Modbus

HRTOS Modbus RTU 模块使用说明

1.1 模块简介

HRTOS Modbus RTU 模块是面向 8051 单片机的 Modbus 从站实现，集成 UART 接收、帧间隔检测、CRC16 校验、功能码处理、寄存器管理和响应发送，并通过 HRTOS 任务与事件机制组织协议处理流程。

当前源码实现了以下五种功能码：

功能码	名称	功能
`0x01`	Read Coils	读取线圈
`0x03`	Read Holding Registers	读取保持寄存器
`0x05`	Write Single Coil	写单个线圈
`0x06`	Write Single Register	写单个保持寄存器
`0x10`	Write Multiple Registers	写多个保持寄存器

默认从站地址为 `1`，保持寄存器数量为 `32`，线圈数量为 `8`。

当前版本是基于现有源码实现的轻量级从站模块。部分协议边界检查和缓冲区容量仍需完善，正式用于实际设备前应完成相应修正与测试。

1.2 模块架构

1.2.1 软件分层

```
应用层
   │
   ├── modbus_reg_read() / modbus_reg_write()
   ├── modbus_coil_read() / modbus_coil_write()
   │
Modbus 从站协议层
   │
   ├── modbus_slave_process()
   ├── 功能码处理与响应生成
   │
帧处理与 CRC 层
   │
```

```
  ├── modbus_frame_receive()
  ├── modbus_frame_timeout()
  ├── modbus_frame_parse()
  └── modbus_crc16()
  │
UART 端口层
  │
  ├── modbus_port_send()
  └── bsp_uart_send_byte()
  │
8051 UART 硬件
```

1.2.2 文件职责

文件	主要职责
`modbus.h`	模块公共接口、共享变量和事件配置
`modbus_config.h`	从站地址、缓冲区、寄存器数量及功能码配置
`modbus.c`	模块初始化、帧解析、协议处理和任务入口
`modbus_frame.h/.c`	接收缓存、帧数据结构、超时检测与帧解析
`modbus_slave.h/.c`	从站地址判断、功能码分发和响应生成
`modbus_crc.h/.c`	Modbus CRC16 计算
`modbus_port.h/.c`	Modbus 与 UART 驱动之间的接口
`modbus_register.h/.c`	保持寄存器和线圈的存储、读写
`bsp_uart.h` 及 UART 驱动	UART 初始化和单字节发送

1.3 配置说明

配置项位于 `modbus_config.h`。

配置宏	默认值	说明
`MODBUS_SLAVE_ADDR`	`1`	从站地址
`MODBUS_RX_BUF_SIZE`	`68`	接收缓存容量，单位为字节
`MODBUS_TX_BUF_SIZE`	`32`	部分响应函数使用的发送缓存大小
`MODBUS_REG_MAX`	`32`	保持寄存器数量
`MODBUS_COIL_MAX`	`8`	线圈数量
`MODBUS_FUNC_READ_HOLDING_REG`	`1`	声明 0x03 功能码配置
`MODBUS_FUNC_WRITE_SINGLE_REG`	`1`	声明 0x06 功能码配置
`MODBUS_FUNC_WRITE_MULTI_REG`	`1`	声明 0x10 功能码配置

注意： 当前 `modbus_slave_process()` 直接通过 `switch` 分发功能码，以上三个 `MODBUS_FUNC_*` 宏尚未参与功能启用或禁用判断。修改宏值并不会自动关闭对应功能。

线圈读写使用 `MODBUS_COIL_MAX` 控制地址范围；保持寄存器读写使用 `MODBUS_REG_MAX` 控制地址范围。

1.4 HRTOS 任务集成

1.4.1 任务职责

模块通过三个任务入口组织工作：

任务入口	任务 ID	优先级	堆栈参数	职责
`uart_isr`	0	9	0	处理 UART 中断上下文中的接收和发送标志
`modbus_task`	1	3	6	初始化模块、等待事件并处理协议帧
`modbus_timer_task`	2	2	6	周期检测接收帧的静默时间

以上参数来自当前 `hrtos_main()` 中的任务创建调用。堆栈参数的实际单位和使用约束以 HRTOS 的 `os_task_create()` 接口定义为准。

1.4.2 初始化与任务创建

当前工程的入口代码如下：

```
void hrtos_main(void)
{
    os_slab_overflow_register(modbus_overflow);

    bsp_uart_init();
    os_event_init(MODBUS_EVENT_RX);

    os_task_create(
        modbus_timer_task,
        2,
        2,
        6
    );

    os_task_create(
        modbus_task,
        1,
        3,
```

```
        6
    );

    os_task_create(
        uart_isr,
        0,
        9,
        0
    );
}
```

`MODBUS_EVENT_RX` 定义为事件 ID `1`。使用前需要初始化该事件。

`modbus_task()` 内部调用 `modbus_init()`，而 `modbus_init()` 又会调用 `modbus_port_init()`，后者再次调用 `bsp_uart_init()`。因此，当前工程存在 UART 初始化重复执行的情况。可以统一初始化职责，避免不必要的重复配置。

1.4.3 数据接收与处理流程

```
UART 接收中断
      │
      ▼
uart_isr()
      │
      ▼
modbus_frame_receive(ch)
      │
      ▼
字节写入接收缓存
      │
      ▼
modbus_timer_task()
周期调用 modbus_frame_timeout()
      │
      ▼
达到当前超时阈值
      │
      ▼
os_event_write(MODBUS_EVENT_RX)
      │
      ▼
modbus_task() 等待事件并继续处理
      │
      ▼
modbus_process()
      │
      ▼
modbus_frame_parse()
      │
      ▼
modbus_slave_process()
      │
      ├── 读取或写入寄存器、线圈
      │
      ▼
生成响应、添加 CRC16
      │
```

```
      ▼
modbus_port_send()
      │
      ▼
UART 发送响应
```

当前超时检测通过任务周期累计计数实现，并非直接使用硬件定时器测量精确的 3.5 字符时间。

1.5 API 参考

1.5.1 模块初始化与处理

1.5.1.1 modbus_init()

```
void modbus_init(void);
```

初始化 Modbus 端口和帧处理状态，并设置模块初始化标志。

当前实现依次调用：

- `modbus_port_init()`
- `modbus_frame_init()`

该函数由 `modbus_task()` 在进入主循环前调用。

1.5.1.2 modbus_process()

```
void modbus_process(void);
```

处理一帧已接收的数据。模块未初始化时直接返回；初始化完成后尝试解析接收缓存，解析成功则调用 `modbus_slave_process()`，最后重置帧状态。

该函数不负责持续接收 UART 字节，也不负责周期计时。

1.5.2 帧处理接口

1.5.2.1 modbus_frame_init()

```
void modbus_frame_init(void);
```

重置接收长度、当前帧的地址、功能码、数据长度、超时计数及帧就绪标志。

1.5.2.2 modbus_frame_receive()

```
unsigned char modbus_frame_receive(unsigned char ch);
```

将一个字节写入接收缓存，并重置帧超时计数。

参数：

- `ch`：收到的 UART 字节。

当前实现的实际返回值始终为 `0`。 虽然注释写有“收到完整帧时返回 1”，但代码并没有实现该返回行为。帧结束由 `modbus_frame_timeout()` 检测。

当接收缓存已满时，函数不会继续写入新字节，但仍会重置超时计数。

1.5.2.3 modbus_frame_timeout()

```
void modbus_frame_timeout(void);
```

当接收缓存非空时，调用一次便递增 `modbus_timeout_cnt`。达到 `MODBUS_FRAME_TIMEOUT` 后，设置帧就绪标志并通过 `os_event_write(MODBUS_EVENT_RX)` 通知处理任务。

默认阈值为 `4`。在当前任务每次延时约 1 ms 的设计下，它是一个基于任务调度的粗粒度超时检测，而非严格精确的 RTU 帧间隔测量。

1.5.2.4 modbus_frame_parse()

```
unsigned char modbus_frame_parse(void);
```

当前实现检查最小接收长度、计算并比较 CRC16，随后提取地址、功能码和数据区。

返回值：

- `1`：基础帧解析成功。
- `0`：接收长度不足、CRC 不匹配或数据区长度超出帧结构容量等情况。

注意： 当前函数没有完成各功能码所需的数据长度检查。基础解析成功并不代表请求符合相应功能码的完整协议格式。

1.5.2.5 modbus_frame_get()

```
modbus_frame_t *modbus_frame_get(void);
```

返回指向当前全局帧结构 `modbus_frame` 的指针。

1.5.3 从站处理接口

1.5.3.1 modbus_slave_process()

```
void modbus_slave_process(modbus_frame_t *frame);
```

根据帧中的从站地址和功能码分发处理请求。

当前代码接受本机地址 `MODBUS_SLAVE_ADDR` 和地址 `0`。地址 `0` 在 Modbus RTU 中用于广播，但当前实现没有对广播请求禁止响应，因此尚未完整遵循广播请求不应获得从站响应的规则。

1.5.4 寄存器与线圈接口

1.5.4.1 modbus_reg_read()

```
unsigned int modbus_reg_read(unsigned char addr);
```

读取指定保持寄存器。默认有效地址为 `0-31`。超出范围时返回 `0`。

1.5.4.2 modbus_reg_write()

```
void modbus_reg_write(
    unsigned char addr,
    unsigned int value
);
```

向有效地址写入 16 位无符号数。超出范围时不执行写入。

1.5.4.3 modbus_coil_read()

```
unsigned char modbus_coil_read(unsigned int addr);
```

读取线圈状态。有效地址为 `0-7`，返回 `0` 表示 OFF，返回非零值表示 ON；地址超出范围时返回 `0`。

1.5.4.4 modbus_coil_write()

```
void modbus_coil_write(
    unsigned int addr,
    unsigned char value
);
```

写入线圈状态。`value` 非零时置位，为零时清零。地址超出范围时不执行写入。

1.5.5 CRC 接口

1.5.5.1 modbus_crc16()

```
unsigned int modbus_crc16(
    unsigned char *data1,
    unsigned int len
);
```

计算 Modbus CRC16 校验值。当前算法使用初值 `0xFFFF` 和反向多项式 `0xA001`。

发送时先放入 CRC 低字节，再放入高字节；接收时也按照低字节在前的顺序组合校验值。

1.5.6 端口接口

1.5.6.1 modbus_port_init()

```
void modbus_port_init(void);
```

调用 `bsp_uart_init()` 初始化 UART。

1.5.6.2 modbus_port_send()

```
void modbus_port_send(
    unsigned char *buf,
    unsigned int len
);
```

循环调用 `bsp_uart_send_byte()` 发送指定缓冲区中的字节。当前发送实现为逐字节阻塞发送。

1.5.6.3 已声明但当前源码未见实现的接口

以下函数在 `modbus_port.h` 中声明，但在本次提供的源码中没有看到函数定义：

```
unsigned char modbus_port_recv(unsigned char *ch);

unsigned char modbus_port_rx_byte(unsigned char ch);
```

如果工程其他文件没有提供实现，不应直接调用这两个接口，否则可能出现链接错误。

1.6 支持的功能码

1.6.1 0x01：读取线圈

读取指定起始地址和数量的线圈状态。

当前实现限制：

- 读取数量必须大于 `0`。
- 读取数量不得超过 `MODBUS_COIL_MAX`。
- 起始地址加读取数量不得超出线圈范围。
- 返回数据按位打包，低位优先映射较低地址的线圈。

1.6.2 0x03：读取保持寄存器

读取连续保持寄存器，并按每个寄存器高字节在前、低字节在后的顺序生成响应。

当前实现限制：

- 数量必须大于 `0`。
- 数量不得超过 `MODBUS_REG_MAX`。
- 起始地址与数量的范围必须合法。
- 单次读取数量还必须受到发送缓存容量限制，确保响应数据、协议头和 CRC 均能放入发送缓存。

默认发送缓存为 `32` 字节。完整响应长度为 `5 + 2 × 数量` 字节，因此在该缓存配置下，单次最多读取 `13` 个保持寄存器。若需要支持更大的读取数量，应同步调整发送缓存容量，并检查目标 MCU 的内存资源。

1.6.3 0x05：写单个线圈

请求中的写入值：

- `0xFF00`：置为 ON。
- `0x0000`：置为 OFF。

其他写入值会被当前实现直接拒绝，不发送正常响应。

成功时返回与请求相对应的地址、功能码和数据。

1.6.4 0x06：写单个保持寄存器

写入一个 16 位保持寄存器。地址有效时更新寄存器，并返回相应的请求字段。

1.6.5 0x10：写多个保持寄存器

请求中包含起始地址、寄存器数量、数据字节数及寄存器数据。当前实现检查数量、数据字节数和寄存器地址范围，然后依次写入寄存器。

成功时返回起始地址和写入数量。

当前实现仍需要增加对请求实际数据长度的检查，避免不完整请求被当作有效数据访问。

1.7 寄存器与线圈存储

1.7.1 保持寄存器

当前寄存器存储使用静态数组：

```
static unsigned int holding_reg[MODBUS_REG_MAX];
```

默认包含 32 个保持寄存器，地址范围为 `0-31`。数组由 C 运行环境进行静态初始化，未显式赋值的元素初始为零。

应用可通过 `modbus_reg_read()` 和 `modbus_reg_write()` 访问寄存器，也可以根据设备需求修改底层读写函数，将寄存器映射到实际传感器、控制参数或其他应用数据。

1.7.2 线圈

当前线圈存储使用位图：

```
static unsigned char coil_table[MODBUS_COIL_MAX / 8 + 1];
```

默认线圈数量为 8，每个线圈对应一个 bit，地址范围为 `0-7`。读写接口负责检查地址并访问相应位。

如果将 `MODBUS_COIL_MAX` 配置为其他数量，应同步检查位图容量计算与实际使用范围。

1.8 UART 配置与移植

当前 BSP 使用经典 8051 UART 和定时器 1：

```
SCON = 0x50;
TMOD &= 0x0F;
TMOD |= 0x20;

TL1 = 0xFD;
TH1 = 0xFD;

ET1 = 0;
TR1 = 1;

EA = 1;
ES = 1;
```

其中，串口配置为 8 位 UART、可变波特率；定时器 1 工作于模式 2，串口中断使能。

在常见的 11.0592 MHz 晶振、标准 8051 串口时钟配置下，该重装值通常用于生成 9600 bps 波特率。移植到其他 MCU 或时钟配置时，应重新核算波特率，不能只复制重装值。

`bsp_uart_send_byte()` 使用 `TI` 标志等待发送完成，因此当前发送路径是阻塞式的。应避免在不适合阻塞的中断或高实时性路径中直接调用长串口发送流程。

移植时主要需要适配：

- UART 初始化与波特率配置。
- UART 接收中断中的数据读取和标志清除。
- UART 单字节发送函数。
- HRTOS 中断入口和中断退出机制。

1.9 常见问题

1.9.1 收不到响应

检查：

- 主站与从站的波特率是否一致。
- 从站地址是否与 `MODBUS_SLAVE_ADDR` 一致。
- UART 的 TX/RX 接线是否正确。
- 串口接收中断是否正常触发。
- 帧超时任务是否持续运行。
- 请求 CRC 是否正确。

1.9.2 CRC 校验失败

检查主站发送数据、线路质量、波特率和 CRC 计算方式。Modbus RTU 的 CRC 在线路上按低字节在前、高字节在后的顺序传输。

1.9.3 寄存器或线圈没有变化

检查请求中的地址、数量、功能码和写入值是否符合当前实现的限制，并确认应用没有在其他位置覆盖相关数据。

1.9.4 大数量读取或写入异常

确认 `0x03` 单次读取数量没有超过发送缓存所允许的上限。当前默认配置下，单次最多读取 13 个保持寄存器。同时，应完善功能码数据长度检查，避免访问尚未接收的数据。

1.9.5 连续请求偶发丢失

当前处理流程在 `modbus_process()` 结束后调用 `modbus_frame_init()`，而任务随后删除接收事件。需要结合实际请求间隔检查接收缓存、事件清除和帧重置之间的时序，确认新一帧数据不会被覆盖或丢失。

1.10 当前版本的实现边界

根据本次提供的源码，使用前应特别留意以下事项：

- `modbus_frame_receive()` 的返回值始终为 `0`，与注释描述不一致。
- `modbus_frame_parse()` 没有验证各功能码所需的完整数据长度。
- `0x03` 发送缓存默认配置为 32 字节，单次读取数量应限制为最多 13 个寄存器；若调整缓存容量，应同步检查内存资源。
- `0x10` 处理需要验证请求数据长度，避免读取未接收的数据。
- 地址 `0` 被接受，但当前实现仍可能对广播请求发送响应。
- `MODBUS_FUNC_*` 宏目前没有控制实际功能码分发。
- `modbus_port_recv()` 和 `modbus_port_rx_byte()` 在提供的源码中只有声明，没有对应实现。
- 当前超时检测依赖约 1 ms 周期任务和计数阈值 `4`，不能直接视为精确的 3.5 字符时间检测。

建议在正式发布版本中优先完善这些边界，再进行异常帧、最大长度帧、连续请求、广播请求和长时间运行测试。

适用范围： HRTOS 8051 平台上的 Modbus RTU 从站模块。

默认配置： 从站地址 1、接收缓存 68 字节、发送缓存 32 字节、保持寄存器 32 个、线圈 8 个。