M030G Emulating MDIO and Parsing Command

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example code uses M030G I²C to receive operation command and SPI to emulate MDIO waveform for controlling Ethernet PHY (IP101GR).
BSP Version	M029G_M030G_M031G_Series_BSP_CMSIS_V3.03.000
Hardware	NuMaker-M030GTD V1.1

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

This example code uses M030G I²C to receive operation command and SPI to emulate serial management interface (MDC/MDIO) waveform for controlling Ethernet PHY.

Most of the NuMicro Family does not support MDIO master interface for PHY device configuration. This example code provides the solution that can be widely used in the NuMicro Family for accessing Ethernet PHY.

This example code requires additional external hardware modules, including a I²C master and an Ethernet PHY. A I²C master sends Write or Read command to trigger SPI to emulate Write or Read waveform. The part number of Ethernet PHY is IP101GR made by IC+.

1.1 Principle

The example code can be divided into two parts, the role of I²C slave for receiving command and the role of MDIO master for accessing Ethernet PHY. The following section will focus on how to parse I²C command, and instructions on how to generate the corresponding MDIO waveform.

1.1.1 I²C Data Format

Figure 1-1 and Figure 1-2 show that the I²C master reads and writes data protocol. A protocol consists of five parts:

- START (STA) or Repeated START (ST) signal generation
  - The master can initiate a transfer by sending a START signal; the START signal denotes the beginning of a new data transmission. It is also allowed to send repeated start condition followed by slave address and an R/W bit.
- Slave address and R/W bit transfer
  - After START or Repeated START condition, the master sends a 7-bit slave address to identify the target device of the communication. After an address byte, a slave sensitive to the transmitted address has to acknowledge the reception.
- Register address of Ethernet PHY transfer
  - After having sent the slave address byte, the master may send 2-byte register address. The master keeps transmitting data after the slave returns acknowledge to the master.
- Data Byte
  - Write operation: The master transfers 2-byte data to the slave.
  - Read operation: The master receives 2-byte data from the slave.
- STOP (STO) signal generation
  - The master can terminate the communication by generating a STOP signal.

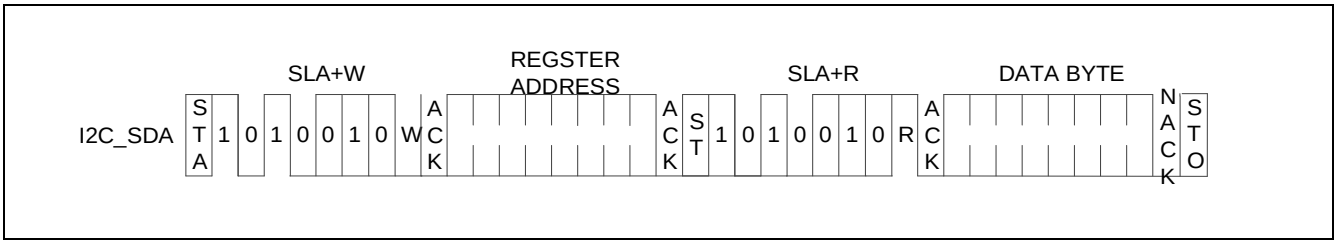

Figure 1-1 I²C Read Data Format

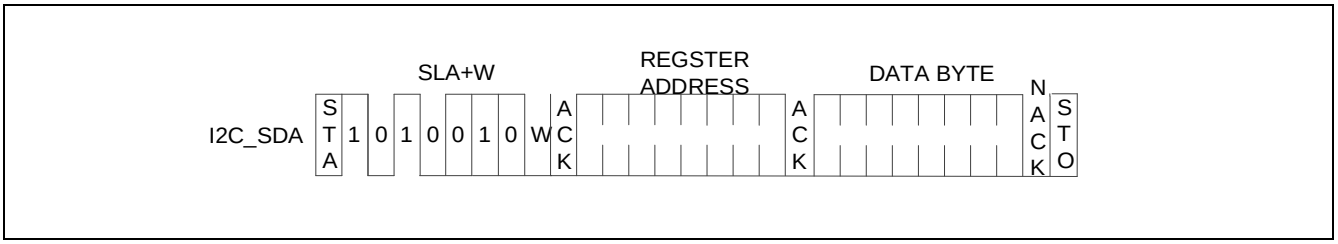

Figure 1-2 I²C Write Data Format

1.1.2 MDIO Data Format

The MDIO data format is shown in Figure 1-3 and MDIO data field description is shown in Table 1-1.

Preamble	ST	OP	PHY_ADR	REG_ADR	TA	DATA
32-bit	2-bit	2-bit	5-bit	5-bit	2-bit	16-bit

Figure 1-3 MDIO Data Format

Field	Width	Description
Preamble	32-bit	32 continuous bits of value 1.
ST	2-bit	Start of frame is 2'01.
OP	2-bit	Op code. 01 = Write operation 10 = Read operation
PHY_ADR	5-bit	PHY address.
REG_ADR	5-bit	Register address.
TA	2-bit	Turnaround time.
DATA	16-bit	Any 16-bit value. In Write operation, the MAC drives MDIO. In Read operation, PHY drives MDIO.

Table 1-1 MDIO Field Description

Since MDIO line is the bi-directional data line, please connect SPI_MOSI pin, SPI_MISO pin and MDIO pin together for realizing the feature. In Read operation, the program must configure the SPI_MOSI pin as GPIO input mode before read data. Figure 1-4 and Figure 1-5 show the program flow to emulate MDIO Write and Read waveform, respectively.

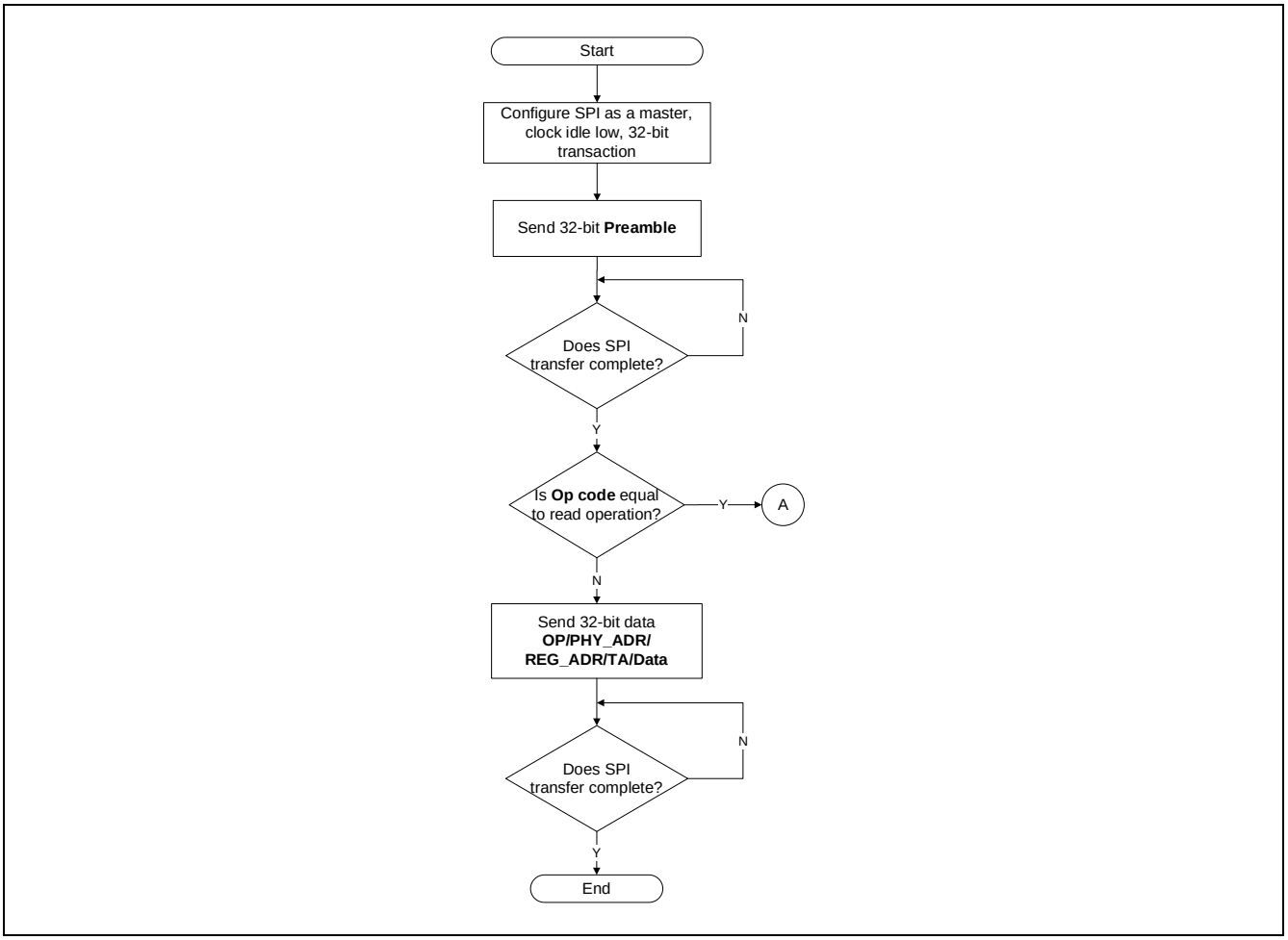

Figure 1-4 SPI Emulate MDIO Write Flow Chart

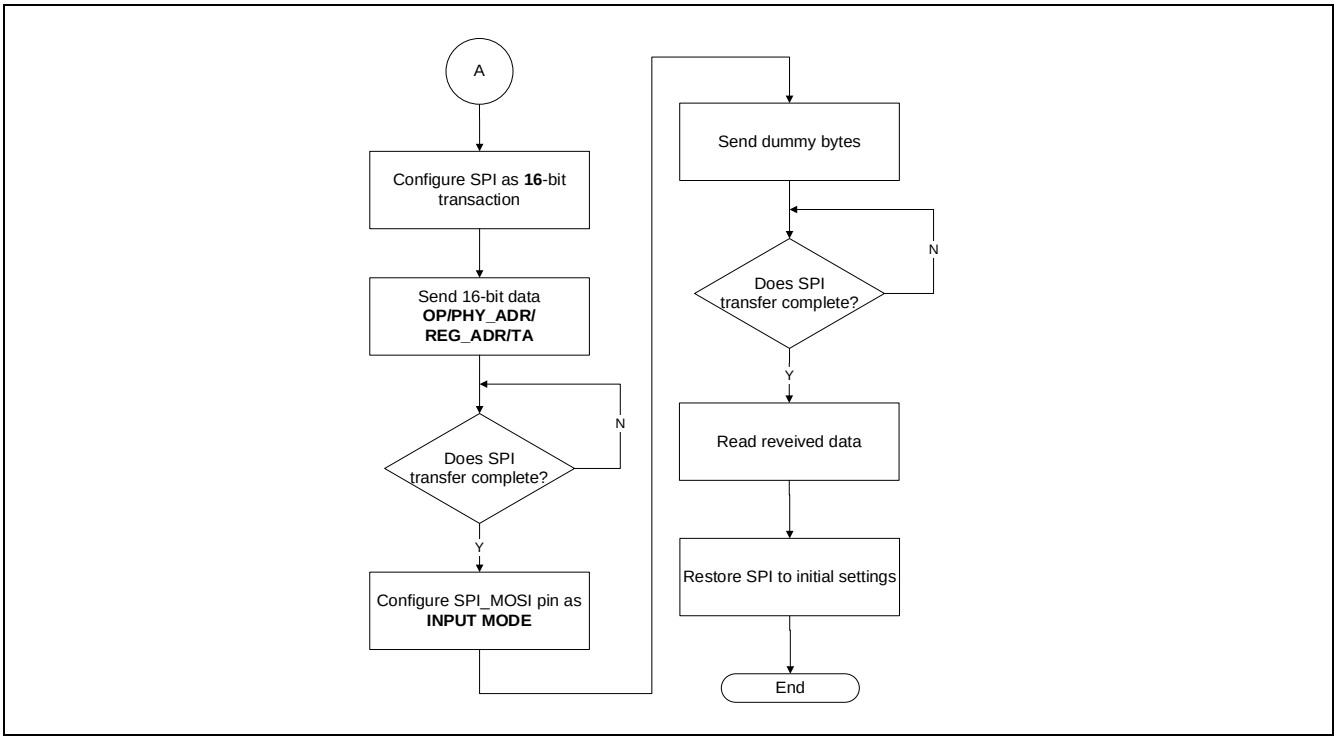

Figure 1-5 SPI Emulate MDIO Read Flow Chart

1.2 Demo Result

The logic analyzer display waveform of MDIO Write and Read operation are shown in Figure 1-6 and Figure 1-7, respectively, which shows the information of configuring register 0x04 as 0x0001 and performing a verify operation.

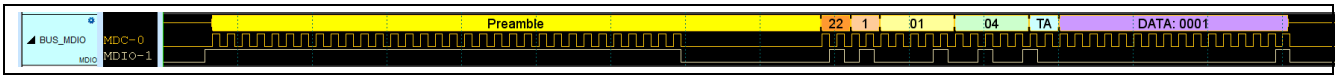

Figure 1-6 MDIO Write Results

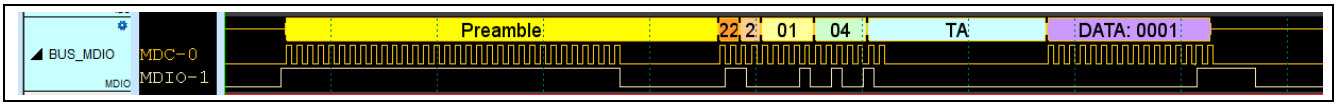

Figure 1-7 MDIO Read Results

2. Code Description

A function to initialize SPI function. The code is located in *spi_transfer.c*.

```
void SPI_Init(void)
{
    /* Set IP clock divider. SPI clock rate = 1MHz */
    SPI_Open(SPI0, SPI_MASTER, SPI_MODE_0, 32, SPI_CLK_FREQ);

    /* Enable the automatic hardware slave select function. Select the SS pin and
       configure as low-active. */
    SPI_EnableAutoSS(SPI0, SPI_SS, SPI_SS_ACTIVE_LOW);
}
```

A function to emulate MDIO waveform via SPI. The code is located in *spi_transfer.c*.

For the Write operation, the program:

- Sends the 32-bit preamble; and
- Sends the variable *MDIO_temp* consists of 2-bit of ST, 2-bit of opcode, 5-bit of PHY address, 5-bit of register address, 2-bit of TA and 16-bit of data.

For the Read operation, the program:

- Sends the 32-bit preamble.
- Configures SPI as 16-bit transaction and send the variable *MDIO_temp* consists of 2-bit of ST, 2-bit of Op code, 5-bit of PHY address, 5-bit of register address and 2-bit of TA.
- Configures SPI_MOSI pin as GPIO input mode for receiving data; and
- After receiving data is finished, re-configures SPI function as initial state.

```
MDIO_RES SPI_MDIO_RW(unsigned char rw, unsigned char phy_addr, unsigned char reg_addr,
unsigned int w_data, unsigned int *r_data)
{
    unsigned int MDIO_temp = 0;

    /* Checking parameters validation */
    if (rw > 0x02)
        return RW_ERROR;

    if (phy_addr > MDIO_PHY_MAX)
        return phy_ERROR;

    if (reg_addr > MDIO_REG_MAX)
        return reg_ERROR;

    if ((rw == MDIO_OP_W) && (w_data > W_DATA_MAX))
        return w_data_ERROR;

    if ((rw == MDIO_OP_R) && (w_data != W_DATA_MAX))
        return r_data_ERROR;
```

```
    /* Clear SPI FIFO */
    SPI_ClearTxFIFO(SPI0);
    SPI_ClearRxFIFO(SPI0);

    /* Prepare the MDIO package with 32-bit data, including 2-bit ST, 2-bit Op code,
    5-bit PHY address, 5-bit Register address, 2-bit TA and 16-bit Data */
    MDIO_temp = (MDIO_ST_CAUSE_22 << MDIO_ST_OFFSET) | (rw << MDIO_OP_OFFSET) |
                (phy_addr << MDIO_PHY_OFFSET) | (reg_addr << MDIO_REG_OFFSET) |
                (MDIO_TA << MDIO_TA_OFFSET) | w_data;

    /* READ operation with 16-bit data, including 2-bit ST, 2-bit Op code,
    5-bit PHY address and 5-bit Register address */
    if (rw == MDIO_OP_R)
        MDIO_temp = (MDIO_temp >> 16) & 0xFFFF;

    /* Send 32-bit preamble */
    SPI_WRITE_TX(SPI0, 0xffffffff);
    /* Check SPI0 busy status */
    while (SPI_IS_BUSY(SPI0));

    /* READ operation with 16-bit data, including 2-bit ST, 2-bit Op code,
    5-bit PHY address, 5-bit Register address and 2-bit TA*/
    /* WRITE operation with 32-bit data, including 2-bit ST, 2-bit Op code,
    5-bit PHY address, 5-bit Register address, 2-bit TA and 16-bit Data */
    if (rw == MDIO_OP_R)
        SPI_SET_DATA_WIDTH(SPI0, 16);

    SPI_WRITE_TX(SPI0, MDIO_temp);
    /* Check SPI0 busy status */
    while (SPI_IS_BUSY(SPI0));

    /* Read data from Ethernet PHY */
    if (rw == MDIO_OP_R)
    {
        /* Configure PA0(MOSI) as input mode for receiving data */
        SYS->GPA_MFPL = (SYS->GPA_MFPL & ~(SYS_GPA_MFPL_PA0MFP_Msk)) |
                        (SYS_GPA_MFPL_PA0MFP_GPIO);
        GPIO_SetMode(PA, BIT0, GPIO_MODE_INPUT);

        SPI_ClearRxFIFO(SPI0);
        /* Send dummy data for receiving data */
        SPI_WRITE_TX(SPI0, 0xFFFF);
        /* Check SPI0 busy status */
        while (SPI_IS_BUSY(SPI0));
        /* Read received data */
        r_data[reg_addr] = SPI_READ_RX(SPI0);

        /* Restore to initial settings */
        SYS->GPA_MFPL = (SYS->GPA_MFPL & ~(SYS_GPA_MFPL_PA0MFP_Msk)) |
                        (SYS_GPA_MFPL_PA0MFP_SPI0_MOSI);
        SPI_SET_DATA_WIDTH(SPI0, 32);
    }
    return PASS;
}
```

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M029G_M030G_M031G_Series_BSP_CMSIS_V3.03.000
- IDE version
  - Keil uVersion 5.38

3.2 Hardware Requirements

- Circuit components
  - NuMaker-M030GTD V1.1
- Pin Connect
  - Connect the I2C_SCL (PF.3) pin and I2C_SDA (PF.2) pin to the I²C master to receive command.
  - Connect the SPI_MOSI (PA.0) pin and SPI_MISO (PA.1) pin to the Ethernet PHY MDIO and connect the SPI_CLK (PA.2) to Ethernet PHY MDC to control Ethernet PHY.
  - Connect the 4.7kΩ pull-up resistor on I2C_SCL pin, I2C_SDA pin and MDIO pin.

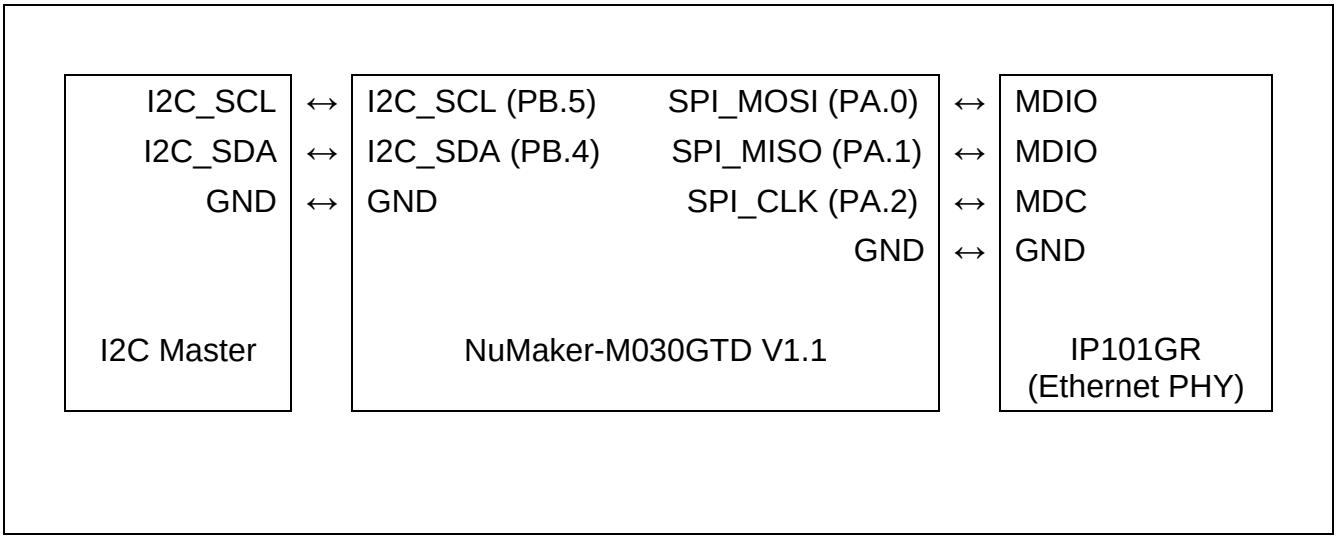

Figure 3-1 Pin Connect

4. Directory Information

The directory structure is shown below.

EC_M030G_I2C_Parse_CMD_SPI_Emulate_MDIO_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_Parse_CMD_SPI_Emulate_MDIO.uvprojx*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6. Revision History

Date	Revision	Description
2024.05.16	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.