お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注 1. 本資料において使用されている「当社」とは、ルネサスエレクトロニクス株式会社およびルネサスエレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

H8/300L Super Low Power (SLP)シリーズ

SCI を使用した SPI™ EEPROM I/F 実現方法

内容

本アプリケーションノートは、SLP シリーズのシリアルコミュニケーションインタフェース（SCI3）を用いて SPI をサポートする方法と、SLP と SPI EEPROM との通信の詳細を説明し、システム設計者の習熟期間を短縮することを目的としています。

はじめに

シリアル周辺インタフェース（SPI）は、マスタとスレーブあるいは周辺デバイスとの 8 ビットの全二重同期シリアル通信のチャネルを提供するものです。SPI は、シリアルコミュニケーションインタフェース（SCI）を用いてホスト CPU から設定できます。

H8/300L Super Low Power (SLP)シリーズのシリアルコミュニケーションインタフェース 3（SCI3）のハードウェアは、SPI シリアル EEPROM と 3 本の信号線を用いたシンプルな接続を提供します。

動作確認デバイス

H8/300L Super Low Power (SLP)シリーズ – H8/38024

目次

1. SPI™インタフェースの概要

SPI には 2 本の制御信号（CS と SCK1）および 2 本のデータ信号（SI1 と SO1）が必要です。CS（チップセレクト）により、対応する周辺デバイスが選択されます。この信号は、ほとんどの場合アクティブロー信号です。未選択の状態では、SO1 信号は Hi-Z で非活性状態です。

マスタは、どの周辺デバイスと通信を行なうかを決定します。クロック信号 SCK1 は、そのデバイスの選択／未選択状態に関わらず供給されます。クロックにより、データ通信の同期をとります。

（SPI の一般的な特長については、SPI および I2C に関するアプリケーションノートで説明します）

1.1 SPI EEPROM の制御

プログラム例では、STMicroelectronics 社の高速クロックの 64K ビットシリアル SPR バス EEPROM M95640 を用いました。M95640 は、シリアル EEPROM 制御用の SPI プロトコルをサポートしています。

命令セットを以下に示します。

命令	説明	命令フォーマット
WREN	**ライトイネーブル**	0000 0110 (0x06)
WRDI	**ライトディスエーブル**	0000 0100 (0x40)
RDSR	**ステータスレジスタのリード**	0000 0101 (0x05)
WRSR	**ステータスレジスタのライト**	0000 0001 (0x01)
READ	**メモリアレイからのリード**	0000 0011 (0x03)
WRITE	**メモリアレイへのライト**	0000 0010 (0x02)

【注】 本アプリケーションノートに記載のプログラム例は、上記 6 コマンドのうち 4 コマンド（WREN、RDSR、READ、WRITE）のみを使用しています。

制御コマンド WREN および WRDI は 1 バイトコマンドで、追加のデータを送受信する必要はありません。

制御コマンド RDSR および WRSR は、動作を完了するには 2 番目のバイトを送信する必要があります。

READ および WRITE コマンドでは、複数バイトの転送が必要です。両コマンドとも、以下のプロトコルになります。

<命令> <アドレス> <N-バイトデータ>

<アドレス>フィールドは、データのリード／ライトを開始する 2 バイトの先頭アドレスを示します。その後、ソース／デスティネーションアドレスを EEPROM が自動的にインクリメントしながら、データが連続的にリード／ライトされます。このデバイスでは、各ライト動作が完了すると自動的に WREN が解除されることに注意してください。

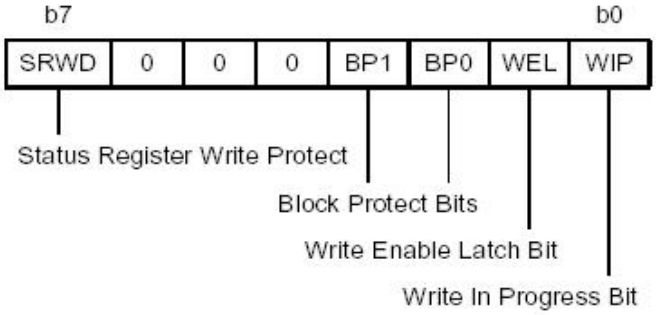
1.2 ビット順の反転

SPI シリアル EEPROM の通信プロトコルでは、データは最上位ビット（MSB）から転送を始めることになっています。同期モードでは、SLP シリーズの SCI インタフェースは最下位ビット（LSB）からデータをシフトイン／アウトします。

この問題を解決するため、ルックアップテーブルを設け、SCI を通して転送するデータのビット順を反転するようにしました。

1.3 ステータスレジスタ

ST M95640 のステータスレジスタには、様々なステータスビットとコントロールビットがあり、以下に示すように、命令実行に応じて読み出されたり該当する値にセットされたりします。



WIP ビット ：Write In Progress（WIP）ビットは、ライトサイクルまたはステータスレジスタのライトサイクルによりメモリがビジー状態であることを示します。

WEL ビット ：Write Enable Latch（WEL）ビットは、内部でライトイネーブルラッチ状態であることを示します。

BP1, BP0 ビット：Block Protect（BP1, BP0）ビットは不揮発性です。ライト命令をソフトウェアで禁止するエリアのサイズを指定します。

SRWD ビット ：Status Register Write Disable（SRWD）ビットは、ライトプロテクト（W）信号と連動して動作します。本ビットとライトプロテクト（W）信号により、デバイスをハードウェアプロテクトモードにすることができます。このモードでは、ステータスレジスタの不揮発性ビット（SRWD、BP1、BP0）がリードオンリー（ライト禁止）ビットになります。

1.4 SPI の実現

SPI インタフェースは以下の点を考慮して設計されました。

- ライト動作ごとに WREN をセットしなくてはならない。
- EEPROM への、または、EEPROM からのリード／ライトが全バイト完了するまで、CS 信号はハイレベルにしない。
- CS 信号がハイレベルになったときに動作が完了する。
- ライトデータ転送が完了してから最大 10ms（最悪の場合）のあいだ、EEPROM の内部はライト動作状態が続く。したがって、ライト転送終了後しばらくは EEPROM は“ビジー”に見える。
- “ビジー”状態は、ステータスレジスタから 0xFF が読み出されることで検出される。
- リード／ライト命令のフォーマットは基本的に同じで、データ信号線だけが異なる。したがって、両動作のプログラムは、同じような方法で作成できる。
- コントローラが SI 信号を駆動すると、SO 信号はハイレベルを保持する。
- EEPROM が SO 信号を駆動するとき、EEPROM は SI 信号をサンプリングしない。

【注】 SPI に関する定番の公式仕様がないため、必ず使用する部品の仕様をデータシートで入念に調べてください。

1.5 CS 信号の制御と実装

アプリケーションは、CS がどの時点で発生したかを正確に特定できなければならないため、CS 信号を注意深く制御し正しく動作させる必要があります。SLP の同期式シリアルポートを用いる場合、次の 2 つの方法があります。

i) 送信シフトレジスタ（TSR）から最後のビットがシフトアウトされ、送信データレジスタ（TDR）に送信待ちのデータがなくなったことを送信終了割り込みイネーブル（TEIE）を用いて CPU に知らせます。
ii) レシーバがイネーブル状態であるなら、受信と送信は同じ SCK 信号を用いるので、データは送信と同期して受信されます。したがって、アプリケーションプログラムは、どちらか一方が受信完了することにより、1 バイトのデータが何時、送信完了したかを把握することができます。

SPI の大半の動作は双方向であり、2 番目が望ましい方法です。というのは、プログラム／データの最小のオーバーヘッドにより、自動的に双方向データ交換と CS 信号の制御の両方を実現できるからです。TEIE を用いた方法では、割り込みハンドラが余計に必要になります。

1.6 リード／ライト制御と実装

データを読み出す場合、対応する命令とアドレスを EEPROM に送信して必要なデータを読み出します。必要な量のデータが読み出されるまで、ダミーデータバイトを送信します。ダミーデータはランダムな値か、0xFF に初期化された値（本アプリケーションノートのプログラム例のように）を用います。受信割り込みハンドラを用いて、データを読み出し、CS 信号を制御します。カウント値が受信する残りバイトの数となるように更新し、この値がゼロになると CS 信号をハイレベルにしてそれ以降の受信割り込みを禁止します。

データを書き込む場合、対応する命令とアドレスを送信した後に、ライトデータを送信します。ここでも、受信割り込みハンドラはカウント値をデクリメントし、必要なバイト数のデータを書き込むと CS 信号を制御します。この場合、受信したデータには意味がありません（実際は、すべて 0xFF の値です）。

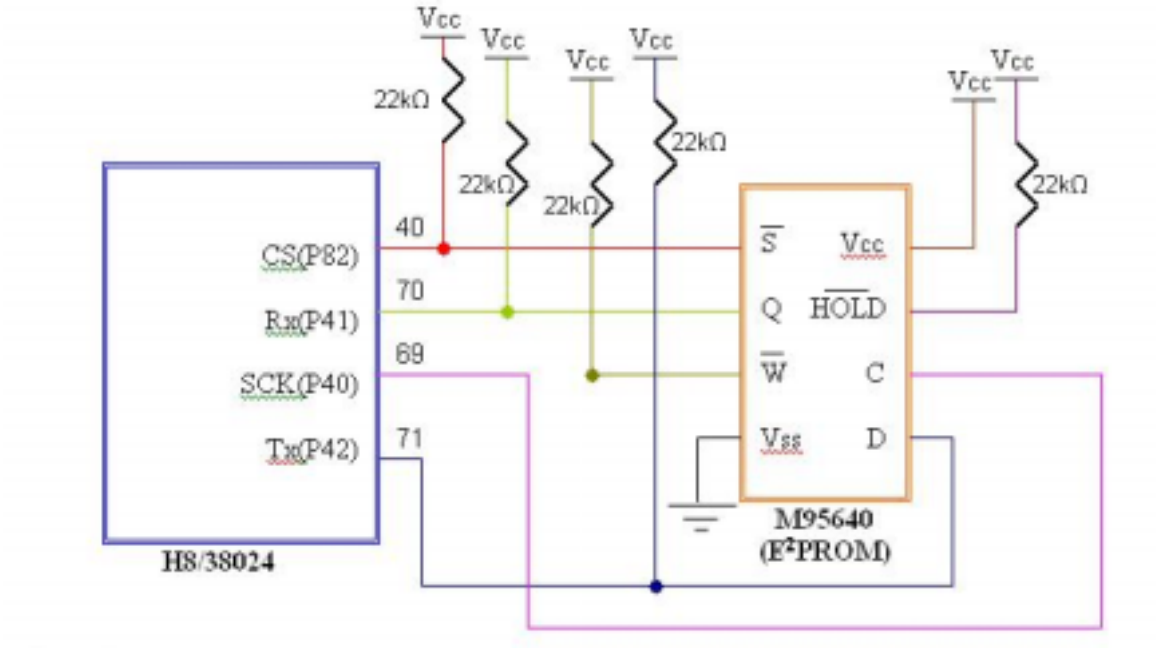
1.7 SPI バスへの接続

ST M95640 は SPI プロトコルに完全に対応しています。すべての命令、アドレス、入力データバイトは、最上位ビットからデバイスにシフトインされます。チップセレクト（S）がローレベルになった後の最初のシリアルクロック（C）の立ち上がりエッジで、シリアルデータ入力（D）がサンプリングされます。

すべての出力データバイトは、最上位ビットからデバイスからシフトアウトされます。命令（メモリアレイからのリードやステータスレジスタのリード命令など）がデバイスに送信された後の最初のシリアルクロック（C）の立ち下がりエッジで、シリアルデータ出力（Q）がラッチされます。

【注】@ 本セクションでの説明は、ST M95640 を用いた場合のもので、デバイスに依存します。ユーザの使用する EEPROM とは異なる場合があります。正しい情報は、該当するデータシートや仕様書を参照してください。

2. ハードウェア設計



*Signal Names

S	Chip Select	Vcc	Supply Voltage
Q	Serial Data Output	HOLD	Hold
W	Write Protected	C	Serial Clock
Vss	Ground	D	Serial Data Input

* The Write Protected (W) and Hold (HOLD) signals should be driven, High or Low as appropriate.

3. 関数の概要

spi.c で使用する関数：

- void SPISetup (void)
- uchar ReadSR (void)
- void SetWREN (void)
- void Write (ushort, ushort, uchar *)
- void WriteByte (ushort, uchar)
- void Read (ushort, ushort, uchar *)
- void main (void)

intprg.c (__interrupt(vect=18)で使用する関数：

- void INT_SCI3(void)

`void SPISetup (void)`

本ルーチンは以下の初期化を行ないます。

i) シリアルコミュニケーションインタフェース（SCI）
   a) ボーレート（BRR）
   b) データ転送フォーマット（SMR）
   c) シリアルコントロールの設定（SCR）
   d) シリアルステータス（SSR）
   * SCI 設定の詳細については、ハードウェアマニュアルを参照してください。

ii) CS 信号（PDR8）により EEPROM を初期化

`uchar ReadSR (void)`

ステータスレジスタのリード命令（RDSR）により、ステータスレジスタを読み出します。CS 信号をローレベルにして EEPROM を初期化し、レジスタの読み出しが完了したらハイレベルにもどす必要があります。

`void SetWREN (void)`

ライト動作の前に、必ず本ルーチンを呼び出してください。本ルーチンは、ライトイネーブル命令（WREN を設定する）を発行します。ライト動作が完了し、CS 信号がハイレベルになると、EEPROM が自動的に WREN ビットをリセットしますので、書き込みの前には必ずこの関数が必要です。

`void WriteByte (ushort, uchar)`

引数： データを書き込むアドレス (ushort) 書き込むデータバイト数 (uchar)

WriteByte ルーチンはライトコマンドを発行し、このあとに引数としてアドレスを渡します。次に、データ引数が EEPROM に書きこまれます。このルーチンでは、単一バイトのみを書き込むことができます。RX 割り込みを用いて完了すると同時に、CS 信号をセットします。

`void Write (ushort, ushort, uchar *)`

引数： データを書き込むアドレス (ushort) 書き込むデータバイト数 (uchar) データが格納されているバッファのアドレス (uchar *)

Write 関数は WRITE 命令を送信し、このあとにビット順を反転した 2 バイトアドレスを EEPROM に渡します。次に、ポーリングを用いたループで、第 3 引数（*DataPtr）で示されたバッファから残りのデータを転送します。

`void Read (ushort, ushort, uchar *)`

引数： データを書き込むアドレス (ushort) 書き込むデータバイト数 (uchar) 　データを格納するバッファのアドレス (uchar *)

Read 関数は Read 命令を送信し、このあとにビット順を反転した 2 バイトアドレスを EEPROM に渡します。次に、ポーリングを用いたループで、残りのデータを読み出します。受信割り込みを用いて転送済みバイト数を更新し、完了すると CS 信号を High にします。受信したデータは第 3 引数（*Buffer）で示されたバッファに格納します。

`void main()`

main 関数では、1 バイトを EEPROM のアドレスに書き込んでから読み出すことにより、EEPROM の制御と通信機能のデモとテストを行ないます。次に、一連のデータを EEPROM に書き込んでから読み出します。

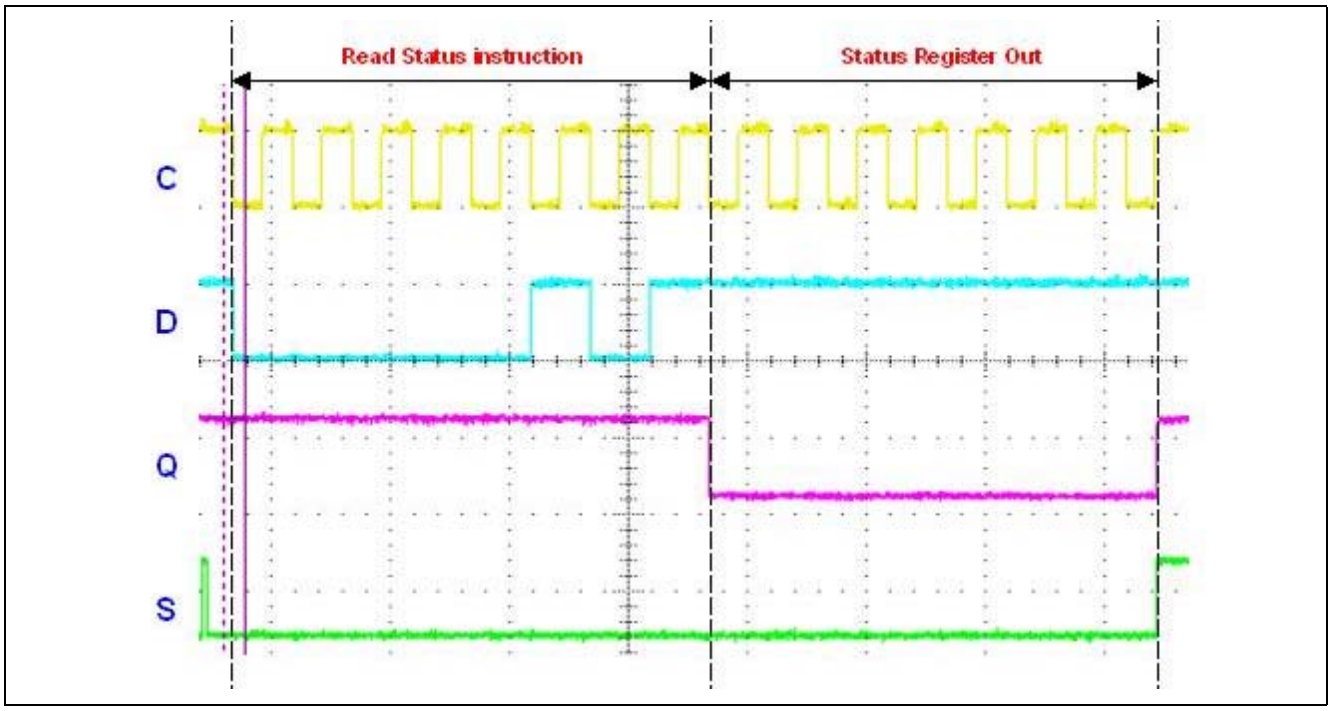
`void INT_SCI3(void)`

受信割り込みを用いて、TSR レジスタから出力された（送信が完了した）バイト数を監視します。リードサイクルが完全に終了した正確なタイミングは、CS 信号がハイレベルになったときです。受信データは必ず RxBuffer に格納され、不要なデータを保存することによるオーバヘッドを最小にしています。また、余計なテストを省く意味もあります。本関数は、ベクタ 18 の割り込みプログラムに挿入してください。

4. プログラムの解析

4.1 ステータスリード

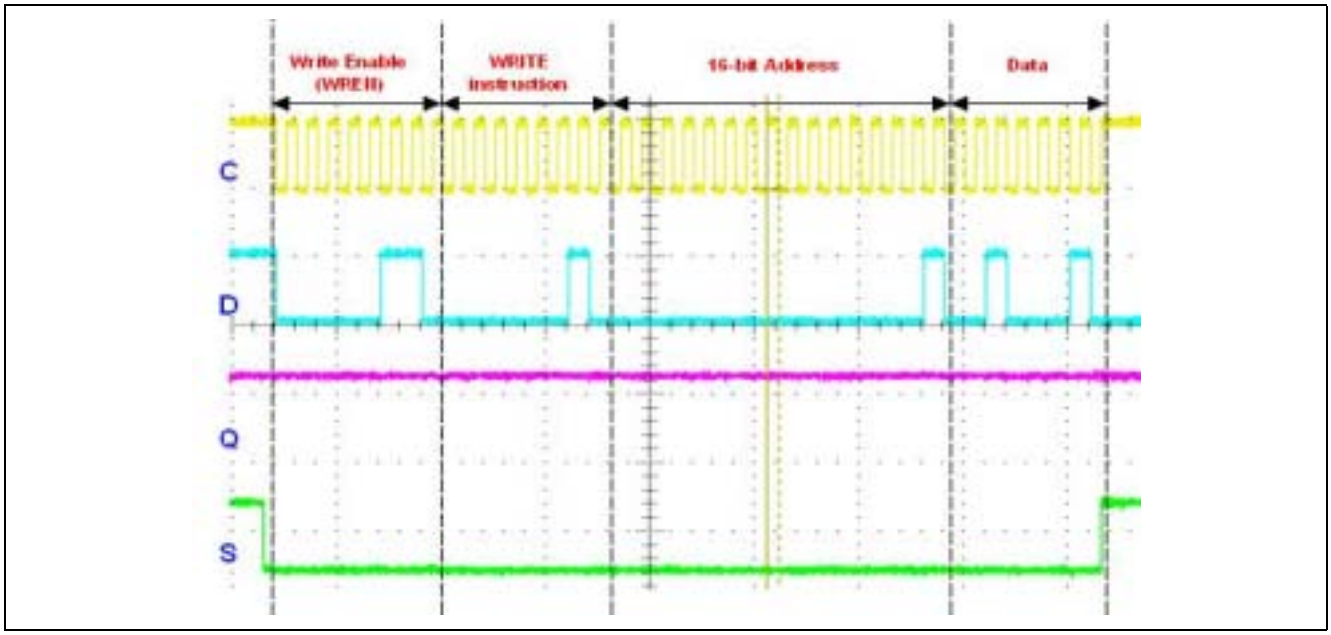
下図にステータスのリード動作を示します。



ステータスのリード動作時間は平均*0.34ms です。

4.2 バイトライト

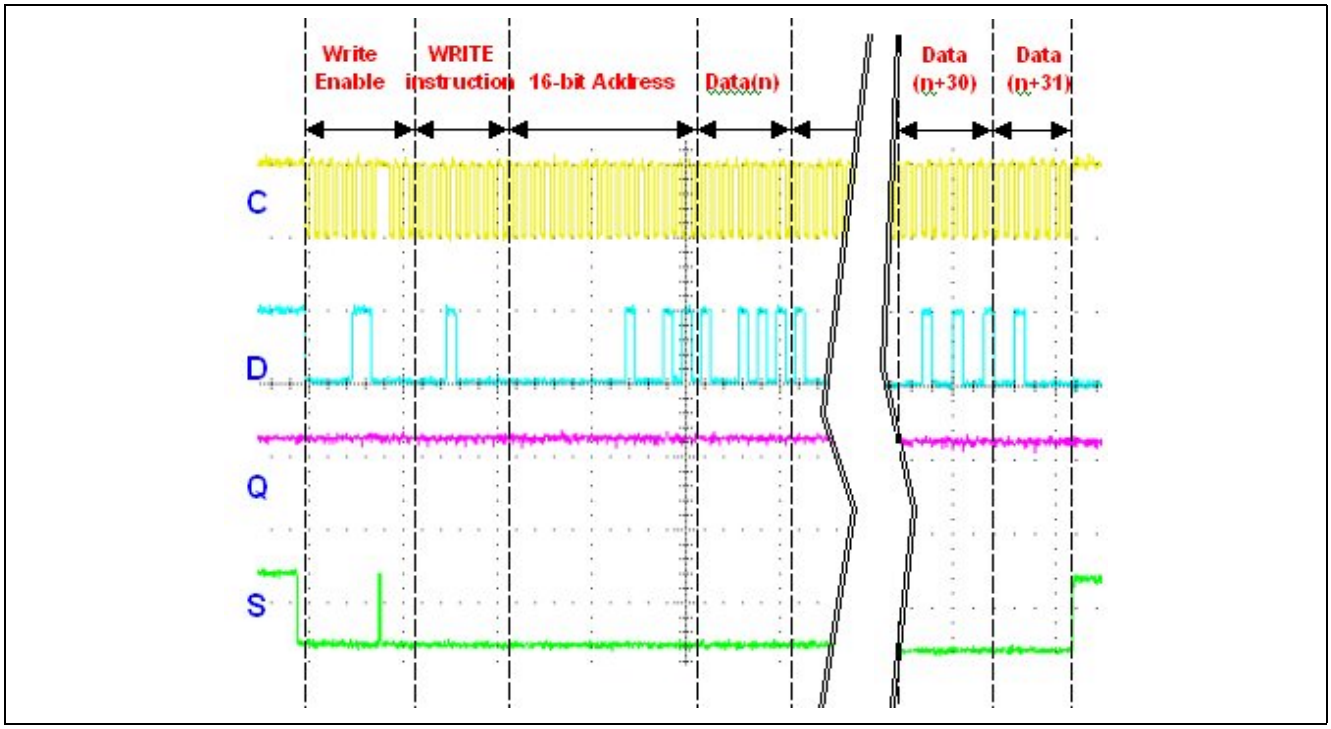
下図に 1 バイトデータのライト動作を示します。



1 バイトデータのライト動作時間は平均*0.85ms です。

4.3 ページライト

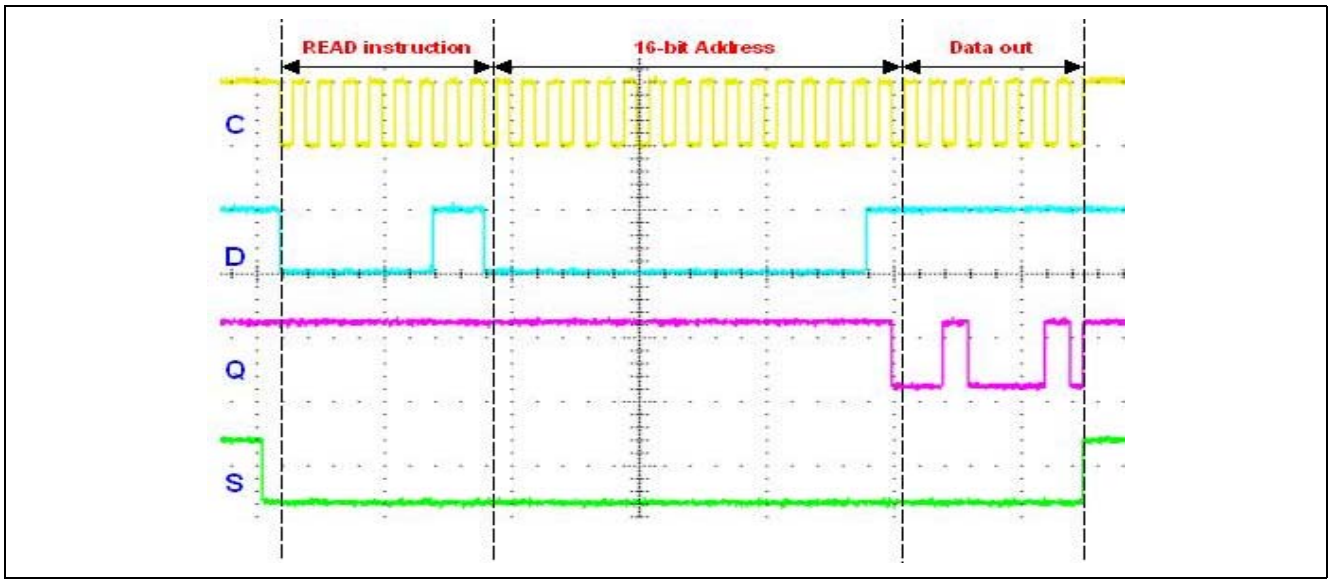
下図に 1 ページ（32 バイト）データのライト動作を示します。



1 ページデータのライト動作時間は平均*5.5ms です。

4.4 バイトリード

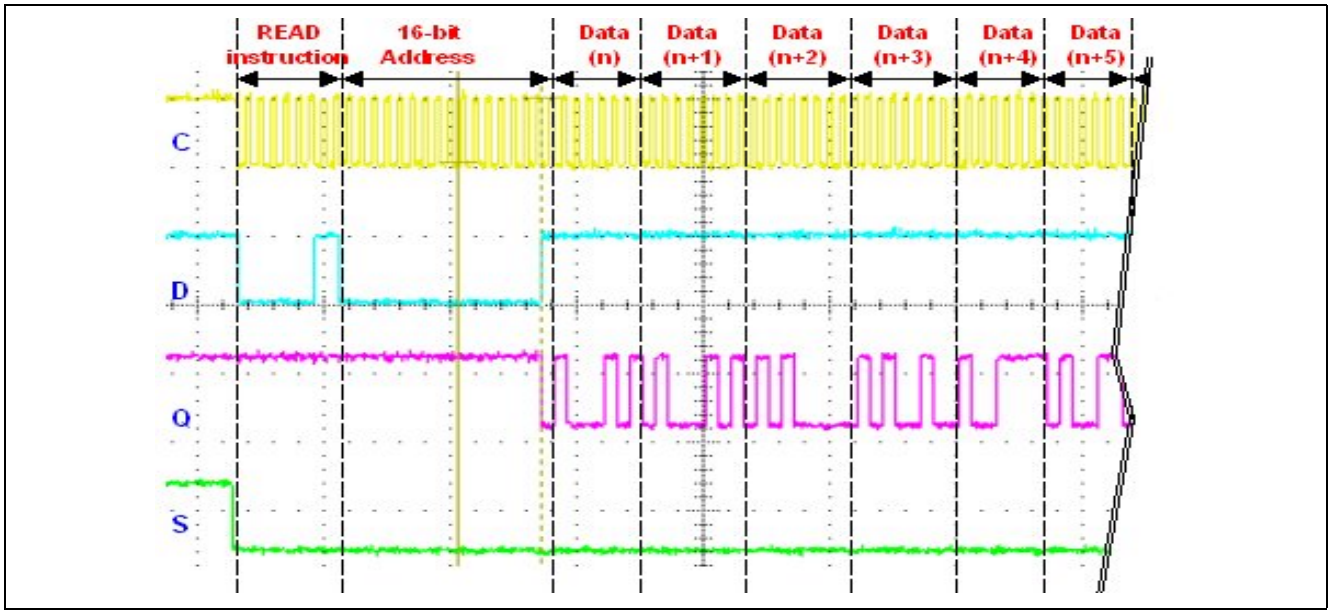
下図に 1 バイトデータのリード動作を示します。



1 バイトデータのリード動作時間は平均*0.53ms です。

4.5 ページリード

下図に 1 ページ（32 バイト）データのリード動作を示します。



1 ページ（32 バイト）データのリード動作時間は平均*4.6ms です。

【注】*上記の平均時間は、すべて 10MHz クロック、1/2 システムクロック分周比を用いた場合の計測値です。

5. プログラム例

```
/*****************************************************************/
/*                                                               */
/*  FILE        :SPI.c                                           */
/*  DATE        :Fri, Dec 20, 2002                               */
/*  DESCRIPTION :Main Program                                    */
/*  CPU TYPE    :H8/38024F                                       */
/*                                                               */
/*  This file is generated by Hitachi Project Generator (Ver.2.1).*/
/*                                                               */
/*****************************************************************/
#include "iodefine.h"
#include "applicationdemo.h"
#include <machine.h>
#include <stdio.h>


//Function Prototypes
void moreByte_access(void);
void byte_access(void);
void delay(void);
void SPISetup (void);
uchar ReadSR (void);
void SetWREN (void);
void Write (ushort, ushort, uchar *);
void WriteByte (ushort, uchar);
void Read (ushort, ushort, uchar *);


//Global Data
ushort RxCount;             //No. of bytes to be received
ushort TxCount;             //No. of bytes to be transmitted
uchar TxBuffer[MAXRXCOUNT]; //Buffer used to store transmit data
uchar *RxBuffer_ptr;        //buffer for received data
uchar *TxBuffer_ptr;        //pointer to data for transmission
uchar RxBuffer[MAXTXCOUNT];
int i=0;
```

```
const char table[256] = {
0x00,0x80,0x40,0xC0,0x20,0xA0,0x60,0xE0,
0x10,0x90,0x50,0xD0,0x30,0xB0,0x70,0xF0,
0x08,0x88,0x48,0xC8,0x28,0xA8,0x68,0xE8,
0x18,0x98,0x58,0xD8,0x38,0xB8,0x78,0xF8,
0x04,0x84,0x44,0xC4,0x24,0xA4,0x64,0xE4,
0x14,0x94,0x54,0xD4,0x34,0xB4,0x74,0xF4,
0x0C,0x8C,0x4C,0xCC,0x2C,0xAC,0x6C,0xEC,
0x1C,0x9C,0x5C,0xDC,0x3C,0xBC,0x7C,0xFC,
0x02,0x82,0x42,0xC2,0x22,0xA2,0x62,0xE2,
0x12,0x92,0x52,0xD2,0x32,0xB2,0x72,0xF2,
0x0A,0x8A,0x4A,0xCA,0x2A,0xAA,0x6A,0xEA,
0x1A,0x9A,0x5A,0xDA,0x3A,0xBA,0x7A,0xFA,
0x06,0x86,0x46,0xC6,0x26,0xA6,0x66,0xE6,
0x16,0x96,0x56,0xD6,0x36,0xB6,0x76,0xF6,
0x0E,0x8E,0x4E,0xCE,0x2E,0xAE,0x6E,0xEE,
0x1E,0x9E,0x5E,0xDE,0x3E,0xBE,0x7E,0xFE,
0x01,0x81,0x41,0xC1,0x21,0xA1,0x61,0xE1,
0x11,0x91,0x51,0xD1,0x31,0xB1,0x71,0xF1,
0x09,0x89,0x49,0xC9,0x29,0xA9,0x69,0xE9,
0x19,0x99,0x59,0xD9,0x39,0xB9,0x79,0xF9,
0x05,0x85,0x45,0xC5,0x25,0xA5,0x65,0xE5,
0x15,0x95,0x55,0xD5,0x35,0xB5,0x75,0xF5,
0x0D,0x8D,0x4D,0xCD,0x2D,0xAD,0x6D,0xED,
0x1D,0x9D,0x5D,0xDD,0x3D,0xBD,0x7D,0xFD,
0x03,0x83,0x43,0xC3,0x23,0xA3,0x63,0xE3,
0x13,0x93,0x53,0xD3,0x33,0xB3,0x73,0xF3,
0x0B,0x8B,0x4B,0xCB,0x2B,0xAB,0x6B,0xEB,
0x1B,0x9B,0x5B,0xDB,0x3B,0xBB,0x7B,0xFB,
0x07,0x87,0x47,0xC7,0x27,0xA7,0x67,0xE7,
0x17,0x97,0x57,0xD7,0x37,0xB7,0x77,0xF7,
0x0F,0x8F,0x4F,0xCF,0x2F,0xAF,0x6F,0xEF,
0x1F,0x9F,0x5F,0xDF,0x3F,0xBF,0x7F,0xFF
};

uchar message[] = "EEPROM SPI Access ";


int main (void)
{
  SPISetup();

  byte_access();

  moreByte_access();

 while(1);

return (0);
}
```

```
void byte_access(void)
{
while(ReadSR() == 0xFF);     //check that we are talking
WriteByte(0x0001,0x33);      //try writing a byte

while(ReadSR() == 0xFF);     //check that we are talking
while(ReadSR()&01 == 0x01);  //wait until write completed(WIP=0)
Read(0x0001,1,&RxBuffer[0]); //read it back
}


void moreByte_access(void)
{
while(ReadSR() == 0xFF);      //check that we are talking
Write (0x0000,15,&message[0]);//write a 15 byte message

while (ReadSR() == 0xFF);     //wait on the internal operation
while(ReadSR()&01 == 0x01);   //wait until write completed(WIP=0)
Read(0x0000,15,&RxBuffer[0]); //read back the message
}


void SPISetup (void)
{
int i;

//the Null buffer could be a constant table to save RAM

for (i=0;i<MAXRXCOUNT;i++)
TxBuffer[i] = 0xFF;

P_SCI3.BRR = 249;        //50K bps
P_SCI3.SMR.BYTE = 0x80;  //sync mode; 8 bits
P_SCI3.SSR.BYTE &=0x87;  //clear error flags
P_SPCR.BYTE = 0xE0;      //select pin P42/TXD is used as TXD
P_SCI3.SCR3.BYTE = 0x30; //enb tx, rx, SCK out

P_IO.PCR8 = 0x04;        //set as output
P_IO.PDR8.BYTE |= 0x04;  //set CS line high
delay();
P_IO.PDR8.BYTE &= 0xFB;  //set CS line low to initialise EEPROM
delay();
P_IO.PDR8.BYTE |= 0x04;  //set CS line high
}


void delay(void)         //delay time approximately 42μs
{
   for(i=0;i<10;i++)
   ;
 }
```

```
//Returns: SR bits as unsigned char
uchar ReadSR (void)
{
unsigned char dummy;

P_SCI3.SSR.BYTE &= 0xBF;                  //clear RDRF
P_IO.PDR8.BYTE &= 0xFB;                   //set CS line low to initialise EEPROM

while ((P_SCI3.SSR.BYTE & 0x80) != 0x80); //wait until ready to Tx
P_SCI3.TDR = READSR;                      //send readSR instruction (05)

while ((P_SCI3.SSR.BYTE & 0x40) != 0x40); //wait to get data0
P_SCI3.SSR.BYTE &= 0xBF;                  //clear RDRF
dummy=P_SCI3.RDR;

while ((P_SCI3.SSR.BYTE & 0x80) != 0x80); //wait until TDRE
P_SCI3.TDR = 0xff;                        //send dummy byte to receive data


while ((P_SCI3.SSR.BYTE & 0x40) != 0x40); //wait to get data1
P_SCI3.SSR.BYTE &= 0xBF;                  //clear RDRF

P_IO.PDR8.BYTE |= 0x04;                   //set CS line high

return(P_SCI3.RDR);
}


void SetWREN (void)
{
P_SCI3.SSR.BYTE &= 0xBF;                  //clear RDRF
P_SCI3.SCR3.BYTE &= 0x20;                 //disable RE
P_IO.PDR8.BYTE &= 0xFB;                   //set CS line low to init. EEPROM

while ((P_SCI3.SSR.BYTE & 0x80) != 0x80); //wait until ready to TX instruction
P_SCI3.TDR = SETWREN;                     //send setWREN instruction (06)

while ((P_SCI3.SSR.BYTE & 0x04) != 0x04); //wait to finish TX (denoted by RX)
P_SCI3.SCR3.BYTE = 0x30;                  //enable TE and RE
P_IO.PDR8.BYTE |= 0x04;                   //set CS line high to complete
}


void WriteByte (ushort Adrs, uchar Data)
{
unsigned char dummycount;

SetWREN();                                //set write enable flag
P_SCI3.SSR.BYTE &= 0xBF;                  //clear RDRF
P_SCI3.SCR3.BYTE = 0x20;                  //disable RE
P_IO.PDR8.BYTE &= 0xFB;                   //set CS line low to start operation
```

```
//First write the instruction
while ((P_SCI3.SSR.BYTE & 0x80) != 0x80); //wait until ready to tx
P_SCI3.TDR = WRITE;                       //write ins (02)

//Then send the address
while ((P_SCI3.SSR.BYTE & 0x80) != 0x80); //wait until TDRE
P_SCI3.TDR = swap((uchar)Adrs>>8);        //MSB of address

while ((P_SCI3.SSR.BYTE & 0x80) != 0x80); //wait until TDRE
P_SCI3.TDR = swap((uchar)Adrs);           //LSB of address

//Finally write data byte
while ((P_SCI3.SSR.BYTE & 0x80) != 0x80); //wait until TDRE
P_SCI3.TDR = swap(Data);                  //write data byte

while ((P_SCI3.SSR.BYTE & 0x04) != 0x04); //wait until TEND
P_SCI3.SCR3.BYTE = 0x30;                  //enable RE
P_IO.PDR8.BYTE |= 0x04;                   //set CS line high to complete
}


void Write (ushort Adrs, ushort Count, uchar *DataPtr)
{
SetWREN();                                //set write enable flag
P_SCI3.SSR.BYTE &= 0xBF;                  //clear RDRF
P_SCI3.SCR3.BYTE &= 0x20;                 //disable RE
P_IO.PDR8.BYTE &= 0xFB;                   //reset CS line to start operation

//issue the Write Instruction
while ((P_SCI3.SSR.BYTE & 0x80) != 0x80); //wait until TDR empty
P_SCI3.TDR = WRITE;                       //Write ins (02)

//send MSB of address
while ((P_SCI3.SSR.BYTE & 0x80) != 0x80); //wait until TDR empty
P_SCI3.TDR = swap((uchar)Adrs>>8);

//send LSB of address
while ((P_SCI3.SSR.BYTE & 0x80) != 0x80); //wait until TDR empty
P_SCI3.TDR = swap((uchar)Adrs);

//now set up for interrupt transfer
TxBuffer_ptr = DataPtr;                   //Set transmit buffer pointer

//the following is for polled
for (i=0;i<Count;i++)
{
while ((P_SCI3.SSR.BYTE & 0x80) != 0x80); //wait until TDR empty
P_SCI3.TDR = swap(*TxBuffer_ptr);
TxBuffer_ptr++;
}
```

```
while ((P_SCI3.SSR.BYTE & 0x04) != 0x04);   //wait until TEND
P_SCI3.SCR3.BYTE = 0x30;                    //enable RE
P_IO.PDR8.BYTE |= 0x04;                     //set CS line high to complete
}


void Read (ushort Adrs, ushort Count, uchar *Buffer)
{
int i;

RxCount = Count;                            //total bytes to be transferred
RxBuffer_ptr = Buffer;                      //set pointer to receive buffer
P_SCI3.SCR3.BYTE = 0x20;                    //disable RE
P_SCI3.SSR.BYTE &= 0xBF;                    //clear RDRF
P_IO.PDR8.BYTE &= 0xFB;                     //reset CS line to start operation

//issue the Read Instruction
while ((P_SCI3.SSR.BYTE & 0x80) != 0x80);   //wait until TDR empty
P_SCI3.TDR = READ;                          //send Read instruction (03)

//send MSB of address
while ((P_SCI3.SSR.BYTE & 0x80) != 0x80);   //wait until TDR empty
P_SCI3.TDR = swap((uchar)Adrs>>8);

//send LSB of address
while ((P_SCI3.SSR.BYTE & 0x80) != 0x80);   //wait until TDR empty
P_SCI3.TDR = swap((uchar)Adrs);

while ((P_SCI3.SSR.BYTE & 0x04) != 0x04);   //wait until TX end
P_SCI3.SCR3.BYTE = 0x70;                    //enable RE, TE and RE Interrupts

//now set up for interrupt transfer
TxBuffer_ptr = TxBuffer;                    //transmit null data

//the following is for polled
for (i=0;i<Count;i++)
{
  while ((P_SCI3.SSR.BYTE & 0x80) != 0x80); //wait until TDR empty
    P_SCI3.TDR = *TxBuffer_ptr;             //swap not required for null data
    TxBuffer_ptr++;
}

while(RxCount);
P_SCI3.SCR3.BYTE = 0x30;                    //enable RE, TE and disable RE Interrupts
P_IO.PDR8.BYTE |= 0x04;                     //set CS line high
}
```

```
/*************************************************************/
/*  FILE        :applicationdemo.h                           */
/*  DATE        :Fri, Dec 20, 2002                           */
/*  DESCRIPTION :Definition of variable                      */
/*  CPU TYPE    :H8/38024F                                   */
/*************************************************************/

//Type definitions
typedef unsigned char uchar;
typedef unsigned short ushort;

//bit patterns for EEPROM access instructions
#define READSR 0xA0              /* (bit reversed 05) */
#define SETWREN 0x60             /* (bit reversed 06) */
#define WRITE 0x40               /* (bit reversed 02) */
#define READ 0xC0                /* (bit reversed 03) */

//system constants
#define MAXRXCOUNT 35
#define MAXTXCOUNT 35            /* 32 bytes of data plus ins & address */

#define swap(x) (table[x])       /* swap macro */
```

参考文献

1. http://www.mct.net/faq/spi.html
2. Leonard Haile, *Interfacing the H8/3644 to a Serial E²PROM.-How to use the SCI Interface to emulate an SPI interface(Revision 1.0*), 8/3/98, Hitachi Semiconductor (America) Inc.
3. M95640/M95320-64/32 Kbit Serial SPI Bus EEPROM With High Speed Clock, 2002, STMicroelectronics. http://us.st.com/stonline/books/pdf/docs/5711.pdf
4. H8/38024 Series, H8/38024F-ZTZT Hardware Manual (version 2.0), 20 Feb 2002, Hitachi Ltd.

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2003.09.19	—	初版発行

安全設計に関するお願い

1. 弊社は品質、信頼性の向上に努めておりますが、半導体製品は故障が発生したり、誤動作する場合があります。弊社の半導体製品の故障又は誤動作によって結果として、人身事故、火災事故、社会的損害などを生じさせないような安全性を考慮した冗長設計、延焼対策設計、誤動作防止設計などの安全設計に十分ご留意ください。

本資料ご利用に際しての留意事項

1. 本資料は、お客様が用途に応じた適切なルネサス テクノロジ製品をご購入いただくための参考資料であり、本資料中に記載の技術情報についてルネサス テクノロジが所有する知的財産権その他の権利の実施、使用を許諾するものではありません。
2. 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例の使用に起因する損害、第三者所有の権利に対する侵害に関し、ルネサス テクノロジは責任を負いません。
3. 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他全ての情報は本資料発行時点のものであり、ルネサス テクノロジは、予告なしに、本資料に記載した製品または仕様を変更することがあります。ルネサス テクノロジ半導体製品のご購入に当たりましては、事前にルネサス テクノロジ、ルネサス販売または特約店へ最新の情報をご確認頂きますとともに、ルネサス テクノロジホームページ(http://www.renesas.com)などを通じて公開される情報に常にご注意ください。
4. 本資料に記載した情報は、正確を期すため、慎重に制作したものですが万一本資料の記述誤りに起因する損害がお客様に生じた場合には、ルネサス テクノロジはその責任を負いません。
5. 本資料に記載の製品データ、図、表に示す技術的な内容、プログラム及びアルゴリズムを流用する場合は、技術内容、プログラム、アルゴリズム単位で評価するだけでなく、システム全体で十分に評価し、お客様の責任において適用可否を判断してください。ルネサス テクノロジは、適用可否に対する責任は負いません。
6. 本資料に記載された製品は、人命にかかわるような状況の下で使用される機器あるいはシステムに用いられることを目的として設計、製造されたものではありません。本資料に記載の製品を運輸、移動体用、医療用、航空宇宙用、原子力制御用、海底中継用機器あるいはシステムなど、特殊用途へのご利用をご検討の際には、ルネサス テクノロジ、ルネサス販売または特約店へご照会ください。
7. 本資料の転載、複製については、文書によるルネサス テクノロジの事前の承諾が必要です。
8. 本資料に関し詳細についてのお問い合わせ、その他お気付きの点がございましたらルネサス テクノロジ、ルネサス販売または特約店までご照会ください。