Byte, ByVal dwTime As Long) As Long
	VB.NET➢ C#➢	Function Usb61mk2WaitReadStatus(ByVal UnitID As Byte, ByVal RDSR As Byte, ByVal bit As Byte, ByVal Value As Byte, ByVal dwTime As Integer) As Integer
機能	ReadStatusRegister がクリアされるのを待つ。	
引数	[IN] UnitID [IN] RDSR [IN] bit [IN] Value [IN] dwTime	：ユニット番号 ：ReadStatuaRegister ：ReadStatuaRegister の RDY 監視 bit 番号 ：ReadStatuaRegister の RDY の値 ：タイムアウトエラー時間(1ms 単位)
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

SPI マスター関数

関数	VC ➢	INT usb61mk2_spi_transmit_master(UCHAR UnitID, PRS_SPI_TRANSFER_INFO pSPITransferInfo);
	VB ➢	Function Usb61mk2SpiTransmitMaster(ByVal UnitID As Byte, pSPITransferInfo) As Long
	VB.NET➢ C#➢	Function Usb61mk2SpiTransmitMaster(ByVal UnitID As Byte, ByRef pSPITransferInfo As Object) As Integer
機能	SPI マスターとしてデバイスへデータをライトしリードしたデータを返す。	

引数	[IN] UnitID ：ユニット番号
	[IN/OUT] pSPITransferInfo ：SPI_TRANSFER 情報が格納されている
	typedef struct _RS_SPI_TRANSFER_INFO {
	UCHAR MultiIO; // Multi I/O を指定する
	// 無効を指定した場合は，Single-SPI 動作とする
	UCHAR Read; // 転送したデータに対するRead Data を返すか指定する
	UCHAR SS; // スレーブセレクト専用PIN の制御を行う
	// 転送開始前にセット（Low Level）
	// 転送完了後にリセット（High Level）を行う
	UCHAR REG; // データ値の参照元の指定
	UCHAR Transbit; // 1byte 未満の転送するビット数を指定する
	// 0の場合は8bit 転送になる
	ULONG WriteBytes; // アクセスするバイト数
	ULONG ReadBytes; // アクセスするバイト数
	UCHAR* pWriteDataBuf; // アクセスしたデータ
	UCHAR* pReadDataBuf; // アクセスしたデータ
	} RS_SPI_TRANSFER_INFO, *PRS_SP_TRANSFERI_INFO;	

戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_spi_read_master_config (UCHAR UnitID, PRS_SPI_TRANSFER_INFO pSPITransferInfo);
	VB ➢	Function Usb61mk2SpiReadMasterConfig(ByVal UnitID As Byte, pSPITransferInfo) As Long
	VB.NET➢ C#➢	Function Usb61mk2SpiReadMasterConfig(ByVal UnitID As Byte, ByRef pSPITransferInfo As Object) As Integer
機能	SPI マスターの通信設定を取得する。	
引数	[IN] UnitID ：ユニット番号 [OUT] pSPITransferInfo ：SPI_TRANSFER 情報が格納されている	
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_spi_write_master_config (UCHAR UnitID, UCHAR fMulti, UCHAR fTransbit, PRS_SPI_TRANSFER_INFO pSPITransferInfo);
	VB ➢	Function Usb61mk2SpiWriteMasterConfig (ByVal UnitID As Byte, ByVal fMulti As Byte, ByVal fTransbit As Byte, pSPITransferInfo) As Long
	VB.NET➢ C#➢	Function Usb61mk2SpiWriteMasterConfig (ByVal UnitID As Byte, ByVal fMulti As Byte, ByVal fTransbit As Byte, ByRef pSPITransferInfo As Object) As Integer
機能	SPI マスターの通信設定。	
引数	[IN] UnitID ：ユニット番号 [IN] fMulti ：MultiIO パラメータの変更 0:しない 1:する [IN] fTransbit ：Transbit パラメータの変更 0:しない 1:する [IN] pSPITransferInfo ：SPI_TRANSFER 情報が格納されている	
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_spi_transmit (UCHAR UnitID, PRS_SPI_TRANSFER_INFO pSPITransferInfo);
	VB ➢	Function Usb61mk2SpiTransmit (ByVal UnitID As Byte, pSPITransferInfo) As Long
	VB.NET➢ C#➢	Function Usb61mk2SpiTransmit (ByVal UnitID As Byte, ByRef pSPITransferInfo As Object) As Integer
機能	usb61mk2_spi_write_master_config の設定で通信。	
引数	[IN] UnitID ：ユニット番号 [IN/OUT] pSPITransferInfo ：SPI_TRANSFER 情報が格納されている	
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_spi_transmit_master_fast(UCHAR UnitID, PRS_SPI_TRANSFER_INFO pWRENInfo, PRS_SPI_TRANSFER_INFO pSPITransferInfo, BYTE RDSR, BYTE bit, BYTE Value, DWORD dwTime);
	VB ➢	Function Usb61mk2SpiTransmitMasterFast (ByVal UnitID As Byte, ByVal pWRENInfo, pSPITransferInfo, ByVal RDSR As Byte, ByVal bit As Byte, ByVal Value As Byte, ByVal dwTime As Long) As Long
	VB.NET➢ C#➢	Function Usb61mk2SpiTransmitMasterFast (ByVal UnitID As Byte, ByVal pWRENInfo As Object, ByRef pSPITransferInfo As Object, ByVal RDSR As Byte, ByVal bit As Byte, ByVal Value As Byte, ByVal dwTime As Integer) As Integer
機能	SPI マスターとしてデバイスへデータをライトしリードしたデータを返す。 (fast モード)	
引数	[IN] UnitID [IN] pWRENInfo [IN/OUT] pSPITransferInfo [IN] RDSR [IN] bit [IN] Value [IN] dwTime	：ユニット番号 ：SPI_TRANSFER 情報が格納されている (WriteEnable 用) ：SPI_TRANSFER 情報が格納されている ：ReadStatuaRegister ：ReadStatuaRegister の RDY 監視 bit 番号 ：ReadStatuaRegister の RDY の値 ：タイムアウトエラー時間(1ms 単位)
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_spi_transmit_master_quad_fast(UCHAR UnitID, PRS_SPI_TRANSFER_INFO pWRENInfo, PRS_SPI_TRANSFER_INFO pSPITransferInfo,BYTE RDSR, BYTE bit, BYTE Value, DWORD dwTime);
	VB ➢	Function Usb61mk2SpiTransmitMasterQuadFast (ByVal UnitID As Byte, ByVal pWRENInfo, pSPITransferInfo, ByVal RDSR As Byte, ByVal bit As Byte, ByVal Value As Byte, ByVal dwTime As Long) As Long
	VB.NET➢ C#➢	Function Usb61mk2SpiTransmitMasterQuadFast (ByVal UnitID As Byte, ByVal pWRENInfo As Object, ByRef pSPITransferInfo As Object, ByVal RDSR As Byte, ByVal bit As Byte, ByVal Value As Byte, ByVal dwTime As Integer) As Integer
機能	SPI マスターとしてデバイスへデータをライトしリードしたデータを返す。 (Quad 用 fast モード)	
引数	[IN] UnitID [IN] pWRENInfo [IN/OUT] pSPITransferInfo [IN] RDSR [IN] bit [IN] Value [IN] dwTime	：ユニット番号 ：SPI_TRANSFER 情報が格納されている (WriteEnable 用) ：SPI_TRANSFER 情報が格納されている ：ReadStatuaRegister ：ReadStatuaRegister の RDY 監視 bit 番号 ：ReadStatuaRegister の RDY の値 ：タイムアウトエラー時間(1ms 単位)
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

SPI スレーブ関数

関数	VC ➢	INT usb61mk2_spi_slave_start (UCHAR UnitID);
	VB ➢	Function Usb61mk2SpiSlaveStart(ByVal UnitID As Byte) As Long
	VB.NET➢ C#➢	Function Usb61mk2SpiSlaveStart(ByVal UnitID As Byte) As Integer
機能	SPI スレーブ開始。	
引数	[IN] UnitID ：ユニット番号	
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_spi_slave_end (UCHAR UnitID);
	VB ➢	Function Usb61mk2SpiSlaveEnd(ByVal UnitID As Byte) As Long
	VB.NET➢ C#➢	Function Usb61mk2SpiSlaveEnd(ByVal UnitID As Byte) As Integer
機能	SPI スレーブ終了。	
引数	[IN] UnitID ：ユニット番号	
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_spi_read_slave_data (UCHAR UnitID, UCHAR Channel,USHORT Offset, USHORT DataLength, PUCHAR pData);
	VB ➢	Function Usb61mk2SpiReadSlaveData(ByVal UnitID As Byte, ByVal Channel As Byte, ByVal Offset As Long, ByVal DataLength As Long, pData) As Long
	VB.NET➢ C#➢	Function Usb61mk2SpiReadSlaveData(ByVal UnitID As Byte, ByVal Channel As Byte, ByVal Offset As Integer, ByVal DataLength As Integer, ByRef pData As Object) As Integer
機能	SPI スレーブバッファメモリ取得。	
引数	[IN] UnitID ：ユニット番号 [IN] Channel ：チャンネル [IN] Offset ：オフセット [IN] DataLength ：データ長 [OUT] pData ：データ	
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_spi_write_slave_data (UCHAR UnitID, UCHAR Channel,USHORT Offset, USHORT DataLength, PUCHAR pData);
	VB ➢	Function Usb61mk2SpiWriteSlaveData(ByVal UnitID As Byte, ByVal Channel As Byte, ByVal Offset As Long, ByVal DataLength As Long, ByVal pData) As Long
	VB.NET➢ C#➢	Function Usb61mk2SpiWriteSlaveData(ByVal UnitID As Byte, ByVal Channel As Byte, ByVal Offset As Integer, ByVal DataLength As Integer, ByVal pData As Object) As Integer
機能	SPI スレーブバッファメモリ設定。	
引数	[IN] UnitID ：ユニット番号 [IN] Channel ：チャンネル [IN] Offset ：オフセット [IN] DataLength ：データ長 [IN] pData ：データ	
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_spi_read_slave_config (UCHAR UnitID, UCHAR Channel, PRS_SPI_SLAVE_INFO pSlaveInfo);
	VB ➢	Function Usb61mk2SpiReadSlaveConfig(ByVal UnitID As Byte, ByVal Channel As Byte, pSPISlaveInfo) As Long
	VB.NET➢ C#➢	Function Usb61mk2SpiReadSlaveConfig(ByVal UnitID As Byte, ByVal Channel As Byte, ByRef pSPISlaveInfo As Object) As Integer
機能	SPI スレーブバッファメモリ取得。	

引数	[IN] UnitID ：ユニット番号
	[IN] Channel ：スレーブチャンネル(0-1)
	[OUT] pSlaveInfo ：SPI Slave 情報が格納されている
	typedef struct _RS_SPI_SLAVE_INFO {
	UCHAR Channel; // チャンネル
	UCHAR Ring; // メモリ領域をリングバッファに使用
	UCHAR Ringp; // ページ領域をリングバッファに使用
	UCHAR EN; // 指定したチャンネルの設定
	UCHAR PageSize; // ページサイズ
	UCHAR WriteCycle; // ライトサイクル
	UCHAR ReadCode; // Read 用オペコード指定する
	UCHAR WriteCode; // Write 用オペコード指定する
	UCHAR WriteEnabelCode; // Write Enable 用オペコード指定する
	UCHAR WriteDisabelCode; // Write Disable 用オペコード指定する
	UCHAR ReadStatusReg; // Read Status Register 用オペコード
	指定する	

	UCHAR WriteStatusReg; // Write Status Register 用オペコード指定する
	UCHAR PE; // Page Erase 用オペコード指定する
	UCHAR CE; // Chip Erase 用オペコード指定する
	UCHAR DOR; // Dual Output Read 用オペコード指定する
	UCHAR QOR; // Quad Output Read 用オペコード指定する
	UCHAR DFOR; // Dual Fast Output Read 用オペコード指定する
	UCHAR QFOR; // Quad Fast Output Read 用オペコード指定する
	UCHAR DPP; // Dual Page Program 用オペコード指定する
	UCHAR QPP; // Quad Page Program 用オペコード指定する
	UCHAR DFPP; // Dual Fast Page Program 用オペコード指定する
	UCHAR QFPP; // Quad Fast Page Program 用オペコード指定する
	UCHAR EraseValue; // Erase コマンド実行時に書き込む値
	UCHAR EraseTime; // Erase コマンド実行時の処理時間(ms)
	UCHAR DummyCycle; // ダミーサイクル数
	UCHAR BitWidth; // インストラクションのビット数
	UCHAR SubAdrsSize; // サブアドレスのバイト数
	UCHAR SampTiming; // サンプリングタイミング
	UCHAR FB; // データオーダー
	UCHAR SS_A; // スレーブセレクト
	UCHAR WIP; // ライトインプロセスビットのビット制御方法
	UCHAR WEL; // ライトイネーブルラッチビットのビット制御方法
	UCHAR ST; // ステータスレジスタ bit2-6 の制御方法
	UCHAR WPEN; // ライトプロテクトイネーブルのビット制御方法
	ULONG MemoryAddress; // メモリアドレス
	ULONG MemoryAddressMask; // メモリアドレスマスク
	} RS_SPI_SLAVE_INFO, *PRS_SPI_SLAVE_INFO;

戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）

関数	VC ➢	INT usb61mk2_spi_write_slave_config (UCHAR UnitID, UCHAR Channel, PRS_SPI_SLAVE_INFO pSlaveInfo);
	VB ➢	Function Usb61mk2SpiWriteSlaveConfig(ByVal UnitID As Byte, ByVal Channel As Byte, ByVal pSPISlaveInfo) As Long
	VB.NET➢ C#➢	Function Usb61mk2SpiWriteSlaveConfig(ByVal UnitID As Byte, ByVal Channel As Byte, ByVal pSPISlaveInfo As Object) As Integer
機能	SPI スレーブバッファメモリ設定。	
引数	[IN] UnitID ：ユニット番号 [IN] Channel ：スレーブチャンネル(0-1) [IN] pSlaveInfo ：SPI Slave 情報が格納されている	
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_spi_read_slave_status (UCHAR UnitID, UCHAR* pInst0, UCHAR* pReg0, UCHAR* pInst1, UCHAR* pReg1);
	VB ➢	Function Usb61mk2SpiReadSlaveStatus(ByVal UnitID As Byte, pInst0 As Byte, pReg0 As Byte,pInst1 As Byte, pReg1 As Byte) As Long
	VB.NET➢ C#➢	Function Usb61mk2SpiReadSlaveStatus(ByVal UnitID As Byte, ByRef pInst0 As Byte, ByRef pReg0 As Byte, ByRef pInst1 As Byte, ByRef pReg1 As Byte) As Integer
機能	SPI スレーブステータス取得。	
引数	[IN] UnitID [OUT] pInst1 [OUT] pReg0 [OUT] pInst1 [OUT] pReg1	：ユニット番号 ：最後に通信したインストラクション (ch0) ：ステータスレジスタ値(ch0) ：最後に通信したインストラクション (ch1) ：ステータスレジスタ値(ch1)
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。(表 4-2 参照)	

関数	VC ➢	INT usb61mk2_spi_write_slave_status (UCHAR UnitID, UCHAR Enable0, UCHAR Reg0, UCHAR Enable1, UCHAR Reg1);
	VB ➢	Function Usb61mk2SpiWriteSlaveStatus(ByVal UnitID As Byte, ByVal Enable0 As Byte, ByVal Reg0 As Byte, ByVal Enable1 As Byte, ByVal Reg1 As Byte) As Long
	VB.NET➢ C#➢	Function Usb61mk2SpiWriteSlaveStatus(ByVal UnitID As Byte, ByVal Enable0 As Byte, ByVal Reg0 As Byte, ByVal Enable1 As Byte, ByVal Reg1 As Byte) As Integer
機能	SPI スレーブステータス設定。	
引数	[IN] UnitID [IN] Enable0 [IN] Reg0 [IN] Enable 1 [IN] Reg 1	：ユニット番号 ：更新 bit (ch0) ：ステータスレジスタ値(ch0) ：更新 bit (ch1) ：ステータスレジスタ値(ch1)
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。(表 4-2 参照)	

関数	VC ➢	INT usb61mk2_get_slave_select (UCHAR UnitID, UCHAR* pMode, UCHAR* pState);
	VB ➢	Function Usb61mk2GetSlaveSelect (ByVal UnitID As Byte, pMode As Byte, pState As Byte) As Long
	VB.NET➢ C#➢	Function Usb61mk2GetSlaveSelect (ByVal UnitID As Byte, ByRef pMode As Byte, ByRef pState As Byte) As Integer
機能	SS 端子のマニュアル制御設定の取得。	
引数	[IN] UnitID [OUT] pMode [OUT] pState	：ユニット番号 ：0:PLD で自動制御 1:マニュアル制御(State 値を反映) ：SS 端子出力(0:Low 出力　1:High 出力)
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_set_slave_select (UCHAR UnitID, UCHAR bMode, UCHAR bState);
	VB ➢	Function Usb61mk2SetSlaveSelect (ByVal UnitID As Byte, ByVal bMode As Byte, ByVal bState As Byte) As Long
	VB.NET➢ C#➢	Function Usb61mk2SetSlaveSelect (ByVal UnitID As Byte, ByVal bMode As Byte, ByVal bState As Byte) As Integer
機能	SS 端子のマニュアル制御設定。	
引数	[IN] UnitID [IN] bMode [IN] bState	：ユニット番号 ：0:PLD で自動制御 1:マニュアル制御(State 値を反映) ：SS 端子出力(0:Low 出力　1:High 出力)
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

スクリプト関数

関数	VC ➢	INT usb61mk2_set_reg (UCHAR UnitID, PRS_REG_INFO pRegInfo);
	VB ➢	Function Usb61mk2SetReg(ByVal UnitID As Byte, ByVal pRegInfo) As Long
	VB.NET➢ C#➢	Function Usb61mk2SetReg(ByVal UnitID As Byte, ByVal pRegInfo As Object) As Integer
機能	装置内で使用する変数レジスタへ値を設定する。	

引数	[IN] UnitID ：ユニット番号
	[IN] pRegInfo ：Register 情報が格納されている
	typedef struct _RS_REG_INFO {
	UCHAR RegNo; // 使用するレジスタ番号（～）
	UCHAR Tx; // 参照元の選択を指定する
	WORD Param; // 参照元毎に値を設定する
	// (DIO:0x0000 ReadData:バイト数 LITERAL:0x0000-0x00FF)
	} RS_REG_INFO, *PRS_REG_INFO;	

戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_get_reg (UCHAR UnitID, PRS_REG_INFO pRegInfo);
	VB ➢	Function Usb61mk2GetReg(ByVal UnitID As Byte, pRegInfo) As Long
	VB.NET➢ C#➢	Function Usb61mk2GetReg(ByVal UnitID As Byte, ByRef pRegInfo As Object) As Integer
機能	装置内で使用する変数レジスタの値を取得する。	

引数	[IN] UnitID ：ユニット番号
	[OUT] pRegInfo ：Register 情報が格納されている
	typedef struct _RS_REG_INFO {
	UCHAR RegNo; // 使用するレジスタ番号. （～）
	UCHAR Tx; // 参照元の選択を指定する
	WORD Param; // 参照元毎に値を設定する
	// (DIO:0x0000 ReadData:バイト数 LITERAL:0x0000-0x00FF)
	} RS_REG_INFO, *PRS_REG_INFO;	

戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_wait_process (UCHAR UnitID, WORD wTime);
	VB ➢	Function Usb61mk2WaitProcess(ByVal UnitID As Byte, ByVal wTime As Long) As Long
	VB.NET➢ C#➢	Function Usb61mk2WaitProcess(ByVal UnitID As Byte, ByVal wTime As Integer) As Integer
機能	装置内で指定した時間処理待ちを行う。	
引数	[IN] UnitID [IN] wTime	：ユニット番号 ：処理を待つ時間を指定する(ms 単位で指定)
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

関数	VC ➢	INT usb61mk2_wait_interrupt (UCHAR UnitID, WORD_wTime);
	VB ➢	Function Usb61mk2WaitInterrupt(ByVal UnitID As Byte, ByVal wTime As Long) As Long
	VB.NET➢ C#➢	Function Usb61mk2WaitInterrupt(ByVal UnitID As Byte, ByVal wTime As Integer) As Integer
機能	装置内で割り込み監視を指定した DIO を対象として、割り込みを検知するまで待つ。	
引数	[IN] UnitID [IN] wTime	：ユニット番号 ：処理を待つ時間を指定する(ms 単位で指定する)
戻値	関数成功時は RS_SUCCESS を、失敗時はエラーコード返す。（表 4-2 参照）	

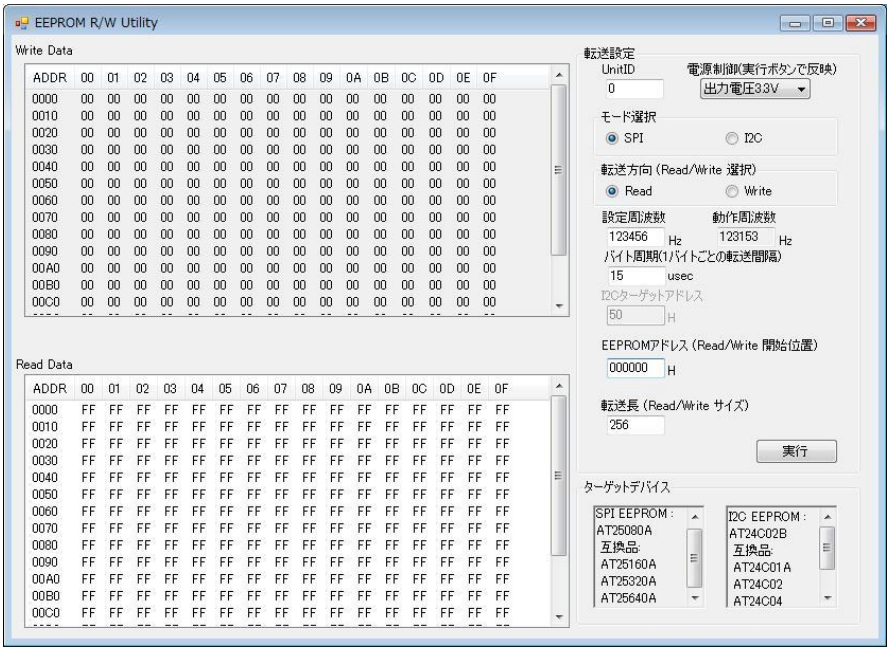
(4-5) サンプルアプリケーションについて

REX-USB61mk2 には、アプリケーション開発を行う際のご参考として以下のサンプルアプリケーションを提供しております。(コーディング例につきましては、各サンプルプログラムのソースコードをご参照ください。)

- SPI/I2C マスター制御 -- EEPROM Read/Write Utility
- SPI/I2C スレーブ動作 -- SPI Slave、I2C Slave
- DIO 制御 -- 基本デジタル入出力、外部イベント監視

【EEPROM Read/Write Utility サンプルアプリケーション説明】

EEPROM R/W Utility サンプルアプリケーション(EEPROMRWUty フォルダー内)は、SPI または I2C インターフェイスを持った EEPROM(ATMEL 社製 AT24C02B, AT25080A)に対して、データの Read または Write をおこなうことができます。
(EEPROM の動作につきましては、各メーカーの仕様書をご参照ください。)



[Unit ID] 本製品に設定された ID を入力します。
[電源制御] 本製品からの電源供給設定を行います。(電源供給なし/1.8V/2.5V/3.3V/5V)
[モード選択] SPI または I2C のどちらかを選択します。
[転送方向] Read または Write を選択します。
[設定周波数] 設定する周波数の値を入力します。(Hz 単位)
[動作周波数] 実際に動作する周波数が表示されます。(設定値に対する近似値)(Hz 単位)
[バイト周期] データ送信時の 1 バイト毎の時間間隔を設定します。
[I2C ターゲットアドレス] I2C ターゲットアドレスを指定します。
[EEPROM アドレス] Read または Write の開始位置を指定します。
[転送長] Read または Write の転送長を指定します。
[Write Data] 転送方向が Write 時に、転送するデータを入力します。
[Read Data] 転送方向が Read 時に、転送されたデータが表示されます。
[実行ボタン] 上記設定での転送が開始されます。

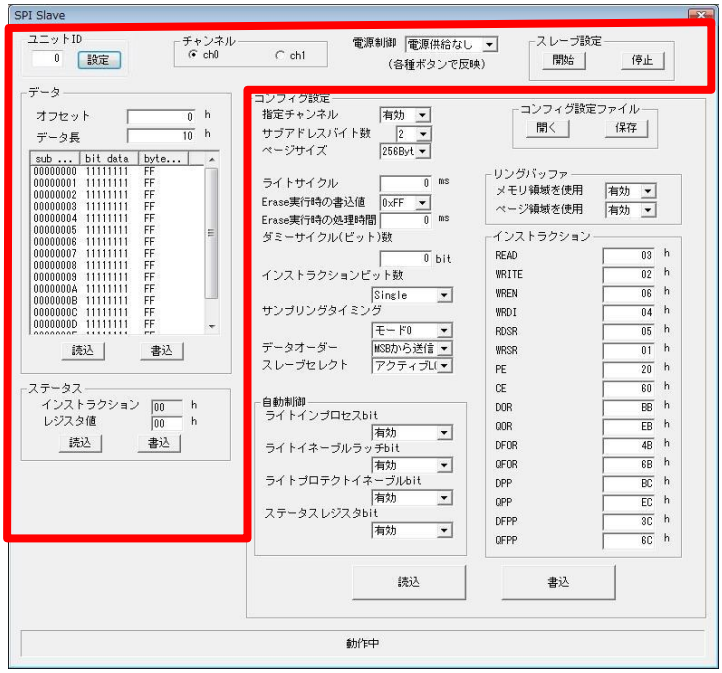
【SPI Slave サンプルアプリケーション説明】

SPI Slave サンプルアプリケーションは、本製品を SPI の Flash ROM や EEPROM のスレーブデバイスとしてエミュレートすることができ、Erase 実行時間やインストラクションの設定を自由に変更することができます。また、実メモリは 1Kbyte ありリングバッファとして使用することが可能です。

(各レジスタ名やコマンド名は SPANSION 製 S25FL032P を想定して記述しています。)

本アプリケーションの設定順序は次の通りとなります。

① ユニット ID の「設定」。② 各種コンフィグ設定と「書込」。③ スレーブ設定の「開始」。



[ユニット ID] 本製品に設定された ID を入力し、[設定]をクリックします。

[チャンネル] スレーブチャンネルを切り替えます。

[電源制御] 本製品からの電源供給設定を行います。(電源供給なし/1.8V/2.5V/3.3V/5V)

・スレーブ設定

[開始]をクリックすると設定した内容でスレーブ動作を開始し、[停止]をクリックすると実行中のスレーブ動作を停止します。

・データ

[オフセット][データ長]を指定し[読込]をクリックすると指定された領域のデータが表示されます。

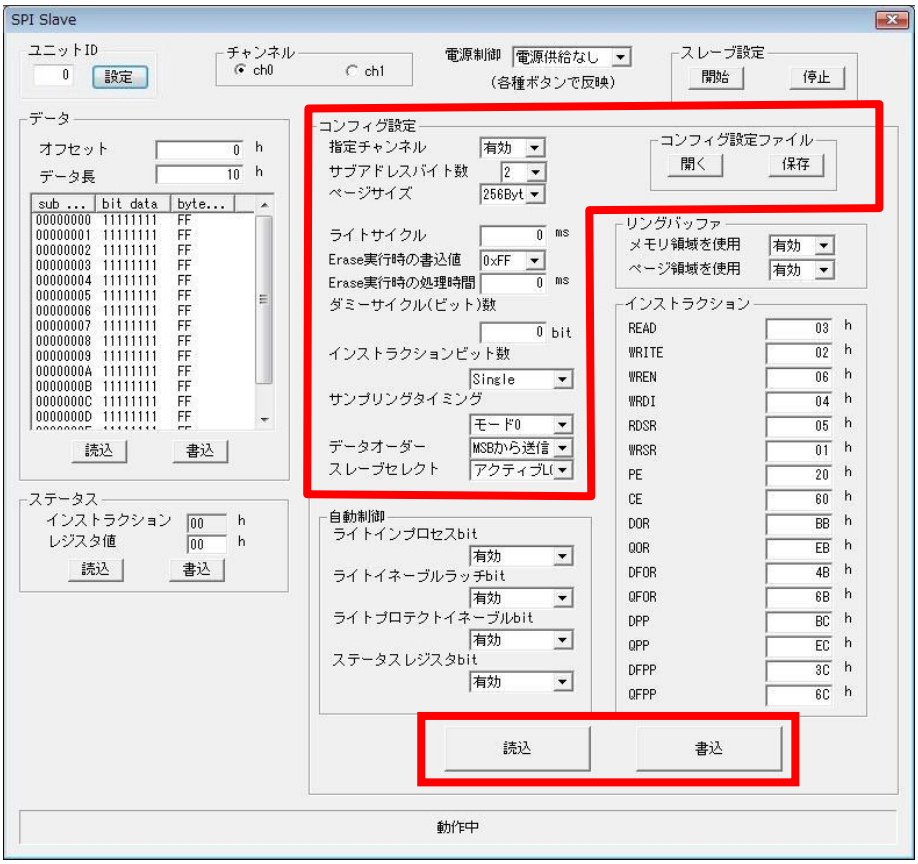
表示データは編集することができ、[書込]をクリックすると表示内容が書き込まれます。

(アドレス 0h～3FFh までの 1Kbyte 分の領域に対して有効です。)

・ステータス

[インストラクション] [読込]をクリックすると、最後にマスターから受け取ったインストラクションが表示されます。(書込はできません。)

[レジスタ値] [読込]をクリックすると現在のステータスレジスタ値が表示されます。また、[ステータスレジスタ bit]の自動制御が「無効」と設定されている場合に、内容を編集して[書込]をクリックしステータスレジスタ値を書き換えることができます。



・コンフィグ設定 ([読込]で本製品に設定されている内容が表示され、[書込]で表示されている内容が本製品へ書き込まれます。)

・コンフィグ設定ファイル [開く]であらかじめ保存されている設定ファイルを読み込み、[保存]で現在の設定内容をファイルに保存します。

(「スレーブサンプル用_Config ファイル」フォルダー内に設定ファイルの一例があります)

[指定チャンネル] 選択されているチャンネルの動作の有効/無効を設定します。

[サブアドレスバイト数] サブアドレスのバイト数を選択します。(1～4byte)

[ページサイズ] ページサイズを選択します。(1～256byte)

[ライトサイクル] 本製品への write が終了するまでの時間を設定します。(ms 単位)

[Erase 実行時の書込値] Erase が実行された場合の書込値を選択します。(0x00/0xFF)

[Erase 実行時の処理時間] Erase 実行が終了するまでの時間を設定します。(ms 単位)

[ダミーサイクル(ビット)数] アドレス送信後の Dummy Byte のビット数を設定します。

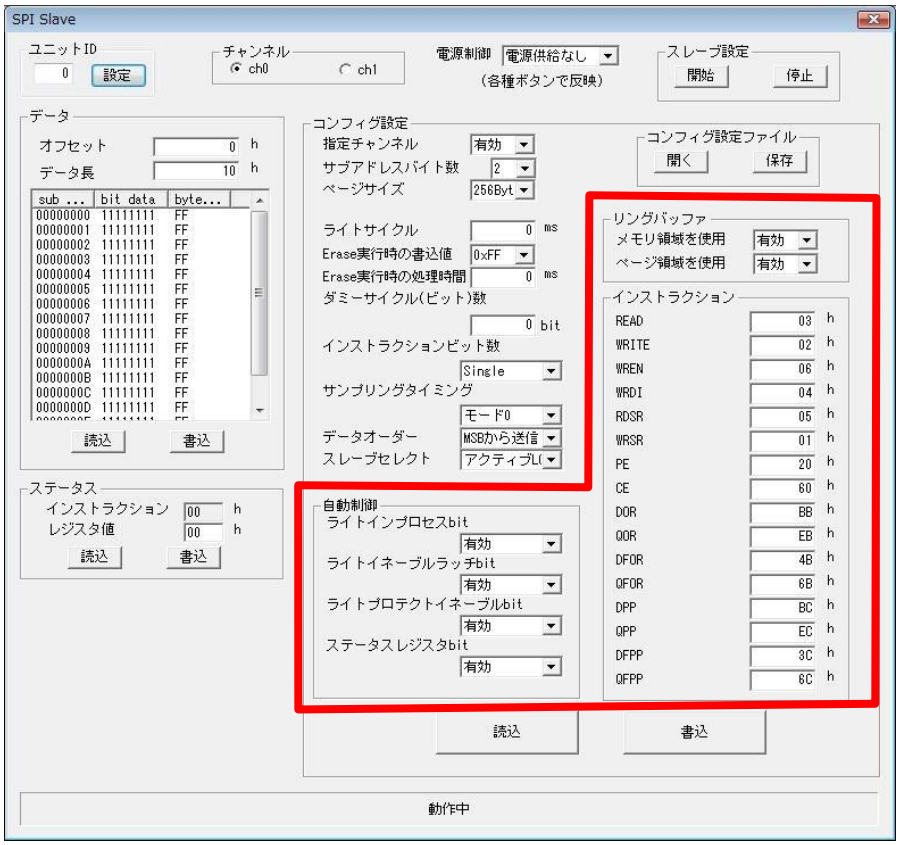
[インストラクションビット数] インストラクション命令の送信幅を選択します。

[サンプリングタイミング] サンプリングするタイミングをモード 0～3 から選択します。

モード	サンプリングエッジ	図
0	立ち上がりエッジ	
1	立ち下がりエッジ	
2	立ち下がりエッジ	
3	立ち上がりエッジ	

[データオーダー] MSB から送信/LSB から送信を選択します。

[スレーブセレクト] スレーブセレクトピンの論理を選択します。(アクティブ Low/High)



・自動制御

[ライトインプロセス bit] ステータスレジスタの WIP ビットの自動制御(有効/無効)を設定します。

[ライトイネーブルラッチ bit] ステータスレジスタの WEL ビットの自動制御(有効/無効)を設定します。

[ライトプロテクトイネーブル bit] ステータスレジスタの SRWD ビットの自動制御(有効/無効)を設定します。

[ステータスレジスタ bit] ステータスレジスタの bit2～bit6 の自動制御(有効/無効)を設定します。

・リングバッファ

[メモリ領域を使用] 有効に設定した場合、指定されたアドレスがメモリ領域を超えた時、メモリ領域の先頭へ戻ります。

[ページ領域を使用] 有効に設定した場合、書き込まれたデータのページサイズ領域を超えた部分がそのページの先頭より書き込まれます。

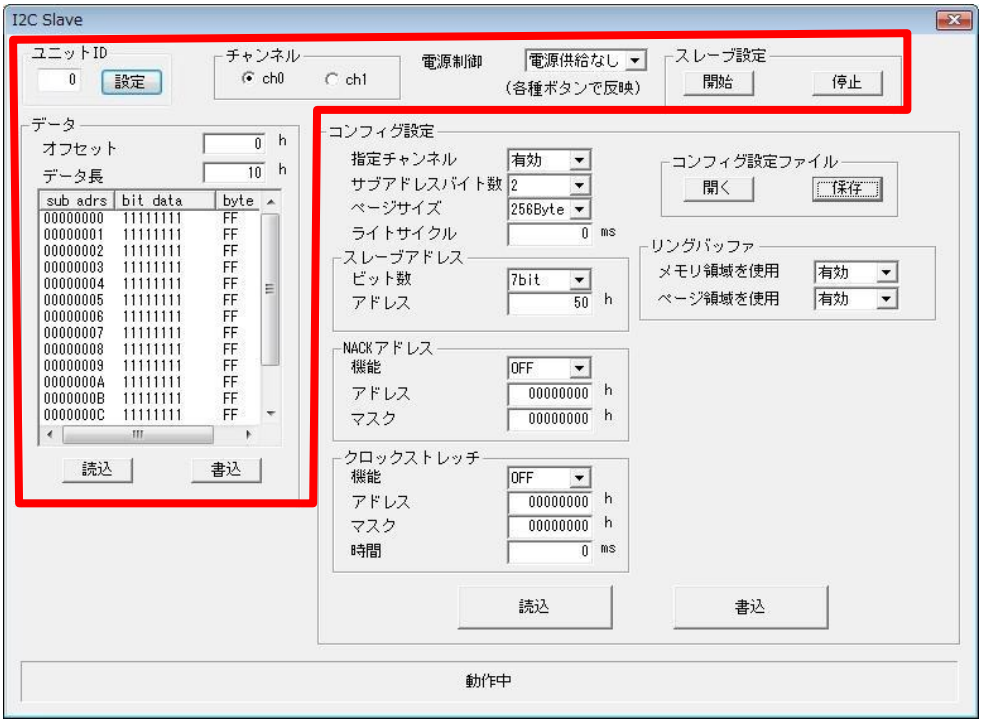
・インストラクション 各インストラクションコマンドのコードを 1 バイトで表示または設定します。

【I2C Slave サンプルアプリケーション説明】

I2C Slave サンプルアプリケーションは、本製品を I2C スレーブとして各種設定を行い動作させることができます。

本アプリケーションの設定順序は次の通りとなります。

① ユニット ID の「設定」。② 各種コンフィグ設定と「書込」。③ スレーブ設定の「開始」。



[ユニット ID]　本製品に設定された ID を入力し、[設定]をクリックします。

[チャンネル]　スレーブチャンネルを切り替えます。

[電源制御]　本製品からの電源供給設定を行います。(電源供給なし/1.8V/2.5V/3.3V/5V)

・スレーブ設定

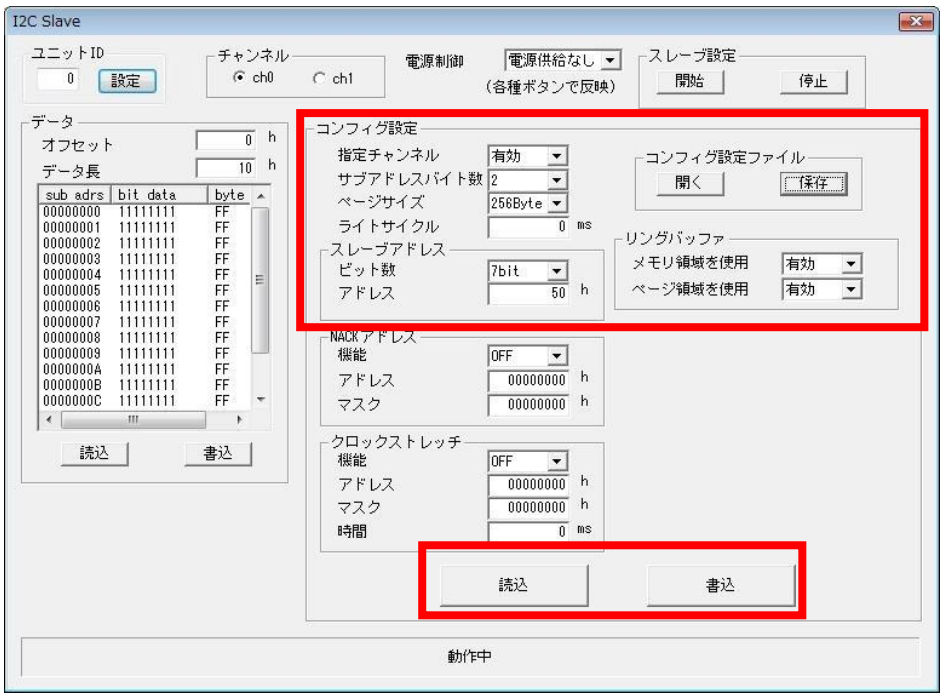
[開始]をクリックすると設定した内容でスレーブ動作を開始し、[停止]をクリックすると実行中のスレーブ動作を停止します。

・データ

[オフセット][データ長]を指定し[読込]をクリックすると指定された領域のデータが表示されます。

表示データは編集することができ、[書込]をクリックすると表示内容が書き込まれます。

(アドレス 0h～3FFh までの 1Kbyte 分の領域に対して有効です。)



・コンフィグ設定 ([読込]で本製品に設定されている内容が表示され、[書込]で表示されている内容が本製品へ書き込まれます。)

・コンフィグ設定ファイル [開く]であらかじめ保存されている設定ファイルを読み込み、[保存]で現在の設定内容をファイルに保存します。

(「スレーブサンプル用_Config ファイル」フォルダー内に設定ファイルの一例があります)

[指定チャンネル] 選択されているチャンネルの動作の有効/無効を設定します。

[サブアドレスバイト数] サブアドレスのバイト数を選択します。(0～4byte)

[ページサイズ] ページサイズを選択します。(1～256byte)

[ライトサイクル] 本製品への write が終了するまでの時間を設定します。(ms 単位)

・スレーブアドレス

[ビット数] 7bit/10bit を選択します。

※ 7bit での設定について

(R/W ビットは含みません)

[例：50h の場合]

0	1	0	1	0	0	0	0	R/W

スレーブアドレス

※ 10bit での設定について(R/W ビットは含みません)

[例：255h の場合]

1	1	1	1	0	1	0	R/W	ACK	0	1	0	1	0	1	0	1

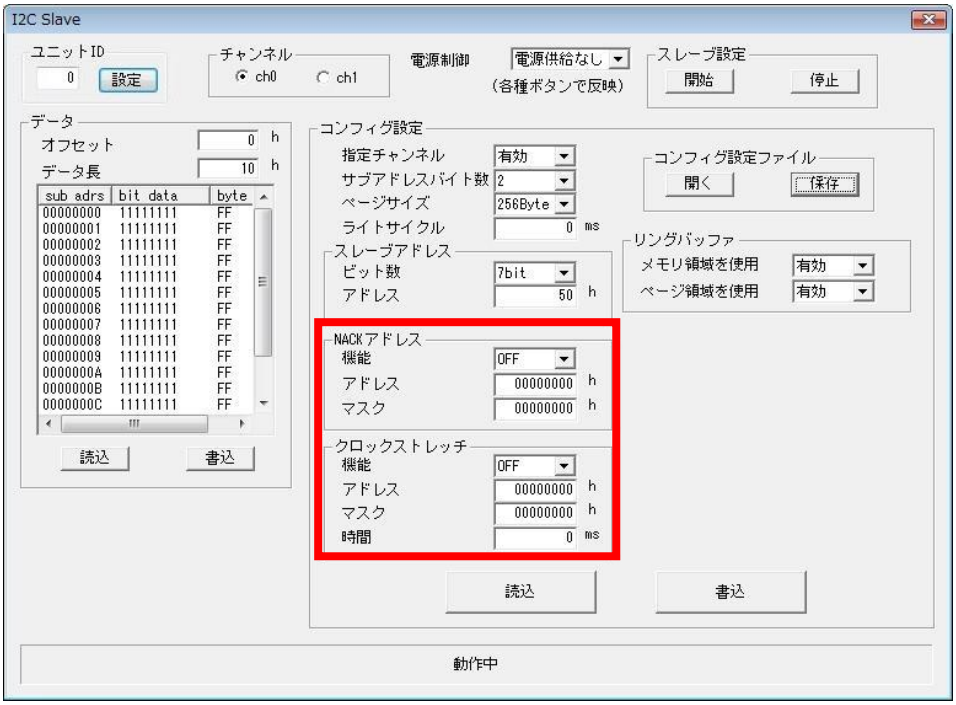
固定パターン　スレーブアドレス

[アドレス] スレーブアドレスを 16 進数で指定します。

・リングバッファ

[メモリ領域を使用] 有効に設定した場合、指定されたアドレスがメモリ領域を超えた時、メモリ領域の先頭へ戻ります。

[ページ領域を使用] 有効に設定した場合、書き込まれたデータのページサイズ領域を超えた部分がそのページの先頭より書き込まれます。



・NACK アドレス

[機能] マスターからアクセスされたアドレスと[マスク]の AND 演算値が[アドレス]と一致した場合に NACK を返すかどうかを設定します。
(下記設定例をご参照ください。)

・クロックストレッチ

[機能] マスターからアクセスされたアドレスと[マスク]の AND 演算値が[アドレス]と一致した場合にクロックストレッチを行うかどうかを設定します。

[時間] クロックストレッチの時間を設定します。
(下記設定例をご参照ください。)

■ [アドレス]と[マスク]の設定例

[アドレス] : 0010 [マスク] : FF1C と設定した場合。

マスターからアクセスされたアドレス / AND 演算 / [マスク] / [アドレス]と一致した場合に設定した動作を実行

0010h & FF1Ch = 0010h

0000&FF1C = 0000 (16 進数表記の“h”は省略しています)
000A&FF1C = 000A
0010&FF1C = 0010 一致
0011&FF1C = 0010 一致
0012&FF1C = 0010 一致
0013&FF1C = 0010 一致
0014&FF1C = 0014
0015&FF1C = 0014
0025&FF1C = 0004
0030&FF1C = 0010 一致

【DIO 基本デジタル入出力サンプルアプリケーション説明】

DIO 基本デジタル入出力サンプルアプリケーションでは、DIO 端子(0～7)への入力値の取得および 1 ビット単位での出力を行うことができます。

[Unit ID] 本製品に設定された ID を入力します。

[電源制御] 本製品からの電源供給設定を行います。(電源供給なし/1.8V/2.5V/3.3V/5V)
入出力ボタンをクリックした時に反映されます。

[入出力方向選択] IN(入力)/OUT(出力)を選択します。

[ビットマップ] 入出力時の Hi(赤)/Low(青)をビットマップ表示します。

[入力] 入力値を 16 進で表示します。

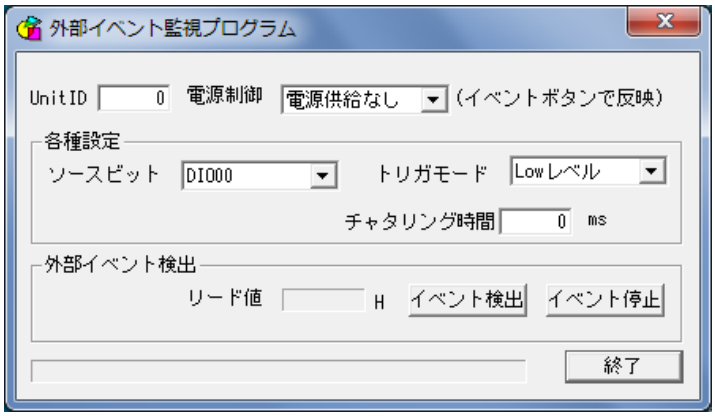
[出力] 出力する値を 16 進で設定します。

[入出力] 設定した内容が実行されます。

[終了] アプリケーションを終了します。

【DIO 外部イベント監視サンプルアプリケーション説明】

DIO 外部イベント監視サンプルアプリケーションでは、指定した DIO 端子の状態(Low/High/立上り/立下り)を監視し、イベントが検出されると DIO 0～7 の値 1 バイト分が読み出されます。



[Unit ID] 本製品に設定された ID を入力します。

[電源制御] 本製品からの電源供給設定を行います。(電源供給なし/1.8V/2.5V/3.3V/5V) イベント検出ボタンをクリックした時に反映されます。

[ソースビット] イベント監視する DIO 0～7 を選択します。

[トリガモード] イベント監視する状態(Low/High/立上り/立下り)を選択します。

[チャタリング時間] イベント検出時のチャタリング防止の時間をミリ秒単位で指定します。(0～500ms)

[リード値] 設定したイベントが検出されると、DIO 0～7 の値 1 バイト分のリード値が 16 進で表示されます。

[イベント検出] 設定した内容でイベント監視を開始します。

[イベント停止] イベント監視中の状態を解除します。

[終了] アプリケーションを終了します。

第5章 仮想COMポートでのコマンド制御

本製品は仮想 COM ポートとしても認識し、ターミナルソフトから専用のコマンドを入力することで以下の制御を行うことができます。

- I2C マスター機能の Read/Write
- SPI マスター機能の Read/Write
- DIO の制御
- SPI/I2C の設定値取得・更新

(5-1) 制御コマンドについて

■ 制御コマンド形式の規則

- 本装置とホスト PC 間の通信は、全てテキスト形式(ASCII コードキャラクタ)とし数値についてもテキストキャラクタで通信を行う。
- 制御コマンド名やパラメーターの大文字小文字の区別はしない。
- 制御コマンドとコマンドパラメーター間は任意の数の空白文字(文字コード：0x20)で区切る。
- 制御コマンドは＜CR＞又は＜LF＞又は＜CR＞＜LF＞で終端する。（文字コード：＜CR＞ 0x0D，＜LF＞ 0x0A）
- コマンド実行後はすべてのコマンド応答が返るまで次のコマンドは受け付けない。
- コマンドの文字列長は終端を含み 128byte 以内とする。

■ プロンプト表示について

本製品に割り当てられた COM ポートをターミナルソフトにてオープンし、[Enter] キーを入力するとプロンプトが出力されます。
オープンする COM ポートの通信パラメーターは以下の設定を行ってください。

ボーレート　　：110/300/600/1200/2400/4800/9600 bps
　　14.4k/19.2k/38.4k/57.6k/115.2k/230.4k/460.8k/921.6k bps
データビット　　：8bit
パリティビット　：なし
ストップビット　：1bit
フロー制御　　：なし

プロンプト出力された状態にて、制御コマンドを入力し本製品を制御することができます。
プロンプトの表示形式はそれぞれ下表のモードを示します。

プロンプト表示	モード
I2C >	I2C モード
SPI >	SPI モード
SPI[SS] >	SPI モード（SS 端子 Enable 中）

■ コマンド応答について

本製品がコマンドを受信するとレスポンスを返します。レスポンス応答の前後には、＜CR＞＜LF＞が付加されます。

■ コマンド履歴について

実行したコマンドは新しいコマンドから順に最大 40 件まで確認することができます。
呼び出した履歴は編集して実行することができます。
(実行方法については HISTORY コマンドの説明をご参照ください。)
ただし、以下の内容は履歴として確認することができません。

- HISTORY コマンド
- コマンドの先頭にスペースを含むもの
- ENTER キーのみのコマンド
- 最新の履歴と一致するコマンド

■ ショートカットキーについて

文字入力以外に下表にあるショートカットキーを使用することができます。

キー	機能
ENTER Ctrl + j Ctrl + m	コマンド実行
BS Ctrl + h	カーソル左 1 文字を削除
DEL Ctrl + d	カーソル位置の 1 文字を削除
HOME Ctrl + a	カーソルを行頭へ移動
END Ctrl + e	カーソルを行末へ移動
十字キー左 Ctrl + b	カーソルを一文字左へ移動
十字キー右 Ctrl + f	カーソルを一文字右へ移動
十字キー上 Ctrl + p	コマンド履歴をひとつ古いものを呼び出す
十字キー下 Ctrl + n	コマンド履歴をひとつ新しいものを呼び出す
Ctrl + c	コマンド入力をキャンセル
Ctrl + k	カーソル位置から行末まで削除
Ctrl + u	カーソル位置左から行頭まで削除
Ctrl + y	最後に行った Ctrl + k または Ctrl + u により削除した文字列をカーソル位置へ挿入する

(5-2) 制御コマンド仕様と使用例について

■ 制御コマンド一覧

制御コマンド名	対応モード	説明
EXMODE	SPI/I2C	SPI/I2C モード変更コマンド
CONFIG	SPI/I2C	設定値の表示コマンド
SET	SPI/I2C	設定値の更新コマンド
WRITE	I2C	I2C マスターWRITE コマンド
READ	I2C	I2C マスターREAD コマンド
TRANS	SPI	SPI マスター通信コマンド
SSSET/SSRESET	SPI	SPI マスターSS 端子制御コマンド
DUMP	SPI/I2C	SPI/I2C のメモリデバイスデータ読み出しコマンド
DIO	SPI/I2C	DIO の制御コマンド
HELP	SPI/I2C	ヘルプコマンド
VER	SPI/I2C	バージョン情報取得コマンド
ECHO	SPI/I2C	エコーコマンド
HISTORY	SPI/I2C	コマンド履歴表示コマンド

■ エラーメッセージ一覧

メッセージ	説明
No such command	指定したコマンドは存在しない
No match current mode	コマンド入力中にモードが変更されてしまった
No match run mode	今のモードに対応していないコマンドが実行された
Not used parameter for mode	今のモードに対応していないパラメーターが指定された
Too much parameters	パラメーターが多すぎる
Invalid option	指定したオプションに間違いがある
Invalid parameter	指定してパラメーターに間違いがある
Not enough option or parameter	オプションもしくはパラメーターが不足している
Not found event of history number	履歴番号に対応するイベントがない
No ack or bus error lock error	I2C 通信エラー
SPI transfer error	SPI 通信エラー
Can not be changed during SSSET	SPI SS セット中はモード変更できない
Timeout error	通信タイムアウト
Not supplied power	電源供給されていない

■ **制御コマンド仕様**

EXMODE

SPI と I2C の 2 種類から使用する機能を選択する。選択していないモード専用のコマンドは動作しない。

コマンド形式		EXMODE [オプション]
コマンド省略形		M [オプション]
適用モード		SPI / I2C
コマンド引数	なし	
オプション	-S	SPI モード選択
	-I	I2C モード選択
	-?	このコマンドのヘルプを表示

使用例

SPI と I2C のモードをトグルで切り替える。

```
I2C > EXMODE

SPI > EXMODE

I2C >
```

-S オプション

SPI モードを選択する。

```
I2C > EXMODE -S

SPI >
```

-I オプション

I2C モードを選択する。

```
SPI > EXMODE -I

I2C >
```

-? オプション

EXMODE の使用方法を表示する。

```
I2C > EXMODE -?

===== Exchange current run mode command help =====

EXMODE [option]

no option : Toggle current mode.
 -S       : Change current mode to SPI.
 -I       : Change current mode to I2C.
 -?       : This help message.

I2C >
```

CONFIG

現在のモードと設定値一覧を表示する。モードにより表示する内容は異なる。

コマンド形式		CONFIG
コマンド省略形		なし
適用モード		SPI / I2C
コマンド引数	なし	
オプション	なし	

使用例

I2C モードの場合

```
I2C > CONFIG

===== CONFIGRATION =====

MODE    : I2C
POWER   : 3.3V
FREQ    : 100kHz
ADJUST1 : 1
ADJUST2 : 0
ADJUST3 : 0
DELY    : 16us
PULLUP  : OFF

I2C >
```

SPI モードの場合

```
SPI > CONFIG

===== CONFIGRATION =====

MODE  : SPI
POWER : 3.3V
FREQ  : 100kHz
DELY  : 16us
BITS  : 8
SAMPLING  : 0
ORDER : LSB

SPI >
```

SET

各モードの設定値の更新を行う。設定できる項目は現在のモード毎に異なる。

コマンド形式		SET オプション SET 設定項目 設定値
コマンド省略形		S オプション S 設定項目 設定値
適用モード		SPI / I2C
コマンド引数	POWER	電源供給の設定（SPI/I2C 共通）
	FREQ	出力周波数の設定（SPI/I2C 共通）
	DELY	データ出力間のインターバルを μSec 単位で設定(SPI/I2C 共通)
	PULLUP	SCL,SDA のプルアップ制御（I2C のみ）
	BITS	転送する BIT 数（SPI のみ）
	SAMPLING	バスサンプリング方法を設定（SPI のみ）
	ORDER	出力するデータオーダーの設定（SPI のみ）
	-?	このコマンドのヘルプを表示
オプション		

設定値説明

POWER

SPI/I2C 共通の設定。電源供給の設定を行う。デフォルト値は OFF。

設定値	説明
OFF	電源供給を行わない
1	1.8V
2	2.5V
3	3.3V
5	5.0V

FREQ

SPI/I2C 共通の設定。1Hz 単位で出力周波数の設定を行う。数字の後に K または M を付けることで kHz 単位 MHz 単位の設定となる。(アルファベットの大文字小文字の区別はしない。)

設定範囲は下記の通りで、デフォルト値は SPI/I2C 共に 100kHz。

モード	範囲
I2C	762Hz～5MHz
SPI	762Hz～50MHz

ADJUST1/ADJUST2/ADJUST3

I2C の設定。通信速度の調整を行う。デフォルトは 0 とする。

設定	対応モード	設定値
ADJUST1	Standard-mode/Fs-mode/Fs-mode Plus	-15～+15
ADJUST2	Hs-mode	-15～+15
ADJUST3	Ultra Fast-mode	-15～+15

DELY

SPI/I2C 共通の設定。送信するデータのバイトとバイトの間に入る待ち時間を設定する。マイクロ秒単位で設定可能で、設定を行わない場合は最短となる。デフォルト値は 0。

モード	範囲
I2C	0，12～65535
SPI	0，12～65535

PULLUP

I2C のみの設定。SDA、SCL 信号線の Pull-up 設定をおこなう。デフォルト値は OFF。

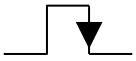
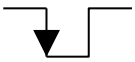
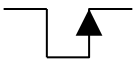
設定値	説明
OFF	プルアップしない
ON	プルアップする

BITS

SPI のみの設定。転送するビット数を指定する。デフォルト値は 8。
(設定可能な範囲は，1～32)

SAMPLING

SPI のみの設定。バスサンプリング方法を指定する。デフォルト値は 0。

パラメーター	サンプリングエッジ	図
0	立ち上がりエッジ	
1	立ち下がりエッジ	
2	立ち下がりエッジ	
3	立ち上がりエッジ	

ORDER

SPI のみの設定。データオーダーを指定する。デフォルト値は MSB。

設定値	説明
MSB	MSB ファースト
LSB	LSB ファースト

使用例

各設定値の更新を行う。 設定項目は 1 回のコマンドで最大 7 個まで指定可能で、設定項目と設定値の間は「=」のみでスペースを含めないように指定する。設定項目間は 1 個以上のスペースで区切る。同じ項目を複数指定した場合は、最後に指定したものが有効となる。
設定値にエラーがある場合は、エラーのある項目のみエラーとなる。

```
I2C > SET item1=value1 item2=value2 item3=value3

ITEM1 : OK
ITEM2 : Invalid parameter
ITEM3 : OK

I2C >
```

-?オプション

SET コマンドの使用方法を表示する。

```
I2C > SET -?

===== Set configuration command help =====

SET option
SET item=value item=value item=value ...
  (Parameters can be specified up to a maximum of 7.)

parameter
 [COMMON]
  POWER   : Set Vcc. OFF -> OFF, 1 -> 1.8V, 2 -> 2.5V, 3 -> 3.3V, 5 -> 5.0V

 [I2C]
  FREQ    : Set frequency. 762 - 5000000Hz. It is possible to put the unit of k or
M
            Max is 3400kHz when HS-mode.
             ex) 100k => 100000Hz, 5M => 5000000Hz
  ADJUST1 : Set adjustment value of speed for Sm, Fm, Fm+. -15 - 15
  ADJUST2 : Set adjustment value of speed for Hs-mode. -15 - 15
  ADJUST3 : Set adjustment value of speed for UF-mode. -15 - 15
  DELY    : Set deley between write byte. 0 or 12～65535us
  PULLUP  : Set pull-up. OFF or ON

 [SPI]
  FREQ    : Set frequency. 762 - 50000000Hz. It is possible to put the unit of k or
M.
             ex) 100k => 100000Hz, 50M => 50000000Hz
  DELY    : Set deley between write byte. 0 or 12～65535us
  BITS    : Set send data bit. 1 - 8
  SAMPLING: Set bus sampling mode. 0 - 3
  ORDER   : Set data order. MSB or LSB

option
  -?       : This help message.

I2C >
```

WRITE

I2C スレーブデバイスに対して書込みを行う。

コマンド形式		WRITE [オプション] [マスターコード] スレーブアドレス サブアドレス [data...]
コマンド省略形		W [オプション] [マスターコード] スレーブアドレス サブアドレス [data...]
適用モード		I2C
コマンド引数	スレーブアドレス	スレーブアドレスは 7bit と 10bit 両対応（16 進）
	サブアドレス	サブアドレスは 4byte サブアドレスに対応（16 進）
	データ	書き込むデータ（16 進）
オプション	-C	スレーブアドレスで指定したデバイスのサブアドレスで指定されたアドレスを開始位置としてデータ（最大 32 バイト）を書き込む
	-N	スレーブアドレスで指定したデバイスにデータ（最大 32 バイト）を書き込む（サブアドレスがないデバイスに使用）
	-H	Hs-mode での通信を行う
	-U	Ultra Fast-Mode での通信を行う
	-?	このコマンドのヘルプを表示

Write コマンドの動作手順

① コマンドを入力する。

7bit アドレスの場合

```
I2C > WRITE 50 22
```

10bit アドレスの場合

```
I2C > WRITE 3a0 22
```

サブアドレスは、1byte から 4byte まで（2 桁から 8 桁の 16 進英数字）で入力できる。

② スレーブアドレスに該当するデバイスの、サブアドレスで指定した値が表示される。

```
I2C > WRITE 50 22

      [WRITE] *address increment mode*
        SLVADD  SUBADDRESS    DATA     DATA'
          50            22     00   ==>
```

③ データを書き換える場合は、カーソル位置に書き換える値を入力し、[Enter]キーを押下する。データを書き換えない場合は、[Enter]キーのみを押下する。
[Enetr]キー押下後 1byte アドレスがインクリメントした場所に移る

```
I2C > WRITE a0 22

      [WRITE] *address increment mode*
        SLVADD  SUBADDRESS    DATA     DATA'
          50            22     00   ==>  01
```

④ １つ前のサブアドレスに戻る場合は、”-”（マイナス）を入力する。

```
I2C > WRITE 50 22

      [WRITE] *address increment mode*
        SLVADD  SUBADDRESS    DATA      DATA'
          50           22       00  ==>  01
          50           23       00  ==>  02
          50           24       00  ==>  -
          50           23       02  ==>  03
```

⑤ 値の書き換えを完了する場合は、[Ctrl + c]キーを押下するとプロンプトに戻る。

```
I2C > WRITE 50 22

      [WRITE] *address increment mode*
        SLVADD  SUBADDRESS    DATA      DATA'
          50           25       00    ==>  04
          50           26       00    ==>  <[Ctrl + c]キーの入力>

I2C >
```

-C オプション

指定したサブアドレスを開始位置として、サブアドレス単位毎に 1 バイトの値を書き込む。1 度で書き込める最大のデータバイト数は 32 バイト。

```
I2C > WRITE -C 50 22 00 01 02 03 04 05

      [WRITE]
        SLVADD  SUBADDRESS    DATA
          50           22       00
          50           23       01
          50           24       02
          50           23       03
          50           25       04
          50           26       05

I2C >
```

-N オプション

サブアドレスが無いデバイスへのデータ書込みに使用するオプション。1 度で書き込める最大のデータバイト数は 32 バイト。

```
I2C > WRITE -N 50 00 01 02 03 04 05

Completed

I2C >
```

-H オプション

Hs-mode での通信を行う場合に使用するオプション。1 度で書き込める最大のデータバイト数は 32 バイト。

```
I2C > WRITE -H 08 50 00 00 01 02 03 04 05

Completed

I2C >
```

-U オプション

Ultra　Fast-Mode での通信を行う場合に使用するオプション。1 度で書き込める最大のデータバイト数は 32 バイト。

```
I2C > WRITE -U 50 00 00 01 02 03 04 05

Completed

I2C >
```

-?オプション

WRITE コマンドの使用方法を表示する。

```
I2C > WRITE -?

===== Write command help =====

WRITE [option] [mastercode] slave [sub] [data ...(max 32 data)]

no option : Write data in dialogue format.
             ex) WRITE slave sub
 -C       : Write specified data to the specified sub address of specified
            slave device.
             ex) WRITE -C slave sub data(max 32)
 -N       : Write specified data to the specified slave device.
             ex) WRITE -N slave data(max 32)
 -H       : Write specified data by Hs-mode.
             ex) WRITE -H mastercode slave data(max 32)
 -U       : Write specified data by Ultra Fast-Mode.
             ex) WRITE -U slave data(max 32)
 -?       : This help message.

I2C >
```

READ

I2C スレーブデバイスからデータの読み出しを行う。

コマンド形式		READ [オプション] [マスターコード] スレーブアドレス サブアドレス [サイズ]
コマンド省略形		R [オプション] [マスターコード] スレーブアドレス サブアドレス [サイズ]
適用モード		I2C
コマンド引数	スレーブアドレス	スレーブアドレスは 7bit と 10bit 両対応（16 進）
	サブアドレス	サブアドレスは 4byte サブアドレスに対応（16 進）
	サイズ	読み出すデータのバイト数（10 進　最大 256）
オプション	-N	スレーブアドレスで指定したデバイスにデータ（最大 256 バイト）を読み込む（サブアドレスがないデバイスに使用）
	-S	オプションなしと同じ動作をするが，シーケンシャルリードを行う
	-H	Hs-mode での通信を行う
	-?	このコマンドのヘルプを表示

使用例

指定したスレーブアドレスを開始位置として指定したサイズ分のデータを読み出す。サイズを指定しない場合は、1 バイト読み出す。

7bit アドレスの場合

```
I2C > READ 50 22 6

      [READ]
        SLVADD  SUBADDRESS    DATA
          50          22       00
          50          23       01
          50          24       02
          50          23       03
          50          25       04
          50          26       05

I2C >
```

10bit アドレスの場合

```
I2C > READ 3a0 22 6

      [READ]
        SLVADD  SUBADDRESS    DATA
          3A0         22       00
          3A0         23       01
          3A0         24       02
          3A0         23       03
          3A0         25       04
          3A0         26       05

I2C >
```

–N オプション

サブアドレスが無いスレーブデバイスから値を読み出す場合に使用する。サイズを指定しない場合は、1 バイト読み出す。

```
I2C > READ -N 50 6

      [READ] slave 50 no sub address format
      +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
  +00:00 01 02 03 04 05                                      ......

I2C >
```

–S オプション

シーケンシャルリードを行ってデータを読み出す。(表示形式は-N オプションと同じ)
サイズを指定しない場合は、1 バイト読み出す。

```
I2C > READ -S 50 22 12

      [READ] slave A50 sequential access
      +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
  20: 00 01 02 03 04 05 41 42 43 44 45 46          ......ABCDEF

I2C >
```

–H オプション

Hs-mode で通信を行い、シーケンシャルリードを行ってデータを読み出す。

```
I2C > READ -H 08 50 22 12

      [READ] slave A50 sequential access
      +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
  20: 00 01 02 03 04 05 41 42 43 44 45 46          ......ABCDEF


I2C >
```

–?オプション

READ コマンドの使用方法を表示する。

```
I2C > READ -?

===== Read command help =====

READ [option] [mastercode] slave [sub] [length]
 length   : read data byte length with decimal. max = 256 (Default 1)

no option : Read data in dialogue format.
             ex) READ slave sub [length]
 -N       : Read data in no sub address format.
             ex) READ -N slave [length]
 -S       : Sequential read operation mode.
             ex) READ -S slave sub [length]
 -H       : Read data by Hs-mode.
             ex) READ -H mastercode slave sub [length]
 -?       : This help message.

I2C >
```

TRANS

SPI スレーブデバイスと通信を行う。SS 端子は自動制御を行い、SSSET 中は自動制御を行わない。

コマンド形式		TRANS [オプション] サイズ [data ...]
コマンド省略形		T [オプション] サイズ [data ...]
適用モード		SPI
コマンド引数	サイズ	データ通信を行うバイト数（10 進　最大 256）
	data	送信するデータ（16 進 最大 32 バイト） 途中にダミークロック出力を行う場合は、 「?x *」（x:クロック数 1～8, *:出力データ）を設定する
オプション	-?	このコマンドのヘルプを表示
	-Q	QUAD での通信を行う
	-D	DUAL での通信を行う

使用例

指定したサイズ分のデータ転送を行う。指定したデータがサイズよりも少ない場合は、足りない分ダミーデータ（FFh）を送信する。

```
SPI > TRANS 10 23 01

      [TRANSFER]
            +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
 +00 [MOSI]: 23 01 FF FF FF FF FF FF FF FF                    #.........
     [MISO]: FF FF 01 02 41 42 43 44 45 46                    ....ABCDEF

SPI >
```

–Q/-D オプション

通信のバス幅を Dual(2bit)バス幅、Quad(4bit)バス幅での通信を行う。最初のインストラクションやサブアドレス等は Single(1bit)バス幅で通信するため、Single と Dual/Quad の切り替わりは識別子「:（コロン)」を指定する。

```
SPI > TRANS -D 10 03 00 00 :

      [TRANSFER]
            +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
 +00 [MOSI]: 03 00 00 FF FF FF FF FF FF FF                    ..........
     [MISO]: FF FF FF 01 41 42 43 44 45 46                    ....ABCDEF

SPI >
```

```
SPI > TRANS -Q 10 02 00 00 : 01 02 03 04 05 06 07

      [TRANSFER]
            +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
 +00 [MOSI]: 02 00 00 01 02 03 04 05 06 07                    ..........
     [MISO]: FF FF FF FF FF FF FF FF FF FF                    ..........

SPI >
```

ダミークロック出力例

```
SPI > TRANS -Q 11 02 ?4 0a : 00 00 01 02 03 04 05 06 07

      [TRANSFER]
             +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
 +00 [MOSI]: 02 0a 00 00 01 02 03 04 05 06 07                  ..........
     [MISO]: FF FF FF FF FF FF FF FF FF FF FF                  ..........

SPI >
```

–?オプション

TRANS コマンドの使用方法を表示する。

```
SPI > TRANS -?

===== TRANS command help =====

TRANS [option] length [data ... (max 32 data)]
 length   : Transfer data byte length with decimal.

no option : Transfer data.
 -?       : This help message.
 -Q       : Quad transfer.
 -D       : Dual transfer.

 dummy clock:
  specified ‘?x *’ in the data.
   x : number of clock (1 - 8)
   * : output data

-Q or -D command example:
 separated by a colon between the dual(quad) and single sending the transmission data
 TRANS -D 256 3B 00 00 00 FF :
 TRANS -Q 16 38 00 00 00 : 01 02 03 04 05 06 07 08 09 0a 0b 0c

SPI >
```

SSSET/SSRESET

SPI スレーブデバイスと通信を行うときの、スレーブセレクトの制御を行う。
通常は TRANS コマンドを実行するときは、自動でスレーブセレクトの制御が行われるが、複数の TRANS コマンドを実行するときにスレーブセレクトをセットしたままとしたい場合に使用する。
SSSET 中はプロンプトに SET 中(SPI[SS])であることを表示し、SET 中に I2C モードへ変更するとエラーを出力する。

コマンド形式		SSSET/SSRESET [オプション]
コマンド省略形		なし
適用モード		SPI
コマンド引数	なし	
オプション	-?	このコマンドのヘルプを表示

使用例

SSSET を呼び出した後は SSRESET を呼び出す必要がある。

```
SPI > SSSET

SPI[SS] > TRANS 10 23 01

      [TRANSFER]
            +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
 +00 [MOSI]: 23 01 FF FF FF FF FF FF FF FF                    #.........
     [MISO]: FF FF 01 02 41 42 43 44 45 46                    ....ABCDEF

SPI[SS] > TRANS 10

      [TRANSFER]
            +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
 +00 [MOSI]: FF FF FF FF FF FF FF FF FF FF                    #.........
     [MISO]: 47 48 49 4A 4B 4C 4D 4E 4F 50                    GHIJKLMNOP

SPI[SS] > SSRESET

SPI >
```

-?オプション

SSSET/SSRESET コマンドの使用方法を表示する。

```
SPI > SSSET -?

===== SSSET/SSRESET command help =====
SSSET / SSRESET command make the control of the SS pin when the SPI communication.
When not performing SSSET, the automatic control is performed by TRANSFER command.

SSSET [option]
SSRESET [option]

no option : Slave Select Control.
 -?       : This help message.

SPI >
```

DUMP

SPI/I2C のメモリデバイスから 256 バイトずつ連続読み出しを行う。

コマンド形式		DUMP [オプション] [スレーブアドレス/インストラクション] [サブアドレス]
コマンド省略形		D [オプション] [スレーブアドレス/インストラクション] [サブアドレス]
適用モード		SPI / I2C
コマンド引数	スレーブアドレス	I2C モードの場合：7bit と 10bit 両対応（16 進）
	インストラクション	SPI モードの場合：1byte の READ コマンドを指定する（16 進）
	サブアドレス	サブアドレスは 4byte サブアドレスに対応（16 進） 指定しない場合は最後に読み出したアドレスの続きからとする
オプション	-?	このコマンドのヘルプを表示

使用例

I2C の場合

スレーブアドレスとサブアドレスを指定して 256byte 読み出す。サブアドレスは 4byte まで指定が可能です、省略した場合は読み出したデータの続きから読み出す。
スレーブアドレスとサブアドレスを省略した場合は、最後に指定したスレーブアドレスを使用して読み出したデータの続きから読み出す。

```
I2C > DUMP 50 0000

      [DUMP] slave 50
      +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
0000: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F        ......
0010: 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F        ......
                                  :
00F0: F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF        ......

I2C > DUMP 50

      [DUMP] slave 50
      +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
0100: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F        ......
0110: 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F        ......
                                  :
01F0: F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF        ......

I2C > DUMP

      [DUMP] slave 50
      +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
0200: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F        ......
0210: 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F        ......
                                  :
02F0: F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF        ......

I2C >
```

SPI の場合

READ コマンドのインストラクションとサブアドレスを指定して 256byte 読み出す。サブアドレスは 4byte まで指定が可能で、省略した場合は読み出したデータの続きから読み出す。

```
SPI > DUMP 03 0000

      [DUMP] instruction 03
      +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
0000: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F        ......
0010: 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F        ......
                                 :
00F0: F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF        ......

SPI > DUMP 03

      [DUMP]  instruction 03
      +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
0100: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F        ......
0110: 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F        ......
                                 :
01F0: F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF        ......

SPI > DUMP

      [DUMP]  instruction 03
      +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
0200: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F        ......
0210: 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F        ......
                                 :
02F0: F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF        ......

SPI >
```

-?オプション

DUMP コマンドの使用方法を表示する。

```
SPI > DUMP -?

===== DUMP command help =====
DUMP command performs continuous reading of data from the memory device.
Read from the last continuation if no subaddress.

DUMP [option] slave or instruction [sub]

no option : Read data from the memory device.
-?        : This help message.

SPI >
```

DIO

DIO の出力入力状態の確認と、設定変更を行います。

コマンド形式		DIO [オプション] [VALUE]
コマンド省略形		I [オプション] [VALUE]
適用モード		SPI / I2C
コマンド引数	VALUE	出力ビットの値を一括設定する（16 進）
オプション	-D	入出力方向を全ビット一括設定する（16 進）
	-0 ～ -7	ビットごとに設定を行う（0 or 1）
	-?	このコマンドのヘルプを表示

使用例

パラメーターがない場合

設定変更は行わず設定値の取得のみを行う。

```
I2C > DIO

      [DIO]
         b7 b6 b5 b4 b3 b2 b1 b0
    IN    1  1  0  0  -  -  -  -
    OUT   -  -  -  -  0  1  0  1

I2C >
```

パラメーターを指定した場合

全ビットに対して出力値の一括設定を行う。下位ビットから順に b0⇒ b7 に対応する。

```
I2C > DIO F0

      [DIO]
         b7 b6 b5 b4 b3 b2 b1 b0
    IN    -  -  -  -  -  -  -  -
    OUT   1  1  1  1  0  0  0  0

I2C >
```

-D オプション

DIO の入出力方向設定を一括で行う。1 を設定すると出力方向、0 を設定すると入力方向になる。

```
I2C > DIO

      [DIO]
         b7 b6 b5 b4 b3 b2 b1 b0
    IN    -  -  -  -  -  -  -  -
    OUT   1  1  1  1  0  0  0  0

I2C > DIO -D AA

      [DIO]
         b7 b6 b5 b4 b3 b2 b1 b0
    IN    -  0  -  0  -  0  -  0
    OUT   1  -  1  -  0  -  0  -

I2C >
```

-0 ～ -7 オプション(b0～b7)

指定したビットのみの設定の変更を行う。

```
I2C > DIO

      [DIO]
         b7 b6 b5 b4 b3 b2 b1 b0
    IN    -  1  -  1  -  0  -  0
    OUT   1  -  1  -  0  -  0  -

I2C > DIO -1 1

      [DIO]
         b7 b6 b5 b4 b3 b2 b1 b0
    IN    -  1  -  1  -  0  -  0
    OUT   1  -  1  -  0  -  1  -

I2C > DIO -7 0

      [DIO]
         b7 b6 b5 b4 b3 b2 b1 b0
    IN    -  1  -  1  -  0  -  0
    OUT   0  -  1  -  0  -  1  -

I2C >
```

-?オプション

DIO コマンドの使用方法を表示する。

```
I2C > DIO -?

===== DIO command help =====

DIO [option] [(value or direction)]
 direction : Set dio bit direction. output -> 1, input -> 0
 value     : Set dio bit value. 1 or 0

no option : Set dio all setting.
 -D       : Set dio all direction.
 -0 - -7  : Set dio every bit setting.
 -?       : This help message.

I2C >
```

HELP

使用可能なコマンドの一覧と概要を表示する。各コマンドの詳細なヘルプは、各コマンドに用意してある「-?」オプションで表示する。

コマンド形式		HELP
コマンド省略形		なし
適用モード		SPI / I2C
コマンド引数	なし	
オプション	なし	

使用例

使用可能なコマンドの一覧を表示する。

```
I2C > HELP

===== Command help =====

EXMODE    : Exchange current mode command (SPI/I2C)
CONFIG    : Get configuration command (SPI/I2C)
SET       : Set configuration command (SPI/I2C)
WRITE     : I2C master write command (I2C)
READ      : I2C master read command (I2C)
TRANS     : SPI master transfer command (SPI)
SSSET     : SPI master Slave Select Manual Control command (SPI)
SSRESET   : SPI master Slave Select Manual Control command (SPI)
DUMP      : Continuous reading of data from the memory device (SPI/I2C)
DIO       : DIO control command (SPI/I2C)
VER       : Version command (SPI/I2C)
ECHO      : Echo command (SPI/I2C)
HISOTRY   : Show command history (SPI/I2C)

? or HELP : This help message. (SPI/I2C)

More help “command -?”.

I2C >
```

VER

バージョン番号の表示を行う。

コマンド形式		VER
コマンド省略形		なし
適用モード		SPI / I2C
コマンド引数	なし	
オプション	なし	

使用例

現在のバージョンを表示する。

```
I2C > VER

REX-USB61Mk2 version 1.00.00
(c)Copyright RATOC Systems, Inc. 2014

I2C >
```

ECHO

ホスト PC から送信された文字（コマンド）のエコーバックの制御を行う。

コマンド形式		ECHO [オプション]
コマンド省略形		なし
適用モード		SPI / I2C
コマンド引数	なし	
オプション	ON	エコーバックを ON
	OFF	エコーバックを OFF

使用例

現在の状態を出力する。状態の変更は行わない。

```
I2C > ECHO

ECHO ON

I2C >
```

ON オプション

エコーバックを ON（有効）にする。

```
I2C > ECHO ON     ← （この入力はエコーバックされない）
I2C >             ← （これ以降の入力はエコーバックされる）
```

OFF オプション

エコーバックを OFF（無効）にする。

```
I2C > ECHO OFF    ← （この入力はエコーバックされる）
I2C >             ← （これ以降の入力はエコーバックされない）
```

HISTORY

コマンドを実行した履歴の一覧表示を行う。また、履歴番号指定で履歴コマンドの実行を行う。

コマンド形式		HISTORY [オプション] [履歴番号]
コマンド省略形		なし
適用モード		SPI / I2C
コマンド引数	履歴番号	一覧表示したときの履歴番号を指定する（10 進）
オプション	-?	このコマンドのヘルプを表示

使用例

コマンド履歴の一覧表示を行う。コマンド履歴が編集中の場合は、履歴番号の後に「*」(アスタリスク)が表示される。

```
I2C > HISTORY

   1  VER
   2  WRITE
   3  EXMODE -I
   4* CONFIG
   5  WRITE a0 22
   6  EXMODE -S

I2C > HISTORY 1
ver

REX-USB61Mk2 version 1.00.00
(c)Copyright RATOC Systems, Inc. 2015

I2C >
```

–?オプション

HISTORY コマンドの使用方法を表示する。

```
I2C > HISTORY -?

===== HISTORY command help =====

HISTORY [option] [history no]
 history no : Set history no.

no option : Show command history list.
 -?        : This help message.

I2C >
```

製品に関するお問い合わせ

REX-USB61mk2 の技術的なご質問やご相談の窓口を用意していますのでご利用ください。

ラトックシステム株式会社
I&L サポートセンター
〒550-0015
大阪市西区南堀江 1-18-4 Osaka Metro 南堀江ビル 8F
TEL 06-7670-5064
FAX 06-7670-5066
<サポート受付時間>
月曜ー金曜（祝祭日は除く）AM 10:00 - PM 1:00
PM 2:00 - PM 5:00

また、インターネットのホームページでも受け付けています。
HomePage ➲ https://www.ratocsystems.com

ご注意

☑本書の内容については、将来予告なしに変更することがあります。
☑本書の内容につきましては万全を期して作成しましたが、万一ご不審な点や誤りなどお気づきになられましたらご連絡願います。
☑本製品および本製品添付のマニュアルに記載されている会社名および製品名は、各社の商品または登録商標です。
☑運用の結果につきましては、責任を負いかねますので、予めご了承願います。

FAX：06-7670-5066

REX-USB61mk2 FAX 質問用紙
（このページをコピーしご使用ください）

●下記ユーザ情報をご記入願います。

法人登録の方のみ	会社名・学校名			
	所属部署			
ご担当者名				
E-Mail				
住所	〒			
TEL		FAX		
シリアルNo.				
ご購入情報	販売店名		ご購入日	

●下記運用環境情報とお問い合わせ内容をご記入願います。

【パソコン/マザーボードのメーカ名と機種名】
【ご利用の OS】
【接続機器】
【お問合せ内容】
【添付資料】

⚠ 個人情報の取り扱いについて

ご連絡いただいた氏名、住所、電話番号、メールアドレス、その他の個人情報は、お客様への回答など本件に関わる業務のみに利用し、他の目的では利用致しません。

http://www.ratocsystems.com/