致尊敬的顾客

关于产品目录等资料中的旧公司名称

NEC电子公司与株式会社瑞萨科技于2010年4月1日进行业务整合（合并），整合后的新公司暨“瑞萨电子公司”继承两家公司的所有业务。因此，本资料中虽还保留有旧公司名称等标识，但是并不妨碍本资料的有效性，敬请谅解。

瑞萨电子公司网址：http://www.renesas.com

2010年4月1日
瑞萨电子公司

【发行】瑞萨电子公司（http://www.renesas.com）

【业务咨询】http://www.renesas.com/inquiry

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

H8/300L SLP 系列

3 个 I²C 器件的 I²C（端口）安装（3 个 I²C 端口）

要点

本应用说明叙述 I²C 总线接口的概要，并证实如何综合多个 I²C 器件、用软件控制 2 个通用 I/O 管脚以及如何连接 H8/38024 SLP 系列。

- Microchip 公司产的 24AA16 16K I²C 系列 EEPROM（读/写）
- Maxim 公司产的 MAX6626 12 位温度传感器（读/写）
- Maxim 公司产的 MAX6953EPL 的 2 线接口 4 位 5x7 矩阵 LED 显示驱动器（写专用）

动作确认器件

H8/38024 SLP

目录

1. I²C™接口的概要

I²C 总线使用由串行数据线（SDA）和串行时钟线（SCL）构成的 2 线接口，进行总线连接的器件之间的信息交换。总线上的各器件具有各自固有的地址，能作为发送器件或者接收器件（因该特定功能而不同）工作。另外，器件被分为主器件或者从属器件。主器件被定义为进行传送的开始、控制（生成所有的帧信号和时钟信号）和停止的器件，从属器件是由主器件指定的器件。

必须注意：因 I²C 器件而协议有些不同。在本应用说明中，解说由总线主控器（H8/38024 SLP MCU）和 3 个不同的从属器件（Microchip 公司产的 24AA16 16K I²C 串行 EEPROM、Maxim 公司产的 MAX6626 12 位温度传感器和 MAX6953EPL 的 2 线接口 4 位 5x7 矩阵 LED 显示驱动器）构成的 I²C 接口。此简单的接口能显示 AX6626 转换的温度，并将温度保存到 E²PROM。必须注意：为了控制 H8/38024 SLP MCU 的 2 个通用 I/O 管脚（P70→SDA 和 P80→SCL），使用软件仿真 I²C 接口。将这 3 个器件的 SDA 管脚和 SCL 管脚分别直接连接到 MCU 的 P70 和 P80。图 1 为该系统框图。

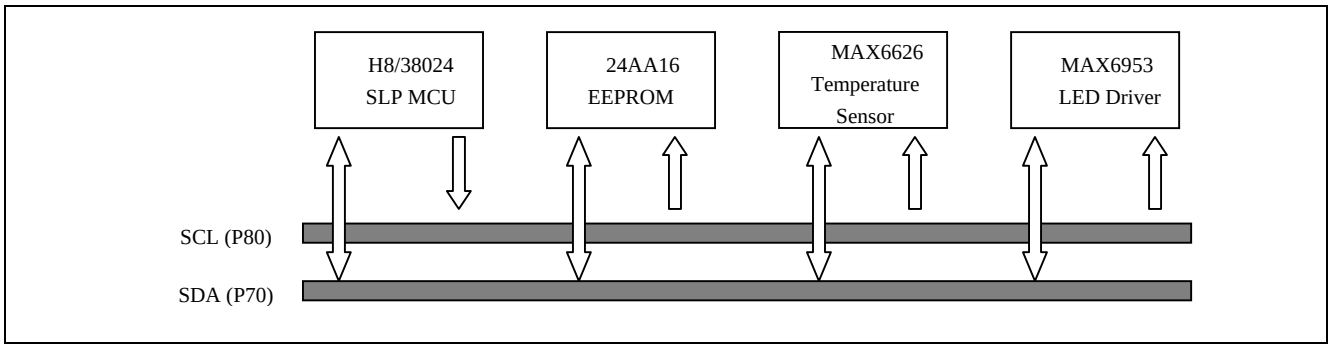


图 1　系统框图

表 1 归纳了各器件的地址（限于 I²C 的 7 位寻址方式）。由于各器件具有固有的 7 位 I²C 地址，所以主器件能知道在和哪个器件通信。通常固定高位地址线，并通过硬件设定低位地址线。在 3 个低位地址线（A2、A1、A0）的情况下，最多有 8 个不同的组合。因此，能最多将 8 个同样器件一起连接到 1 条总线。在连接另 1 个 I²C 器件时，需要使用唯一的地址，将已分配的 SDA 和 SCL 管脚连接到总线。

表 1　器件地址

器件	地址（16 进制数）
Microchip 公司产的 24AA16 EEPROM	A0 –块 0 A2 –块 1 A4 –块 2 A6 –块 3 A8 –块 4 AA –块 5 AC –块 6 AE –块 7
Maxim 公司产的 MAX6626 温度传感器	90
Maxim 公司产的 MAX6953 LED 驱动器	B0

以下的章节说明这 3 个从属器件的特点、硬件的设计以及软件。

2. Microchip 公司产的 24AA16 E²PROM

Microchip 公司的 24AA16 是由 2 线接口的 256×8 位存储器（8 块）构成的 16K 位 EEPROM，支持最多 16 个字节的页写。该框图如**图 2** 所示。

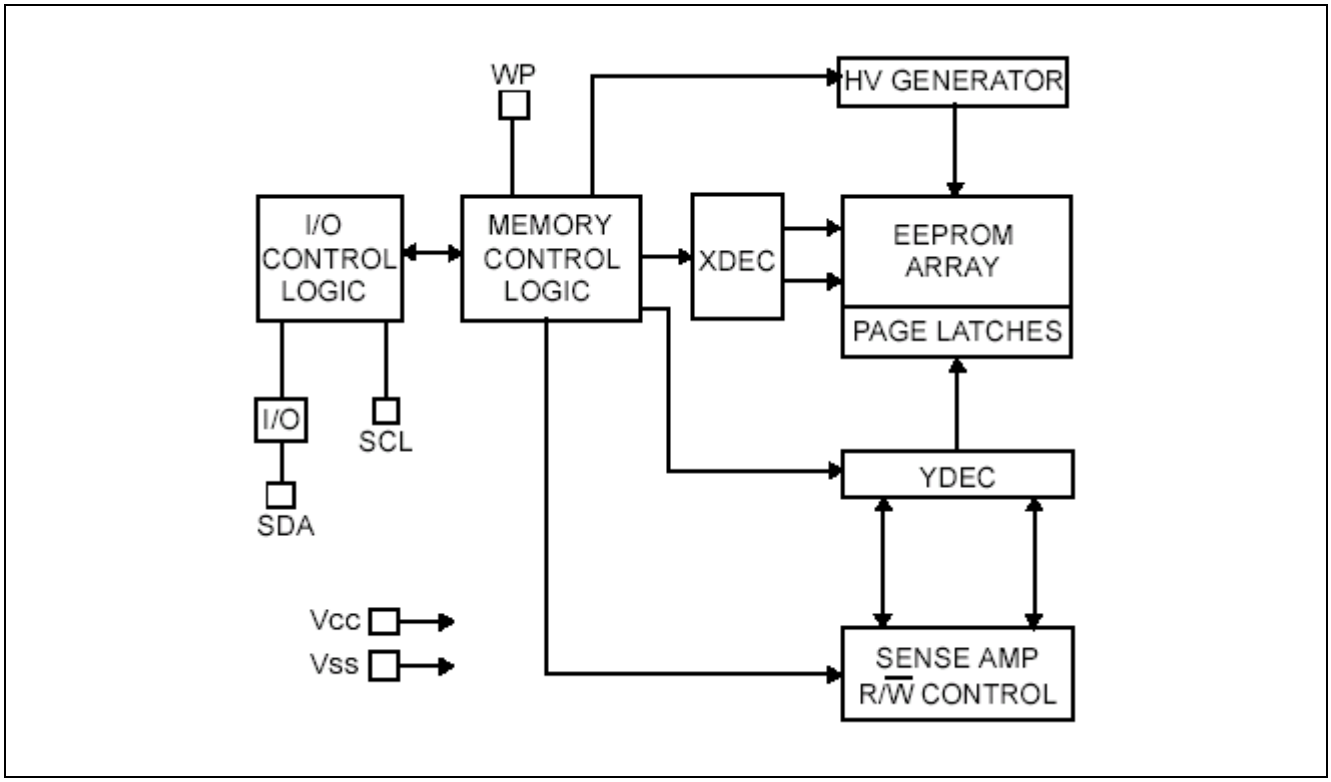


图 2　Microchip 公司产的 24AA16 E²PROM 框图

2.1　总线协议

通过总线主控器发出的 2 个固有的总线状态，控制（帧形成）I²C 总线上的数据传送。这些总线状态是开始和停止条件。在释放总线时，两线都为 High 电平。

在 SCL 线处于 High 电平的状态下，如果 SDA 从 High 电平变为 Low 电平，就为开始条件；另外，在 SCL 线处于 High 电平的状态下，如果 SDA 从 Low 电平变为 High 电平时，就为停止条件。在 SCL 处于 High 电平期间，需要数据在 SDA 线上总是有效（稳定）。只在 SCL 的 Low 电平期间允许更改 SDA 线。1 个 SCL 时钟脉冲发送 1 个数据位。

接着开始条件被发送的总线信息的最初 8 位是从属地址字段和数据方向（R/$\overline{W}$）位。数据方向位（最低位）控制主器件的发送（0=写）或者接收（1=读）。

应答位是在主器件送来的应答时钟脉冲（字节发送的第 9 个 High 电平 SCL 时钟脉冲）期间内，由接收器件（主器件或者从属器件）发送到 SDA 线上的 Low 电平信号。在从属器件因忙而不能发送数据的情况下，或者在主器件需要通知数据发送结束的情况下，发送否认应答（在第 9 个 High 电平 SCL 时钟脉冲期间，SDA 为 High 电平）。

在发送开始条件和从属地址后，根据需要在主器件和接收器件之间进行数据交换。当最后字节及其应答的交换结束时，主器件就发送结束总线使用的停止条件。

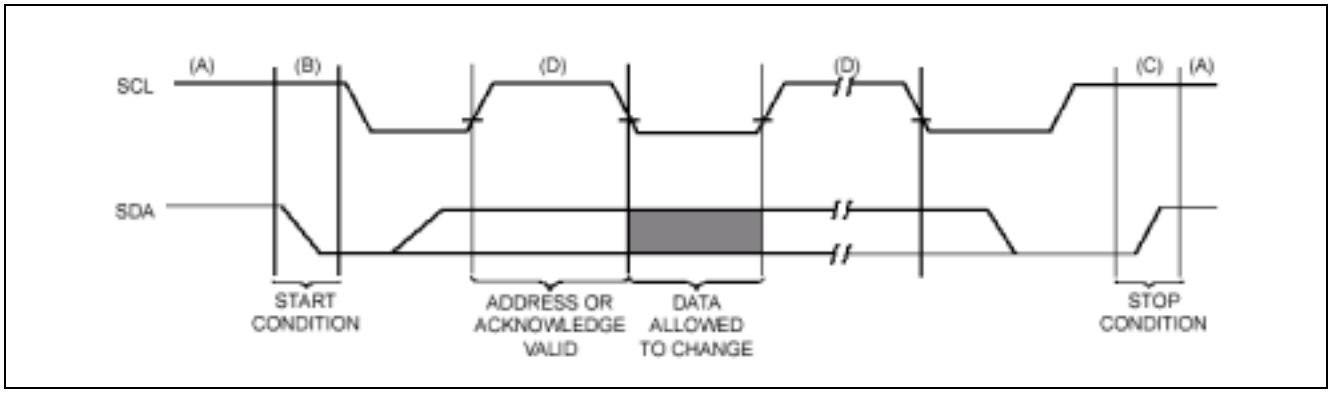
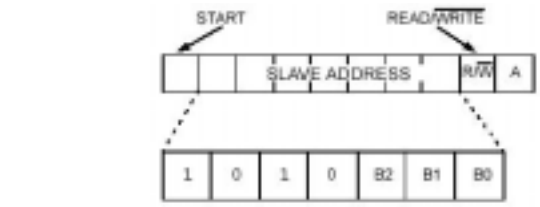


图 3　数据传送的顺序

2.1.1　器件的寻址

控制字节是接在来自主器件的开始条件后接收的最初字节。在 24AA16 的情况下，对于读写操作，将控制码的最初 4 位设定为 2 进制数 1010。后 3 位是时钟选择位（B2、B1、B0），用于选择存取存储器的哪个 256 字块。实际上这些位是字地址的 MSB 3 位。必须注意：根据协议存储器容量被限制在 8 块，1 块为 256 字。因此，协议对每个系统只能支持 1 个 24AA16。控制字节的最后位定义要执行的操作。如果置“1”，就选择读操作；如果置“0”，就选择写操作。24AA16 在开始条件后，监视 SDA 总线，检查被发送来的器件类型识别符。如果接收 1010 码，从属器件就将应答信号输出到 SDA 线。根据 R/$\overline{W}$ 位的状态，24AA16 选择读操作或者写操作。



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

图 4　控制字节

2.1.2 位传送和数据的有效性

在开始条件和停止条件之间，从发送器件传送到接收器件的数据字节数由主器件决定。从 MSB 连续传送各字节（8 位），接着传送应答位。在开始条件后，当时钟信号在 High 电平期间并且数据线稳定时，就表示数据线上的数据有效。必须在时钟信号为 Low 电平期间更改数据线上的数据。对于数据的每 1 位，有 1 个时钟脉冲。

2.1.3 应答

各接收器件必须在接收指定的各字节后生成应答。另外，主器件必须生成对应此应答位的追加时钟脉冲。在 24AA16 中，如果内部程序周期在执行中，就不生成应答。

2.2 写操作

如果将从属地址的 R/$\overline{W}$ 位设定为“0”，就开始写操作。写操作有字节写操作和页写操作 2 种。

2.2.1 字节写

字节写操作对 EEPROM 的随机地址进行写操作，需要发送以下的内容：

- 开始条件（主器件）
- EEPROM 器件地址（主器件）和 R/$\overline{W}$ =0
- 应答位（EEPROM）
- 写对象的 EEPROM 字地址（主器件）
- 应答位（EEPROM）
- 被写的数据字节（主器件）
- 应答位（EEPROM）
- 停止条件（主器件）

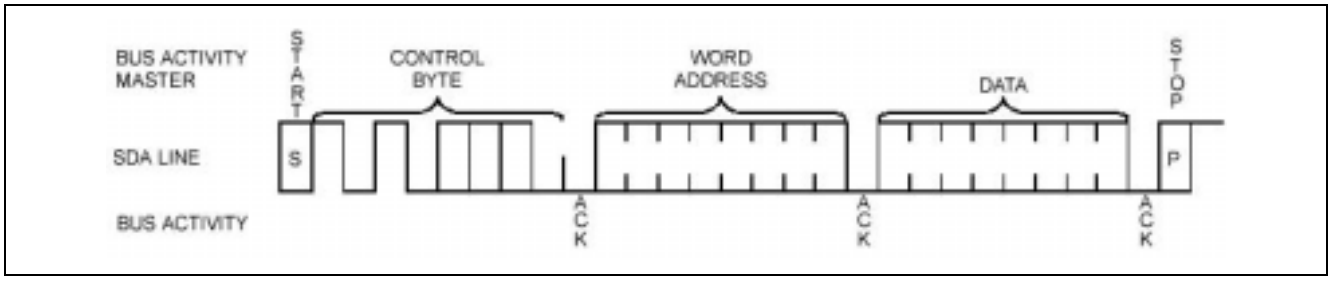


图 5　字节写

2.2.2 页写

页写操作最多能对 EEPROM 写 16 位。在页写期间，EEPROM 自动递增字节间的内部地址指针。

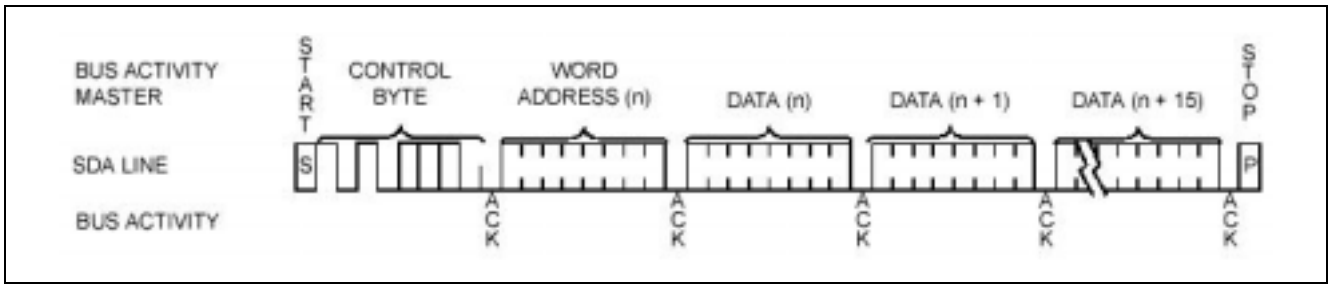


图 6　页写

2.3 读操作

支持当前地址读、随机读和顺序读的3种读操作。除了通过器件的地址字节将R/$\overline{W}$ 位设定为1以外，读操作和写操作相同。

2.3.1 当前地址读

在当前地址读模式中，从最新的存取位置读数据。此当前地址读顺序如下：

- 开始条件（主器件）
- EEPROM 器件地址（主器件）和 R/$\overline{W}$ =1
- 应答位（EEPROM）
- 被读的数据字节（从被递增 1 的从属器件的最新存储器地址传送的 EEPROM 字节）
- 否认应答位（主器件）
- 停止条件（主器件）

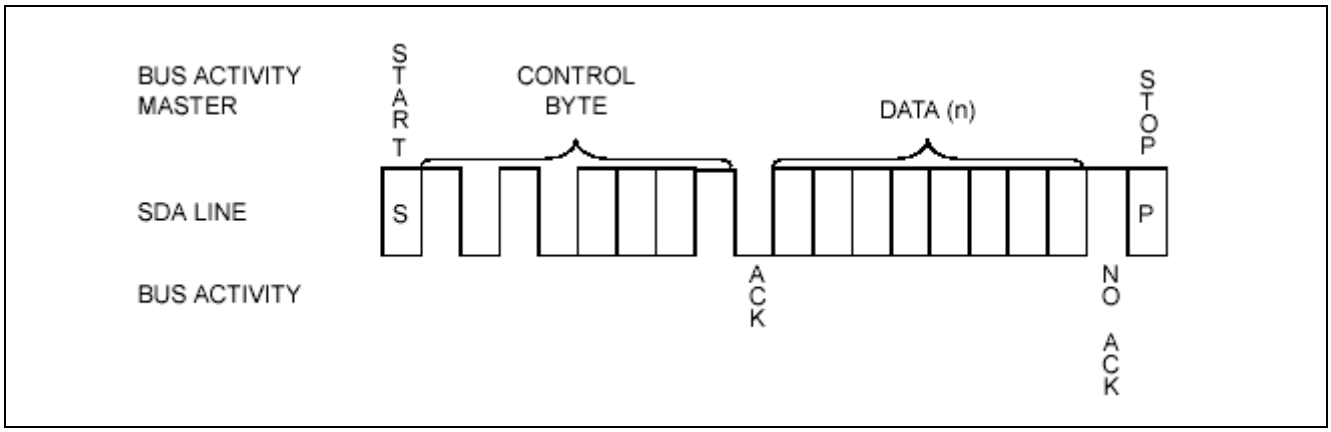


图 7　当前地址读

2.3.2 随机读

如上所述，随机读模式以虚字节写周期开始，然后是当前地址读模式周期（主器件以开始条件、器件地址和对象字地址的顺序发送）。

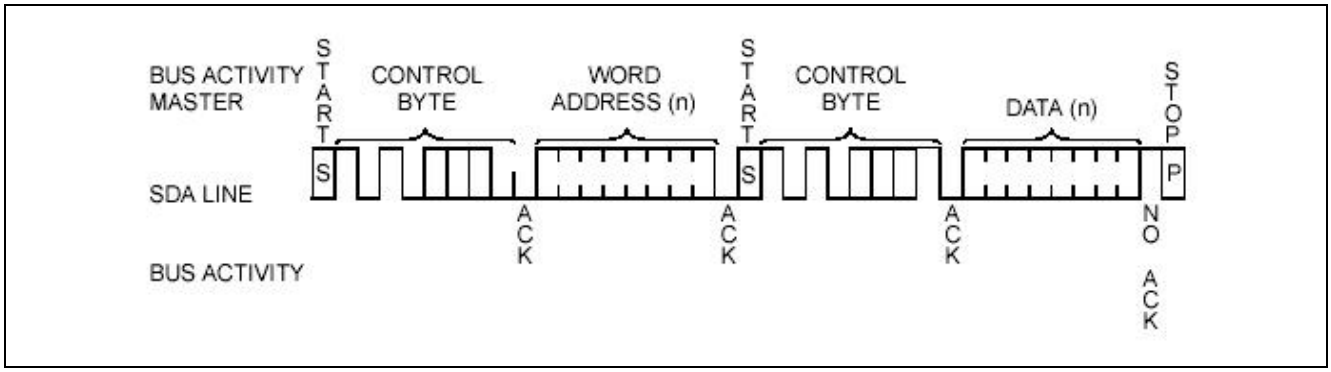


图 8 随机读

2.3.3 顺序读

顺序读模式由随机读开始。主器件在传送（通过否认应答）1 字节后，不结束读操作而在接收各数据后返回有效的应答。根据此应答，从属 EEPROM 继续进行读操作，发送下一个数据字节。顺序读操作在读最新字节后，主器件发行否认应答，接着发送停止条件。

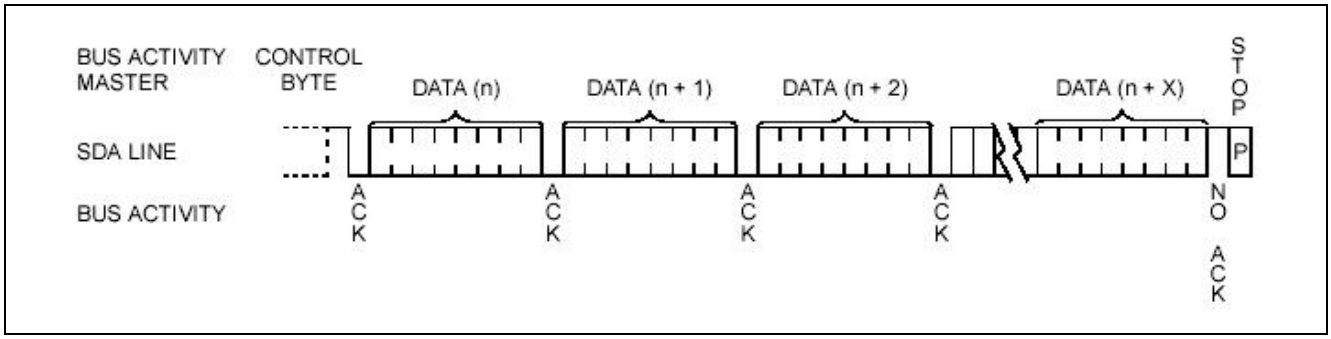


图 9 顺序读

3. I²C 温度传感器

MAX6626 由温度传感器、可编程超高温报警和 I²C 兼容串行接口等构成。使用内置的 A/D 转换器将芯片的温度转换为数字值。转换结果被保存到温度寄存器，随时能通过串行接口读取。如果转换结果超过可编程高温寄存器的值，就启动专用报警输出（OT）。该寄存器能保存错误数，在报警启动前发生错误数能自由设定。这能在噪声多的环境下防止假报警。此器件作为从属器件，支持字节/字单位的读/写命令。功能框图如图 10 所示。

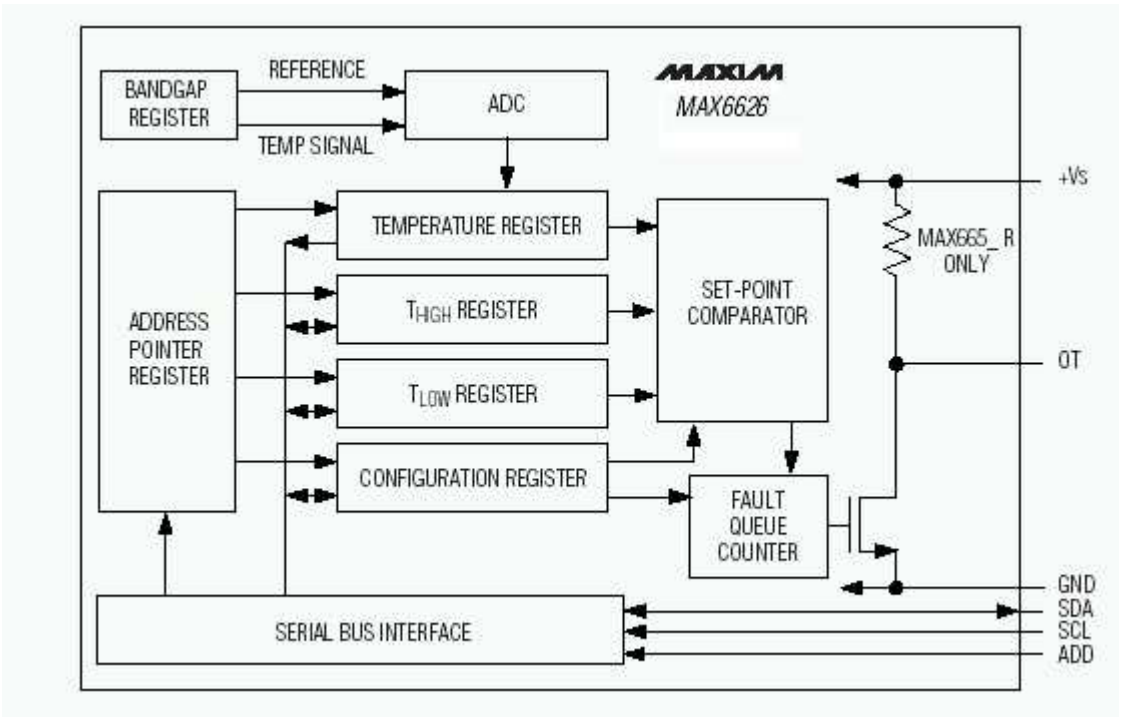


图 10　温度传感器的框图

3.1　寻址

能通过 ADD 管脚设定 4 个不同的地址，并最多能将 4 个 MAX6626 连接到相同的总线。表 2 归纳了不同的地址选择。在此接口中，ADD 管脚连接 GND，地址设定为 90（16 进制数）。

表 2　ADD 连接

ADD 连接	I²C-兼容地址
GND	100 1000
V_S	100 1001
SDA	100 1010
SCL	100 1011

3.2 控制寄存器

在以下的寄存器中定义操作。

为了决定操作的寄存器，首先设定指针寄存器。

表 3 指针寄存器

D7	D6	D5	D4	D3	D2	D1	D0	寄存器
0	0	0	0	0	0	0	0	温度
						0	1	结构
						1	0	T_{LOW}
						1	1	T_{HIGH}

温度（TEMP）寄存器是 12 位只读寄存器，保持最新的温度数据。寄存器的长度为 16 位，未使用的位被屏蔽为 0。使用 2 个补数格式，以°C 表示数字温度，该 LSB 对应 0.0625°C。

表 4 温度寄存器

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

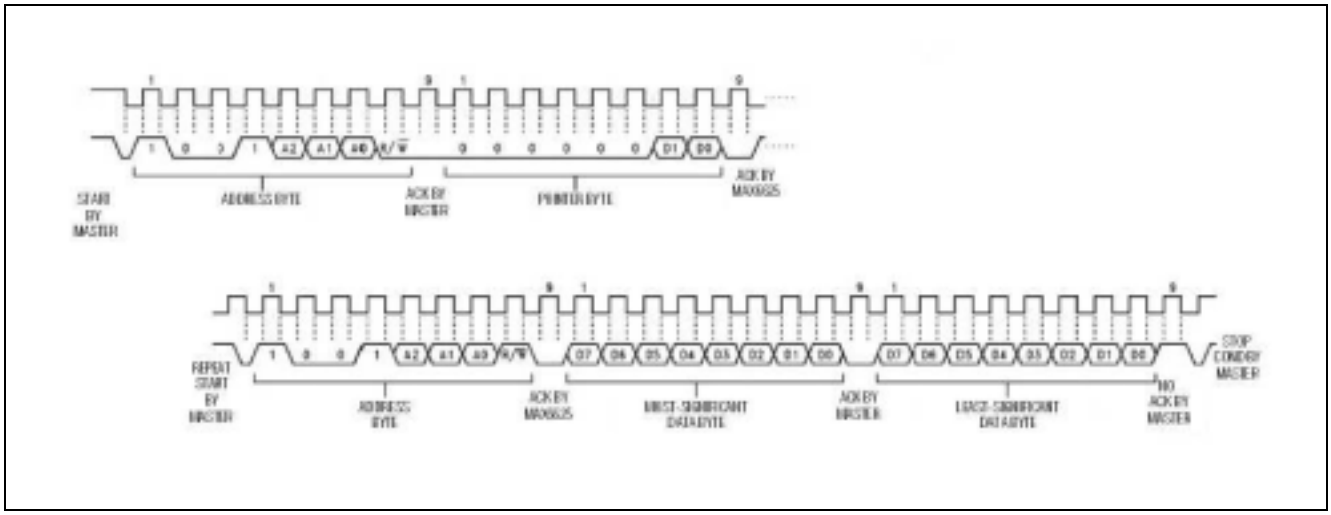


图 11 2 字节寄存器的读（TEMP、T_{HIGH} 和 T_{LOW}）

结构寄存器是 8 位可读写寄存器，包含错误等待队列的项目数、温度报警极性选择、中断模式选择和关机控制位。请参照**表 5** 的位结构。

表 5　结构寄存器

D7	D6	D5	D4	D3	D2	D1	D0	错误数
			错误等待队列的项目数		OT 极性	比较器或者中断模式	关机	
0	0	0	0	0	0: Low 电平有效 1: High 电平有效	0: 比较器 1: 中断	0: 通常运行 1: 关机	1
			0	1				2
			1	0				4
			1	1				6

结构寄存器的读操作和写操作的时序图如图 12 和图 13 所示。

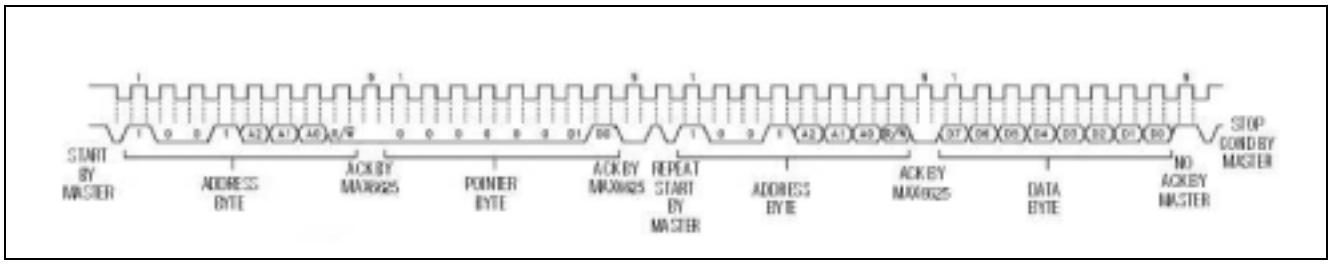


图 12　结构寄存器的读操作

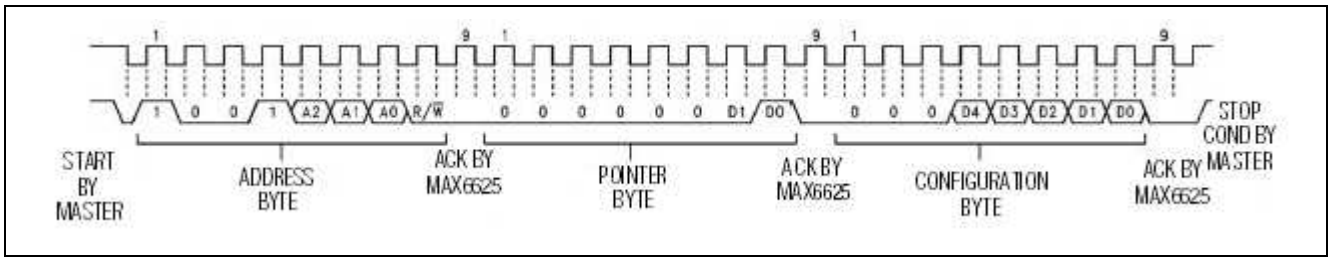


图 13　结构寄存器的写操作

高温（T_{HIGH}）寄存器是 9 位可读写寄存器，包含超高温报警的值。***低温***（T_{LOW}）寄存器是 9 位可读写寄存器，当超高温报警在比较器模式中被清除之前包含应该下降的温度。有关这 2 个寄存器的位结构请参照表 6。读操作和写操作的时序图分别如图 11 和图 14 所示。

表 6　T_{HIGH} 和 T_{LOW} 的寄存器

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

注：1. D15：MSB 是符号位。
2. D6～D0：读时为全 0，不能写。
3. LSB=0.5°C

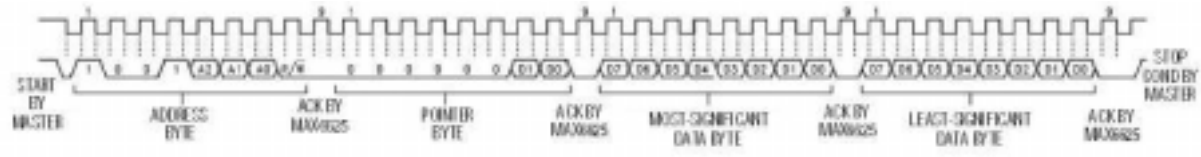


图 14　T_{HIGH} 和 T_{LOW} 的写操作

3.3　温度转换

使用内部带隙，生成 A/D 转换所需的稳定的温度基准电压和与绝对温度（PTAT）成比例的信号。数字转换后的 PTAT 信号分辨率为 0.0625°C，该转换速率为 133ms。温度寄存器保存最新的转换值。

3.4　超高温报警

能通过结构寄存器设定专用的超高温输出管脚（OT）的极性和模式（中断和比较器）。错误等待队列的项目数决定报警的动作。

- 自由设定错误等待队列，通过设定 OT 报警启动前发生的容许范围外的温度连续读取次数，能在噪声多的环境下避免假报警。容许范围外是指读取的温度在 T_{HIGH} 以上或者在 T_{LOW} 以下。
- 在比较器模式中，当转换值连续超过 T_{HIGH} 寄存器值的次数和队列的项目一致时，置 OT 信号。当转换值连续低于 T_{LOW} 寄存器值的次数和队列的项目一致时，清除 OT 信号。例如，假设将 T_{HIGH}、T_{LOW} 和错误等待队列的项目数各自设定为+75°C、+50°C、4，在转换值连续 4 次超过+75°C 前，OT 信号不被设置。同样地，在转换值连续 4 次低于+50°C 前，OT 信号不被清除。比较器模式不需通过主器件就能自动清除 OT 错误，这是理想的冷启动。
- 中断模式中，在特定的条件下，通过 OT 信号设置超高温和超低温的错误报警置。在接通电源时，如果错误等待队列被清除，IC 就在监视 T_{LOW} 错误后寻找 T_{HIGH} 错误。在 T_{LOW} 错误后，监视 T_{HIGH} 错误。每当 OT 信号被正确清除时，重复该处理。如果发生任意错误，IC 就保持置位状态直到通过读相应的寄存器清除为止。其次，器件监视相反型的错误。例如，假设将 T_{HIGH}、T_{LOW} 和错误等待队列的项目数各自设定为+75°C、+50°C、4，在转换值连续 4 次超过+75°C 前，OT 信号不被设置。另外，OT 信号通过读温度寄存器被清除，然后在转换值连续 4 次低于+50°C 后，OT 信号被设置。

3.5 关机

在关机模式中将温度寄存器设定为 H’8000，A/D 转换器为 off（器件的电流降低 1μA）。当从关机状态开始启动时，就在最初的温度转换结束前，温度寄存器的值为 H’8000。

4. I²C 4 位 5×7 矩阵 LED 显示驱动器

MAX6953 是能驱动 4 位阴极列 5×7 点阵显示器的串行接口的显示驱动器。此驱动器除用户能定义的 24 个字符的字体数据以外，由 ASCII 码的 104 字符字体、多路扫描电路、行驱动器、列驱动器以及保存各位的 SRAM 等构成。LED 的段电流能在内部以数字形式给每位设定亮度控制。另外，还有将低功率关机模式、段的闪烁和将全部 LED 设定为 ON 的测试模式等功能。LED 驱动器的功能框图如图 15 所示。

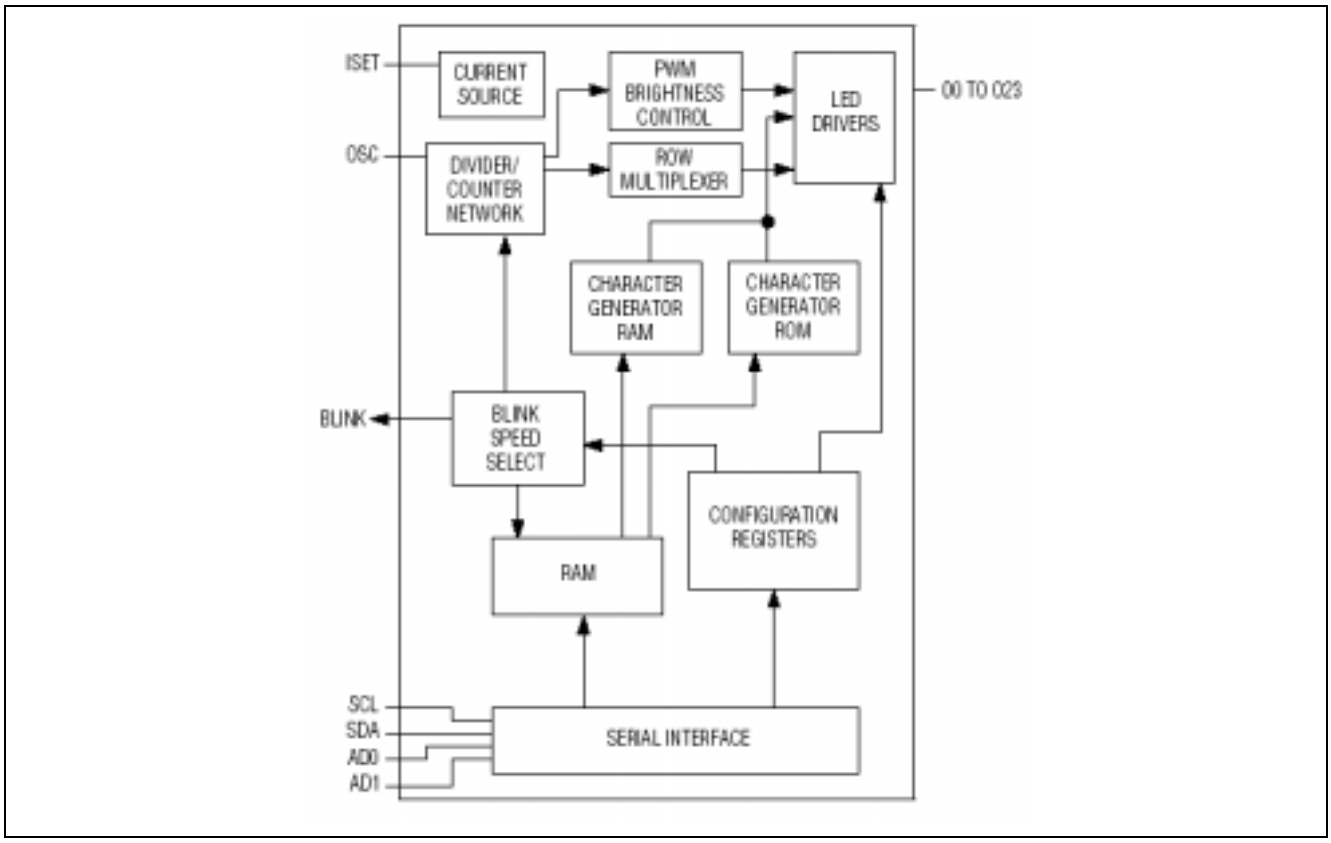


图 15　MAX6953 的框图

4.1　串行寻址

MAX6953 通过 I²C 兼容 2 线接口，作为发送和接收数据的从属器件运行。串行数据（SDA）和时钟（SCL）线用于实现主器件（H8/38024）和从属器件（MAX6953）之间的双向通信。主器件进行和 MAX6953 间的全部数据交换，生成用于同步数据传送的 SCL 时钟。

4.2　开始条件和停止条件

在接口不忙的情况下，SCL 和 SDA 保持 High 电平。主器件在 SCL 为 High 电平的状态下将 SDA 从 High 电平变为 Low 电平，开始发送。主器件在结束和从属器件的通信时，在 SCL 为 High 电平的状态下将 SDA 从 Low 电平变为 High 电平，发送停止条件。在发送停止条件后，能开始其他的总线通信。

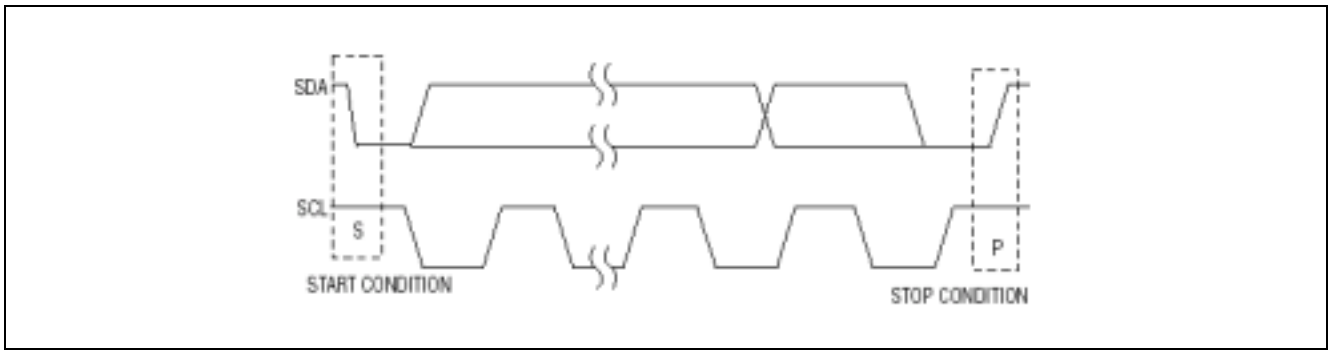


图 16　开始条件和停止条件

4.3　位传送

1 个时钟传送 1 位数据。SDA 线上的数据必须在 SCL 为 High 电平期间维持稳定状态。

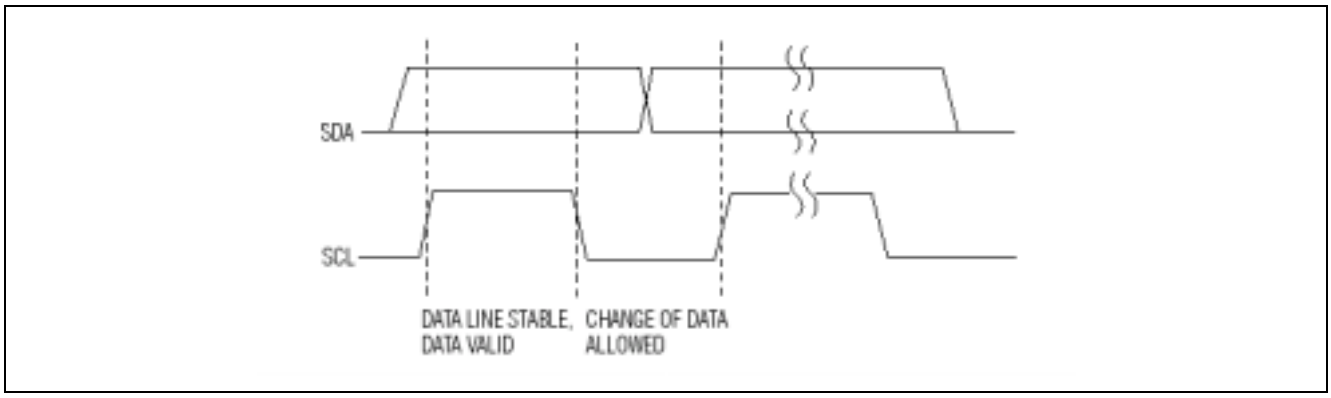


图 17　位传送

4.4　应答

应答位用于接收器件进行信号交换并接收各数据字节，是同步于时钟的第 9 位，请参照图 18。因此，各传送字节实际上需要 9 位，主器件生成第 9 个时钟脉冲。另外，用于应答的时钟脉冲在 High 电平的期间，为了使 SDA 线稳定在 Low 电平，接收器件将 SDA 下拉。在主器件给 MAX6953 发送数据时，从 MAX6953 发送应答位。在 MAX6953 给主器件发送数据时，由于主器件为接收器件，所以主器件发送应答位。

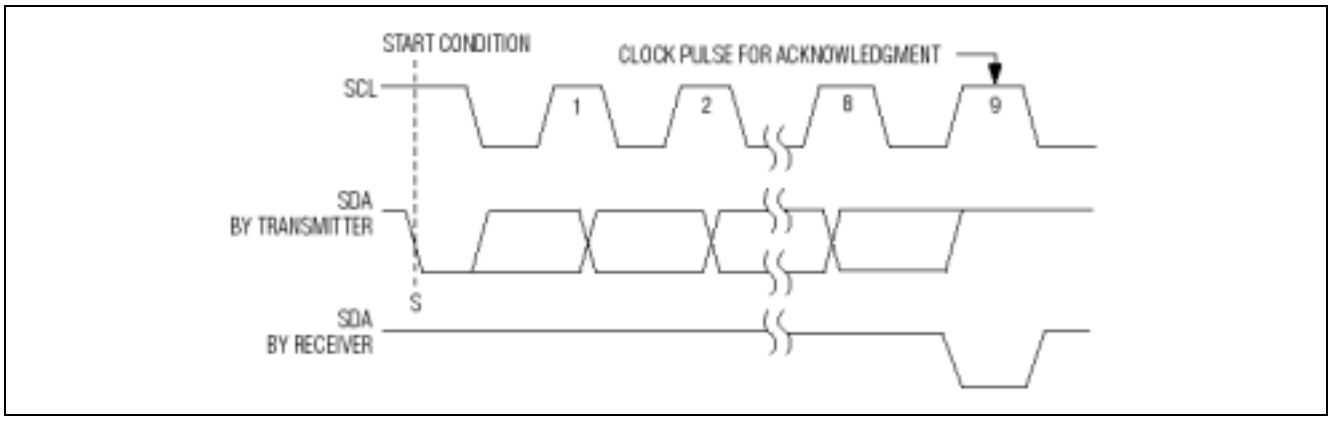


图 18　应答

4.5　从属地址

MAX6953 是 7 位从属地址，接着 7 位从属地址后的第 8 位是 R/$\overline{W}$ 位。写命令为 Low 电平，读命令为 High 电平。MAX6953 从属地址的前 3 位（A6、A5、A4）总是为 101，由地址输入管脚 AD1 和 ADO 决定从属地址位的 A3、A2、A1、A0 的值。能将 AD1 和 ADO 连接到 GND、V+、SDA 或者 SCL。**表 7** 表示全部可能的 AD1、AD0 连接和指定 MAX6953 的对应地址。必须注意：A0～AE（16 进制数）的地址被分配给 EEPROM，B0 的地址被分配给 MAX6953。

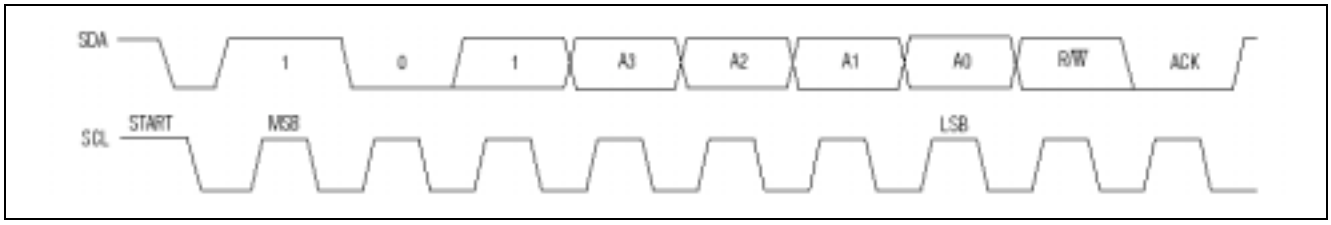


图 19　从属地址

表 7　MAX6953 器件图

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　写信息格式

写 MAX6953 后，接着发送设定为 0 的 R/$\overline{W}$ 的从属地址，至少连续发送 1 个字节的信息。最初的字节信息是命令字节，由下一个字节决定应该写的寄存器。在接收命令字节后，如果检测到停止条件，就保存命令字节，结束运行（图 20）。

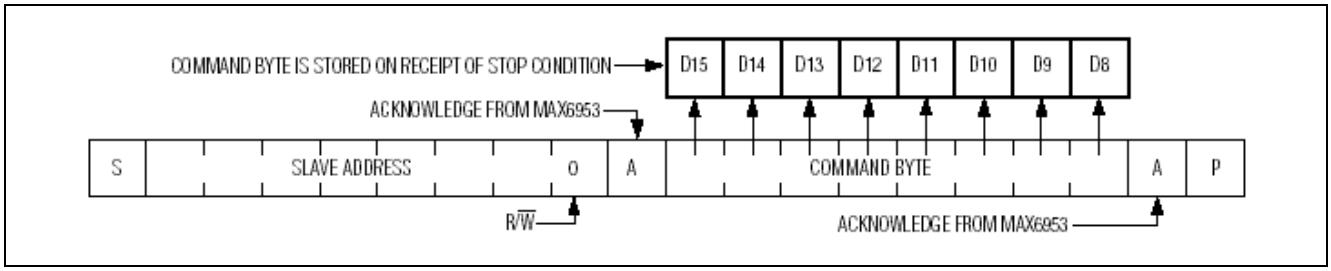


图 20　接收命令字节

在命令字节后接收的字节全部是数据字节。将最初的数据字节保存到由命令字节选择的 MAX6953 的内部寄存器（图 21）。

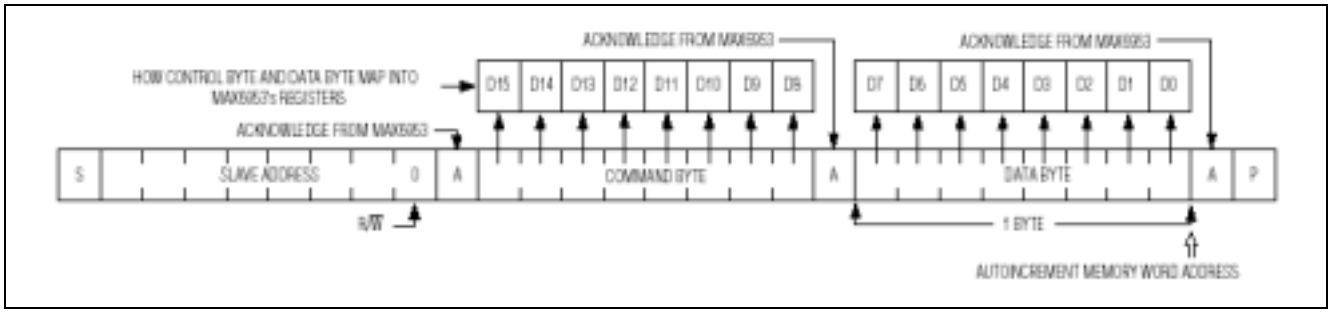


图 21　接收的命令字节和 1 个数据字节

在检测到停止条件前发送多个数据字节时，通常因为命令字节地址自动递增，所以将这些数据字节保存到下一个 MAX6953 的内部寄存器（请参照表 8 和图 22）。

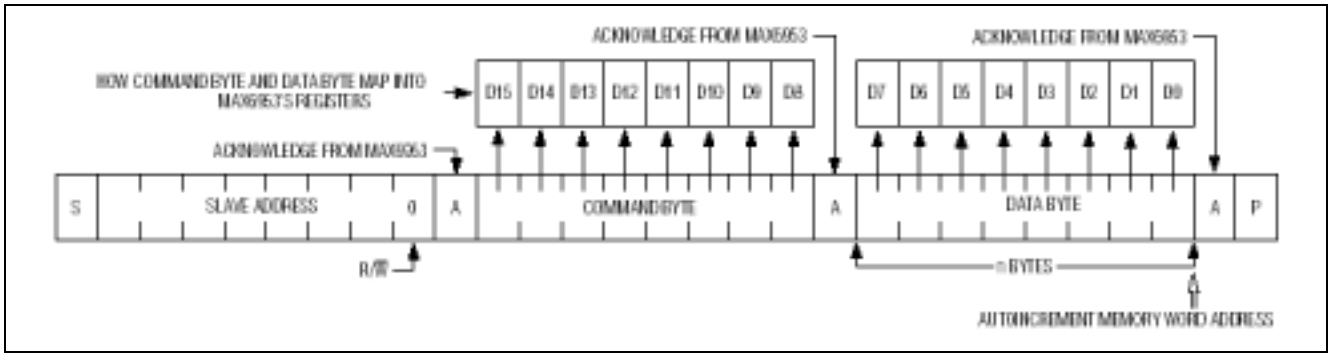


图 22　接收的 n 个数据字节

表 8　命令地址的自动递增方式

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.
x0000111 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　读信息格式

被保存的命令字节和在写操作时被用作地址指针一样，在读操作时被用作地址指针。通常，在使用表 8 说明的相同方式读取各数据字节后，此地址指针自动递增。因此，读操作由写操作开始（图 20）。主器件从最初被初始化的命令字节指定的寄存器中读取数据后，能连续从 MAX6953 中读 n 个字节（图 22）。

5. 程序清单

在以下的 2 个源文件中记述的功能：

- I2C.c
  - 包含 main 函数。
  - 初始化串行通信接口（SCI）和温度传感器。
  - 测试 EEPROM、温度传感器和 LED 驱动器。
- RW.c
  - 包含仿真 SDA 和 ACL 的一般功能。

Main 函数的流程图如图 23，进行以下的操作：

1. 初始化 SCI（2400bps、1 个停止位、禁止奇偶校验）、温度传感器和 LED 驱动器。
2. 测试字节写、字节读、页写、当前地址和顺序地址的读等的 EEPROM，测试结果通过 SCI 发送到 PC。
3. 测定温度，通过 SCI 发送到 PC，然后显示在 1、2、3 位。
4. 0 位显示“0”到“9”。
5. 重复 2 到 4 的操作。

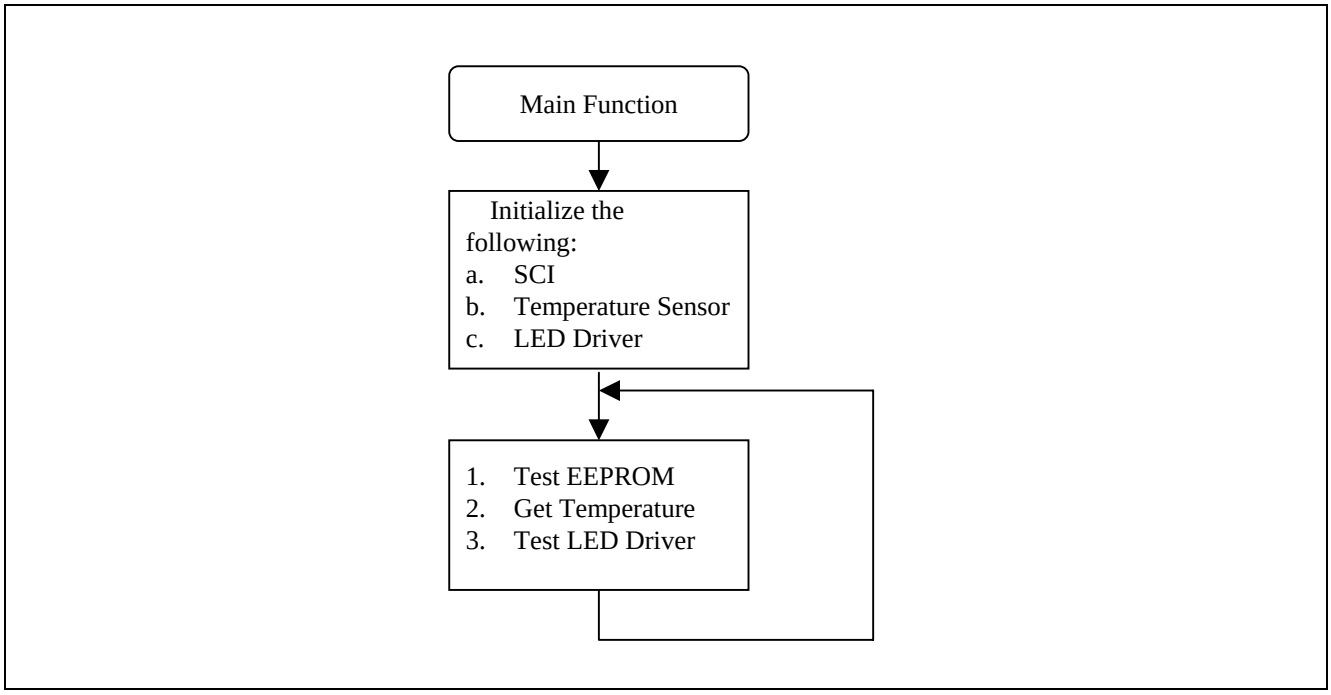


图 23　main 函数的流程图

```
/************************************************************************/
/*                                                                      */
/*  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)
      {
```

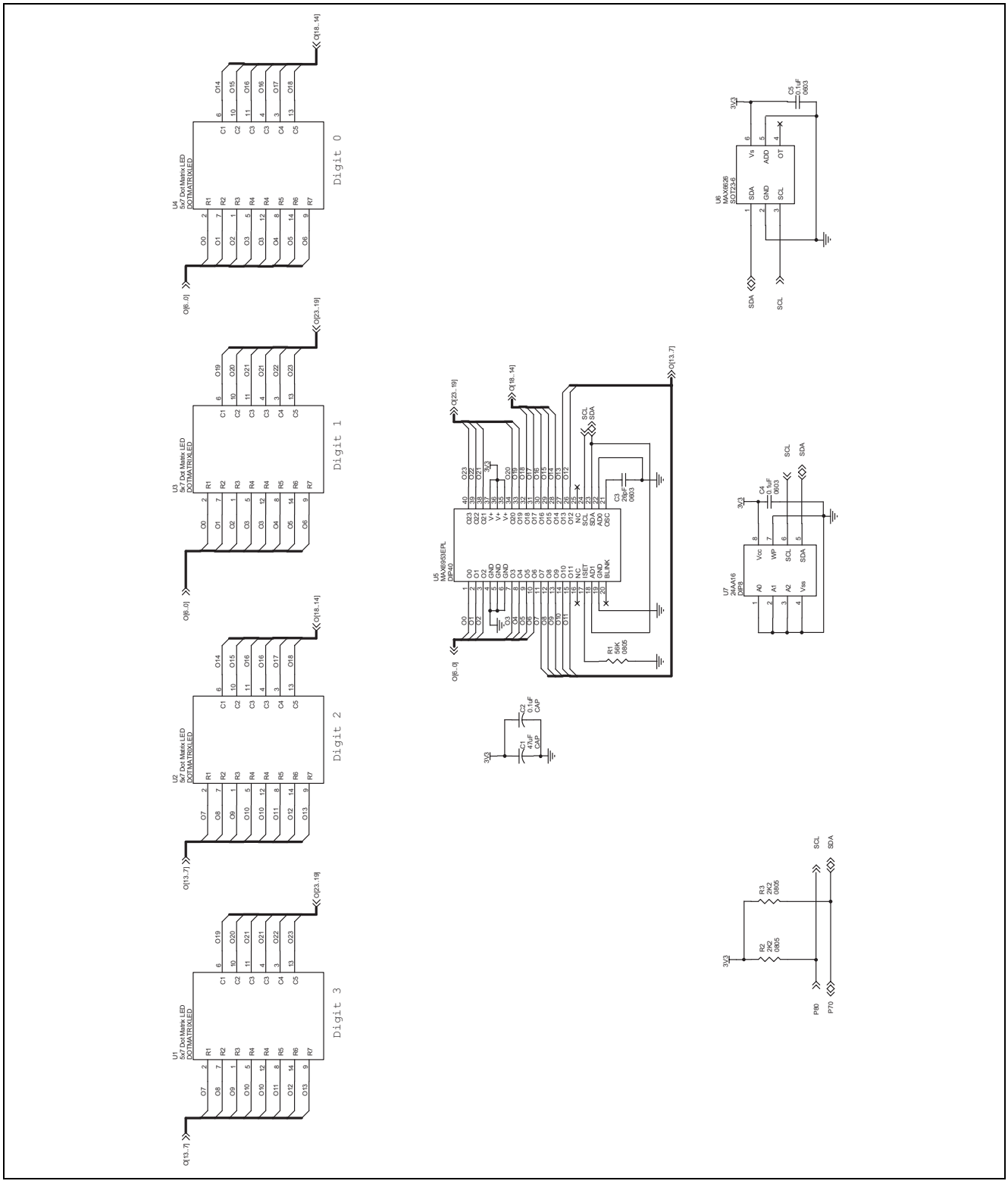
```
            PutStr(" -> ERR_RESPONSE");
            return(ERR_RESPONSE);
         }
      }

      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. 硬件的设计


Digit 0
Digit 1
Digit 2
Digit 3
5x7 Dot Matrix LED
DOTMATRIXLED
MAX6958EFL
DIP40
MAX6626
SOT23-6
24AA16
DIP8
SCL
SDA
3V3
P80
P70

7. 参考文献

1. The I²C-Bus Specification (Version 2.1), January 2000, Koninklijke Philips Electronics N.V.
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,)

公司主页和咨询窗口

有关本应用说明的技术方面的咨询请发邮件到下面的邮箱。

瑞萨科技公司主页　　http://www.cn.renesas.com
亚洲地区技术支持中心　　E-Mail：support.asia@renesas.com

修订记录

Rev.	发行日	修订内容	
		页	修订要点
1.00	2006.03.28	—	初版发行

Cautions

Keep safety first in your circuit designs!

1. Renesas Technology Corp. puts the maximum effort into making semiconductor products better and more reliable, but there is always the possibility that trouble may occur with them. Trouble with semiconductors may lead to personal injury, fire or property damage.
   Remember to give due consideration to safety when making your circuit designs, with appropriate measures such as (i) placement of substitutive, auxiliary circuits, (ii) use of nonflammable material or (iii) prevention against any malfunction or mishap.

Notes regarding these materials

1. These materials are intended as a reference to assist our customers in the selection of the Renesas Technology Corp. product best suited to the customer's application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Renesas Technology Corp. or a third party.
2. Renesas Technology Corp. assumes no responsibility for any damage, or infringement of any third-party's rights, originating in the use of any product data, diagrams, charts, programs, algorithms, or circuit application examples contained in these materials.
3. All information contained in these materials, including product data, diagrams, charts, programs and algorithms represents information on products at the time of publication of these materials, and are subject to change by Renesas Technology Corp. without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corp. or an authorized Renesas Technology Corp. product distributor for the latest product information before purchasing a product listed herein.
   The information described here may contain technical inaccuracies or typographical errors.
   Renesas Technology Corp. assumes no responsibility for any damage, liability, or other loss rising from these inaccuracies or errors.
   Please also pay attention to information published by Renesas Technology Corp. by various means, including the Renesas Technology Corp. Semiconductor home page (http://www.renesas.com).
4. When using any or all of the information contained in these materials, including product data, diagrams, charts, programs, and algorithms, please be sure to evaluate all information as a total system before making a final decision on the applicability of the information and products. Renesas Technology Corp. assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
5. Renesas Technology Corp. semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Renesas Technology Corp. or an authorized Renesas Technology Corp. product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.
6. The prior written approval of Renesas Technology Corp. is necessary to reprint or reproduce in whole or in part these materials.
7. If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination.
   Any diversion or reexport contrary to the export control laws and regulations of Japan and/or the country of destination is prohibited.
8. Please contact Renesas Technology Corp. for further details on these materials or the products contained therein.

注意

本文只是参考译文，前页所载英文版“Cautions”具有正式效力。

请遵循安全第一进行电路设计

1. 虽然瑞萨科技尽力提高半导体产品的质量和可靠性，但是半导体产品也可能发生故障。半导体的故障可能导致人身伤害、火灾事故以及财产损害。在电路设计时，请充分考虑安全性，采用合适的如冗余设计、利用非易燃材料以及故障或者事故防止等的安全设计方法。

关于利用本资料时的注意事项

1. 本资料是为了让用户根据用途选择合适的瑞萨科技产品的参考资料，不转让属于瑞萨科技或者第三者所有的知识产权和其它权利的许可。
2. 对于因使用本资料所记载的产品数据、图、表、程序、算法以及其它应用电路的例子而引起的损害或者对第三者的权力的侵犯，瑞萨科技不承担责任。
3. 本资料所记载的产品数据、图、表、程序、算法以及其它所有信息均为本资料发行时的信息，由于改进产品或者其它原因，本资料记载的信息可能变动，恕不另行通知。在购买本资料所记载的产品时，请预先向瑞萨科技或者经授权的瑞萨科技产品经销商确认最新信息。
   本资料所记载的信息可能存在技术不准确或者印刷错误。因这些错误而引起的损害、责任问题或者其它损失，瑞萨科技不承担责任。
   同时也请通过各种方式注意瑞萨科技公布的信息，包括瑞萨科技半导体网站。
   （http：//www.renesas.com）
4. 在使用本资料所记载部分或者全部数据、图、表、程序以及算法等信息时，在最终做出有关信息和产品是否适用的判断前，务必对作为整个系统的所有信息进行评价。由于本资料所记载的信息而引起的损害、责任问题或者其它损失，瑞萨科技不承担责任。
5. 瑞萨科技的半导体产品不是为在可能和人命相关的环境下使用的设备或者系统而设计和制造的产品。在研讨将本资料所记载的产品用于运输、交通车辆、医疗、航空宇宙用、原子能控制、海底中继器的设备或者系统等特殊用途时，请与瑞萨科技或者经授权的瑞萨产品经销商联系。
6. 未经瑞萨科技的书面许可，不得翻印或者复制全部或者部分资料的内容。
7. 如果本资料所记载的某产品或者技术内容受日本出口管理限制，必须在得到日本政府的有关部门许可后才能出口，并且不准进口到批准目的地国家以外的国家。
   禁止违反日本和（或者）目的地国家的出口管理法和法规的任何转卖、挪用或者再出口。
8. 如果需要了解本资料所记载的信息或者产品的详细，请与瑞萨科技联系。