

# MC9S08JM による USB アプリケーションのカスタマイズ

## MC9S08JM デバイス向けフリースケール USB スタックの詳細

執筆 : Derek Liu, Jos? Ruiz, Eduardo Viramontes  
中国システム / アプリケーション・チームおよび RTAC Americas

### 1 USB スタックの紹介

MC9S08JM デバイスは、フリースケールの Flexis™ マイクロコントローラ・シリーズの製品です。Flexis シリーズには、1つの開発ツールと同一のペリフェラル群を共有するピン互換の 8 ビットと 32 ビットのマイクロコントローラがあります。Flexis は、8 ビットと 32 ビットの互換性を実現するフリースケールの Controller Continuum の接点を形成します。

8 ビットの MC9S08JM シリーズの MCU は、オンチップ・トランシーバを装備するフルスピード USB 2.0 のデバイスで、クラス最高の USB モジュール性能、システム・インテグレーション、およびソフトウェア・サポートを提供します。MC9S08JM ファミリの MCU の USB モジュールは、7 個のエンドポイントと 256 バイトの RAM を搭載します。

USB に対応するデバイスは、デバイスとホスト間で通信トランザクションを実行するためのソフトウェアが必要です。USB モジュールは、物理シグナリングやデータ・フィルタなどの USB プロトコルのさまざまなコンポーネントを内蔵しますが、これらのレイヤの制御にはソフトウェアを使用します。このソフトウェアが USB スタックです。USB スタックは、USB モジュールの設定、USB のエニュメレーション、トランザクション・タイプの設定、エンドポイントの操作、およびデータの送受信などのトランザクションを処理します。スタックを利用すると、設定が完了した後のユーザ・コードの操作が容易であるため、迅速なアプリケーション開発が可能です。

以降では、MC9S08JM デバイス向けフリースケール USB スタックのソフトウェア構造を説明し、その設定方法について解説し、USB アプリケーションを迅速に開発するための手順を紹介します。本

### 目次

1	USB スタックの紹介	1
1.1	制限事項	2
2	USB モジュールの概要	2
3	スタックの構造	3
3.1	メイン・プログラムの構造	3
4	アプリケーションのカスタマイズ	6
4.1	エンドポイントの宣言	6
4.2	データの送信および受信	8
5	USB アプリケーションの起動	9
6	USB とペリフェラルの通信ブリッジ (SPI、IIC、および SCI)	9
6.1	アプリケーションの説明	9
6.2	アプリケーションの設定	9
6.3	ブリッジのプロトコル	10
6.4	スタックの用途	12

書を十分に理解するには、USB プロトコルの基本的な知識が必要です。

### 1.1 制限事項

本書およびスタック・ソフトウェアは、MC9S08JM の USB モジュールで情報を転送するための基本的な処理を対象とします。エンドポイントの設定以外のバルク、割込み、またはアイソクロナス転送などの上位レベルの転送については説明しません。インプリメンテーションの詳細については、MC9S08JM デバイス向け USB-Lite スタック (CMX 提供) および AN3565 (『USB and Using the USB-Lite by CMX Stack for the MC9S08JM Devices』) を参照してください。スタックの最新バージョンは、<http://www.freescale.com> で USB を検索してダウンロードできます。

## 2 USB モジュールの概要

JM60/32/16/8 デバイスには、USB のホストとデバイス間で 7 個の通信パイプを設定するための 7 個のエンドポイントがあります。

エンドポイント 0 は双方向 (IN および OUT) で、おもにコントロール転送に使用します。エンドポイント 0 はすべての USB デバイスに必要です。

エンドポイント 1 ~ 6 は単方向で、任意の期間で IN または OUT 方向のみの通信を行うように設定できます。

エンドポイント 5 および 6 は、ピンポン・バッファとも呼ばれるダブル・バッファ構造です。そのため、MCU が 1 つのバッファを操作しているときに、もう 1 つでシリアル・インタフェース・エンジン (SIE) の制御によるホストとの通信が可能です。このタイプのエンドポイントでは、MCU の待機時間が短縮されるために通信効率は向上します。

モジュールのサポートする通信のタイプは、コントロール、バルク、アイソクロナス、および割込みの各転送です。

USB モジュールの詳細については、AN3560 (『The USB device development with JM60/16』) を参照してください。

## 3 スタックの構造

### 3.1 メイン・プログラムの構造

この項では、フローチャートを用いながらソフトウェア構造について説明します。

#### 3.1.1 システム初期化

このコードは、USB モジュールを初期化して USB ホストとの接続の準備を整えます。以下のコンポーネントを初期化します。

- バス・クロック (24 MHz に設定する)
- プルアップ抵抗
- 電圧レギュレータ
- エンドポイント

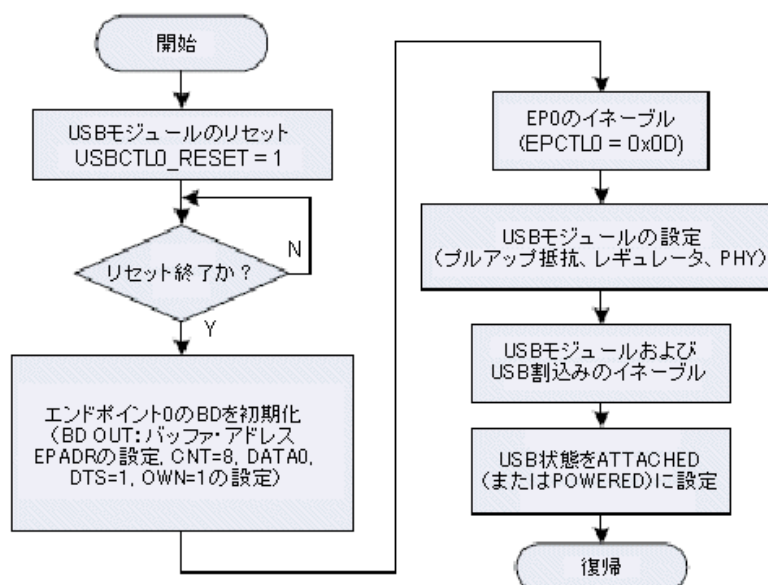


図 1. 初期化ルーチン

ファームウェアは、USBCTL0 レジスタの RESET ビットをセットして、USB モジュールをリセットし、すべてのレジスタをデフォルト値にリセットします。

ファームウェアは、USB RAM を初期化します。対象となるのは、特にエンドポイント 0 のバッファ・ディスクリプタ (BD) レジスタです。エンドポイント 0 の BD レジスタを OUT に設定します。EPADR レジスタは、USB RAM のエンドポイント・バッファを指示するように設定します。BC レジスタは、長さ 8 のデータ・パケットを受信するように 8 に設定します。ステータス/制御レジスタは、DATA0 パケットを受信するように設定します (DTS = 1, OWN = 1, DATA0/1 = 0)。

次に、エンドポイント 0 をイネーブルし、USB モジュールを設定してシステム・ハードウェア・デザインに従ってプルアップ抵抗、レギュレータ、および PHY (物理層) をイネーブルします。

最後に、USB モジュールおよび USB 割込みをイネーブルします。USB デバイスの状態は ATTACHED に設定します。ただし、USB が電源内蔵型なら USB デバイスの状態を POWERED に設定します。ファームウェアは、USB デバイスがバスに接続されたことを検出した後で ATTACHED に変更することができます。

### 3.1.2 メイン・ループ

メイン・ループは、初期化関数のコールおよび USB スタックのステータス変化のポーリングを行います。メイン・ループでは、受信フラグを監視して受信バッファがフルになるタイミングを知ることができます。

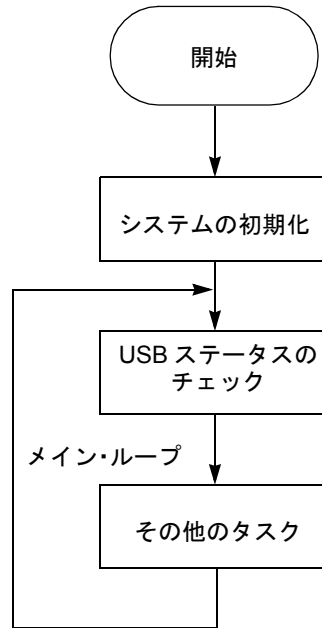


図 2. メイン・ループ

### 3.1.3 USB 割り込みサブルーチン

以下に、USB 割り込みサブルーチンのフローチャートを示します。USB ISR (割り込みサブルーチン) は、USB のイベントおよびステータスの変化を管理します (それ以外の情報はメイン・ループの USB ステータス・チェックで管理する)。USB 割り込みフラグは1つずつチェックします。ある割り込みフラグがセットされていてその割り込みがイネーブルなら、ISR は対応する関数にジャンプします。

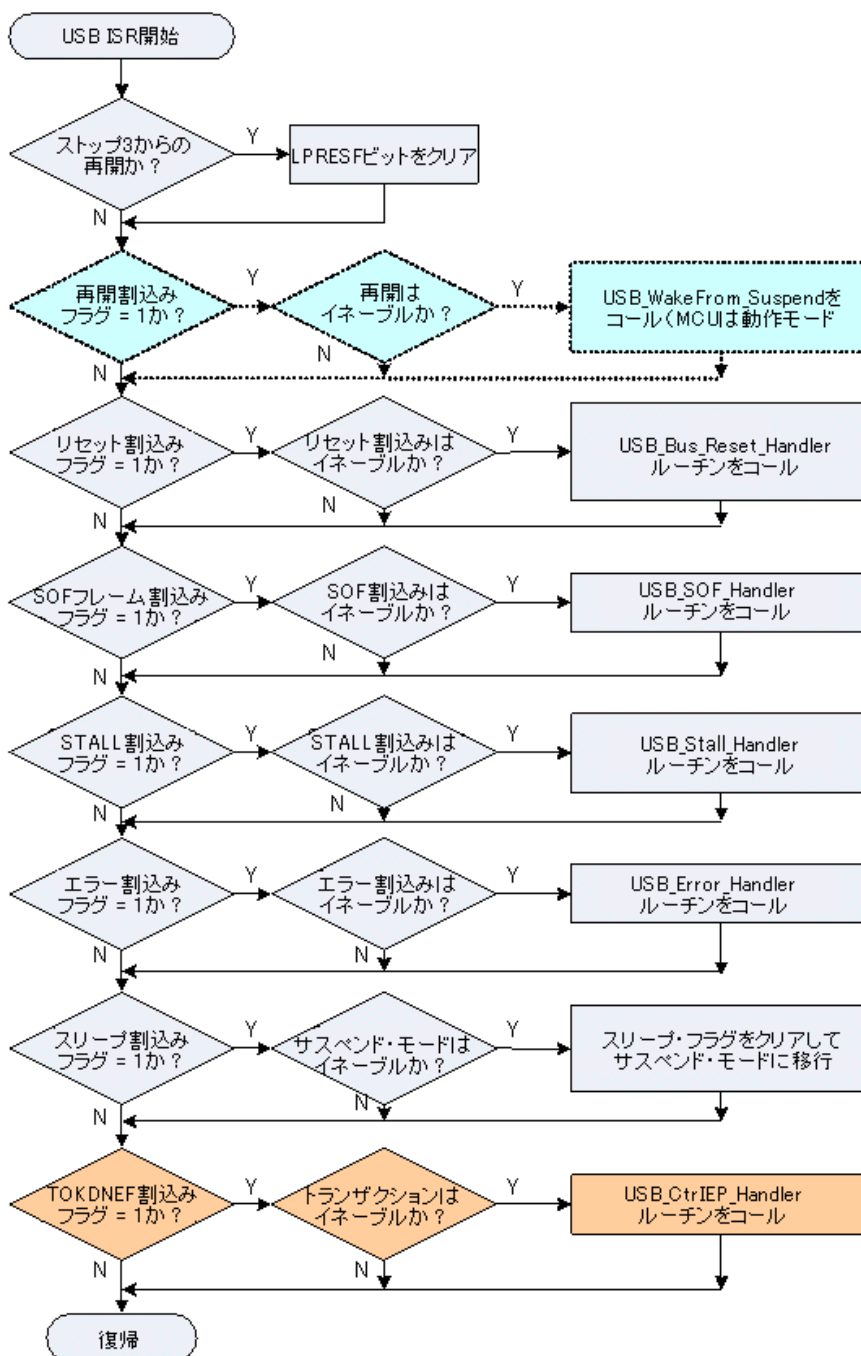


図 3. USB 割り込みサブルーチン

## 4 アプリケーションのカスタマイズ

以下の各項では、USB アプリケーションの実行に必要な処理について解説します。

### 4.1 エンドポイントの宣言

エンドポイント宣言は、アプリケーションのカスタマイズに不可欠の手順です。エンドポイント宣言では、IN または OUT、サポートする転送タイプ（アイソクロナス、割込み、バルク）、動作するエンドポイントと動作方法、各エンドポイントで保持するデータ量などを定義します。

#### 4.1.1 USB ディスクリプタ

すべての USB デバイスには、ホストに対して以下のデバイス情報を通知するための階層構造のディスクリプタが定められています。

- デバイスの種類
- デバイスの製造者
- サポートされる USB バージョン
- 設定方法の種類
- エンドポイント数およびそのタイプ
- その他

最も一般的な USB ディスクリプタは以下の通りです。

- デバイス・ディスクリプタ
- コンフィギュレーション・ディスクリプタ
- スtring・ディスクリプタ
- エンドポイントディスクリプタ
- インタフェース・ディスクリプタ

以上の各ディスクリプタは、デバイス動作に関する簡単な記述を行います。

ディスクリプタ・エントリは、Usb\_Description.c ファイルに格納されています。

デバイスをカスタマイズするには、目的のアプリケーションに応じてすべてのディスクリプタを変更する必要があります。

各テーブル・エントリの目的を示した Usb\_Description.c ファイルを参照してください。

#### 4.1.2 パイプ

USB 通信はパイプ（論理チャネル）がベースとなります。パイプとは、ホストからエンドポイントとして指定されるデバイスの論理エンティティまでの接続部分のことです。

各パイプ（エンドポイントとの接続）は、OUT 方向（ホストからデバイス）の場合と IN 方向（デバイスからホスト）の場合があります。IN および OUT のトランザクションはホスト側から見た方向であるため、OUT は常にホストからデバイスへのデータ転送で、IN は常にデバイスからホストへのデータ転送です。

#### 4.1.3 USB メモリのアクセス

USB RAM メモリには、USB SIE が共有する 256 バイトのバンクがあります。詳細については AN3560 を参照してください。バンクの先頭アドレスは 0x1860 で、このアドレスが SIE のアドレス 0x00 アドレスとなります。この共有メモリには、すべてのエンドポイントの情報が格納されます。エンドポイントの設定作業には、アプリケーションのすべてのエンドポイント・データをこのメモリ空間にライトする処理も含まれます。

この共有メモリへのアクセスはセマフォで管理されます。つまり、USB バッファへのアクセスは完全に制御されます。USB バッファの調整を行う場合は、各エンドポイントのユーザ・バッファ（共有メモリ外）を実装して SIE 処理時のこのロケーションへの不正アクセスを防ぐ必要があります。

USB モジュールは、USB エンドポイント通信を効率的に管理する目的でバッファ・ディスクリプタ・テーブル（BDT）を実装します。この BDT は、アクティブな各エンドポイントのステータスと制御の情報を提供します。

USB RAM の先頭の 32 エントリは、すべてのエンドポイントの BDT 値を格納しているか予約済みです。BDT テーブルは、次のトランザクション（IN または OUT）のサイズ、USB RAM のエンドポイント・バッファのオフセット、および同期化ビッ

ト (Data0 または Data1) を格納します。スタック・コードの BDT エントリは C 言語の構造体で、BDT へのアクセスはこれらの構造体を通して行います。

SIE は、16 バイト・モードでのみ共有 RAM へのアクセスが可能です。そのため、各エンドポイント・バッファはバッファ・サイズが 8 バイトの場合でも 16 バイトにアラインしている必要があります。たとえば、先頭の 2 つのバッファは通常はエンドポイント 0 (EP0) の IN および OUT 用で先頭の 32 バイトは BDT テーブル用であるため、EP0 の IN は 0x20 (SIE の場合) または 0x1880 (コアの場合) で開始し、EP0 の OUT は 0x30 (SIE) または 0x1890 (コア) で開始します。この規則は各ユーザ・エンドポイントにも適用されます。EP1 が 8 バイト長または 16 バイト長ならそのバッファは 0x40 (SIE) または 0x18A0 (コア) で開始し、EP2 は 0x50 (SIE) または 0x18B0 (コア) で開始します。

エンドポイント・バッファのベース・アドレス宣言は、右シフトを 2 回行った後で BDT オフセット・レジスタの EPx\_Set.Addr に格納します (x はエンドポイント番号)。

たとえば EP1 バッファが 0x40 (SIE) または 0x18A0 (コア) で開始する場合は、以下のようになります。

1. まず、エンドポイント・バッファを宣言します。

```
/* User Buffer for Endpoint 1 (User space) */
byte EP1_Buffer[EP_MAX_SIZE];
/* Endpoint 1 buffer (Shared space) */
byte EP1_BDT_Buffer[EP_MAX_SIZE] @0x18A0;
```

2. 次に、BDT 構造体でベース・アドレスを初期化します。

```
SIE address = EP1_Set.Addr = (0x40) >>2 = 0x10
/* Size of next transaction (this field is updated in each transaction)*/
EP1_Set.Cnt = 0x00;
/* Base address of Endpoint buffer */
EP1_Set.Addr = 0x10;
/* Semaphores for buffer access and sync signals */
EP1_Set.Stat._byte = _SIE|_DATA0|_DTS;
```

#### 4.1.4 エンドポイントのイネーブル

各エンドポイントをイネーブルするには、方向およびハンドシェイク設定をライトします。

```
EPCTLx= EP_IN|HSHK_EN; in case of IN Endpoint and handshake enabled
EPCTLx= EP_OUT|HSHK_EN; in case of OUT Endpoint and handshake enabled
```

#### 4.1.5 エnumレーション・プロセス

エnumレーション・プロセスとは、デバイスが USB バスに接続されたことを検出し、通信速度を (ハードウェアに基づいて) 判定した直後にホストが実行する、一連の処理のことです。ホストは、デバイス (本書の場合は MC9S08JM) にその設定を問い合わせる所定のアドレスを割り当てます。デバイス側は、すべてのエンドポイント、バッファ・サイズ、および転送タイプ (割込みやバルクなど) に関する情報をホスト側に転送します。これにより、ホストとデバイス間の予定された方式による通信がイネーブルされます。この段階では、すべての通信はエンドポイント 0 (制御エンドポイントとも呼ばれる) で実行されます。

スタック・コードは、エnumレーション・プロセスを管理する準備が整っているため、エンドポイントの設定が正しければ最終処理を実行します。処理の内容は、メイン・プログラムの無限ループ内で初期化ルーチンの Initialize\_USBModule() をコールして Check\_USBBus\_Status() をポーリングし、USB 通信ステータスの適切なリフレッシュを確実に実行することです。

スタックの USB 通信ステータスは、物理的接続の最初の検出からデバイスを使用するための状態の設定までを対象とするステート・マシンに準拠します。以下に、このステート・マシンの概念図を示します。

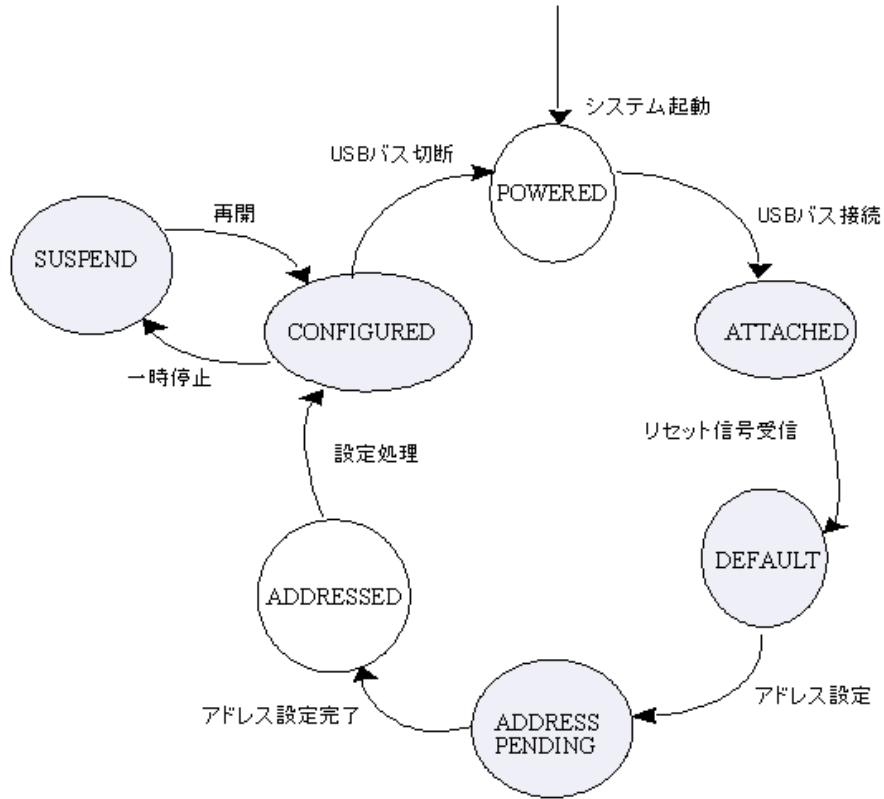


図 4. USB Communication State Machine (★訳なし)

前述したように、Check\_USBBus\_Status() 関数は割り込みドライブ・イベントが発生すると図に示すようにステータスをリフレッシュします。つまり、USB デバイスは CONFIGURED 状態のときに使用することができます。ステータス値は、Usb\_Device\_State 変数でリフレッシュされます。

## 4.2 データの送信および受信

前述したように、エンドポイントを正しく設定すれば送信と受信の関数を使用してホスト・デバイスと通信することができます。コードをコールする前に通信ステータスをチェックして、USB モジュールがホストでエニュメレートされていることを確認してください。USB 通信の現在の状態は、前述のように Usb\_Device\_State 変数に格納されます。割り込みが発生すると、Check\_USBBus\_Status() およびその他の関数がコールされてステータスが更新されます。データは、Usb\_Device\_State の内容が USB ホストが USB デバイス設定を認識したことを意味する CONFIGURED の場合のみ、送信が可能です。

### 4.2.1 データの送信

データの送信が可能なのは、IN に設定したエンドポイントだけです (名称はすべてホスト側から見た場合で、IN エンドポイントはホストにデータを送信する)。USB スタックがデータを送信する場合、データをエンドポイント・バッファにライトし、送信するデータ量を定義して、シリアル・インタフェース・エンジンに対して該当のエンドポイント・バッファの制御を許可します。次に IN トークンを受信するときは、データを送信してトランザクションが完了すると割り込みを生成します。以下に、この手順をまとめておきます。

- 目的のエンドポイント・バッファの EPx\_Buffer[n] にデータをライトする (x はエンドポイント番号、n はアレイサイズ)。
- IN エンドポイントの API 関数をコールする : void EndPoint\_IN(UINT8 u8Endp, UINT8 u8EPSize)。引数の u8Endp はエンドポイント番号で、USB\_User\_API.h ファイルは各エンドポイント番号の定義を格納し、u8EPSize は送信バイト数を表す。
  - 以下のマクロを使用して u8Endp 引数のエンドポイントから送信する。
    - エンドポイント 0 は EP0

- エンドポイント 1 は EP1
- エンドポイント 2 は EP2
- エンドポイント 3 は EP3
- エンドポイント 4 は EP4

以上の処理は、IN エンドポイントに設定した 1～4 の任意のエンドポイントで有効です。

#### 4.2.2 受信データのリード

以上で、割込みを使用するための USB スタックのライトは完了です。OUT に設定したエンドポイントに対してデータを送信すると、割込みが生成され、スタックはフラグをリフレッシュしてエンドポイントに準備の整ったデータがあることをユーザ API に通知します。CheckEndPointOUT(UINT8 u8Endp) 関数は、エンドポイントにデータがあるときは 1 を返し、データがないときは 0 を返します。以前と同じく、u8Endp はエンドポイント番号で、定義は USB\_User\_API.h に格納されます。

以上の処理は、OUT エンドポイントに設定した 1～4 の任意のエンドポイントで有効です。

## 5 USB アプリケーションの起動

1. エンドポイントを設定します。
  - a) 目的のエンドポイントの BDT テーブルをライトします。
  - b) トランザクション・タイプを指定します。
  - c) ベンダ ID および製品 ID を指定します。
  - d) Usb\_description.c にエンドポイント・ディスクリプタをライトします。エンドポイント・ディスクリプタの詳細については、USB 標準リビジョン 2.0 ([www.usb.org](http://www.usb.org)) を参照してください。
2. Usb\_Device\_State が CONFIGURED 状態であることを確認します。
3. USB のエニュメレート (設定) が完了したら、OUT エンドポイントによるデータのチェックまたは IN エンドポイントによるデータ送信を開始します。

## 6 USB とペリフェラルの通信ブリッジ (SPI、IIC、および SCI)

### 6.1 アプリケーションの説明

通信ブリッジは、1つの通信ポートから入力を取り込んでそれを別のポートで送信するアプリケーションです。この場合、ブリッジは USB のデータを MC9S08JM で選択される 3つのペリフェラルの 1つに接続するために使用されます。その3つとは、シリアル・ペリフェラル・インタフェース (SPI)、インタ・インテグレートド回路 (IIC)、およびシリアル通信インタフェース (SCI) です。アプリケーションの柔軟性を強化するため、選択したペリフェラルを設定するためのプロトコルが定義されています。この設定機能を簡単に利用できるように、MC9S08JM ペリフェラルの最も重要なコンポーネントだけが搭載されています (選択した SCI モジュールが Windows? の COM ポートのように動作させる設定オプションなど)。3つのすべてのモジュールは所定の値に初期設定されるため、必要な場合は設定を行わずにブリッジからデータを送信することが可能です。

Windows の WinUSB ドライバに接続される Windows GUI を利用すると、MC9S08JM ペリフェラルを設定してブリッジでデータの送信および受信を行うことが可能です。

### 6.2 アプリケーションの設定

この時点で、USB モジュールはベンダ仕様のデバイスとして動作するように設定されています。WinUSB ドライバは、PC ドライバによる通信が可能な所定のベンダ ID を使用します。この設定は Usb\_Descriptor.c ファイルにライトされます。使用するエンドポイントは 4 つです (USB プロトコルに必須のエンドポイント 0 を除く)。すべてのエンドポイントはバルク転送が基準で、2 つは 16 バイト長、2 つは 32 バイト長です。エンドポイント 1 は OUT エンドポイントで、ホストからのコマンドはこのパイプを通して MC9S08JM に送信されます。エンドポイント 2 は IN エンドポイントで、コマンドに対するステータス応答はこのエンドポイントからホストに送信されます。エンドポイント 3 およびエンドポイント 4 は 32 バイトのデータ・パイプです。エンドポイント 3 は、ホストからデータを受信して指定されたペリフェラルに送信します。エンドポイント 4 は、ペリフェラルのデータを取得してホストに送信します。

### 6.3 ブリッジのプロトコル

このプロトコルは、MC9S08JM ペリフェラルを設定します。これにより、アプリケーションの柔軟な動作が可能になります。コマンドおよびステータスの各パイプのデータ形式は以下の通りです。

$$|1 \text{ バイト : ペリフェラル ID} | 1 \text{ バイト : コマンド} | 14 \text{ バイト : データ} | \quad \text{方程式 1}$$

このアプリケーション・サンプルでは 3 つの通信ペリフェラルを使用するために、3 つの異なる ID を使用します。SPI は 0x03、IIC は 0x40、SCI は 0x05 です。

以下の表に、使用するコマンドの一覧を示します。

表 1. SPI - ホストからデバイス

パイプ	コマンド	説明	データ
エンド ポイント 3	0x00	SPI バスへのデータ転送	アプリケーション・データ (最大 31 バイト)
エンド ポイント 1	0x01	SPI ポートの送信ボーレートを定義する。ボーレートは、ボーレートを生成するデバイス性能に制限される。目的のボーレートが生成できない場合は対応する近似値で生成される。SPI では、ボーレートはシリアル・クロック周波数で表される。	2 バイト バイト 1 - ボーレート・プリスケール分周値。最大値は 7 で、プリスケール値はバイト 1 の値 + 1。 MC9S08JM データシートの「SPI ボーレートのプリスケール分周値」の表を参照。 バイト 2 - ボーレート分周値。MC9S08JM データシートの「SPI ボーレートの分周値」の表を参照。 計算式： ボーレート = バス・レート (24 MHz) / (バイト 1 * バイト 2)
エンド ポイント 1	0x02	8 ビットまたは 16 ビット転送	1 バイト 8 ビットは 0、16 ビットは 1
エンド ポイント 1	0x03	マスタまたはスレーブ	1 バイト マスタは 0、スレーブは 1
エンド ポイント 1	0x04	全二重または半二重 (SPI 半二重は 1 ワイヤによる送受信も意味する)	1 バイト 全二重は 0、半二重 / 1 ワイヤは 1
エンド ポイント 1	0x05	クロック・フェーズ	1 バイト データ転送の第 1 サイクルの途中でシリアル・クロックの先頭がエッジ発生する場合は 0 データ転送の第 1 サイクルの先頭でシリアル・クロックの先頭がエッジ発生する場合は 1
エンド ポイント 1	0x06	クロック極性	1 バイト アクティブ High の SPI クロックは 0 (アイドルは Low)、アクティブ Low の SPI クロックは 1 (アイドルは High)
エンド ポイント 1	0x07	シフト方向または最上位ビット優先転送	1 バイト 最上位ビット優先転送は 0、最下位ビット優先転送は 1

表 2. デバイスからホスト

パイプ	コマンド	説明	データ
エンド ポイント 4	0x00	SPI バスからのデータ転送	アプリケーション・データ (最大 31 バイト)
エンド ポイント 2	0x01	ボーレート ACK	0xFF - 設定許可 0x00 - 設定棄却
エンド ポイント 2	0x02	8 ビットまたは 16 ビット送信 ACK	0xFF - 設定許可 0x00 - 設定棄却
エンド ポイント 2	0x03	マスタまたはスレーブ設定 ACK	0xFF - 設定許可 0x00 - 設定棄却

表 2. デバイスからホスト

エンド ポイント 2	0x04	全二重または半二重設定 ACK	0xFF – 設定許可 0x00 – 設定棄却
エンド ポイント 2	0x05	クロック・フェーズ設定 ACK	0xFF – 設定許可 0x00 – 設定棄却
エンド ポイント 2	0x06	クロック極性設定 ACK	0xFF – 設定許可 0x00 – 設定棄却
エンド ポイント 2	0x07	シフト方向または最上位ビット優先転送設定 ACK	0xFF – 設定許可 0x00 – 設定棄却

表 3. IIC –ホストからデバイス

パイプ	コマンド	説明	データ
エンド ポイント 3	0x00	IIC バスへのデータ転送	アプリケーション・データ (最大 31 バイト)
エンド ポイント 1	0x01	ボーレート設定	2 バイト バイト 1 – ボーレート通倍値 0 → 1 倍 1 → 2 倍 2 → 4 倍 バイト 2 – ボーレート分周値。分周値は MC9S08JM データシートの「IIC の分周値および ホールド値」の表を参照。計算式： ボーレート = バス・レート (24 MHz) / (通倍値 * 分周値)
エンド ポイント 1	0x02	マスタ / スレーブ設定	スレーブは 0x00 マスタは 0x01
エンド ポイント 1	0x03	スレーブ・アドレス設定 – マスタの場合はデータ転送先のアドレス。スレーブの場合は IIC モジュールでスレーブ・アドレスとして設定されるアドレス。	バイト 1 – 7 ビット・アドレスは 0、10 ビット・アドレスは 1 バイト 2 – 7 ビット・アドレスまたは 10 ビット・アドレスの下位側 バイト 3 – 10 ビット・アドレスの上位側 3 ビット (バイトの右にシフトされる)

表 4. デバイスからホスト

パイプ	コマンド	説明	データ
エンド ポイント 4	0x00	IIC バスからのデータ転送	アプリケーション・データ (最大 31 バイト)
エンド ポイント 2	0x01	ボーレート設定 ACK	0xFF – 設定許可 0x00 – 設定棄却
エンド ポイント 2	0x02	マスタ / スレーブ設定 ACK	0xFF – 設定許可 0x00 – 設定棄却
エンド ポイント 2	0x03	スレーブ・アドレス設定 ACK	0xFF – 設定許可 0x00 – 設定棄却

表 5. SCI –ホストからデバイス

パイプ	コマンド	説明	データ
エンド ポイント 3	0x00	SCI ポートへのデータ転送	アプリケーション・データ (最大 31 バイト)
エンド ポイント 1	0x01	ボーレート設定	2 バイト バイト 1 – ボーレート分周値の上位側、最大値は 31 バイト 2 – ボーレート分周値の下位側 計算式： ボーレート = バス・レート (24 MHz) / (16 * ボーレート分周値)
エンド ポイント 1	0x03	パリティ設定	0x00 – なし 0x01 – 奇数 0x02 – 偶数

表 6. デバイスからホスト

パイプ	コマンド	説明	データ
エンド ポイント 4	0x00	SCI ポートからのデータ転送	アプリケーション・データ (最大 31 バイト)
エンド ポイント 2	0x01	ボーレート設定 ACK	0xFF – 設定許可 0x00 – 設定棄却
エンド ポイント 2	0x03	パリティ設定 ACK	0xFF – 設定許可 0x00 – 設定棄却

このアプリケーション・ノートで紹介する PC GUI は、設定を選択してデータの送受信を行います。

## 6.4 スタックの用途

前述したように、このアプリケーションは EndPoint\_IN(UINT8 u8Endp,UINT8 u8EPSize) および CheckEndPointOUT(UINT8 u8Endp) の各関数を使用してスタックを処理します。エンドポイントは、アプリケーション設定で説明したようにはベンダ仕様のエンドポイントとして使用されます。USB スタックのユーザ関数は USB\_User\_API.c ファイルに格納されます。これらの関数は、bridge.c ファイルとの通信を行う Protocol\_Handler.c ファイルで管理されます。アプリケーションはレイヤ構造であるため、移植や変更は簡単です。図 5 に、レイヤ・モデルを示します。

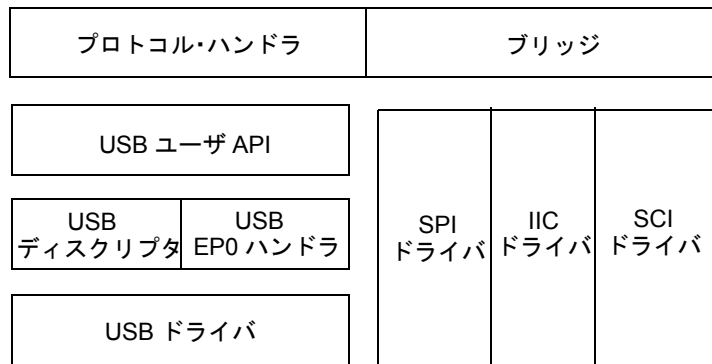


図 5. アプリケーション・レイヤ・モデル

図から分かるように、このモデルではユーザ・アプリケーションに簡単に機能を追加することができます。ブリッジおよびプロトコル・ハンドラと名付けられたレイヤは、USB コードを使用する他の任意のアプリケーション・レベル・コードです。SPI、IIC、または SCI の各レイヤは、アナログ・デジタル・コンバータ、PWM 出力、またはディスプレイなどの他のハードウェア・レベル・ドライバに変更することができます。



## How to Reach Us:

### Home Page:

www.freescale.com

### Web Support:

http://www.freescale.com/support

### USA/Europe or Locations Not Listed:

Freescale Semiconductor, Inc.  
Technical Information Center, EL516  
2100 East Elliot Road  
Tempe, Arizona 85284  
+1-800-521-6274 or +1-480-768-2130  
www.freescale.com/support

### Europe, Middle East, and Africa:

Freescale Halbleiter Deutschland GmbH  
Technical Information Center  
Schatzbogen 7  
81829 Muenchen, Germany  
+44 1296 380 456 (English)  
+46 8 52200080 (English)  
+49 89 92103 559 (German)  
+33 1 69 35 48 48 (French)  
www.freescale.com/support

### Japan:

Freescale Semiconductor Japan Ltd.  
Headquarters  
ARCO Tower 15F  
1-8-1, Shimo-Meguro, Meguro-ku,  
Tokyo 153-0064, Japan  
0120 191014 or +81 3 5437 9125  
support.japan@freescale.com

### Asia/Pacific:

Freescale Semiconductor Hong Kong Ltd.  
Technical Information Center  
2 Dai King Street  
Tai Po Industrial Estate  
Tai Po, N.T., Hong Kong  
+800 2666 8334  
support.asia@freescale.com

### For Literature Requests Only:

Freescale Semiconductor Literature Distribution Center  
P.O. Box 5405  
Denver, Colorado 80217  
1-800-441-2447 or 303-675-2140  
Fax: 303-675-2150  
LDCForFreescaleSemiconductor@hibbertgroup.com

本書に記載された情報は、システムおよびソフトウェア開発者がフリースケール製品を使用できるよう補助することのみを目的としています。本書に記載された情報に基づく集積回路の設計/製造に関する明示的または暗黙のライセンスを許諾するものではありません。

この技術資料は、英語版ドキュメントを翻訳したあと技術校正が入っておりませんので、参考資料としてお取扱ください。設計の際には、必ず弊社担当までご相談ください。

当社は、本書に記載した製品について、信頼性、機能または設計を改善するために予告なく変更を加える権限を保有しています。当社はここに記載した製品、回路の適用、使用に起因するいかなる責務をも負うものではなく、また、当社の特許権または第三者の権利に基づくライセンスを許諾するものではありません。仕様として記述される「標準 (Typical)」パラメータは各用途において変化する場合があり、実際の性能は長期間で変動する可能性があります。「標準」パラメータを含むすべての動作パラメータは、利用者側で技術担当者が使用環境に応じて適切な値に設定することが求められます。当社の製品は、外科的に人体に移植することを意図したシステムの構成部品として、または、他の生命維持を意図した用途に、または、当社の製品の不具合により人体に危害を加えたり死に至らしめるかもしれない状況が発生するような用途に使用するために、設計、意図または認可されているものではありません。購入者が万一このような意図または認可されていない用途のために当社の製品を購入あるいは使用する場合、購入者は、当社およびその役員、従業員、子会社、関連会社、代理店に対し、直接または間接を問わず、当該使用に関連した傷害や死についてのすべての申し立て（たとえ、当社が部品の設計や製造において不注意であったという主張であったとしても）から生ずるすべての請求、費用、損害、および相当の弁護士費用を補償し、被害が及ばないものとするものとします。



Freescale および Freescale のロゴマークは、フリースケール社の商標です。文中に記載されている他社の製品名、サービス名等は、それぞれ各社の商標です。

© Freescale Semiconductor, Inc. 2008. All rights reserved.

AN3564J

Rev. 0

07/2008

(原文 AN3041 Rev. 0, 11/2007)