MN316
OpenCPU 外围接口开发指导手册

NB-IoT 系列

版本：V1.1.0

日期：2021 年 2 月

中移物联网有限公司
iot.10086.cn

服务与支持

如果您有任何关于模组产品及产品手册的评论、疑问、想法，或者任何无法从本手册中找到答案的疑问，请通过以下方式联系我们。

中移物联网有限公司

OneMO 官网：onemo10086.com

邮箱：SmartModule@cmiot.chinamobile.com

客户服务热线：400-110-0866

微信公众号：CMOneMO

文档声明

注意

本手册描述的产品及其附件特性和功能，取决于当地网络设计或网络性能，同时也取决于用户预先安装的各种软件。由于当地网络运营商、ISP，或当地网络设置等原因，可能也会造成本手册中描述的全部或部分产品及其附件特性和功能未包含在您的购买或使用范围之内。

责任限制

除非合同另有约定，中移物联网有限公司对本文档内容不做任何明示或暗示的声明或保证，并且不对特定目的适销性及适用性或者任何间接的、特殊的或连带的损失承担任何责任。

在适用法律允许的范围内，在任何情况下，中移物联网有限公司均不对用户因使用本手册内容和本手册中描述的产品而引起的任何特殊的、间接的、附带的或后果性的损坏、利润损失、数据丢失、声誉和预期的节省而负责。

因使用本手册中所述的产品而引起的中移物联网有限公司对用户的最大赔偿（除在涉及人身伤害的情况中根据适用法律规定的损害赔偿外），不应超过用户为购买此产品而支付的金额。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。公司保留随时修改本手册中任何信息的权利，无需进行提前通知且不承担任何责任。

商标声明

中国移动 China Mobile 为中国移动注册商标。

本手册和本手册描述的产品中出现的其他商标、产品名称、服务名称和公司名称，均为其各自所有者的财产。

进出口法规

出口、转口或进口本手册中描述的产品（包括但不限于产品软件和技术数据），用户应遵守相关进出口法律和法规。

隐私保护

关于我们如何保护用户的个人信息等隐私情况，请查看相关隐私政策。

操作系统更新声明

操作系统仅支持官方升级；如用户自己刷非官方系统，导致安全风险和损失由用户负责。

固件包完整性风险声明

固件仅支持官方升级；如用户自己刷非官方固件，导致安全风险和损失由用户负责。

版权所有©中移物联网有限公司。保留一切权利。

关于文档

修订记录

版本	发布日期	作者	描述
V1.0.0	2020/12/24	任亚洲	初版
V1.1.0	2021/2/23	张雄威	增加串口及 GPIO 中断使用说明

目录

1 接口介绍

1.1 资源介绍

类型	最大组数	说明
UART	2	无流控功能
SPI	1	主模式
IIC	1	主模式
ADC	1	采样精度 10 位
GPIO	6	无
VBAT	1	无

1.2 引脚定义

引脚号	引脚名	描述
31、32	VBAT	VBAT
38	ADC	ADC
36	I2C_SCL	I2C 接口
37	I2C_SDA	
35	GPIO1	GPIO
7	GPIO2	GPIO
8	GPIO3	GPIO
39	UART1_TX	串口 1
40	UART1_RX	
16	GPIO5	GPIO
34	GPIO0	GPIO
11	GPIO4	GPIO
9	UART0_TXD	串口 0
10	UART0_RXD	
6	SPI_CLK	SPI 接口
5	SPI_MOSI	
4	SPI_MISO	
3	SPI_SS	

2 接口函数

2.1 UART

串口中断服务程序及内部调用接口，必须使用编译器参数加载在内存中运行，实现方法为：在函数声明阶段加入“attribute__((section(".ramtext")))”修饰参数，例：“void user_uart0_callback(char *rcv_data, int len) __attribute__((section(".ramtext")));”。

SDK 提供的可在中断服务程序中调用的接口将在第 4 章节介绍。

2.1.1 初始化

初始化
函数原型
int cm_uart_init(UART_BUS dev, UART_LINE_CONFIGURATION *config, UART_RECV_CALLBACK callback)
参数描述
<dev>
串口设备号，参考 UART_BUS。
<config>
串口配置，参考 UART_LINE_CONFIGURATION。
<callback>
串口接收中断回调函数，函数原型 typedef void(*UART_RECV_CALLBACK)(char *rcv_data, int len)。
备注
– 校验位仅 UART_BUS_1 有效； – 串口波特率支持 2400、4800、9600、19200、38400、57600、115200，单位 bps。
示例

```
#define UART0_RX_BUF_SIZE        1024 * 2                    //串口接收中断回调函数。
static uint8 uart0_rx_buf[UART0_RX_BUF_SIZE] = {0};
static uint32 uart0_rx_size = 0;
void user_uart0_callback(char *rcv_data, int len)
{
    if (uart0_rx_size < UART0_RX_BUF_SIZE)
    {
        memcpy(&uart0_rx_buf[uart0_rx_size], rcv_data, len);
        uart0_rx_size += len;
    }
}
UART_LINE_CONFIGURATION uart_config={9600,                  //串口初始化配置。
UART_DATA_BITS_8, UART_PARITY_NONE,
UART_STOP_BITS_1};
cm_uart_init(UART_BUS_0, &uart_config, user_uart0_callback);
```

2.1.2 去初始化

去初始化
函数原型
void cm_uart_deinit(UART_BUS dev)
参数描述
<dev>
串口设备号，参考 UART_BUS。
备注
仅对已初始化的串口有效。
示例
cm_uart_deinit(UART_BUS_0);

2.1.3 发送数据

发送数据
函数原型
int cm_uart_write(UART_BUS dev, char *data, unsigned short len)
参数描述
<dev>
串口设备号，参考 UART_BUS。
<data>
待发送数据首地址。
<len>
待发送数据长度。
备注
发送数据前需先初始化串口。
示例
char test_str[] ="I have a dream";　　//发送数据。 cm_uart_write(UART_BUS_0, test_str, strlen(test_str));

2.2 SPI

2.2.1 初始化

初始化
函数原型
int cm_spi_init(SPI_BUS dev, SPI_CONFIGURATION *config)
参数描述
<dev>
SPI 设备，参考 SPI_BUS。
<config>
SPI 配置，参考 SPI_CONFIGURATION。
备注
SPI 仅支持主模式，频率为 39.168MHz，工作频率为该频率分频得到。
示例

SPI_CONFIGURATION config = {SPI_CLK_MODE0, SPI_CLK_DIV16}; //SPI 初始化配置。
cm_spi_init(SPI_BUS_0, &config);

2.2.2 去初始化

去初始化
函数原型
void cm_spi_deinit(SPI_BUS dev)
参数描述
<dev>
SPI 设备，参考 SPI_BUS。
备注
仅对已初始化的 SPI 有效。
示例
cm_spi_deinit(SPI_BUS_0);

2.2.3 写数据

写数据
函数原型
int cm_spi_write(SPI_BUS dev, char *data, unsigned short len)
参数描述
<dev>
SPI 设备，参考 SPI_BUS。
<data>
待写数据。
<len>
待写数据长度。
备注
写数据前需先初始化 SPI。
示例

char test[] = {0x06}; //写数据。
cm_spi_write(SPI_BUS_0, test, sizeof(test));

2.2.4 读数据

读数据
函数原型
int cm_spi_read(SPI_BUS dev, char *w_data, unsigned short w_len, char *r_data, unsigned short r_len)
参数描述
<dev>
SPI 设备，参考 SPI_BUS。
<w_data>
写命令。
<w_len>
写命令长度。
<r_data>
读取数据存放地址，需先申请内存。
<r_len>
读取数据长度。
备注
读数据前需先初始化 SPI。
示例

char w_cmd[] = {0x9f}; //读数据。
char r_data [3]={0};
cm_spi_read(SPI_BUS_0, w_cmd, sizeof(w_cmd), r_data, sizeof(r_data));

2.3 I2C

2.3.1 初始化

初始化	
函数原型	
int cm_i2c_init(I2C_BUS dev, I2C_CONFIGURATION *config)	
参数描述	
<dev>	
I2C 设备，参考 I2C_BUS。	
<config>	
I2C 配置，参考 I2C_CONFIGURATION。	
备注	
仅支持主机模式。	
示例	
I2C_CONFIGURATION config = {I2C_MODE_MASTER, I2C_SPEED_100K}; cm_i2c_init(I2C_BUS_0, &config);	//I2C 初始化配置。

2.3.2 去初始化

去初始化
函数原型
void cm_i2c_deinit(I2C_BUS dev)
参数描述
<dev>
I2C 设备，参考 I2C_BUS。
备注
仅对已初始化的 I2C 有效。
示例
cm_i2c_deinit(I2C_BUS_0);

2.3.3 写数据

写数据
函数原型
int cm_i2c_write(I2C_BUS dev, unsigned char addr, char *data, unsigned char len)
参数描述
<dev>
I2C 设备，参考 I2C_BUS。
<addr>
I2C 设备地址。
<data>
待写数据。
<len>
待写数据长度。
备注
写数据前需先初始化 I2C，写数据长度不能超过 8 字节。
示例

unsigned char slave_addr=0x50;
char wdata[4]={0x00,0x10,0x55,0xAA};
cm_i2c_write(I2C_BUS_0, slave_addr, wdata, sizeof(wdata));

2.3.4 读数据

读数据
函数原型
int cm_i2c_read(I2C_BUS dev, unsigned char addr, char *data, unsigned char len)
参数描述
<dev>
I2C 设备，参考 I2C_BUS。
<addr>
I2C 设备地址。
<data>
读取数据存放地址，需先申请内存。
<len>
读取数据长度。
备注
读数据前需先初始化 I2C，读取数据长度不能超过 8 字节。
示例

unsigned char slave_addr=0x50;
char rdata[2]={0};
cm_i2c_read(I2C_BUS_0, slave_addr, rdata, sizeof(rdata));

2.4 GPIO

GPIO 中断服务程序及内部调用接口，必须使用编译器参数加载在内存中运行，实现方法为：在函数声明阶段加入“attribute__((section(".ramtext")))”编译器参数，例：“void gpio_irq_handle(GPIO_NUM pin) __attribute__((section(".ramtext")));”。

SDK 提供的可在中断服务程序中调用的接口将在第 4 章节介绍。

2.4.1 初始化

初始化
函数原型
int cm_gpio_init(GPIO_NUM pin, GPIO_MODE_E mode, GPIO_INT_CONFIG *config)
参数描述
<pin>
待配置的引脚。
<mode>
待配置的模式值。
<config>
中断配置。
备注
– GPIO0、GPIO5 默认状态为输入下拉，GPIO1、GPIO2、GPIO3、GPIO4 默认状态为输入上拉； – GPIO0、GPIO5 仅支持下拉，GPIO1、GPIO2、GPIO3、GPIO4 仅支持上拉； – 若不使用中断时，config 传入 NULL； – 同一时刻只能有一个外部 GPIO 中断触发源。
示例

void gpio_irq_handle(GPIO_NUM pin) //中断处理函数。
{
cm_printf("gpio%d get irq handle\n", pin);
}
GPIO_INT_CONFIG config = {GPIO_INTERRUPT_LEVEL_HIGH, gpio_irq_handle}; //GPIO 初始化配置。
cm_gpio_init(GPIO_NUM_0, GPIO_MODE_IN_PD, &config);

2.4.2 去初始化

去初始化
函数原型
void cm_gpio_deinit(GPIO_NUM pin)
参数描述
<pin>
待去初始化的引脚。
备注
去初始化后引脚恢复默认状态。
示例
cm_gpio_deinit (GPIO_NUM_0);

2.4.3 读取输入电平

读取输入电平	
函数原型	
int cm_gpio_get_input(GPIO_NUM pin, unsigned char *value)	
参数描述	
<pin>	
待操作的引脚。	
<value> 输入电平	
0	低电平
1	高电平
示例	
unsigned char level = 0; cm_gpio_get_input(GPIO_NUM_0, &level);	

2.4.4 输出电平

输出电平	
函数原型	
int cm_gpio_set_output(GPIO_NUM pin, unsigned char value)	
参数描述	
<pin>	
待操作的引脚。	
<value> 输出电平	
0	低电平
1	高电平
示例	
cm_gpio_set_output(GPIO_NUM_0, 0);	

2.5 ADC

ADC
函数原型
unsigned int cm_adc_read(void)
备注
读取 ADC 电压，支持的范围 0~1000mV。
示例
unsigned int vol = 0； vol = cm_adc_read();

2.6 VBAT

VBAT
函数原型
unsigned int cm_vbat_read(void)
备注
读取 VBAT 电压，单位 mv。模组支持电压范围 3.1V ~ 4.2V。
示例
unsigned int vol = 0； vol = cm_vbat_read();

3 注意事项

3.1 使用限制

引脚类型	使用限制
串口	– 校验位仅串口 1 有效； – 串口波特率支持 2400、4800、9600、19200、38400、57600、115200，单位 bps； – 无流控功能。
SPI	SPI 仅支持主模式，频率为 39.168MHz，工作频率为该频率分频得到。
I2C	– 仅支持主机模式； – 写数据长度不能超过 8 字节。
GPIO	– GPIO0、GPIO5 默认状态为输入下拉，GPIO1、GPIO2、GPIO3、GPIO4 默认状态为输入上拉； – GPIO0、GPIO5 仅支持下拉，GPIO1、GPIO2、GPIO3、GPIO4 仅支持上拉； – 同一时刻只能有一个外部 GPIO 中断触发源。
ADC	仅支持的范围 0~1000mV。

3.2 使用注意

使用外设接口前，必须使用 cm_set_standby(0)接口关闭模组 Standby 功能，使用完成后可调用 cm_set_standby(1)开启 Standby 功能。

4 附录

4.1 参考文档

序号	文档名称	备注
[1]	MN316_OpenCPU MANUAL	API 接口手册
[2]	MN316_OpenCPU 资源综述	资源介绍

4.2 缩略语

缩写	英文全称
RRC	Radio Resource Control
EPS	Evolved Packet System

4.3 ramtext 接口

函数名	功能
osTimerStart	启动定时器
osTimerStop	停止定时器
osTimerIsRunning	获取定时器状态
osMessageQueuePut	写队列
osMessageQueueGet	读队列
osSemaphoreAcquire	获取信号量
osSemaphoreRelease	释放信号量
osMutexRelease	释放锁
cm_gpio_get_input	读取 GPIO 输入
cm_gpio_set_output	设置 GPIO 输出
cm_assert	断言
std_c	C 语言标准库