# Application Note

## 32-bit Cortex™-M0 MCU
## NuMicro® Family

I2C EEPROM应用指南

*目录*

# 1 简介

本文描述了NUC1xx在I2C EEPROM方面的应用

# 2 I2C 性能

- 兼容AMBA APB接口
- 兼容Philips I2C标准，支持主模式
- 主/从模式最快传输速度1M位/秒( 超快模式 )
- 主从之间双向数据传输
- 多主总线（无中央主机）
- 主机同时发送仲裁可以避免串行数据总线上的数据损坏
- 串行时钟同步允许不同比特率的设备通过串行总线通讯
- 串行时钟同步作为握手机制可以暂停及恢复传输
- 内嵌14位时间溢出计时器在I2C总线挂起和时间溢出时产生I2C中断
- 输出高电平需要上拉电阻，不支持同一总线上的单个设备断电
- 可编程控制时钟实现多种速率控制
- 支持7位寻址模式
- I2C总线控制器支持多地址识别（4个从地址屏蔽选择）

## 3 I2C 功能描述

CPU通过如下6个特殊功能寄存器与SIO口相连：**I2CON**(控制寄存器，C0H)，**I2STATUS**(状态寄存器，BDH), **I2DAT** (数据寄存器, BCH), **I2ADDR** (地址寄存器, C1H), **I2CLK** (时钟控制寄存器，BEH) 以及 **I2TOC** (时间溢出计时器, BFH) 所有I2C功能寄存器的8-31位都没有定义，这些位没有任何功能并且读到的值都是零。当I2C口通过设置ENS1为高而使能时，内部状态将被I2CON和I2C硬件逻辑控制。当一个新的状态码被产生并存储到I2STATUS中时，I2C中断标志(SI)自动置位，如果EI2此时为高，I2C中断将会产生，I2STATUS的高5位保存内部状态码，低3位总是0，这些内容一直保持到软件清除SI位，I2C在NUC1xx的基地址是0x4002_0000和0x4012_0000

### 3.1 地址寄存器 (I2ADDR)

I2C口配有4个从地址寄存器，当作为主模式时这些寄存器是没用的，在从模式中，高7位保存MCU的从地址，I2C将对符合任意一个从地址的内容作出反应

I2C口支持“全呼”功能，如果I2C硬件GC位置位，硬件将响应全呼地址（00H），清除GC位禁止全呼功能。当GC位置位，I2C在从模式可以接收到总线上主机发出的地址为（00H）的全呼信号。主模式中，在发出全呼信号到I2C总线前必须先清除AA位

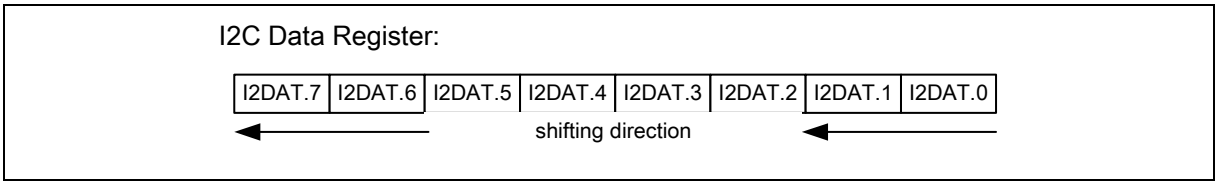
I2C总线控制器配有4个地址屏蔽寄存器支持多地址识别，当地址屏蔽寄存器中某一位设置为1时，接收到的地址相应位被屏蔽，被设为0时，接收到的地址相应位与地址寄存器中的值是一致的

| 符号 | 定义 | 地址 | MSB BIT ADDRESS, SYMBOL LSB<br>Bit 31 Bit7 Bit0 | | | 复位 |
|---|---|---|---|---|---|---|
| I2ADDR0 | I2C 地址0 | I2C_BA+04H | Reserved | ADDR0.6 ~ ADDR0.0 | GC0 | 0000 0000H |
| I2ADDR1 | I2C 地址1 | I2C_BA+18H | Reserved | ADDR1.6 ~ ADDR1.0 | GC1 | 0000 0000H |
| I2ADDR2 | I2C 地址2 | I2C_BA+1cH | Reserved | ADDR2.6 ~ ADDR2.0 | GC2 | 0000 0000H |
| I2ADDR3 | I2C 地址3 | I2C_BA+20H | Reserved | ADDR3.6 ~ ADDR3.0 | GC3 | 0000 0000H |
| I2ADRM0 | I2C 地址屏蔽0 | I2C_BA+24H | Reserved | ADRM0.6 ~ ADRM0.0 | Reserved | 0000 000XH |
| I2ADRM1 | I2C 地址屏蔽1 | I2C_BA+28H | Reserved | ADRM1.6 ~ ADRM1.0 | Reserved | 0000 000XH |
| I2ADRM2 | I2C 地址屏蔽2 | I2C_BA+2CH | Reserved | ADRM2.6 ~ ADRM2.0 | Reserved | 0000 000XH |
| I2ADRM3 | I2C 地址屏蔽3 | I2C_BA+30H | Reserved | ADRM3.6 ~ ADRM3.0 | Reserved | 0000 000XH |

### 3.2 数据寄存器 (I2DAT)

这个寄存器保存了8位要发送的数据或刚收到的数据，CPU可以在该寄存器没有移位操作的时候对其直接读写操作，这种情况发生时SIO在一个特定的状态并且串行中断标志位(SI)置位，I2DAT中的数据一直保持到SI位被清零。当数据被移出的时候总线上的数据被同时移入，I2DAT总是保存总线上最后一次出现的数据，就算仲裁失败也可以保证主机发送到从机的数据正确。

I2DAT和应答位组成一个9位移位寄存器，应答位由SIO硬件控制,CPU不能读写，串行数据在SCL时钟脉冲上升沿移入I2DAT。当1个字节数据被移入I2DAT，应答位（ACK或NACK）在第9个时钟脉冲返回。串行数据在SCL时钟脉冲下降沿从I2DAT中移出，在上升沿移入I2DAT



I2C 数据移位方向

| 标号 | 定义 | 地址 | MSB BIT ADDRESS, SYMBOL LSB<br>Bit31 Bit7 Bit0 | | 复位 |
|---|---|---|---|---|---|
| I2DAT | I2C DATA REGISTER | I2C_BA+08H | Reserved | I2DAT.7 ~ I2DAT.0 | 000000xxh |

### 3.3 控制寄存器 (I2CON)

CPU可以读写这个8位寄存器，其中2位受硬件影响：SI位在I2C硬件请求中断时置位，STO位在总线出现STOP条件时被清零，当ENS1="0"时STO位也被清零

EI 使能中断

ENSI 使能I2C功能模块。当ENSI=1时I2C功能使能，SDA1和SCL1口必须被设为逻辑高

STA I2C 开始标志，设置STA=1进入主模式，当总线空闲时I2C硬件发送一个START或重复START条件到总线上

STO I2C 停止标志，在主模式中设置STO可以发送STOP条件到总线上，I2C硬件将检测总线，如果一个STOP条件被检测到，STO位被硬件自动清零。在从模式中，设置STO可以复位I2C硬件为“未赋址”从模式，也就是已经不在从接收模式，不会再接收主发送设备的数据

SI I2C 中断标志，当新的SIO状态出现在I2STATUS寄存器中，SI标志被硬件置位，如果EA和EI 2位都被置位，将会产生I2C中断请求。SI位必须被软件写“1”清零。

AA 应答产生控制位，当AA=1时，在地址或数据接收之前，应答位（SDA低电平）将在如下两种情况的应答时钟脉冲周期被返回：1）从机应答主机发送的地址，2)接收设备应答发送设备的数据。当AA=0时，在地址或数据接收之前，无应答位（SDA高电平）将在SCL线应答时钟脉冲周期被返回

| 标号 | 定义 | 地址 | MSB BIT ADDRESS, SYMBOL LSB<br>Bit31 Bit7 Bit6 Bit5 Bit4 Bit3 Bit2 Bit0 | | | | | | | | 复位 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| I2CON | I2C CONTROL REGISTER | I2C_BA+00H | Reserved | EI | ENS1 | STA | STO | SI | AA | - | - | 0000 0000H |

### 3.4 状态寄存器 (I2STATUS)

I2STATUS是一个8位只读寄存器，低3位总是0，高5位包含了状态码，共有26种可能的状态码，当I2STATUS为F8H时，没有中断产生，所有其他I2STATUS值对应于定义的SIO状态，其中任一个状态进入后状态中断就会产生（SI=1）。一个有效的状态码将在硬件将SI置位后一个机器周期进入I2STATUS并一直保持到软件清除SI后一个机器周期。

此外，状态00H表示总线错误，在帧格式的错误位置出现START或STOP条件将产生总线错误。错误位置例如：正在传输一个地址字节，数据字节或者应答位的时候。要想从I2C总线错误中恢复过来，需要设置STO，清除SI，进入未赋址从模式，然后清除STO释放总线并等待新的通讯，在总线错误发生的过程中I2C总线无法识别停止条件

| 符号 | 定义 | 地址 | MSB BIT ADDRESS, SYMBOL LSB<br>Bit31 Bit8 Bit7 Bit0 | | 复位 |
|---|---|---|---|---|---|
| I2STATUS | I2C STATUS REGISTER | I2C_BA+0CH | Reserved | 8 bit status data | 000000F8H |

### 3.5 I2C 波特率控制寄存器 (I2CLK)

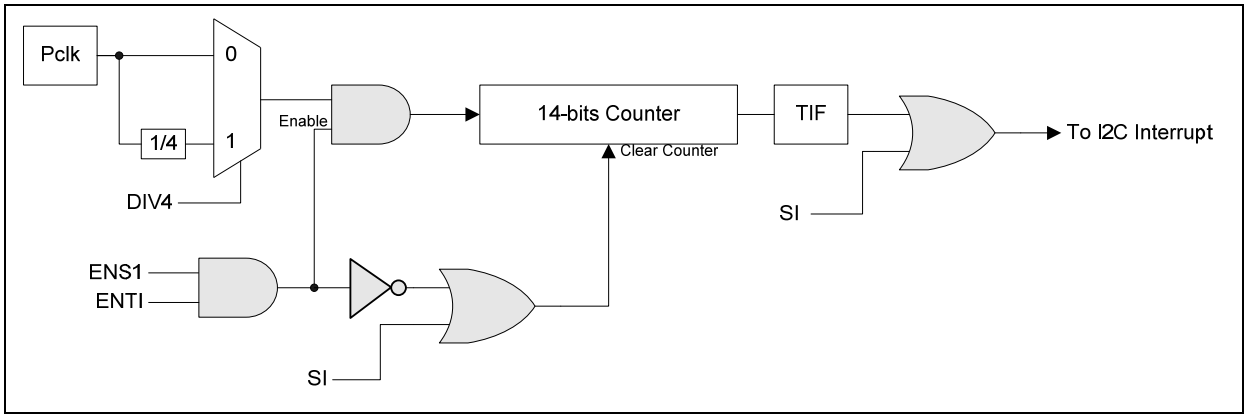
当SIO在主模式时，I2C数据波特率取决于I2CLK寄存器，SIO在从模式时自动与主I2C设备时钟同步，最高到400KHz。

I2C设定数据波特率为： I2C数据波特率 = PCLK /(4x(I2CLK+1))，如果PCLK=16MHz, I2CLK = 40(28H), 所以 I2C数据波特率= 16MHz/(4X (40 +1)) = 97.5K位/秒，如下表所示

| 符号 | 定义 | 地址 | MSB BIT ADDRESS, SYMBOL LSB<br>Bit31 Bit8 Bit7 Bit0 | | 复位 |
|---|---|---|---|---|---|
| I2CLK | I2C CLOCK RATE | I2C_BA+10H | Reserved | I2CLK.7 ~ I2CLK.0 | 0000 0000H |

### 3.6 I2C时间溢出计数寄存器 (I2TOC)

一个14位时间溢出计数器可以用来处理I2C总线挂起事件，如果时间溢出计数器被使能，计数器立即开始计数直到溢出（TIF=1）并产生I2C中断请求，ENTI清零停止计数。当时间溢出计数使能时，设置标志SI为高复位计数器并在SI清零后重新开始计数。如果I2C总线挂起，将导致I2STATUS和SI标志在一段时间内不会被更新，14位计数器可能溢出并通过I2C中断通知CPU，请参考下图，用户可以对TIF位写1清0.



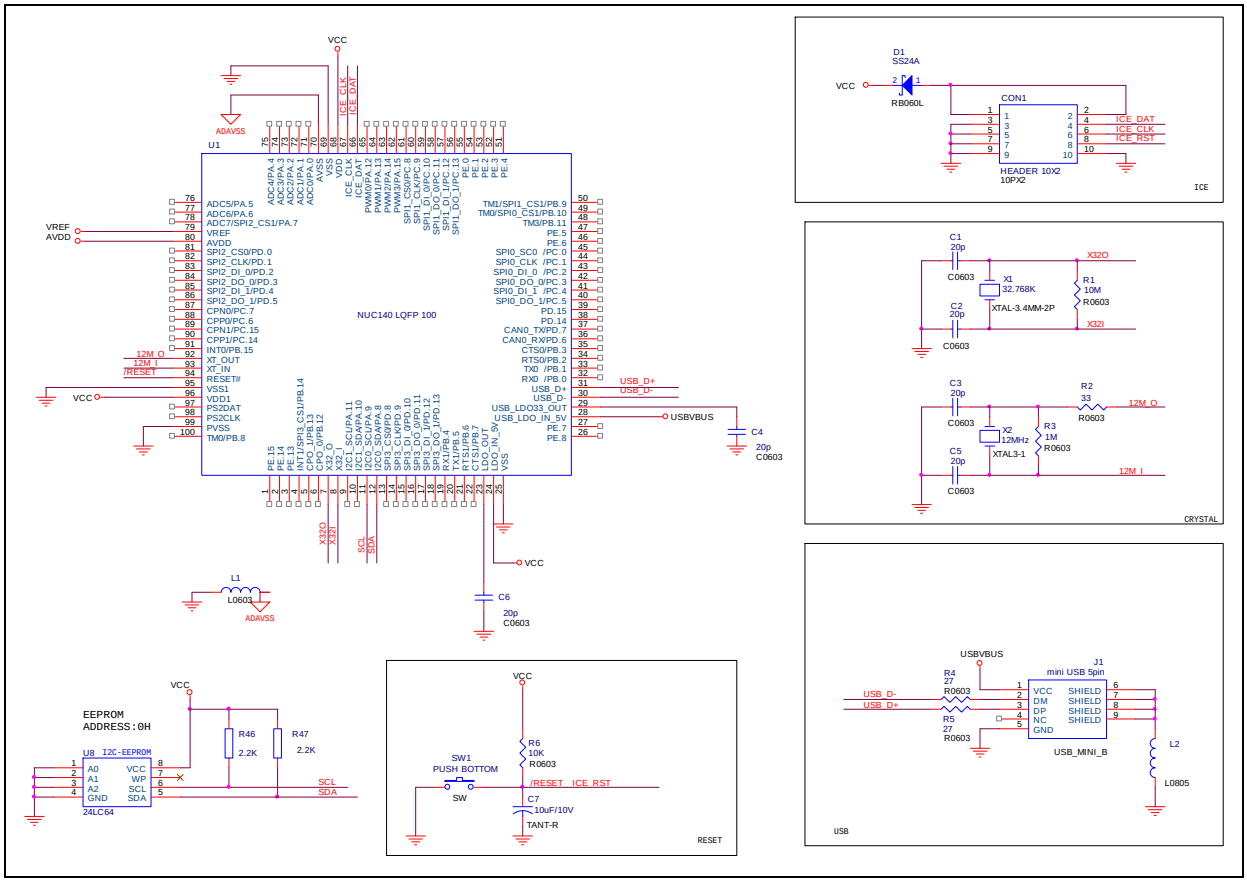

I2C 时间溢出计数器框图

| 符号 | 描述 | 地址 | MSB BIT ADDRESS, SYMBOL LSB | | | | 复位 |
|---|---|---|---|---|---|---|---|
| | | | Bit31 | Bit2 | Bit1 | Bit0 | |
| I2TOC | I2C TIME-OUT COUNTER REGISTER | I2C_BA+14H | Reserved | ENTI | DIV4 | TIF | 0000 0000B |

## 4 EEPROM

I2C EEPROM样片24LC64, 请参考24LC64规格书

## 5 电路



## 6 示例代码

```
#include <stdio.h>
#include "NUC1xx.h"
#include "Driver\DrvGPIO.h"
#include "Driver\DrvI2C.h"
#include "Driver\DrvSYS.h"
#include "Driver\DrvUART.h"

/*---------------------------------------------------------------------------------------------------------*/
/*MAIN                                                                          Function*/
/*---------------------------------------------------------------------------------------------------------*/

int main (void)
{
        uint32_t u32HCLK;
        /* SYSCLK =>12Mhz*/
        UNLOCKREG();
    SYSCLK->PWRCON.XTL12M_EN = 1;
        u32HCLK = DrvSYS_GetHCLK() * 1000;



        /* Set I2C I/O */
        DrvGPIO_InitFunction(FUNC_I2C0);

        /* Open I2C0 and set clock = 100Kbps */
        DrvI2C_Open(I2C_PORT0, u32HCLK, 100000);

        //send i2c start
        DrvI2C_Ctrl(I2C_PORT0, 1, 0, 0, 0);             //set start
        while (I2C0->CON.SI == 0);                      //poll si flag
        //send writer command
        I2C0->DATA = 0XA0;                              //send writer command
        DrvI2C_Ctrl(I2C_PORT0, 0, 0, 1, 0);             //clr si flag
        while( I2C0->CON.SI == 0 );                     //poll si flag
```

```
//send address high
I2C0->DATA = 0X00;
DrvI2C_Ctrl(I2C_PORT0, 0, 0, 1, 1);          //clr si and set ack
while( I2C0->CON.SI == 0 );                  //poll si flag
//send address low
I2C0->DATA = 0X01;
DrvI2C_Ctrl(I2C_PORT0, 0, 0, 1, 1);          //clr si and set ack
while( I2C0->CON.SI == 0 );                  //poll si flag
//send data
I2C0->DATA = 0X55;                           //write data to
DrvI2C_Ctrl(I2C_PORT0, 0, 0, 1, 1);          //clr si and set ack
while( I2C0->CON.SI == 0 );                  //poll si flag
//send i2c stop
DrvI2C_Ctrl(I2C_PORT0, 0, 1, 1, 0);          //send stop
DrvI2C_Close(I2C_PORT0);



/* Open I2C0 and set clock = 100Kbps */
DrvI2C_Open(I2C_PORT0, u32HCLK, 100000);
//send i2c start
DrvI2C_Ctrl(I2C_PORT0, 1, 0, 0, 0);          //set start
while (I2C0->CON.SI == 0);                   //poll si flag

//send writer command
I2C0->DATA = 0XA0;
DrvI2C_Ctrl(I2C_PORT0, 0, 0, 1, 0);          //clr si
while( I2C0->CON.SI == 0 );                  //poll si flag

//send address high
I2C0->DATA = 0X00;
DrvI2C_Ctrl(I2C_PORT0, 0, 0, 1, 1);          //clr si and set ack
while( I2C0->CON.SI == 0 );                  //poll si flag

//send address low
```

```
        I2C0->DATA = 0X01;
        DrvI2C_Ctrl(I2C_PORT0, 0, 0, 1, 1);          //clr si and set ack
        while( I2C0->CON.SI == 0 );                  //poll si flag

        //send start flag
        DrvI2C_Ctrl(I2C_PORT0, 1, 0, 1, 0);          //clr si and send start
        while( I2C0->CON.SI == 0 );                  //poll si flag

        //send read command
        I2C0->DATA = 0XA1;
        DrvI2C_Ctrl(I2C_PORT0, 0, 0, 1, 0);          //clr si
    while( I2C0->CON.SI == 0 );                      //poll si flag

        //resive data
        I2C0->DATA = 0X00;
        DrvI2C_Ctrl(I2C_PORT0, 0, 0, 1, 0);          //clr si
        while( I2C0->CON.SI == 0 );                  //poll si flag

        //send i2c stop
        DrvI2C_Ctrl(I2C_PORT0, 0, 1, 1, 0);          //clr si and set stpo
        DrvI2C_Close(I2C_PORT0);

    while(1);
}
```

版本历史

| REV. | DATE | DESCRIPTION |
|---|---|---|
| 0.01 | March 11, 2010 | 1. Initially issued. |

## Important Notice

**Nuvoton products are not designed, intended, authorized or warranted for use as components in systems or equipment intended for surgical implantation, atomic energy control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, combustion control instruments, or for other applications intended to support or sustain life. Further more, Nuvoton products are not intended for applications wherein failure of Nuvoton products could result or lead to a situation wherein personal injury, death or severe property or environmental damage could occur.**

**Nuvoton customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Nuvoton for any damages resulting from such improper use or sales.**

---

**Please note that all data and specifications are subject to change without notice. All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.**