飞思卡尔半导体
应用笔记

文件编号：AN3291
第 1 版，03/2007

如何在M68HC08、HCS08和HCS12微控制器上应用IIC模块

作者： Stanislav Arendarik
应用工程师
捷克共和国，罗斯诺夫

1 简介

此应用笔记是如何在飞思卡尔的微控制器上应用 IIC 模块的一个示例。IIC 模块可以分别在主模式或从模式下使用。在这种情况下，由于 IIC 总线主要用于在微控制器（MCU）和 IIC 外设之间的通信，因此在主模式时与串行 EEPROM 进行通信。IIC 总线可以在两个微控制器（MCU）之间直接进行通信， 然而 SPI 总线却更适用于这种应用。

此应用笔记总结了通用IIC总线状态和定义，并提供了如何与串行EEPROM进行通信的示例（24C16和24C512）。您可以轻松地用另外一个IIC器件取代EEPROM，但是必须改变将其标识为从器件的IIC地址字节。

目录

freescale™
飞思卡尔半导体

2 IIC 总线摘要

IIC总线是基于主机和从机间线与（开漏）连接的双向、两线式总线。为实现正确的功能，需要使用上拉电阻。每次数据的传输都由主机发出，并且得到从机的应答或响应。

2.1 IIC总线术语

下面是在这个应用笔记中使用到的IIC总线术语：

- **发送器（Transmitter）**- 发送数据到总线的器件。一个发送器可以根据自身要求，将数据置于总线上（主发送器），或者对来自另外一个器件的数据请求进行响应（从发送器）。
- **接收器（Receiver）**- 从总线接收数据的器件。
- **主机（Master）**- 初始化传输，产生时钟信号和终止传输的器件。一个主机可以被当作一个接受器或发射器。
- **从机（Slave）**- 由主机寻址的器件。从机可能成为接收器或发送器。
- **多主机（Multi-master）**- 不止一个的主机可以共存于一个总线上，而不产生冲突或数据丢失。
- **仲裁（Arbitration）**- 预先设定程序，一次授权一个主机来控制总线。
- **同步（Synchronization）**- 预先设定程序，使两个或更多主机所提供的时钟信号同步。
- **SDA** - 数据信号线（串行数据）。
- **SCL** - 时钟信号线（串行时钟）。

SDA和SCL是通过一个电流源或上拉电阻与正电源相连接的双向信号线。当总线空闲时，这两条信号线都保持高电平。在标准模式下，IIC总线上的数据能以高达100kbit/S的速度传输，在快速模式下传输速度为400kbit/s，在高速模式下则高达3.4Mbit/s。与总线连接的器件数量仅取决于总线电容，其限度为400pF。

2.2 位传输

SDA线上的数据必须在时钟处于高电平期间保持稳定。在SCL线上的时钟信号处于低电平时，SDA线的高电平状态或低电平状态才能产生变化。（见图1）

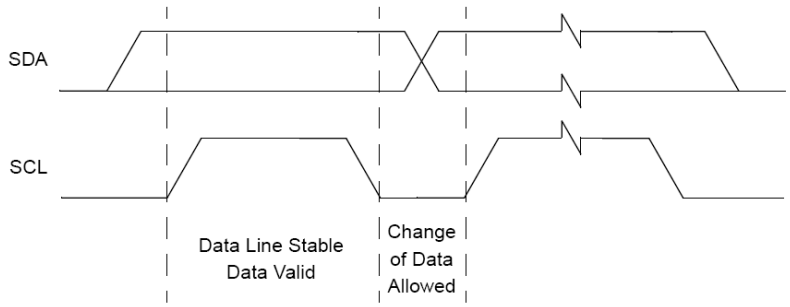

图1: 位传输

2.3 起始条件和停止条件(START and STOP Conditions)

在IIC总线程序内出现的一种独特的状态，被定义为起始条件和停止条件(START and STOP Conditions)（见图2）。当SCL线处于高电平时，SDA线从高电平向低电平跳变时定义为起始（START）条件。当SCL线处于高电平时，SDA线从低电平向高电平跳变时为停止（STOP）条件。起始（START）条件和停止（STOP）条件只能由主机产生。如果在起始（START）之前和停止（STOP）条件之后SDA和SCL线保持在一个高电平，那么总线将是空闲的。

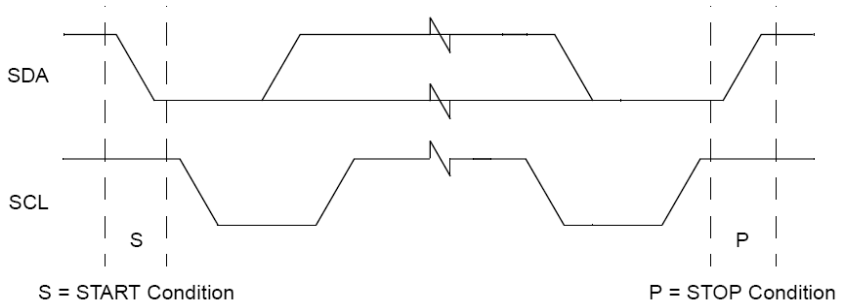

图2: 起始条件和停止条件

2.4 总线通信

总线传输协议是基于一个起始于应答位（ACK）信号的字节传输之上。字节传输从最高位（MSB）开始，到最低位结束（LSB）。传输的每位都由SCL正时钟脉冲计数。第9个SCL脉冲为ACK位。

每个总线传输都由主机发起，主机将起始（START）条件发给总线。而连续的下一个字节序列取决于通过IIC总线传输的数据。

2.5 控制字节

控制字节总是跟随于起始（START）状态。控制字节选择并激活总线上特定的器件；见图3的控制字节结构。前四位是从机的控制代码。例如，串行EEPROM的控制字节是A0（控制代码是二进制的“1010”）。接下来的三位是对连接在总线上的EEPROM芯片的片选信号。EEPROM使用这三位进行多器件操作，但是如果内存地址字节位数大于8位或16位，它们将被用作内部EEPROM地址的更高位。这个地址取决于EEPROM的容量。例如，如果容量处于16kbits的范围内，这三个字节将是地址中最高的三位。

第8位是R/W（读取/写入）位。这一位决定了对从机所请求的操作：

- 如果字节为0，就意味着处于写入操作状态。
- 如果字节为1，就意味着处于读取操作状态。

第九位是ACK位，并且它总是由从机产生。

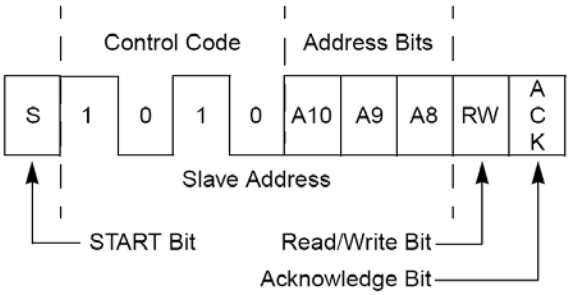

图3：控制字节结构

2.6 地址字节

地址字节将地址信息带给从机。这个字节决定了在EEPROM中的实际地址，将向这个地址读取或写入数据。地址信息结构取决于所连接的EEPROM的容量。高达16kbits的EEPROM使用单地址字节，但是具有更高容量的EEPROM使用双字节地址（地址H和地址L）。首先发送地址H。图4显示了单字节地址的地址字节结构，图5中显示双字节地址的地址字节结构。EEPROM型24C16和24C512作为通信器件使用。

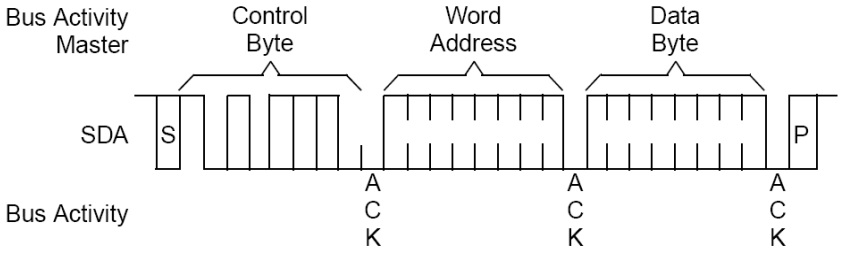

图4：单字节地址IIC写入格式

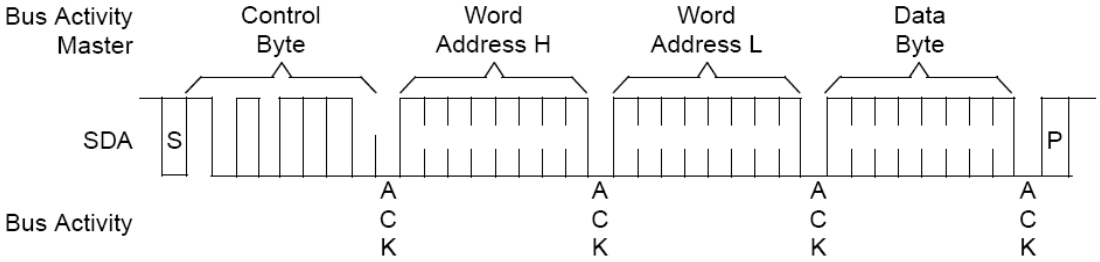

图5：双字节地址IIC写入格式

2.7 应答

在SCL线上的第9脉冲期间，应答位（ACK）在SDA线上是低电平脉冲。SCL脉冲总是由主机产生。接收器产生ACK脉冲。这就意味着在写操作中，在每一个字节被传输之后，从机发出ACK信号。在读操作中，当控制字节和地址字节由主机发出时，从机发出ACK信号；当主机接收来自从机的数据时，主机将会发出ACK信号。

2.8 读取/写入格式

在IIC总线上有两种数据传输：

- 在从机中读取数据
- 将数据写入从机

当执行写入操作时，一个数据字节将紧随地址字节之后。这个数据字节包括了在地址字节中将被写入地址中的实际数据；见图4和图5。

当读取操作执行通信协议的第一部分时：

1. 主机将实际地址写入从机。
2. 主机将重复的起始（START）条件发送到总线，带有一个等于1的读取/写入模式的控制字节。
3. 从机发送ACK并且立即发送实际要求的数据。

见图6的读取操作结构。

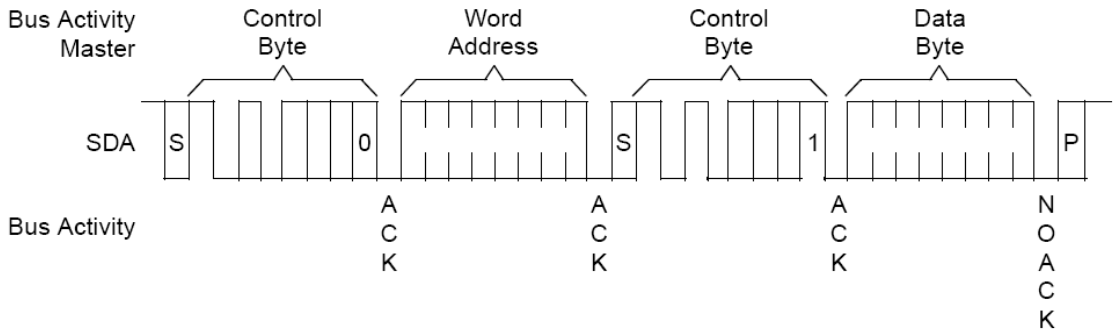

图 6：读取操作结构

见图 4、图 5 和图 6 中的读取字节和写入字节操作。第二种数据传输采用页写入和页读取的形式。想要获得这些类型的更多信息，敬请参阅从机数据表。

3 用于微控制器的IIC软件程序

您可以在MCU数据手册中获得IIC模块的详细说明。下面是其简介：

- **地址寄存器** - HC908中用到的地址寄存器与S08和S12微控制器中的不同。在HC908的内核中，地址寄存器的内容在起始（START）条件后将自动发送，数据寄存器的内容依据该状态而定。这个地址寄存器在从模式下也用作其自身的IIC从机地址。在S08和S12内核中，地址寄存器仅在从模式下激活，并在总线上组成了其自己的微控制器IIC从机地址。这可以被用在总线上有多微控制器连接时使用。
- **分频寄存器** - 决定SCL的时钟速度。总线频率作为IIC模块的源频率。
- **控制寄存器** - 功能性的控制所有的IIC模块。
- **状态寄存器** - 包括关于实际总线状态的信息。
- **数据寄存器** - 作为所有在总线传输的信息的输入和输出点。

3.1 IIC 的初始化

主要的任务是设置合适的SCL时钟信号的速度。这些时钟是基于总线频率的。下一个任务是设置IIC模块功能 - 即使用或不使用中断服务程序、激活IIC模块以及设置从模式。敬请参考例1、例2和例3。

例1. 用于HC908微控制器的软件（使用MC68HC908AP64）

```
void init_I2C (void)
{
   MMCR1_MMEN = 1;       // Enable IIC;
   MMCR1_MMCLRBB = 1;    // Clear bus busy flag;
   MMCR1_MMTXAK = 1;
   MMCR2_MMRW = 0;       // R/W bit = 0;
   MMCR2_MMAST = 0;      // Slave mode actually;
   MMCR2_MMNAKIF = 0;
   MMCR2_MMALIF = 0;     // Clear flags;
   MMSR = 0;             // Clear all flags (Status register);
   MMFDR = 3;            // Set speed to 25kHz for Bus = 4MHz;
}
```

例2. 用于9S12微控制器的软件（使用MC9S12DP256B）

```
void Init_I2C(void)
{
    IBFD = 0x4C;          // frequency divider register: Bus = 8MHz => SCL = 91kHz;
    IBAD = 0x00;          // slave address of this module;
    IBCR = 0x80;          // enable I2C module and No interrupts;
    IBSR_IBAL = 1;        // clear the IBAL flag;
}
```

例3. 用于9S08微控制器的软件（使用MC9S08GB60）

```
void Init_I2C (void)
{
    IIC1C_IIC1EN = 1;     // Enable I2C;
    IIC1C_TXAK = 1;       // not generate ACK by master after transfer;
    IIC1C_MST = 0;        // Slave mode actually;
    IIC1F = 0x99;         // Set speed to 50kHz for Bus = 18.8743MHz;
                          // 12.5k->0x39; 50k->0x99; 100k->0x59;
    IIC1S_BUSY = 1;       // Clear bus busy flag;
    IIC1S_SRW = 0;        // R/W bit = 0;
    IIC1S_IIC1IF = 0;
    IIC1S_ARBL = 0;       // Clear flag;
}
```

3.2 写入功能

我们将首先讨论在一个地址上写入从机的情况。为了完成这一任务，通过在从机地址字节之后发送起始（START）位，激活与从机之间的通信。这些位的内容已在2.6节（即“地址字节”）讨论。当主机发送这个字节时，从机在第9个脉冲间发送ACK位。从机必须对主机发送的每个字节进行应答。

下一个字节（第二个字节）是内存阵列的内部地址。根据内存的容量，这个地址可以是一个字节或两个字节。从机必须对主机发送的每个字节进行应答。

第三个字节（对于大存储器而言是第四个字节）是数据字节 - 此数据将写入一个特定的地址中。从机的ACK信号位于这个数据字节之后。当主机收到这个ACK位时，它会产生一个停止（STOP）位。此时，从机内部将断开与IIC总线连接，并且在内部处理写入操作。当从机处理所要求的操作（写入操作）时，它不能响应主机的下一个请求。因此，主机必须等待一段时间，这段时间由从机数据表所定义，或者它必须周期性的向从机发送请求并且检测正确的响应（来自从机的ACK信号）。

例4. HC908微控制器的写入字节（使用MC68HC908AP64和24C16）

```
void I2C_write_byte (byte addr, byte wr_data)
{
   MMCR2_MMRW = 0;                // Set Write mode;
   MMADR = own_sl_addr;           // set combined address of Slave for write;
   MMDTR = addr;                  // set address of Slave to read;
      //-------start of transmit bytes to IIC bus-----
   MMCR2_MMAST = 1;               // Start transfer - Master bit = 1;
   while (!(MMSR_MMTXBE));        // wait till data transferred;
   while (MMSR_MMRXAK);           // wait for ACK from slave;
      //-----Slave ACK occurred------------
   MMDTR = wr_data;               // write data byte into EEPROM;
   while (!(MMSR_MMTXBE));        // wait till data transferred;
   MMDTR = 0xFF;                  // generate SCL impulse for slave to send ACK bit;
   while (MMSR_MMRXAK);           // wait for ACK from slave;
   MMCR2_MMAST = 0;               // STOP bit;
}
```

例6和例7用于更大容量的EEPROM（24C512）。如例5中所定义的那样，字地址可以分为两个部分（两个字节）：低字节和高字节。

例5. 分为低字节和高字节的字地址

```
typedef union
{
  word EE_Address;
  struct
  {
    byte Address_H;
    byte Address_L;
  }Bytes;
}tAddr;

tAddr sADDR;
#define Address sADDR.EE_Address
#define Addr_L  sADDR.Bytes.Address_L
#define Addr_H  sADDR.Bytes.Address_H
```

例6. 9S08微控制器的写入字节（使用MC9S08GB60和24C16）

```
void IIC_write_byte(word addr, byte data)
{
  Address = addr;          // load address;
  temp = (Addr_H & 0x07) << 1;
  IIC1C_TXAK = 0;          // RX/TX = 1; MS/SL = 1; TXAK = 0;
  IIC1C |= 0x30;           // And generate START condition;
        //-------start of transmit first byte to IIC bus-----
  IIC1D = IIC_SLAVE | temp;  // Address the slave and set up for master transmit;
  while (!IIC1S_IICIF);  // wait until IBIF;
  IIC1S_IICIF=1;           // clear the interrupt event flag;
  while(IIC1S_RXAK);       // check for RXAK;
        //-----Slave ACK occurred------------
  IIC1D = Addr_L;             // Send low byte of the word address;
  while (!IIC1S_IICIF);  // wait until IBIF;
  IIC1S_IICIF=1;           // clear the interrupt event flag;
  while(IIC1S_RXAK);       // check for RXAK;
        //-----Slave ACK occurred------------
  IIC1D = data;
  while (!IIC1S_IICIF);  // wait until IBIF;
  IIC1S_IICIF=1;           // clear the interrupt event flag;
  while(IIC1S_RXAK);       // check for RXAK;
        //-----Slave ACK occurred------------
  IIC1S_IICIF=1;           // clear the interrupt event flag;
  IIC1C_MST = 0;           // generate STOP condition;
}
```

例7. 9S12微控制器的写入字节（使用MC9S12XDT512）

```
void IIC_write_byte(word addr, byte data)
{
  Address = addr;             // load address;
  IIC0_IBCR_TXAK = 0;         // TXAK = 0;
  IIC0_IBCR |= 0x30;         // RX/TX = 1; MS/SL = 1; TXAK = 0;
                              // And generate START condition;
        //-------start of transmit first byte to IIC bus-----
  IIC0_IBDR = slavewrite;     // Address the slave and set up for master transmit;
```

```
  while (!IIC0_IBSR_IBIF);  // wait until IBIF;
  IIC0_IBSR_IBIF=1;          // clear the interrupt event flag;
  while(IIC0_IBSR_RXAK);     // check for RXAK;
        //-----Slave ACK occurred------------
  IIC0_IBDR = Addr_H;        // Send high byte of the word address;
  while (!IIC0_IBSR_IBIF);  // wait until IBIF;
  IIC0_IBSR_IBIF=1;          // clear the interrupt event flag;
  while(IIC0_IBSR_RXAK);     // check for RXAK;
        //-----Slave ACK occurred------------
  IIC0_IBDR = Addr_L;        // Send low byte of the word address;
  while (!IIC0_IBSR_IBIF);  // wait until IBIF;
  IIC0_IBSR_IBIF=1;          // clear the interrupt event flag;
  while(IIC0_IBSR_RXAK);     // check for RXAK;
        //-----Slave ACK occurred------------
  IIC0_IBDR = data;
  while (!IIC0_IBSR_IBIF);  // wait until IBIF;
  IIC0_IBSR_IBIF=1;          // clear the interrupt event flag;
  while(IIC0_IBSR_RXAK);     // check for RXAK;
        //-----Slave ACK occurred------------
IIC0_IBCR_MS_SL = 0; // generate STOP condition;
}
```

3.3 读取功能

读取功能与写入功能相似。首先，将地址（字节和文字）写入从机。来自从机的ACK位总是位于这些写循环之后。当地址正确地写入从机时，主机产生重复起始（REPEAT START）条件。然后，主机再一次发送控制字节，但是采用的是等于1的读取/写入位。这就意味着读自从机的功能。从机的ACK字节位于这个字节之后。然后，主模式必须切换到接收模式，并且执行从IIC数据寄存器（IICDR）读取数据的功能。这个指令产生了SCL脉冲，使从机可以从要求的地址中发送数据。

有两个可能的选项：

- 如果主机不需要来自从机的下一个数据字节，它将发送NOACK（ACK=1）信号和停止（STOP）条件。
- 如果主机需要来自从机的下一个数据字节，它会产生ACK位（ACK=0）。

从机识别此ACK位后，将内部的地址指示器移动到下一个地址。然后，主机执行从IICDR中读取数据的功能，为从机发送数据产生下一系列的SCL脉冲。此序列可以无限的执行下去。

例8. HC908系列的读取字节（用于 HC908AP64）

```
byte I2C_read_byte (byte addr)//output is byte "rd_data";
{
   MMCR2_MMRW = 0;            // Set Write mode;
   MMADR = own_sl_addr;       // set combined address of Slave for write;
   MMDTR = addr;              // set address;
        //-------start of transmit first byte to IIC bus-----
   MMCR2_MMAST = 1;           // Start transfer - Master bit = 1;
   while (!(MMSR_MMTXBE));    // wait till data transferred;
   while (MMSR_MMRXAK);       // wait for ACK from slave;
```

```
      //-----Slave ACK occurred------------
   MMCR2_MMRW = 1;            // set read operation;
   MMCR1_REPSEN = 1;          // enable repeat Start bit;
   MMCR1_MMTXAK = 0;          // Master will generate ACK;
   MMCR2_MMAST = 1;           // Start transfer - Master bit = 1;
      //------start of transmit Repeat start & "A1" to IIC bus-------
   MMDTR = 0xFF;              // send repeated start and combined address;
   while (!(MMSR_MMTXBE));    // wait till data transferred;
   while (MMSR_MMRXAK);       // wait for ACK from slave;
   MMDTR = 0xFF;              // send SCL clocks for the slave to send data;
   MMCR1_MMTXAK = 1;          // Disable master ACK after read byte from Slave;
   while (!(MMSR_MMRXBF));    // wait till data received;
   rd_data = MMDRR;           // read received data;
   MMCR2_MMAST = 0;           // generate STOP bit - End of transfer;
   return rd_data;
}
```

例9和例10用于更大容量的EEPROM（24C512）。字地址可以分为两部分（两个字节）：低字节和高字节（见例5）。

例9. 9S08系列的读取字节（使用MC9S08GB60和24C16）

```
typedef union
{
   word EE_Address;
   struct
   {
      byte Address_H;
      byte Address_L;
   }Bytes;
}tAddr;

tAddr ADDR;

#define Address ADDR.EE_Address
#define Addr_L  ADDR.Bytes.Address_L
#define Addr_H  ADDR.Bytes.Address_H

byte IIC_read_byte(word addr)
{
  Address = addr;
  temp = (Addr_H & 0x07) << 1;
  IIC1C_TXAK = 0;           // RX/TX = 1; MS/SL = 1; TXAK = 0;
  IIC1C |= 0x30;            // And generate START condition;
  IIC1D = IIC_SLAVE | temp;   // Address the slave and set up for master transmit;
  while (!IIC1S_IICIF);     // wait until IBIF;
  IIC1S_IICIF=1;            // clear the interrupt event flag;
  while(IIC1S_RXAK);        // check for RXAK;
        //-----Slave ACK occurred------------
  IIC1D = Addr_L;           // Send low byte of word address;
  while (!IIC1S_IICIF);     // wait until IBIF;
  IIC1S_IICIF=1;            // clear the interrupt event flag;
  while(IIC1S_RXAK);        // check for RXAK;
        //-----Slave ACK occurred------------
  IIC1C_RSTA = 1;           // set up repeated start;
  IIC1D = IIC_SLAVE | temp | 1;  // (slave_address) | (RW = 1);
  while (!IIC1S_IICIF);     // wait until IBIF;
```

```
  IIC1S_IICIF=1;           // clear the interrupt event flag;
  while (IIC1S_RXAK);      // check for RXAK;
        //-----Slave ACK occurred------------
  IIC1C_TX = 0;            // set up to receive;
  IIC1C_TXAK  = 1;         // acknowledge disable;
  RD_data = IIC1D;         // dummy read;
  while (!IIC1S_IICIF);    // wait until IBIF;
  IIC1S_IICIF=1;           // clear the interrupt event flag;
  IIC1C_MST = 0;           // generate stop signal;
  RD_data = IIC1D;         // read right data;
  return RD_data;
}
```

例10．9S12系列的读取字节（使用MC9S12DT512和24C512）

```
typedef union
{
   word EE_Address;
   struct
   {
      byte Address_H;
      byte Address_L;
   }Bytes;
}tAddr;

tAddr ADDR;

#define Address ADDR.EE_Address
#define Addr_L  ADDR.Bytes.Address_L
#define Addr_H  ADDR.Bytes.Address_H

void IIC_read_byte(word addr)
{
  Address = addr;
  IIC0_IBCR_TXAK = 0;          // TXAK = 0;
  IIC0_IBCR |= 0x30;          // RX/TX = 1; MS/SL = 1; TXAK = 0;
                               // And generate START condition;
  IIC0_IBDR = slavewrite;      // Address the slave and set up for master transmit;
  while (!IIC0_IBSR_IBIF);     // wait until IBIF;
  IIC0_IBSR_IBIF=1;            // clear the interrupt event flag;
  while(IIC0_IBSR_RXAK);       // check for RXAK;
        //-----Slave ACK occurred------------
  IIC0_IBDR = Addr_H;          // Send high byte of word address;
  while (!IIC0_IBSR_IBIF);     // wait until IBIF;
  IIC0_IBSR_IBIF=1;            // clear the interrupt event flag;
  while(IIC0_IBSR_RXAK);       // check for RXAK;
        //-----Slave ACK occurred------------
  IIC0_IBDR = Addr_L;          // Send low byte of word address;
  while (!IIC0_IBSR_IBIF);     // wait until IBIF;
  IIC0_IBSR_IBIF=1;            // clear the interrupt event flag;
  while(IIC0_IBSR_RXAK);       // check for RXAK;
        //-----Slave ACK occurred------------
  IIC0_IBCR_RSTA = 1;          // set up repeated start;
  IIC0_IBDR = slaveread;       // (slave_address) | (RW = 1);
  while (!IIC0_IBSR_IBIF);     // wait until IBIF;
  IIC0_IBSR_IBIF=1;            // clear the interrupt event flag;
  while (IIC0_IBSR_RXAK);      // check for RXAK;
```

```
        //-----Slave ACK occurred------------
   IIC0_IBCR_TX_RX = 0;         // set up to receive;
   IIC0_IBCR_TXAK  = 1;         // acknowledge disable;
   RD_data = IIC0_IBDR;         // dummy read;
   while (!IIC0_IBSR_IBIF);     // wait until IBIF;
   IIC0_IBSR_IBIF=1;            // clear the interrupt event flag;
   IIC0_IBCR_MS_SL = 0;         // generate stop signal;
   RD_data = IIC0_IBDR;         // read right data;
 }
```

例11给出了9S12微控制器内核及24C512 EEPROM模块写入-模块读取功能的一个程序。

例11．9S12微控制器及24C512 EEPROM的模块写入-模块读取功能（使用MC9s12XDT512)

```
void IIC_write_block(word addr, byte len)
{
  byte page;

  Address = addr;              // load address;
  page = len;                  // load length of data pack to be written;
  if(page > MAX_PAGE) page = MAX_PAGE;  // set limit;
  IIC0_IBCR_TXAK = 0;          // TXAK = 0;
  IIC0_IBCR |= 0x30;           // RX/TX = 1; MS/SL = 1; TXAK = 0;
                               // And generate START condition;
  IIC0_IBDR = slavewrite;      // Address the slave and set up for master transmit;
  while (!IIC0_IBSR_IBIF);     // wait until IBIF;
  IIC0_IBSR_IBIF=1;            // clear the interrupt event flag;
  while(IIC0_IBSR_RXAK);       // check for RXAK;
  IIC0_IBDR = Addr_H;          // Send high byte of word address;
  while (!IIC0_IBSR_IBIF);     // wait until IBIF;
  IIC0_IBSR_IBIF=1;            // clear the interrupt event flag;
  while(IIC0_IBSR_RXAK);       // check for RXAK;
  IIC0_IBDR = Addr_L;          // Send low byte of word address;
  while (!IIC0_IBSR_IBIF);     // wait until IBIF;
  IIC0_IBSR_IBIF=1;            // clear the interrupt event flag;
  while(IIC0_IBSR_RXAK);       // check for RXAK;
  for(i=0;i<page;i++)
  {
    IIC0_IBDR = WRData[i];
    while (!IIC0_IBSR_IBIF);   // wait until IBIF;
    IIC0_IBSR_IBIF=1;          // clear the interrupt event flag;
    while(IIC0_IBSR_RXAK);     // check for RXAK;
  }
  IIC0_IBCR_MS_SL = 0;         // generate stop signal;
}

void IIC_read_block(word addr,word len)
{
  byte dummy;

  Address = addr;
  length = len;
  if(length > MAX_LENGTH) length = MAX_LENGTH;
  IIC0_IBCR_TXAK = 0;         // TXAK = 0;
  IIC0_IBCR |= 0x30;         // RX/TX = 1; MS/SL = 1;
                              // And generate START condition;
```

```
  IIC0_IBDR = slavewrite;     // Address the slave and set up for master transmit;
  while (!IIC0_IBSR_IBIF);  // wait until IBIF;
  IIC0_IBSR_IBIF=1;           // clear the interrupt event flag;
  while(IIC0_IBSR_RXAK);      // check for RXAK;
  IIC0_IBDR = Addr_H;        // Send high byte of word address;
  while (!IIC0_IBSR_IBIF);  // wait until IBIF;
  IIC0_IBSR_IBIF=1;           // clear the interrupt event flag;
  while(IIC0_IBSR_RXAK);      // check for RXAK;
  IIC0_IBDR = Addr_L;        // Send low byte of word address;
  while (!IIC0_IBSR_IBIF);  // wait until IBIF;
  IIC0_IBSR_IBIF=1;           // clear the interrupt event flag;
  while(IIC0_IBSR_RXAK);      // check for RXAK;
  IIC0_IBCR_RSTA = 1;        // set up repeated start;
  IIC0_IBDR = slaveread;
  while (!IIC0_IBSR_IBIF);  // wait until IBIF;
  IIC0_IBSR_IBIF=1;           // clear the interrupt event flag;
  while (IIC0_IBSR_RXAK);     // check for RXAK;
  IIC0_IBCR_TX_RX = 0;        // set up to receive;
  dummy = IIC0_IBDR;          // dummy read;
  for(i=0;i<length-1;i++)
  {
    while (!IIC0_IBSR_IBIF); // wait until IBIF;
    IIC0_IBSR_IBIF=1;         // clear the interrupt event flag;
    RDData[i] = IIC0_IBDR;    // save data to RAM;
  }
  IIC0_IBCR_TXAK  = 1;       // acknowledge disable;
  while (!IIC0_IBSR_IBIF);  // wait until IBIF;
  IIC0_IBSR_IBIF=1;           // clear the interrupt event flag;
  IIC0_IBCR_MS_SL = 0;        // generate stop signal;
  RDData[i] = IIC0_IBDR;      // save data to RAM;
}
```

3.4 中断应用举例

这一节列出了当IIC中断服务程序被使用时的两个选项：

- 微控制器用作主机，并从使用的从机中读取或写入
- 微控制器作为IIC总线上的从机

3.4.1 MCU作为主机

当主软件循环不能查询到IIC通信序列时，就会使用此程序。第一个动作仅在主循环中进行；这是第一次写入IIC数据寄存器。然后，IIC中断服务程序维持着整个通信。例12解释了如何设置微控制器。

例12．IIC模块初始化程序

```
void Init_IIC(void)
{
  IICF = 0;         // frequency divider register: Bus = 8MHz => SCL = 400kHz;
  IICA = 0;         // slave address of this module;
  IICS = 0x12;      // clear the IBAL and IICIF flags;
  IICC = 0xC0;      // enable IIC module with interrupt;
```

```
}
```

例13显示了用于主循环中EEPROM 24AA256和MC9S08QG8的读取模块和写入模块的功能。

例13. 读取模块和写入模块功能（使用24AA256 EEPROM和MC9S08QG8）

```
void IIC_read_block(word addr,byte len)
{
  Address = addr;
  length = len;
  i = 0;                  // for interrupt only;
  IIC_Res_flg = 0;        // for interrupt only;
  flag = 10;              // defines receive function in the interrupt service routine;
  if(length > MAX_LENGTH) length = MAX_LENGTH;
  IICC_TXAK  = 0;         // RX/TX = 1; MS/SL = 1, TXAK = 0;
  IICC |= 0x30;           // And generate START condition;
  IICD = IIC_SLAVE;       // Address the slave and set up for master transmit;
}

void IIC_write_block(word addr, byte len)
{
  Address = addr;          // load address;
  length = len;            // load length of data pack to be written;
  i = 0;                   // for interrupt only;
  IIC_Res_flg = 0;         // for interrupt only;
  flag = 2;                // defines transmit function in the interrupt service routine;
  if(length > MAX_PAGE) length = MAX_PAGE;  // set limit;
  IICC_TXAK = 0;           // RX/TX = 1; MS/SL = 1; TXAK = 0;
  IICC |= 0x30;            // And generate START condition;
  IICD = IIC_SLAVE;        // Address the slave and set up for master transmit;
}
```

标记变量就是在IIC中断服务程序中的字节变量，用以识别处理哪个通信协议。

例14显示了在主循环中串行EEPROM（24LC16B）和微控制器（S12DP256B）的读取功能。

例14. 主循环中的EEPROM（24LC16B）和MCU（S12DP256B）

```
void i2c_read_byte(unsigned char read_addr)
{
  flag =  0;
  temp = read_addr;
  IBCR_MS_SL = 1;         // Set transmit and master mode;
  IBCR_TX_RX = 1;         // And generate start condition;
  IBDR = slave_IIC_addr;
}
```

中间变量是一个普通的字节变量；标记变量就是在IIC中断服务程序中的字节变量，用以识别处理哪个通信协议的字节。然后，中断服务程序代码如例15如示。

例15.用于MC9S08QG8和24C512 EEPROM的中断服务程序编码

```
__interrupt void isrViic(void)
{
  IICS_IICIF = 1;               // clear the interrupt event flag;
  switch(flag)
  {
    case 2:                     // TX mode;
    {
      IICD = Addr_H;            // Send high byte of word address;
      flag++;
      return;
    }
    case 3:                     // TX mode;
    {
      IICD = Addr_L;            // Send low byte of word address;
      flag++;
      return;
    }
    case 4:                     // TX mode;
    {
      if(length)
      {
        IICD = WR_Data[i];
        length--;
        i++;
        return;
      }
      else
      {
        IICC_MST = 0;           // generate stop signal;
        flag = 0;
        IIC_Res_flg = IIC_OK;   // END of write cycle;
        return;
      }
    }
    case 10:
    {
      IICD = Addr_H;             // Send high byte of word address;
      flag++;
      return;
    }
    case 11:
    {
      IICD = Addr_L;            // Send low byte of word address;
      flag++;
      return;
    }
    case 12:
    {
      IICC_RSTA = 1;           // set up repeated start;
      IICD = IIC_SLAVE | 1;
```

```
      flag++;
      return;
    }
    case 13:
    {
      IICC_TX = 0;              // set up to receive;
      dummy = IICD;             // dummy read;
      flag++;
      return;
    }
    case 14:
    {
      if(length > 1)
      {
        RD_Data[i] = IICD;      // save data to RAM;
        length--;
        i++;
        return;
      }
      else if(length == 1)
      {
        IICC_TXAK  = 1;         // acknowledge disable;
        RD_Data[i] = IICD;       // save data to RAM;
        i++;
        flag++;
        return;
      }
    }
    case 15:
    {
      IICC_MST = 0;              // generate stop signal;
      //RD_Data[i] = IICD;       // save data to RAM;
      flag = 0;
      IIC_Res_flg = IIC_OK;
      return;
    }
  }
}
```

3.4.2 MCU作为从机

当微控制器用作IIC总线上的从机时，将使用此程序。微控制器需要获得所有为此而设计的IIC字节。有效字节的识别是基于保存在IBAD寄存器中的从机自身地址的匹配之上的。如果主机发送一个地址到IIC总线，当这个地址与其在IBAD寄存器中的自身地址相匹配时，从机产生一个IIC中断。然后，例如，从机将所有IIC总线中的字节保存到RAM缓冲器并随后进行处理。

例16显示出与同类的主MCU相连接（两个EVBS12DP256B板互联）的从微控制器（S12DP256B）

变量可定义为：

```
byte RD_data[20];
byte i2c_cnt, temp;
```

例16. 从初始化程序

```
void Init_IIC(void)
{
  IBFD = 0x2B;         // frequency divider register: Bus = 8MHz => SCL = 91kHz;
  IBAD = My_IIC_addr; // slave address of this module;
  IBCR = 0xC0;         // enable IIC module and interrupts, RX mode, Slave mode, ACK = 0;
  IBSR_IBAL = 1;       // clear the IBAL flag;
}
```

例17. 中断服务程序

```
__interrupt void I2C_ISR(void)
{
  if(IBSR_IAAS)
  {
    IBCR_TX_RX = IBSR_SRW;    // set Rx/Tx mode in accordance to received calling address byte
and clear the IAAS bit;
    IBSR = IBSR_IBIF_MASK;    // clear the flag;
    temp = IBDR;              // dummy read to initiate the read data byte;
    i2c_cnt = 0;              // clear the received data counter;
  }
  else
  {
    IBSR = IBSR_IBIF_MASK;    // clear the flag;
    RD_data[i2c_cnt] = IBDR; // save received data to data buffer;
    i2c_cnt++;
  }
}
```

4 结论

此应用笔记解释了在飞思卡尔的HC908、9S08和9S12中使用IIC模块的情况。用户程序在所提到的微控制器和板上进行了测试。串行EEPROM经常被用作IIC器件来使用并可用作一个极好的示例。如果除了IIC器件（例如，实时时钟[RTC]）您需要使用其它器件，您可以按照数据表所描述的那样，改变器件地址（RTC为0xD0，EEPROM为0xA0）并正确地管理字节序列。

本页有意留空

本页有意留空

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

文件编号：AN3291
第1版
03/2007

freescale™
飞思卡尔半导体