M030G I²C Slave 400 kbps

NuMicro® 32 位系列微控制器范例代码介绍

文件信息

应用简述	本范例代码使用 M030G 实现 I²C Slave 模式支持 400 kbps
BSP 版本	M029G_M030G_M031G_Series_BSP_CMSIS_V3.04.000
开发平台	NuMaker-M030GTD_V1.1 NuMaker-M031GTD_V1.0

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1. 概述

本范例代码使用 M030G 实现 I²C Slave 模式下支持 400 kbps 速率。

I²C 总线只需要两根线分别是 SDA（串行数据线）和 SCL（串行时钟线），都是双向 I/O 线。接口电路为开漏输出，需通过上拉电阻接到电源 VCC。协议规定标准模式下可达 100 kbps，快速模式下可达 400 kbps，高速模式下可达 3.4 Mbps。本参考代码介绍基于 M030G 如何实现 I²C Slave 支持 400 kbps。

1.1 原理

M030G 自带 2 个 I²C 外设，都可以设置为 Master 或 Slave。各自有一个独立的中断向量和中断状态寄存器。

当 I²C 外设设置为 Slave，如果 I²C 总线有数据在传输，I²C 外设一旦检测总线上状态发生变化，就会产生 I²C 中断。固件需要根据 I²C 协议，对不同的状态进行相应的处理。在 MCU 主频固定的情况，I²C 中断对各状态处理的更快，I²C 速率也就会更高。

当产生 I²C 中断的时候，I²C 中断标识 SI 将置 1。在将 SI 清零之前，总线由 I²C 外设控制，中断状态寄存器一直保持不变。SI 清零后，I²C 外设立即释放总线，开始后续的传输。因此可以在全部获取当前有效状态后，尽早将 SI 清零，以提高传输速率。

针对某些应用例如光模块，I²C 从机有多个设备地址，而且数据结构较复杂。当从机处于发送数据状态，如果此时需要通过查找表找到待发送数据，这样会比较耗时，进而影响 I²C 通信速率。如果可以在其他状态下提前准备待发送数据，当从机处于发送数据状态，直接将提前准备好的数据发送出去，也可以提高传输速率。

1.2 执行结果

本范例代码正确编译后烧录到评估板 M030G 作从机；为了主机能输出更高速率时钟信号，将 *I2C_Master_400_kbps.bin* 烧录到另一块评估板 M031G 实现主机。通过设置主、从机板拨码开关使能其各自虚拟串口，然后使用 2 条 USB 线将主、从机连接到同一 PC。在 PC 上打开 2 个对应的串口调试助手，然后根据主机串口提示，输入数据进行测试，如图 1-1 所示。

```
CPU @ 72000000Hz

Enter any a data to start write. 0xA0[0x00~0x03] = 0x00~0x03.
Write 0xA0[0x00~0x03] finished.

Enter any a data to start current read. 0xA0[0x04~0x07] = 0x04~0x07.
->A0[0x4] is 0x4
->A0[0x5] is 0x5
->A0[0x6] is 0x6
->A0[0x7] is 0x7
Current read A0[0x04~0x07] successful.

Enter any a data to start random read. 0xA0[0x20~0x23] = 0x20~0x23.
->A0[0x20] is 0x20
->A0[0x21] is 0x21
->A0[0x22] is 0x22
->A0[0x23] is 0x23
Random read A0[0x20~0x23] successful.
```

图 1-1 主机串口调试信息

I²C Master 向从机设备地址为 0xA0 寄存器起始地址为 0x00，连续写 0x00、0x01、0x02、0x03，波形如图 1-2 所示。

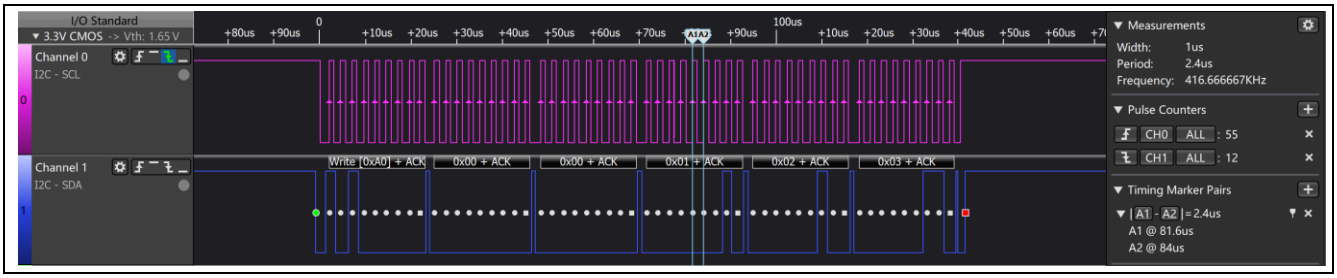

图 1-2 连续写

之后 I²C Master 向从机设备地址为 0xA0，执行当前连续读 4 个字节，所读数据为 0x04、0x05、0x06、0x07，波形如图 1-3 所示。

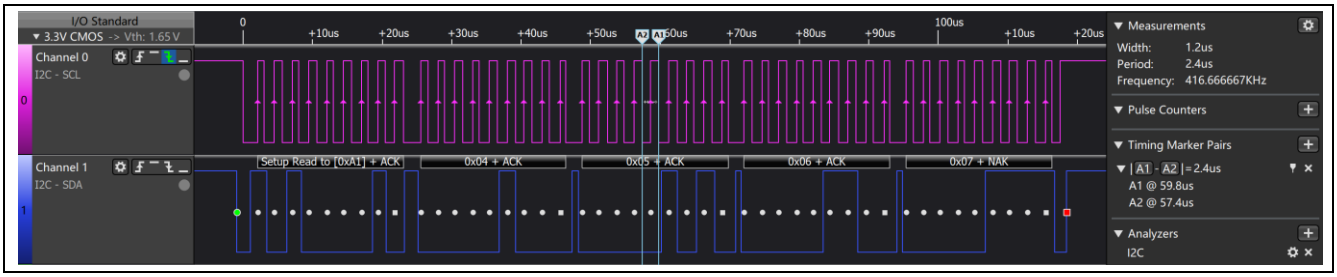

图 1-3 当前地址连续读

之后 I²C Master 向从机设备地址为 0xA0 寄存器起始地址为 0x20，连续读 4 个字节，所读数据为 0x20，0x21，0x22，0x23，波形如图 1-4 所示。

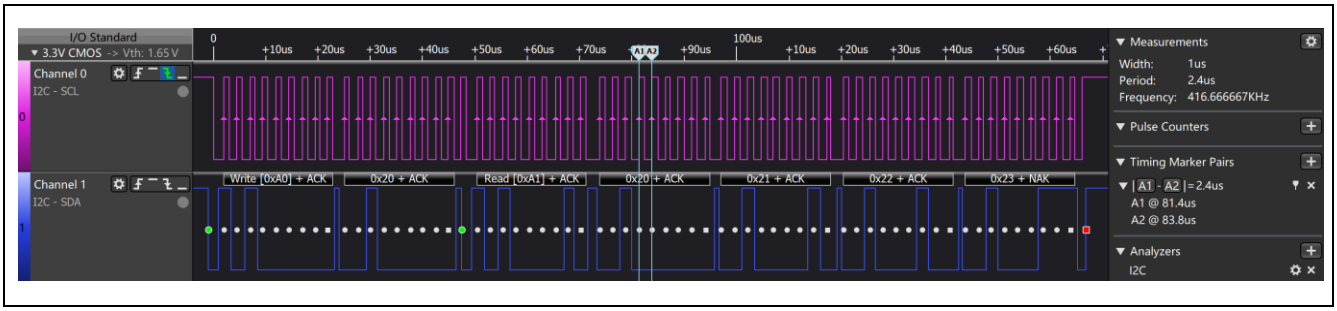

图 1-4 随机地址连续读

2. 代码介绍

从机设备地址、Table 数据长度及 I²C 各中断状态宏定义，代码位于 main()函数。

```
#define I2C_DEV_ADDR                        0xA0
#define TABALE_SIZE                         0x100
#define SLAVE_TRANSMIT_DATA_ACK             0xB8
#define SLAVE_TRANSMIT_ADDRESS_ACK          0xA8
#define SLAVE_TRANSMIT_REPEATSTART_OR_STOP  0xA0
#define SLAVE_RECEIVE_ADDRESS_ACK           0x60
#define SLAVE_RECEIVE_DATA_ACK              0x80
#define SLAVE_TRANSMIT_DATA_NACK            0xC0
#define SLAVE_RECEIVE_DATA_NACK             0x88
```

初始化各 Table 数据，代码位于 main()函数。

```
/* Initialize every table */
for (i = 0; i < TABALE_SIZE; i++)
{
    g_u8Tab_A0[i] = i;
    g_u8Tab_A2[i] = i + 1;
}
```

I²C 初始化函数，设置从机设备地址分别为 0xA0、0xA2，代码位于 I2C0_Init ()函数。

```
/* Open I2C module and set bus clock */
I2C_Open(I2C0, 100000);

/* Set I2C 4 Slave Addresses */
I2C_SetSlaveAddr(I2C0, 0, (I2C_DEV_ADDR >> 1), 0); /* Slave Address : 0xA0        */
I2C_SetSlaveAddr(I2C0, 1, 0x00, 0);                /* Slave Address : not be used */
I2C_SetSlaveAddr(I2C0, 2, 0x00, 0);                /* Slave Address : not be used */
I2C_SetSlaveAddr(I2C0, 3, 0x00, 0);                /* Slave Address : not be used */
I2C_SetSlaveAddrMask(I2C0, 0, 0x01);               /* 1:Mask Enabled.
                           Set bit0 for supporting two slave addresses(0xA0,0xA2) */

/* Enable I2C interrupt */
I2C_EnableInt(I2C0);
NVIC_EnableIRQ(I2C0_IRQn);
```

进入 I²C 中断后，先读取状态寄存器及数据寄存器数据，然后清 SI 标志。代码位于 I2C0_IRQHandler()函数中。

```
    /* Read status register of I2C0 */
    u32Sta = (uint8_t)I2C_GET_STATUS(I2C0);

    /* Read data register of I2C0 */
    u8Dat0 = I2C_GET_DATA(I2C0);

    /* Clear SI,Set AA */
    I2C_SET_CONTROL_REG(I2C0, I2C_CTL_SI_AA);
```

产生中断后如果状态寄存器为 0xA0 即接收到 Restart 信号，此时准备待发送数据，并存储于 u8PreDat_A0。代码位于 I2C0_IRQHandler()函数。

```
    else if (u32Sta == SLAVE_TRANSMIT_REPEATSTART_OR_STOP)  /* A STOP or repeated START
                                has been received while still addressed as Slave/Receiver*/
    {
        blData = FALSE;

        /* Prepare the first data to be sent later.*/
        if (blDevAddr_A0)
        {
            u8PreDat_A0 = g_u8Tab_A0[Data_Addr_A0++];
        }
        else
        {
            u8PreDat_A2 = g_u8Tab_A2[Data_Addr_A2++];
        }
    }
```

产生中断后如果状态寄存器为 0xA8 即接收到 SLA+R 信号，此时直接发送前面已准备数据 u8PreDat_A0，然后再准备下一个待发送数据，并存储于 u8PreDat_A0。代码位于 I2C0_IRQHandler()函数中。

```
        else if (u32Sta == SLAVE_TRANSMIT_ADDRESS_ACK)    /* Own SLA+R has been receive;
ACK has been return */
        {
            if (--u8Dat0 == I2C_DEV_ADDR)
            {
                blDevAddr_A0 = TRUE;
                I2C_SET_DATA(I2C0, u8PreDat_A0);          /* Send first data */
                u8PreDat_A0 = g_u8Tab_A0[Data_Addr_A0++]; /* Prepare second data */
            }
            else
            {
                blDevAddr_A0 = FALSE;
                I2C_SET_DATA(I2C0, u8PreDat_A2);
                u8PreDat_A2 = g_u8Tab_A2[Data_Addr_A2++];
            }
        }
```

产生中断后如果状态寄存器为 0xB8 即 I2CDAT 中数据已发送完成，此时直接发送前面已准备数据 u8PreDat_A0，然后再准备下一个待发送数据，并存储于 u8PreDat_A0。代码位于 I2C0_IRQHandler()函数中。

```
        if (u32Sta == SLAVE_TRANSMIT_DATA_ACK) /* Data byte in I2CDAT has been transmitted
ACK has been received */
        {
            if (blDevAddr_A0)
            {
                I2C_SET_DATA(I2C0, u8PreDat_A0);/* Send second data */
                u8PreDat_A0 = g_u8Tab_A0[Data_Addr_A0++];
```

```
            }
            else
            {
                I2C_SET_DATA(I2C0, u8PreDat_A2);
                u8PreDat_A2 = g_u8Tab_A2[Data_Addr_A2++];
            }
        }
```

3. 软件与硬件需求

3.1 软件需求

- BSP 版本
  - M029G_M030G_M031G_Series_BSP_CMSIS_V3.04.000
- IDE 版本
  - Keil uVision 5.36

3.2 硬件需求

- 电路组件
  - NuMaker-M030GTD_V1.1
  - NuMaker-M031GTD_V1.0
- 线路示意图
  - 将从机 I2C0 SCL (PF.3) 、I2C SDA(PF.2) 、GND 引脚分别与主机 SCL (PB.4)、SDA (PB.4)、GND 相连。注意在从机上分别使用跳线将 PF.3 与 JP7 短接，实现 SCL 引脚上拉；将 PF.2 与 JP8 短接实现 SDA 引脚上拉。

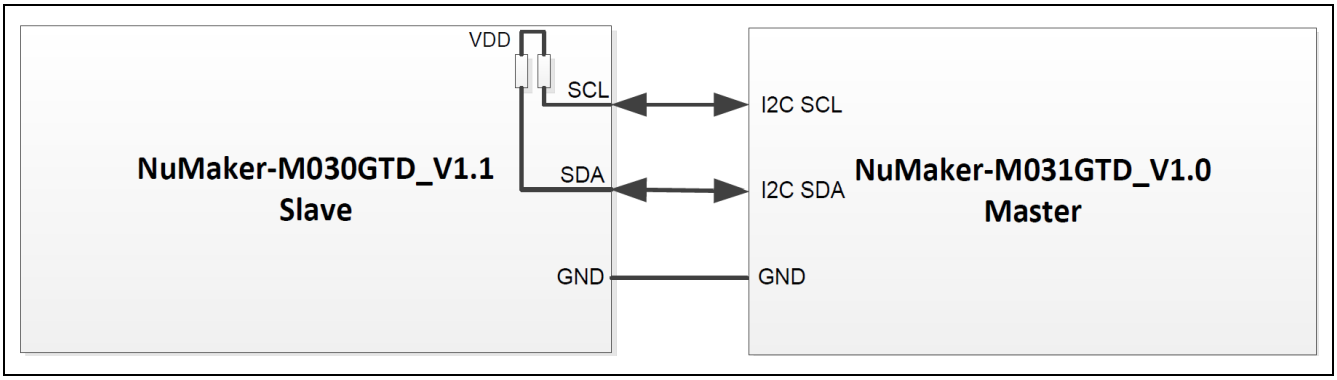

图 3-1 线路示意图

4. 目录信息

目录结构如下图所示。

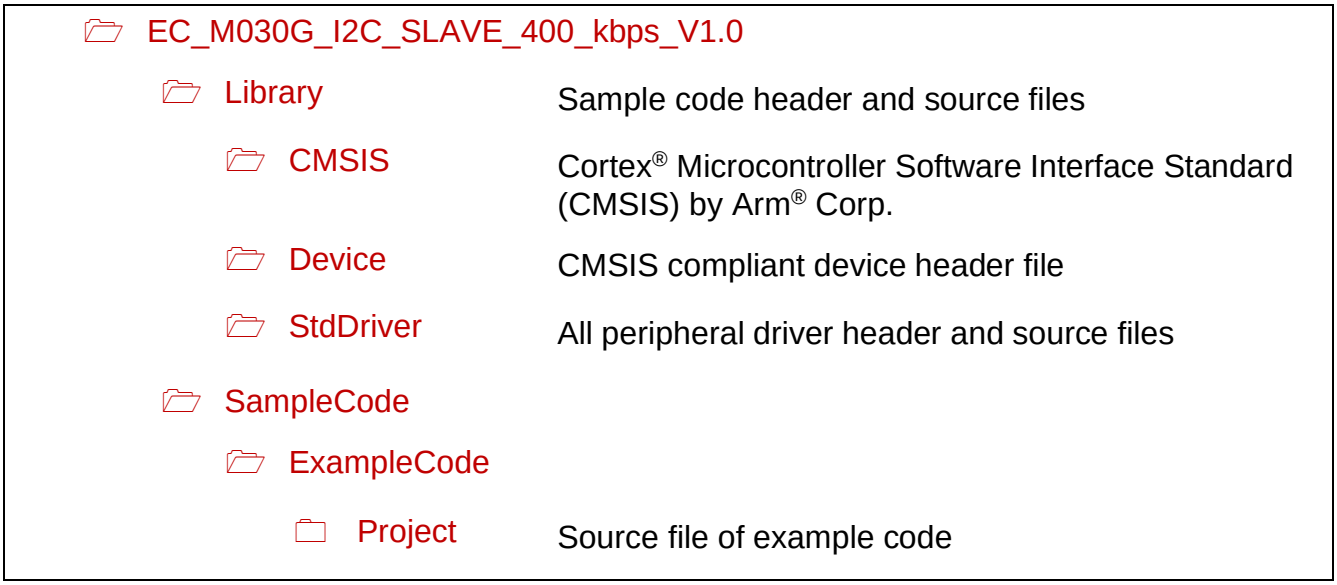

图 4-1 目录信息

5. 范例程序执行

1. 根据目录信息章节进入 ExampleCode 路径中的 KEIL 文件夹，双击 *M030G_I2C_Slave_400_kbps.uvproj*。
2. 进入编译模式界面
   - 编译
   - 下载代码至内存
   - 进入 / 离开仿真模式
3. 进入仿真模式界面
   - 执行代码

6. 修订纪录

Date	Revision	Description
2025.03.31	1.00	初始发布。

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

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.