M030G 模擬MDIO 波型與解析封包

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 I2C 接口接收操作命令并使用 SPI 接口仿真 MDIO 波形用以控制以太网络 PHY(IP101GR)。
BSP 版本	M029G_M030G_M031G_Series_BSP_CMSIS_V3.03.000
开发平台	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. 概述

本范例代码使用 M030G I²C 接口来接收操作指令，并使用 SPI 接口来仿真 MDIO 波形用以控制以太网络 PHY。

大多数 NuMicro® 家族微控制器不支持 MDIO 接口，本范例代码提供一种存取以太网络 PHY 的解决方案，可广泛应用于 NuMicro® 家族微控制器。

本范例代码需另接外部硬件模块，包含一 I²C 主机与以太网络 PHY。I²C 主机发送读取/写入指令来触发 SPI 接口仿真 MDIO 读取/写入波形用以存取 IC+ 的以太网络 PHY(IP101GR)。

1.1 原理

本范例代码可以分为两个部分，I²C 从机用来接收指令，MDIO 主机用来存取以太网络 PHY。以下将重点介绍 I²C 封包格式与说明如何透过 SPI 接口来仿真 MDIO 波形。

1.1.1 I²C 封包格式

图 1-1 与图 1-2 分别表示 I²C 读取/写入指令的封包格式。

- 开始 (STA) 或者重复开始(Repeated STA) 信号
  - I²C 主机透过发送 STA 信号或 Repeated STA 信号来发起传输。STA 信号表示新数据传输的开始。接着发送 I²C 从机地址和 R/W 位。
- 从机地址和读/写位传输
  - 在 STA 信号或 Repeated STA 信号后，I²C 主机发送 7 位从机地址来识别通信的目标装置。I²C 从机确认接收后会传送一 ACK 信号。
- 以太网络 PHY 的寄存器地址传输
  - 传送 I²C 从机地址后，I²C 主机传送 2 字节以太网络 PHY 的寄存器地址。当 I²C 从机向 I²C 主机传送 ACK 信号后，I²C 主机接着传送数据。
- 数据传输

- 写操作: I²C 主机传送 2 字节数据给 I²C 从机。
- 读操作: I²C 主机从 I²C 从机接收 2 字节数据。

- 停止信号(STO)
  - I²C 主机可以透过产生停止信号(STO) 信号来结束通信。

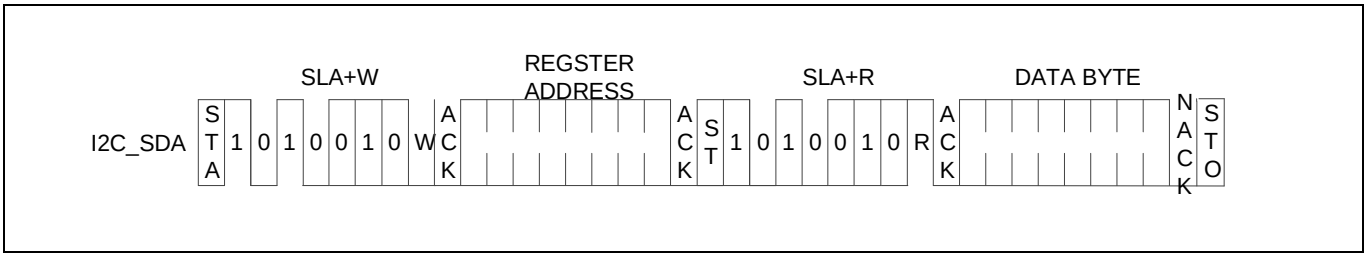

图 1-1 I²C 读取指令封包格式

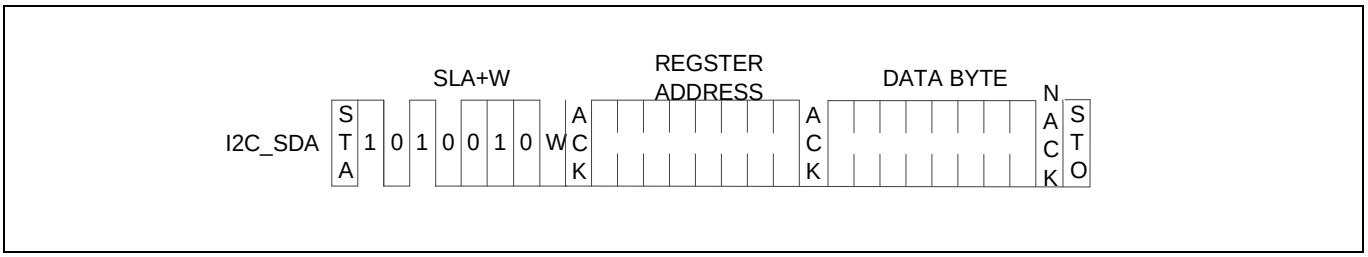

图 1-2 I²C 写入指令封包格式

1.1.2 MDIO 资料格式

MDIO 数据格式如图 1-3 所示，字段说明如表 1-1 所示。

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

图 1-3 MDIO 数据格式

字段	数据宽度	说明
Preamble	32-bit	32 个连续 2'1
ST	2-bit	帧开始标志 2'01
OP	2-bit	操作码

		01 = 写操作 10 = 读操作
PHY_ADR	5-bit	PHY 地址
REG_ADR	5-bit	寄存器地址
TA	2-bit	状态转换时间
DATA	16-bit	MDIO 资料

表 1-1 MDIO 数据字段介绍

由于 MDIO 是双向数据线，故需将 SPI_MOSI 脚位、SPI_MISO 脚位和 MDIO 脚位连接在一起以实现此功能。在读操作中，程序必须先将 SPI_MOSI 脚位配置为 GPIO 输入模式，才能读取数据。图 1-4 与图 1-5 分别描述 SPI 接口仿真 MDIO 写操作与读操作的程序流程。

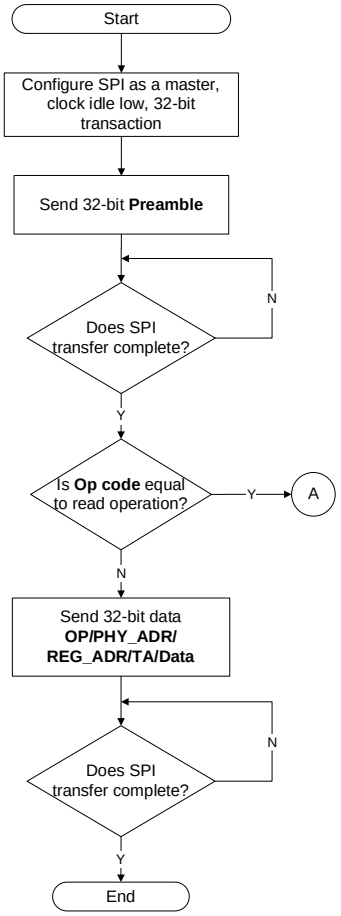

图 1-4 SPI 模拟 MDIO 写操作流程图

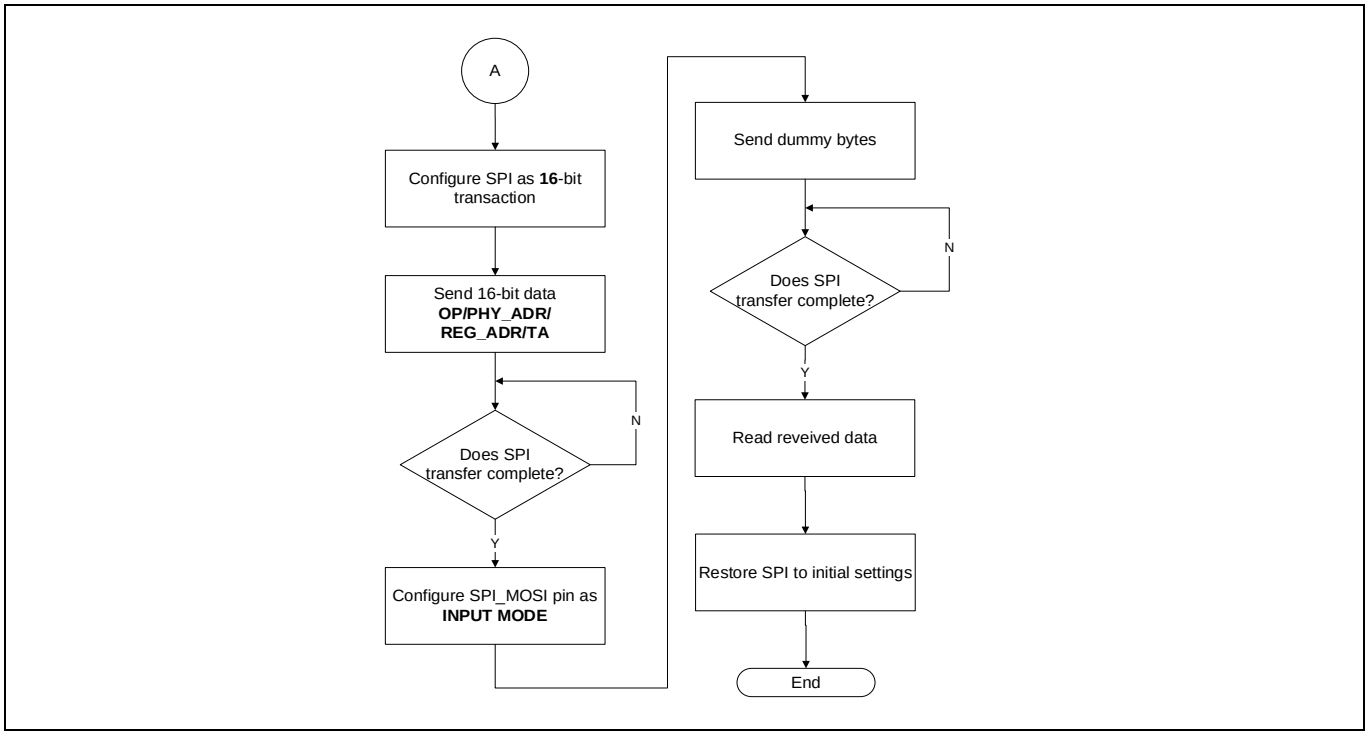

图 1-5 SPI 模拟 MDIO 读操作流程图

1.2 执行結果

本范例代码执行结果如图 1-6 与图 1-7 所示，分别对应 MDIO 写操作和读操作波型。演示将寄存器 0x04 配置为 0x0001 并进行验证。

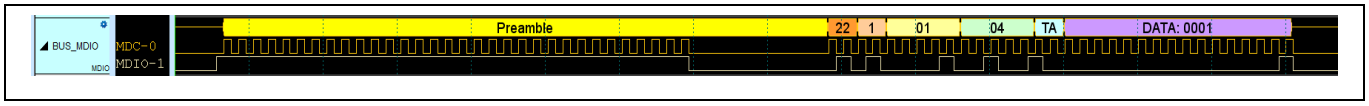

图 1-6 MDIO 写操作波型

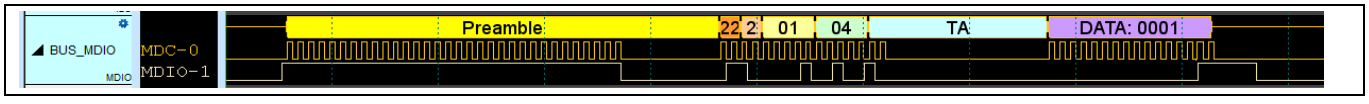

图 1-7 MDIO 读操作波型

2. 代码介绍

初始化 SPI 接口，代码位于 *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);
}
```

初始化系统后，透过 SPI 接口仿真 MDIO 波型。代码位于 spi_transfer.c。

写操作

- 传送 32 位 Preamble。
- 传送 MDIO_temp，包含 ST (2 位)、opcode(2 位)、PHY 地址(5 位)、寄存器地址(5 位)、TA (2 位) 与 Data(16 位)。

读操作

- 传送 32 位 Preamble。
- 设定 SPI 接口数据宽度为 16 位与传送 MDIO_temp，包含 ST (2 位)、opcode(2 位)、PHY 地址(5 位)、寄存器地址(5 位) 与 TA (2 位) 组成。
- 设定 SPI_MOSI 脚位为 GPIO input 模式来接收数据。
- 接收完毕后，需重新设定 SPI 接口为初始设定。

```
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. 软件与硬件需求

3.1 软件需求

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

3.2 硬件需求

- 电路组件
  - NuMaker-M030GTD V1.1
- 线路示意图
  - 将 I2C_SCL (PF.3) pin 脚与 I2C_SDA (PF.2) pin 脚分别连接到 I2C 主机，用以接收操作指令。
  - 将 SPI_MOSI (PA.0) pin 脚与 SPI_MISO (PA.1) pin 脚连接到 Ethernet PHY 的 MDIO pin 脚; SPI_CLK (PA.2) pin 脚连接到 Ethernet PHY 的 MDC pin 脚用存取 Ethernet PHY 寄存器。
  - 分别连接 4.7 k 欧姆的上拉电阻于 I2C_SCL (PF.3) pin 脚、I2C_SDA (PF.2) pin 脚与 MDIO pin 脚。

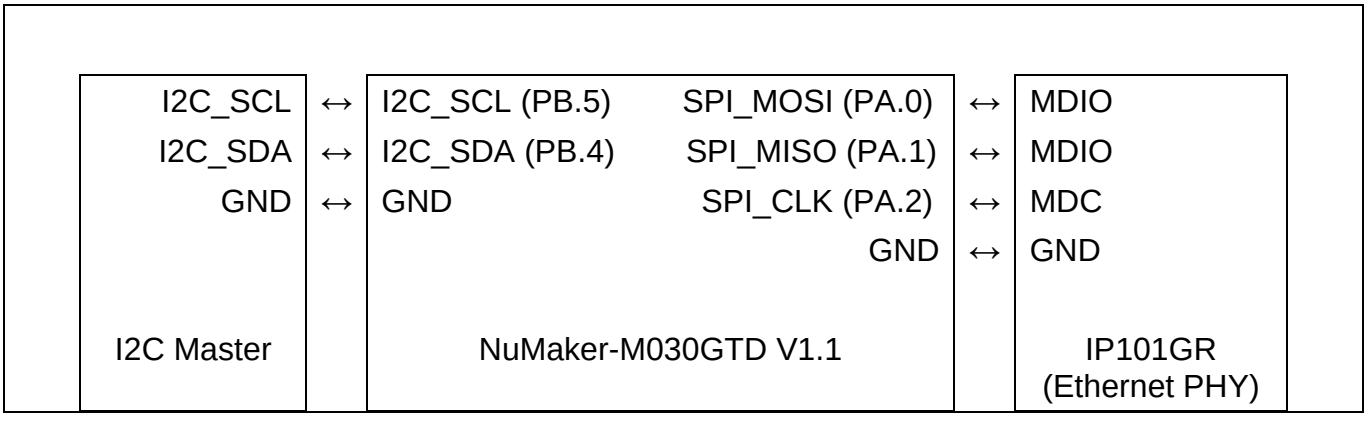

图 3-1 線路连接图

4. 目录信息

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

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_Parse_CMD_SPI_Emulate_MDIO.uvprojx。
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
2024.05.16	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.