REX-USB61

USB-SPI/I2C Converter

ユーザーズマニュアル

2021 年 11 月

第 8.0 版

RATOC Systems, Inc.

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

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-USB61 を利用することによって SPI/I2C バスを持つ様々なデバイスを PC 上から簡単に制御することができます。

[制御ユーティリティを提供]

SPI/I2C 制御ユーティリティでは、SPI/I2C および GPO(ポート出力)の制御を行うことができ、設定ファイルやログファイルの保存が可能となります。
(詳細につきましては第 3 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以下の制御が可能となります。

- 本装置から外部デバイスに対し、3.3V 又は 5.0V の電源を供給可能。(ただし供給電流は 100mA 以下)
- 外部から電源端子に電圧を入力することで SPI/I2C/スレーブ選択ポート/パラレルアウトポートの入出力レベルを 1.8V～5.0V に対応可能。
- SPI/I2C およびマスター/スレーブの切り替え。(SPI はマスターのみ)
- SPI/I2C バスの周波数を指定。
- I2C モード時 4bit 分のデジタル出力が可能。

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。
(関数の詳細につきましては第 4 章の(4-4)を、 サンプルアプリケーションにつきましては(4-6)をご参照ください。)

[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 Full Speed Device
接続コネクタ	USB mini B コネクタ
電源電圧	5V (USB バスパワーから取得)
消費電流	100mA
サポート インターフェイス	SPI　マスター　　最大周波数 12MHz I2C　マスター/スレーブ　対応周波数 47KHz～1MHz
外部入出力レベル	3.3V/5V　但し外部電源により 1.8V～5.0V 可能。
外形寸法	57(W) x　75(D) x 18(H) mm
重量	約 60g ケーブル部含まず。
動作環境	温度:5～55℃ 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
インストール用 設定ファイル インストーラー	REX-USB61 用設定ファイル(USB61.inf) (Windows Vista x32/XP x32/XP x64/2000 用) Windows 11/10/8.1/7/Vista x64 用インストーラー
ユーティリティ スクリプトファイル	SPI/I2C 制御ユーティリティ(Usb61Uty.exe) スクリプトファイル(I2C_script.txt/SPI_script.txt)
サンプルプログラム	SPI/I2C 送受信サンプルプログラム I2C スレーブ用サンプルプログラム (VC6.0/VB6.0/VB2005/C#)
ライブラリ	SPI/I2C 機器制御用ライブラリ (usb61api.dll) VC 用ヘッダファイル (usb61def.h) VB 用標準モジュール (usb61api.bas) VB 用コードファイル (usb61api.vb)
ActiveX コントロール	ActiveX コントロール (usb61api.ocx)
アンインストールユーティリティ	INF 削除ユーティリティ(Windows XP x32/XP x64/2000) (USB61_uninst.exe)
対応 OS	Windows 11/10/8.1/7/Vista/XP/2000 ※ 64bit 版 OS にも対応

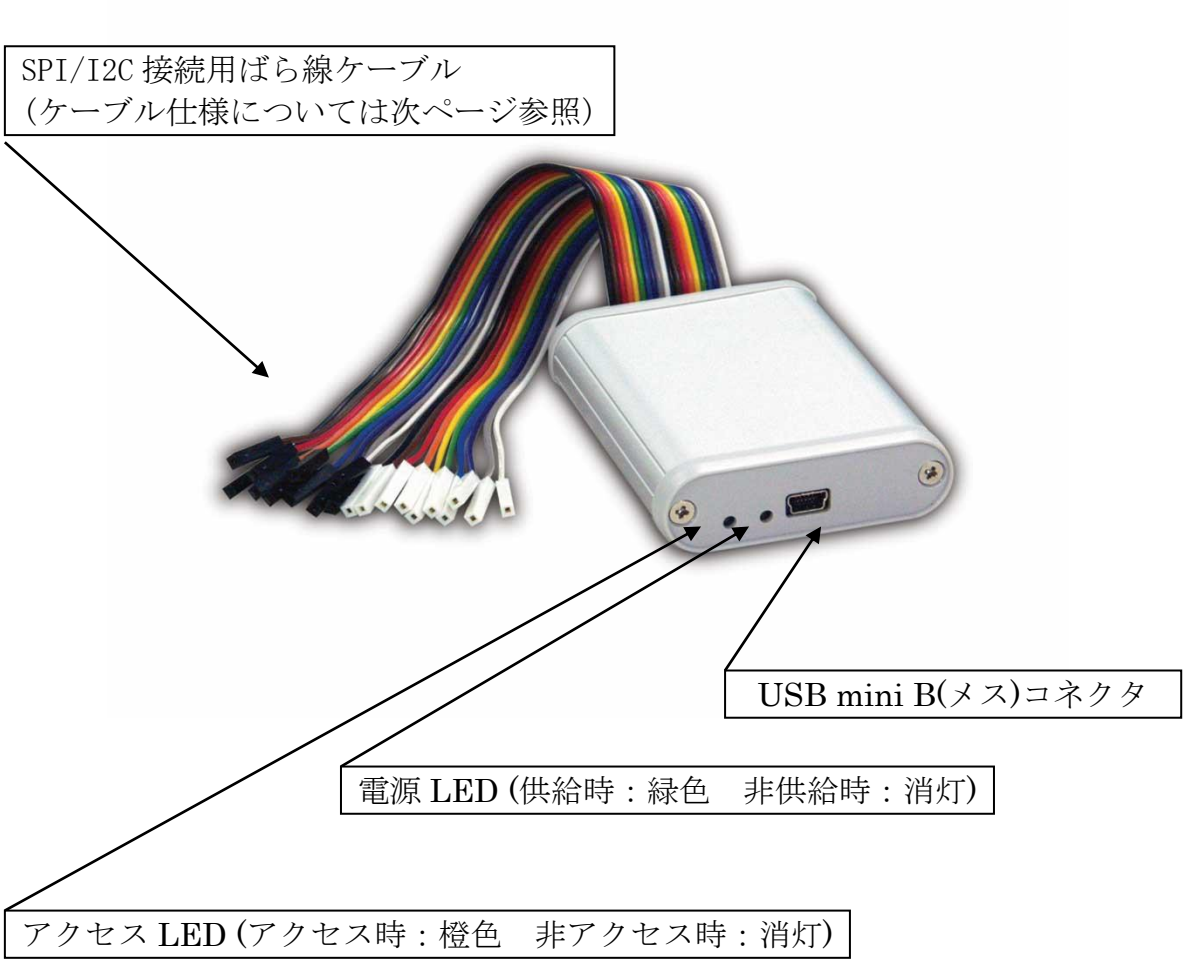
※ 本製品は複数台での使用には対応しておりません。

(REX-USB61mk2 は複数台接続に対応しております。)

(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-USB61 本体
- ☑ USB A - mini B ケーブル
- ☑ SPI/I2C 接続用ばら線ケーブル
- ☑ 保証書



(1-3) ケーブル仕様

REX-USB61 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	Power	ターゲットデバイス電源入出力 (出力時 5V or 3.3V @100mA) (入力時 1.8V ～ 5V)
２	黒	赤	Power	ターゲットデバイス電源入出力 (出力時 5V or 3.3V @100mA) (入力時 1.8V ～ 5V)
３	黒	橙	1MHz－SCL	I2C 用クロック信号 (401KHz～1MHz バス電圧 5V 専用) (プルアップ抵抗 10kΩ)
４	黒	黄	1MHz－SDA	I2C 用データ信号 (401KHz～1MHz バス電圧 5V 専用) (プルアップ抵抗 10kΩ)
５	黒	緑	SCL	I2C 用クロック信号 (47KHz～400KHz 1.8～5V) (プルアップ抵抗 10kΩ)
６	黒	青	SDA	I2C 用データ信号 (47KHz～400KHz 1.8～5V) (プルアップ抵抗 10kΩ)
７	黒	紫	SCK	SPI 用クロック信号 (12MHz 対応 1.8～5V)
８	黒	灰	SDO	SPI 用データアウト信号 (12MHz 対応 1.8～5V)
９	黒	白	SDI	SPI 用データイン信号 (12MHz 対応 1.8～5V)
１０	黒	黒	Reserve	使用しません。 (接続しないでください。)

※ I2C 401KHz～1MHz 対応線(3,4)と SPI 信号線(7～9)は同時に接続しないでください。

ピン番号	ハウジング色	ケーブル色	信号名	用途
１１	白(灰)	茶	GND	グランド
１２	白(灰)	赤	GND	グランド
１３	白(灰)	橙	DO0	SPI 用 SS0 / I2C 用 PORT0 (1.8～5V)
１４	白(灰)	黄	DO1	SPI 用 SS1 / I2C 用 PORT1 (1.8～5V)
１５	白(灰)	緑	DO2	SPI 用 SS2 / I2C 用 PORT2 (1.8～5V)
１６	白(灰)	青	DO3	SPI 用 SS3 / I2C 用 PORT3 (1.8～5V)
１７	白(灰)	紫	GND	グランド
１８	白(灰)	灰	GND	グランド
１９	白(灰)	白	N.C	ノーコネクト
２０	白(灰)	黒	N.C	ノーコネクト

(1-4) 各モードについて

SPI /I2C バスそれぞれの、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	スレーブの選択、任意データの送信、 スレーブから受信したデータ表示を行う
I2C バス	マスターモード	特定のアドレスに対して任意のデータを送信、 スレーブからの受信データ表示を行う。
	スレーブモード	自己アドレスに受信したデータ表示､受信データに対するマスターへの任意データの送信を行う。

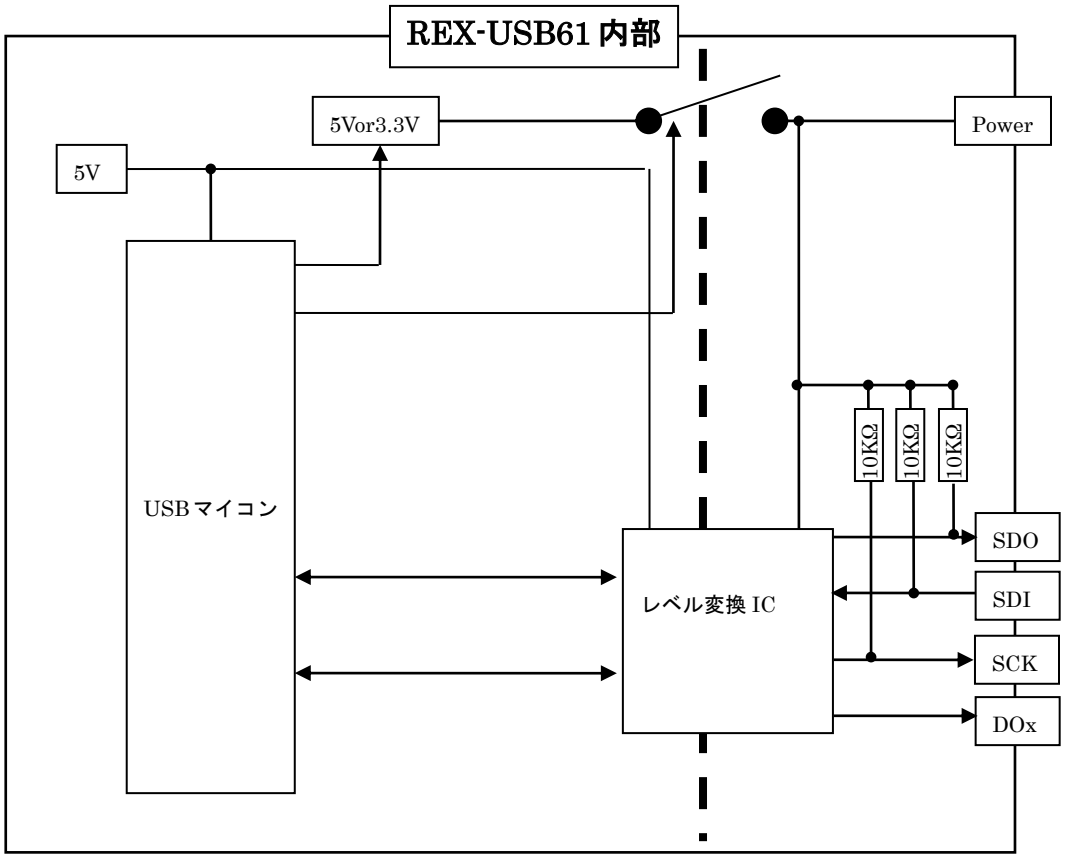
REX-USB61 のマスターモードまたはスレーブモードは､ユーティリティソフトウェアまたはライブラリ関数により指定いたします。

(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-USB61 の電源部分について**

  REX-USB61 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-USB61 の Power ピンをターゲットデバイスの電源と接続してください。**

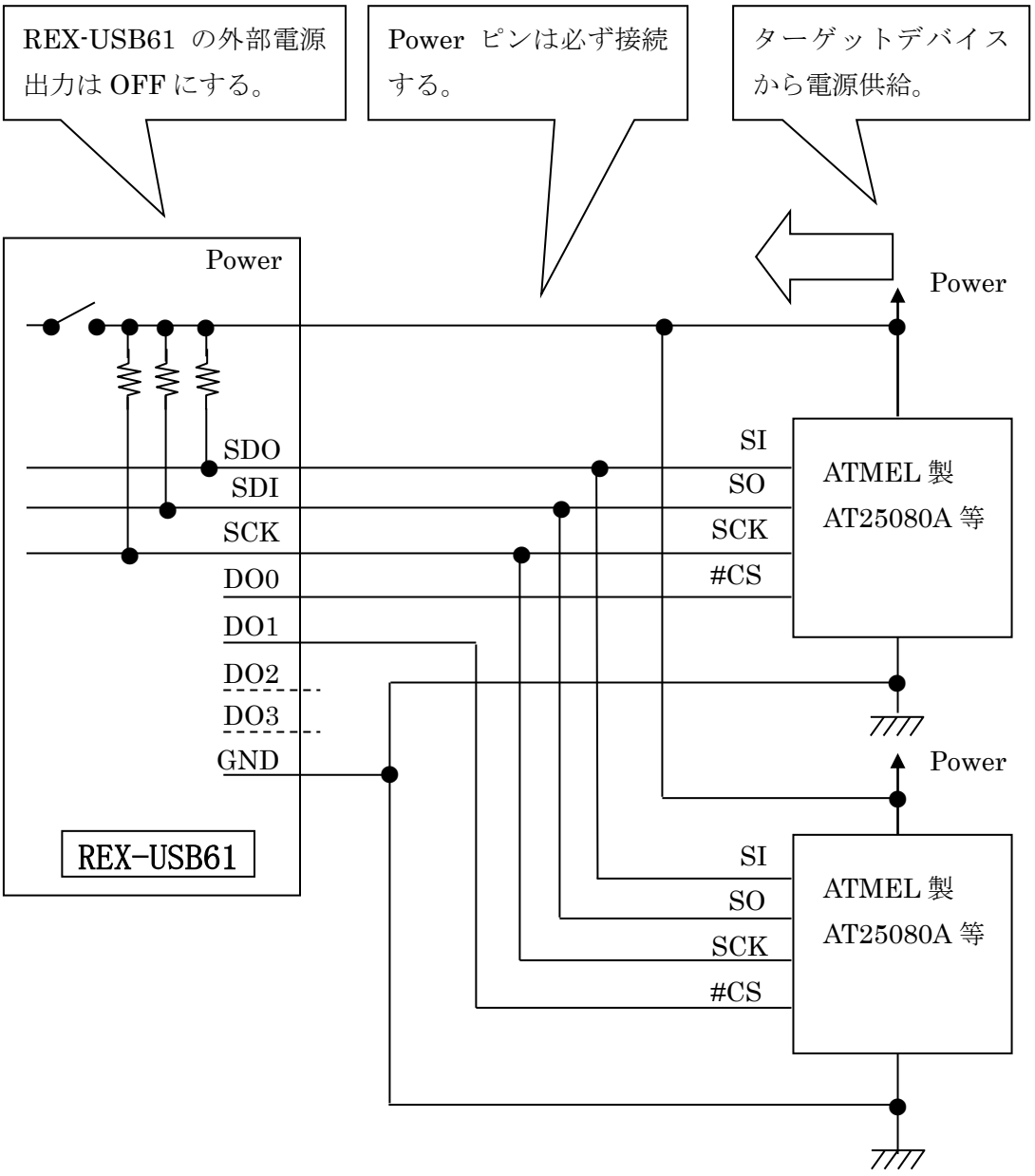


【注意】

接続デバイスの取り付け・取り外し時には、本製品と接続デバイスへの電源供給を行わないでください。
(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）

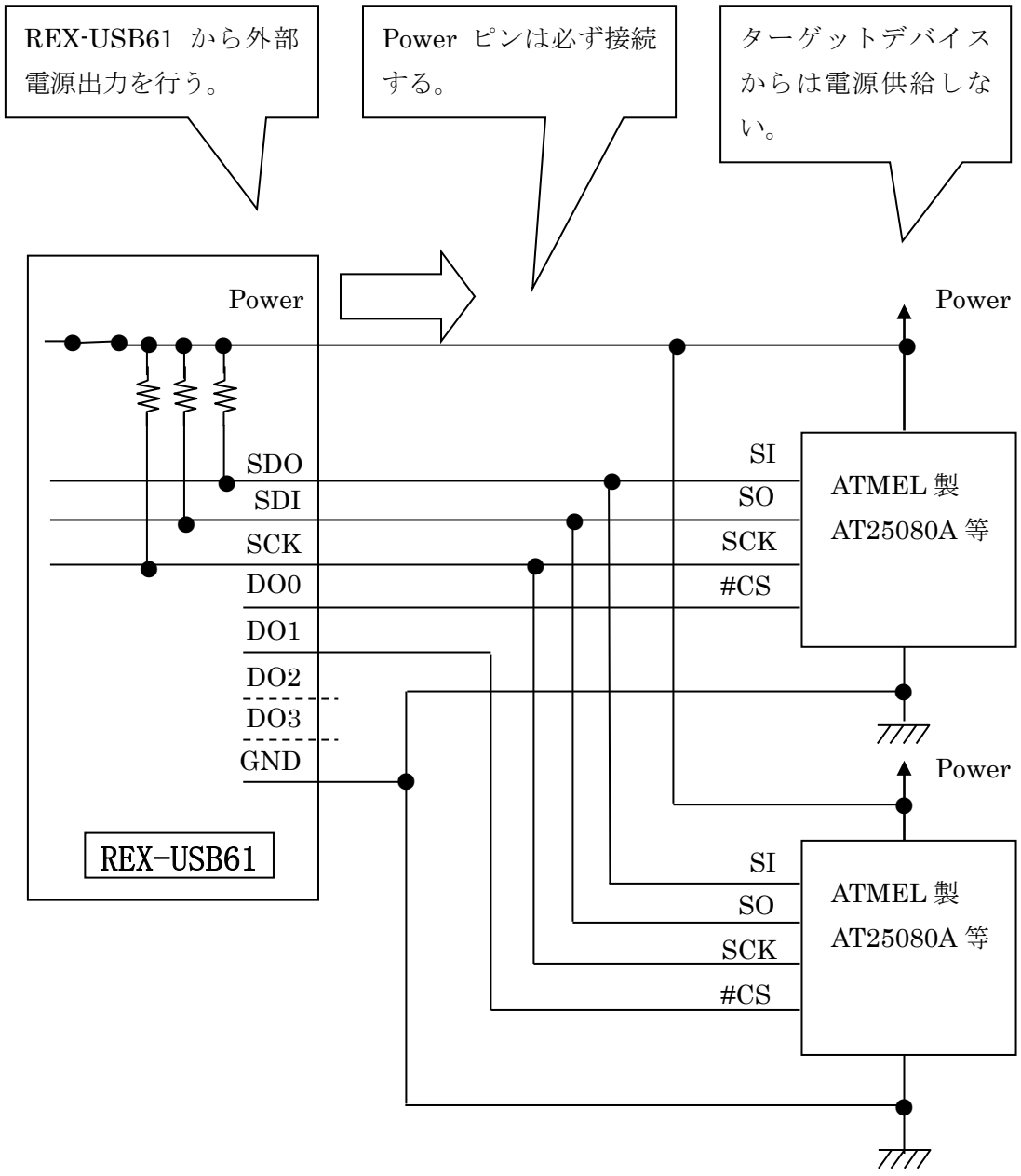
ターゲットデバイスに電源がある場合は、ユーティリティソフトウェアまたはライブラリ関数を利用したアプリケーションにて電源供給を無効にしてください。
(該当するライブラリ関数は usb61_power_control()となります。
参照：(4-4)API 関数詳細)



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(3.3V／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。

(該当するライブラリ関数は usb61_power_control()となります。
参照：(4-4)API 関数詳細)

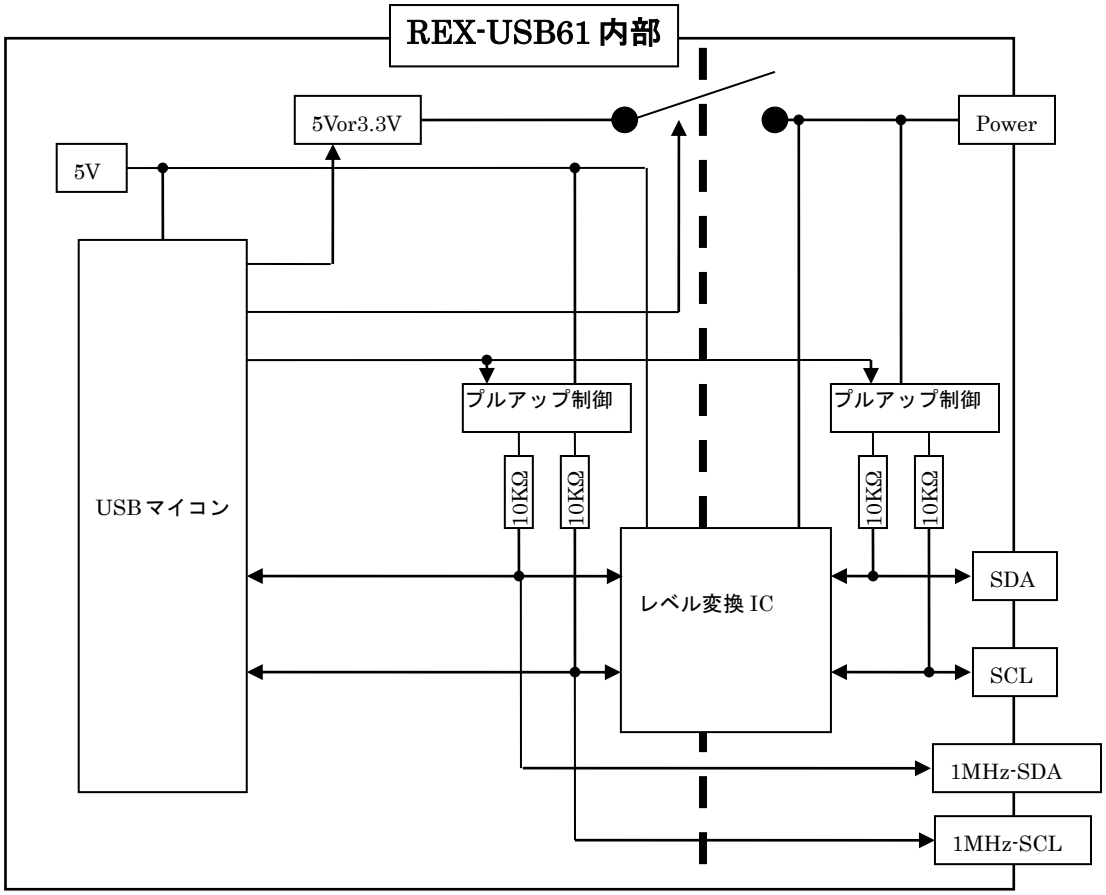


(1-6) I2C デバイスとの接続例について

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-USB61 の電源部分について**

  REX-USB61 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-USB61 の Power ピンをターゲットデバイスの電源と接続してください。**

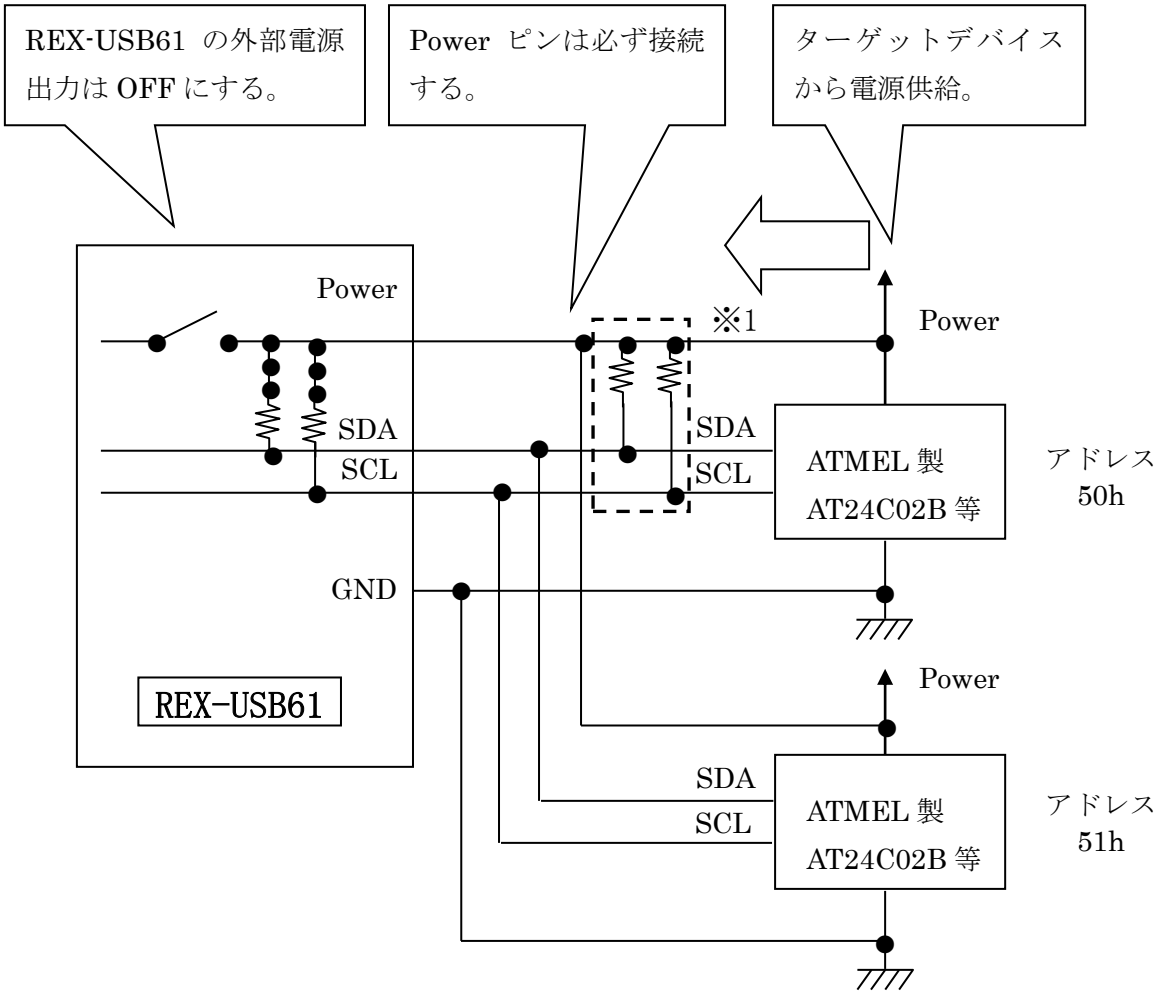


【注意】

接続デバイスの取り付け・取り外し時には、本製品と接続デバイスへの電源供給を行わないでください。
(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 usb61_power_control()となります。
参照：(4-4)API 関数詳細)

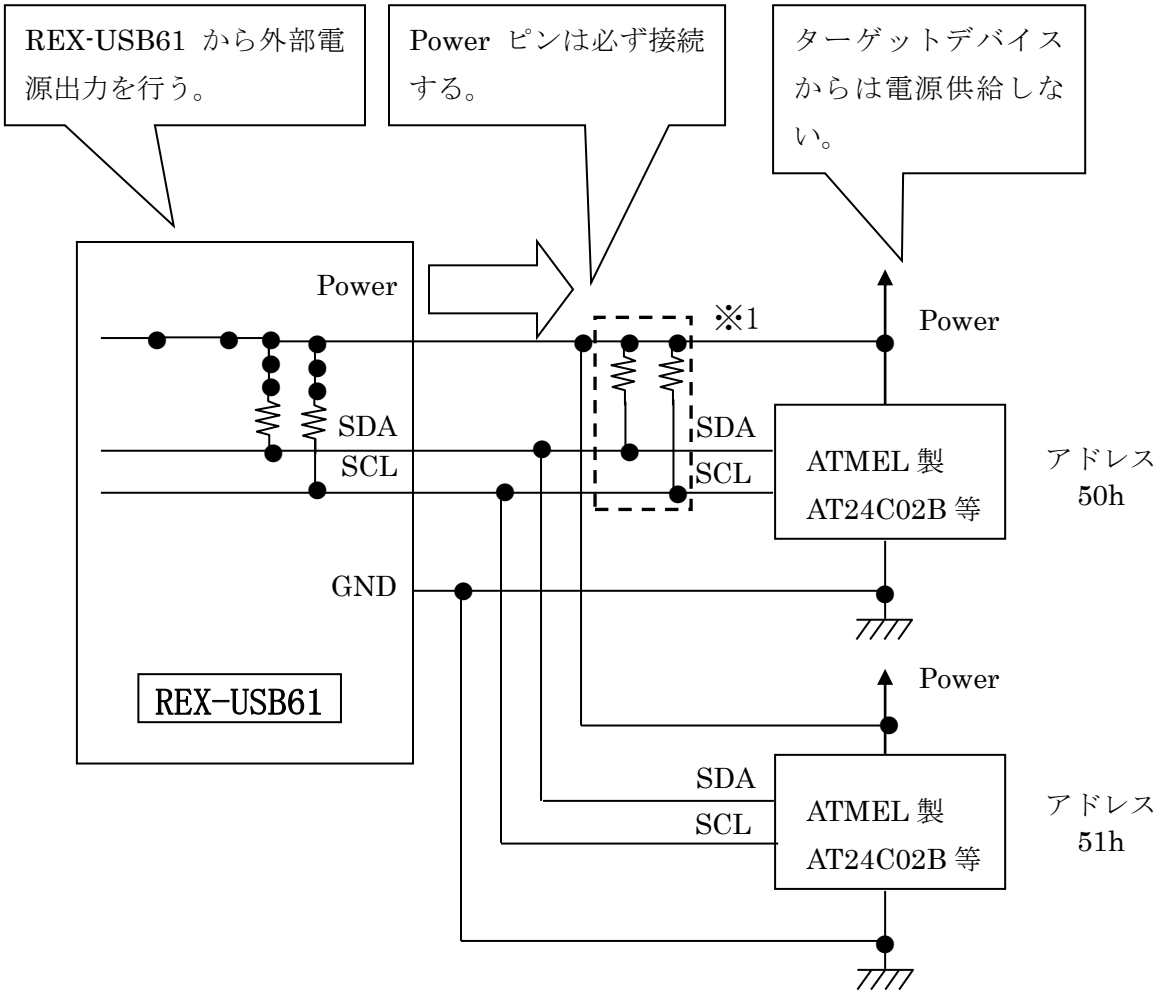


※1 本機の内蔵プルアップ抵抗は 10KΩ です。
必要に応じてプルアップ抵抗を追加してください。

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(3.3V／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。

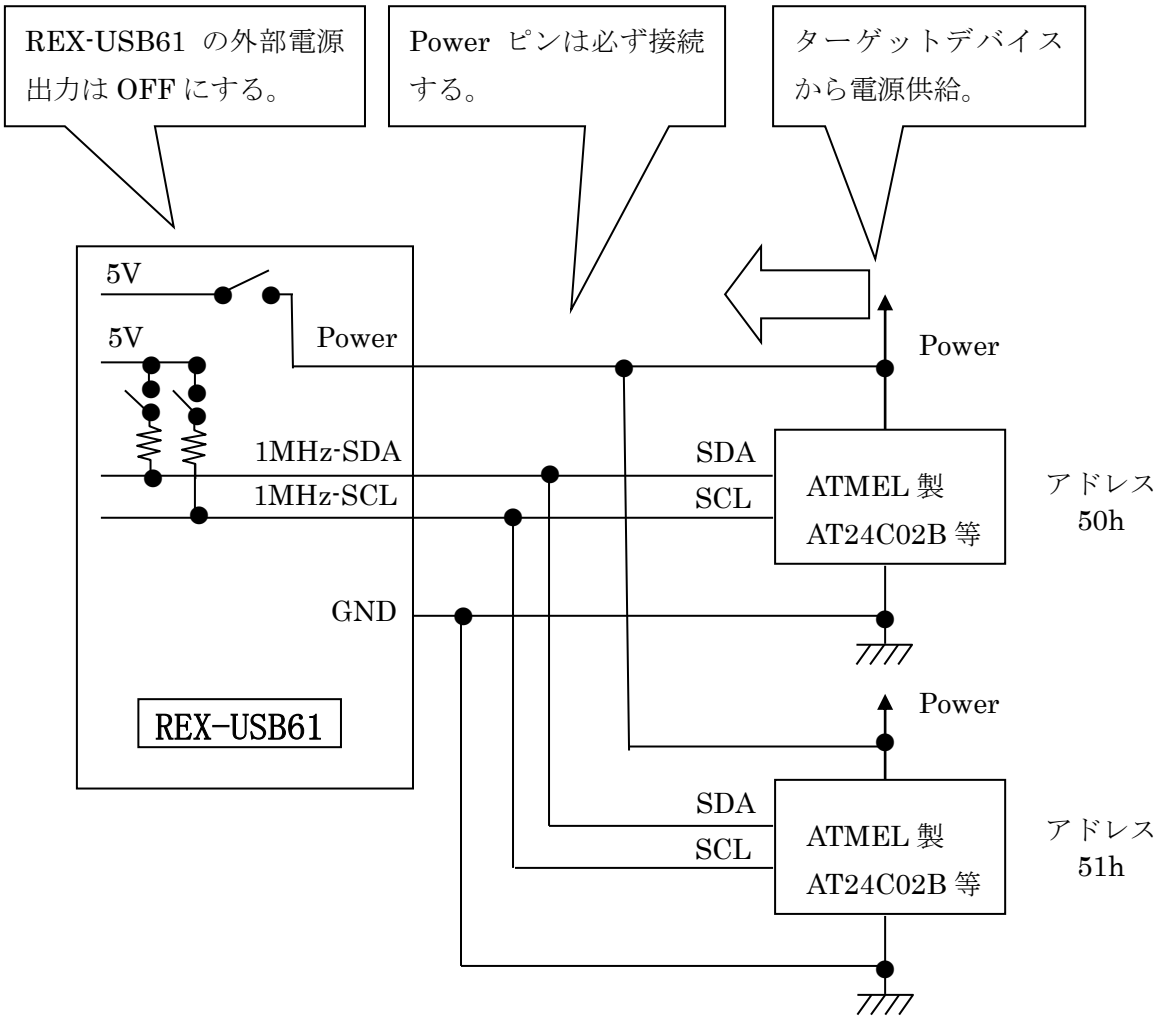
（ 該当するライブラリ関数は usb61_power_control()となります。
参照：(4-4)API 関数詳細 ）



※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［1MHz-SCL / 1MHz-SDA］
(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 usb61_power_control()となります。
参照：(4-4)API 関数詳細)

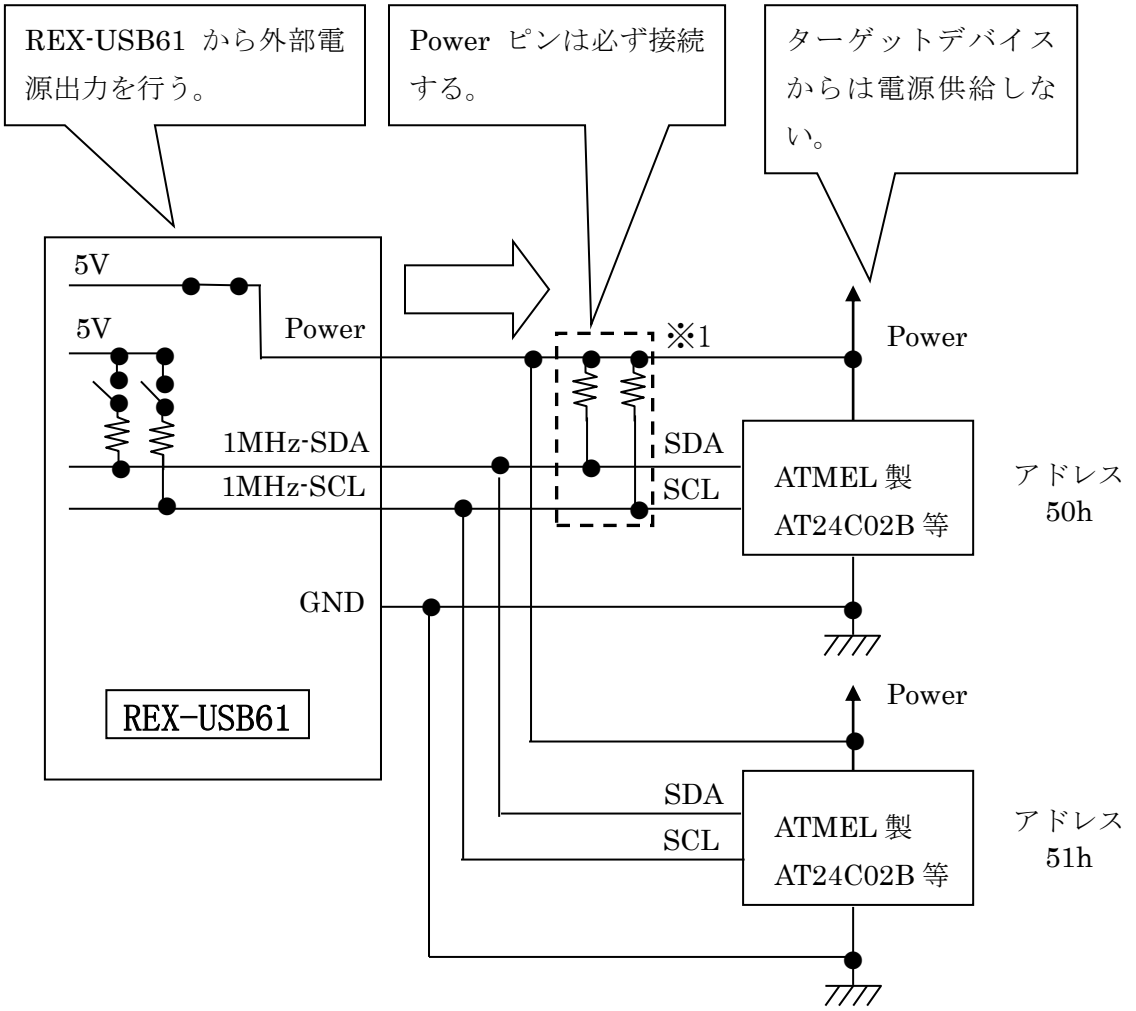


※ プルアップを ON にする場合、全てのデバイスが電源供給された状態で ON にしてください。

※ ターゲットデバイスから電源供給する場合は、I2C バスのプルアップ抵抗を外部に取り付けないでください。

I2C 接続［1MHz-SCL / 1MHz-SDA］
(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(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。
(該当するライブラリ関数は usb61_power_control()となります。
参照：(4-4)API 関数詳細)



※ プルアップを ON にする場合、全てのデバイスが電源供給された状態で ON にしてください。

※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 Windowsセットアップ

● 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

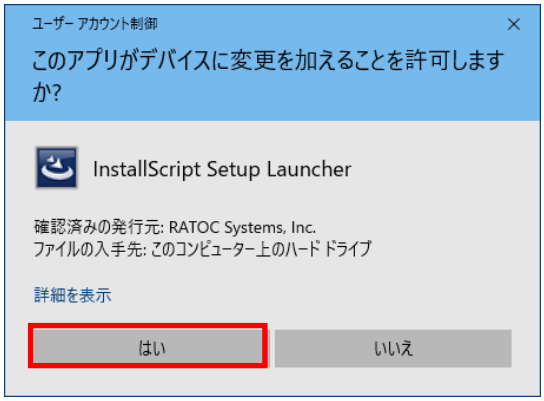
弊社ホームページを開き、画面右上部の検索欄に「USB61 ダウンロード」と入力して検索します。 https://www.ratocsystems.com/

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 11/10/8.1/7/Vista x64 セットアップ

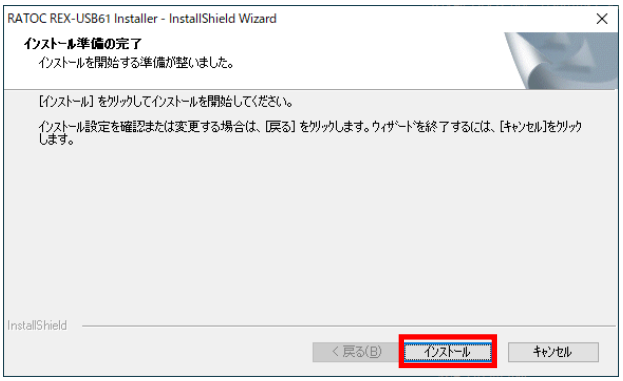
PC の電源を ON にし、本製品を USB ポートへ接続する前に以下の手順にてドライバーのセットアップを行ってください。

ダウンロードした USB61_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」をクリックします。



「RATOC REX-USB61 Installer セットアップへようこそ」で「次へ(N)」をクリックします。

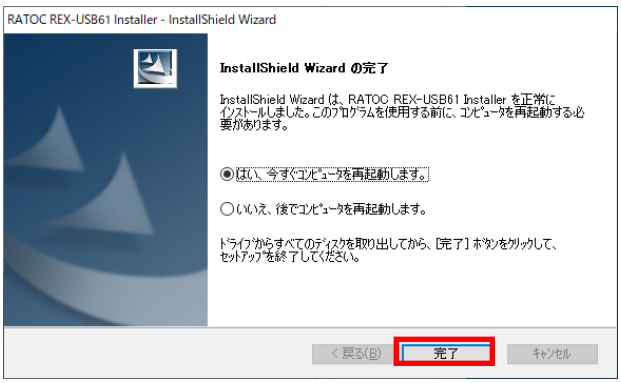
「インストール準備の完了」で「インストール」をクリックします。



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-USB61 を PC に接続すると自動的にインストールが完了します。

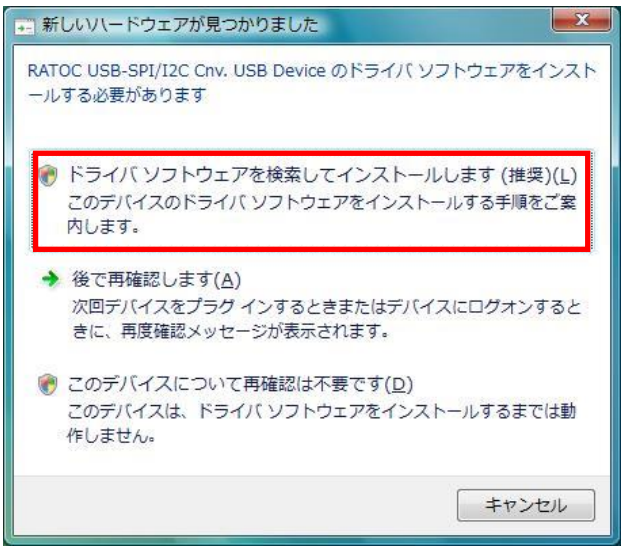


「(2-5) REX- USB61 設定内容の確認」へ進み、インストールの確認を行ってください。

(2-2) Windows Vista x32 セットアップ

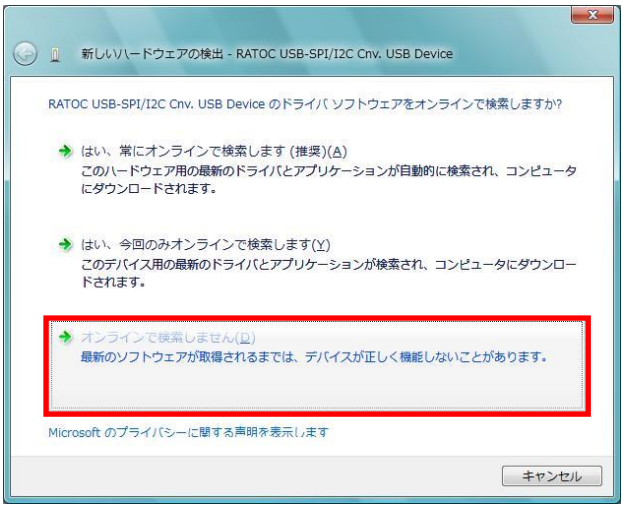
PC の電源を ON にし、本製品を USB ポートへ接続すると、新しいハードウェアの検出ウィザードが起動しますので、以下の手順でインストールを行ってください。

「新しいハードウェアが見つかりました」のダイアログが表示されますので、「ドライバソフトウェアを検索してインストールします(推奨)(L)」をクリックします。

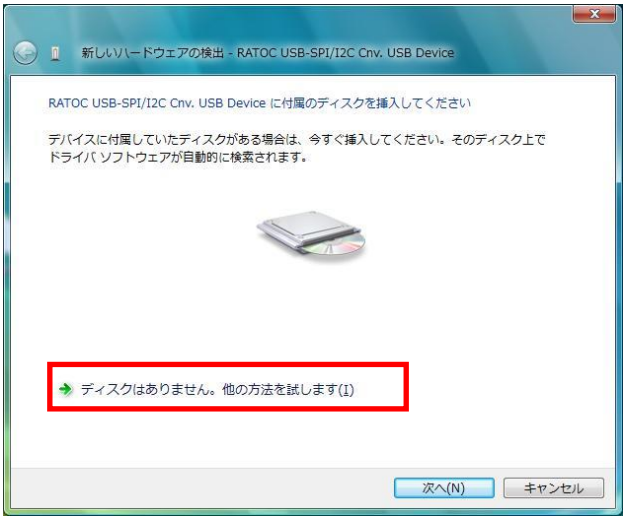


ユーザアカウント制御画面が表示される場合は「はい」をクリックします。

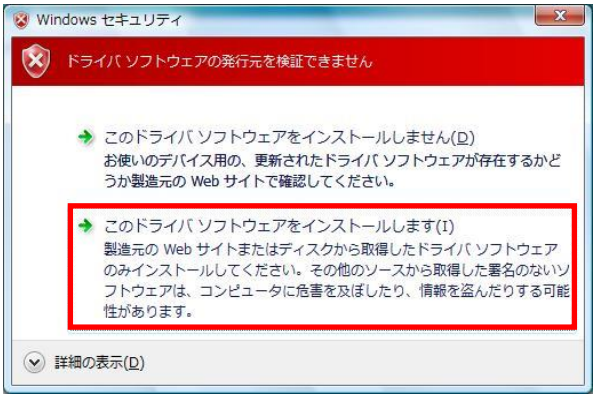
「RATOC REX-SPI/I2C Cnv. USB Device のドライバソフトウェアをオンラインで検索しますか？」で「オンラインで検索しません(D)」をクリックします。



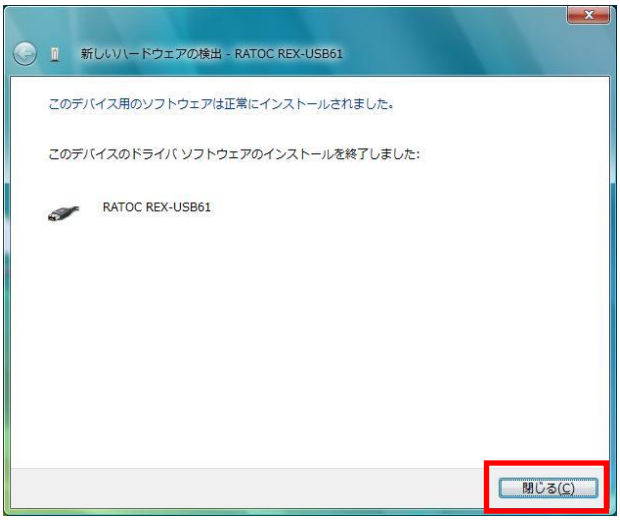
「ディスクはありません。他の方法を試します」を選択し、ダウンロードしたフォルダーを指定します。



Windows セキュリティ画面で、「このドライバソフトウェアをインストールします(I)」をクリックします。



以上で REX-USB61 のインストールは完了です。

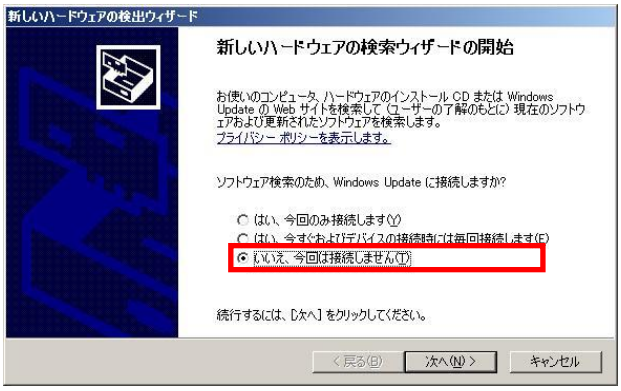


「(2-5) REX- USB61 設定内容の確認」へ進み、インストールの確認を行ってください。

(2-3) Windows XP x32/XP x64 セットアップ

PC の電源を ON にし、本製品を USB ポートへ接続すると、新しいハードウェアの検出ウィザードが起動しますので、以下の手順でインストールを行ってください。

「新しいハードウェアの検索ウィザードの開始」のダイアログが表示されますので、「いいえ、今回は接続しません(T)」を選択し「次へ(N)」をクリックします。



「一覧または特定の場所からインストールする」を選択し、ダウンロードしたフォルダーを指定します。

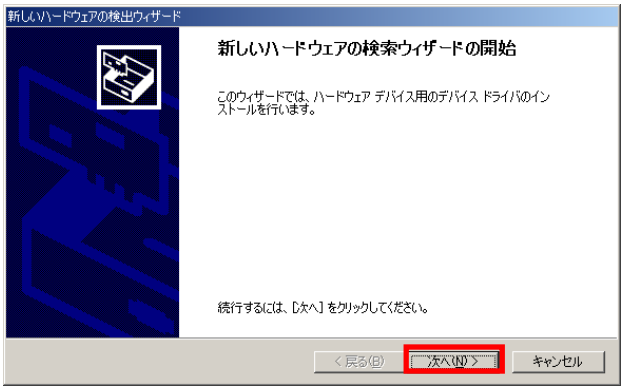
以上で REX-USB61 のインストールは完了です。

「(2-5) REX- USB61 設定内容の確認」へ進み、インストールの確認を行ってください。

(2-4) Windows 2000 セットアップ

PC の電源を ON にし、本製品を USB ポートへ挿入すると、新しいハードウェアの検出ウィザードが起動しますので、以下の手順でインストールを行ってください。

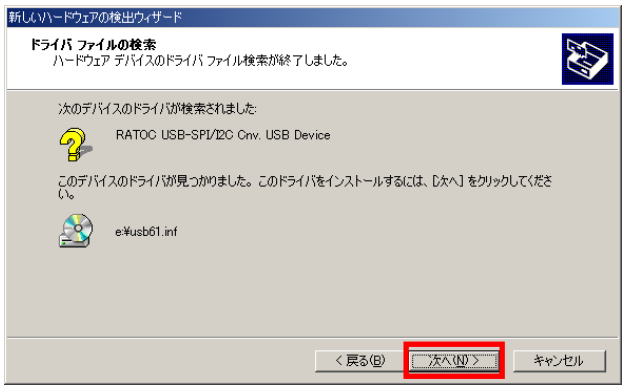
「新しいハードウェアの検索ウィザードの開始」のダイアログが表示されますので、「次へ(N)」をクリックします。



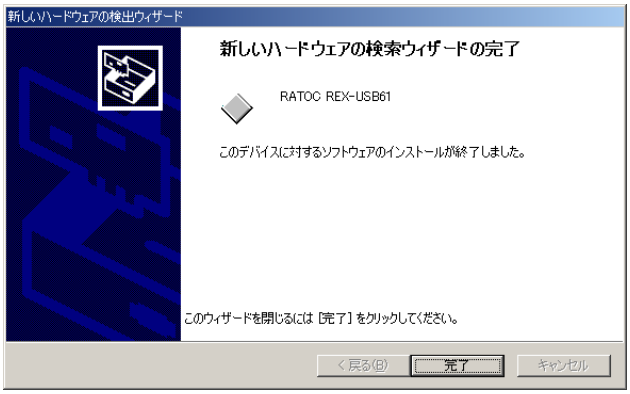
検索方法の選択画面で、「デバイスに最適なドライバを検索する(推奨) (S)」を選択し、「次へ(N)」ボタンをクリックします。

「場所を指定」を選択し、ダウンロードしたフォルダーを指定します。

ドライバファイルが見つかりましたら、「次へ(N)」ボタンをクリックします。



以上で REX-USB61 のインストールは完了です。

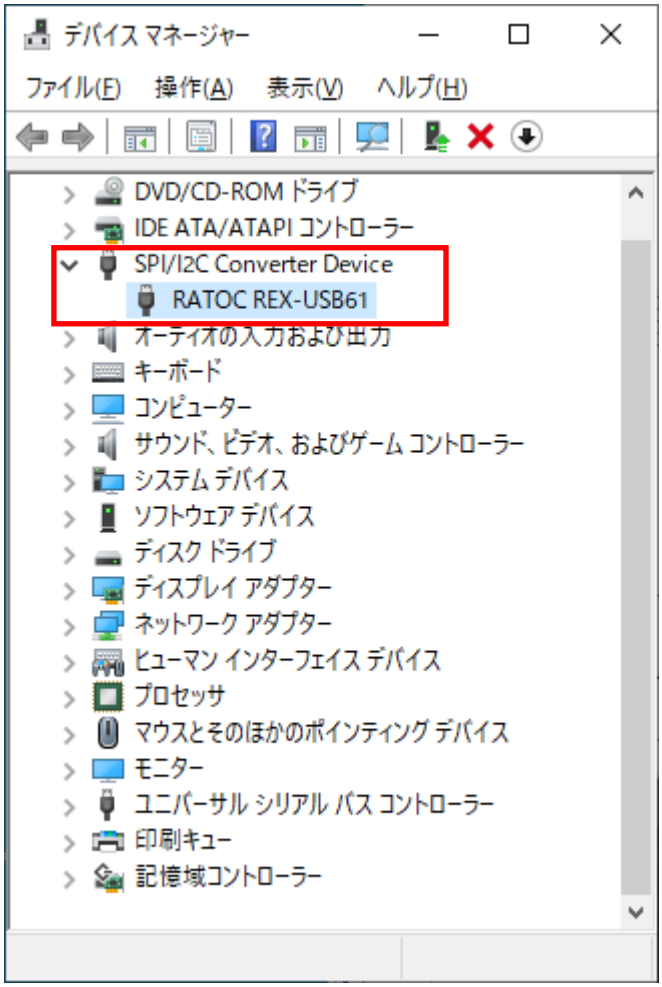


「(2-5) REX- USB61 設定内容の確認」へ進み、インストールの確認を行ってください。

(2-5) REX- USB61 設定内容の確認

コントロールパネルの表示をクラシック表示に切り替え、「デバイスマネージャー」を起動します。
(※ Windows XP x32/XP x64/2000 では、コントロールパネルのシステムを起動し「システムのプロパティ」の「ハードウェア」タブから「デバイスマネージャ」ボタンをクリックします。)

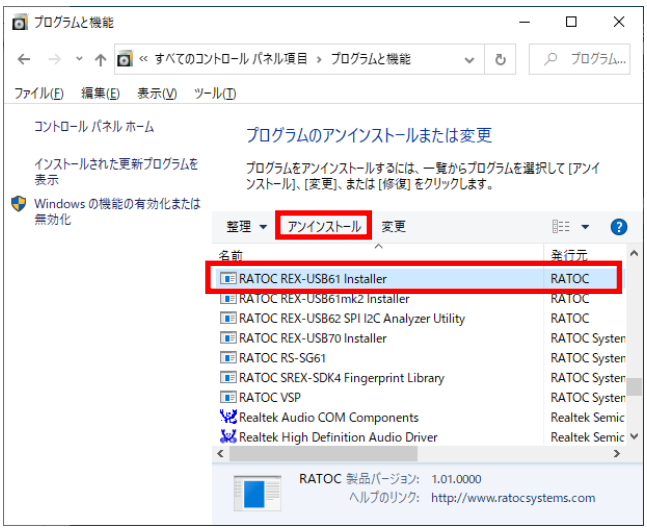
「SPI /I2C Converter Device」クラスの下に「RATOC REX-USB61」が正常に認識されていることを確認してください。



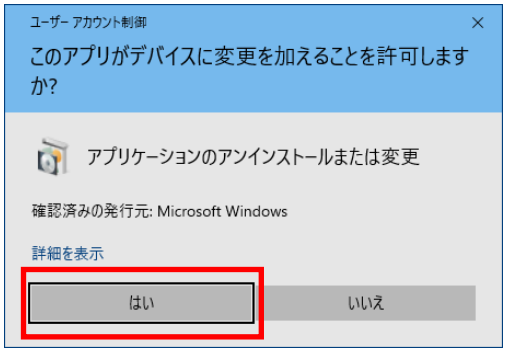
(2-6) Windows 11/10/8.1/7/Vista x64 でのアンインストール方法

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-USB61 Installer」を選択し、「アンインストール」をクリックします。

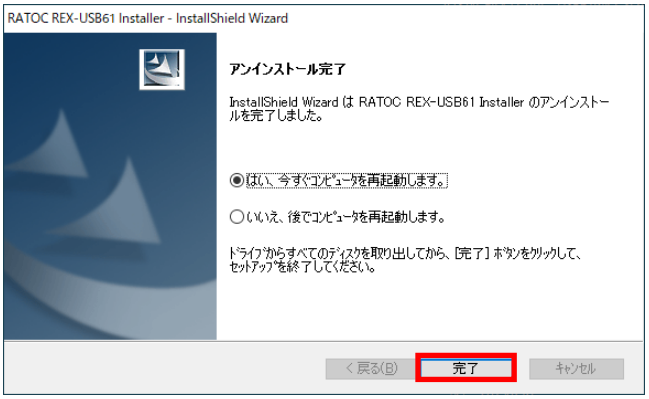


ユーザーアカウント制御画面が表示される場合は「はい」をクリックします。



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-USB61 のアンインストールは完了です。



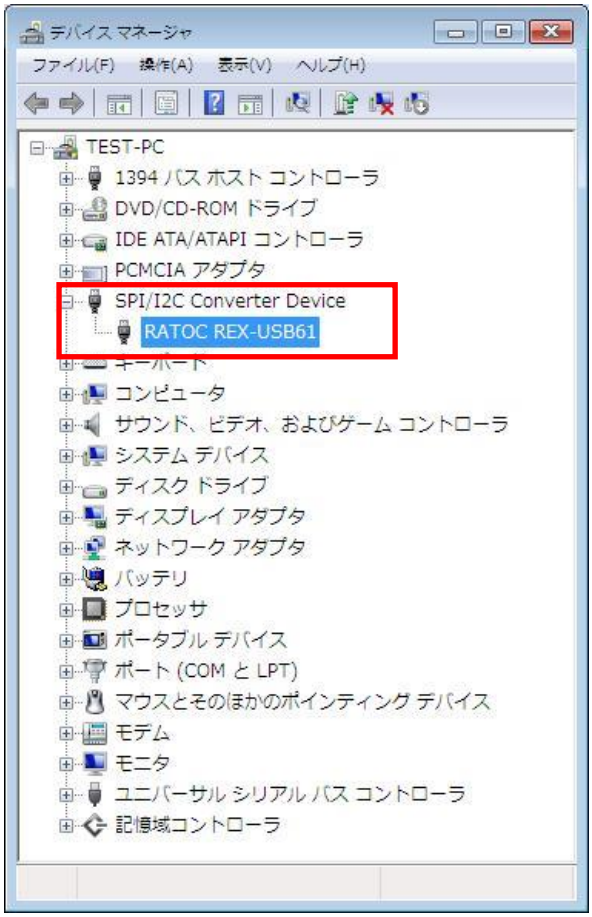
(2-7) Windows Vista x32/XP x32/XP x64/2000 での

アンインストール方法

REX-USB61 のアンインストールを行うためには以下の「ドライバーの削除」と「INF ファイルの削除」を行います。
(Windows Vista では「ドライバーの削除」のみでアンインストールは完了です。)

・ドライバーの削除

コントロールパネルの表示をクラシック表示に切り替え、「デバイスマネージャ」を起動します。
(※ Windows XP x32/XPx64 2000 では、コントロールパネルのシステムを起動し「システムのプロパティ」の「ハードウェア」タブから「デバイスマネージャ」ボタンを押します。)



「RATOC REX-USB61」上で右クリックをし「削除」をクリックしてください。

Windows Vista では「このデバイスのドライバソフトウェアを削除する」にチェックを入れ「OK」ボタンをクリックします。

・INF ファイルの削除

(Windows XP x32/XPx64/2000)

ダウンロードした
Uninst_XP¥USB61_uninst.exe を
実行します。

右図画面が表示されましたら
「OK」ボタンをクリックします。

完了の画面が表示されましたら
「OK」ボタンをクリックしてくだ
さい。

以上で、REX-USB61 のアンインストールは完了です。

第3章 SPI/I2C制御ユーティリティについて

● 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「USB61 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。

https://www.ratocsystems.com › usb61_download

REX-USB61ダウンロード[RATOC] - RATOC Systems

(3-1) ユーティリティの機能について

Usb61Uty.exe は、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、以下の機能を持ちます。

- 動作モードを SPI/I2C 用に切り替える
- SPI デバイスの制御（マスター動作）
- I2C デバイスの制御（マスター／スレーブ動作）
- PORT ピンの制御
- 各種設定値の Read/Write
- 設定ファイルの保存（BIN ファイル形式）
- 設定ファイルのロード
- ログファイルの保存（CSV ファイル形式）

表 3-1 ユーティリティ機能一覧

	機能	
共通項目	ターゲットデバイスへの電源供給	
	データ間の時間間隔の設定	
	転送ログのファイル保存	
	SPI/I2C バスの切り替え	
SPI バス	マスター	クロック極性の指定
		クロック位相の指定
		先行ビットの指定
		周波数の指定
		スレーブセレクトピンの指定（上限 4 つ）
		転送データの作成
		転送データの編集
		転送データの単一送信(ステップ実行)
		転送データの一括送信
		転送データの繰り返し送信
		転送データのファイル保存
		保存ファイルからの転送データの読み出し
I2C バス	マスター	周波数の指定
		転送データの作成
		転送データの編集
		転送データの単一送信(ステップ実行)
		転送データの一括送信
		転送データの繰り返し送信
		転送データのファイル保存

		保存ファイルからの転送データの読み出し
		バスリセットの発行
		PORT ピンへの出力
	スレーブ	周波数の指定
		マスターへのレスポンスデータの指定
		スレーブアドレスの指定

(3-2) ユーティリティの説明

本ユーティリティの画面および各機能について説明いたします。

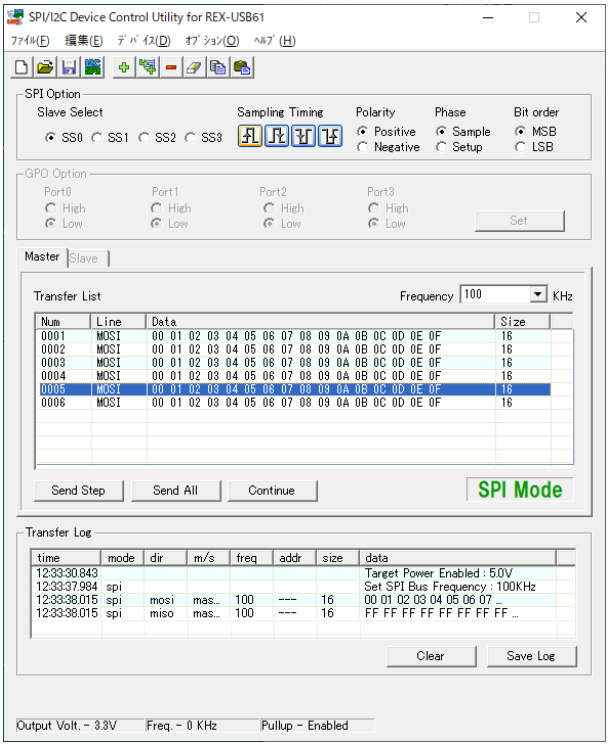


図 3-1. SPI マスターモード

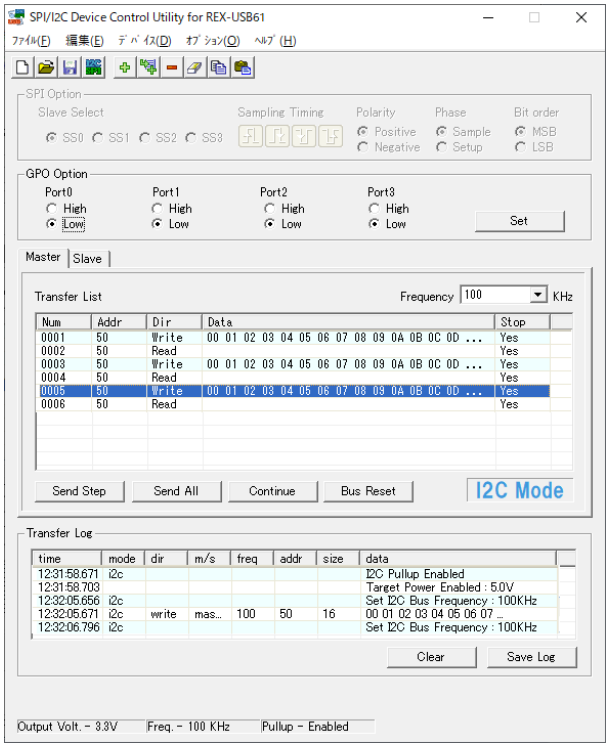


図 3-2. I2C マスターモード

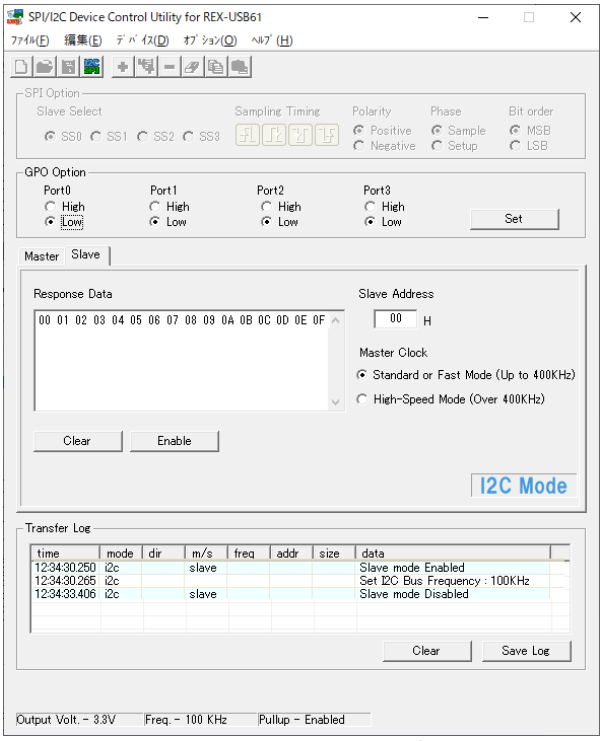


図 3-3. I2C スレーブモード

メニューバーについて

ファイル(F)

- 新規作成　：新しい設定ファイルを作成する。
- 開く　：設定ファイルを開く。
- 上書き保存　：現在の設定を上書き保存する。
- 名前をつけて保存　：現在の設定を新しく名前を付けて保存する。
- アプリケーションの終了　：アプリケーションを終了する。

※ 終了処理以外はマスター動作時しか扱うことができません。

編集(E)

- 追加　：新たな転送データを転送リストの最後尾へ追加する。
- 挿入　：新たな転送データを現在選択している転送データ番号へ挿入する。
- 削除　：選択した転送リストを削除する。
- 消去　：選択した転送リストの内容を消去する。
- コピー　：選択した転送リストの内容をコピーする。
- 貼り付け　：コピーした転送リストの内容を選択した番号へ貼り付ける。

※ 本処理はマスター動作時しか扱うことができません。

デバイス(D)

- SPI/I2C 切替　：SPI/I2C バスの切り替えを行う。

オプション(O)

- 各種設定　：I2C バスの信号線のプルアップ状態の切り替えを行う。
  デバイスへの電源供給の有効／無効の切り替えを行う。
  電源電圧の指定を行う。(3.3V、5.0V)
  1 バイト毎の時間間隔を設定する。
- リストビュー/スクリプト切替　：リスト表示とスクリプト表示の切替を行います。

ヘルプ(H)

- バージョン情報　：本アプリケーションのバージョン情報を表示する。

ツールバーについて

 メニューバー[ファイル]の「新規作成」と同じ。
 メニューバー[ファイル]の「開く」と同じ。
 メニューバー[ファイル]の「上書き保存」と同じ。
 SPI/I2Cのモード切り替えを行う。
 メニューバー[編集]の「追加」と同じ。
 メニューバー[編集]の「挿入」と同じ。
 メニューバー[編集]の「削除」と同じ。
 メニューバー[編集]の「消去」と同じ。
 メニューバー[編集]の「コピー」と同じ。
 メニューバー[編集]の「貼り付け」と同じ。

各種コントロールについて

SPI Option

- SPI Option ：SPI 動作時の設定を行う。
- Slave Select ：スレーブセレクトピンの選択を行う。
- Sampling Timing ：クロックのどの状態でデータのサンプリングを行うかを指定する。
- Polarity ：極性の選択。正パルス(Positive)か負パルス(Negative)を選択。
- Phase ：位相の選択。サンプリング先行(Sampling)かセットアップ先行(Setup)かを選択。
- Bit order ：データ順の選択を行う。MSB 先行か LSB 先行かを選択。

※ Sampling Timing と Polarity, Phase の設定は互いに連動します。

GPO Option

- Port0～3 ：PORT の各ポートの設定を行う（出力のみ)。
- High／Low ：各ポートの値の設定／表示を行う。
- Set ：Port の各ポートへ出力を行う。

Master

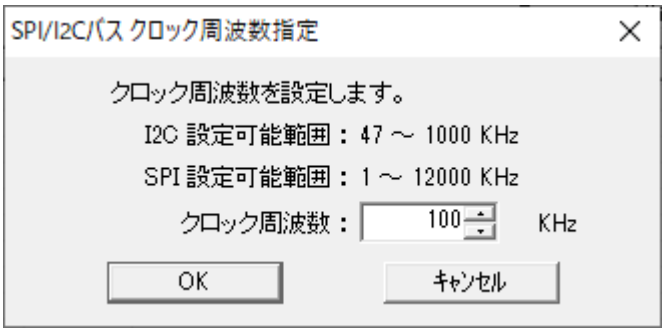
- Transfer List ：設定した転送内容の表示を行う。
  - ① Num ：転送データ番号。
  - ② Addr ：デバイスアドレス。
  - ③ Dir ：転送方向。Read あるいは Write と表示される。
  - ④ Line ：データライン名の表示。MOSI あるいは MISO と表示される。
  - ⑤ Data ：データ内容を表示。
  - ⑥ Stop ：ストップコンディションを発行するか否かの表示。
  - ⑦ Size ：データサイズ。
- Send ：選択したデータを転送する。
- Send All ：リストビュー内の設定項目をすべて転送する。
- Continue ：リストビュー内の設定項目を繰り返し連続で転送する。
- Bus Reset ：I2C のバスリセットを発行する。

Slave

- Slave Address ：スレーブアドレスを指定する。
  注) アドレスの指定方法は Page.4-16 をご参照ください。
- Response Data ：マスター側へ返送するデータを指定する。
- Clear ：返送データを消去する。
- Enable ：スレーブ動作を有効にする。
- Master Clock ：[Standard or Fast Mode (Up to 400KHz)]
  動作周波数(400KHz までの場合)
  [High-Speed Mode (Over 400KHz)]
  動作周波数(400KHz を超える場合)

Master/Slave共通

- Frequency ：周波数の設定／表示を行う。
  「指定周波数」を選択すると 1KHz 単位で設定可能ですが、SPI では実際には設定可能な近似値が設定されます。
  (I2C:47KHz～1MHz / SPI:12MHz まで)
  ※ 近似値の計算方法については「第 4 章 API 関数仕様」の usb61_spi_set_freq()関数の機能欄をご参照ください。



［クロック周波数指定画面］

- Device Mode ：現在の動作モードを表示。
  (SPI Mode あるいは I2C Mode と表示)
- Output Volt ：現在の出力電圧を画面左下へ表示。
- Freq ：現在の動作周波数を画面左下へ表示。
- Pullup ：現在の I2C バスプルアップ状態を画面左下へ表示。

Log

- Transfer Log ：転送内容のログ表示を行う。
  ① time ：ログを追加した時間を表示(hh:mm:ss:msec)。
  ② mode ：SPI/I2C 転送モードの表示(spi/i2c)。
  ③ dir ：転送方向の表示(read/write, miso/mosi)。
  ④ m/s ：マスター/スレーブモードの表示(master/slave)。
  ⑤ freq ：動作周波数の表示(KHz 単位)。
  ⑥ addr ：I2C スレーブアドレスの表示(16 進表示)。
  ⑦ size ：データ転送長の表示(10 進表示)。
  ⑧ data ：転送データの表示(8 バイト以降は省略して表示されます)。
- Clear ：転送ログの内容を消去する。
- Save Log ：ログの保存を行う (CSV ファイル形式)。

転送データ編集画面

リストビュー(Transfer List)内の項目をダブルクリックすることで転送データの編集画面が表示されます。

図 3-4. SPI モード 転送データ編集画面

図 3-5. I2C モード 転送データ編集画面

- I2C 設定 ：I2C バスの転送設定を行う。
  - ① スレーブアドレス：デバイスアドレスの指定。16 進数で入力。
    注）アドレスの指定方法は Page.4-16「スレーブアドレスについて」をご参照ください。
  - ② 10bit ：10 ビットアドレスを指定する場合はチェックを入れる。
    注）アドレスの指定方法は Page.4-16「スレーブアドレスについて」をご参照ください。
  - ③ 転送方向 ：転送方向の指定。Read あるいは Write で指定する。
  - ④ ストップコンディション：ストップコンディションを発行するか否かを指定。

- 転送データ設定 ：転送データ内容あるいはファイル名を表示。(16 進表示)
  - ① バイナリで指定 ：エディットボックスへ直接転送するデータを入力。
  - ② ファイルから指定：バイナリファイルからデータを指定。
  - ③ ファイルを選択 ：バイナリファイルを選択。
  - ④ データ転送長 ：データサイズ（10 進表示)。最大 65535 バイト。

オプション設定

メニューから「オプション(0)」→「各種設定」を選択すると，次の各設定を行うことができます。

- I2Cバスのプルアップ設定
- ターゲットデバイスへの電源供給設定
- データ間のインターバル設定

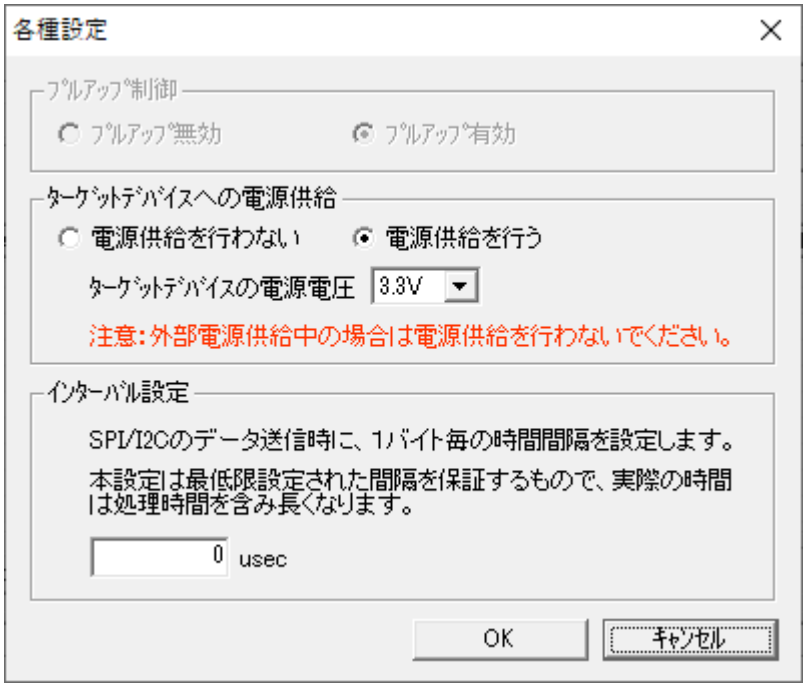


図 3-6. オプション設定画面

・プルアップ無効／有効　　：I2C バスラインのプルアップを行うか否かを選択。
(5V、1MHz ピン[401KHz～1000KHz]使用時の I2C のみ選択可)

・ 電源供給を行わない／行う　　：ターゲットデバイスへの電源供給を行うか否かを選択。
：3.3V 又は、5.0V から選択。

注意：外部電源供給中の場合は電源供給を行わないでください。

・ インターバル設定　　：データ送信時、1 バイト毎の時間間隔を設定します。

(3-3)　ユーティリティを使用した制御例について

※ 以下の説明は、ATMEL 社製 AT24C02B、AT25080A を使用した例となります。

・　SPI マスターモード

[SPI/I2C 切替]

SPI/I2C 切替により、SPI Mode に切替えます。
(「SPI Mode」の部分をクリックしても切替可能)

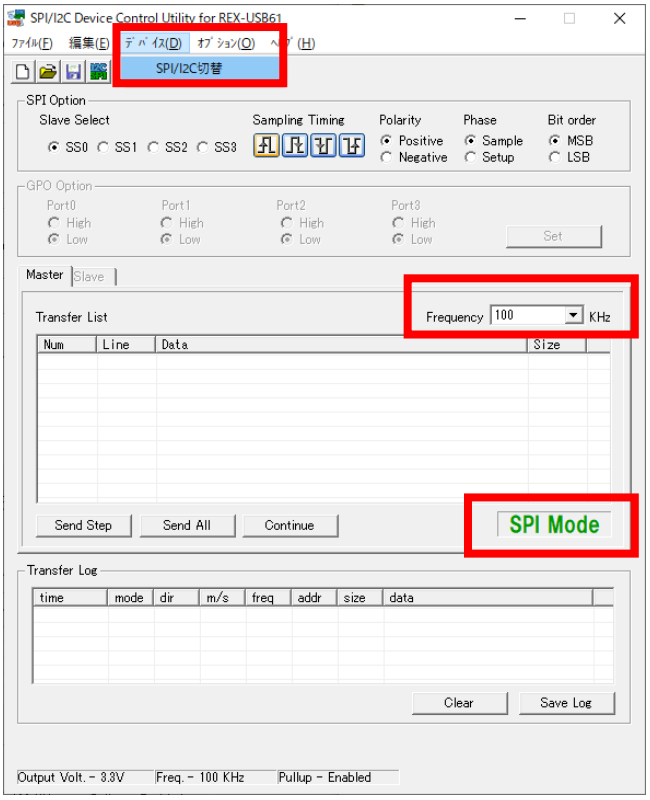


図 3-7.　ユーティリティ初期設定

[周波数設定]

Frequency にある周波数を設定します。

「指定周波数」を選択すると 1KHz 単位で設定可能ですが、SPI では実際には設定可能な近似値が設定されます。
(I2C:47KHz～1MHz / SPI:12MHz まで)
※ 近似値の計算方法については「第 4 章 API 関数仕様」の usb61_spi_set_freq()関数の機能欄をご参照ください。

[電源供給の設定]

「オプション」-「各種設定」より電源供給の設定を行います。

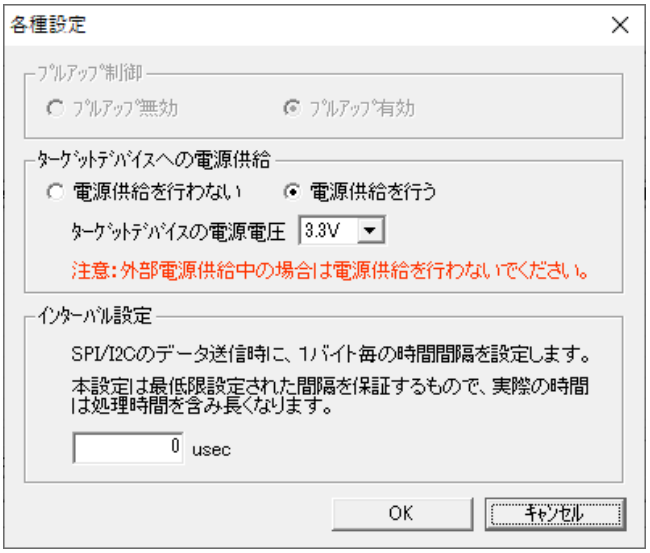


図 3-8.　電源供給の設定

[インターバル設定]

送信するデータの 1 バイト毎の時間間隔を設定します。

例：50h 番地に 11 22 33 44 55 66 77 88 を書き込み、50h 番地から 8byte 分のデータを読み込む。

[データ入力（Write / Read)]

「Transfer List」内をダブルクリックし、書き込むデータを 16 進数で入力します。例では次のようになります。

(1 行目)
06h --- Write Enable ビットをセットします。
(2 行目)
02h --- Write 命令
00h 50h --- 書き込む番地
11h 22h.. --- 書き込むデータ（8byte)
(3 行目)
03h --- Read 命令
00h 50h --- 読み込む番地
00h 00h.. --- Read 用のダミーデータ(8byte)
(この場合、8byte のデータが返されます。)

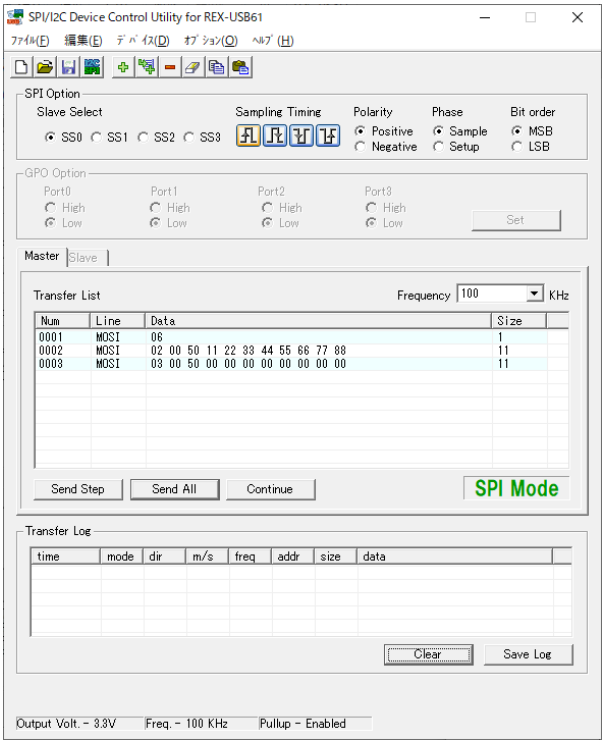


図 3-9. データ入力

[実行（Write / Read)]

「Send All」ボタンにより「Transfer List」内に記述したデータが送信されます。

「Transfer Log」内には送受信されたデータのログが表示されます。

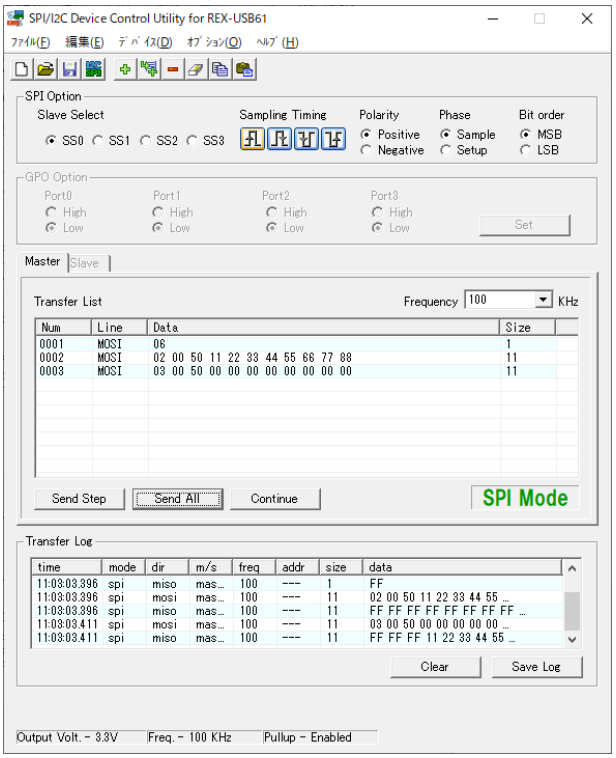


図 3-10. 実行

・ I2C マスターモード

[SPI/I2C 切替]
[周波数設定]
[電源供給の設定]
[インターバル設定]
Page. 3-8 での手順と同様に、I2C Mode に切替え、周波数/電源供給/インターバルの設定を行います。
(Master タブを選択状態にしておきます。)

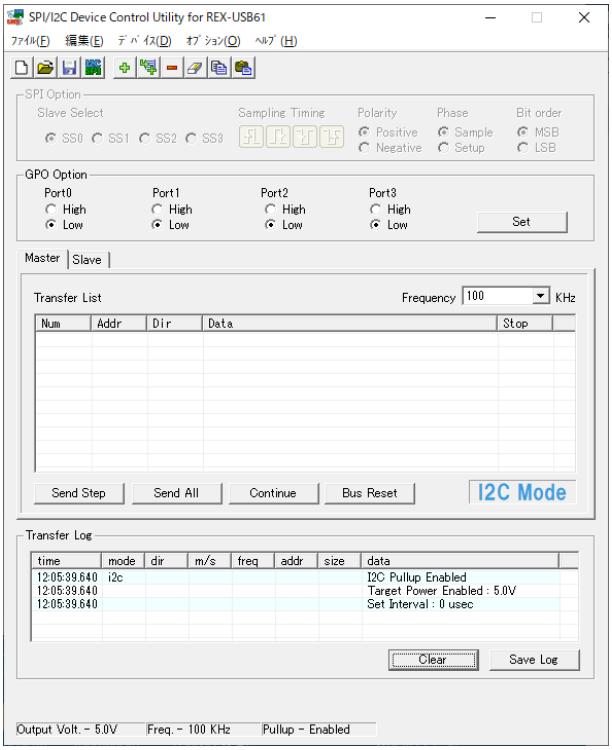
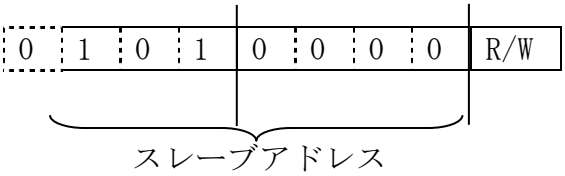


図 3-11. ユーティリティ初期設定

※ スレーブアドレスについて
(R/W ビットは含みません。)
[例：50h の場合]



例：スレーブアドレス 50h のデバイスの 0000h 番地に 11 22 33 44 を書き込み、
スレーブアドレス 50h のデバイスの 0000h 番地から 4byte 分のデータを読み込む。

[データ入力 (Write / Read)]
「Transfer List」内をダブルクリックし、各項目を設定します。
・スレーブアドレス --- 7bit で指定。
※ 設定方法については、上記「スレーブアドレスについて」をご参照ください。
(10bit 指定を行う場合は「10bit」にチェックを入れます。)
・ 転送方向 --- Write / Read を選択します。
・ ストップコンディション --- ストップコンディションの発行を設定します。
・ 転送データ --- 16 進で指定。
・ データ転送長 --- Write の場合は転送データ長が自動的に表示され、Read の場合は読み込むデータサイズを指定します。(Byte)

図 3-12. 転送データの編集

例では次のようになります。
(1 行目) <Write>
・ スレーブアドレス --- 50 h (7bit 指定)
・ 転送方向 --- Write
・ ストップコンディション --- あり(Yes)
・ 転送データ
00h 00h --- 書き込む番地
11h 22h.. --- 書き込むデータ
(2 行目) <Read のための Write>
・ スレーブアドレス --- 50 h (7bit 指定)
・ 転送方向 --- Write
・ ストップコンディション --- なし(No)
・ 転送データ
00h 00h --- Read する番地
(3 行目) <Read>
・ スレーブアドレス --- 50 h (7bit 指定)
・ 転送方向 --- Read
・ ストップコンディション --- あり(Yes)
・ データ転送長 --- 4(表示されません)

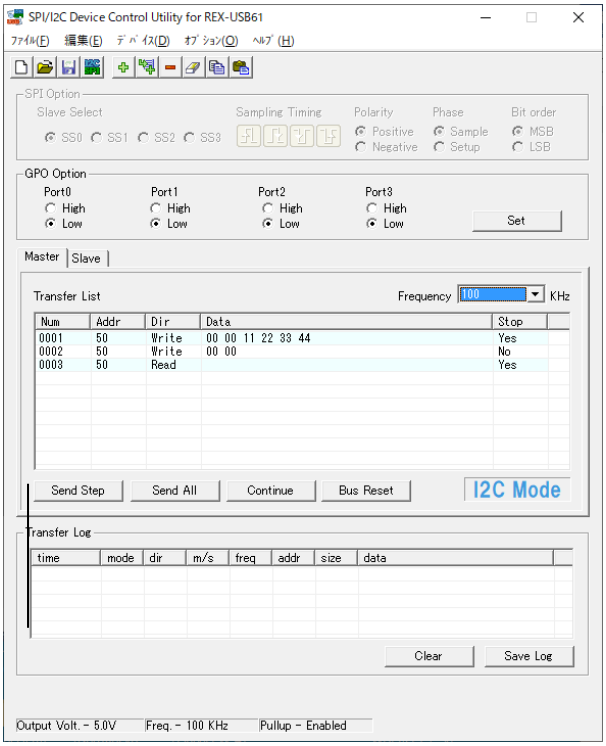


図 3-13. データ入力

[実行 (Write / Read)]
「Send All」ボタンにより「Transfer List」内に記述したデータが送信されます。

「Transfer Log」内には送受信されたデータのログが表示されます。

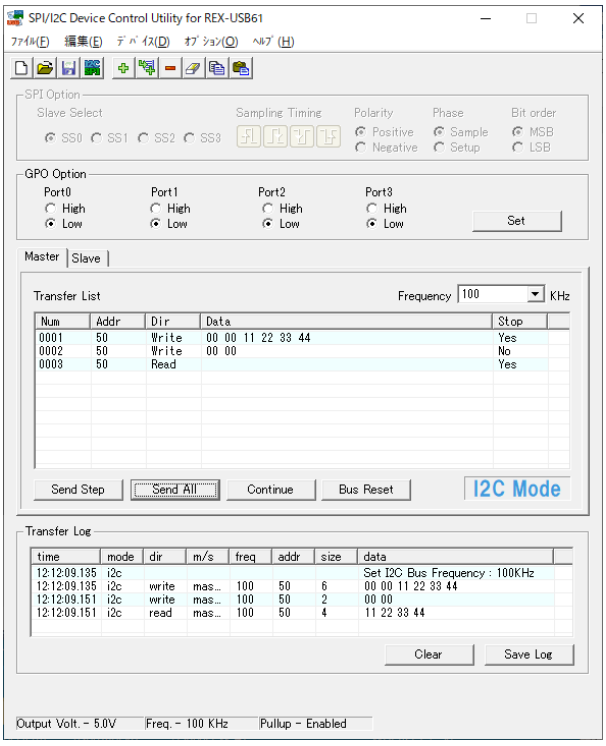


図 3-14. 実行

・ I2C スレーブモード

例：スレーブアドレス 50h に転送されたデータを読み込む。

[SPI/I2C 切替]
[周波数設定]
[電源供給の設定]
[インターバル設定]

Page. 3-8 での手順と同様に、I2C Mode に切替え、周波数/電源供給/インターバルの設定を行います。
(Slave タブを選択状態にしておきます。)

「Slave Address」にスレーブアドレスを 7bit で指定し、
「Master Clock」の動作周波数を選択します。

マスター側からデータが転送されると、「Response Data」および「Transfer Log」に Read データが表示されます。

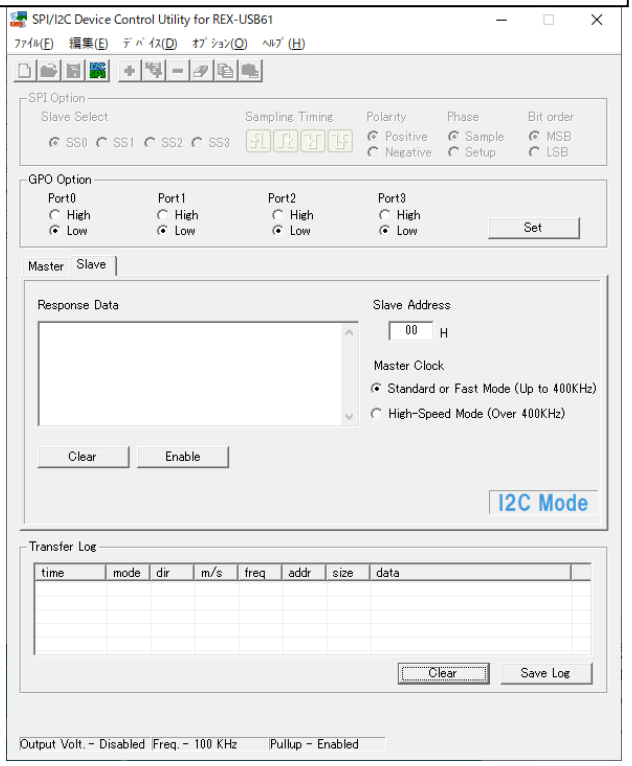


図 3-15. I2C スレーブモード設定

(3-4) スクリプト記述の文法について

本ユーティリティには、デバイスへのアクセスを記述したスクリプトファイルに対応しております。スクリプトファイル内にはコメント文の記述も可能で、スクリプトファイルを使用することでデバイスへの一連のアクセスを行うことができます。

スクリプトファイル内では以下の規則に従って記述する必要があります。

◆ SPI/I2C 共通の文法

SPI と I2C の両方に共通な文法は次の通りです。

- 数値の定義
  数値は 10 進数、16 進数で記述します。指定できる数値は 0～65536 までとし、16 進数を表す場合は数値の後ろに[h]または[H]をつけます。数値を連続して記述する場合は数値と数値の間に[,]を入れます。
- 文字の定義
  アルファベット文字の大文字/小文字の区別はありません。コメントは日本語に対応しています。
- 文法
  命令文と命令文または数値の間にはスペース(半角)または TAB を入れてください。
  全角文字は文法エラーとなります。

表 3-2 共通命令表

命令	#
意味	行中の「#」以降はコメント文として取り扱います。
パラメータ	なし
命令	MODE=
意味	SPI、I2C のモードを指定します。 MODE 命令にデフォルト設定はなく、指定されていない場合は文法エラーとなります。設定後は途中でモードを切り替えることはできません。
パラメータ	SPI I2C

<table>
<tr><td>命令</td><td>FREQUENCY=</td></tr>
<tr><td>意味</td><td>使用する周波数を設定します。
周波数は 1KHz 単位で設定が可能ですが、SPI では実際に設定される周波数は、本製品の周波数計算仕様による近似値となります。
設定値と実際に設定される周波数の関係については「第 4 章 API 関数仕様」の usb61_spi_set_freq()関数の機能欄をご参照ください。

周波数設定を行わない場合の初期値は下記となります。
<table><tr><td>モード</td><td>周波数</td></tr><tr><td>SPI</td><td>100KHz</td></tr><tr><td>I2C</td><td>100KHz</td></tr></table>周波数はいつでも変更することができます。</td></tr>
<tr><td>パラメータ</td><td>SPI、I2C で下記の設定が可能です。
<table><tr><td>モード</td><td>設定値</td></tr><tr><td>SPI</td><td>1～12000</td></tr><tr><td>I2C</td><td>47～ 1000</td></tr></table></td></tr>
<tr><td colspan="2"></td></tr>
<tr><td>命令</td><td>INTERVAL=</td></tr>
<tr><td>意味</td><td>送信するデータのバイト間に入る待ち時間間隔を設定します。(単位はμS)

待ち時間設定を行わない場合の初期値は 0 となります。</td></tr>
<tr><td>パラメータ</td><td>0～65535 の数値を指定(0μS～65535μS)</td></tr>
<tr><td colspan="2">※ 本設定は最低限設定された間隔を保証するもので、実際の時間は処理時間を含み長くなります。</td></tr>
</table>

<table>
<tr><th>命令</th><td>POWER=</td></tr>
<tr><td>意味</td><td>パラメータで設定された電源出力を行います。
電源はいつでも変更することができます。
電源設定を行わない場合の初期値は「出力 OFF」となります。</td></tr>
<tr><td>パラメータ</td><td><table><tr><th>出力</th><th>設定値</th></tr><tr><td>出力 OFF</td><td>OFF</td></tr><tr><td>出力 3.3V</td><td>ON3</td></tr><tr><td>出力 5.0V</td><td>ON5</td></tr></table></td></tr>
<tr><td colspan="2"></td></tr>
<tr><th>命令</th><td>WAIT=</td></tr>
<tr><td>意味</td><td>次の命令を実行するまでの待ち時間を設定します。
単位は 100mS です。(100mS～60 秒)</td></tr>
<tr><td>パラメータ</td><td>1～600 の数値を指定 (100mS～60 秒)</td></tr>
<tr><td colspan="2"></td></tr>
<tr><th>命令</th><td>REPEAT=nn</td></tr>
<tr><td>意味</td><td>REPEAT 命令文の次に書かれた{ }内の命令を指定回数繰り返します。
{ }がない場合は直後の命令のみを繰り返します。
※ 使用方法については Page.3-17「REPEAT コマンドの使用方法について」をご参照ください。</td></tr>
<tr><td>パラメータ</td><td>nn=1～65536{…}</td></tr>
<tr><td colspan="2"></td></tr>
<tr><th>命令</th><td>PULLUP=</td></tr>
<tr><td>意味</td><td>SDA、SCL 信号線の Pull-up 設定を行います。
初期値は Pull-up する(ON)となり、
電源電圧 5V、周波数 1MHz の設定時のみ OFF とすることが可能です。
上記の設定以外で本命令を実行するとエラーとなります。</td></tr>
<tr><td>パラメータ</td><td>ON または OFF</td></tr>
<tr><td colspan="2"></td></tr>
<tr><th>命令</th><td>FILEn</td></tr>
<tr><td>意味</td><td>n にはファイル番号が入り、最大 5 ファイルまで使用可能です。
“”(半角のダブルクォーテーション)で囲ったファイルからデータの送受信を行います。
データはバイナリデータとして扱います。
ファイルの指定は、フルパスではなくファイル名で行い、送信時に同じディレクトリ内にファイルが無い場合はエラーとなります。
受信の場合は新たにファイルが作成されます。</td></tr>
<tr><td>パラメータ</td><td>n=1～5
“ファイル名”</td></tr>
</table>

命令	END
意味	END まで読み込んだスクリプトが実行されます。 END 以降に記述された内容については、何も行われません。(スクリプトの読み込みを END で中断)
パラメータ	なし

◆ I2C 専用の文法

表 3-3 I2C 命令表

命令	ADDRESSMODE=
意味	I2C アドレスを 7 ビットモードか 10 ビットモードに設定します。(初期値は 7 ビットモード)
パラメータ	7 または 10
命令	ADDRESS=
意味	I2C アドレスを指定します。 アドレスはいつでも変更することが可能ですが、アドレス指定する前に「READ」や「WRITE」がある場合、文法エラーとなります。
パラメータ	0～1023
命令	READ
意味	指定されたバイト数分読み出しを行います。
パラメータ	xxH 読み出しバイト数を指定 1～65536 まで
命令	READF
意味	指定されたバイト数分読み出しを行い、ファイルへの保存を行います。 データは FILEn で指定されたファイル名で保存されます。 既存のファイルが指定された場合は追加書き込みされます。
パラメータ	xxH FILEn 読み出しバイト数、保存ファイルを指定（バイト指定は 1～65536 まで）
命令	WRITE
意味	指定されたデータを書き込みます。書き込みデータが複数ある場合はカンマで区切ります。
パラメータ	xxH, xxH, … 書き込みデータをバイト単位で指定
命令	WRITEF
意味	ファイルからデータを送信します。データはバイナリデータとして扱われます。 書込みデータは FILEn で指定されたファイルから読み込まれます。
パラメータ	FILEn 書き込みデータをファイルで指定
命令	STOP
意味	ストップビットを送信します。
パラメータ	なし

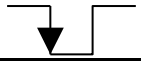
<table>
<tr><th>命令</th><td>RESET</td></tr>
<tr><td>意味</td><td>バスにリセットを発生させます(STOP ビット送信)</td></tr>
<tr><td>パラメータ</td><td>なし</td></tr>
<tr><td colspan="2"></td></tr>
<tr><th>命令</th><td>GPO=</td></tr>
<tr><td>意味</td><td>DO0～DO3(13～16 番ピン)へのポート出力を指定します。</td></tr>
<tr><td>パラメータ</td><td>出力するビットに 1 を指定します。

10 進で表記する場合は 0～15 を指定
16 進で表記する場合は 0h～Fh を指定

<table>
<tr><th>Bit3</th><th>Bit2</th><th>Bit1</th><th>Bit0</th></tr>
<tr><td>DO3</td><td>DO2</td><td>DO1</td><td>DO0</td></tr>
</table>
</td></tr>
</table>

◆ SPI 専用の文法

表 3-4 SPI 命令表

<table>
<tr><td>命令</td><td>SS=n</td></tr>
<tr><td>意味</td><td>スレーブセレクトピンの設定を行います。初期値は 0 となります。
<table><tr><td>パラメータ</td><td>SSx</td></tr><tr><td>0</td><td>SS0</td></tr><tr><td>1</td><td>SS1</td></tr><tr><td>2</td><td>SS2</td></tr><tr><td>3</td><td>SS3</td></tr></table></td></tr>
<tr><td>パラメータ</td><td>n=0～3</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td>命令</td><td>SAMPLING=n</td></tr>
<tr><td>意味</td><td>バスサンプリング方法を指定します。初期値は 0 となります。
<table><tr><td>パラメータ</td><td>サンプリングエッジ</td><td>図</td></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></td></tr>
<tr><td>パラメータ</td><td>n=0～3</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td>命令</td><td>FB=</td></tr>
<tr><td>意味</td><td>ファーストビットを指定します。初期値は MSB となります。</td></tr>
<tr><td>パラメータ</td><td>MSB または LSB</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td>命令</td><td>SSSET</td></tr>
<tr><td>意味</td><td>SS 命令で指定されたスレーブセレクト信号を Low にします。</td></tr>
<tr><td>パラメータ</td><td>なし</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td>命令</td><td>SSRESET</td></tr>
<tr><td>意味</td><td>SS 命令で指定されたスレーブセレクト信号を High にします。</td></tr>
<tr><td>パラメータ</td><td>なし</td></tr>
</table>

その他の機能	
SPI では Read/Write に特別なコマンドを用意せず、記述された値を書き込みます。 SPI はその性質上、書き込みと同時にデータ読み出しを行うため、読み出しのみを行うことはできません。	
意味	読み出したデータは FILEn で指定したファイルへ保存されます。(FILEn を指定した場合) 既存のファイルが指定された場合は追加で書き込みされます。
パラメータ	xxH, xxH, … FILEn 書き込むデータをバイト単位で指定し、 読み出しデータをファイルへ保存する場合。
意味	FILEm で指定したファイルからデータを書き込みます。 読み出したデータは FILEn で指定したファイルへ保存されます。(FILEn を指定した場合) 既存のファイルが指定された場合は追加で書き込みされます。
パラメータ	FILEm FILEn 書き込むデータをファイル指定し、 読み出しデータをファイルへ保存する場合。

◆ REPEAT コマンドの使用方法について

ここでは REPEAT スクリプトと{ }内処理と STOP に関する動作の補足説明をいたします。

スクリプトコード	動作説明
REPEAT=10 READ 1 STOP	10 バイトのデータを受信後、STOP コンディションを送信します。

REPEAT=10
{
READ 1 STOP
}
	「1 バイト毎に STOP コンディションを送信する」
	を 10 回繰り返します。

REPEAT=10 READ 1 STOP	10 バイトのデータを受信後、STOP コンディションを送信します。

REPEAT=10
{
READ 1
}
STOP
	10 バイトのデータを受信後、STOP コンディションを送信します。

REPEAT=10
{
READ 1
STOP
}
	「1 バイトのデータを受信後、STOP コンディションを送信する」
	を 10 回繰り返します。

(3-5)　スクリプト例

スクリプトファイルの使用方法について説明いたします。

「オプション」-「リストビュー/スクリプト切替」よりスクリプト記述表示に変更してください。

各ボタンの説明は次のようになります。
「Load」 --- スクリプトファイルを読み込みます。
「Save」--- 記述した内容をファイルに保存します。
「Clear」--- 表示された内容を消去します。
「Execute」--- スクリプトを実行します。
「Stop」--- 処理中のスクリプト実行を停止します。

実行結果は「Transfer Log」に表示されます。

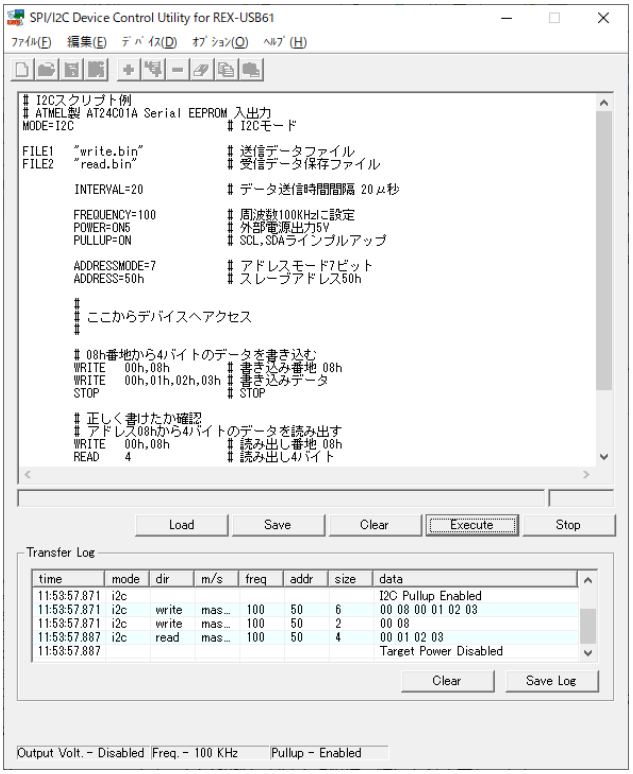


図 3-16.　スクリプト使用例

※　スクリプトファイルはテキストファイルで、テキストエディタ(メモ帳等)で作成・編集することができます。

以下に I2C および SPI デバイスを制御するスクリプト例(Write / Read)について記述します。
(スクリプトファイルはダウンロードした[Utility]フォルダー内に収録されています。
I2C_script.txt/ SPI_script.txt)

・I2C スクリプト例:(スレーブアドレス 50h のデバイスの 0008h 番地から 4 バイトのデータ(00h 01h 02h 03h)を書き込み、正しく書けているかを読み出す。また、ファイル内のデータを書き込み、正しくかけているかをファイルへ読み出す。周波数は 100KHz/外部電源出力は 5V とする。)

```
# I2C スクリプト例
# ATMEL 製 AT24C01A Serial EEPROM 入出力
MODE=I2C                        # I2C モード

        FILE1   "write.bin"     # 送信データファイル
        FILE2   "read.bin"      # 受信データ保存ファイル

        INTERVAL=20             # データ送信時間間隔 20μ秒
        FREQUENCY=100           # 周波数 100KHz に設定
        POWER=ON5               # 外部電源出力 5V
        PULLUP=ON               # SCL,SDA ラインプルアップ
        ADDRESSMODE=7           # アドレスモード 7 ビット
        ADDRESS=50h             # スレーブアドレス 50h

        #
        # ここからデバイスへアクセス
        #

        # アドレス 0008h から 4 バイトのデータを書き込む
        WRITE  00h,08h          # 書き込み番地 0008h
        WRITE  00h,01h,02h,03h  # 書き込みデータ
        STOP                    # STOP

        # 正しく書けたか確認
        # アドレス 0008h から 4 バイトのデータを読み出す
        WRITE  00h,08h          # 読み出し番地 0008h
        READ   4                # 読み出し 4 バイト
        STOP                    # STOP
```

#(次ページへ続く)

```
#(前ページからの続き)

        # アドレス 0008h から FILE1 のデータを書き込む
        WRITE  00h,08h          # 書き込み番地  0008h
        WRITEF         FILE1    # 書き込みデータ(FILE1 には 4 バイトのバイナリデータ)
        STOP                    # STOP

        # 正しく書けたか確認
        # アドレス 0008h から読み出したデータを FILE2 へコピーする
        WRITE  00h,08h          # 読み出し番地  0008h
        READF 04h      FILE2    # 4 バイトの読み出しデータを FILE2 へコピー
        STOP                    # STOP

        POWER=OFF               # 外部電源出力 0V

END
```

・SPI スクリプト例：(1500h 番地から 4 バイトのデータ(00h 01h 02h 03h)を書き込み、正しく書けているかを読み出す。また、ファイル内のデータを書き込み、正しくかけているかをファイルへ読み出す。周波数は 3MHz/外部電源出力は 5V とする。)

```
# SPI スクリプト例
# ATMEL 製 AT25080 Serial EEPROM 入出力
MODE=SPI                # SPI モード

        FILE1   "write.bin"             # 送信データファイル
        FILE2   "read.bin"              # 受信データ保存ファイル

        POWER=ON5                       # 外部電源出力 5V
        INTERVAL=20                     # データ送信時間間隔 20μ秒
        FREQUENCY=3000                  # 周波数 3MHz に設定
        SAMPLING=0                      # データ更新エッジを指定
        FB=MSB                          # ビット列の順序を指定する
        SS=0                            # スレーブセレクトピン 0 を選択

        #
        # ここからデバイスへアクセス
        #

        # 1500h 番地から 4 バイトのデータを書き込む
        SSSET                           # SS 信号を Low
        06h                             # オペコード WREN
        SSRESET                         # SS 信号を High
        SSSET                           # SS 信号を Low
        02h,15h,00h                     # オペコード WRITE＋書き込み番地
        00h,01h,02h,03h                 # 書き込みデータ
        SSRESET                         # SS 信号を High
```

(次ページへ続く)

```
# (前ページからの続き)
        # 正しくかけたか確認
        # 1500h 番地から 4 バイトのデータを読み出す
        SSSET                           # SS 信号を Low
        03h,15h,00h                     # オペコード READ+読み出し番地
        REPEAT=4                        # 次の命令を 4 回繰り返し
        00h                             # ダミーライトで読み出し 1 バイト
        SSRESET                         # SS 信号を High

        # 1500h 番地から FILE1 のデータを書き込む
        SSSET                           # SS 信号を Low
        06h                             # オペコード WREN
        SSRESET                         # SS 信号を High
        SSSET                           # SS 信号を Low
        02h,15h,00h                     # オペコード WRITE＋書き込み番地
        FILE1                           # FILE1 のデータをライト
        SSRESET                         # SS 信号を High

        # 正しくかけたか確認
        # 1500h 番地から読み出したデータを FILE2 へコピーする
        SSSET                           # SS 信号を Low
        03h,15h,00h                     # オペコード READ+読み出し番地
        FILE1   FILE2                   # FILE1 からダミーライト
                                        # FILE2 へ読み出したデータの保存
        SSRESET                         # SS 信号を High

        POWER=OFF                       # 外部電源出力 0V

    END
```

第4章 API関数仕様

(4-1) VC での使用について

本 API 関数は、REX-USB61 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（usb61def.h)、ライブラリファイル（usb61api.lib/64bit 版は LIB¥VC_x64 フォルダー内） を用意しています。
(usb61api.dll はドライバーインストール時にシステム内にコピーされます)
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ます。
ライブラリ関数のインポート宣言は以下の通りです（usb61def.h より抜粋)。
※ ユーザ定義型の記述については、ヘッダファイル usb61def.h を参照してください。

```
#define USB61LIB_API __declspec(dllimport)

USB61LIB_API HANDLE WINAPI usb61_open( RS_STATUS *pStatus );
USB61LIB_API RS_STATUS WINAPI usb61_close( HANDLE hUsb61Device );
USB61LIB_API RS_STATUS WINAPI usb61_power_control( HANDLE hUsb61Device,
                                                   UINT fPowerState );
USB61LIB_API RS_STATUS WINAPI usb61_mode_change( HANDLE hUsb61Device,
                                                 UINT fDeviceMode,
                                                 USHORT i2cSlaveAddr );
USB61LIB_API RS_STATUS WINAPI usb61_set_interval( HANDLE hUsb61Device,
                                                  USHORT IntervalCnt );
USB61LIB_API RS_STATUS WINAPI usb61_gpo_write( HANDLE hUsb61Device,
                                               UINT fPortVal );
USB61LIB_API RS_STATUS WINAPI usb61_get_fw_version( HANDLE hUsb61Device,
                                                    UCHAR* pFWMajorVer,
                                                    UCHAR* pFWMinorVer );
USB61LIB_API RS_STATUS WINAPI usb61_get_dll_version( HANDLE hUsb61Device,
                                                     UCHAR* pDllMajorVer,
                                                     UCHAR* pDllMinorVer );
USB61LIB_API RS_STATUS WINAPI usb61_get_hw_info( HANDLE hUsb61Device,
                                                 PRS_HARDWARE_INFO pHardwareInfo );
USB61LIB_API RS_STATUS WINAPI usb61_i2c_pullup( HANDLE hUsb61Device,
                                                RS_I2C_PULLUP fI2cPullup );
USB61LIB_API RS_STATUS WINAPI usb61_i2c_bus_reset( HANDLE hUsb61Device );
```

(次ページへ続く)

```
USB61LIB_API RS_STATUS WINAPI usb61_i2c_set_freq( HANDLE hUsb61Device,
                                                  RS_I2C_FREQ fI2cFreq );
USB61LIB_API RS_STATUS WINAPI usb61_i2c_set_freq_ex( HANDLE hUsb61Device,
                                                     USHORT Frequency,
                                                     USHORT *pActualFrequency );
USB61LIB_API RS_STATUS WINAPI usb61_i2c_read_master( HANDLE hUsb61Device,
                                                     USHORT SlaveAddress,
                                                     UINT fI2cOption,
                                                     USHORT ReadBytes,
                                                     UCHAR *pReadBuf );
USB61MLIB_API RS_STATUS WINAPI usb61_i2c_read_master_ex(HANDLE hUsb61Device,
                                                     USHORT SlaveAddress,
                                                     UINT fI2cOption,
                                                     USHORT ReadBytes,
                                                     UCHAR *pReadBuf );
USB61LIB_API RS_STATUS WINAPI usb61_i2c_write_master( HANDLE hUsb61Device,
                                                      USHORT SlaveAddress,
                                                      UINT fI2cOption,
                                                      USHORT WriteBytes,
                                                      UCHAR *pWriteBuf );
USB61LIB_API RS_STATUS WINAPI usb61_i2c_read_slave( HANDLE hUsb61Device,
                                                    RS_NOTIFY_TYPE nType,
  void (CALLBACK EXPORT* lpfnReadEvent)(USHORT ReadBytes, UCHAR *pReadBuf),
                                                    HWND hWnd );
USB61LIB_API RS_STATUS WINAPI usb61_i2c_set_response_data( HANDLE hUsb61Device,
                                                           USHORT ResponseBytes,
                                                           UCHAR *pResponseBuf );
USB61LIB_API RS_STATUS WINAPI usb61_spi_set_freq( HANDLE hUsb61Device,
                                                  UINT fDataMode,
                                                  USHORT Frequency,
                                                  USHORT *pActualFrequency);
USB61LIB_API RS_STATUS WINAPI usb61_spi_transmit_master( HANDLE hUsb61Device,
                                                         RS_SPI_SS fSlaveSelect,
                                                         USHORT TransmitSize,
                                                         UCHAR *pSendBuf,
                                                         UCHAR *pRecvBuf );
USB61LIB_API RS_STATUS WINAPI usb61_spi_transmit_master_hold_ss
                                                 ( HANDLE hUsb61Device,
                                                   RS_SPI_SS fSlaveSelect,
                                                   USHORT TransmitSize,
                                                   UCHAR *pSendBuf,
                                                   UCHAR *pRecvBuf );
```

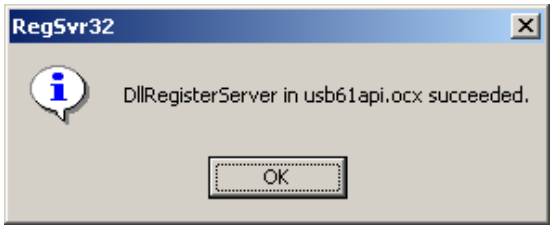
(4-2) VB / Visual C#での使用について

Visual BASIC および Visual C#のアプリケーションから 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りなす。
64bit 版アプリケーションを作成される場合は LIB 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。(ダウンロード提供)

(1) ActiveX の登録

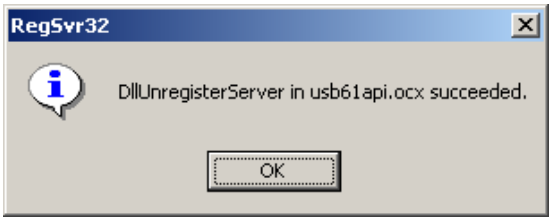
第 2 章 Windows セットアップを参照しドライバのインストールを行ってください。
自動的に DLL,ActiveX のコピーが行われます。
usb61api.ocx の登録は、管理者権限でコマンドプロンプトを起動し
>regsvr32 usb61api.ocx
と実行します。



登録成功メッセージ

(2) ActiveX の削除

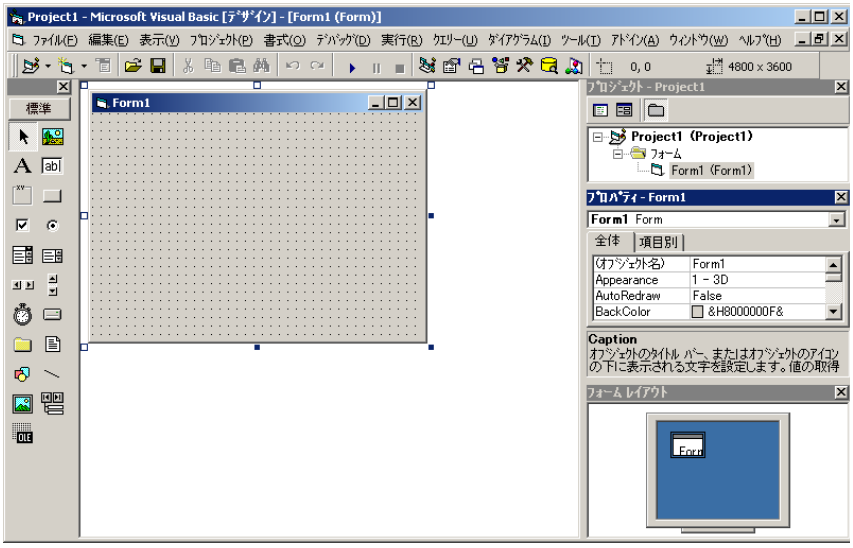
登録から削除する際にはコマンドプロンプトから
>regsvr32 /u usb61api.ocx
と実行します。



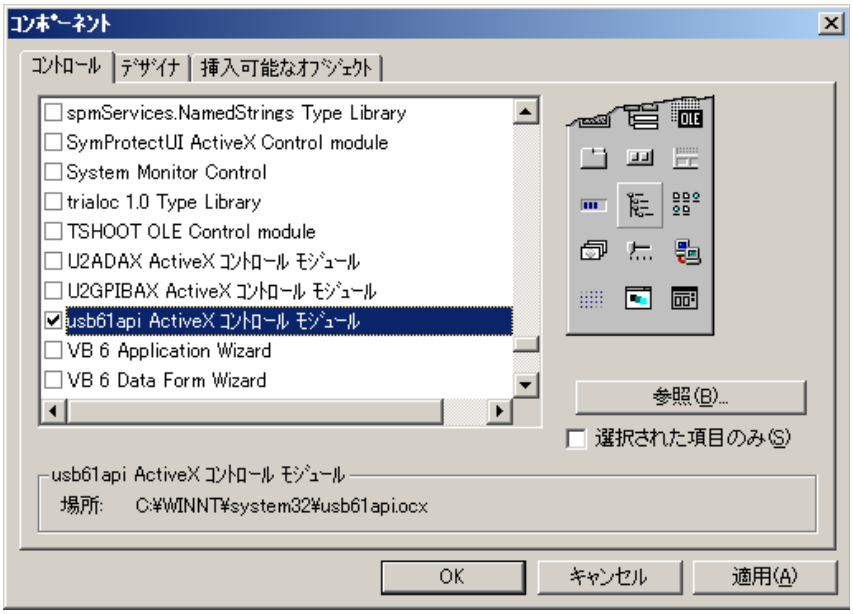
登録削除成功メッセージ

(3) VB6 での ActiveX 参照方法

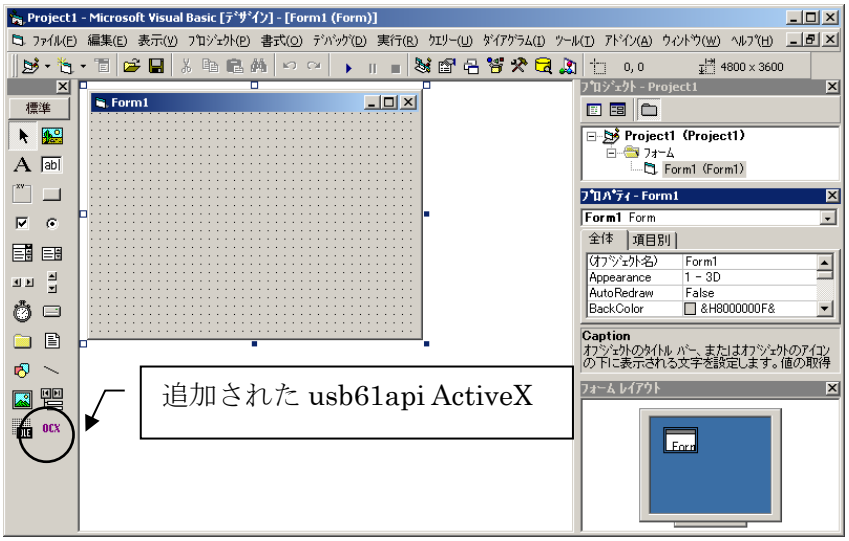
新しいプロジェクトを作成します。



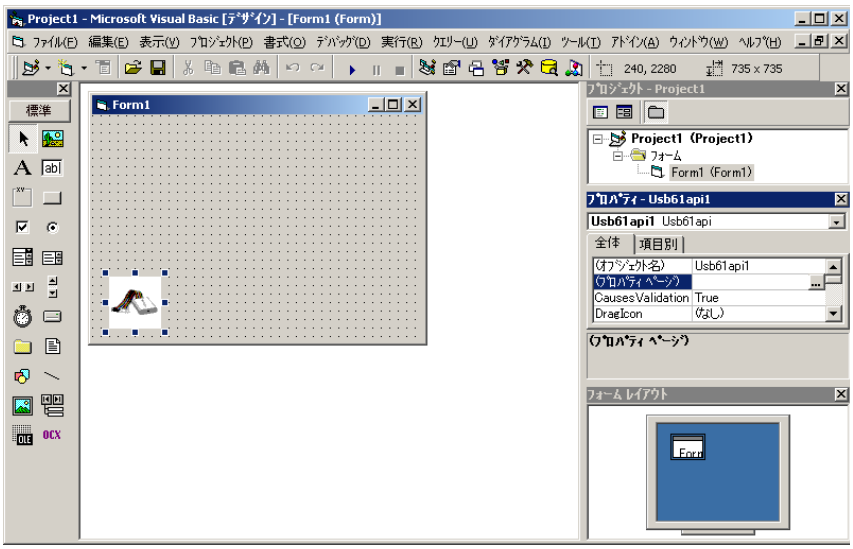
プロジェクトメニューのコンポーネントを選択します。コントロール一覧の、「usb61api ActiveX ｺﾝﾄﾛｰﾙﾓｼﾞｭｰﾙ」にチェックを入れて OK ボタンをクリックします。



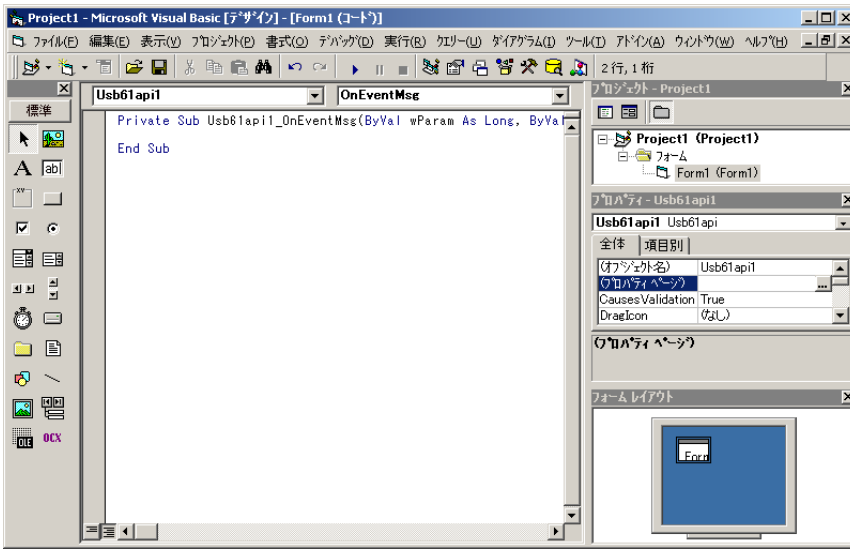
usb61api ActiveX コンポーネントが追加されます。



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。
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。

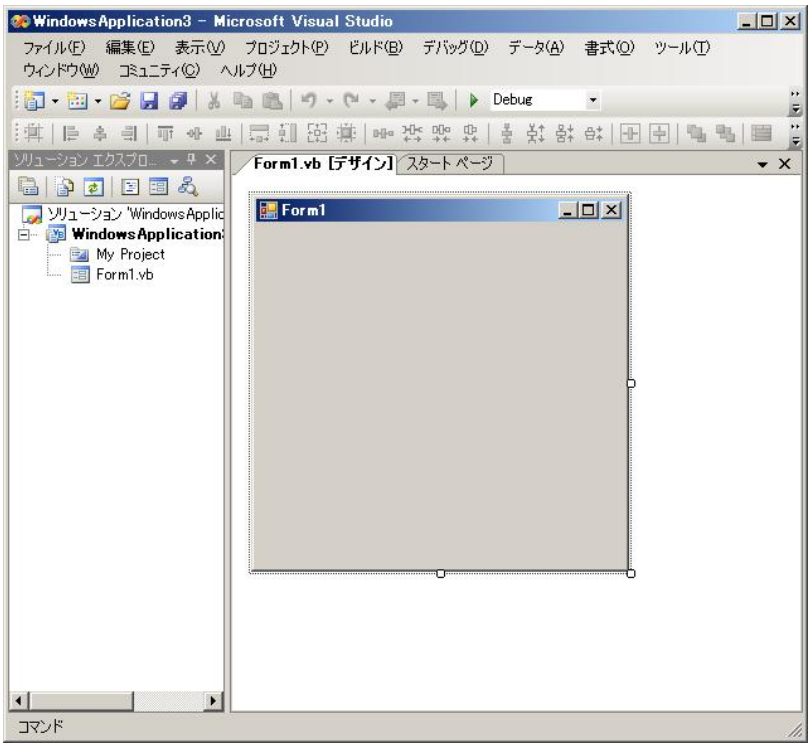


オブジェクトをダブルクリックすると、イベント発生時の呼び出されるサブルーチン
Sub Usb61api1_OnEventMsg (…)が表示されます。関数仕様の説明を参照願います。

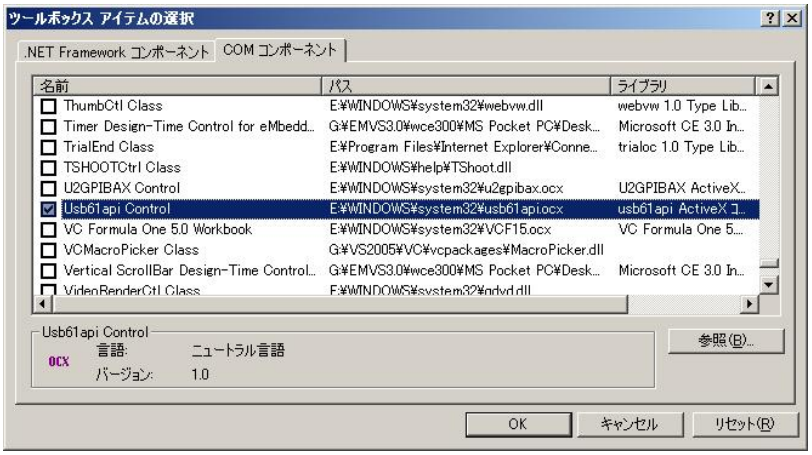


(4) VB.NET / Visual C#での ActiveX 参照方法

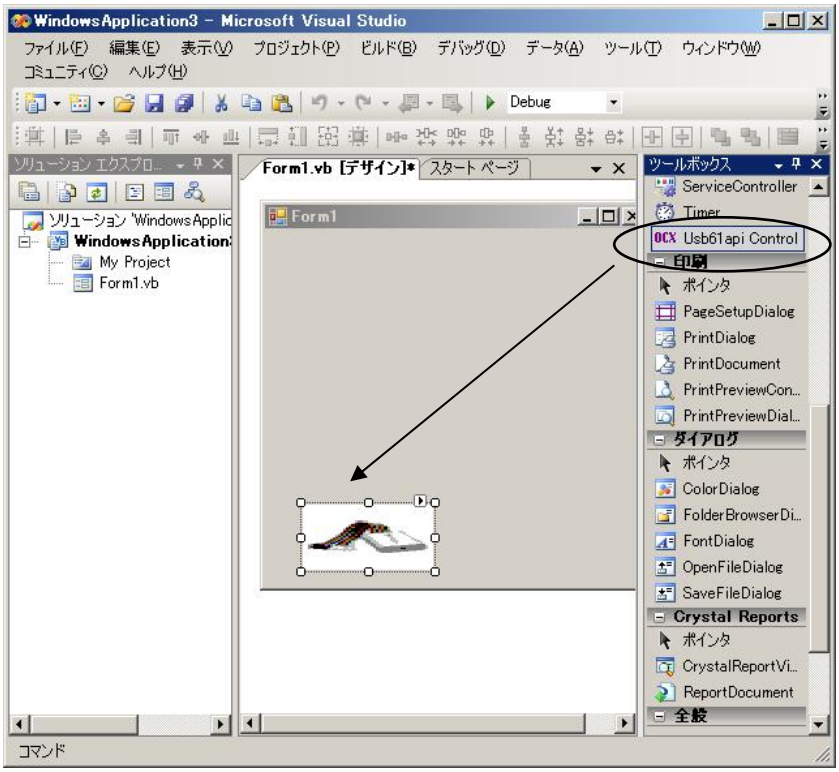
新しいプロジェクトを作成します。



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、「Usb61apiControl」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。



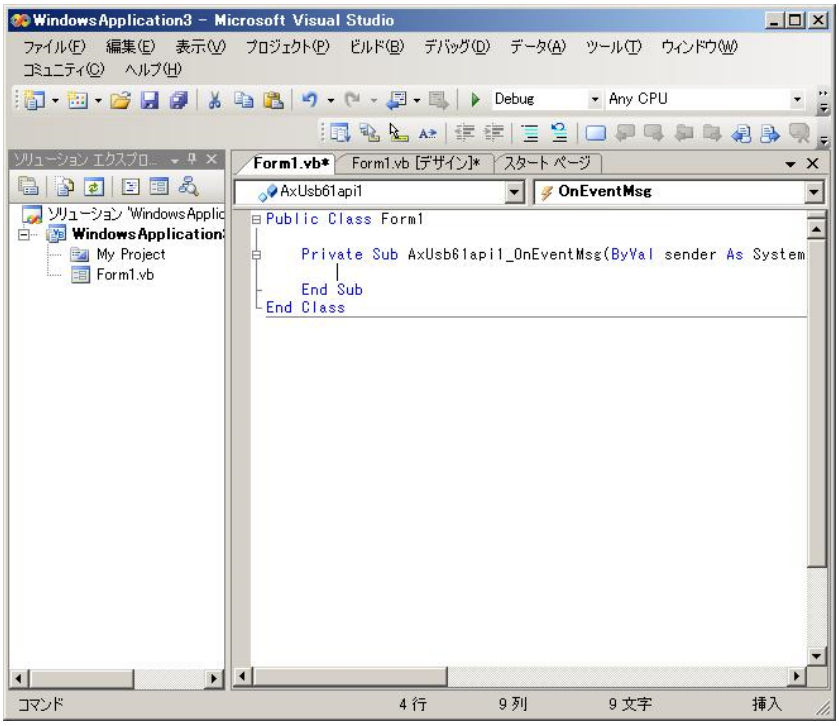
貼り付けたオブジェクトのプロパティ内の「Visible」を False にして、実行時表示されないようにしておきます。

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 AxUsb61api1_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 関数の一覧を示します。

表 4-1 API 関数一覧

関数名	機能
usb61_open()	デバイスの使用を開始する
usb61_close()	デバイスの使用を終了する
usb61_power_control()	ターゲットデバイスへの電源供給
usb61_get_fw_version()	ファームウェアのバージョン取得
usb61_get_dll_version()	DLL のバージョン取得
usb61_get_hw_info()	ハードウェアの情報取得
usb61_mode_change()	SPI/I2C，マスター/スレーブモード切替
usb61_set_interval()	送信データ 1 バイト毎の時間間隔を設定
usb61_gpo_write()	Port ピンへのデータ出力。
usb61_i2c_pullup()	I2C のプルアップ設定（SDA，SCL）
usb61_i2c_bus_reset()	I2C バスリセット。
usb61_i2c_set_freq()	I2C インターフェイス周波数の設定
usb61_i2c_set_freq_ex()	I2C インターフェイス周波数の設定 (1KHz 単位で設定)
usb61_i2c_read_master()	I2C バス経由のリード（マスター側）
usb61_i2c_read_master_ex()	I2C バス経由のリード（マスター側） ※サブアドレスも指定
usb61_i2c_write_master()	I2C バス経由のライト（マスター側）
usb61_i2c_read_slave()	I2C バス経由のリード（スレーブ側）
usb61_i2c_set_response_data()	I2C マスターへ転送するデータをセット （スレーブ側）
usb61_spi_set_freq()	SPI インターフェイス周波数の設定 (1KHz 単位で設定)
usb61_spi_transmit_master()	SPI バス経由のデータ転送 （マスター側）※SS ラインを High へ戻す。
usb61_spi_transmit_master_hold_ss()	SPI バス経由のデータ転送 （マスター側）※SS ラインを High へ戻さない。

(4-4) API 関数詳細

以下に API 関数の詳細を示します。
(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の呼び出し方法および関数定義は、VB6 サンプル EEPROMRWUtyVB/C#サンプル EEPROMRWUtyCS をご参照ください。)

共通関数

関数	VC ➢	HANDLE usb61_open(RS_STATUS *pStatus);
	VB ➢	Function Usb61Open (pStatus As Long) As Long
	VB.NET➢	Function Usb61Open (ByRef pStatus As Integer) As Integer
	C#➢	int Usb61Open(ref int pStatus)
機能	デバイスオープン処理。デバイスの使用を開始する。	
引数	[OUT] pStatus： 成功時は RS_SUCCESS を、失敗時はエラーコードを受け取る。	
戻値	成功時はデバイスハンドルを、失敗時は INVALID_HANDLE_VALUE を返す。	

関数	VC ➢	RS_STATUS usb61_close(HANDLE hUsb61Device);
	VB ➢	Function Usb61Close (ByVal hUsb61Device As Long) As Long
	VB.NET➢	Function Usb61Close (ByVal hUsb61Device As Integer) As Integer
	C#➢	int Usb61Close(int hUsb61Device)
機能	デバイスクローズ処理。デバイスの使用を終了する。	
引数	[IN] hUsb61Device ： デバイスハンドルをセット。	
戻値	成功時は RS_SUCCESS を、失敗時はエラーコードを返す。(表 4-3 参照)	

関数	VC ➢	RS_STATUS usb61_power_control(HANDLE hUsb61Device, UINT fPowerState);
	VB ➢	Function Usb61PowerControl (ByVal hUsb61Device As Long, ByVal fPowerState As Long) As Long
	VB.NET ➢	Function Usb61PowerControl(ByVal hUsb61Device As Integer, ByVal fPowerStateAs Integer) As Integer
	C#➢	int Usb61PowerControl(int hUsb61Device, int fPowerState)
機能	ターゲットデバイスへの電源供給を行う。	
引数	[IN] hUsb61Device : デバイスハンドルをセット。 [IN] fPowerState : 電源供給およびターゲットの電源電圧を指定。 RS_PWRCTRL_OFF 電源供給を行わない。 RS_PWRCTRL_ON | RS_OUTPUT_3_3V ターゲットの電源電圧 3.3V。 RS_PWRCTRL_ON | RS_OUTPUT_5_0V ターゲットの電源電圧 5.0V。 (3.3V および 5.0V を設定する場合は、RS_PWRCTRL_ON と RS_OUTPUT_3_3V、RS_OUTPUT_5_0V をビット演算子の OR(|)を使用して設定します。)	
戻値	成功時は RS_SUCCESS を、失敗時はエラーコードを返す。(表 4-3 参照)	

関数	VC ➢	RS_STATUS usb61_get_fw_version(HANDLE hUsb61Device, UCHAR *pFwMajorVer, UCHAR *pFwMinorVer);
	VB ➢	Function Usb61GetFwVersion(ByVal hUsb61Device As Long, pFWMajorVer As Byte, pFWMinorVer As Byte) As Long
	VB.NET ➢	Function Usb61GetFwVersion(ByVal hUsb61Device As Integer, ByRef pFWMajorVer As Byte, ByRef pFWMinorVer As Byte) As Integer
	C#➢	int Usb61GetFwVersion(int hUsb61Device, ref byte pFWMajorVer, ref byte pFWMinorVer)
機能	ファームウェアのバージョンを取得する。	
引数	[IN] hUsb61Device : デバイスハンドルをセット。 [OUT] *pFwMajorVer : ファームウェアのメジャーバージョンが格納される。(16 進値) [OUT] *pFwMinorVer : ファームウェアのマイナーバージョンが格納される。(16 進値)	
戻値	成功時は RS_SUCCESS を、失敗時はエラーコードを返す。(表 4-3 参照)	

関数	VC ➢	RS_STATUS usb61_get_dll_version(HANDLE hUsb61Device, UCHAR *pDllMajorVer, UCHAR *pDllMinorVer);
	VB ➢	Function Usb61GetDllVersion(ByVal hUsb61Device As Long, pDllMajorVer As Byte, pDllMinorVer As Byte) As Long
	VB.NET ➢	Function Usb61GetDllVersion(ByVal hUsb61Device As Integer, ByRef pDllMajorVer As Byte, ByRef pDllMinorVer As Byte) As Integer
	C#➢	int Usb61GetDllVersion(int hUsb61Device, ref byte pDllMajorVer, ref byte pDllMinorVer)
機能	DLL のバージョンを取得する。	
引数	[IN] hUsb61Device ： デバイスハンドルをセット。 [OUT] *pDllMajorVer ： DLL のメジャーバージョンが格納される。 (16 進値) [OUT] *pDllMinorVer ： DLL のマイナーバージョンが格納される。 (16 進値)	
戻値	成功時は RS_SUCCESS を、失敗時はエラーコードを返す。(表 4-3 参照)	

関数	VC ➢	RS_STATUS usb61_get_hw_info(HANDLE hUsb61Device, PRS_HARDWARE_INFO pHardwareInfo);
	VB ➢	Function Usb61GetHwInfo(ByVal hUsb61Device As Long, pHardwareInfo As Byte) As Long
	VB.NET ➢	Function Usb61GetHwInfo(ByVal hUsb61Device As Integer, ByRef pHardwareInfo As Object) As Integer
	C#➢	int Usb61GetHwInfo(int hUsb61Device, ref object pHardwareInfo)
機能	ハードウェアの設定を取得する。	

引数	[IN] hUsb61Device ：デバイスハンドルをセット。
	[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 モード
	UCHAR MasterSlaveAct; // マスター／スレーブ動作
	USHORT Frequency; // インターフェイスの周波数
	UCHAR OutputVolt; // ターゲットデバイスへの出力電圧
	} RS_HARDWARE_INFO, *PRS_HARDWARE_INFO;
	
	RS_HARDWARE_INFO 構造体は usb61def.h で定義されています。
	
	VB での使用例
	Dim pHardWareBuf() As Byte
	Dim HardWareInfo As RS_HARDWARE_INFO
	
	ReDim pHardWareBuf(10) As Byte
	rsStatus = Usb61api.Usb61GetHwInfo(m_hDeviceHandle,
	pHardWareBuf)
	If rsStatus <> RS_SUCCESS Then
	’エラー処理
	Else
	HardWareInfo.DeviceMode = pHardWareBuf(0)
	HardWareInfo.MasterSlaveAct = pHardWareBuf(1)
	HardWareInfo.Frequency =
	pHardWareBuf(3)*256 + pHardWareBuf(2)
	HardWareInfo.OutputVolt = pHardWareBuf(4)
	End If	

戻値	成功時は RS_SUCCESS を、失敗時はエラーコードを返す。(表 4-3 参照)	

関数	VC ➢	RS_STATUS usb61_mode_change(HANDLE hUsb61Device, UINT fDeviceMode, USHORT i2cSlaveAddr);
	VB ➢	Function Usb61ModeChange(ByVal hUsb61Device As Long, ByVal fDeviceMode As Long, ByVal i2cSlaveAddr As Integer) As Long
	VB.NET ➢	Function Usb61ModeChange(ByVal hUsb61Device As Integer, ByVal fDeviceMode As Integer, ByVal i2cSlaveAddr As Short) As Integer
	C#➢	int Usb61ModeChange(int hUsb61Device, int fDeviceMode, short i2cSlaveAddr)
機能	SPI／I2C モード，マスター／スレーブ動作の切り替えを行う。	
引数	[IN] hUsb61Device ： デバイスハンドルをセット。 [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。 RS_DEVMODE_MASTER マスター動作。 RS_DEVMODE_SLAVE スレーブ動作。 ※ 本引数にはモードと動作をビット演算子の OR(|)を使用して設定します。 例：RS_DEVMODE_SPI | RS_DEVMODE_MASTER (SPI マスター) [IN] i2cSlaveAddr ： I2C スレーブモードの場合は I2C ターゲットデバイスのアドレスを指定する。	
戻値	成功時は RS_SUCCESS を、失敗時はエラーコードを返す。(表 4-3 参照)	

関数	VC ➢	RS_STATUS usb61_set_interval(HANDLE hUsb61Device, USHORT IntervalCnt);
	VB ➢	Function Usb61SetInterval(ByVal hUsb61Device As Long, ByVal IntervalCnt As Long) As Long
	VB.NET➢	Function Usb61SetInterval(ByVal hUsb61Device As Integer, ByVal IntervalCnt As Integer) As Integer
	C#➢	int Usb61SetInterval(int hUsb61Device, int intervalCnt)
機能	SPI/I2C のデータ送信時の 1 バイト毎の送信間隔を設定する。(1μS 単位) ※ 本設定は最低限設定された間隔を保証するもので、実際の時間は処理時間を含み長くなります。(本関数を呼び出さない場合は 0μS) **※ 本関数は usb61_mode_change()の後に呼び出してください。**	
引数	[IN] hUsb61Device : デバイスハンドルをセット。 [IN] IntervalCnt : データ送信間隔を指定する。 (μS 単位 0～65535μS)	
戻値	成功時は RS_SUCCESS を，失敗時はエラーコードを返す。(表 4-3 参照)	

関数	VC ➢	RS_STATUS usb61_i2c_pullup(HANDLE hUsb61Device, RS_I2C_PULLUP fI2cPullup);
	VB ➢	Function Usb61I2cPullup(ByVal hUsb61Device As Long, ByVal fI2cPullup As Integer) As Long
	VB.NET➢	Function Usb61I2cPullup(ByVal hUsb61Device As Integer, ByVal fI2cPullup As Short) As Integer
	C#➢	int Usb61I2cPullup(int hUsb61Device, short fI2cPullup)
機能	I2C バスのプルアップ設定を行う。(SDA, SCL の各ピン)	
引数	[IN] hUsb61Device : デバイスハンドルをセット。 [IN] fI2cPullup : プルアップ設定を指定する。 ※ I2C、SPI 時は常に ENABLE を設定します。(1MHz I2C のみ選択可) 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	成功時は RS_SUCCESS を，失敗時はエラーコードを返す。(表 4-3 参照)	

GPO（I2C 時のみ）

関数	VC ➢	RS_STATUS usb61_gpo_write(HANDLE hUsb61Device, UINT fPortVal);
	VB ➢	Function Usb61GpoWrite(ByVal hUsb61Device As Long, ByVal fPortVal As Long) As Long
	VB.NET➢	Function Usb61GpoWrite(ByVal hUsb61Device As Integer, ByVal fPortVal As Integer) As Integer
	C#➢	int Usb61GpoWrite(int hUsb61Device, int fPortVal)
機能	GPO ピンへデータを出力する。	
引数	[IN] hUsb61Device ： デバイスハンドルをセット。 [IN] fPortVal ： GPO ピンに書き込む値をフラグで指定。 RS_GPO_NONE すべての PORT に Low(=0)をセットする。 RS_GPO_PORT0 PORT0 に High(=1)を出力する。 RS_GPO_PORT1 PORT1 に High(=1)を出力する。 RS_GPO_PORT2 PORT2 に High(=1)を出力する。 RS_GPO_PORT3 PORT3 に High(=1)を出力する。 ※ 複数ポートを同時に設定するには、ビット演算子の OR(|)を使用します。 例：RS_GPO_PORT0 | RS_GPO_PORT1 (PORT1 と PORT2 へ出力)	
戻値	成功時は RS_SUCCESS を、失敗時はエラーコードを返す。(表 4-3 参照)	

I2C 共通

関数	VC ➢	RS_STATUS usb61_i2c_bus_reset(HANDLE hUsb61Device);
	VB ➢	Function Usb61I2cBusReset(ByVal hUsb61Device As Long) As Long
	VB.NET➢	Function Usb61I2cBusReset(ByVal hUsb61Device As Integer) As Integer
	C#➢	int Usb61I2cBusReset(int hUsb61Device)
機能	I2C バスリセットを行う。 (I2C バスが使用中であった場合にストップコンディションを発行する。)	
引数	[IN] hUsb61Device : デバイスハンドルをセット。	
戻値	成功時は RS_SUCCESS を、失敗時はエラーコードを返す。(表 4-3 参照)	

関数	VC ➢	RS_STATUS usb61_i2c_set_freq(HANDLE hUsb61Device, RS_I2C_FREQ fI2cFreq);
	VB ➢	Function Usb61I2cSetFreq(ByVal hUsb61Device As Long, ByVal fI2cFreq As Integer) As Long
	VB.NET➢	Function Usb61I2cSetFreq(ByVal hUsb61Device As Integer, ByVal fI2cFreq As Short) As Integer
	C#➢	int Usb61I2cSetFreq(int hUsb61Device, short fI2cFreq)
機能	I2C インターフェイスの周波数を設定する。	
引数	[IN] hUsb61Device : デバイスハンドルをセット。 [IN] fI2cFreq : I2C バスの周波数を指定する。 RS_I2C_FREQ_1M 1MHz RS_I2C_FREQ_400K 400KHz RS_I2C_FREQ_100K 100KHz	
戻値	成功時は RS_SUCCESS を、失敗時はエラーコードを返す。(表 4-3 参照)	

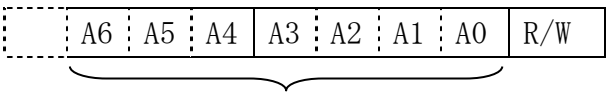
関数	VC ➢	RS_STATUS usb61_i2c_set_freq_ex(HANDLE hUsb61Device, USHORT Frequency USHORT *pActualFrequency);
	VB ➢	Function Usb61I2cSetFreqEx(ByVal hUsb61Device As Long, ByVal Frequency As Long, pActualFrequency As Long) As Long
	VB.NET➢	Function Usb61I2cSetFreqEx(ByVal hUsb61Device As Integer, ByVal Frequency As Integer, ByRef pActualFrequency As Integer) As Integer
	C#➢	int Usb61I2cSetFreqEx(int hUsb61Device, int frequency, ref int pActualFrequency)
機能	I2C バスの動作周波数を 1KHz 単位で指定する。 設定可能な値は 47～1000(KHz) Frequency に設定した値が、実際に設定された値として pActualFrequency に返される。	
引数	[IN] hUsb61Device : デバイスハンドルをセット。 [IN] Frequency : I2C バスの動作周波数を指定する。 [OUT] pActualFrequency: 実際に設定された I2C バス動作周波数が返る。	
戻値	成功時は RS_SUCCESS を、失敗時はエラーコードを返す。(表 4-3 参照)	

I2C マスター

関数	VC ➢	RS_STATUS usb61_i2c_read_master(HANDLE hUsb61Device, USHORT SlaveAddress, UINT fI2cOption, USHORT ReadBytes, UCHAR *pReadBuf);
	VB ➢	Function Usb61I2cReadMaster(ByVal hUsb61Device As Long, ByVal SlaveAddress As Integer, ByVal fI2cOption As Long, ByVal ReadBytes As Integer, pReadBuf As Byte) As Long
	VB.NET➢	Function Usb61I2cReadMaster(ByVal hUsb61Device As Integer, ByVal slaveAddress As Short, ByVal fI2cOption As Integer, ByVal readBytes As Short, ByRef pReadBuf As Object) As Integer
	C#➢	int Usb61I2cReadMaster(int hUsb61Device, short slaveAddress, int fI2cOption, short readBytes, ref object pReadBuf)
機能	I2C バス経由でデータを読み出す。(マスター側)	
引数	[IN] hUsb61Device : デバイスハンドルをセット。 [IN] SlaveAddress : ターゲットデバイスのアドレスを指定する。 ※ 下記参照 [IN] fI2cOption : I2C バス通信時の特殊設定。(表 4-2 参照) [IN] ReadBytes : 読み出すデータのサイズを指定する。 [OUT] pReadBuf : 読み出したデータを格納するバッファへのポインタ。	
戻値	成功時は RS_SUCCESS を、失敗時はエラーコードを返す。(表 4-3 参照)	

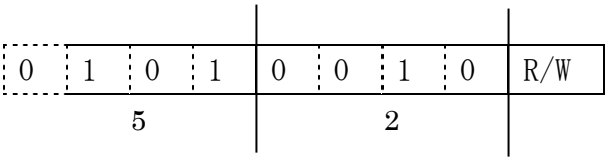
※ スレーブアドレスについて

7bit、10bit で指定してください。R/W ビットは含みません。



スレーブアドレス

[例：52h の場合]



関数	VC ➢	RS_STATUS usb61_i2c_read_master_ex(HANDLE hUsb61Device, USHORT SlaveAddress, USHORT SubAddress, UINT fI2cOption, USHORT ReadBytes, UCHAR *pReadBuf);
	VB ➢	Function Usb61I2cReadMasterEx(ByVal hUsb61Device As Long, ByVal SlaveAddress As Integer, ByVal SubAddress As Integer, ByVal fI2cOption As Long, ByVal ReadBytes As Integer, pReadBuf As Byte) As Long
	VB.NET➢	Function Usb61I2cReadMasterEx(ByVal hUsb61Device As Integer, ByVal slaveAddress As Short, ByVal subAddress As Short, ByVal fI2cOption As Integer, ByVal readBytes As Short, ByRef pReadBuf As Object) As Integer
	C#➢	int Usb61I2cReadMasterEx(int hUsb61Device, short slaveAddress, ushort subAddress, int fI2cOption, short readBytes, ref object pReadBuf)
機能	I2C バス経由でデータを読み出す。(マスター側) usb61_i2c_read_master()とは、リードを行う前に関数内部で呼び出し位置を指定するライト(サブアドレスの指定)を行う点が異なる。	
引数	[IN] hUsb61Device : デバイスハンドルをセット。 [IN] SlaveAddress : ターゲットデバイスのアドレスを指定する。 ※ Page.4-18 下記「スレーブアドレスについて」を参照。 [IN] SubAddress : サブアドレスを指定する。 (2 バイトアドレスまで対応) [IN] fI2cOption : I2C バス通信時の特殊設定。(表 4-2 参照) [IN] ReadBytes : 読み出すデータのサイズを指定する。 [OUT] pReadBuf : 読み出したデータを格納するバッファへのポインタ。	
戻値	成功時は RS_SUCCESS を、失敗時はエラーコードを返す。(表 4-3 参照)	

関数	VC ➢	RS_STATUS usb61_i2c_write_master(HANDLE hUsb61Device, USHORT SlaveAddress, UINT fI2cOption, USHORT WriteBytes, UCHAR *pWriteBuf);
	VB ➢	Function Usb61I2cWriteMaster(ByVal hUsb61Device As Long, ByVal SlaveAddress As Integer, ByVal fI2cOption As Long, ByVal WriteBytes As Integer, ByVal pWriteBuf As Byte) As Long
	VB.NET➢	Function Usb61I2cWriteMaster(ByVal hUsb61Device As Integer, ByVal slaveAddress As Short, ByVal fI2cOption As Integer, ByVal writeBytes As Short, ByVal pWriteBuf As Object) As Integer
	C#➢	int Usb61I2cWriteMaster(int hUsb61Device, short slaveAddress, int fI2cOption, short writeBytes, object pWriteBuf)
機能	I2C バス経由でデータを書き込む。(マスター側)	
引数	[IN] hUsb61Device : デバイスハンドルをセット。 [IN] SlaveAddress : ターゲットデバイスのアドレスを指定する。 ※参照 Page.4-14「スレーブアドレスについて」 [IN] fI2cOption : I2C バス通信時の特殊設定。(表 4-2 参照) [IN] WriteBytes : 書き込むデータのサイズを指定する。 [IN] pWriteBuf : 書き込むデータが格納されたバッファへのポインタ。	
戻値	成功時は RS_SUCCESS を、失敗時はエラーコードを返す。(表 4-3 参照)	

表 4-2 I2C バス通信フラグ

I2C バス通信フラグ	値	内容
RS_I2C_FLAG_NONE	0x00	フラグなし。
RS_I2C_FLAG_10BIT_ADDR	0x01	10 ビットデバイスのアドレスを指定する場合にセットする。
RS_I2C_FLAG_STOP	0x02	ストップコンディションを発行する場合にセットする。
RS_I2C_FLAG_1BYTE_SA	0x04	リード前に1バイトサブアドレスを送信する。
RS_I2C_FLAG_2BYTE_SA	0x0C	リード前に2バイトサブアドレスを送信する。

I2C スレーブ

関数	VC ➢	RS_STATUS usb61_i2c_read_slave(HANDLE hUsb61Device, RS_NOTIFY_TYPE nType, void (CALLBACK EXPORT* lpfnReadEvent) (USHORT ReadBytes, UCHAR *pReadBuf), HWND hWnd);
	VB ➢	Function Usb61I2cReadSlave(ByVal hUsb61Device As Long, ByVal nType As Integer) As Long
	VB.NET➢	Function Usb61I2cReadSlave(ByVal hUsb61Device As Integer, ByVal nType As Short) As Integer
	C#➢	int Usb61I2cReadSlave(int hUsb61Device, short nType)
機能	I2C バス経由でデータを読み出す。(スレーブ側) 関数実行後はバックグラウンドでマスターからデータを受け取るまで待機します。 本関数呼び出し前に関数 usb61_i2c_set_response_data()をコールし、マスターへ転送するデータをあらかじめセットしておく必要があります。 VC では受信処理が完了した際に、アプリケーション側のコールバック関数またはユーザー定義メッセージに通知されます。 VB/VB.net/C#では ActiveX コントロールを介しイベントとして通知されます。	
引数	[IN] hUsb61Device : デバイスハンドルをセット。 [IN] nType : ステート通知方法。 RS_NOTIFY_CALLBACK コールバック関数により通知。(VC のみ) RS_NOTIFY_USER_MSG ユーザー定義メッセージにより通知。 [IN] lpfnReadEvent : アプリケーション側に通知されるコールバック関数。 引数に上位アプリケーションが用意した lpfnReadEvent コールバック関数を渡す。 lpfnReadEvent のコールバック関数の名前は ReadIsComplete である必要はないが, 次のように定義しなければならない。 void CALLBACK EXPORT ReadIsComplete(USHORT ReadBytes, UCHAR *pReadBuf); [IN] hWnd : ユーザー定義メッセージを通知するウィンドウのハンドルを指定。 ユーザー定義メッセージによる通知を行わない場合は NULL を指定。	
戻値	成功時は RS_SUCCESS を、失敗時はエラーコードを返す。(表 4-3 参照)	

<table>
<tr><td rowspan="4">関数</td><td>VC ➢</td><td>VC では使用しません</td></tr>
<tr><td>VB ➢</td><td>Function Usb61GetData(ByVal wParam As Long,
ByVal lParam As Long,
pBuf As Byte) As Long</td></tr>
<tr><td>VB.NET ➢</td><td>Function Usb61GetData(ByVal wParam As Integer,
ByVal lParam As Integer,
ByRef pBuf As Object) As Integer</td></tr>
<tr><td>C#➢</td><td>int Usb61GetData(int wParam, int lParam, ref object pBuf)</td></tr>
<tr><td>機能</td><td colspan="2">Usb61I2cReadSlave 関数を用いて「WM_USB61_MSG」に通知された時にデータを取得する</td></tr>
<tr><td>引数</td><td colspan="2">[IN] wParam : 読み出すデータのサイズを指定する。
[IN] lParam : 読み出したデータが格納されたアドレス
[OUT] pBuf : 読み出したデータを格納するバッファへのポインタ。
使用例
Private Sub Usb61api_OnEventMsg(ByVal wParam As Long, ByVal lParam As Long)
’ステータスコード
Dim rsStatus As Long
Dim pBuf() As Byte

ReDim pBuf(wParam) As Integer

rsStatus = Usb61api.Usb61GetData(wParam, lParam, pBuf)
End Sub</td></tr>
<tr><td>戻値</td><td colspan="2">常に RS_SUCCESS</td></tr>
</table>

関数	VC ➢	RS_STATUS usb61_i2c_set_response_data(HANDLE hUsb61Device, USHORT ResponseBytes, UCHAR *pResponseBuf);
	VB ➢	Function Usb61I2cSetResponseData(ByVal hUsb61Device As Long, ByVal ResponseBytes As Integer, ByVal pResponseBuf As Byte) As Long
	VB.NET➢	Function Usb61I2cSetResponseData(ByVal hUsb61Device As Integer, ByVal responseBytes As Short, ByVal pResponseBuf As Object) As Integer
	C#➢	int Usb61I2cSetResponseData(int hUsb61Device, short responseBytes, object pResponseBuf)
機能	I2C スレーブとしてマスターへ転送するデータをセットする。 マスターからデータが送られてきたら、あらかじめセットしておいたデータを転送する 。	
引数	[IN] hUsb61Device : デバイスハンドルをセット。 [IN] ResponseBytes : マスターへ転送するデータのサイズをセットする。 [IN] pResponseBuf : マスターへ転送するデータが格納されるバッファへのポインタ。	
戻値	成功時は RS_SUCCESS を、失敗時はエラーコードを返す。(表 4-3 参照)	

SPI マスター

関数	VC ➢	RS_STATUS usb61_spi_set_freq(HANDLE hUsb61Device, UINT fDataMode, USHORT Frequency, USHORT *pActualFrequency);
	VB ➢	Function Usb61SpiSetFreq(ByVal hUsb61Device As Long, ByVal fDataMode As Long, ByVal Frequency As Long, pActualFrequency As Long) As Long
	VB.NET➢	Function Usb61SpiSetFreq(ByVal hUsb61Device As Integer, ByVal fDataMode As Integer, ByVal frequency As Integer, ByRef pActualFrequency As Integer) As Integer
	C#➢	int Usb61SpiSetFreq(int hUsb61Device, int fDataMode, int frequency, ref int pActualFrequency)
機能	SPI バスの動作周波数を 1KHz 単位で指定する。 設定可能な値は 1～12000(KHz) 設定された Frequency から設定可能な近似値が計算され、実際に設定された値として pActualFrequency に返される。 ※ 近似値(pActualFrequency)の計算方法 6024 ÷ Frequency の整数部分を X とする。 (X ≧ 1020 の場合) X ÷ 16 の整数部分を Y とする。 6024 ÷ (Y × 16)の整数部分が pActualFrequency に返される。 (256 ≦ X < 1020 の場合) X ÷ 4 の整数部分を Y とする。 6024 ÷ (Y × 4)の整数部分が pActualFrequency に返される。 (X < 256 の場合) 6024 ÷ X の整数部分が pActualFrequency に返される。 ただし、次の入力周波数は特別設定値を持っているため、Frequency と同じ値が pActualFrequency に返される。 1, 750, 3000, 12000(KHz) また、Frequency が 3013(KHz)以上の場合は pActualFrequency に 12000(KHz)が返される。	

引数	[IN] hUsb61Device : デバイスハンドルをセット。 [IN] fDataMode : SPI 転送時のデータモードをフラグで指定。 RS_SPI_PHASE_SETUP_SAMPLE 立ち下がりエッジでサンプリング。 RS_SPI_PHASE_SAMPLE_SETUP 立ち上がりエッジでサンプリング。 RS_SPI_POLARITY_POSITIVE クロックの反転なし。 RS_SPI_POLARITY_NEGATIVE クロックを反転する。 RS_SPI_MSB_FIRST 上位ビット先行。 RS_SPI_LSB_FIRST 下位ビット先行。 ※ エッジ・クロックの反転・先行ビットの組み合わせを、ビット演算子の OR(|)を使用して設定します。 例：RS_SPI_PHASE_SETUP_SAMPLE | RS_SPI_POLARITY_POSITIVE | RS_SPI_MSB_FIRST [IN] Frequency : SPI バスの動作周波数を設定する。 [OUT] pActualFrequency : 実際に設定された SPI バス周波数が返される。
戻値	成功時は RS_SUCCESS を、失敗時はエラーコードを返す。(表 4-3 参照)

関数	VC ➢	RS_STATUS usb61_spi_transmit_master(HANDLE hUsb61Device, RS_SPI_SS fSlaveSelect, USHORT TransmitSize, UCHAR *pSendBuf, UCHAR *pRecvBuf);
	VB ➢	Function Usb61SpiTransmitMaster(ByVal hUsb61Device As Long, ByVal fSlaveSelect As Integer, ByVal TransmitSize As Integer, ByVal pSendBuf As Byte, pRecvBuf As Byte) As Long
	VB.NET➢	Function Usb61SpiTransmitMaster(ByVal hUsb61Device As Integer, ByVal fSlaveSelect As Short, ByVal transmitSize As Short, ByVal pSendBuf As Object, ByRef pRecvBuf As Object) As Integer
	C#➢	int Usb61SpiTransmitMaster(int hUsb61Device, short fSlaveSelect, short transmitSize, object pSendBuf, ref object pRecvBuf)
機能	SPI バス経由でデータ転送を行う。(マスター側) データ転送時、ダミーデータを受け取る。 データ転送後、SS ラインを High へ戻します。	
引数	[IN] hUsb61Device : デバイスハンドルをセット。 [IN] fSlaveSelect : スレーブセレクトピン番号を指定する。 RS_SPI_SS0 スレーブセレクトピン 0 を選択。 RS_SPI_SS1 スレーブセレクトピン 1 を選択。 RS_SPI_SS2 スレーブセレクトピン 2 を選択。 RS_SPI_SS3 スレーブセレクトピン 3 を選択。 [IN] TransmitSize : データ転送長。 [IN] pSendBuf : 送信データを格納するバッファへのポインタ。 [OUT] pRecvBuf : 受信データを格納するバッファへのポインタ。	
戻値	成功時は RS_SUCCESS を、失敗時はエラーコードを返す。(表 4-3 参照)	

関数	VC ➢	RS_STATUS usb61_spi_transmit_master_hold_ss (HANDLE hUsb61Device, RS_SPI_SS fSlaveSelect, USHORT TransmitSize, UCHAR *pSendBuf, UCHAR *pRecvBuf);
	VB ➢	Function Usb61SpiTransmitMasterHoldSS(ByVal hUsb61Device As Long, ByVal fSlaveSelect As Integer, ByVal TransmitSize As Integer, ByVal pSendBuf As Byte, pRecvBuf As Byte) As Long
	VB.NET➢	Function Usb61SpiTransmitMasterHoldSS(ByVal hUsb61Device As Integer, ByVal fSlaveSelect As Short, ByVal transmitSize As Short, ByVal pSendBuf As Object, ByRef pRecvBuf As Object) As Integer
	C#➢	int Usb61SpiTransmitMasterHoldSS(int hUsb61Device, short fSlaveSelect, short transmitSize, object pSendBuf, ref object pRecvBuf)
機能	SPI バス経由でデータ転送を行う。(マスター側) データ転送時、ダミーデータを受け取る。 データ転送後、SS ラインを High へ戻しません。 SS ラインを High へ戻すには関数 usb61_gpo_write を使用します。	
引数	[IN] hUsb61Device : デバイスハンドルをセット。 [IN] fSlaveSelect : スレーブセレクトピン番号を指定する。 RS_SPI_SS0 スレーブセレクトピン 0 を選択。 RS_SPI_SS1 スレーブセレクトピン 1 を選択。 RS_SPI_SS2 スレーブセレクトピン 2 を選択。 RS_SPI_SS3 スレーブセレクトピン 3 を選択。 [IN] TransmitSize : データ転送長。 [IN] pSendBuf : 送信データを格納するバッファへのポインタ。 [OUT] pRecvBuf : 受信データを格納するバッファへのポインタ。	
戻値	成功時は RS_SUCCESS を、失敗時はエラーコードを返す。(表 4-3 参照)	

(4-5) エラーコード一覧

表 4-3 エラーコード一覧

エラーコード	値	内容
RS_SUCCESS	0	成功。
RS_OK	0	関数の呼び出しが正常に終了した。
RS_DEVICE_FOUND	0	デバイスを検出した。
RS_DEVICE_CONNECT	0	デバイスが接続されている。
RS_UNABLE_TO_LOAD_LIBRARY	-1	ライブラリがロードできない。
RS_UNABLE_TO_LOAD_DRIVER	-2	ドライバがロードできない。
RS_UNABLE_TO_LOAD_FUNCTION	-3	関数の呼び出しが行えない。
RS_INCOMPATIBLE_LIBRARY	-4	使用しているライブラリに互換性がない。
RS_INCOMPATIBLE_DEVICE	-5	本デバイスでは動作できない。
RS_COMMUNICATION_ERROR	-6	SPI/I2C 通信エラー。
RS_UNABLE_TO_OPEN	-7	オープン処理を行えない。
RS_UNABLE_TO_CLOSE	-8	クローズ処理を行えない。
RS_INVALID_HANDLE	-9	無効なハンドル。
RS_CONFIG_ERROR	-10	コンフィグエラー。
RS_TIMEOUT	-11	タイムアウト。
RS_OUT_OF_RANGE	-12	範囲外の値が設定された。
RS_DEVICE_NOT_FOUND	-20	デバイスが見つからない。
RS_DEVICE_NOT_CONNECT	-21	デバイスが接続されていない。
RS_DEVICE_OPEN_EXIST	-22	デバイスがすでにオープンされている。
RS_I2C_NOT_AVAILABLE	-100	I2C バスが使用できない。
RS_I2C_NOT_ENABLED	-101	I2C バスが無効である。
RS_I2C_READ_ERROR	-102	I2C バス上のリード処理時にエラー。
RS_I2C_WRITE_ERROR	-103	I2C バス上のライト処理時にエラー。
RS_I2C_BAD_CONFIG	-104	I2C バスの設定が間違っている。
RS_I2C_TIMEOUT	-105	I2C バス上でタイムアウトが起きた。
RS_I2C_DROPPED_EXCESS_BYTES	-106	I2C バス上でデータを取りこぼした。
RS_I2C_BUS_ALREADY_FREE	-107	I2C バスはすでにバスフリー状態。
RS_I2C_WRITE_COLLISION	-108	I2C ライト時にデータの衝突が発生。
RS_I2C_READ_OVERFLOW	-109	I2C リード時にオーバーフローが発生。
RS_I2C_NACK_DETECT	-110	I2C 通信時に NO ACK を検出。
RS_I2C_OUTRANGE	-111	I2C の設定時に範囲外の値が設定された。
RS_SPI_NOT_AVAILBLE	-200	SPI バスが使用できない。
RS_SPI_NOT_ENABLED	-201	SPI バスが無効である。
RS_SPI_WRITE_ERROR	-202	SPI バス上のライト処理時にエラー。
RS_SPI_READ_ERROR	-203	SPI バス上のリード処理時にエラー。
RS_SPI_BAD_ CONFIG	-204	SPI バスの設定が間違っている。
RS_SPI_TIMEOUT	-205	SPI バス上でタイムアウトが起きた。
RS_SPI_DROPPED_EXCESS_BYTES	-206	SPI バス上でデータを取りこぼした。
RS_SPI_WRITE_OVERFLOW	-207	SPI ライト時にオーバーフローが発生。
RS_SPI_OUTRANGE	-208	SPI の設定時に範囲外の値が設定された。
RS_GPO_NOT_AVAILABLE	-300	Port が使用できない。
RS_FAILURE	-400	その他のエラー。

※ 上記以外の正の値が返される場合は Win32 のエラーコードとなります。

(4-6) サンプルアプリケーションについて

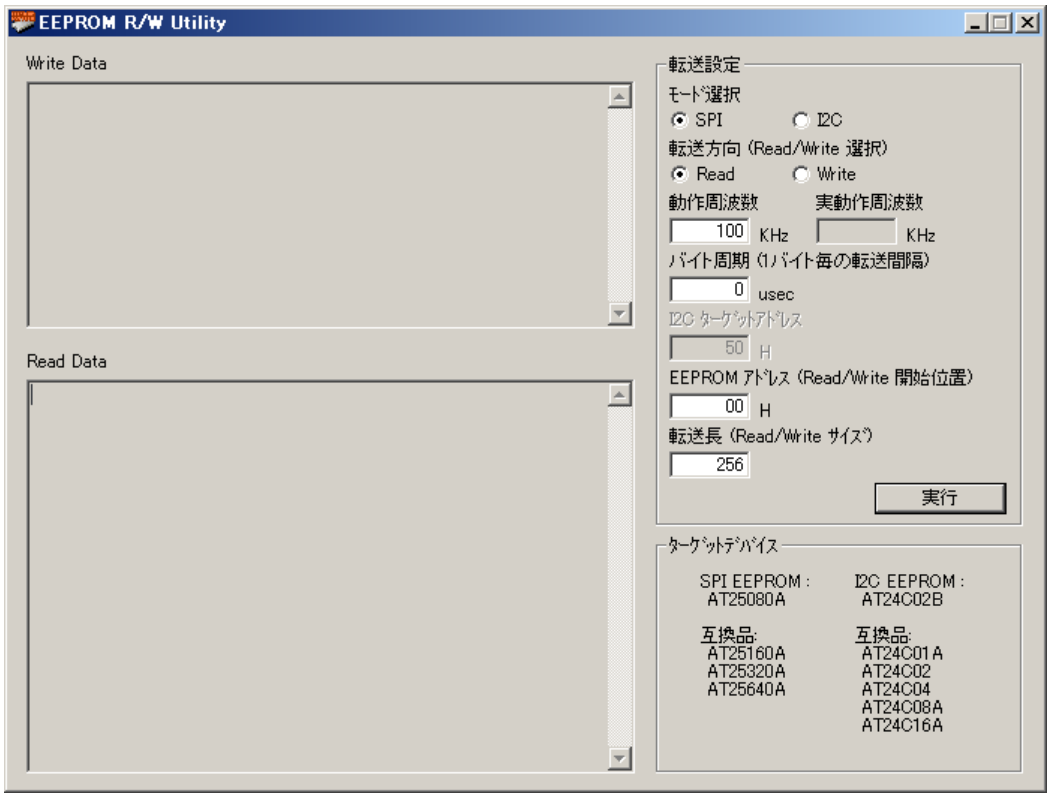
REX-USB61 には、アプリケーション開発を行う際のご参考としてサンプルアプリケーションを提供しております。

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。

I2cSlaveSample サンプルアプリケーション[I2CSlaveSample フォルダー内]は、本製品を I2C スレーブとして動作させることができます。

【EEPROM R/W Utility サンプルアプリケーション説明】

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。
動作周波数・・・設定する周波数の値を入力します。
実動作周波数・・・入力した動作周波数から計算された実際の周波数が返されます。
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。
Read/Write 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。



【サンプルアプリケーション画面】

(プログラミング例につきましては、ソースコードをご参照ください。
また、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。)

【I2cSlaveSample サンプルアプリケーション説明】

I2C スレーブアドレス・・・本製品のスレーブアドレスを設定します。

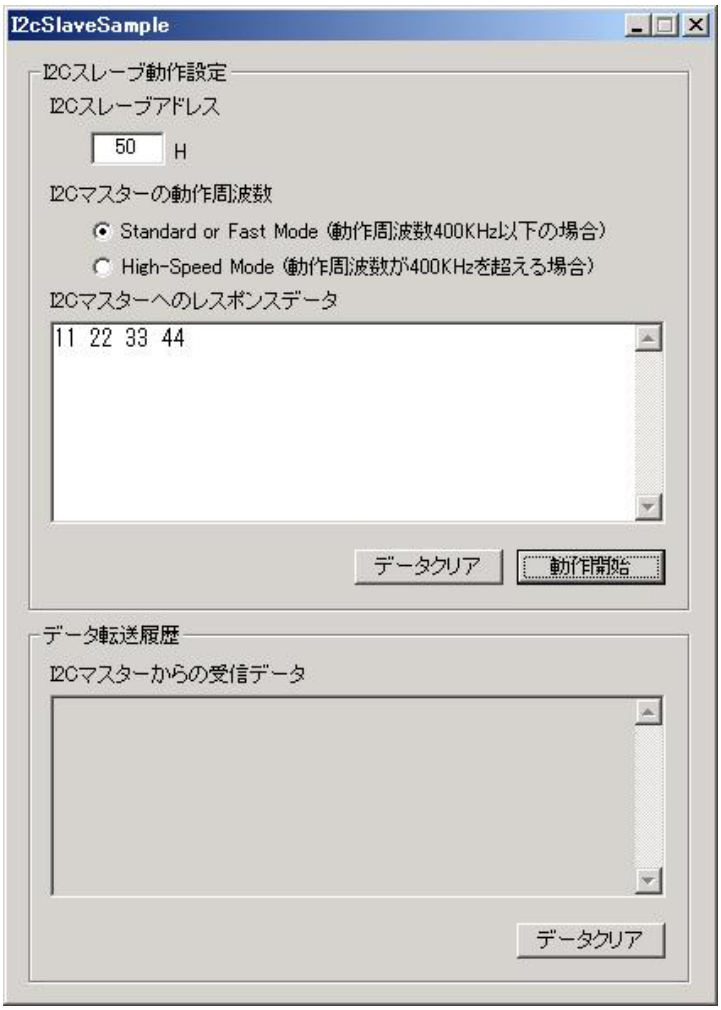
I2C マスターの動作周波数・・・マスター側の動作周波数を選択します。

I2C マスターへのレスポンスデータ・・・設定した I2C スレーブアドレスにリードアクセスが合った場合、設定したデータがマスター側へ送信されます。

動作開始・・・I2C スレーブ動作を開始します。

I2C マスターからの受信データ・・・マスターから送信されたデータが表示されます。

データクリア・・・表示されているデータを消去します。



【サンプルアプリケーション画面】

(プログラミング例につきましては、ソースコードをご参照ください。)

(4-7) API 関数を使用したアプリケーション作成例について

API 関数を使用して制御アプリケーションを作成する場合の例を記述いたします。
(C++での例となりますので、その他の開発環境や詳細な内容につきましては
サンプルプログラムソースをご参照ください。)

例 (EEPROM R/W Utility [I2C]):
SDA より 1 バイトデータ ff(16 進数)を出力する(C++)
(エラー処理等は行っておりません。)

```
HANDLE hDeviceHandle;        // デバイスのハンドル
BYTE DeviceAddr;             // ターゲットデバイスのアドレス
WORD DataLen;                // 転送データ長
USHORT i2cFreq;              // 設定周波数
WORD ActualFreq;             // 実際の周波数
USHORT IntervalCnt;          // バイト周期
BYTE Data;                   // 転送データ

// デバイスハンドルを取得
hDeviceHandle = usb61_open(&rsStatus);

// ターゲットデバイスへ 5.0V を供給
// REX-USB61 から電源を供給せずにターゲットから電源供給する場合は
// RS_PWRCTRL_OFF をセットしてください
usb61_power_control(hDeviceHandle, RS_PWRCTRL_ON | RS_OUTPUT_5_0V);

// I2C マスターモードに変更
usb61_mode_change(hDeviceHandle, RS_DEVMODE_I2C | RS_DEVMODE_MASTER, NULL );

// バイト周期を設定する (モード変更後)
usb61_set_interval(hDeviceHandle, IntervalCnt);

// I2C バスのプルアップを行う
usb61_i2c_pullup( hDeviceHandle, RS_I2C_PULLUP_ENABLE );

// 周波数をセット
usb61_i2c_set_freq_ex( hDeviceHandle, i2cFreq, &ActualFreq );

// SDA より 1 バイトデータ 0xff(16 進数)を出力
// ※DeviceAddr にはターゲットデバイスのアドレスを指定(R/W ビットは含めないでください)
// ここでは 0x00 としています
DeviceAddr = 0x00;    // アドレス 0x00 のデバイスに
Data       = 0xFF;    // 0xFF を
DataLen    = 1;       // 1 バイト送信
usb61_i2c_write_master( hDeviceHandle, DeviceAddr, RS_I2C_FLAG_STOP, DataLen, &Data );

// REX-USB61 の使用を終了
usb61_close(hDeviceHandle);
hDeviceHandle = NULL;
```

例 (EEPROM R/W Utility [SPI]) :

SDO より 1 バイトデータ ff(16 進数)を出力する(C++)
(エラー処理等は行っておりません。)

```
HANDLE hDeviceHandle;       // デバイスのハンドル
WORD DataLen;               // 転送データ長
USHORTspiFreq;              // 設定周波数
WORD ActualFreq;            // 実際の周波数
BYTE pWriteBuf;             // 転送データを格納
BYTE pReadBuf;              // 受信データを格納
UINT uiFlag;                // エッジ・クロックの反転・先行ビットの組み合わせ

// デバイスハンドルを取得
hDeviceHandle = usb61_open(&rsStatus);

// ターゲットデバイスへ 5.0V を供給
// REX-USB61 から電源を供給せずにターゲットから電源供給する場合は
// RS_PWRCTRL_OFF をセットしてください
usb61_power_control(hDeviceHandle, RS_PWRCTRL_ON | RS_OUTPUT_5_0V);

// SPI マスターモードに変更
usb61_mode_change(hDeviceHandle, RS_DEVMODE_SPI | RS_DEVMODE_MASTER, NULL );

// バイト周期を設定する (モード変更後)
usb61_set_interval(hDeviceHandle, IntervalCnt);

// 周波数をセット
uiFlag  =  RS_SPI_PHASE_SAMPLE_SETUP | RS_SPI_POLARITY_POSITIVE
         | RS_SPI_MSB_FIRST;
usb61_spi_set_freq( hDeviceHandle, uiFlag, spiFreq, &ActualFreq );

// Write Enable コマンド(0x06)を送信
pWriteBuf[0] = 0x06;  // WREN コマンド
DataLen = 1           // 転送データ長
usb61_spi_transmit_master( hDeviceHandle, RS_SPI_SS0, DataLen, pWriteBuf, pReadBuf );

// SDO より 1 バイトデータ 0xff(16 進数)を出力
// Write コマンド(0x02) + EEPROM アドレス(0x0000) + 書込みデータ(0xff)を送信 (計 4 バイト)
pWriteBuf[0] = 0x02   // Write コマンド
pWriteBuf[1] = 0x00   // EEPROM アドレス(上位バイト)
pWriteBuf[2] = 0x00   // EEPROM アドレス(下位バイト)
pWriteBuf[3] = 0xff   // 書込みデータ
DataLen = 4           // 転送データ長
usb61_spi_transmit_master( hDeviceHandle, RS_SPI_SS0, DataLen, pWriteBuf, pReadBuf );
// Read を行う場合は、pReadBuf の 4 バイト目以降がデータとなります。

// REX-USB61 の使用を終了
usb61_close(hDeviceHandle);
hDeviceHandle = NULL;
```

例 (I2cSlaveSample [I2C]) :

スレーブとして動作させ、マスター側からリード命令が行われると、あらかじめ準備したレスポンスデータがマスターへ送信されます。
また、マスター側からの送信データは受信イベント(またはコールバック関数)によって通知され、アプリケーション上へ表示します。 (C++)
(エラー処理等は行っておりません。)

```
HANDLE hDeviceHandle;          // デバイスのハンドル
char        csSlaveAddr[16];   // スレーブアドレス格納用
ULONG       ulSlaveAddr;       // スレーブアドレス
char        *stopstring;
USHORT      usFreq;            // 設定する周波数
USHORT      usActualFreq;      // 実際の周波数
BYTE ResponseBuf[255];         // レスポンスデータ
WORD ResponseBytes;            // レスポンスデータ転送長

// デバイスハンドルを取得
hDeviceHandle = usb61_open(&rsStatus);

// マスター側から電源供給
// REX-USB61 からは電源供給しない
usb61_power_control(hDeviceHandle, RS_PWRCTRL_OFF);

// スレーブアドレス
GetDlgItemText(hwnd, IDE_SLAVE_ADDRESS, csSlaveAddr, sizeof(csSlaveAddr));
ulSlaveAddr = strtoul(csSlaveAddr, &stopstring, 16);

// I2C スレーブモードに変更
usb61_mode_change(hDeviceHandle, RS_DEVMODE_I2C | RS_DEVMODE_SLAVE,
                  (USHORT)ulSlaveAddr);

// プルアップを有効にする
usb61_i2c_pullup(hDeviceHandle, (RS_I2C_PULLUP)fPullup );

// 周波数をセット
usb61_i2c_set_freq_ex(hDeviceHandle, usFreq, &usActualFreq );

// レスポンスデータのセット
// usb61_i2c_set_response_data(hDeviceHandle, ResponseBytes, ResponseBuf );

// I2C スレーブとしてデータ受信待機
// ハンドルの格納
g_hwnd = hwnd;
// 受信イベントの通知にメッセージを使用する場合
usb61_i2c_read_slave(hDeviceHandle, RS_NOTIFY_USER_MSG, NULL, g_hwnd );

・
・

(受信イベントが通知されるとメッセージを受取り、マスター側から送信されたデータの表示を行う)
```

製品に関するお問い合わせ

REX-USB61 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-USB61 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/