テクニカルノート

MT25Q Serial NOR Flash Memory Software Device Drivers

はじめに

本テクニカル・ノートでは、フラッシュ ソフトウェア デバイス ドライバのインターフェース仕様に類似したインターフェースを使用する MT25Q シリアル NOR フラッシュ デバイス向けの C 言語ライブラリ ソース コードについて説明します。 MT25Q.c および MT25Q.h ファイルには、対応シリアル NOR フラッシュ デバイスにアクセス用のライブラリが含まれます。 Micron シリアル NOR フラッシュ デバイスおよび対応デバイスは、さまざまな構成と密度で提供されています。 有効パーツ番号は、Micron パーツカタログに記載されています (www.micron.com).

本テクニカルノートは、MT25Q データシートの情報を複製または変更するものではありません。 終始一貫してデータシートを参照しています。 説明を十分に理解するには、適切なデータシートのコピーが必要になります。

本テクニカルノートでは、付随のソースコードの変更に関する情報を提供します。 ソースコードは、可能な限りプラットフォーム非依存型として作成されており、コンパイルおよび起動にはユーザーによる変更を必要とします。

本テクニカルノートでは、個別の対象ハードウェアのソースコードの変更方法を説明します。 ソースコードには、使用方法やコードの作成手順に関する説明コメントが提供されています。

本ドキュメントで提供されているソフトウェアは、対象プラットフォームでテストされ、C および C++環境で使用可能です。 サイズも小さくどんな対象ハードウェアにも適用することができます。

ソフトウェア ドライバの使用

ソフトウェア ドライバは、C 言語でのアプリケーションコードの開発プロセスを簡潔化します。本ソフトウェア ドライバは、すべての新しいソフトウェア ドライバで展開しているフラッシュ デバイス ドライバ インターフェースを基にしています。 将来的なデバイスの変更での、アプリケーション環境におけるコード変更の必要がなくなります。

ソフトウェア ドライバ インターフェースにより、ユーザーは、特定のアプリケーションに必要な高レベルのコードの書き込みに集中することができます。 高レベルコードは、低レベルコードを呼び出してデバイスを評価します。 ユーザーは特別な指示シーケンスの詳細を考慮する必要はありません。 ソースコードはシンプルで維持にも手間はかかりません。 提供されているドライバで開発されたコードは、次の 3 層に仕分けられます。

- ハードウェア専用バス操作
- 低レベルコード
- ユーザーによって書かれた高レベルコード

MT25Q デバイスと通信するための、ハードウェア専用レジスタ READ および WRITE 操作を必要とする C 言語の低レベルコード これらの展開は、C コードが起動するマイクロプロセッサおよびマイクロコントローラーに依存し、またマイクロプロセッサのアドレススペースのメモリ場所に依存するため、これらの操作の展開は、ハードウェア-プラットフォーム依存型です。 ユーザーは、既存のハードウェア プラットフォームに適した C コードを書かなければなりません。 ガイド フレームワークは、Serialize.h および Serialize.c で提供されています。

ユーザーによって書かれた高レベルコードは、低レベルコードを呼び出してデバイスを評価します。 これにより、使用されているコードをシンプルにし、維持にも手間がかかりません。 ユーザーの高レベルコードは、その他のシリアル NOR フラッシュ デバイスにも簡単に適用することができます。

アプリケーションを開発する際は、

1. シンプルなプログラムを書いて、提供されている低レベルコードでテストし、対象ハードウェアおよびソフトウェア環境で意図している通りに動作することを検証します。
2. 高レベルコードを希望するアプリケーション向けに書きます。 アプリケーションが、低レベルコードを呼び出してシリアル NOR フラッシュ デバイスにアクセスします。
3. アプリーションのソースコードをすべてテストします。

ドライバのポート (ユーザー変更領域)

ソフトウェア ドライバに適用されたすべての変更は、ヘッダーファイルからご覧いただけます。 指定領域 (「ユーザー変更領域」という) には、次のセクションで説明するアイテムが含まれます。これらのアイテムは新しいハードウェアにソフトウェア ドライバをポートされなければなりません。

基本データタイプ

ソースコードの説明にあるように、以下の基本データタイプに対応するためにコンパイラの使用が必要かどうか確認して必要であれば変更します。

表 1: 基本データ Typedefs

```
typedef unsigned char uint8; (8 bits)
typedef char sint8; (8 bits)
typedef unsigned short uint16; (16 bits)
typedef unsigned short uint16; (16 bits)
typedef short_sint16; (16 bits)
typedef unsigned int uint32; (32 bits)
typedef int sint32; (32 bits)
typedef int sint32; (32 bits)
```

デバイス タイプ

READ ID コマンドに基づいて、起動時に正しいデバイスを自動的に検出します。 検出は Driver_Init()関数 実行中に発生します。 512Mb デバイスでは新規の定義は必要ありません。

タイムアウト

タイムアウトはコードループで展開されており、操作を終了に導き、そうでない場合は永続的に操作は終了しません。 ここでは 2 つの可能性が挙げられます。

- ANSI ライブラリ関数が time.h exist で宣言されている。 既存のコンパイラが time.h に対応する場合、定義文 TIME_H_EXISTS が有効にされなければなりません。 これにより、既存の評価ハードウェアの性能によるタイムアウト設定への変更を防止することができます。

```
#define TIME_H_EXISTS
```

- オプション COUNT_FOR_A_SECOND. 既存のコンプライアが time.h に対応していない場合は、定義文 TIME_H_EXISTS は使用できません。 この場合、COUNT_FOR_A_SECOND 値を定義して 1 秒の遅延を作成しなければなりません。 例えば、1 秒の遅延を発生させるために 100,000 回ループしなければならない場合、COUNT_FOR_A_SECOND の値は 100,000 でなければなりません。

```
#define COUNT_FOR_A_SECOND (chosen value)
```

注記: この遅延はハードウェア性能に依存しているため、ハードウェアが変更された際はその都度更新しなければなりません。

本ドライバは特定の構成でテストされており、およびその他の対象プラットフォームでは異なる性能データを持つ場合があります。 COUNT_FOR_A_SECOND の値を変更する必要があるかもしれません。 コードの早期タイムアウトを防止して正しい実行を実現する

には、ユーザーが正しい値を展開することに依存しています。 データシートにあるように、適したタイムアウト値は必要に応じて各関数で構成されています。

追加サブルーチン

```
#define VERBOSE
```

ソフトウェア ドライバでは、VERBOSE ステートメントの定義を使用して Flash- ErrStr() 関数を有効化し、デバイスからのリターン コードの文字列を生成します。

C ライブラリ関数

以下の表では、ユーザーに以下の関数のソースコードを提供しています。

表 2: 関数名と説明

関数	概要
Driver_Init	ドライバを初期化して、デバイスを自動的に検出します。 この関数は、その他すべての関数の前に、最初に呼び出します。 関数が、Flash_WrongType 値を返す場合は、デバイスが認識されていないことを意味します。 (サンプル コードを参照。)
DataProgram	拡張、デュアル、クアッド プロトコルを含む特定のコマンドを使用してデバイスのメモリに 1 ページ以上書き込みます。 FlashDataProgram を呼び出します。
DataRead	特定のメソッドを使用してデバイスからデータを読み取ります。 これらのメソッドには、次に挙げるすべての SPI プロトコル タイプが含まれます。 拡張 SPI、デュアルおよびクアッド プロトコル。 FlashDataRead を呼び出します。
FlashBulkErase	BULK ERASE コマンドを送信して全メモリを消去します。
FlashClearFlagStatusRegister	CLEAR FLAG STATUS REGISTER コマンドが、ステータス レジスタの消去ビットをクリアします。
FlashDataProgram	拡張、デュアル、クアッド プロトコルを含む特定のコマンドを使用してデバイスのメモリに 1 ページ以上書き込みます。
FlashDataRead	拡張、デュアル、クアッド プロトコルを含む特定のコマンドを使用してデバイスのデータを読み取ります。
FlashEnter4ByteAddressMode	ENTER 4 BYTE ADDRESS MODE コマンドが 4 バイト アドレス モードを有効化します。
FlashEnterDeepPowerDown	DEEP POWER-DOWN コマンドを送信して、デバイスを最低消費電力モード (ディープ パワー ダウン モード) に設定します。 このルーチンを呼び出した後、デバイスは RELEASE FROM DEEP POWER-DOWN コマンド以外のコマンドには応答しなくなります。 FlashReleaseDeepPowerDown() ルーチンを使用します。
FlashExit4ByteAddressMode	EXIT 4 BYTE ADDRESS MODE コマンドが 4 バイト アドレス モードを終了します。
FlashGenProgram	<PAGE SIZE> バイトまでのデータをデバイスにプログラミングします。
FlashLockSector	ステータス レジスタにあるブロック保護ビットを使用している複数のセクタをロックします。 データシートの保護されている領域サイズの表を参照してこのルーチンの使い方を決定してください。
FlashOTPProgram	デバイスの全 OTP 領域をプログラミングしてロックします。
FlashOTPRead	幾つかの OTP レジスタを読み取ります。
FlashPasswordProgram	8 バイト パスワードをデバイスにプログラムします。 これは 1 回きりの操作です。
FlashPasswordRead	8 バイト パスワードをリードバックします。 この操作はパスワード保護モードが設定された後は無視されます。
FlashPasswordUnlock	パスワード保護モードがセットされていると、パスワード (8 バイト) はグローバル凍結ビットを無効にして、不揮発性ロック ビットを変更できるようにしなくてはなりません。
FlashPermanentProtectionBitErase	すべての永続的なロック ビットを消去します。 グローバル凍結ビットは無効にしなければなりません。
FlashPermanentProtectionBitProgram	保護したいセクタの永続的な (不揮発性) ロック ビットを設定します。 グローバル凍結ビットは無効にしなければなりません。

表 2: 関数名と説明 (Continued)

関数	概要
FlashPermanentProtectionBitRead	永続的な (不揮発性) ロック ビットを読み込んで、どのブロックがロックされているか確認します。
FlashProgramAdvancedSecProt	パスワードとセクタ保護を有効または無効にするビットを書き込みます。
FlashProgramEraseResume	中断された消去またはプログラム操作を再開します。 消去およびプログラム操作の両方が中断されている場合は、プログラム操作が再開します。 プログラム操作が完了したら、再開コマンドを発行して消去操作を再開しなければなりません。
FlashProgramEraseSuspend	消去またはプログラム操作を中断します。
FlashReadAdvancedSecProt	セクタ保護レジスタを読み取ります。
FlashReadDeviceIdentification	READ IDENTIFICATION を送信して製造元識別子 (20h) とデバイス識別子を読み取ります。
FlashReadExtAddrReg	拡張アドレス レジスタ コンテンツを読み取ります。
FlashReadFlagStatusRegister	フラッグ ステータス レジスタのコンテンツを読み取ります。
FlashReadGlobalFreezeBit	グローバル凍結ビット値を読み取ります。
FlashReadNVConfigurationRegister	不揮発性構成レジスタを読み取ります。
FlashReadStatusRegister	READ STATUS REGISTER レジスタを送信してステータス レジスタを読み取ります。
FlashReadVolatileConfigurationRegister	READ VOLATILE CONFIGURATION REGISTER コマンドが揮発性構成レジスタを有効にして読み取りを可能にします。
FlashReadVEConfigReg	揮発性の強化された構成レジスタを読み取ります。
FlashReleaseDeepPowerDown	RELEASE FROM DEEP POWER-DOWN コマンドを送信して、デバイスをディープ パワー ダウン モードから解放します。
FlashSectorErase	セクタを消去します。
FlashSubSectorErase	4KB サブセクタを消去します。
FlashUnlockAllSector	ステータス レジスタ ビット BP[0:2]で保護されているすべてのセクタのロックを解除します。 ステータス レジスタには、ゼロ (0) が書き込まれます。
FlashWriteDisable	ライトイネーブル ラッチ (WEL) ビットをクリアします。
FlashWriteEnable	WEL ビットを設定します。
FlashWriteExtAddrReg	拡張アドレス レジスタの値を設定します。 最初の 2 ビットは、アドレス ビット A[24:25]を示します。 3 バイト アドレスを使用する際は、読み取り、書き込み、および消去操作には、これらのビットの影響が及びます。
FlashWriteGlobalFreezeBit	グローバル凍結ビットを設定またはクリアします。 このビットを設定すると、書き込み揮発性および不揮発性セクタ ロック ビットが無効になります。
FlashWriteNVConfigurationRegister	不揮発性構成レジスタに書き込みます。
FlashWriteStatusRegister	ステータス レジスタに書き込みます。
FlashWriteVolatileConfigurationRegister	揮発性構成レジスタに書き込みます。
FlashWriteVEConfigReg	揮発性の強化された構成レジスタに書き込みます。

サンプル コード

以下のソースコードは、ドライバの使用方法を示すサンプルです。 ドライバは、READ、PROGRAM、ERASE 操作を実行します。

表 3: クイック テスト サンプル コード

```
#include <stdio.h> #include "mt25q.h" #include "Serialize.h" int main(int argc, char
** argv) { FLASH_DEVICE_OBJECT fdo; /* flash device object */ ParameterType para; /*
parameters used for all operation */ ReturnType ret; /* return variable */ uint8
rbuffer[16]; /* read buffer */ uint8 wbuffer[16] = /* write buffer */ { 0xBE, 0xEF,
0xFE, 0xED, 0xBE, 0xEF, 0xFE, 0xED, 0xBE, 0xEF, 0xFE, 0xED, 0xBE, 0xEF, 0xFE,
0xED }; SpiDriverInit(); /* initialize your SPI interface */ ret = Driver_In-
it(&fdo); /* initialize the flash driver */ if (Flash_WrongType == ret)
{ printf("Sorry, no device detected.\n"); return -1; } fdo.GenOp.SectorErase(0); /*
erase first sector */ para.PageProgram.udAddr = 0; /* program 16 byte at address 0
*/ para.PageProgram.pArray = wbuffer; para.PageProgram.udNrOfElementsInArray = 16;
fdo.GenOp.DataProgram(PageProgram, &para); para.Read.udAddr = 0; /* read 16 byte at
address 0 */ para.Read.pArray = rbuffer; para.Read.udNrOfElementsToRead = 16;
fdo.GenOp.DataRead(Read, &para); /* now rbuffer contains /* written elements */
printf("The first device byte is: 0x%x\n", rbuffer[0]); return 0; }
```

ソフトウェア制約

本ドキュメントで記載されているソフトウェアは、MT25Q デバイスのすべての機能を展開するものではありません。 エラーが発生した際は、ソフトウェアはエラーメッセージを戻します。 この場合、ユーザーは、コマンドをもう一度実行するか、必要であればデバイスを交換します。

結果

MT25Q シリアル NOR フラッシュ デバイスは、組み込みシステムやその他のコンピューター システムに理想的な製品です。 マイクロプロセッサーのインターフェースを簡単に構築でき、C 言語を使用したシンプルなソフトウェア ドライバがこれらのデバイスを駆動します。

フラッシュ デバイス ドライバ 標準に対応するアプリケーションが、同一のインターフェースを持つフラッシュ デバイスをコードを変更することなく展開します。 新しいデバイスをコントロールする際に必要となるのは、新しいソフトウェア ドライバの際コンパイルのみです。

デバイス ドライバ インターフェースが、変更可能な構成、コンパイラー非依存型のデータタイプ、複数のフラッシュ デバイスへの一意のアクセスモードを実現します。

改訂履歴

改訂 A – 9/13

- 初期リリース

8000 S. Federal Way, P.O. Box 6, Boise, ID 83707-0006, Tel: 208-368-4000
www.micron.com/products/support Sales inquiries: 800-932-4992