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

I/O ポートを使用した I²C™ EEPROM I/F 実現方法

内容

本アプリケーションノートは、I²C バスインタフェースの概要と、SLP シリーズ 38024（I²C マスタ）を 2 本の信号線を用いた Mircochip 社製 24AA16 16K I²C シリアル EEPROM（I²C スレーブ）に接続するアプリケーションの詳細を説明します。

はじめに

H8/300L Super Low Power (SLP)シリーズは、種々の内蔵機能を備えており、組み込みアプリケーションをシステム設計する上での要求事項を簡単化します。内蔵機能としては、低消費電力モード、クロック発振器、ウォッチドッグタイマ（WDT）、シリアルコミュニケーションインタフェース（SCI3）、各種外部／内蔵割り込みなどがあります。

しかしアプリケーションによっては、さらに外部の周辺機能（I/O 拡張や専用メモリなど）を付加する必要があります。標準的な周辺バス接続として、Inter-IC（I²C）バスと呼ばれる単純な 2 本の信号線を用いた双方向のシリアルインタフェースを用いることができます。SLP シリーズは専用の I²C 用ハードウェアを内蔵していませんが、2 本の I/O 端子をソフトウェアでコントロールしてこのシリアルバスインタフェースをシミュレートできます。

動作確認デバイス

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

目次

1. I²C ™インタフェースの概要

I²C バスは、シリアルデータ信号（SDA）とシリアルクロック信号（SCL）の 2 本の信号線によるインタフェースを用いて、バスに接続したデバイス間で情報をやり取りします。バス上の各デバイスは固有のアドレスを持ち、機能に応じてトランスミッタまたはレシーバとして動作します。

さらに、デバイスはマスタとスレーブとして機能を持たせることもできます。マスタは、転送を開始・制御（すべてのフレーミング信号とクロック信号を生成）・終了させるデバイスです。スレーブは、マスタからアクセスされるデバイスです。

（I²C 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）

この章では、次の 3 点を説明します。

1. I²C の制御
2. ライト動作
3. リード動作

1.1 I²C EEPROM の制御

1.1.1 START（開始）および STOP（停止）条件

I²C バスでは、データ転送はバスマスタが生成する 2 つの別々のバス状態によって制御（フレーミング）されます。バスが空いているとき、2 本の信号線はどちらもハイレベルになります。2 つのバス状態は、開始（START）ビット条件と停止（STOP）ビット条件です。

開始条件は、SDA がハイレベルからローレベルに遷移し、SCL がハイレベルの状態です。停止条件は、SDA がローレベルからハイレベルに遷移し、SCL がハイレベルの状態です。SCL がハイレベルのときは、SDA 上のデータは常に有効（安定）でなくてはなりません。SCL がローレベルのときのみ、SDA 信号レベルが変化します。SCL の 1 周期ごとに、1 ビットのデータが送信されます。

開始条件に続いてバスメッセージとして送信される先頭バイト（8 ビット）は、7 ビットのスレーブアドレスフィールドとデータ方向（R/W）ビットです（この説明は、I²C の 7 ビットアドレッシングモードのみを対象にしています）。データ方向ビット（最下位ビット）は、マスタがスレーブにデータを送信するのか（0 = ライト）、スレーブから受信するのか（1 = リード）を指定します。

アクノリッジビットは SDA 上のローレベル信号で、マスタが送ったアクノリッジクロックパルス（バイト送信の 9 番目のハイレベル SCL クロック）の期間中に受信側デバイス（マスタまたはスレーブ）が送信します。レシーバがデータを受信できない場合（ビジー状態のスレーブレシーバ）またはデータ条件の終了を指示したい場合（マスタであるレシーバ）、ノンアクノリッジを送信します（9 番目のハイレベル SCL クロックの期間中にハイレベルの SDA を送ります）。

開始条件およびスレーブアドレス送信に続き、マスタとレシーバのあいだで要求に応じてデータ転送を行ないます。最終バイトを転送し、そのアクノリッジが返るとただちに、マスタは停止条件を発行してバスの使用を終了します。

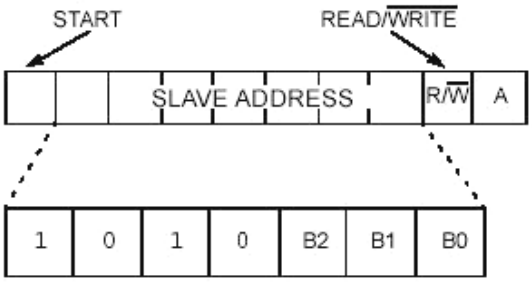
上述の I²C バス動作の概要により、シリアル EEPROM インタフェースを進める上で必要な基礎事項（I²C 7 ビットアドレスッシングモードの実用上で必要な知識です）が明らかになりました。

（I²C の詳細な構成については、SPI および I²C に関するアプリケーションノートを参照してください）

（また、I²C バスがサポートする各モードの詳細については、Phillips 社発行の I²C バス仕様書を参照してください）

1.1.2 デバイスアドレスの指定

マスタデバイスから開始条件の次に送られてくる先頭バイトは、コントロールバイトです。コントロールバイトには 4 ビットのコントロールコードがあり、24AA16 ではリードおよびライト動作の場合 2 進数の 1010 になります。



コントロールバイトの次の 3 ビットはブロックセレクトビット（B2、B1、B0）です。これにより、マスタデバイスが、8 つの 256 ワードメモリブロックのなかからどのブロックにアクセスするかを選択します。このビットは、実際はワードアドレスの最上位 3 ビットです。I²C プロトコルにより、メモリサイズは 256 ワード x 8 ブロックに決められていますので、1 つのシステムでサポートできる 24AA16 メモリは 1 個のみであることに注意してください。

コントロールバイトの最終ビットは、実行すべき動作を定義します。ビットが‘1’の場合はリード動作で、‘0’のときはライト動作です。開始条件のあと、24AA16 は SDA バスを監視して、送信されてくるデバイスタイプ ID をチェックし、コード 1010 を検出すると、スレーブデバイスとして SDA にアクノリッジ信号を出力します。R/W ビットにしたがい、24AA16 はリードまたはライト動作を選択します。

Operation	Control Code	Block Select	R/$\overline{W}$
Read	1010	Block Address	1
Write	1010	Block Address	0

【注】 コントロールバイトはデバイスにより異なります。ご使用の前に、コントロールバイトの詳細について、デバイスのデータシートまたは I²C バス仕様書を参照してください。

1.1.3 ビット転送とデータの有効性

開始条件から停止条件のあいだに、トランスミッタからレシーバに対して送信するデータのバイト数には制限はなく、マスタデバイスが決めることができます。各バイト（8 ビット固定）は、最上位ビットからシリアルに転送され、うしろにアクノリッジビットが続きます。

開始条件のあと、クロック信号がハイレベルの期間はデータ信号が安定し、このときの信号が有効なデータです。

信号上のデータは、クロック信号がローレベルの期間に変更しなくてはなりません。データ 1 ビットにつき、クロックは 1 パルスです。

1.1.4 アクノリッジ

アドレス指定された受信側デバイスは、1 バイト受信するごとにアクノリッジを出力しなくてはなりません。マスタデバイスは、このアクノリッジビットに合せて、クロックを 1 パルス追加で出力しなくてはなりません。

24AA16 では、内部の書き込みサイクルが進行中のあいだはアクノリッジビットを出力しません。

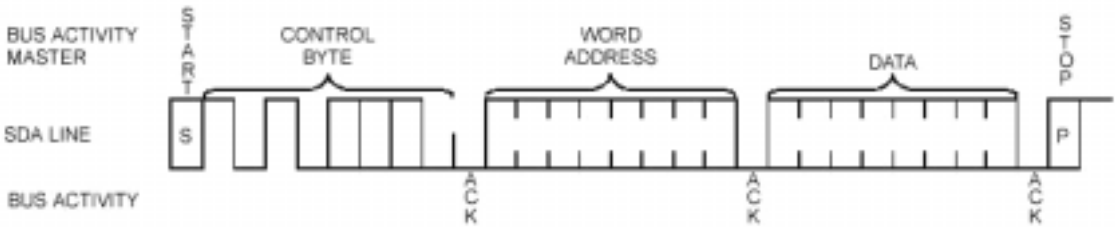
1.2 ライト動作

スレーブアドレスの R/W ビットが‘0’にセットされると、ライト動作が開始します。ライト動作にはバイトライトとページライトの 2 種類があります。

1.2.1 バイトライト

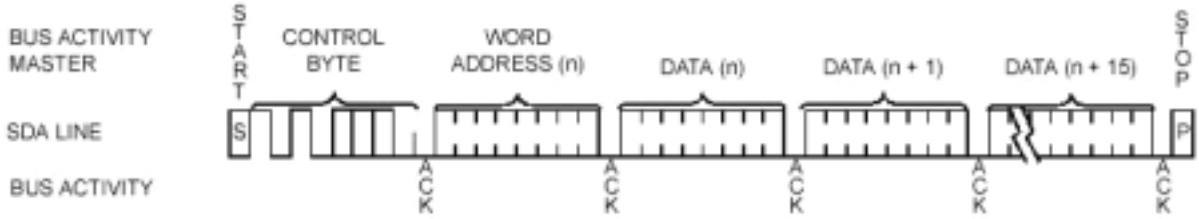
バイトライト動作では、ランダムな EEPROM アドレスに書き込むことができます。バイトライト動作では、以下の送信が必要です。

- 開始条件（マスタ）
- EEPROM デバイスアドレスと R/W ビット= 0（マスタ）
- アクノリッジビット（EEPROM）
- 書き込み対象の EEPROM ワードアドレス（マスタ）*1
- アクノリッジビット（EEPROM）
- 書き込むデータバイト（マスタ）
- アクノリッジビット（EEPROM）
- 停止条件（マスタ）



1.2.2 ページライト

ページライト動作は上述のバイトライト動作とほぼ同様ですが、バイトライトは 1 バイトを書き込むのに対して、ページライトは EEPROM アドレスと停止ビットの転送のあいだで 16 バイト*2 を転送できる点だけが異なります。ページライト動作中は、EEPROM はバイトごとに内部アドレスポインタを自動的にインクリメントします。



【注】 1. **ワードアドレス数はデバイスにより異なります。詳細は該当するデバイスのデータシートを参照してください。**
2. **ページライト動作のバイト数はデバイスにより異なります。該当するデバイスのデータシートで確認してください。**

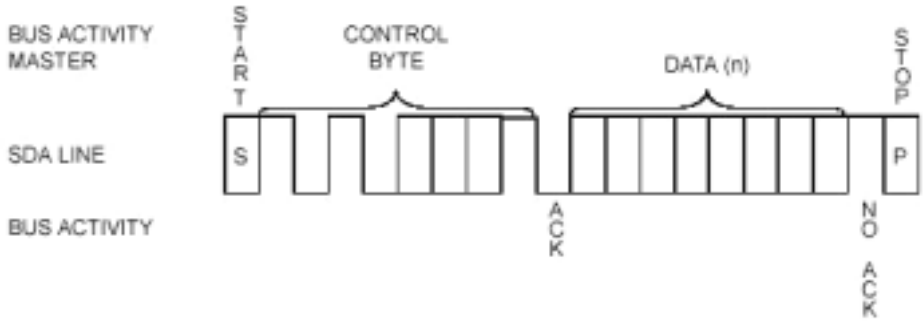
1.3 リード動作

リード動作は、カレントアドレス、ランダム、シーケンシャルの 3 種類をサポートしています。リード動作もライト動作と同様の方法で開始しますが、デバイスアドレスバイトの場合 R/W ビットが 1 にセットされる点だけが異なります。

1.3.1 カレントアドレスリード

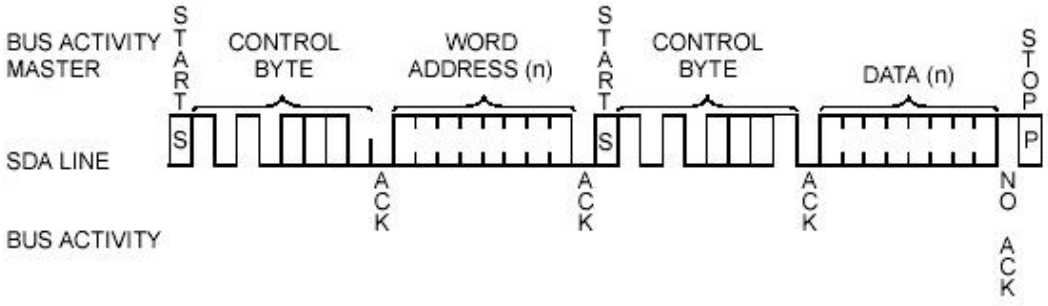
カレントアドレスリードモードの場合、スレーブによりマスタに転送されたデータは、最後にアクセスしたアドレス+1 の場所から読み出されるので EEPROM バイトアドレスは書き込まれません。このタイプのリード動作は次のように行なわれます。

- 開始条件（マスタ）
- EEPROM デバイスアドレスと R/W ビット= 1（マスタ）
- アクノリッジビット（EEPROM）
- 読み出したデータバイト（指定したスレーブの最後にアクセスしたメモリアドレス+ 1 の場所から転送した EEPROM バイト）
- ノンアクノリッジビット（マスタ）
- 停止条件（マスタ）



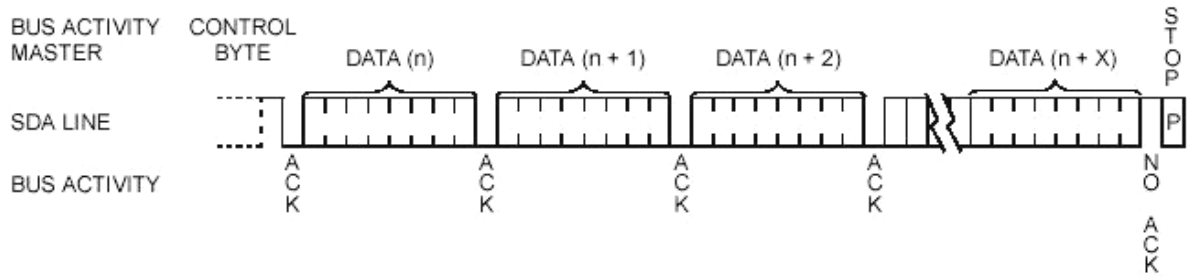
1.3.2 ランダムリード

ランダムリードモードはダミーバイトライトサイクルで開始し（マスタは開始条件、デバイスアドレス、対象ワードアドレス*[1]をこの順で送信）、次に上述のカレントアドレスモードサイクルが続きます。

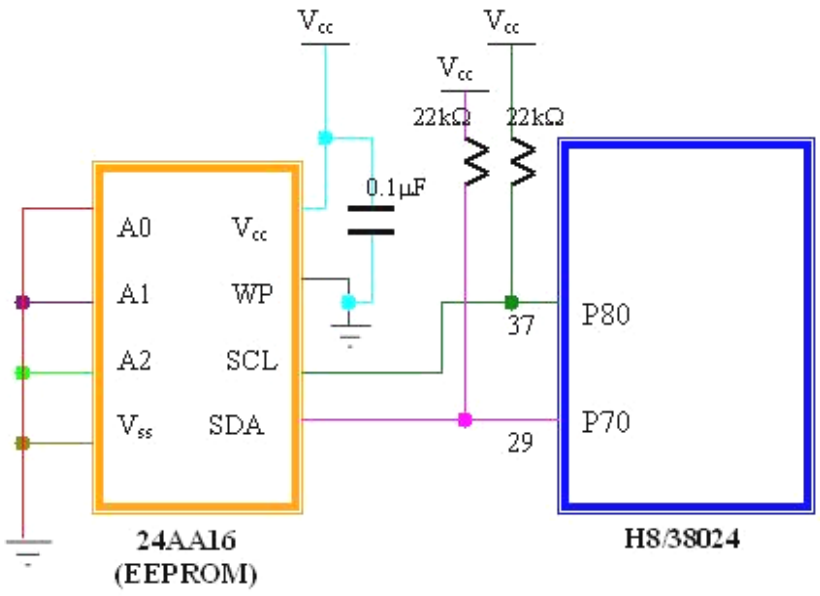


1.3.3 シーケンシャルリード

シーケンシャルリードモードは、ランダムリードで開始します。マスタは 1 バイトの転送後にノンアクノリッジでリード動作を終了させるのでなく、各データバイトを受け取ったあとに有効なアクノリッジ信号を返します。このアクノリッジを受け、スレーブ EEPROM はリード動作を続けて次のデータバイトを送信します。シーケンシャルリードは、マスタが最後のバイトを読み出したあとにノンアクノリッジを発行して終了させるまで続けられ、このあとに停止条件が続きます。



2. ハードウェア設計



*Signal Name

A0,A1,A2	*User Configuration Chip Selects*
V_{ss}	*Ground*
SDA	*Serial Data*
SCL	*Serial Clock*
WP	*Write Protected Input*
V_{cc}	*Supply Voltage*

3. 関数の概要

I2C.c で使用する関数：

- void main(void)

RW.c で使用する関数：

- unsigned char SclIn (void)
- unsigned char SdaIn (void)
- void SclOut (unsigned char)
- void SdaOut (unsigned char)
- void Delay(void)
- void Delay2x(void)
- unsigned char CheckBusState(void)
- void SendStartBit(void)
- void SendBit (unsigned char)
- unsigned char GetBit (void)
- unsigned char GetAck (void)
- unsigned char SendByte(unsigned char)
- unsigned char GetByte(void)
- void SendStopBit(void)
- unsigned char I2cWrite(unsigned char, unsigned char *,unsigned char, unsigned char)
- unsigned char I2cRead(unsigned char, unsigned char *,unsigned char, unsigned char)
- char CheckWriteReady(void)
- unsigned char I2cCurrentRead(unsigned char , unsigned char *)

`void main(void)`
main 関数では、I²C プロトコルでの EEPROM の制御のデモとテストを行ないます。バイトライトとページライトで EEPROM に書き込んでから、カレントアドレスリード、シーケンシャルリード、またはランダムリードで EEPROM から読み出します。

`unsigned char SclIn (void)`
本関数は、ポート端子 SclIn を入力と出力のいずれとして使うかを制御します。ビットを 0 にクリアすると入力端子となります。また、SCL ポートがローレベルかハイレベルかをチェックします。

`unsigned char SdaIn (void)`
本関数は、ポート端子 SdaIn を入力と出力のいずれとして使うかを制御します。ビットを 0 にクリアすると入力端子となります。また、SDA ポートがローレベルかハイレベルかをチェックします。

`void SclOut (unsigned char )`

引数： SCL 信号レベルのチェック結果 (unsigned char)

本関数は、ポート端子 SclOut を入力と出力のいずれとして使うかを制御します。ビットを 1 にセットすると出力端子となります。

`void SdaOut (unsigned char)`

引数： SCL 信号レベルのチェック結果 (unsigned char)

本関数は、ポート端子 SdaOut を入力と出力のいずれとして使うかを制御します。ビットを 1 にセットすると出力端子となります。

`void Delay(void)`
誤って不要な信号を送ってしまわないように、SCL の立ち下がりエッジの未確定領域を避けるための最小の内部遅延時間を生成します。

`void Delay2x(void)`
関数 void Delay(void)をもとに、2 倍の遅延時間を生成します。

`unsigned char CheckBusState(void)`
本関数は、I²C バスが空き状態であるか（SCL と SDA の両者がハイレベル）ビジー状態であるかを検出します。

`void SendStartBit(void)`
本関数は開始条件を送信します。

`void SendBit (unsigned char)`

引数： 送信するデータビット(unsigned char)

ビットフォーマットのデータを送信します。

`unsigned char GetBit (void)`
ビットフォーマットで入力されるデータを受信します。

`unsigned char GetAck (void)`
本関数は GetBit と同様の機能ですが、マスタは ACK が返っているか（SDA がローレベル）どうかを検出する前に SDA をハイレベルにしなくてはならないため、より注意が必要な動作です。

`unsigned char SendByte(unsigned char)`

引数： 送信するバイトデータ (unsigned char)

最上位ビット（MSB）からバイトを送信します。

`unsigned char GetByte(void)`
最上位ビット（MSB）からバイトを受信します。

`void SendStopBit(void)`
停止条件を送信して動作を終了させます。

```
unsigned char I2cWrite(unsigned char, unsigned char *,unsigned char, unsigned char )
```

引数：
スレーブアドレス (unsigned char),
データを格納しているバッファのアドレス (unsigned char*),
データを書き込むワードアドレス(unsigned char),
データ長 (unsigned char)

本関数は、以下の手順で、バイトライトとページライトの両方に適用できるライト動作を行ないます。

i) バス状態をチェックする
ii) 開始条件を送信する
iii) コントロールコードとスレーブアドレスを送信する
iv) 書き込み先のワードアドレスを送信する
v) 書き込みデータ長にしたがい、EEPROM にデータを書き込む
vi) 停止条件.

```
unsigned char I2cRead(unsigned char, unsigned char *,unsigned char, unsigned char )
```

引数：
スレーブアドレス (unsigned char),
データを格納するバッファのアドレス (unsigned char*),
データを読み出すワードアドレス(unsigned char),
データ長 (unsigned char)

本関数は、以下の手順で、ランダムリードとシーケンシャルリードの両方に適用できるリード動作を行ないます。

i) バス状態をチェックする
ii) 開始条件を送信する
iii) ダミーのコントロールコードとスレーブアドレスを送信する
iv) 読み出しをするワードアドレスを送信する
v) SDA 信号をハイレベルにする
vi) 開始ビット
vii) コントロールコードとスレーブアドレスを送信する
viii) 読み出しデータ長にしたがい、EEPROM からデータを読み出す
ix) 停止条件

```
char CheckWriteReady(void)
```

24AA16 などの Microchip 社製デバイスは、ライトサイクル中はアクノリッジを送信しないため、本関数を用いてサイクルが完了したことを確定します。サイクルが完了すると、デバイスは ACK を返し、マスタは次のリードまたはライトコマンドに進むことができます。

```
unsigned char I2cCurrentRead(unsigned char , unsigned char *)
```

引数： スレーブアドレス (unsigned char), データを格納するバッファのアドレス (unsigned char*),

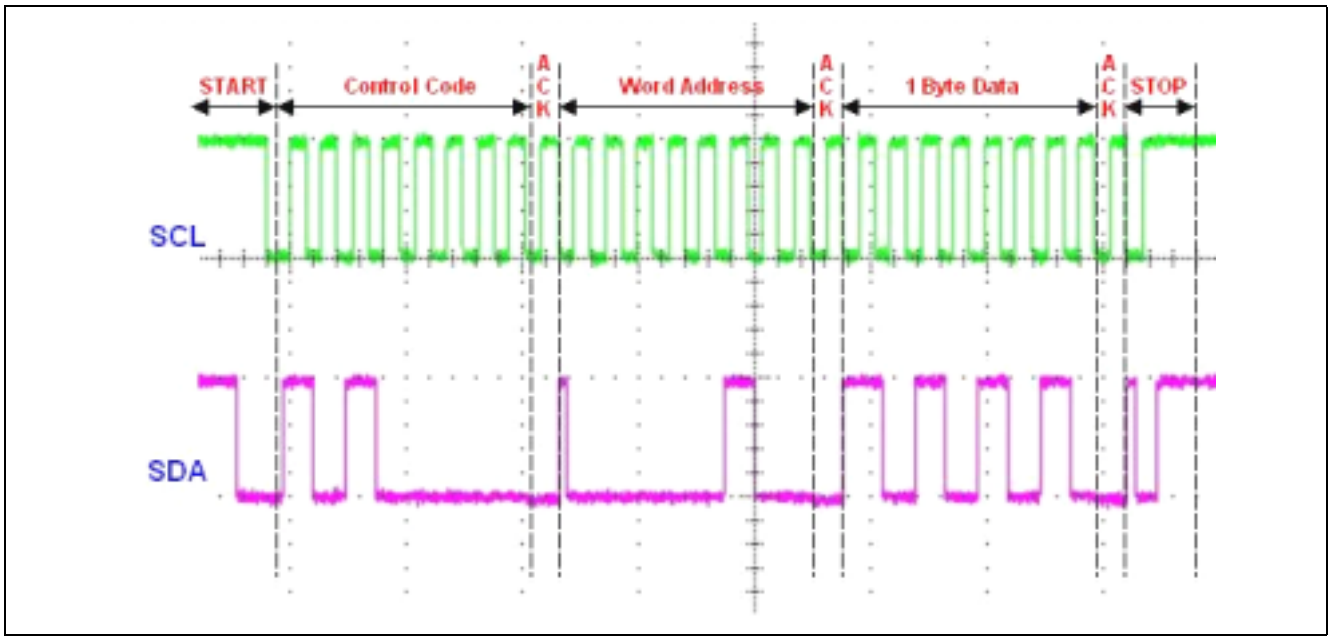
本関数は、以下の手順で、ランダムリードとシーケンシャルリードの両方に適用できるリード動作を行ないます。

i) バス状態をチェックする
ii) 開始条件を送信する
iii) コントロールコードとスレーブアドレスを送信する
iv) EEPROM から 1 バイトのデータを読み出す
v) 停止条件

4. プログラムの解析

4.1 バイトライト

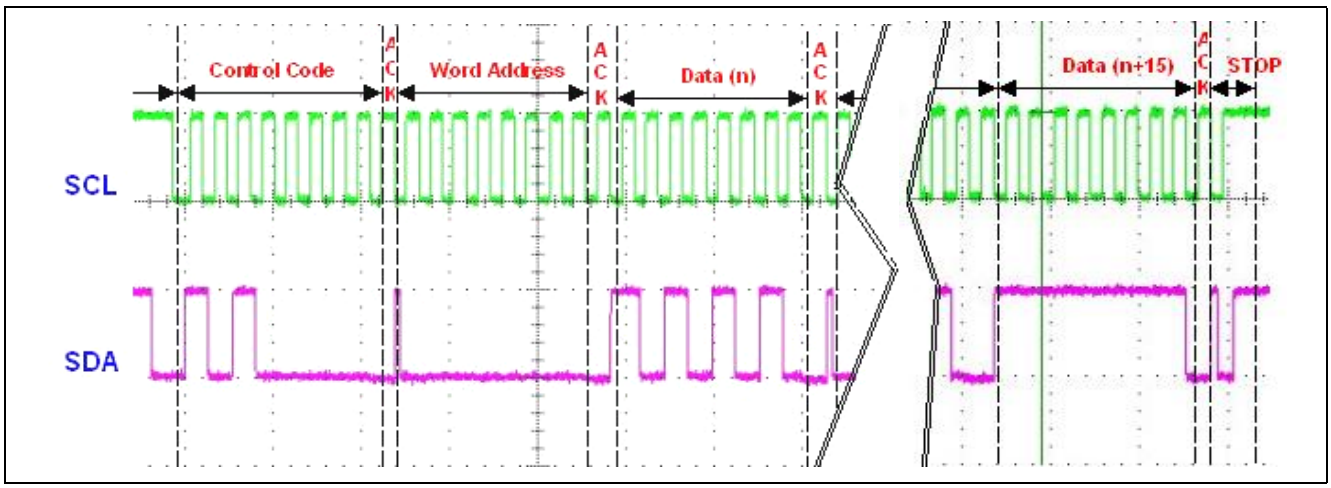
下図にバイトライト動作を示します。



1 バイトのライト動作時間は平均*16.0ms です。

4.2 ページライト

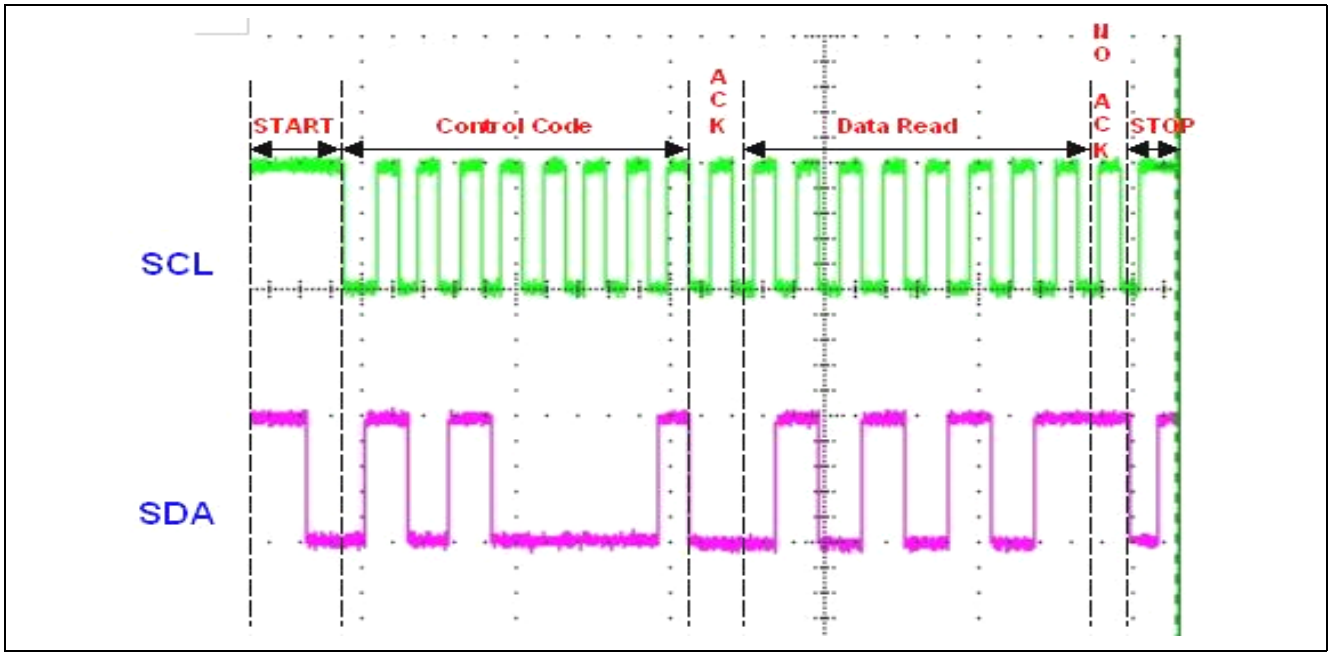
下図にページライト動作を示します。



1 ページ（16k バイト）のライト動作時間は平均*89.5ms です。

4.3 カレントリード

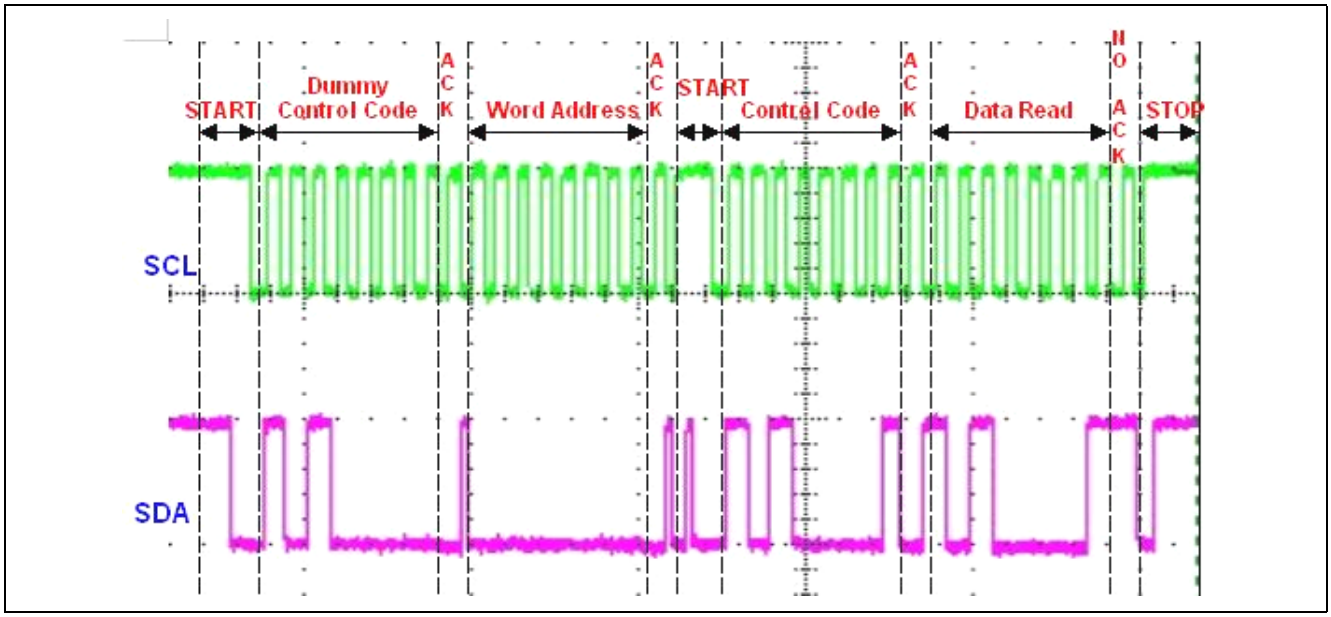
下図にカレントリード動作を示します。



カレントリードでの 1 バイトのリード動作時間は平均*11.2ms です。

4.4 ランダムリード

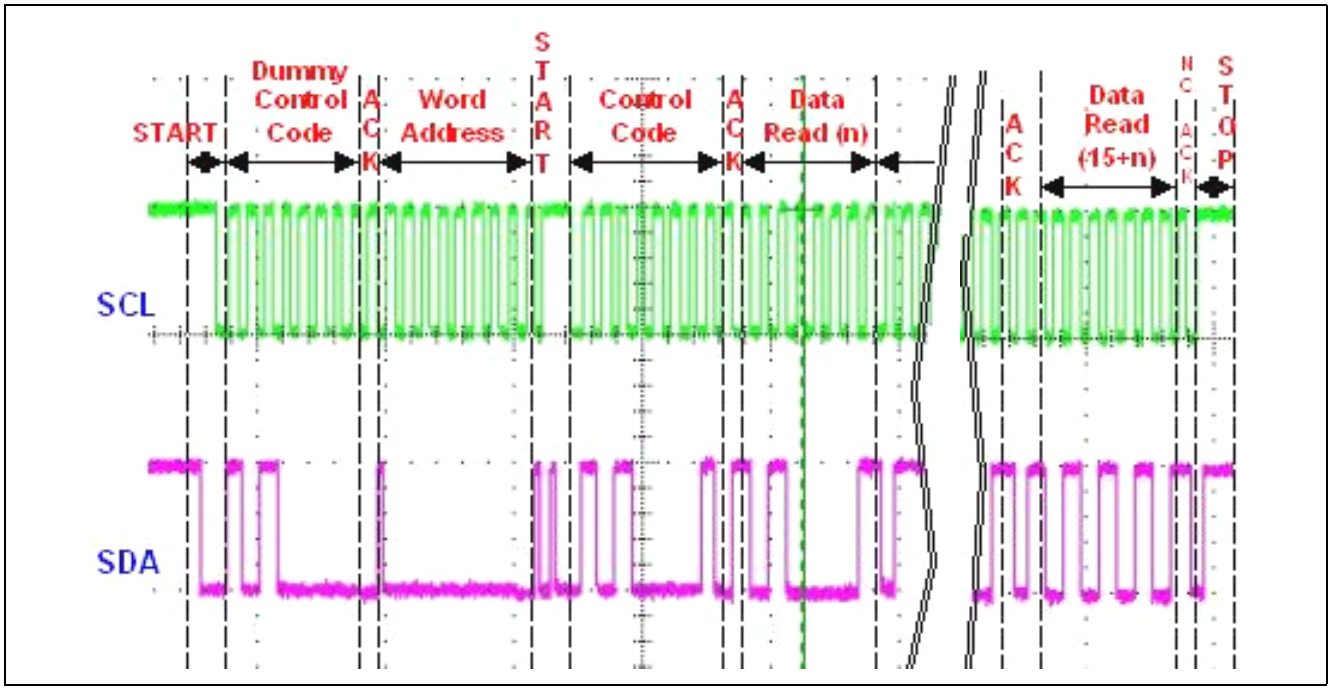
下図にランダムリード動作を示します。



ランダムリードでの 1 バイトのリード動作時間は平均*22.3ms です。

4.5 シーケンシャルリード

下図にシーケンシャルリード動作を示します。



シーケンシャルリードでの 1 ページ（16k バイト）のリード動作時間は平均*97.8ms です。

【注】＊上記の平均時間は、すべて次の条件での計測値です。

i) 10MHz **クロック**
ii) 1/2 **システムクロック分周比**

5. プログラム例

```
/************************************************************************/
/*                                                                      */
/*  FILE        :I2C.c                                                  */
/*  DATE        :Fri, Dec 27, 2002                                      */
/*  DESCRIPTION :Main Program                                           */
/*  CPU TYPE    :H8/38024F                                              */
/*                                                                      */
/*  This file is generated by Hitachi Project Generator (Ver.2.1).      */
/*                                                                      */
/************************************************************************/

#include "iodefine.h"
#include "I2C.h"
#include <stdio.h>

#define NODE_ADDR       0xa0

unsigned int i;

unsigned char buf[16]={0xaa, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
                       0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff  };

void main(void)
{
  byte_write();

  for (i=0;i<1000;i++);               //delay approximately 500ms

  page_write();

  current_address_read();

  random_or_sequential_read();
}
```

```
void byte_write(void)
{
  //byte write
  i = I2cWrite(NODE_ADDR, buf, 1, 0x00);
     if (CheckWriteReady() ==1)
  i = I2cWrite(NODE_ADDR, buf, 2, 0x02);
     if (CheckWriteReady() ==1)
  i = I2cWrite(NODE_ADDR, buf, 1, 0x04);
}

void page_write(void)
{
  //page write
  i = I2cWrite(NODE_ADDR, buf, 16, 0x00);
}

void current_address_read(void)
{
  //current address read
  i = I2cCurrentRead(NODE_ADDR, buf);
}

void random_or_sequential_read(void)
{
  //random / sequential read
  i = I2cRead(NODE_ADDR, buf,16, 0x00);
}
```

```
/************************************************************************/
/*                                                                      */
/*  FILE        :RW.c                                                   */
/*  DATE        :Fri, Dec 27, 2002                                      */
/*  DESCRIPTION :Function Program                                       */
/*  CPU TYPE    :H8/38024F                                              */
/*                                                                      */
/*  This file is generated by Hitachi Project Generator (Ver.2.1).      */
/*                                                                      */
/************************************************************************/



#include "i2c.h"
#include "iodefine.h"

char CheckWriteReady(void);


/* Check SCL signal level */
unsigned char SclIn (void)
{
    SCL_IO_REG &= SCL_IO_RESET_BIT;            //It is Input
    if(SCL_DATA_REG & SCL_DATA_SET_BIT)        //check Port status
{
      return(HIGH);
    }
    else
    {
      return(LOW);
    }
}

/* Check SDA signal level */
unsigned char SdaIn (void)
{
    SDA_IO_REG &= SDA_IO_RESET_BIT;            //It is Input
    if(SDA_DATA_REG & SDA_DATA_SET_BIT)        //check Port status
{
       return(HIGH);
    }
    else
    {
      return(LOW);
    }
}
```

```
/* Drive SCL bus */
void SclOut (unsigned char status)
{
if (status == LOW)
{
    SCL_DATA_REG = 0;                   //Drive Port LOW
    SCL_IO_REG |= SCL_IO_SET_BIT;       //Port is output

    }
   else
   {
        SCL_DATA_REG = 1;               //Port is Input & using external pull-up
                                        //resistor to go high
        SCL_IO_REG |= SCL_IO_SET_BIT;   //Port is output
      }
}

/* Drive SDA bus */
void SdaOut (unsigned char status)
{
    if (status == LOW)
   {
        SDA_DATA_REG = 0;               //Drive Port LOW
        SDA_IO_REG |= SDA_IO_SET_BIT;   //Port is output
   }
   else
   {
     SDA_DATA_REG = 1;                  //Port is Input & using external pull-up
                                        //resistor to go high
     SDA_IO_REG |= SDA_IO_SET_BIT;      //Port is output
   }
}

void Delay(void)                        //Delay approximately 10ms
{
    unsigned char i;

    i=0;
    while(i<20)
    {
        i++;
    }
}

void Delay2x(void)                     //Delay approximately 20ms
{
    Delay();
    Delay();
}
```

```
/* All codes below here are independent with hardware, such as microprocessor,
I/O port */
unsigned char CheckBusState(void)
{
    if ((SclIn() == HIGH) && (SdaIn() == HIGH))
    {
      return(TRUE);
    }
    else
    {
      return(FALSE);
    }
}

/* It is nice to send out data at the middle of clock low */
void SendBit (unsigned char data_byte)
{
    SclOut(LOW);
    Delay();
    if (data_byte != 0)
    {
      SdaOut(HIGH);
    }
    else
    {
      SdaOut(LOW);
    }

    Delay();
    SclOut(HIGH);
    while (SclIn() != HIGH) {}  //wait for slow device to release clock
    Delay2x();
}

void SendStartBit(void)
{
    Delay();
    SdaOut(LOW);

    Delay2x();
    Delay2x();

    SclOut(LOW);
    Delay();
}
```

```
/* It is nice to sample data input at the middle of clock high */
unsigned char GetBit (void)
{
    unsigned char temp;

    SclOut(LOW);
    temp = SdaIn();
    Delay2x();

    SclOut(HIGH);
    while (SclIn() != HIGH) {}  //wait for slow device to release clock
    Delay();

    temp = SdaIn();
    Delay();

    return(temp);
}

/* Getting ACK is similar to GetBit, but it is a little tricky since master
must pull SDA high
before it find out whether there is ACK (SDA is low) or not */
unsigned char GetAck (void)
{
    unsigned char temp;

    SclOut(LOW);
    Delay();

    SdaOut(HIGH);
    temp = SdaIn();
    Delay();

    SclOut(HIGH);
    while (SclIn() != HIGH) {}  //wait for slow device to release clock
    Delay();

    temp = SdaIn();
    Delay();

    return(temp);
}
```

```
/* Note master need get ACK after sending out every byte */
unsigned char SendByte(unsigned char data_byte)
{
    unsigned char i;
    unsigned char mask;

    mask = 0x80;        //send out MSB first
    for (i=0; i<8; i++)
   {
        SendBit(data_byte & mask);
        mask /= 2;
     }

    return(GetAck());
}

unsigned char GetByte(void)
{
    unsigned char temp1, temp2;
    unsigned char i,mask;

    mask = 0x80;
    temp2 = 0;
    for (i=0; i<8; i++)
    {
        temp1 = GetBit() * mask;
        temp2 += temp1;
        mask /= 2;
    }

    return(temp2);
}

void SendStopBit(void)
{
    SclOut(LOW);
    Delay();

    SdaOut(LOW);
    Delay();

    SclOut(HIGH);
    Delay2x();

    SdaOut(HIGH);
}
```

```
unsigned char I2cWrite(unsigned char slave_addr, unsigned char *buf_ptr,
unsigned char length,unsigned char word_addr) {
    unsigned int i;

    if( CheckBusState() != TRUE)
    {
     return(BUS_BUSY);
    }

    SendStartBit();

    if (SendByte((slave_addr) & 0xfe) != LOW)   //Send address and write command
    {
     return(NO_RESPONSE);
    }

    if (SendByte(word_addr) != LOW)             //Send low word address
    {
    return(NO_RESPONSE);
    }


   for(i=0; i<length; i++)
   {
        if(SendByte(*buf_ptr++) != LOW)
       {
            return(ERR_RESPONSE);
       }
   }

    SendStopBit();

    return(OP_DONE);
}
```

```
unsigned char I2cRead(unsigned char slave_addr, unsigned char *buf_ptr,
unsigned char length,unsigned char word_addr)
{
    unsigned char i=0,j=0;
    unsigned char DataBuffer[500];
    unsigned char LastData;

    if( CheckBusState() != TRUE)
   {
        return(BUS_BUSY);
     }

    SendStartBit();

    if (SendByte((slave_addr) & 0xfe) != LOW)    //Send address and read command
   {
        return(NO_RESPONSE);
     }

   if (SendByte(word_addr) != LOW)              //Send high word address
   {
     return(NO_RESPONSE);
     }

    SendStopBit();                               //Pull-up SDA line
    SendStartBit();

    if (SendByte((slave_addr) | 0x01) != LOW)    //Send address and read command
   {
        return(NO_RESPONSE);
     }

    for(i=0; i<length-1; i++)
   {
      DataBuffer[i]= GetByte();
     SendBit(LOW);                               //ack it low
     }

    DataBuffer[i]= GetByte();                    //get last data and ack high

    SendBit(HIGH);
    SendStopBit();

    return(OP_DONE);
}
```

```
/*Since Microchip devices such as 24AA16 will not acknowledge during a write
cycle, this can be used to determined when the cycle is complete. So that the
master can proceed with next operation*/
char CheckWriteReady(void)
{
   unsigned int i=0;

   while(i<4)
   {
       SendStartBit();

       if (SendByte((0xa0) | 0x00) == LOW)
       {
        SendStopBit();
        return (1);
        }

        SendStopBit();
     i++;
      }
}

unsigned char I2cCurrentRead(unsigned char slave_addr, unsigned char *buf_ptr)
{
    unsigned char i=0,j=0;
    unsigned char DataBuffer[500];
    unsigned char LastData;

   SendStartBit();

    if (SendByte((slave_addr) | 0x01) != LOW)  //Send address and read command
   {
        return(NO_RESPONSE);
     }

    *buf_ptr = GetByte();                     //get last data and ack high

    SendBit(HIGH);
    SendStopBit();

    return(OP_DONE);
}
```

```
/*****************************************************************/
/*                                                               */
/*  FILE        :i2c.h                                           */
/*  DATE        :Fri, Dec 27, 2002                               */
/*  DESCRIPTION :Definition of constant and functions            */
/*  CPU TYPE    :H8/38024F                                       */
/*                                                               */
/*  This file is generated by Hitachi Project Generator (Ver.2.1).*/
/*                                                               */
/*****************************************************************/

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

/* Set as 1010 binary for read and write operation (device dependent)*/
#define NODE_ADDR       0xa0

/* Misc */
#define HIGH    1
#define LOW     0

#if !defined(TRUE)
#define TRUE    1
#endif

#if !defined(FALSE)
#define FALSE   0
#endif

#define OP_DONE         0x00
#define BUS_BUSY        0x01
#define NO_RESPONSE     0x02
#define ERR_RESPONSE    0x04

// SDA and SCL port def.

/* control SDA port as input or output */
#define SDA_IO_REG              P_IO.PCR7.BYTE
#define SDA_IO_SET_BIT          0x01    //output
#define SDA_IO_RESET_BIT        0xfe    //input

/* check SDA port low or high */
#define SDA_DATA_REG            P_IO.PDR7.BYTE
#define SDA_DATA_SET_BIT        0x01
#define SDA_DATA_RESET_BIT      0xfe

/* control SCL port as input or output */
#define SCL_IO_REG              P_IO.PCR8.BYTE
#define SCL_IO_SET_BIT          0x01    //output
#define SCL_IO_RESET_BIT        0xfe    //input
```

```
/* check SCL port low or high */
#define SCL_DATA_REG            P_IO.PDR8.BYTE
#define SCL_DATA_SET_BIT        0x01
#define SCL_DATA_RESET_BIT      0xfe

//I2C modules
void byte_write(void);
void page_write(void);
void current_address_read(void);
void random_or_sequential_read(void);
void startbit(void);
void stopbit(void);
void sendbit(unsigned char );
void controlbyte(unsigned char );
void ReadBuffer(void);
extern char checkwrirdy();
unsigned char getACK(void);
unsigned char getbit(unsigned char);
unsigned char I2cWrite(unsigned char slave_addr, unsigned char *buf_ptr,
unsigned char length,unsigned char word_addr);
unsigned char I2cRead(unsigned char slave_addr, unsigned char *buf_ptr,
unsigned char length,unsigned char word_addr);
unsigned char I2cCurrentRead(unsigned char slave_addr, unsigned char
*buf_ptr);
```

参考文献

1. The I²C-Bus Specification (Version 2.1), January 2000, Phillips Semiconductor.
2. 24AA16/ 24LC16B 16K I²C Serial EEPROM, 2002, Microchip Technology Inc. http://www.microchip.com/download/lit/pline/memory/ic/21703b.pdf
3. Zhen Jiang, I²C Interface With The Hitachi SH Microprocessor (Revision 1.1), 21 Jan 1998, Hitachi Micro System, Inc.
4. http://www.esacademy.com/faq/i2c/
5. H8/38024 Series, H8/38024F-ZTAT Hardware Manual (version 2.0), 20 Feb 2002, Hitachi Ltd.
6. Leonard Haile, Hitachi H8/3437 Series Microcontroller I²C Peripheral-A practical SMBus/I²C Firmware Design Guide (Revision 1.2), 12 June 1998, Hitachi Semiconductor (America) Inc.

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。