M030G I²C Slave 400 kbps

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example code uses the M030G series microcontroller (MCU) to implement I²C slave mode support for 400 kbps.
BSP Version	M029G_M030G_M031G_Series_BSP_CMSIS_V3.04.000
Hardware	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. Overview

Based on the M030G series microcontroller (MCU), this example code implements I²C slave support of 400 kbps.

The I²C bus requires only two wires, namely SDA (serial data line) and SCL (serial clock line), which are bi-direction I/O lines. The interface circuit is an open leakage output and needs to be connected to the power supply VCC through a pull-up resistor. The protocol specifies up to 100 kbps in standard mode, 400 kbps in fast mode, and 3.4 Mbps in high-speed mode. This reference code describes how to achieve I²C slave support of 400 kbps based on M030G.

1.1 Principle

The M030G supports two I²C devices, both of which can be set as master or slave. Each has an independent interrupt vector and interrupt status register.

When the I²C device is set to slave mode, if the I²C bus has data in transmission, once the I²C device detects the status change on the bus, it will generate an I²C interrupt. The firmware needs to handle different states according to the I²C protocol. In the case of a fixed MCU main frequency, the I²C interrupt processes each state faster, and the I²C rate will be higher.

When an I²C interrupt is generated, the I²C interrupt identifier, also called SI, will be set to 1. The bus is controlled by an I²C device and the interrupt status register remains unchanged until the SI is cleared. After the SI is cleared, the I²C device immediately releases the bus and begins subsequent transmission. Therefore, the SI can be cleared as soon as possible after all the current valid state is obtained to improve the transmission rate.

For some applications such as optical modules, the I²C slave has multiple device addresses and a complex data structure. When the slave is in data sending state, it takes time to find the data to be sent from lookup table, which affects the I²C communication rate. If data can be prepared in advance in other states, when the slave is in the data sending state, the prepared data can be directly sent to improve the transmission rate.

1.2 Demo Result

Compile this sample code and program it to M030G of the NuMaker-M030GTD. The M030G serves as the I²C slave. In order for the host to output a clock signal with a higher rate, program the *I2C_Master_400_kbps.bin* file to M031G of the NuMaker-M031GTD. The M031G serves as the I²C master. Enable their virtual serial ports by setting DIP switches on the master board and slave board. Then connect the master and slave to the same PC using two USB cables. Open two corresponding serial debugging assistants on the PC, and then input data according to the prompts of the host serial port for testing, as shown in Figure 1-1.

```
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.
```

Figure 1-1 Debugging Information About the Serial Port on the Host

The I²C master writes 0x00, 0x01, 0x02, and 0x03 to the I²C slave. The device address of the slave is 0xA0, and the start address of the register is 0x00. The waveform is shown in Figure 1-2.

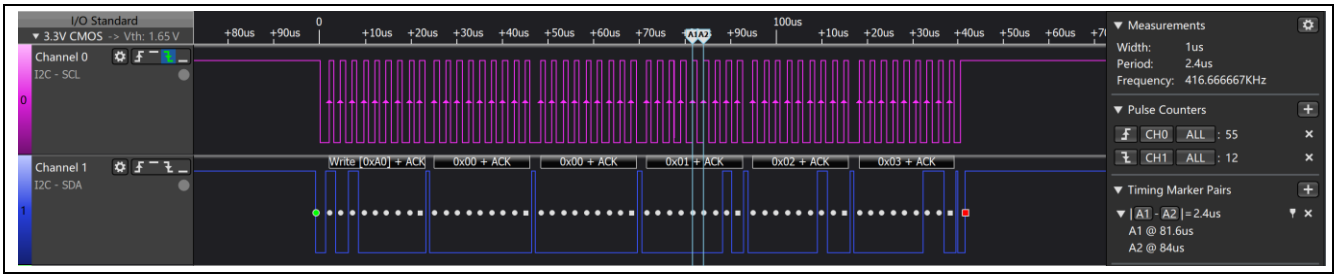

Figure 1-2 Continuous Writing

The I²C master reads four bytes consecutively from the I²C slave. The read data are 0x04, 0x05, 0x06, and 0x07. The device address of the slave is 0xA0. The waveform is shown in Figure 1-3.

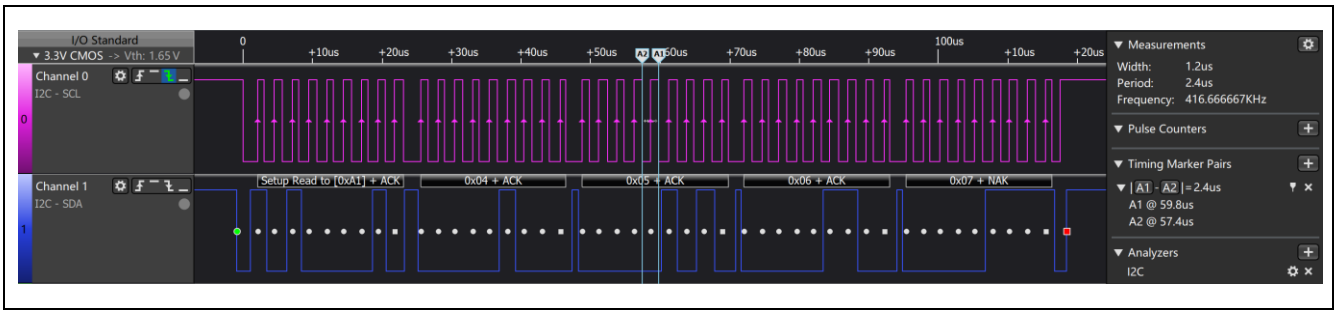

Figure 1-3 Current Continuous Read

The I²C master reads four bytes consecutively from the I²C slave. The data read are 0x20, 0x21, 0x22, and 0x23. The device address of the slave is 0xA0 and the register address is 0x20. The waveform is shown in Figure 1-4.

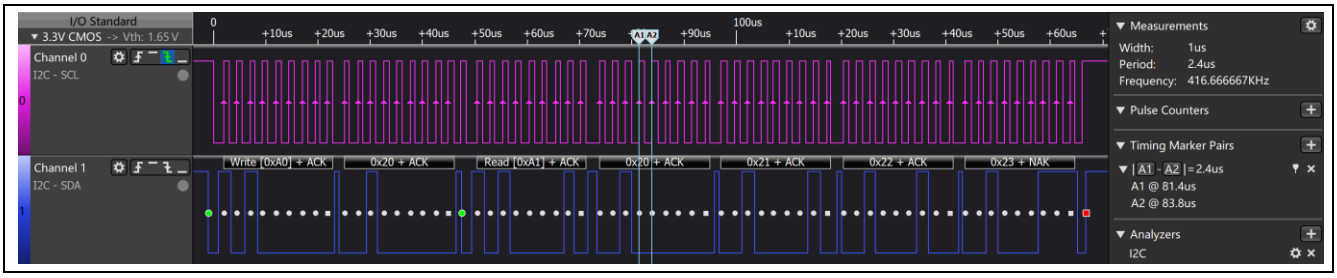

Figure 1-4 Random Address Read Continuously

2. Code Description

The slave's device address, table data length and I²C interrupt states are defined in the main() function.

```
#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
```

Each table data is initialized. The code is located in the main() function.

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

The I²C initialization function sets the slave's device addresses to 0xA0 and 0xA2, respectively. The code is located in the I2C0_Init () function.

```
/* 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);
```

After entering the I²C interrupt, first read data of the status register and data register, and then clear the SI flag. The code is located in the I2C0_IRQHandler() function.

```
    /* 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);
```

If the value of the status register is 0xA0, that means the restart signal has been received, prepare data for the next data sending state, which is stored in the variable u8PreDat_A0. The code is located in the I2C0_IRQHandler() function.

```
    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++];
        }
    }
```

If the value of the status register is 0xA8, that means the SLA+R signal has been received. At this time, the prepared data u8PreDat_A0 is directly sent, and then the next data to be sent is prepared and stored in u8PreDat_A0. The code is located in the I2C0_IRQHandler() function.

```
        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++];
            }
        }
```

If the value of status register is 0xB8, that means the data in I2CDAT has been sent. At this time, the prepared data u8PreDat_A0 is directly sent, and then the next data to be sent is prepared and stored in u8PreDat_A0. The code is located in the I2C0_IRQHandler() function.

```
        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. Software and Hardware Requirements

3.1 Software Requirements

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

3.2 Hardware Requirements

- Circuit components
  - NuMaker-M030GTD V1.1
  - NuMaker-M031GTD V1.0
- Pin Connect
  - Connect the I2C0 SCL (PF.3), I2C SDA(PF.2), and GND of the slave to the SCL (PB.5), SDA (PB.4), and GND of the host respectively. Note that the PF.3 and JP7 need to be connected to achieve pull-up on the SCL pin using a jumper and the PF.2 and JP8 need to be connected to achieve pull-up on the SDA pin using a jumper at the slave.

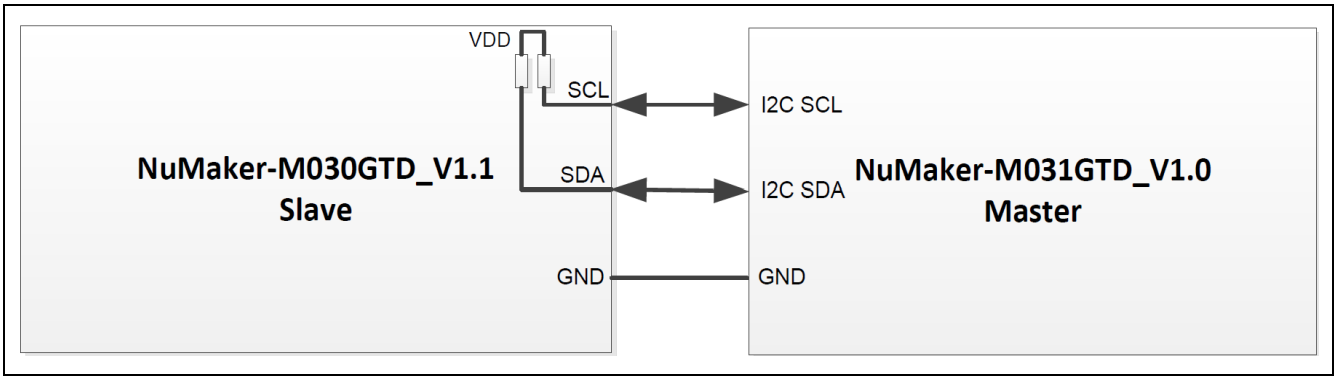

Figure 3-1 Pin Connect

4. Directory Information

The directory structure is shown below.

EC_M030G_I2C_SLAVE_400_kbps_V1.00	
Library	Sample code header and source files
CMSIS	Cortex® Microcontroller Software Interface Standard (CMSIS) by Arm® Corp.
Device	CMSIS compliant device header file
StdDriver	All peripheral driver header and source files
SampleCode	
ExampleCode	
Project	Source file of example code

Figure 4-1 Directory Structure

5. Example Code Execution

1. Browse the sample code folder as described in the Directory Information section and double-click *M030G_I2C_Slave_400_kbps.uvproj*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6. Revision History

Date	Revision	Description
2025.03.31	1.00	Initial version.

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.