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 SLP シリーズ

## 3 つの I²C デバイスへの I²C (ポート) 実装 (3I²C ポート)

### 要旨

本アプリケーションノートでは I²C バスインタフェースの概要を説明します。また複数の I²C デバイスをどのように統合化し、2 本の汎用 I/O ピンをソフトウエア制御し，どのように H8/38024 SLP シリーズにインタフェースできるかを実証します。

- Microchip 社製の 24AA16 16K I²C シリーズ EEPROM (読み出し/書き込み)
- Maxim 社製の MAX6626 12 ビット温度センサ (読み出し/書き込み)
- Maxim 社製の MAX6953EPL 2 線インタフェース 4 桁 5 x 7 マトリクス LED ディスプレイドライバ (書き込み専用)

### 動作確認デバイス

H8/38024 SLP

### 目次

## 1. I2C™ インタフェースの概要

I2C バスではシリアルデータライン (SDA) とシリアルクロックライン (SCL) で構成される 2 線のインタフェースを使用し，バスに接続されたデバイス間で情報を交換します。バス上の各デバイスは，それぞれ特定のアドレスを持ち，送信デバイスまたは受信デバイスとして機能できます (その特定機能により異なります)。さらに，デバイスはマスタまたはスレーブに分類されます。マスタは転送を開始し，制御し (すべてのフレーミング信号やクロック信号を生成する)，また停止するデバイスとして定義されます。一方，スレーブデバイスはマスタデバイスによって指定された所望のデバイスです。

I2C デバイスによってプロトコルが多少異なるので注意してください。本アプリケーションノートではバスマスタ (H8/38024 SLP MCU) と 3 つの異なるスレーブデバイス (Microchip 社製 24AA16 16K I2C シリアル EEPROM，Maxim 社製 MAX6626 12 ビット温度センサ，MAX6953EPL 2 線インタフェース 4 桁 5 x 7 マトリクス LED ディスプレイドライバ) から構成される I2C インタフェースを説明します。この簡易インタフェースでは、MAX6626 で変換した温度を表示し，E2PROM に格納することができます。I2C インタフェースは、H8/38024 SLP MCU の 2 本の汎用 I/O ピン (P70→SDA と P80→SCL) を制御するためにソフトウエアを使用してシミュレーションされるので注意してください。これら 3 つのデバイスの SDA ピンと SCL ピンはそれぞれ MCU の P70 と P80 に直接接続されます。図 1 がそのシステムブロック図です。

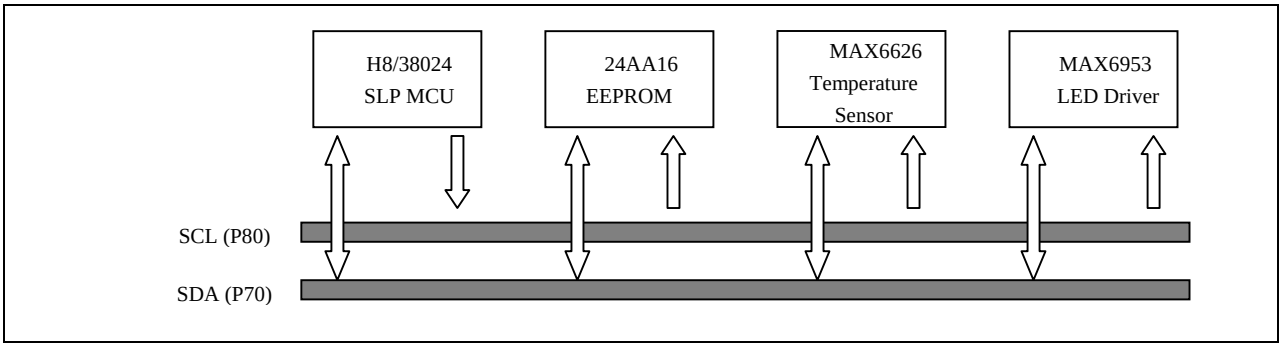



**図 1　システムブロック図**

各デバイスのアドレスは表 1 に要約されています (I2C の 7 ビットアドレッシングモードに限定します)。各デバイスは固有の 7 ビット I2C アドレスを持っているので，マスタはどのデバイスと通信しているのかを理解できます。通常は，上位アドレスラインが固定され，下位アドレスラインがハードウェアで設定されます。3 つの下位アドレスライン (A2，A1，A0) の場合は、8 通りまでの異なる組み合わせがあります。したがって，最大 8 つまでの同一デバイスを 1 つのバスに一緒にインタフェースすることができます。もう 1 つの I2C デバイスをインタフェースするには，ユニークなアドレスを使い，割り当てられた SDA と SCL ピンをバスに接続する必要があります。

**表 1　デバイスアドレス**

| デバイス | アドレス (16 進数) |
|---|---|
| Microchip 社製の 24AA16 EEPROM | A0 –ブロック 0<br>A2 –ブロック 1<br>A4 –ブロック 2<br>A6 –ブロック 3<br>A8 –ブロック 4<br>AA –ブロック 5<br>AC –ブロック 6<br>AE –ブロック 7 |
| Maxim 社製の MAX6626 温度センサ | 90 |
| Maxim 社製の MAX6953 LED ドライバ | B0 |

これら 3 つのスレーブデバイスの特長とハードウェアの設計，および，ソフトウエアの説明は以下の章で取り上げます。

## 2. Microchip 社製の 24AA16 E2PROM

Microchip 社の 24AA16 は，2 線インタフェースの 256×8 ビットメモリからなる 8 ブロックで構成される 16 K ビットの EEPROM です。16 バイトまでのページ書き込みをサポートします。図 2 にそのブロック図を示します。

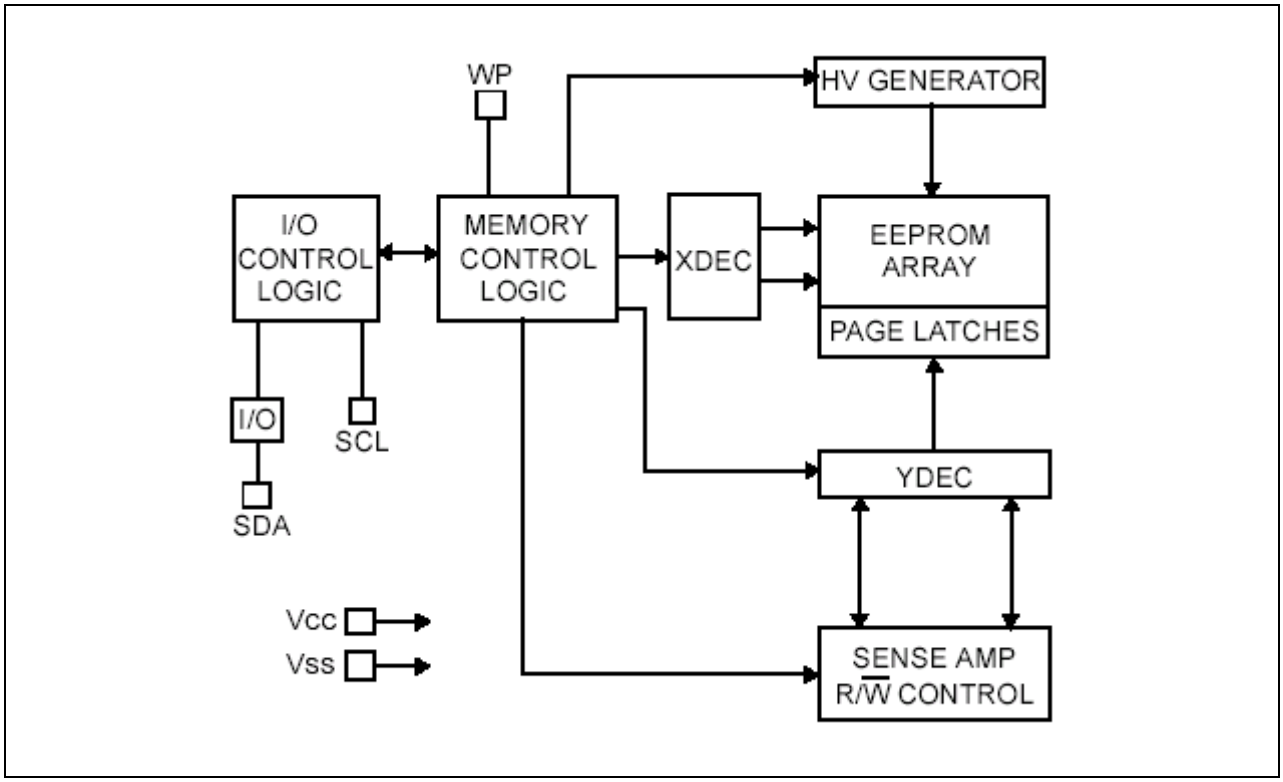



**図 2　Microchip 社製 24AA16 E2PROM のブロック図**

### 2.1　バスプロトコル

I2C バス上でのデータ転送は，バスマスタが発する 2 つの固有のバス状態により制御され (またフレーム化され) ます。これらのバス状態は開始，停止条件です。バスが開放されているとき，両ラインは High になります。

SCL ラインが High レベルの状態で，SDA が High レベルから Low レベルへ変化したときが開始条件となります。また，SCL ラインが High の状態で，SDA が Low レベルから High レベルへ変化したときが停止条件となります。データは SCL が High の間，SDA ライン上では常に有効 (安定) である必要があります。SDA ラインは，SCL の Low 期間中にだけ変更することが許されています。1 つの SCL クロックパルスで，　1 つのデータビットが送信されます。

開始条件に引き続きバス上のメッセージで送られる最初の 8 ビットは，スレーブアドレスフィールドとそれに伴うデータ方向または R/$\overline{W}$ ビットです。データ方向ビット (最下位ビット) では，マスタが送信 (0 = 書き込み)，または受信 (1 = 読み出し) するのを制御します。

アクノリッジビットは，マスタが送信したスレーブからのクロックパルス (バイト送信の 9 番目の High レベル SCL クロックパルス) の期間内に，受信デバイス (マスタまたはスレーブ) から SDA ライン上に送信される Low レベル信号です。スレーブがビジー状態でデータの受信が不可能な場合，または，マスタがデータ送信の終了を通知する必要がある場合，ノンアクノリッジが送信されます (9 番目の High SCL クロックパルス期間，SDA が High)。

開始条件とスレーブアドレスの送信に次いで，データは必要に応じてマスタと受信デバイスとの間で交換されます。最終バイトとそのアクノリッジの交換が終了すると，マスタはバスの使用を終了する停止条件を送信します。

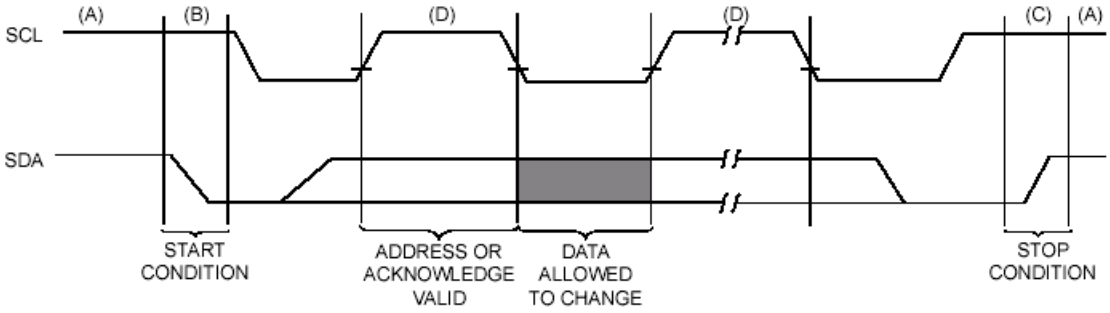
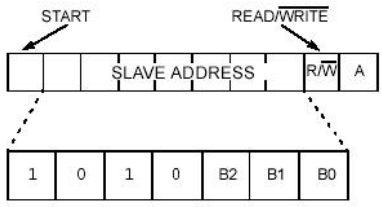



**図 3　データ転送シーケンス**

### 2.1.1　デバイスのアドレッシング

制御バイトは，マスタデバイスからの開始条件に続いて受信する最初のバイトです。24AA16 の場合，読み出しと書き込みの両動作において，制御コードの最初の 4 ビットが 2 進数 1010 に設定されています。次の 3 ビットは，ブロック選択ビット (B2，B1，B0) です。メモリのどの 256 ワードブロックをアクセスするのかを選択するために使用します。これらビットは，実際はワードアドレスの MSB の 3 ビットです。プロトコルにより，メモリのサイズが 1 ブロック 256 ワードの 8 ブロックに制限されるので注意してください。したがって，プロトコルはシステムごとに 1 つの 24AA16 しかサポートできません。制御バイトの最後のビットは，実行する動作を定義します。「1」に設定すると，読み出し動作が選択されます。「0」に設定すると，書き込み動作が選択されます。24AA16 は，開始条件に続き，SDA バスをモニタし，送信されているデバイスタイプ識別子をチェックします。1010 コードを受信すると，スレーブデバイスは SDA ラインにアクノリッジ信号を出力します。R/$\overline{W}$ ビットの状態によって 24AA16 は，読み出しまたは書き込み動作を選択します。




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

**図 4　制御バイト**

### 2.1.2 ビット転送とデータの有効性

開始条件と停止条件の間で送信デバイスから受信デバイスに転送されるデータバイト数は，マスタデバイスによって決定されます。各バイト (8 ビット) は MSB から連続的に転送され，アクノリッジビットが続きます。開始条件後，クロック信号が High の期間，データラインが安定しているとき，データライン上のデータは有効であることを示します。データライン上のデータはクロック信号が Low の間に変更しなければなりません。データ 1 ビットに対し，1 クロックパルスが存在します。

### 2.1.3 アクノリッジ

各受信デバイスは，指定され各バイトを受信後，アクノリッジを生成しなければなりません。またマスタデバイスは，このアクノリッジビットに呼応する追加のクロックパルスを生成しなければいけません。24AA16 では，内部のプログラミングサイクルが実行中であれば，アクノリッジビットを生成しません。

## 2.2 書き込み動作

スレーブアドレスの R/$\overline{W}$ ビットが「0」に設定されると，書き込み動作を開始します。書き込み動作にはバイト書き込みとページ書き込みの 2 種類があります。

### 2.2.1 バイト書き込み

バイト動作は EEPROM のランダムなアドレスに書き込みを行います。バイト書き込み動作は以下の送信を必要とします。

- 開始条件 (マスタ)
- EEPROM デバイスアドレス (マスタ) と R/$\overline{W}$ = 0
- 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)
- アクノリッジビット (EEPROM)
- 書き込まれるデータバイト (マスタ)
- アクノリッジビット (EEPROM)
- 停止条件 (マスタ)

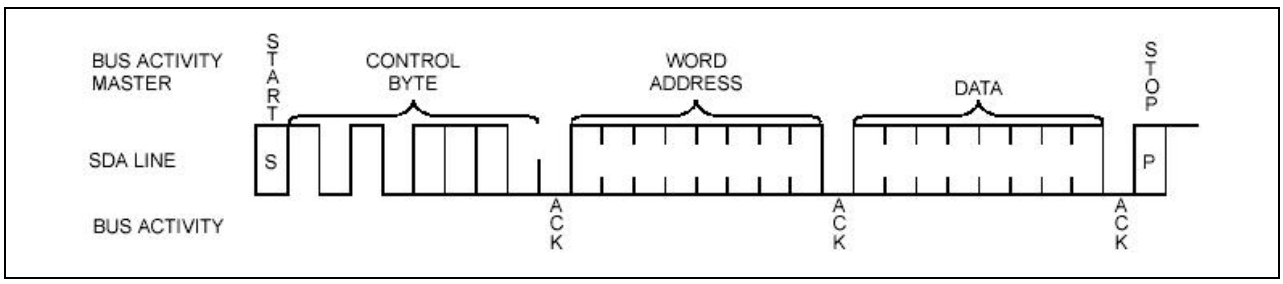



**図 5 バイト書き込み**

### 2.2.2 ページ書き込み

ページ書き込み動作では，16 ビットまで EEPROM に書き込みできます。ページ書き込みをする間，EEPROM はバイト間の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。

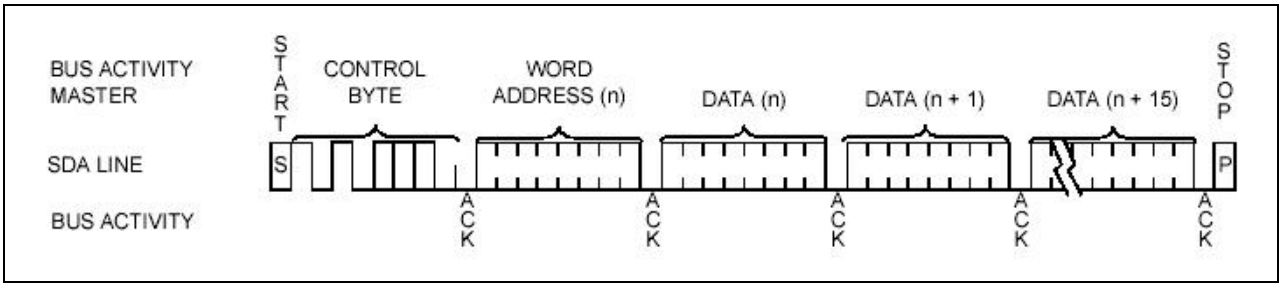



**図 6 ページ書き込み**

## 2.3 読み出し動作

現行アドレス，ランダム読み出し，順次読み出しの3種類の読み出し動作をサポートします。読み出し動作は，R/$\overline{W}$ ビットがデバイスのアドレスバイトで1に設定される以外は，書き込み動作と同じです。

### 2.3.1 現行アドレス読み出し

現行アドレス読み出しモードでは，最新のアクセス位置からデータが読み出されます。この現行読み出しシーケンスは以下のとおりになります。

- 開始条件 (マスタ)
- EEPROM デバイスアドレス (マスタ) と R/$\overline{W}$ = 1
- アクノリッジビット (EEPROM)
- 読み出すデータバイト (1 インクリメントされたスレーブの最新のメモリアドレスから送信された EEPROM バイト)
- 非アクノリッジビット (マスタ)
- 停止条件 (マスタ)

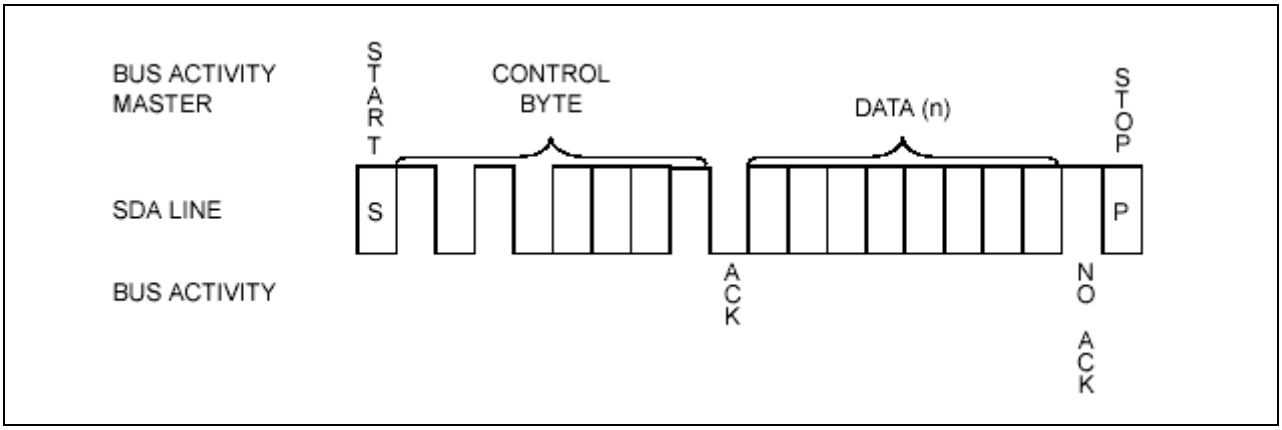



**図 7 現行アドレスの読み出し**

### 2.3.2 ランダム読み出し

ランダム読み出しモードは，前述したように，ダミーバイト書き込みサイクルで開始され，現行アドレス書き込みモードサイクルが続きます (開始条件を送信し，マスタはデバイスアドレスと対象ワードアドレスに続きます)。

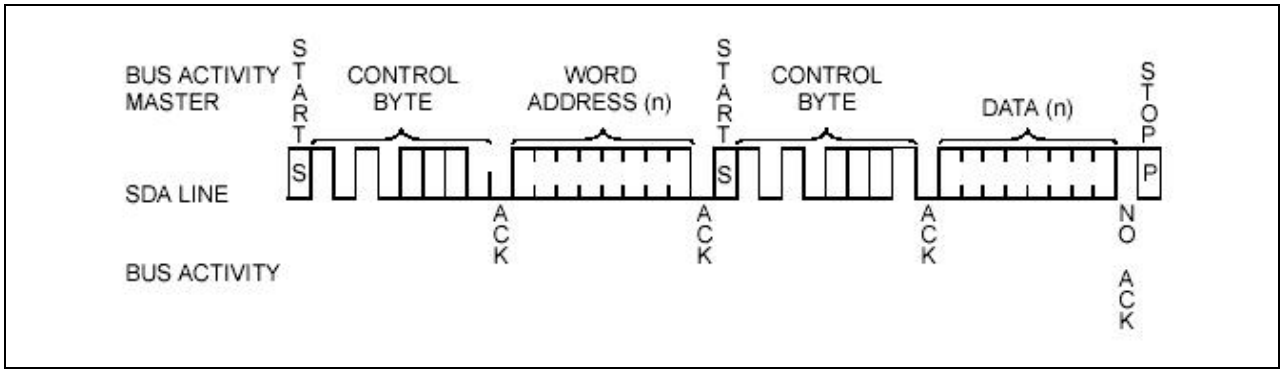



**図 8　ランダム読み出し**

### 2.3.3 順次読み出し

順次読み出しモードは，ランダム読み出しにより開始されます。マスタは，(非アクノリッジで) 1 バイト転送後，読み出しを終了する代わりに，各データバイトの受信後に，有効なアクノリッジを返します。このアクノリッジによって，スレーブ EEPROM は読み出し動作を継続して，次のデータバイトを送信します。順次読み出しは，最新バイトの読み出し後，マスタが非アクノリッジを発行し，停止条件に続きます。

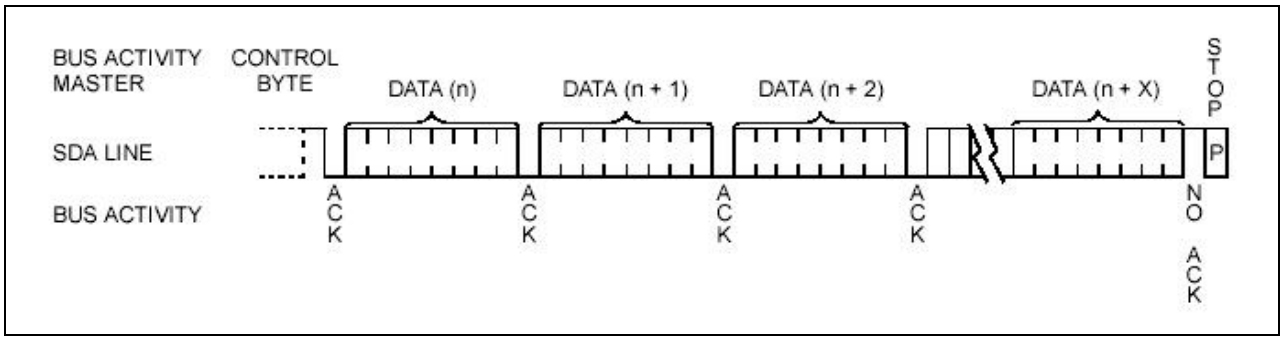



**図 9　順次読み出し**

## 3. I2C 温度センサ

MAX6626 は，温度センサ，プログラム可能な温度過昇アラーム，I2C 互換のシリアルインタフェースなどで構成されています。チップの温度は内蔵された A/D 変換器を使用してデジタル値に変換されます。変換結果は，温度レジスタに格納され，シリアルインタフェースを介していつでも読み出しできます。変換結果が，プログラム可能な高温度レジスタの値を越えた場合，専用アラーム出力 (OT) が起動します。そのレジスタは，アラームが起動するまでに発生するフォルト数を自由に設定できるフォルトエラー数を格納できます。これはノイズの多い環境の中でアラームの異常トリップを防止します。このデバイスはスレーブとして機能し，バイト/ワード単位の読み出し/書き込みコマンドをサポートします。機能ブロック図を図 10 に示します。

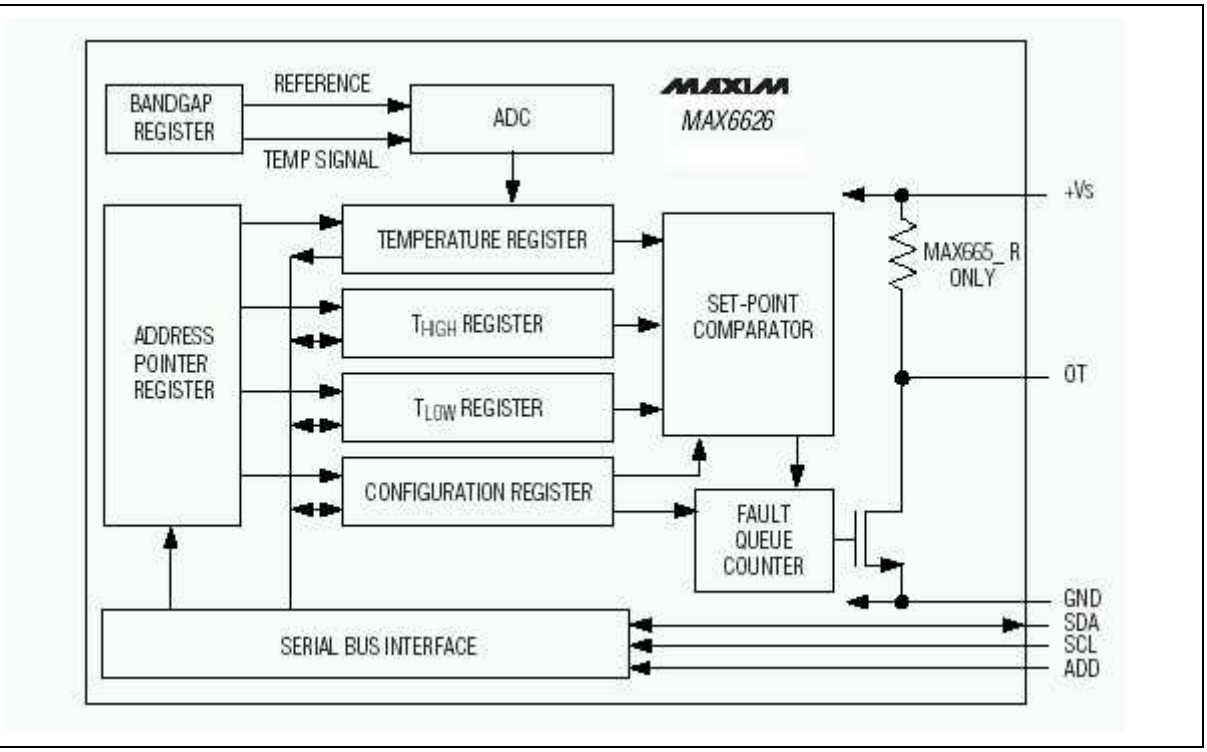



**図 10　温度センサのブロック図**

### 3.1　アドレッシング

ADD ピンにより 4 つの異なるアドレスが設定でき，最大で 4 つの MAX6626 を同じバスに接続できます。表 2 は異なるアドレス選択を要約したものです。このインタフェースでは，ADD ピンが GND に接続されアドレスは 90 (16 進数) に設定されます。

**表 2　ADD 接続**

| ADD 接続 | I2C-互換アドレス |
|---|---|
| GND | 100 1000 |
| $V_S$ | 100 1001 |
| SDA | 100 1010 |
| SCL | 100 1011 |

## 3.2 制御レジスタ

動作は以下のレジスタで定義されます。

**ポインタ**レジスタは，動作させるレジスタを決定するために，最初に設定されます。

**表 3 ポインタレジスタ**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | レジスタ |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **温度** |
| | | | | | | 0 | 1 | **構成** |
| | | | | | | 1 | 0 | $T_{LOW}$ |
| | | | | | | 1 | 1 | $T_{HIGH}$ |

**温度** (TEMP) レジスタは 12 ビットの読み出し専用レジスタで，最新の温度データを保持します。レジスタの長さは 16 ビットで，未使用のビットは 0 にマスクされています。デジタル温度は 2 つの補数形式を使い°C で表され，その LSB は 0.0625°C に対応します。

**表 4 温度レジスタ**

| D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2-D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MSB **(符号)** | **ビット** 11 | **ビット** 10 | **ビット** 9 | **ビット** 8 | **ビット** 7 | **ビット** 6 | **ビット** 5 | **ビット** 4 | **ビット** 3 | **ビット** 2 | **ビット** 1 | LSB | **未使用ビット** 0 |

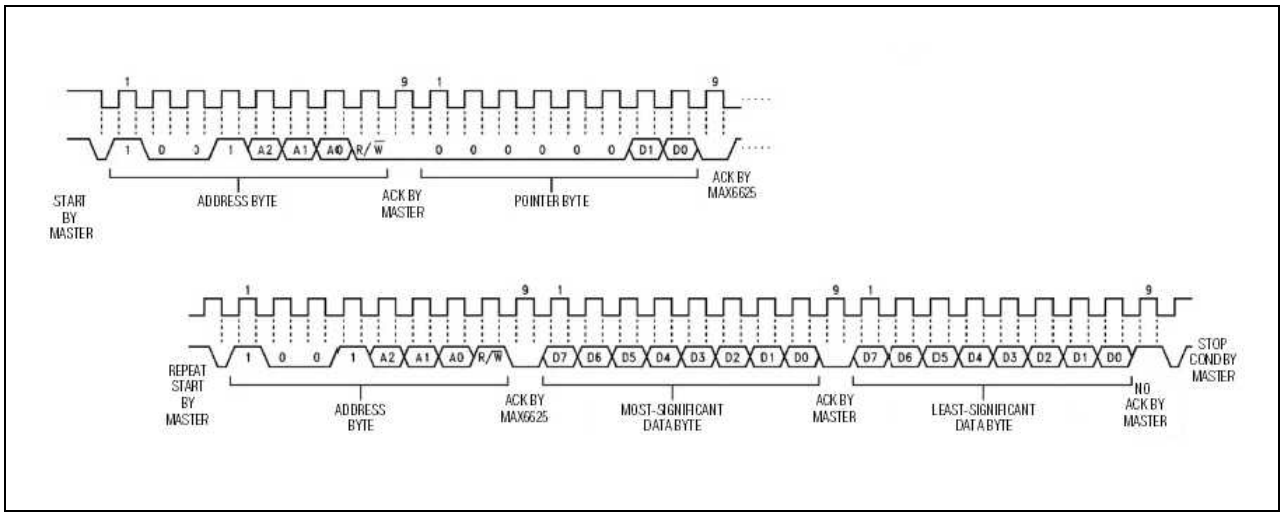



**図 11 2 バイトレジスタの読み出し** (TEMP，$T_{HIGH}$，$T_{LOW}$)

構成レジスタは読み出し/書き込み可能な 8 ビットのレジスタです。また，このレジスタはフォルト待ち行列の項目数，温度アラーム極性選択，割り込みモード選択，シャットダウン制御ビットを含みます。表 5 のビット構造を参照してください。

表 5　構成レジスタ

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | フォルト数 |
|---|---|---|---|---|---|---|---|---|
| | | | フォルト待ち行列の項目数 | | OT 極性 | コンパレータまたは割り込みモード | シャットダウン | |
| 0 | 0 | 0 | 0 | 0 | 0: アクティブ Low<br>1: アクティブ High | 0: コンパレータ<br>1: 割り込み | 0: 通常動作<br>1: シャットダウン | 1 |
| | | | 0 | 1 | | | | 2 |
| | | | 1 | 0 | | | | 4 |
| | | | 1 | 1 | | | | 6 |

図 12 と図 13 に構成レジスタからの読み出しと構成レジスタへの書き込みのタイミングチャートを示します。

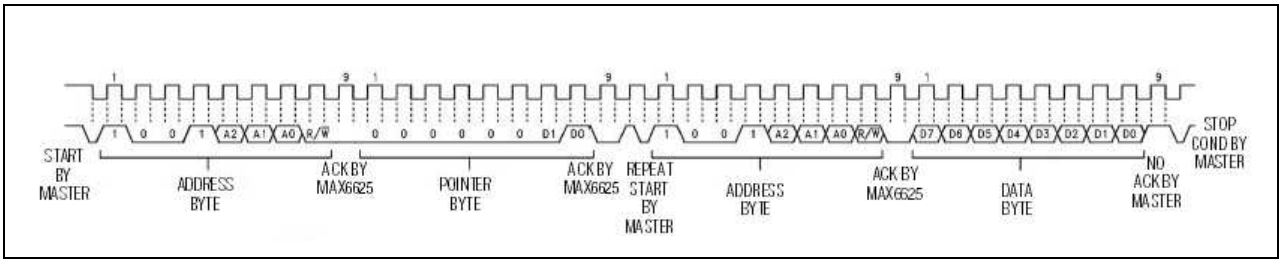



図 12　構成レジスタからの読み出し

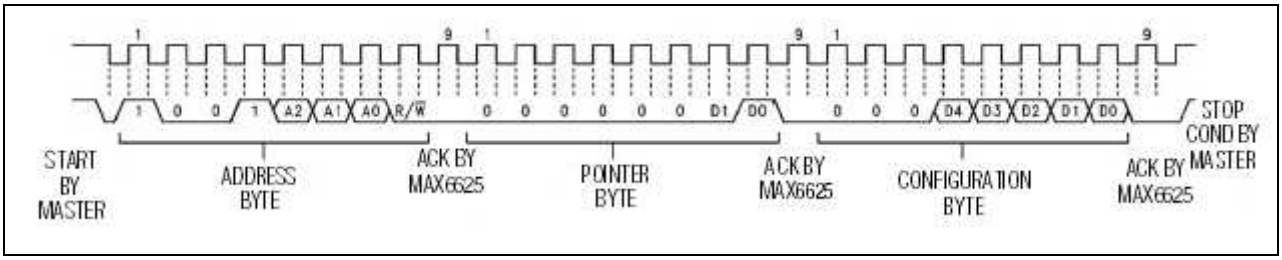



図 13　構成レジスタへの書き込み

***高温度*** ($T_{HIGH}$) レジスタは読み出し/書き込み可能な 9 ビットのレジスタです。また，このレジスタに過大温度アラームの値が含まれています。***低温度*** ($T_{LOW}$) レジスタは，読み出し/書き込み可能な 9 ビットのレジスタです。このレジスタは過大温度アラームがコンパレータモードでデアサートされる前に，低下すべき温度を含みます。これら 2 つのレジスタのビット構成に関しては，表 6 を参照してください。読み出しと書き込みのタイミングチャートを、それぞれ図 11 と図 14 に示します。

表 6　$T_{HIGH}$ と $T_{LOW}$ のレジスタ

| D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MSB | ビット 7 | ビット 6 | ビット 5 | ビット 4 | ビット 3 | ビット 2 | ビット 1 | LSB | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

注：1. D15：MSB は符号ビットです。
2. D6～D0：すべて 0 を読み出し、書き込みはできません。
3. LSB = 0.5°C

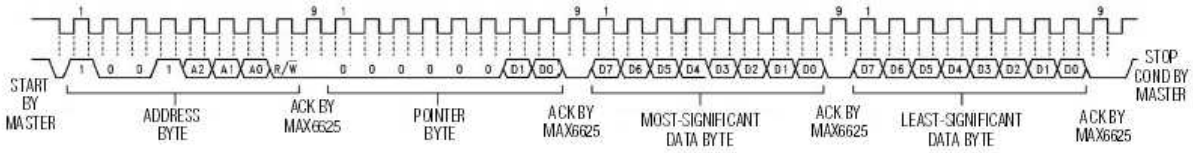



図 14　$T_{HIGH}$ と $T_{LOW}$ の書き込み

## 3.3　温度変換

内蔵バンドギャップを用い，A/D 変換に必要な安定した温度の基準電圧と，絶対温度 (PTAT) に比例する信号が生成されます。デジタル変換された PTAT 信号の分解能は 0.0625°C です。その変換レートは 133 ms です。温度レジスタには最新の変換値が格納されています。

## 3.4　過大温度アラーム

専用の過大温度出力ピン (OT) の極性とモード (割り込みとコンパレータ) は，構成レジスタで設定できます。フォルト待ち行列の項目数はアラームの動作を決めます。

- フォルト待ち行列を自由に設定し，OT アラームが起動する前に起こる許容範囲外の温度の連続読み出し回数を設定することにより，ノイズの多い環境の中でアラームの異常トリップ動作を回避することができます。許容範囲外とは，読み出し温度が $T_{HIGH}$ 以上か $T_{LOW}$ 以下のことです。
- コンパレータモードでは，変換値が $T_{HIGH}$ レジスタの値を連続的に越えた回数が行列の項目と一致したとき，OT 信号がアサートされます。変換値が $T_{LOW}$ レジスタの値を連続的に下回った回数が行列の項目と一致したとき、OT 信号がデアサートされます。たとえば $T_{HIGH}$，$T_{LOW}$，フォルト待ち行列の項目数がそれぞれ+75°C，+50°C，4 に設定されるとします。変換 4 回連続が+75°C を越えるまで OT 信号はアサートされません。同じように、4 回の連続変換が+50°C より低くなるまで OT はデアサートされません。コンパレータモードは，マスタの介入なしで OT フォルトの自発的なクリアが可能です。これは冷却ファンの駆動に理想的です。
- 割り込みモードでは，特定の条件のもとで，OT 信号により，過大温度と過少温度フォルトのアラームがアサートされます。電源投入時に，フォルト待ち行列がクリアされた場合，IC は $T_{LOW}$ フォルトのモニタ後 $T_{HIGH}$ フォルトを検索します。$T_{LOW}$ フォルトの後は，$T_{HIGH}$ フォルトをモニタします。OT 信号が正しくデアサートされるたびにこの処理は繰り返されます。どちらかのフォルトが発生すると，所望のレジスタの読み出しによってデアサートされるまで，IC はアサート状態を保持します。次に，デバイスは反対のタイプのフォルトをモニタします。たとえば $T_{HIGH}$，$T_{LOW}$，フォルト待ち行列の項目数が+75°C，+50°C，4 にそれぞれ設定されているとします。OT 信号は変換 4 回連続して+75°C を越えるまでアサートされません。そして，OT 信号は温度レジスタの読み出しでデアサートされます。さらに，OT ピンは+50°C より低い変換 4 回連続した後にアサートされます。

## 3.5 シャットダウン

シャットダウンモードでは，温度レジスタは H’8000 に設定されます。また，A/D 変換器はオフです (デバイスの電流が 1 μA 低減します)。シャットダウン状態から起動すると，最初の温度変換が完了するまで，温度レジスタの値は H’8000 です。

## 4. I2C 4 桁 5 × 7 マトリクス LED ディスプレイドライバ

MAX6953 は，カソード列 5×7 ドットマトリクスディスプレイの 4 桁を駆動できるシリアルインタフェースのディスプレイドライバです。このドライバは，ユーザが定義できる 24 文字のフォントデータのほかに， ASCII コードの 104 の文字フォント，マルチプレックススキャン回路，行ドライバ，列ドライバ，および各桁を格納するための SRAM などで構成されています。LED のセグメント電流は，内部で桁ごとに輝度制御をデジタル的に設定できます。また，低電力シャットダウンモード，セグメント点滅，全 LED を ON にするテストモードなどの機能もあります。図 15 に LED ドライバの機能ブロック図を示します。

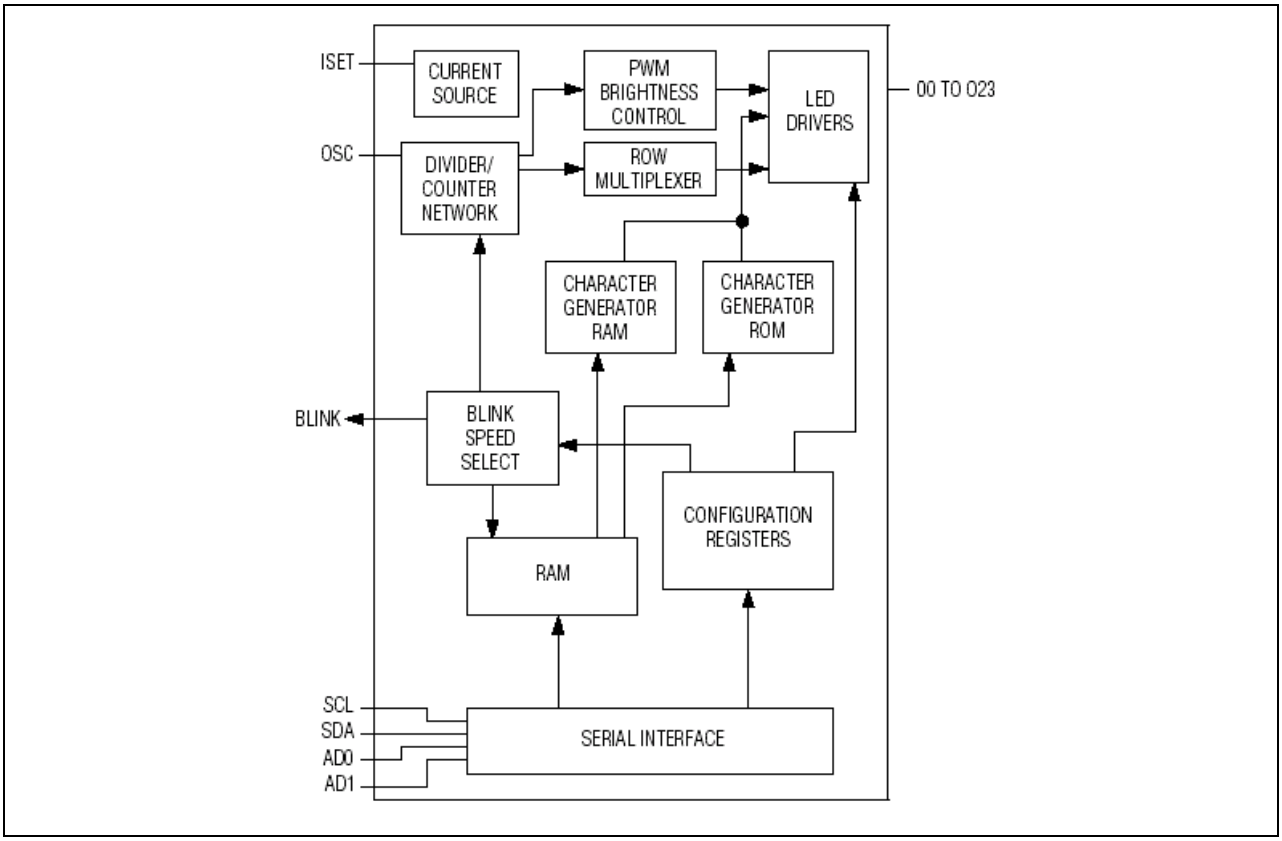



**図 15　MAX6953 のブロック図**

### 4.1　シリアルアドレッシング

MAX6953 は I2C 互換 2 線式インタフェースを介してデータを送受信するスレーブとして動作します。シリアルデータ (SDA) とクロック (SCL) のラインは，マスタ (H8/38024) とスレーブ (MAX6953) の間で双方向通信を実現するために使用されます。マスタは MAX6953 との間の，すべてのデータのやりとりを行い，データ転送の同期に使用する SCL クロックを生成します。

### 4.2　開始条件と停止条件

インタフェースがビジーでない場合、SCL と SDA は High の状態のままです。マスタは，SCL が High の状態のとき，SDA を High から Low に変化することで，送信を開始します。マスタがスレーブとの通信を終了するときは、SCL が High の状態のとき SDA を Low から High に変化することで停止条件を発行します。停止条件発行後に，他のバス通信を開始できます。

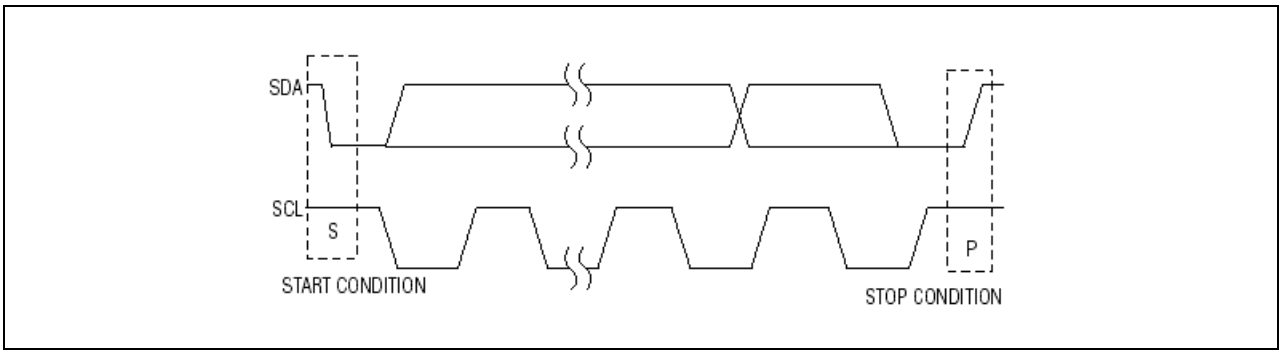



**図 16　開始条件と停止条件**

## 4.3　ビット転送

1 クロックにつき 1 ビットのデータが転送されます。SDA ライン上のデータは SCL が High の間，安定した状態を維持しなければいけません。

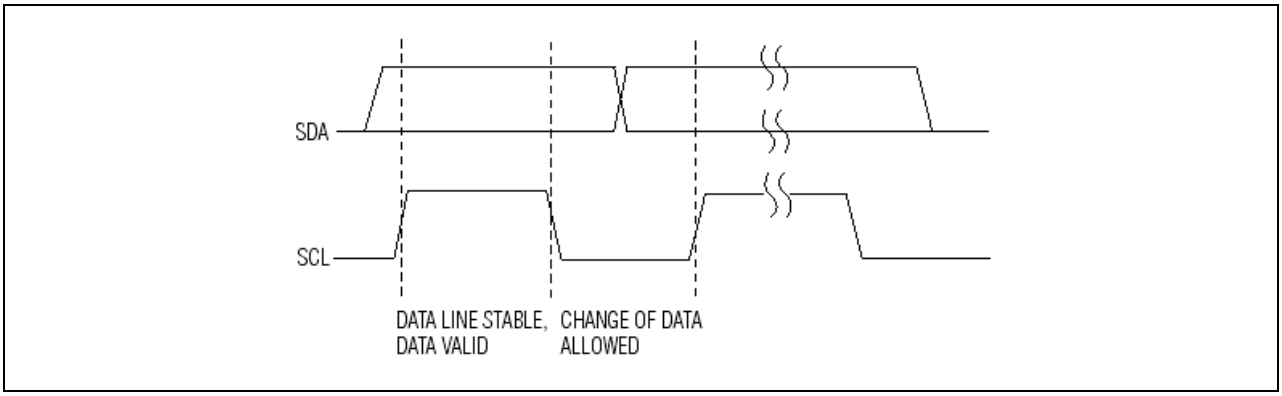



**図 17　ビット転送**

## 4.4　アクノリッジ

アクノリッジビットは，受信デバイスが各データバイトをハンドシェークし，受信するために使用します。クロックに同期した 9 番目のビットです。図 18 を参照してください。したがって，各転送バイトは事実上 9 ビットを必要とします。マスタは 9 番目のクロックパルスを生成します。そしてアクノリッジ用クロックパルスが High の期間 SDA ラインが Low に安定するように，受信デバイスは SDA をプルダウンします。マスタが MAX6953 に送信しているときは，アクノリッジビットは MAX6953 から送信されます。MAX6953 がマスタに送信しているときは，マスタが受信デバイスなのでマスタがアクノリッジビットを送信します。

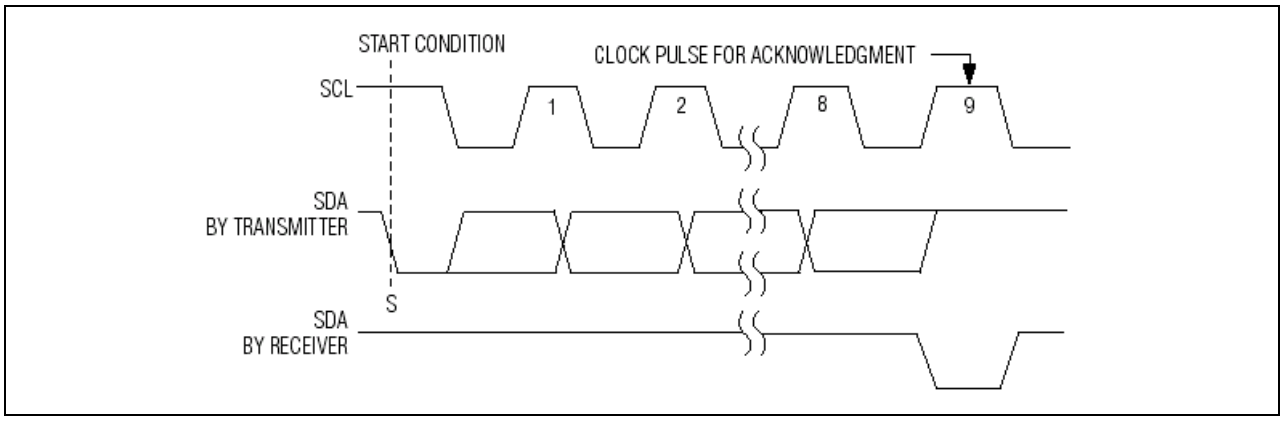



**図 18　アクノリッジ**

## 4.5　スレーブアドレス

MAX6953 は 7 ビットのスレーブアドレスです。7 ビットのスレーブアドレスに続く 8 番目のビットは R/$\overline{W}$ ビットです。書き込みコマンドでは Low に，読み出しコマンドでは High になります。MAX6953 スレーブアドレスの最初の 3 ビット (A6，A5，A4) は常に 101 です。アドレス入力ピン AD1 と ADO によって，スレーブアドレスビットの A3，A2，A1，A0 の値が決まります。AD1 と ADO は GND，V+，SDA，または SCL に接続することができます。表 7 では可能なすべての AD1，AD0 接続と，MAX6953 が指定される対応アドレスを示します。A0～AE (16 進数) のアドレスはすでに EEPROM に割り当てられ，B0 のアドレスは MAX6953 に割り当てられることに注意してください。

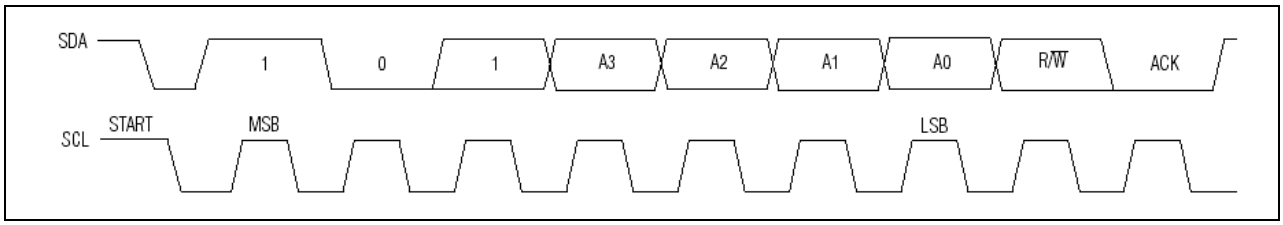



**図 19　スレーブアドレス**

表 7　MAX6953 デバイスマップ

| PIN | | DEVICE ADDRESS | | | | | | |
|---|---|---|---|---|---|---|---|---|
| AD1 | AD0 | A6 | A5 | A4 | A3 | A2 | A1 | A0 |
| GND | GND | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| GND | V+ | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| GND | SDA | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| GND | SCL | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| V+ | GND | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| V+ | V+ | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| V+ | SDA | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| V+ | SCL | 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| SDA | GND | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| SDA | V+ | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| SDA | SDA | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| SDA | SCL | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| SCL | GND | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| SCL | V+ | 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| SCL | SDA | 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| SCL | SCL | 1 | 0 | 1 | 1 | 1 | 1 | 1 |

## 4.6　書き込みメッセージ形式

MAX6953 への書き込みに続いて，0 に設定された R/$\overline{W}$ のスレーブアドレスが送信され，少なくとも 1 バイトの情報が続いて送信されます。最初のバイト情報はコマンドバイトで，次のバイトによって，書き込むべきレジスタを決定します。コマンドバイトの受信後，停止条件が検出された場合，コマンドバイトを格納し動作は終了します(図 20)。

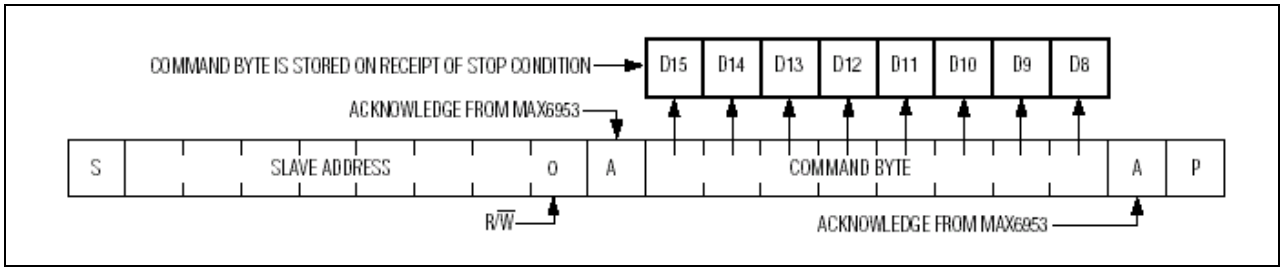



図 20　受信コマンドバイト

コマンドバイト後に受信されるバイトはすべてデータバイトになります。最初のデータバイトは，コマンドバイトで選択された MAX6953 の内部レジスタに格納されます (図 21)。

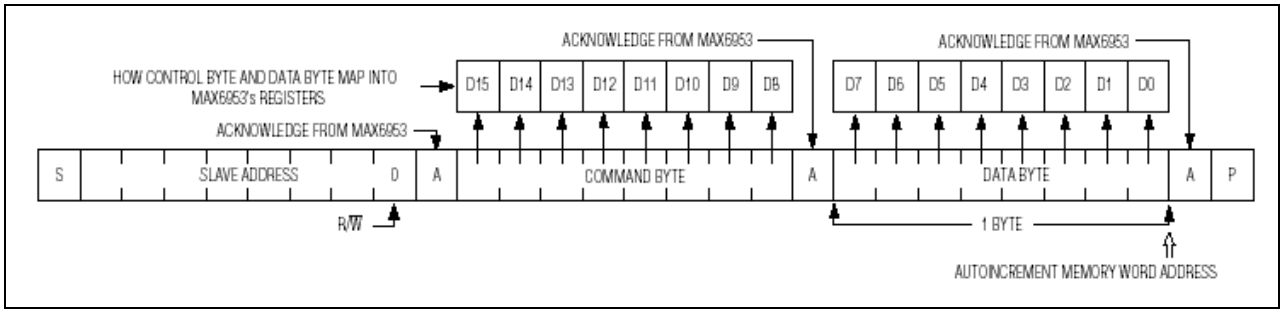



図 21　受信されたコマンドバイトと 1 データバイト

停止条件が検出される前に複数のデータバイトが送信される場合，通常，コマンドバイトアドレスは自動的にインクリメントされるので，これらのデータバイトは次の MAX6953 の内部レジスタに格納されます (表 8 と図 22 を参照してください)。

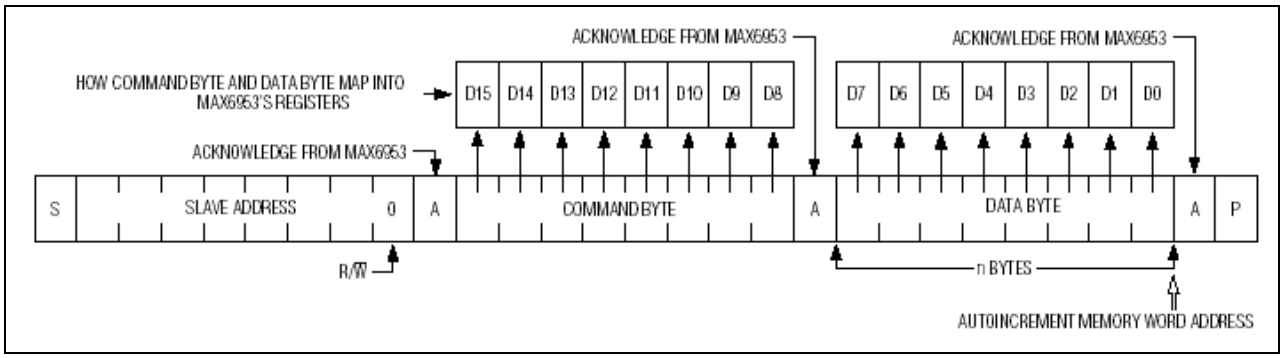



**図 22　受信された n データバイト**

**表 8　コマンドアドレスの自動インクリメント方式**

| COMMAND BYTE ADDRESS RANGE | AUTOINCREMENT BEHAVIOR |
|---|---|
| x0000000 to x0000100 | Command byte address autoincrements after byte read or written. |
| x0000101 | Command byte address remains at x0000101 after byte read or written, but the font address pointer autoincrements. |
| x0000110 | Factory reserved; do not write to this register. |
| x000111 to x1111110 | Command byte address autoincrements after byte read or written. |
| x1111111 | Command byte address remains at x1111111 after byte read or written. |

## 4.7　読み出しメッセージ形式

格納されたコマンドバイトは書き込み時にアドレスポインタとして使用されるのと同様，格納されたコマンドバイトは読み込み時にアドレスポインタとして使用されます。通常，表 8 で説明した同じ方式で各データバイトが読み出された後，このアドアレスポインタは自動的にインクリメントされます。したがって，読み出し動作は，書き込み動作により開始されます (図 20)。マスタは，最初に初期化されたコマンドバイトで指定されたレジスタからデータバイトを読み出した後，引き続き連続して MAX6953 から n バイトを読み出すことができます (図 22)。

## 5. プログラムリスト

機能は、以下の 2 つの C ソースファイルに記述されています。

- I2C.c
  - main 関数が含まれています。
  - シリアル通信インタフェース (SCI) と温度センサを初期化します。
  - EEPROM，温度センサ，LED ドライバをテストします。
- RW.c
  - SDA と ACL をエミュレーションするための一般機能が含まれています。

main 関数のフローチャートは図 23 のとおりで，以下の動作が行われます。

1. SCI (2400 bps，1 ストップビット，パリティディセーブル)，温度センサ，LED ドライバを初期化します。
2. バイト書き込み，バイト読み出し，ページ書き込み，現行アドレスと順次アドレスからの読み出しなど EEPROM をテストします。テスト結果は SCI を経由し PC に送信されます。
3. 温度を測定し，SCI を経由し PC に送信します。そして 1，2，3 の桁に表示します。
4. 0 の桁に「0」から「9」を表示します。
5. 2 から 4 の動作を繰り返します。

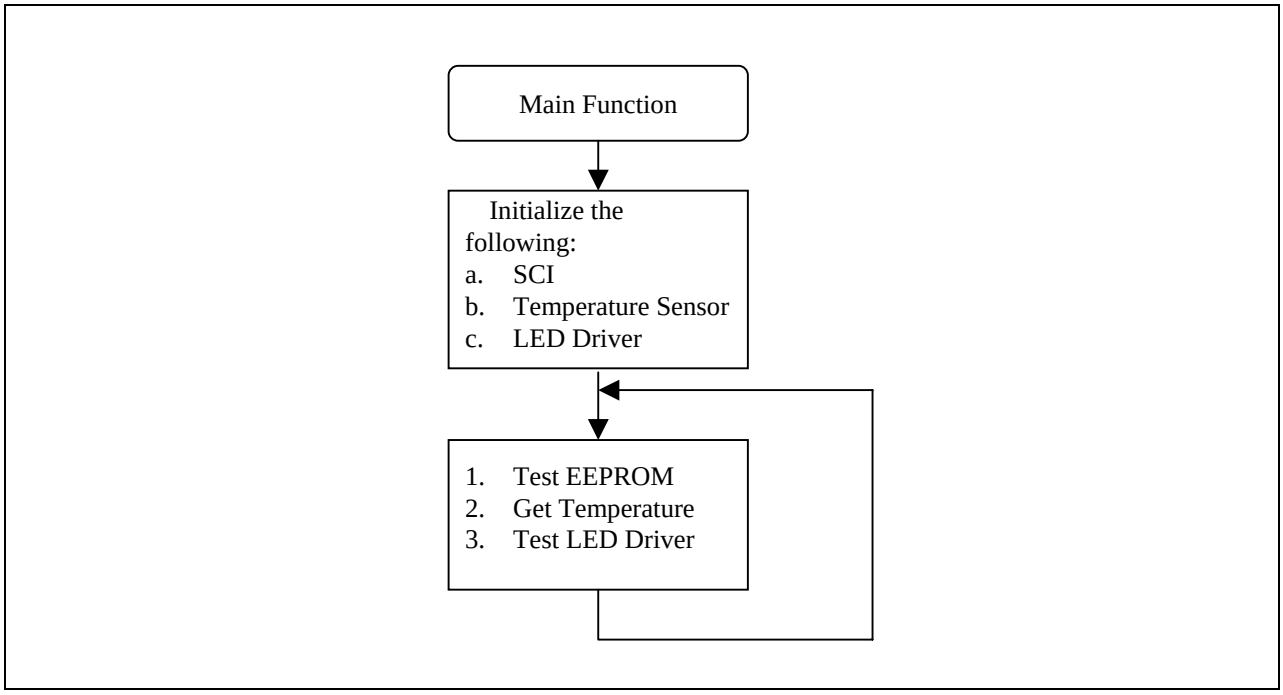



**図 23　main 関数のフローチャート**

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

#include "iodefine.h"
#include "i2c.h"
#include <stdio.h>
#include <machine.h>

//---------------------------------------------------------------------------

//Device Addresses
#define  EEPROM_ADDR         0xA0  //B'10100000x
#define  T_SENSOR_ADDR       0x90  //B'10010000x
#define  LED_DRIVER_ADDR     0xB0  //B'10110000x

//---------------------------------------------------------------------------

//LED Driver Registers
#define  DIGIT_0             0x60
#define  DIGIT_1             0x61
#define  DIGIT_2             0x62
#define  DIGIT_3             0x63

#define  DIGIT_0_1_INT_REG   0x01
#define  DIGIT_2_3_INT_REG   0x02

//---------------------------------------------------------------------------

/*
   main()

   a. Initializes Serial Communication Interface (SCI) for debugging
   b. Initializes temperature sensor
   c. Initializes LED driver
   d. Repeat the following
      1. Test the EEPROM
      2. Obtain temperature reading
      3. Test the LED Driver
*/

void main(void)
{
   init_sci();

   init_temp_sensor();
```

```
   init_led_driver();

   PutStr("\r\nBeep Beep Beep");

   while(1)
   {
      test_eeprom();
      test_temp_sensor();
      test_led_matrix();
      wait(5); //short delay
   }
}

//---------------------------------------------------------------------

/*
   test_led_matrix() – display 0 to 9 on Digit 0
*/

void test_led_matrix(void)
{
   char  display_char;

   for (display_char = '0' ; display_char <= '9' ; display_char++)
   {
      LEDprint(display_char, DIGIT_0);
      wait(10);   //short delay
   }
}

//---------------------------------------------------------------------

/*
   test_eeprom()

   a. byte write
   b. byte read
   c. page write
   d. current address read
   e. sequential address read
*/

void test_eeprom(void)
{
   unsigned char buf[16] = {0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
                            0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF};

   unsigned char  return_byte;
   unsigned char  *ptr;

   PutStr("\r\n\nEEPROM Testing:");
```

```
    //Byte Write
    PutStr("\r\nByte Write");
    if (I2cWrite(EEPROM_ADDR, buf, 1, 0x00) != OP_DONE)
       PutStr(" -> Fail!");
    else
       PutStr(" -> OK");

    //Need to check if write is complete
    if (CheckWriteReady() == 1)
    {
       //Byte Read
       PutStr("\r\nByte Read");
       return_byte = I2cRead(EEPROM_ADDR, ptr, 1, 0x00);
    }

    //Page Write
    PutStr("\r\nPage Write");
    if (I2cWrite(EEPROM_ADDR, buf, 16, 0x00) != OP_DONE)
       PutStr(" -> Fail!");
    else
       PutStr(" -> OK");

    //Need to check if write is complete
    if (CheckWriteReady() == 1)
    {
       //Current Address Read
       PutStr("\r\nCurrent Address Read");
       return_byte = I2cCurrentRead(EEPROM_ADDR, buf, 0x00);

       //Sequential Read
       PutStr("\r\nSequential Read");
       return_byte = I2cRead(EEPROM_ADDR, ptr, 16, 0x00);
    }
}

//---------------------------------------------------------------------

/*
   test_temp_sensor()

   a. Get temperature reading
   b. Convert from binary to floating point
   c. Transmit temperature to PC via SCI
   d. Display temperature on Digits 1, 2 and 3
*/

void test_temp_sensor(void)
{
   unsigned char  return_byte;
   unsigned char  tens, ones, tenths;

   unsigned int   return_code;
```

```
    float       degree;

    //read from temperature sensor
    return_code = I2cRead_T_Sensor(T_SENSOR_ADDR, 0x00);

    if (return_code == 0x8000)
       PutStr("SHUT DOWN!");
    else
       PutStr("\r\n\nTemperature : ");

    degree = ConvertBinary2Temp(return_code);

    //For example, temperature = 37.1 degree
    //tens = 3, ones = 7 & tenths = 1
    tens = 0;
    ones = 0;
    tenths = 0;

    while (degree >= 10)
    {
       tens++;
       degree -= 10;
    }

    while (degree >= 1)
    {
       ones++;
       degree -= 1;
    }

    while (degree >= 0.1)
    {
       tenths++;
       degree -= 0.1;
    }

    //Transmit to PC via SCI
    char_put(tens + 0x30);
    char_put(ones + 0x30);
    char_put(0x2E);           //decimal point
    char_put(tenths + 0x30);

    //Display on dot-matrix LED
    LEDprint(tens + 0x30, DIGIT_1);
    LEDprint(ones + 0x30, DIGIT_2);
    LEDprint(tenths + 0x30, DIGIT_3);
}

//---------------------------------------------------------------------

/*
   init_temp_sensor()
*/
```

```
void init_led_driver(void)
{
   unsigned char  return_byte;
   //Configure MAX6953 driver: wake from shutdown mode
   SendStartBit();
   SendByte((LED_DRIVER_ADDR) & 0xfe);
   SendByte(0x04);   //configuration register
   SendByte(0x01);   //select normal operation as power-on default ->
shutdown
   SendStopBit();

   //Set the intensity register for digits 0 & 1 to 6/16 duty cycle
   //Set the intensity register for digits 2 & 3 to 6/16 duty cycle
   //Write 0x66 to both 0x01 and 0x02 reg of MAX6953EPL
   SendStartBit();
   SendByte((LED_DRIVER_ADDR) & 0xfe);   //Send Slave Address byte
   SendByte(DIGIT_0_1_INT_REG);          //Send COMMAND byte
   SendByte(0x66);                       //Send data byte 00-min, FF-max
   SendStopBit();                        //Send stop bit
   SendStartBit();
   SendByte((LED_DRIVER_ADDR) & 0xfe);   //Send Slave Address byte
   SendByte(DIGIT_2_3_INT_REG);          //Send COMMAND byte
   SendByte(0x66);                       //Send data byte 00-min, FF-max
   SendStopBit();                        //Send stop bit
}

//---------------------------------------------------------------------

/*
   init_temp_sensor()
*/

void init_temp_sensor(void)
{
   unsigned char  return_byte;

   SendStartBit();
   SendByte((T_SENSOR_ADDR) & 0xfe);     //Send slave address byte
   SendByte(0x01);                       //Configuration Register of sensor
   SendByte(0x00);                       //Wake up device
   SendStopBit();                        //Send stop bit

   SendStartBit();
   SendByte((T_SENSOR_ADDR) & 0xfe);     //Send slave address byte
   SendByte(0x01);                       //Configuration Register of sensor
   SendStopBit();

   SendStartBit();
   SendByte((T_SENSOR_ADDR) | 0x01);     //Read from Configuration Register
   return_byte = GetByte();
   SendStopBit();

   //THIGH = 80 degrees
```

```
    //TLOW  = 0 degrees
    SendStartBit();
    SendByte((T_SENSOR_ADDR) & 0xfe);     //Send Slave Address byte
    SendByte(0x03);                       //Set Max Temperature of Sensor
    SendByte(0x50);                       //msbByte
    SendByte(0x00);                       //lsbByte
    SendStopBit();                        //Send stop bit

    SendStartBit();
    SendByte((T_SENSOR_ADDR) & 0xfe);     //Send slave address byte
    SendByte(0x02);                       //Set Min Temperature of Sensor
    SendByte(0x00);                       //msbByte
    SendByte(0x00);                       //lsbByte
    SendStopBit();                        //Send stop bit
}

//---------------------------------------------------------------------

/*
    This routine is written for MAXIM 12-bit Temperature Sensors.
    MAX6626 is a 12-bit i2c compatible sensors.

    input:
    unsigned char slave_addr - refer to the address preset
    unsigned char ptr_reg - refer to the pointer register
        0x00 temperature
        0x01 configuration
        0x02 high-temperature
        0x03 low-temperature

    return:
    unsigned int - current temperature in 16bits
*/

unsigned int I2cRead_T_Sensor(unsigned char slave_addr, unsigned char
ptr_reg)
{
    unsigned int theWORD;
    unsigned char msbBYTE, lsbBYTE;

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

    SendStartBit();

    //Send slave address with write command
    if (SendByte((slave_addr) & 0xfe) != LOW)
        return(NO_RESPONSE);

    //Send Pointer byte
    if (SendByte(ptr_reg) != LOW)
        return(NO_RESPONSE);

    SendStopBit();      //Send STOP bit
```

```
    SendStartBit();

    //Send slave address with read command
    if (SendByte((slave_addr) | 0x01) != LOW)
       return(NO_RESPONSE);

    msbBYTE = GetByte();

    SendBit(LOW);             //Ack it low!

    lsbBYTE = GetByte();

    SendStopBit();            //Send STOP bit

    theWORD = (unsigned int)msbBYTE << 8;
    theWORD = theWORD + lsbBYTE;

    return(theWORD);
}

//---------------------------------------------------------------------

/*
   ConvertBinary2Temp() Converts temperature from binary to floating point
*/

float ConvertBinary2Temp(unsigned int temp)
{
   float degree;
   float scaleMX;
   int   temp1;

   scaleMX = 0.0625;

   temp1 = temp & 0x7FFF;    //throw away signed bit
   temp1 = temp1>>4;   //get rid of last 4 bits(lsb)

   degree = (float)temp1 * scaleMX;

   return(degree);
}

//---------------------------------------------------------------------

/*
   LEDprint(): Display on the matrix LED.
*/

void LEDprint(char character, unsigned char digit_position)
{
   unsigned char  error_code;

   error_code = I2cMatrixLEDdriver(LED_DRIVER_ADDR, digit_position,
```

```
                                    character);
}

//---------------------------------------------------------------------

/*
   This routine is used for MAXIM Matrix LED Display Driver.
   MAX6953 is a 2-wire I2C interface driver.

   slave_addr is the address preset for MAX6953.

   command_byte refer to the command instruction to be given to MAX6953.

   data_byte refer to the 8-bit data
*/

unsigned char I2cMatrixLEDdriver(unsigned char slave_addr, unsigned char
                                    command_byte, unsigned char data_byte)
{
   /*
   Command Address:

   StartBit [S] -> Slave Address (7bit + 1 R/W bit) -> ACK (MAX6953) ->
   COMMAND Byte -> ACK (MAX6953) -> DATA byte -> ACK (MAX6953) -> StopBit
[P]

   Refer to MAX6953 data sheet for command and data instruction
   */

   //Check if I2C bus is busy
   if (CheckBusState() != TRUE)
      return(BUS_BUSY);

   SendStartBit();                        //Send start bit
   //Send slave address and write command
   if (SendByte((slave_addr) & 0xfe) != LOW)
      return(NO_RESPONSE);

   if (SendByte(command_byte) != LOW)    //Send COMMAND byte
      return(NO_RESPONSE);

   if (SendByte(data_byte) != LOW)       //Send DATA byte
      return(NO_RESPONSE);

   SendStopBit();                         //Send stop bit
}

//---------------------------------------------------------------------

/*
   init_sci() : Sets up the Serial Communication Interface for debugging
*/

void init_sci(void)
```

```
{
   //SCR3 : |TIE|RIE|TE|RE|MPIE|TEIE|CKE1|CKE0|
   //TIE : Transmit interrupt enable
   //RIE : Receive interrupt enable
   //TE  : Transmit enable
   //RE  : Receive enable
   //MPIE : Multiprocessor interrupt enable
   //TEIE : Transmit end interrupt enable
   //CKE1 : Clock enable 1
   //CKE0 : Clock enable 0

   //CKE1 = CKE0 = 0
   //asynchronous mode, internal clock source, SCK32 functions as I/O port
   P_SCI3.SCR3.BYTE &= 0x00; //clear TE & RE

   //SMR : |COM|CHR|PE|PM|STOP|MP|CKS1|CKS0| : |0|0|0|0|0|0|0|0|
   //COM : Communication Mode  : 0 : asynchronous mode
   //CHR : Character Length    : 0 : character length = 8 bits
   //PE  : Parity Enable       : 0 : parity bit addition and checking
disabled
   //PM  : Parity Mode         : 0 : even parity (no effect since no
parity)
   //STOP: Stop Bit Length     : 0 : 1 stop bit
   //MP  : Multiprocessor Mode : 0 : multiprocessor comm function disabled
   //|CKS1|CKS0| : Clock Select: |0|0| : clock source for baud rate gen =
clk
   P_SCI3.SMR.BYTE = 0x00;

   //For clk = 10MHz, bit rate = 2400 bps, n = 0, N = 64
   P_SCI3.BRR = 64;

   //minimum of 1-bit delay = 417ns
   nop();
   nop();
   nop();

   //SPCR : |---|---|SPC32|---|SCINV3|SCINV2|---|---| : |1|1|1|0|0|0|0|0|
   //SPC32 = 1 : P42 functions as TXD32 output pin
   //need to set TE bit in SCR3 after setting this bit to 1
   //SCINV3 = 0 : TXD32 output data is not inverted
   //SCINV2= 0 : RXD32 input data is not inverted
   //Bits 7 and 6 are reserved and always read as 1
   //Bits 4, 1 and 0 are reserved and only 0 can be written to these bits
   P_SCI3.SPCR.BYTE = 0xE0;

   P_SCI3.SCR3.BYTE |= 0x30; //Set TE & RE
}

//---------------------------------------------------------------------

/*
   char_put() : Transmits a character to the PC for debugging purposes.
*/

void char_put(char OutputChar)              //Serial Port
{
```

```
    //SSR  : |TDRE|RDRF|OER|FER|PER|TEND|MPBR|MPBT|
    //TDRE : transmit data register empty
    //RDRF : receive data register full
    //OER  : overrun error
    //FER  : framing error
    //PER  : parity error
    //TEND : transmit end
    //MPBR : Multiprocessor bit receive
    //MPBT : Multiprocessor bit transfer
    while ((P_SCI3.SSR.BIT.TDRE) == 0);        //Wait for TDRE = 1

    P_SCI3.TDR = OutputChar;
}

//---------------------------------------------------------------------

/*
    PutStr() : Transmits a string of characters to the PC for debugging
purposes.
*/

void PutStr(char *str)
{
    while (*str != 0)
    {
        char_put(*str++);
    }
}

//---------------------------------------------------------------------

/*
    wait(): Generates a software delay.
*/

void wait(unsigned int time)
{
    unsigned int   i, j;

    for (i = 0 ; i < time ; i++)
    {
        for (j = 0 ; j < 3500 ; j++)
        {
        }
    }
}

//---------------------------------------------------------------------
```

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

//----------------------------------------------------------------------

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

//----------------------------------------------------------------------

/*
   SclIn()
  Defines the SCL as an input pin and checks the port status (low or high).
*/

unsigned char SclIn(void)
{
   SCL_IO_REG &= SCL_IO_RESET_BIT;       //Set to Input

   if (SCL_DATA_REG & SCL_DATA_SET_BIT)  //Check pin status
   {
         return(HIGH);
   }
   else
   {
      return(LOW);
   }
}

//----------------------------------------------------------------------

/*
   SdaIn()
  Defines the SDA as an input pin and checks the port status (low or high).
*/

unsigned char SdaIn(void)
{
   SDA_IO_REG &= SDA_IO_RESET_BIT;            //Set to Input

   if (SDA_DATA_REG & SDA_DATA_SET_BIT)
   {
      return(HIGH);                           //Check pin status
   }
   else
```

```
    {
        return(LOW);
    }
}

//---------------------------------------------------------------------

/*
    SclOut()
    Defines the SCL pin as an output pin and sets it to the level
    determined by the parameter.
*/

void SclOut(unsigned char status)
{
    if (status == LOW)
    {
        SCL_DATA_REG = 0;                   //Drive Port LOW
    }
    else
    {
        SCL_DATA_REG = 1;                   //Drive Port High
    }

    SCL_IO_REG |= SCL_IO_SET_BIT;           //Set to output
}

//---------------------------------------------------------------------

/*
    SdaOut()
    Defines the SDA as an output pin and sets it to the level determined by
    the parameter.
*/

void SdaOut(unsigned char status)
{
    if (status == LOW)
    {
        SDA_DATA_REG = 0;                   //Drive Port LOW
    }
    else
    {
        SDA_DATA_REG = 1;                   //Drive Port High
    }

    SDA_IO_REG |= SDA_IO_SET_BIT;           //Set to output
}

//---------------------------------------------------------------------

/*
    Delay()
```

```
   Provide an internal minimum delay time to bridge the undefined
   region of a falling edge of SCL to avoid unintended generation
   of unwanted signal.
*/

void Delay(void)
{
   unsigned char  i = 0;

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

//---------------------------------------------------------------------

void Delay2x(void)
{
   Delay();
   Delay();
}

//---------------------------------------------------------------------
//All codes below here are independent with hardware, such as
microprocessor, //I/O port, or etc.

/*
   CheckBusState()
Determine whether the I2C bus is free (both SCL and SDA = HIGH) or in busy
state.
*/

unsigned char CheckBusState(void)
{
   if ((SclIn() == HIGH) && (SdaIn() == HIGH))
   {
      return(TRUE);
   }
   else
   {
      return(FALSE);
   }
}

//---------------------------------------------------------------------

/*
   SendStartBit(): Issues a START condition
*/

void SendStartBit(void)
{
```

```
    Delay();
    SdaOut(LOW);
    Delay2x();
    Delay2x();
    Delay2x();
    Delay2x();
    SclOut(LOW);
    Delay();
}

//---------------------------------------------------------------------

/*
    SendBit(): Send out data in bit format
*/

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();
}

//---------------------------------------------------------------------

/*
    GetBit(): Receive data input in bit format
*/

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);
}

//---------------------------------------------------------------------

/*
   GetAck():
   Getting ACK is similar to GetBit, but this is critical operation since
   master must pull SDA high before it finds out whether there is a 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);
}

//---------------------------------------------------------------------

/*
   SendByte(): Send out a byte starting with most significant bit (MSB)
first.
*/

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 >>= 1;
   }
```

```
   return(GetAck());
}

//---------------------------------------------------------------------

/*
   GetByte(): Get a byte of data starting with most significant bit (MSB)
*/

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 >>= 1;
   }
   return(temp2);
}

//---------------------------------------------------------------------

/*
   SendStopBit(): Send a STOP condition to terminate the operation
*/

void SendStopBit(void)
{
   SclOut(LOW);
      Delay();
      SdaOut(LOW);
      Delay();
      SclOut(HIGH);
      Delay2x();
      SdaOut(HIGH);
}

//---------------------------------------------------------------------

/*
   I2cWrite()

   a. Byte Write
      1. Start Bit
      2. Control Byte
      3. Ack
```

```
      4. Word Address
      5. Ack
      6. Data
      7. Ack
      8. Stop Bit

   b. Page Write
      1. Start Bit
      2. Control Byte
      3. Ack
      4. Word Address
      5. Ack
      6. Data(n)
      7. Ack
      8. Data(n + 1)
      9. Ack
         ...
      10.   Data(n + 15)
      11.   Ack
      12.   Stop Bit
*/

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

   if (CheckBusState() != TRUE)
   {
      PutStr(" -> BUS_BUSY!");
      return(BUS_BUSY);
   }
   SendStartBit();

   //Send address and write command
   if (SendByte((slave_addr) & 0xfe) != LOW)
   {
      PutStr(" -> NO_RESPONSE-1");
      return(NO_RESPONSE);
   }

   //Send word address
   if (SendByte(word_addr) != LOW)
   {
      PutStr(" -> NO_RESPONSE-2");
      return(NO_RESPONSE);
   }

   for (i = 0 ; i < length ; i++)
   {
      //Write data
      if (SendByte(*buf_ptr++) != LOW)
      {
```

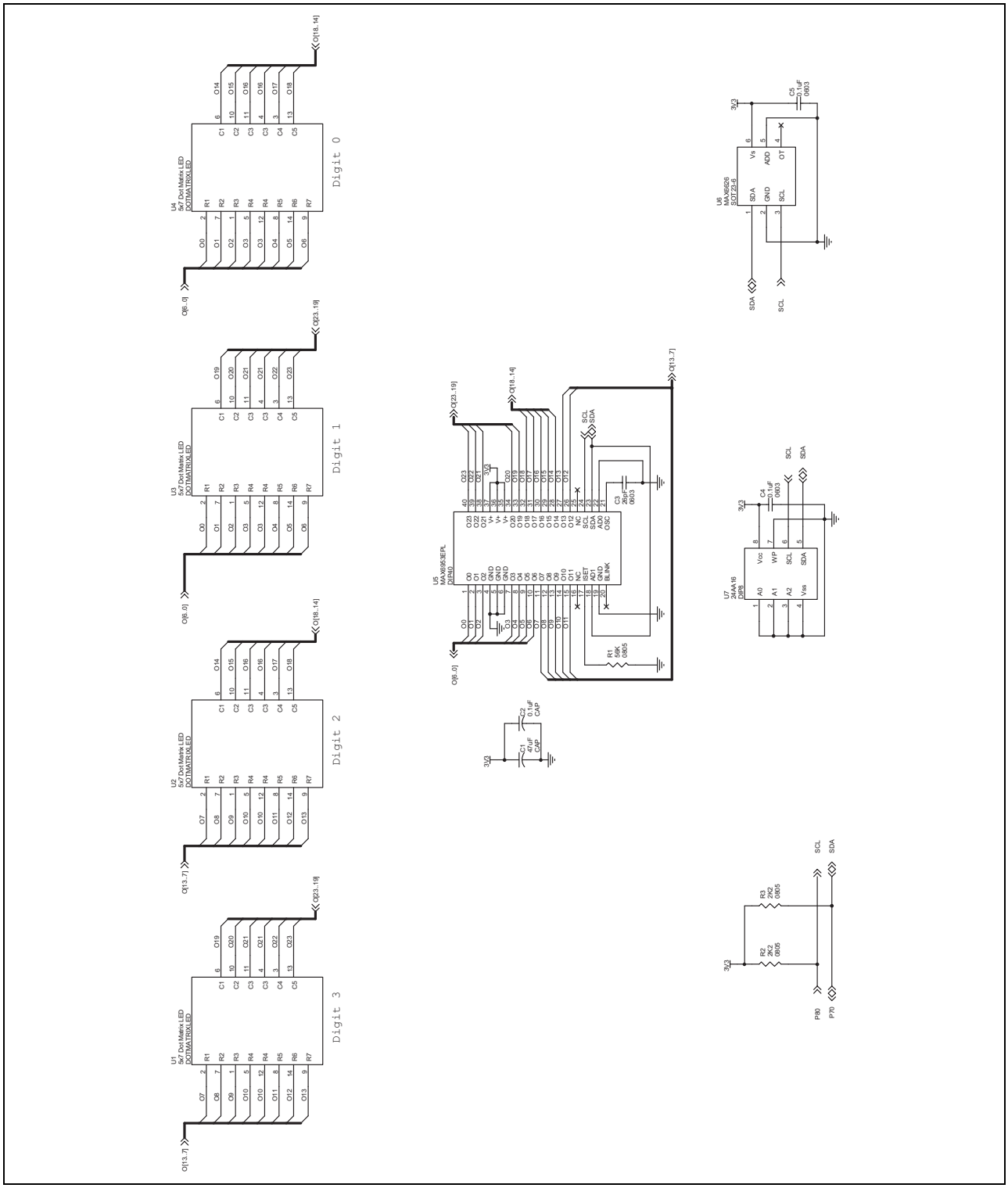
```
            PutStr(" -> ERR_RESPONSE");
            return(ERR_RESPONSE);
         }
      }

      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  ref_data[16] = {0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66,
                                     0x77, 0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xDD,
                                     0xEE, 0xFF};
      unsigned char  DataBuffer[256];
      unsigned char  error = 0;

      if (CheckBusState() != TRUE)
      {
         PutStr(" -> BUS_BUSY");
         return(BUS_BUSY);
      }

      SendStartBit();

      //Send dummy address and write command
      if (SendByte((slave_addr) & 0xfe) != LOW)
      {
         PutStr(" -> NO_RESPONSE-1!");
         return(NO_RESPONSE);
      }

      //Send high word address
      if (SendByte(word_addr) != LOW)
      {
         PutStr(" -> NO_RESPONSE-2!");
         return(NO_RESPONSE);
      }

      SdaOut(HIGH);                    //Pull-up SDA line

      SendBit(HIGH);

      SendStartBit();

      //Send address and read command
      if (SendByte((slave_addr) | 0x01) != LOW)
      {
```

```
      PutStr(" -> NO_RESPONSE-3!");
      return(NO_RESPONSE);
   }

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

   //Get last data byte and ack high
   DataBuffer[length - 1] = GetByte();
   SendBit(HIGH);
   SendStopBit();

   for (i = 0 ; i < length ; i++)
   {
      if (DataBuffer[i] != ref_data[word_addr + i])
      {
         error++;
      }
   }

   if (error)
   {
      PutStr(" -> Incorrect Data!");
   }
   else
   {
      PutStr(" -> OK");
   }

   return(OP_DONE);
}

//---------------------------------------------------------------------

unsigned char I2cCurrentRead(unsigned char slave_addr,
                     unsigned char *buf_ptr,
                     unsigned char word_addr)
{
   unsigned char  ref_data[16] = {0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66,
                                  0x77, 0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xDD,
                                  0xEE, 0xFF};

   SendStartBit();

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

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

   SendBit(HIGH);
   SendStopBit();

   if (*buf_ptr != ref_data[word_addr])
   {
      PutStr(" -> Incorrect Data!");
   }
   else
   {
      PutStr(" -> OK");
   }

   return(OP_DONE);
}

//-------------------------------------------------------------------------

/*
   Since Microchip devices such as 24AA16 will not acknowledge during
   the internal write cycle, this can be used to determined when this
   cycle is complete so that the master can proceed with next operation.

   Acknowledge Polling
   a. Send write command
   b. Send stop condition to initiate write cycle
   c. Send start bit
   d. Send control byte with r/w_n = 0
   e. If device acknowledge, goto f. Else go to c
   f. Ready for next operation

   Note that (c) to (e) - internal write cycle
*/

char CheckWriteReady(void)
{
   unsigned int   i = 0;

   while (i < 4)
   {
      SendStartBit();

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

      SendStopBit();

      i++;
```

```
    }

    return (0);
}

//---------------------------------------------------------------------
```

## 6. ハードウェアの設計



## 7. 参考文献

1. The I²C-Bus Specification (Version 2.1), January 2000, Philips Semiconductor.
2. 24AA16/24LC16B 16K I²C Serial EEPROM, 2002, Microchip Technology Inc.
3. MAX6626 12-bit Temperature Sensor with I²C-compatible Serial Interface, 2002, Maxim Integrated Products.
4. MAX6953 2-wire Interfaced 4-digit 5x7 Matrix LED Display Driver, 2002, Maxim Integrated Products.
5. Serial Peripheral Interface (SPI™) & Inter-IC (I²C™), 2003, Renesas Technology Corp.
   (Application Note ref. no: AN0303011, http://sg.renesas.com,)
6. Application Note on Interfacing to EEPROM with I²C™ Emulation (Port), 2003, Renesas Technology Corp.
   (Application Note ref. no: AN0303012, http://sg.renesas.com,)

## 改訂記録

| Rev. | 発行日 | 改訂内容 | |
|---|---|---|---|
| | | ページ | ポイント |
| 1.00 | 2004.08.06 | — | 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。