M3069 マイコンボード
シリアルインタフェース・ライブラリ
関数リファレンス

テクノウェーブ株式会社

目次

1. はじめに

□ 「M3069 マイコンボード シリアルインタフェース・ライブラリ」について

「M3069 マイコンボード シリアルインタフェース・ライブラリ」は、パソコン用のアプリケーションプログラムから、M3069 シリーズのマイコンボード(『USBM3069(F)』、『LANM3069』)を介して、SPI™、Microwire™、I²C™などのシリアルインタフェースを持つ装置を制御するためのライブラリです。

本ライブラリは、M3069 マイコンボード独自のアタッチメントファームという機構を利用しています。シリアル信号の制御はボード上のマイコンで直接行いますので、パソコンからポート操作で制御するよりも高速に通信することができます。

本ライブラリによる機能拡張は、「USBM ライブラリ」による I/O 機能に追加して行われますので、シリアルインタフェース用として使用しない端子は、そのまま入出力ポートとして利用可能です。また、アナログ入出力、タイマ、外部バスなどの機能も通常どおりご利用いただけます。

アタッチメントファームとは

M3069 シリーズのマイコンボードでは、マイコン用の小さなプログラムを、パソコンからボードの RAM 上にダウンロードして、付属のファームウェア(システムファーム)に新たな機能を追加することができます。このマイコン用プログラムのことをアタッチメントファームと呼んでいます(図 1)。

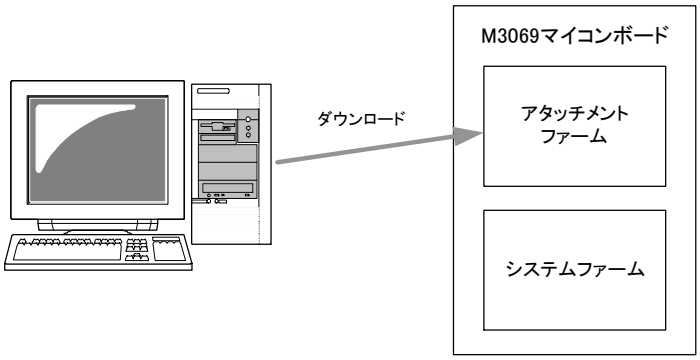

図 1 アタッチメントファームのダウンロード

アタッチメントファームは、マイコン側で実行されますので、通信のための遅延時間が無く、リアルタイム性の高い処理が可能となります。また、普段はパソコン上にファイルとして保存され、必要に応じてダウンロードされるため、機能変更やメンテナンスが容易です。

注意点として、通常、アタッチメントファームは、ユーザーメモリの一部にダウンロードされるため、メモリの一部が使用できなくなります。アプリケーションの中で、ユーザーメモリを利用する場合には、アタッチメントファームのダウンロード領域と重複しないようにする必要があります。

アタッチメントファームについての詳細は別紙「M3069 マイコンボード ユーザーファーム開発マニュアル」を参照してください。

SPI は Motorola Inc. の商標です。
Microwire は National Semiconductor Corp. の商標です。
I²C-bus は NXP B.V. の商標です。

□ 本リファレンスの対応ファイルとバージョンについて

本リファレンスは下記のバージョンのファイル内容を元に記載されています。過去のバージョンや、将来のバージョンについては記載内容と異なる場合がございますのでご注意ください。

表 1　ライブラリの対応バージョン

ファイル名	バージョン番号	対応 OS
M3069Serial.dll	1.0.x.x	Windows 98SE 以降

表 2　ATF ファイルの対応バージョン

ファイル名	バージョン番号	対応システムファーム
M3069Serial.atf	1.0.x	3.0.1 以降

□ 本マニュアル内の表記について

本マニュアル内では対応製品(『USBM3069(F)』、『LANM3069』)を「デバイス」と表記する場合があります。インタフェースを明記する場合には「USB デバイス」、「LAN デバイス」などのように表記します。

本リファレンス内でハードウェアの電気的状態について記述する必要がある場合には、下記のように表記します。

表 3　電気的状態の表記方法

表記	状態
“ON”	電流が流れている状態、スイッチが閉じている状態、オープンコレクタ(オープンドレイン)出力がシンク出力している状態。
“OFF”	電流が流れていない状態、スイッチが開いている状態、オープンコレクタ(オープンドレイン)出力がハイインピーダンスの状態。
“Hi”	電圧がロジックレベルのハイレベルに相当する状態。
“Lo”	電圧がロジックレベルのローレベルに相当する状態。
“Z”	端子がハイインピーダンスの状態。

また、数値について「0x」、「&H」 、「H’」はいずれもそれに続く数値が 16 進数であることを表します。“0x10”、“&H1F”、“H’20”などはいずれも 16 進数です。同様に「B’」に続く数値は 2 進数であることを表します。例えば“B’01000001”のように表記されます。数値の最初に特別な表記が無い場合は 10 進数です。

2. 使用準備

□ 実行時に必要なファイル

本ライブラリを使用するには、以下のファイルが必要になります。パソコンのシステムフォルダ(C:¥Windows¥System32 など)、または、ライブラリを利用する「.exe」ファイルと同じフォルダにコピーしてください。

表 4　実行時に必要なファイル

ファイル名	バージョン	説明
USBM3069.dll	3.3.0.1 以降	製品付属のライブラリ(USBM ライブラリ)。通常、ドライバやツールのインストール時にシステムフォルダにコピーされます。
M3069Serial.atf[1]	1.0.2 以降	マイコンにダウンロードするアタッチメントファーム
M3069Serial.dll	1.0.0.1 以降	ライブラリ本体

1 使用するにはボードのシステムファーム 3.0.1 以降が必要です。

□ 開発時に必要なファイル

以下のファイルはアプリケーションプログラムの開発時に必要になるファイルです。適宜、プロジェクトに追加して使用してください。

表 5　開発に必要なファイル

開発環境	ファイル名	使用方法
Visual C++ など	USBM3069.lib	リンクされるようにプロジェクトに追加してください。
	M3069Serial.lib	リンクされるようにプロジェクトに追加してください。
	USBM3069.h	ソースファイルでインクルードします。
	M3069Serial.h	ソースファイルでインクルードします。
Visual Basic 6.0, VBA など	USBM3069.bas	標準モジュールとして追加します。
	M3069Serial.bas	標準モジュールとして追加します。
Visual Basic .NET (2005)	USBM3069.vb	プロジェクトに追加します。
	M3069Serial.vb	プロジェクトに追加します。

□ ATF ファイルの情報

図 2 は使用する ATF ファイルの情報です。本ライブラリ使用中は、ユーザーメモリの「プログラムの先頭アドレス」から、「プログラムの最終アドレス」の範囲が使用できなくなりますので、ご注意ください

図 2 「M3069Serial.atf」の情報

3. 使用方法

□ SPI インタフェースの使用方法

SPI インタフェースについて

SPI は同期式のシリアル通信方式です。チップセレクト(CS)、クロック(CLK)、データ出力(DO)、データ入力(DI)の 4 つの信号[1]で通信を行います。同じく 4 つの信号を使用する通信方式に Microwire と呼ばれるものもありますが、これら 2 つの方式にはあまり違いがありません。本ライブラリでは SPI インタフェース用の関数で、これら 2 つの通信方式をサポートします。

ハードウェアの接続

図 3、図 4 は SPI インタフェースの IC との接続例です。この例では P40～P43 の 4 つの端子を使用していますが、入出力を切り替え可能な端子であれば、他の端子も使用できます。具体的には、P40～P47、PA0～PA7 の端子が使用可能です。

以降の例では、この図のように P40 = CLK、P41 = DO、P42 = DI、P43 = CS として使用する場合について説明を行います。

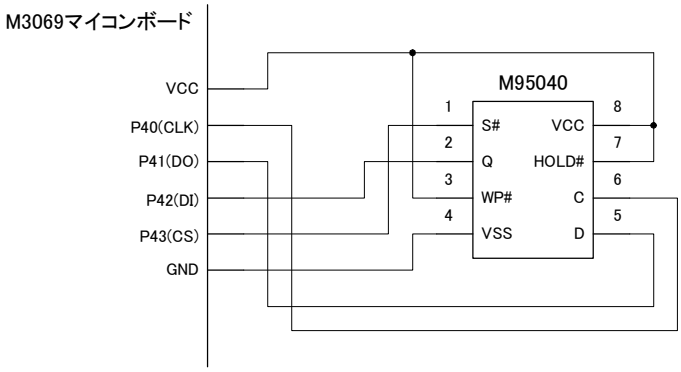

図 3　SPI インタフェース EEPROM「M95040」の接続例

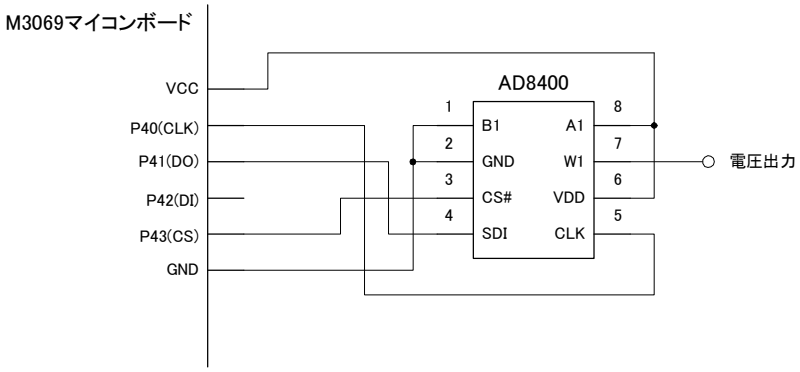

図 4　デジタルポテンショメータ「AD8400」の接続例

[1] データ出力とデータ入力に同じ信号ラインを使う場合もありますが、本ライブラリではそれぞれに信号ラインを割り当てた 4 線式の通信のみサポートします。

初期化作業

リスト 1 は SPI インタフェースの C 言語での初期化例です。

リスト 1

```
1  //接続
2  m_hDev = USBM_Open();
3
4  if(m_hDev){
5    //初期化
6    USBM_Initialize(m_hDev);
7
8    //ATF ファイルのダウンロード
9    if(ATF_SerialAttach(m_hDev)){
10     AfxMessageBox(_T("ATF ファイルのダウンロードに失敗しました"));
11     return;
12   }
13   //CS 信号(P43)を予め"Hi"に設定
14   USBM_PortWrite8(m_hDev, USBM_P4, 0x08);
15
16   //P40(CLK), P41(DO), P43(CS)を出力に設定
17   USBM_PortSetDir(m_hDev, USBM_P4, 0x0b);
18
19   //SPI に使用する端子を設定
20   ATF_SPISetPort(m_hDev, USBM_P4, 0, 1, 2, USBM_P4, 0x08);
21 }
```

2～6 行目は、ボードへの接続作業です。「USBM ライブラリ」を使用する場合と同様です。

9 行目で「M3069Serial.atf」をボードへダウンロードしています。ファイルは DLL ファイルと同じ順序で検索されますので、通常は呼び出し元の「.exe」ファイルと同じフォルダにおいてください。

14 行目は、接続相手のセレクト信号が負極性の場合に、CS 信号(P43)の初期値を"Hi"にするための処理です。「93C46」のようにセレクト信号が正極性となっている IC と接続する場合は必要ありません。

17 行目は、使用する P4 ポートの入出力方向を設定しています。この例のように、信号の方向設定は自動ではありませんので、*USBM_PortSetDir()* 関数を使用して別途行う必要があります。

20 行目は、使用するポートとビット番号を設定しています。CS 信号だけは、他の信号と別のポートを指定可能です。また、ビットの指定も番号ではなく、ビットマップで指定することに注意してください。この例では、P4 ポートの 3 ビット(0x08 と対応)を CS として使用します。

SPI の通信方法

リスト 2 は「AD8400」に SPI インタフェースを通じて 10 ビットのデータを送信する例です。また、図 5 は実際に出力された信号です。

リスト 2

```
1  //10 ビット書き込み(CSPOL=0,ORD=0, CPHA=0, CPOL=0)
2   ATF_SPIWriteRead(m_hDev, data, 10, NULL, 0, 0);
```

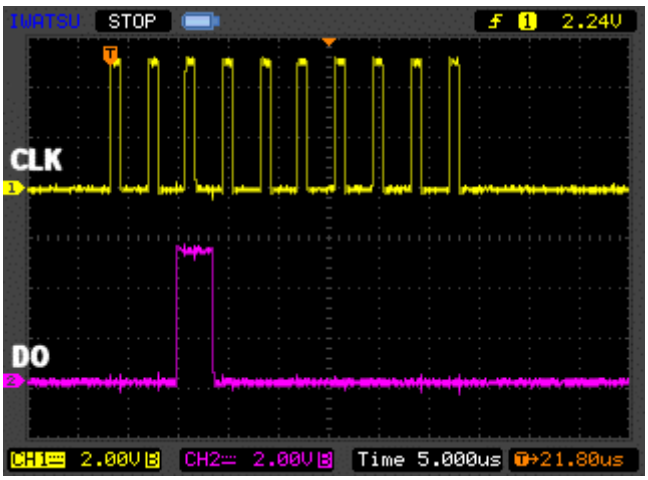

図 5　SPI の出力信号

リスト 3

```
TW_STATUS ATF_SPIWriteRead(TW_HANDLE hDev, void *pWrite, long nWriteBits,
                           void *pRead, long nReadBits, BYTE Mode);
```

リスト 3 は *ATF_SPIWriteRead()* 関数のプロトタイプ(C 言語)です。この関数はマスタ(マイコンボード)側から、任意のビット数のデータを出力した後、スレーブ(接続先の IC)から指定ビット数のデータを受信するための関数です。リスト 2 の例では、受信ビット数を 0 として送信だけを行っています。

ATF_SPIWriteRead() 関数の最後の引数(*Mode*)は、送受信の際の転送モードを示しています。SPI ではマスタとなる装置が、入出力のタイミングの基準となるクロックを出力します(本ライブラリではマイコンボード側が必ずマスタとなります)。ただし、送信側が出力データを操作するタイミングや、受信側がサンプリングを行うタイミングに厳密な規定があるわけではありません。そのため、本ライブラリでは送受信のタイミングを調整する機能を備えています。それを設定するのが送受信用の関数に用意されている *Mode* 引数です(表 6)。

表 6　*ATF_SPIWriteRead()* の *Mode* 引数

ビット	7	6	5	4	3	2	1	0
意味				CSPOL	ORD	CPOL	CPHA(T)	CPHA(R)

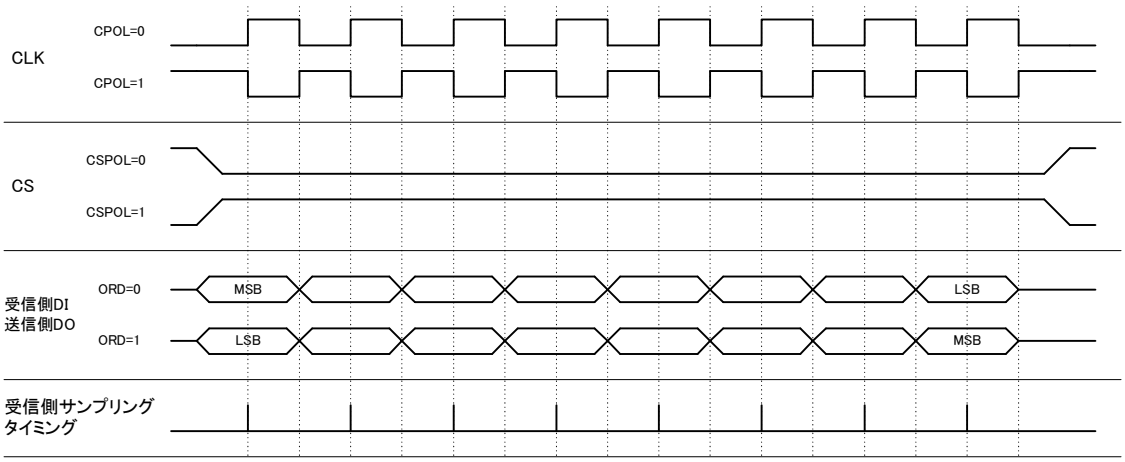

図 6　CPHA＝0 の場合の転送

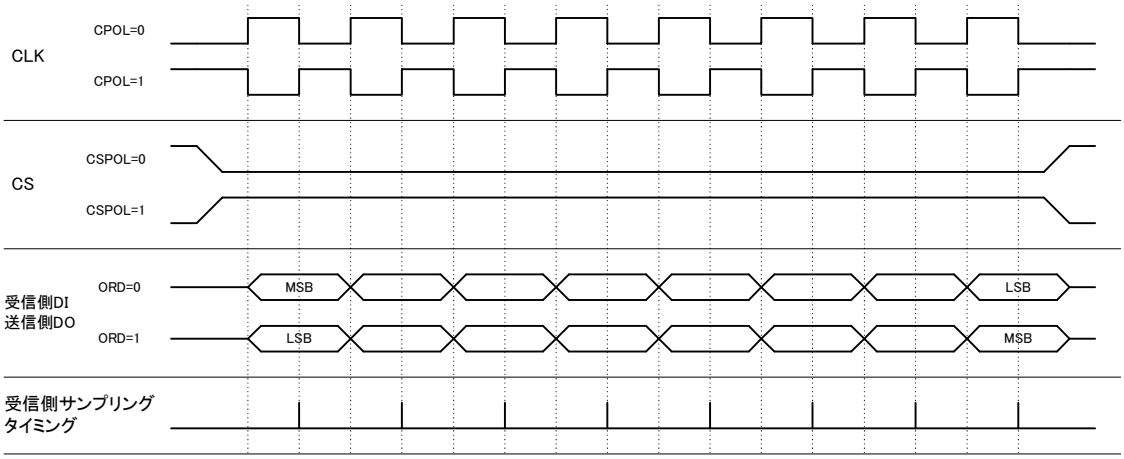

図 7　CPHA＝1 の場合の転送

Mode 引数の各ビットの値による送受信のタイミングの違いを示したのが図 6、および、図 7 です。*ATF_SPIWriteRead()* 関数には *CPHA* が 2 種類用意されています。*CPHA(T)* がマイコンボードから送信する場合で、*CPHA(R)* はマイコンボードが受信する場合のタイミングに適用されます。送受ともクロックの立ち上りに同期する（93C46 のような）一部の IC では、送信と受信でタイミングを変更する必要があります。

ATF_SPIWrite()、*ATF_SPIRead()* 関数では *Mode* 引数は以下のようになっています。

表 7　*ATF_SPIWrite()*、*ATF_SPIRead()* の *Mode* 引数

ビット	7	6	5	4	3	2	1	0
意味					CSPOL	ORD	CPOL	CPHA

データを送信する場合、図 8、および、図 9のような順序で送信されます。最後の 1 バイトに満たないデータは、MSB から送信するか、LSB から送信するかによって、どちら側にデータを詰めるかが変わりますので注意が必要です。受信する場合は、最初に受信したバイトがバッファの先頭から格納さ

れ、最後の 1 バイトに満たないデータは、送信の場合と同様の方向に詰められます。

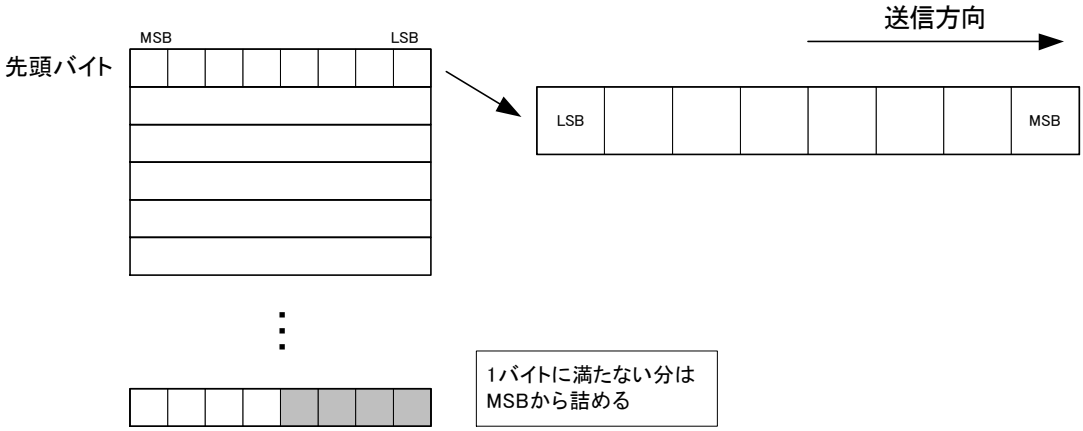

図 8　MSB から送信する場合(ORD=0)

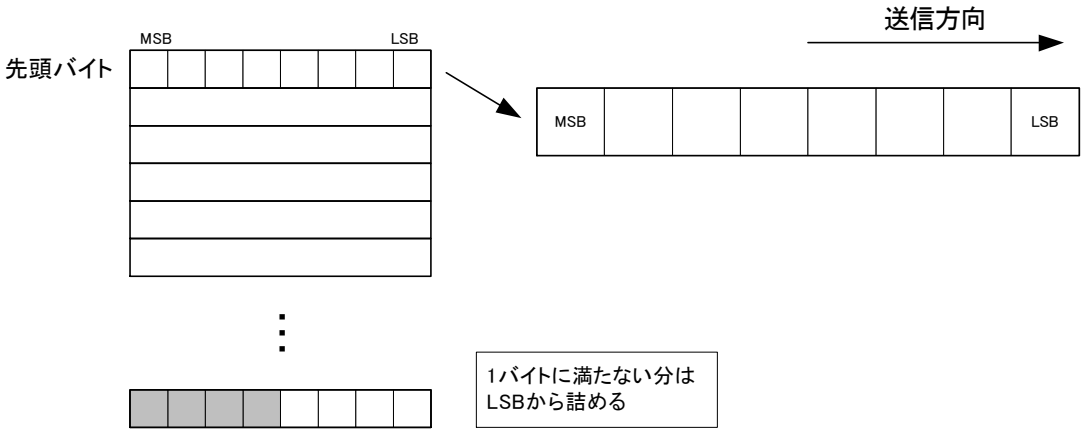

図 9　LSB から送信する場合(ORD=1)

EEPROM との通信

EEPROM は SPI や Microwire インタフェースの代表的な IC です。本ライブラリでは EEPROM の読み書きが簡単に行えるように専用の関数を用意しています。

ATF_EE25 で始まる関数は SPI インタフェースの EEPROM の制御関数です。代表的な IC は AT25010(Atmel Corporation)、M95010（STMicroelectronics）などです。

ATF_EE93 で始まる関数は Microwire インタフェースの EEPROM の制御関数です。代表的な IC は AT93C46(Atmel Corporation)、M93C46(STMicroelectronics)などです。

各関数の使用方法については、関数リファレンスやサンプルプログラムを参照してください。

□ I²C インタフェースの使用方法

I²C インタフェースについて

I²C-bus は 2 線式の同期シリアル通信の規格です。クロック(SCL)とデータ(SDA)の 2 つの信号で通信を行います。やはり、クロックを供給し通信を開始するマスタと、アクセスを待つスレーブ間で通信を行います。

SCL は必ずマスタが出力し、SDA は双方向です。どちらの信号もオープンコレクタで出力することになっています。

I²C はバスですので、同じ信号線上に複数の装置を接続することができます。規格では複数のマスタがバス上に存在しても構いませんが、本ライブラリでは複数マスタの通信には対応していません。ボードを接続するバスには、他のマスタデバイスを接続しないでください。スレーブは複数存在しても構いません。

ハードウェアの接続

図 10 は I²C インタフェースの IC との接続例です。この例では P40 と P41 の 2 つの端子を使用していますが、SPI 同様に入出力を切り替え可能な端子であれば、他の端子も使用できます。具体的には、P40～P47、PA0～PA7 の端子が使用可能です。

以降の例では、この図のように P40 = SCL、P41 = SDA として使用する場合について説明を行います。

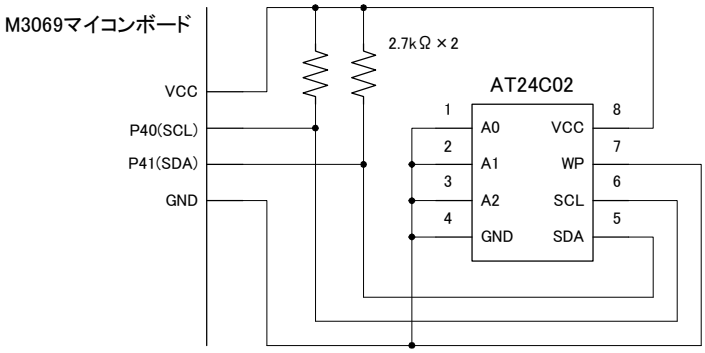

図 10　I²C インタフェース EEPROM「AT24C02」の接続例

初期化作業

リスト 4 は I²C インタフェースの C 言語での初期化例です。

リスト 4

```
1  //接続
2  m_hDev = USBM_Open();
3
4  if(m_hDev){
5    //初期化
6    USBM_Initialize(m_hDev);
7
8    //ATF ファイルのダウンロード
```

```
9     if(ATF_SerialAttach(m_hDev)){
10      AfxMessageBox(_T("ATF ファイルのダウンロードに失敗しました"));
11      return;
12    }
13
14    //I2C に使用する端子を設定
15    ATF_I2CSetPort(m_hDev, USBM_P4, 0, 1, 0);
16
17    //ウェイトの設定。スタンダードモード(100kHz 以下)の IC に接続する場合必要
18    ATF_I2CSetWait(m_hDev, 14);
19  }
```

ボードへの接続、ATF ファイルのダウンロードは SPI の場合と同様です。I²C インタフェースでは、ポートの入出力方向の操作はライブラリ内で行いますので、*USBM_PortSetDir()* の呼び出しは必要ありません。

15 行目は、I²C インタフェースで使用するポートと、ビット番号を設定しています。リスト 5 はここで使用している *ATF_I2CSetPort()* のプロトタイプ(C 言語)です。

リスト 5

```
TW_STATUS ATF_I2CSetPort(TW_HANDLE hDev, DWORD Port, long SclBit, long SdaBit, BYTE DDR);
```

ATF_I2CSetPort() 関数の最後の引数(*DDR* 引数)は、I²C インタフェースで使用するビット以外の入出力方向を指定するためのものです。

I²C インタフェースでは通信の際、使用する端子の入出力方向を切り替える必要があります。そのため、端子の入出力方向を操作するための、DDR というレジスタを頻繁に書き換えます。しかし、DDR は書き込み専用のレジスタなので、ビット操作で SCL と SDA のビットだけを操作することができません。そのため、*DDR* 引数で SCL と SDA 以外の端子方向を設定しておき、他のビットには毎回その値を書き込むようになっています。

図 11 は SCL=ビット 0、SDA=ビット 1 のときの、DDR への書き込みの様子です。ビット 2-7 は、*DDR* 引数で与えた値が書き込まれます。

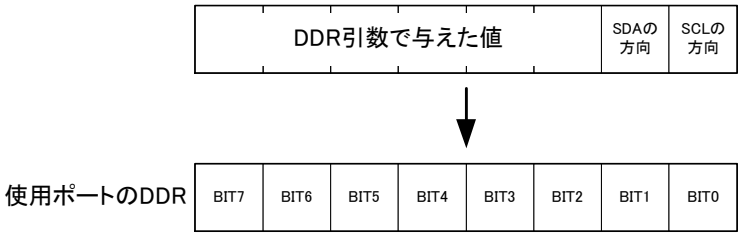

図 11　DDR へのアクセス

上記のことから、I²C インタフェースで使用中のポートは、途中で入出力方向を切り替えても、次の

I²C インタフェースへのアクセスで *ATF_I2CSetPort()* 関数で設定した方向に戻ってしまいますので、ご注意ください。

18 行目はウェイトの設定をしています。スタンダードモード(100kHz 以下)の装置に接続する際は、14 以上のウェイト値を設定する必要があります。

I²C の通信方法

図 12、図 13 は、それぞれ I2C の基本的な書き込み、および、読み出しの様子を示したものです。どちらも、最初の 1 バイトはマスタから送信され、スレーブの選択と、書き込み／読み出し方向の指示を行います。データは必ず 8 ビット単位で転送を行い、MSB から順に送信することになっています。

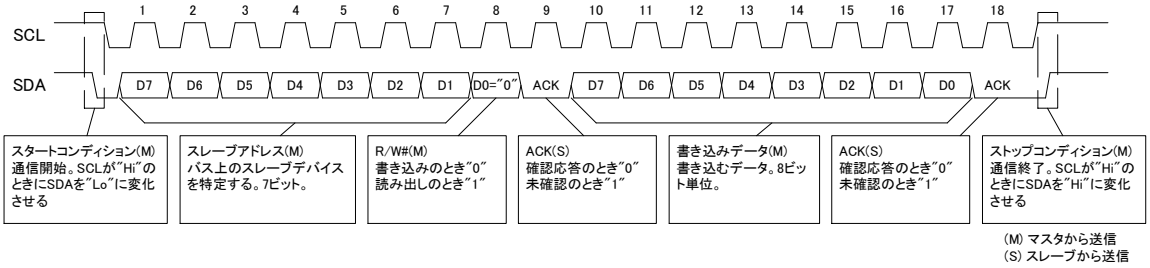

図 12　マスタからスレーブへの書き込み

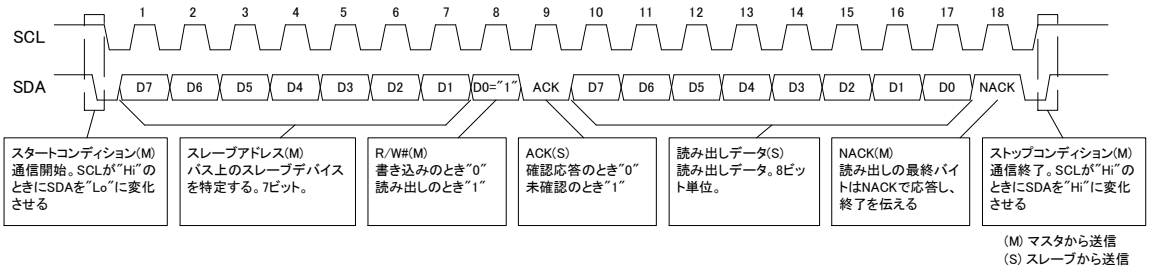

図 13　スレーブからマスタへの読み出し

本ライブラリで書き込み、読み出しを行うには、*ATF_I2CWrite()* 、*ATF_I2CRead()* 関数を使用します。リスト 6 にそれぞれのプロトタイプ(C 言語)を示します。

リスト 6

```
TW_STATUS ATF_I2CWrite(TW_HANDLE hDev, BYTE SlaveAddr, void *pData, long *pnWrite, long Condition)
TW_STATUS ATF_I2CRead(TW_HANDLE hDev, BYTE SlaveAddr, void *pData, long *pnRead, long Condition)
```

SlaveAddr 引数は通信の第 1 バイトで送信されるスレーブアドレスで、バス上のスレーブデバイスを特定します。この引数には、7 ビットのデータを MSB から詰めて渡します。書き込み／読み出しの方向を示すビットは自動的にセットされますので、ビット 0 は無視されます。

最後の引数 *Condition* は、書き込み、または、読み出しサイクルの前後に、スタートコンディション、

ストップコンディションを出力するかどうかを指定します。単純に1回の書き込みや読み出しで転送が終了する場合には、スタートコンディション、ストップコンディション両方を出力するようにします。しかし、複数の書き込みや、読み出しサイクルを続けて行うような場合には、この引数を制御する必要があります。

例えば、I²C インタフェースの EEPROM は、メモリの値を読み出す場合に、スレーブアドレスとは別に、メモリ番地を示すアドレス値を書き込む必要があります。このような場合、まずメモリアドレスを指定するための書き込みサイクルを実行し、続いて実際にメモリデータを取り出す読み出しサイクルを実行します。このとき、マスタはメモリアドレスの書き込みサイクルの後にはストップコンディションを出力せず、再びスタートコンディションを出力して新しい読み出しサイクルを開始します(図 14)。

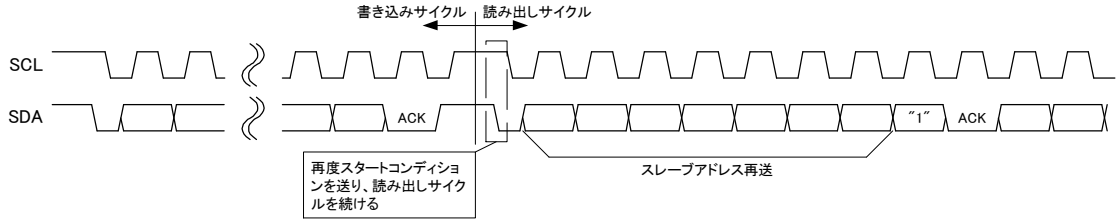

図 14　スタートコンディションの再送

実際には、EEPROM と接続する場合は、次に説明する専用の関数が用意されていますので、それらをご使用ください。

EEPROM との通信

SPI の場合と同様、EEPROM との通信には、専用の関数、*ATF_EE24Write()*、*ATF_EE24Read()* が用意されています。I²C インタフェースの EEPROM としては、AT24C02(Atmel Corporation)、M24C02(STMicroelectronics)などがあります。

EEPROM デバイスのスレーブアドレスは上位 4 ビットが固定("B'1010")となっています。残り 3 ビットでバス上のデバイスを選択することができます(図 15)。

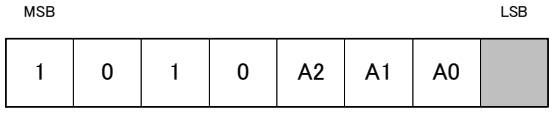

図 15　EEPROM のスレーブアドレス

図 10 の接続例では EEPROM デバイスのアドレスは全て"0"に設定されていますので、関数に与える *SlaveAddr* 引数の値は"B'10100000"(H'A0)となります。

メモリアドレスのビット数が 9、10、11、17 などの EEPROM では、メモリアドレスの一部をスレーブアドレスの A2、A1、A0 の代わりに入力する仕様となっていますが、これらはライブラリ内で自動的に処理されますので呼び出し側で意識する必要はありません。

4. 関数リファレンス

各関数の説明は、C 言語、Visual Basic®(6.0 以前)、Visual Basic.NET (Visual Basic 2005)それぞれにおけるプロトタイプ、変数の説明、動作説明の順になっています。

ほとんどの関数の戻り値は 32 ビットの整数で関数の実行結果を表します(以下参照)。関数がそれ以外の特別な戻り値を返す場合は、各関数の動作説明の欄で内容を示します。

□ 関数の戻り値

表 8 は、「USBM ライブラリ」の関数の戻り値です。戻り値を示す各定数は、「USBM ライブラリ」の各言語用の定義ファイル(拡張子が「.h」、「.bas」、「.vb」のファイル)中で定義されています。

これに加えて、本ライブラリでは表 9 の値が関数から返される場合があります。

表 8 関数の戻り値

定数	値	意味
TW_OK	0x00000000	正常終了
TW_INVALID_HANDLE	0x00000001	デバイスのハンドルが無効
TW_DEVICE_NOT_FOUND	0x00000002	デバイスが見つからない
TW_IO_ERROR	0x00000004	送受信中にエラーが発生した
TW_INSUFFICIENT_RESOURCES	0x00000005	リソースエラー(デバイスの最大接続数を超えた場合など)
TW_INVALID_ARGS	0x00000010	関数に渡された引数が無効
TW_NOT_SUPPORTED	0x00000011	サポートされない機能
TW_OTHER_ERROR	0x00000012	
TW_TIMEOUT	0xffff0001	送信または受信処理がタイムアウトした
TW_FILE_ERROR	0xffff0002	ファイル操作に関するエラーが発生した
TW_MEMORY_ERROR	0xffff0003	メモリの確保に失敗した
TW_DATA_NOT_FOUND	0xffff0004	有効なデータが見つからなかった
TW_SOCKET_ERROR	0xffff0005	Winsock のエラー(多くの場合 WSAGetLastError() を呼び出すことさらに詳しい情報を得ることができます)
TW_ACCESS_DENIED	0xffff0006	デバイスとの認証作業に失敗した
TW_ATF_ERR_FILE_VERSION	0xffff0101	対応するバージョンより新しい ATF ファイルフォーマット
TW_ATF_ERR_ILLEGAL_FILE	0xffff0102	ATF ファイルの内容が不正
TW_ATF_ERR_SERVICE_VERSION	0xffff0103	ファームのバージョンが ATF ファイルの要求より古い

表 9 ライブラリ固有の戻り値

定数	値	意味
TW_I2CS_TIMEOUT	0xfffffe01	I2C で SCL が解放されなかった場合に返ります。
TW_I2CS_NACK	0xfffffe02	I2C で正常に ACK が受信できなかった場合に返ります。

□ データバッファを渡す場合の注意

バージョン 6.0 以前の Visual Basic で関数に配列を渡す場合、「USBM ライブラリ」では直接 SAFEARRAY 型の配列を渡せる仕様となっていますが、本ライブラリでは SAFEARRAY 型の配列渡しをサポートしていません。データバッファとして配列を渡す場合は、以下の例のように配列の先頭要素を渡してください。

表 10　Visual Basic 6.0 以前での配列渡し

```
Dim Data(255) As Byte

ATF_EE25Read g_hDev, 0, Data(0), 256, 9
```

□ 初期化／終了用関数

ATF_SerialAttach()

```
TW_STATUS ATF_SerialAttach(TW_HANDLE hDev)
Function ATF_SerialAttach(ByVal hDev As Long) As Long
Function ATF_SerialAttach(ByVal hDev As Integer) As Integer
```

hDev　：デバイスのハンドル

シリアルインタフェース用のアタッチメントファーム(M3069Serial.atf)をデバイスにダウンロードします。他のライブラリ関数の使用に先立って必ず呼び出す必要があります。

ATF_SerialDetach()

```
TW_STATUS ATF_SerialDetach(TW_HANDLE hDev)
Function ATF_SerialDetach(ByVal hDev As Long) As Long
Function ATF_SerialDetach(ByVal hDev As Integer) As Integer
```

hDev　：デバイスのハンドル

シリアルインタフェース用のアタッチメントファームの使用を終了します。

□ SPI／Microwire インタフェース汎用関数

使用するポートの入出力方向の設定は自動では行われませんので *USBM_PortSetDir()* 関数で設定する必要があります。

極性の違うCSを同時に扱うことができませんので、同時に接続する場合は、それぞれにアクセスする前に *ATF_SPISetPort()* で設定を変更してください。

ATF_SPISetPort()

```
TW_STATUS ATF_SPISetPort(TW_HANDLE hDev, DWORD Port, long CLKBit,
                         long DOBit, long DIBit, DWORD CSPort, BYTE CSBits)

Function ATF_SPISetPort(ByVal hDev As Long, ByVal Port As Long, ByVal CLKBit As Long,
                        ByVal DOBit As Long, ByVal DIBit As Long, ByVal CSPort As Long,
                        ByVal CSBits As Byte) As Long

Function ATF_SPISetPort(ByVal hDev As Integer, ByVal Port As Integer, ByVal CLKBit As Integer,
                        ByVal DOBit As Integer, ByVal DIBit As Integer, ByVal CSPort As Integer,
                        ByVal CSBits As Byte) As Integer
```

hDev　: デバイスのハンドル
Port　: CLK,DO,DI 端子として使用するポート(USBM_P4,USBM_PA)
CLKBit : CLK に使用するビット番号(0-7)
DOBit　: DO に使用するビット番号(0-7)
DIBit　: DI に使用するビット番号(0-7)
CSPort : CS に使用するポート(USBM_P4,USBM_PA)
CSBits : CS に使用するビット(ビットマップで指定)

SPI インタフェースに使用するポートを指定します。CS に使用するポートは他のビットと別に指定できます。CSBits のみ番号ではなくビットマップで指定することに注意してください。例えばビット 7 を使用する場合の設定値は 0x80 です。CSBits は複数指定可能ですので複数デバイスへの同時書き込みなどが可能です（読み出しは同時にはできません）。アクセス前後に CS の操作をしない場合、CSPort,CSBits とも 0 としてください。

ATF_SPISetWait()

```
TW_STATUS ATF_SPISetWait(TW_HANDLE hDev, WORD Wait)

Function ATF_SPISetWait(ByVal hDev As Long, ByVal Wait As Integer) As Long

Function ATF_SPISetWait(ByVal hDev As Integer, ByVal Wait As Short) As Integer
```

hDev : デバイスのハンドル
Wait : ウェイトパラメータ

ウェイトを設定します。ウェイトパラメータは 1 増える毎に約 480ns ずつクロックサイクルが伸びます。

ATF_SPIWriteRead()

```
TW_STATUS ATF_SPIWriteRead(TW_HANDLE hDev, void *pWrite, long nWriteBits,
                           void *pRead, long nReadBits, BYTE Mode)

Function ATF_SPIWriteRead(ByVal hDev As Long, ByRef pWrite As Any, ByVal nWriteBits As Long,
                          ByRef pRead As Any, ByVal nReadBits As Long, ByVal Mode As Byte) As Long

Function ATF_SPIWriteRead(ByVal hDev As Integer, ByVal pWrite As Object,
                          ByVal nWriteBits As Integer, ByVal pRead() As Byte,
                          ByVal nReadBits As Integer, ByVal Mode As Byte) As Integer
```

hDev : デバイスのハンドル
pWrite : [in]書き込みデータ
nWriteBits : 書き込みビット数
pRead : [out]読み出したデータの格納先
nReadBits : 読み出しビット数
Mode : 動作モード
ビット0 読み出し時クロック位相(CPHA)
ビット1 書き込み時クロック位相(CPHA)
ビット2 クロック極性(CPOL)
ビット3 データオーダー(ORD) 0:MSB から転送、1:LSB から転送
ビット4 CS 極性(CSPOL) 0:"Lo"アクティブ、1:"Hi"アクティブ

SPI インタフェースに書き込みを行なった後、読み出しを行ないます。アクセスの前に CS 対応ビットをアクティブにし、アクセス後にインアクティブに戻します。アクセス中の CS のレベルは Mode のビット4で決まります。

転送ビット数が 8 で割り切れない場合、最終バイトは MSB から転送する場合は MSB 側に、LSB から転送する場合には LSB 側にデータを詰めてください。

クロック位相は書き込み時と読み出し時に別々の設定が可能です。例えば 93C46 のようなデバイスにアクセスする場合､書き込み時のクロック位相ビットは0､読み出し時のクロック位相ビットを1とします。この場合、書き込み時にはスレーブ側がクロックの立ち上りでデータをサンプリングし、読み出し時にはマスタ側がクロックの立下りでデータをサンプリングします。

ATF_SPIWrite()

```
TW_STATUS ATF_SPIWrite(TW_HANDLE hDev, void *pWrite, long nWriteBits, BYTE Mode)

Function ATF_SPIWrite(ByVal hDev As Long, ByRef pWrite As Any,
                      ByVal nWriteBits As Long, ByVal Mode As Byte) As Long

Function ATF_SPIWrite(ByVal hDev As Integer, ByVal pWrite As Object,
                      ByVal nWriteBits As Integer, ByVal Mode As Byte) As Integer
```

hDev : デバイスのハンドル
pWrite : [in]書き込みデータ
nWriteBits : 書き込むビット数(0-2048)
Mode : 動作モード
ビット0 クロック位相(CPHA)
ビット1 クロック極性(CPOL)
ビット2 データオーダー(0:MSB から転送、1:LSB から転送)

SPI インタフェースからデータを出力します。CS(チップセレクト)信号は操作されません。別途ポート操作などで制御する必要があります。

ATF_SPIRead()

```
TW_STATUS ATF_SPIRead(TW_HANDLE hDev, void *pRead, long nReadBits, BYTE Mode)

Function ATF_SPIRead(ByVal hDev As Long, ByRef pRead As Any,
                     ByVal nReadBits As Long, ByVal Mode As Byte) As Long

Function ATF_SPIRead(ByVal hDev As Integer, ByVal pRead() As Byte,
                     ByVal nReadBits As Integer, ByVal Mode As Byte) As Integer
```

hDev : デバイスのハンドル
pRead : [out]読み出したデータの格納先
nReadBits : 読み出すビット数(0-2048)
Mode : 動作モード
ビット0 クロック位相(CPHA)
ビット1 クロック極性(CPOL)
ビット2 データオーダー(0:MSB から転送、1:LSB から転送)

SPI インタフェースからデータを入力します。CS(チップセレクト)信号は操作されません。別途ポート操作などで制御する必要があります。

□ SPI インタフェース EEPROM 制御用関数

SPI インタフェースの EEPROM を制御するための関数です。*ATF_SPISetPort()* 関数で SPI インタフェースを初期化した後、呼び出しが可能になります。ウェイトを指定するには *ATF_SPISetWait()* 関数を呼び出します。

ATF_EE25ReadStatus()

```
TW_STATUS ATF_EE25ReadStatus(TW_HANDLE hDev, BYTE *pStatus)
Function ATF_EE25ReadStatus(ByVal hDev As Long, ByRef pStatus As Byte) As Long
Function ATF_EE25ReadStatus(ByVal hDev As Integer, ByRef pStatus As Byte) As Integer
```

hDev : デバイスのハンドル
pStatus : [out]ステータスの格納先

ステータスを読み出します。

ATF_EE25WriteStatus()

```
TW_STATUS ATF_EE25WriteStatus(TW_HANDLE hDev, BYTE Status)
Function ATF_EE25WriteStatus(ByVal hDev As Long, ByVal Status As Byte) As Long
Function ATF_EE25WriteStatus(ByVal hDev As Integer, ByVal Status As Byte) As Integer
```

hDev : デバイスのハンドル
Status : 書き込むステータス値

ステータスを書き込みます。書き込み許可コマンドは自動で送信されます。

ATF_EE25Write()

```
TW_STATUS ATF_EE25Write(TW_HANDLE hDev, DWORD Addr, void *pData, long nWrite, long nAddrBits)
Function ATF_EE25Write(ByVal hDev As Long, ByVal Addr As Long, ByRef pData As Any,
                       ByVal nWrite As Long, ByVal nAddrBits As Long) As Long
Function ATF_EE25Write(ByVal hDev As Integer, ByVal Addr As Integer, ByVal pData As Object,
                       ByVal nWrite As Integer, ByVal nAddrBits As Integer) As Integer
```

hDev : デバイスのハンドル
Addr : EEPROM のアドレス
pData : [in]書き込みデータ
nWrite : 書き込みバイト数(0-256)
nAddrBits : EEPROM のアドレスのビット数(8-32)

EEPROM に書き込みを行います。書き込みはバイト単位で行われます。書き込み許可コマンドは自動で送信されます。EEPROM にページライト機能がある場合は、複数データを一度に書き込めます。

ATF_EE25Read()

```
TW_STATUS ATF_EE25Read(TW_HANDLE hDev, DWORD Addr, void *pData, long nRead, long nAddrBits)

Function ATF_EE25Read(ByVal hDev As Long, ByVal Addr As Long, ByRef pData As Any,
                      ByVal nRead As Long, ByVal nAddrBits As Long) As Long

Function ATF_EE25Read(ByVal hDev As Integer, ByVal Addr As Integer, ByVal pData() As Byte,
                      ByVal nRead As Integer, ByVal nAddrBits As Integer) As Integer
```

hDev	: デバイスのハンドル
Addr	: EEPROM のアドレス
pData	: [out]読み出したデータの格納先
nRead	: 読み出しバイト数(0-256)
nAddrBits	: EEPROM のアドレスのビット数(8-32)

EEPROM からデータを読み出します。読み出しはバイト単位で行われます。EEPROM にシーケンシャルリード機能がある場合は複数データを一度に読めます。

□ Microwire インタフェース EEPROM 制御用関数

Microwire インタフェースの EEPROM を制御するための関数です。*ATF_SPISetPort()* 関数で SPI インタフェースを初期化した後、呼び出しが可能になります。ウェイトを指定するには *ATF_SPISetWait()* 関数を呼び出します。

ATF_EE93WriteEnable()

```
TW_STATUS ATF_EE93WriteEnable(TW_HANDLE hDev, BOOL flgEnable, long nAddrBits)

Function ATF_EE93WriteEnable(ByVal hDev As Long, ByVal flgEnable As Long,
                             ByVal nAddrBits As Long) As Long

Function ATF_EE93WriteEnable(ByVal hDev As Integer, ByVal flgEnable As Integer,
                             ByVal nAddrBits As Integer) As Integer
```

hDev : デバイスのハンドル
flgEnable : 書き込み許可/禁止
TRUE(0 以外) 書き込み許可
FALSE(0) 書き込み禁止
nAddrBits : アドレスビット数(0-32)

書き込み許可／禁止コマンドを送信します。

ATF_EE93EraseAll()

```
TW_STATUS ATF_EE93EraseAll(TW_HANDLE hDev, long nAddrBits)

Function ATF_EE93EraseAll(ByVal hDev As Long, ByVal nAddrBits As Long) As Long

Function ATF_EE93EraseAll(ByVal hDev As Integer, ByVal nAddrBits As Integer) As Integer
```

hDev : デバイスのハンドル
nAddrBits : アドレスのビット数(0-32)

全てのメモリ領域を消去します。書き込み終了後、チップがレディになるのを待って関数から戻りますので、ATF_EE93IsReady()の呼び出しは必要ありません。

ATF_EE93WriteAll()

```
TW_STATUS ATF_EE93WriteAll(TW_HANDLE hDev, WORD Data, long nAddrBits)

Function ATF_EE93WriteAll(ByVal hDev As Long, ByVal Data As Integer,
                          ByVal nAddrBits As Long) As Long

Function ATF_EE93WriteAll(ByVal hDev As Integer, ByVal Data As Short,
                          ByVal nAddrBits As Integer) As Integer
```

hDev : デバイスのハンドル
Data : 書き込むデータ(16 ビット値)
nAddrBits : アドレスのビット数(0-32)

全てのメモリ領域に同じ値を書き込みます。書き込み終了後、チップがレディになるのを待って関数から戻りますので、ATF_EE93IsReady()の呼び出しは必要ありません。

ATF_EE93Write()

```
TW_STATUS ATF_EE93Write(TW_HANDLE hDev, DWORD Addr, void *pData, WORD nDataWords, long nAddrBits)

Function ATF_EE93Write(ByVal hDev As Long, ByVal Addr As Long, ByRef pData As Any,
                       ByVal nDataWords As Integer, ByVal nAddrBits As Long) As Long

Function ATF_EE93Write(ByVal hDev As Integer, ByVal Addr As Integer, ByVal pData As Object,
                       ByVal nDataWords As Short, ByVal nAddrBits As Integer) As Integer
```

```
hDev       : デバイスのハンドル
Addr       : 書き込みを行うアドレス
pData      : [in]書き込むデータ
nDataWords : 書き込むデータ数(ワード単位、0-128)
nAddrBits  : アドレスのビット数(0-32)
```

データを書き込みます。書き込みは 16 ビット単位で行われますので注意してください。EEPROM にページライト機能がある場合は、複数データを一度に書き込めます。書き込み終了後、チップがレディになるのを待って関数から戻りますので、ATF_EE93IsReady()の呼び出しは必要ありません。

ATF_EE93Read()

```
TW_STATUS ATF_EE93Read(TW_HANDLE hDev, DWORD Addr, void *pData, WORD nDataWords, long nAddrBits)

Function ATF_EE93Read(ByVal hDev As Long, ByVal Addr As Long, ByVal pData As Any,
                      ByVal nDataWords As Integer, ByVal nAddrBits As Long) As Long

Function ATF_EE93Read(ByVal hDev As Integer, ByVal Addr As Integer, ByVal pData() As Byte[2],
                      ByVal nDataWords As Short, ByVal nAddrBits As Integer) As Integer
```

```
hDev       : デバイスのハンドル
Addr       : 読み出しを行うアドレス
pData      : [out]読み出したデータの格納先
nDataWords : 読み出すデータ数(ワード単位、0-128)
nAddrBits  : アドレスのビット数(0-32)
```

データを読み出します。読み出し 16 ビット単位で行われますので注意してください。EEPROM にシーケンシャルリード機能がある場合は複数データを一度に読めます。

ATF_EE93IsReady()

```
TW_STATUS ATF_EE93IsReady(TW_HANDLE hDev, BOOL *pflgReady)

Function ATF_EE93IsReady(ByVal hDev As Long, ByRef pflgReady As Long) As Long

Function ATF_EE93IsReady(ByVal hDev As Integer, ByRef pflgReady As Integer) As Integer
```

```
hDev      : デバイスのハンドル
pflgReady : [out]チップのアクセス可否
            TRUE(1)  アクセス可能
            FALSE(0) 書き込み/消去動作中。アクセス不可
```

チップにアクセス可能か調べます。

[2] Short 型のオーバーロードが定義されています。

□ I²C インタフェース汎用関数

ATF_I2CSetPort()

```
TW_STATUS ATF_I2CSetPort(TW_HANDLE hDev, DWORD Port, long SclBit, long SdaBit, BYTE DDR)

Function ATF_I2CSetPort(ByVal hDev As Long, ByVal Port As Long, ByVal SclBit As Long,
                        ByVal SdaBit As Long, ByVal DDR As Byte) As Long

Function ATF_I2CSetPort(ByVal hDev As Integer, ByVal Port As Integer, ByVal SclBit As Integer,
                        ByVal SdaBit As Integer, ByVal DDR As Byte) As Integer
```

hDev　: デバイスのハンドル
Port　: 使用するポート(USBM_P4,USBM_PA)
SclBit : SCL として使用するビットの番号(0-7)
SdaBit : SDA として使用するビットの番号(0-7)
DDR　: 使用する DDR の規定値

I2C インタフェースとして使用するポートを設定します。引数 DDR は使用するポートの DDR(ビットの入出力方向を操作するレジスタ)に設定する値を与えます。I2C では出力がオープンコレクタとなっているため使用するポートの DDR を操作します。しかし、DDR はビット単位の操作がきないので、毎回全てのビットが書換えられてしまい、I2C で使用しないビットも上書きされてしまいます。そのため、このライブラリでは I2C で使用しないビットの状態を保存するために予め DDR の値を記録しておき、他のビットにはその値を書き込むようにしています。

ATF_I2CSetWait()

```
TW_STATUS ATF_I2CSetWait(TW_HANDLE hDev, BYTE Wait)

Function ATF_I2CSetWait(ByVal hDev As Long, ByVal Wait As Byte) As Long

Function ATF_I2CSetWait(ByVal hDev As Integer, ByVal Wait As Byte) As Integer
```

hDev : デバイスのハンドル
Wait : リード/ライトのウェイト設定

I2C の転送速度を制御します。Wait の値は 1 毎に 480nsec ずつクロックサイクルが伸びます。ファーストモード(400KHz 以下)の場合 0 以上、スタンダードモード(100KHz 以下)の場合 14 以上としてください。

ATF_I2CWrite()

```
TW_STATUS ATF_I2CWrite(TW_HANDLE hDev, BYTE SlaveAddr, void *pData, long *pnWrite, long Condition)

Function ATF_I2CWrite(ByVal hDev As Long, ByVal SlaveAddr As Byte, ByRef pData As Any,
                      ByRef pnWrite As Long, ByVal Condition As Long) As Long

Function ATF_I2CWrite(ByVal hDev As Integer, ByVal SlaveAddr As Byte, ByVal pData As Object,
                      ByRef pnWrite As Integer, ByVal Condition As Integer) As Integer
```

hDev : デバイスのハンドル
SlaveAddr : スレーブアドレス
pData : [in]書き込むデータ
pnWrite : [in]書き込むデータのバイト数(256 以下)
[out]実際に書き込めたバイト数
Condition : 送信前後にスタートコンディション、ストップコンディションを出力するかどうかを指定します。以下を OR で結合します。
I2C_START(0x01) 送信前にスタートコンディションを出力します。
I2C_STOP(0x02) 送信後にストップコンディションを出力します。

戻り値 : 以下のライブラリ独自のステータスを返す場合があります
TW_I2CS_TIMEOUT(0xfffffe01) SCL の解放待ちでタイムアウトした
TW_I2CS_NACK(0xfffffe02) ACK が返らなかった

I2C インタフェースを通じてデータを書き込みます。スレーブアドレスはスタートコンディションを出力する場合のみ送信されます。スレーブアドレスは MSB から詰めて入力します。LSB は自動で R/W がセットされます。

ATF_I2CRead()

```
TW_STATUS ATF_I2CRead(TW_HANDLE hDev, BYTE SlaveAddr, void *pData, long *pnRead, long Condition)

Function ATF_I2CRead(ByVal hDev As Long, ByVal SlaveAddr As Byte, ByRef pData As Any,
                     ByRef pnRead As Long, ByVal Condition As Long) As Long

Function ATF_I2CRead(ByVal hDev As Integer, ByVal SlaveAddr As Byte, ByVal pData() As Byte,
                     ByRef pnRead As Integer, ByVal Condition As Integer) As Integer
```

hDev : デバイスのハンドル
SlaveAddr : スレーブデバイスのアドレス
pData : [out]読み出したデータの格納先
pnRead : [in]読み出すデータのバイト数(256 以下)
[out]実際に読み出したデータのバイト数
Condition : 受信前後にスタートコンディション、ストップコンディションを出力するかどうかを指定します。以下を OR で結合します。
I2C_START(0x01) 送信前にスタートコンディションを出力します。
I2C_STOP(0x02) 送信後にストップコンディションを出力します。

戻り値 : 以下のライブラリ独自のステータスを返す場合があります
TW_I2CS_TIMEOUT(0xfffffe01) SCL の解放待ちでタイムアウトした
TW_I2CS_NACK(0xfffffe02) ACK が返らなかった

I2C インタフェースからデータを読みます。スレーブアドレスはスタートコンディションを出力する場合のみ送信されます。スレーブアドレスは MSB から詰めて入力します。LSB は無視されます。ストップコンディションを出力する場合は読み出しの最終バイトには ACK を返しません。

□ I²C インタフェース EEPROM 制御用関数

I²C インタフェースの EEPROM を制御するための関数です。*ATF_I2CSetPort()* 関数で I²C インタフェースを初期化した後、呼び出しが可能になります。ウェイトを指定するには *ATF_I2CSetWait()* 関数を呼び出します。

ATF_EE24Write()

```
TW_STATUS ATF_EE24Write(TW_HANDLE hDev, BYTE SlaveAddr, DWORD Addr,
                        void *pData, long *pnWrite, long nAddrBits)

Function ATF_EE24Write(ByVal hDev As Long, ByVal SlaveAddr As Byte, ByVal Addr As Long,
                       ByRef pData As Any, ByRef pnWrite As Long, ByVal nAddrBits As Long) As Long

Function ATF_EE24Write(ByVal hDev As Integer, ByVal SlaveAddr As Byte,
                       ByVal Addr As Integer, ByVal pData As Object,
                       ByRef pnWrite As Integer, ByVal nAddrBits As Integer) As Integer
```

```
hDev      : デバイスのハンドル
SlaveAddr : スレーブアドレス。
Addr      : アクセスするメモリアドレス
pData     : [in]書き込むデータ
pnWrite   : [in]書き込むデータのバイト数(256 以下)
            [out]実際に書き込めたバイト数
nAddrBits : メモリデバイスのアドレスビット数
```

I2C の EEPROM に書き込みを行います。スレーブアドレスは MSB から詰めて入力します。EEPROM にアクセスする場合は上位 4 ビット B'1010 とします。LSB は自動的に R/W がセットされます。アドレスビット数が 9,10,11,17 のデバイスについては自動的にアドレスの上位ビットをスレーブアドレスに格納します。

ATF_EE24Read()

```
TW_STATUS ATF_EE24Read(TW_HANDLE hDev, BYTE SlaveAddr, DWORD Addr,
                       void *pData, long *pnRead, long nAddrBits)

Function ATF_EE24Read(ByVal hDev As Long, ByVal SlaveAddr As Byte, ByVal Addr As Long,
                      ByRef pData As Any, ByRef pnRead As Long, ByVal nAddrBits As Long) As Long

Function ATF_EE24Read(ByVal hDev As Integer, ByVal SlaveAddr As Byte,
                      ByVal Addr As Integer, ByVal pData() As Byte,
                      ByRef pnRead As Integer, ByVal nAddrBits As Integer) As Integer
```

```
hDev      : デバイスのハンドル
SlaveAddr : スレーブアドレス。
Addr      : アクセスするメモリアドレス
pData     : [out]読み出したデータ
pnRead    : [in]読み出すデータのバイト数(256 以下)
            [out]実際に読み出したデータのバイト数
nAddrBits : メモリデバイスのアドレスビット数
```

I2C の EEPROM から読み出します。スレーブアドレスは MSB から詰めて入力します。EEPROM にアクセスする場合は上位 4 ビット B'1010 とします。LSB は自動的に R/W がセットされます。アドレスビット数が 9,10,11,17 のデバイスについては自動的にアドレスの上位ビットをスレーブアドレスに格納します。

サポート情報

製品に関する情報、最新のファームウェア、ユーティリティなどは弊社ホームページにてご案内しております。また、お問い合わせ、ご質問などは下記までご連絡ください。

テクノウェーブ(株)
URL : http://www.techw.co.jp
E-mail : support@techw.co.jp

改訂記録

年月	版	改訂内容
2007 年 11 月	初	
2007 年 12 月	2	「AD8400」の型式表示の誤りを修正