FreeModbus V1.6 instructions

1. **Brief introduction**

FreeModbus is an open source Modbus protocol stack, but only the slave is open source, and the host source code is **charged**. At the same time, no better open source Modbus host protocol stack was found on the Internet, so this FreeModbus protocol stack supporting host mode was developed. The version number of this version of FreeModbus has been changed to V1.6, and the features are as follows:

- The newly added host source code is consistent with the style and interface of the original slave;

- Support the host and slave to run in the same protocol stack;

- Support real-time operating system and bare metal transplantation;

- Provide a variety of request modes for applications, users can choose blocking or non-blocking mode, custom timeout time, etc., to facilitate flexible calls at the application layer;

- Support all commonly used Modbus methods.

1.1 File structure

|  |  |
| --- | --- |
| Source File | Description |
| FreeModbus\modbus\mb.c | Provide Modbus slave settings and polling related interfaces for the application layer |
| FreeModbus\modbus\mb\_m.c | Provide Modbus host settings and polling related interfaces for the application layer |
| FreeModbus\modbus\functions\mbfunccoils.c | Slave coil related functions |
| FreeModbus\modbus\functions\mbfunccoils\_m.c | Host coil related functions |
| FreeModbus\modbus\functions\mbfuncdisc.c | Slave Discrete Input Related Functions |
| FreeModbus\modbus\functions\mbfuncdisc\_m.c | Discrete input related functions of the host |
| FreeModbus\modbus\functions\mbfuncholding.c | Related functions of slave holding register |
| FreeModbus\modbus\functions\mbfuncholding\_m.c | Host holding register related functions |
| FreeModbus\modbus\functions\mbfuncinput.c | Related functions of slave input register |
| FreeModbus\modbus\functions\mbfuncinput\_m.c | Host input register related functions |
| FreeModbus\modbus\functions\mbfuncother.c | Other Modbus functions |
| FreeModbus\modbus\functions\mbutils.c | Some small tools that need to be used in the protocol stack |
| FreeModbus\modbus\rtu\mbcrc.c | CRC check function |
| FreeModbus\modbus\rtu\mbrtu.c | Slave RTU mode setting and state machine |
| FreeModbus\modbus\rtu\mbrtu\_m.c | Host RTU mode setting and state machine |
| FreeModbus\port\port.c | Implement hardware porting part of the interface |
| FreeModbus\port\portevent.c | Implement slave event porting interface |
| FreeModbus\port\portevent\_m.c | Implement host event and error handling porting interface |
| FreeModbus\port\portserial.c | Serial port porting |
| FreeModbus\port\portserial\_m.c | Host serial port porting |
| FreeModbus\port\porttimer.c | slave timer porting |
| FreeModbus\port\porttimer\_m.c | Host timer porting |
| FreeModbus\port\user\_mb\_app.c | Define the slave data buffer, realize the callback interface of the slave Modbus function |
| FreeModbus\port\user\_mb\_app\_m.c | Define the host data buffer, realize the callback interface of the host Modbus function| |

> Note: All files with the suffix \_m are the files that must be used in the master mode. If the slave mode is used, these files are not required.

----------

## Two, transplant

The transplantation of the protocol stack mainly includes two aspects, hardware and software. Users need to make their own choices according to their needs.

> Note: All the following descriptions are mainly for the introduction of Modbus master mode.

### 2.1, software

The software supports the transplantation based on bare metal and real-time operating systems; supports a single host and a single slave to run independently at the same time. In addition, users can also modify the event callback interface of the protocol stack so that the interface requested by the host adopts blocking and non-blocking modes; in terms of host resource waiting, the user can also set the waiting timeout time, etc. Many functions will be introduced one by one.

#### 2.1.1, operating system and bare metal

Both the operating system and the bare metal current protocol stack are supported, but I personally recommend the use of a real-time operating system, because this will make interface calls and interface porting easier. The currently transplanted operating systems include Chinese [RT-Thread][1] (see project source code for details), UCOS and FreeRTOS.

The file involved in the process of operating system and bare metal migration is `FreeModbus\port\portevent\_m.c`

This file mainly has the following interfaces that need to be transplanted by users

|Interface |Function Description|

|:----- |:----|

|xMBMasterPortEventInit |Host event initialization|

|xMBMasterPortEventPost |Host send event|

|xMBMasterPortEventGet |Host Get Event|

|vMBMasterOsResInit |Host operating system resource initialization|

|xMBMasterRunResTake |Host Resource Acquisition|

|vMBMasterRunResRelease |Host resource release|

|vMBMasterErrorCBRespondTimeout |Host response timeout callback interface|

|vMBMasterErrorCBReceiveData |Host receives data error callback interface|

|vMBMasterErrorCBExecuteFunction |Host executes Modbus method error callback interface|

|vMBMasterCBRequestScuuess |Host request execution success callback interface|

|eMBMasterWaitRequestFinish |The host waits for the request to complete the processing callback interface|

When porting \*\*based on operating system\*\*, the technology of operating system thread synchronization is mainly used. The Modbus protocol stack itself needs to use the event mechanism of the operating system to realize the sending notification and waiting for the event. At the same time, the user requests the Modbus function. The thread of the Modbus protocol stack and the thread of the Modbus protocol stack (Modbus Poll thread) need to realize the synchronization of the two threads through the event mechanism; the host protocol stack also needs a semaphore occupied by the host resource, which is initialized to 1 by default, and the semaphore is used to ensure that multiple threads When sending a host request, only one thread can use the host.

When \*\*based on bare metal transplantation\*\*, the event notification mechanism needs to be implemented through software simulation. Both event waiting and resource waiting have to be implemented with user-defined delay and flag variables, which are more implemented than the thread synchronization mechanism in the operating system mode. It's a lot more complicated.

#### 2.1.2, data buffer

The location defined by the data buffer is at the top of the `FreeModbus\port\user\_mb\_app\_m.c` file, with a total of \*\*4\*\* data types.

By default, FreeModbus slaves use \*\*one-dimensional array\*\* as the data structure of the buffer area. The host can store the data of all slaves in the network, so the host uses a \*\*two-dimensional array\*\* to store all slave node data. The column number of the two-dimensional array represents registers, coils and discrete addresses, and the row number represents the slave node ID, but it needs to be reduced by one. For example, `usMRegHoldBuf[2][1]` means the slave ID is 3, and the register address is maintained The slave data is 1.

#### 2.1.3, Modbus data processing callback interface

Modbus has 4 different data types, and all Modbus functions operate around these data types. Because different user data buffer structures may be different, the corresponding Modbus data processing methods are also different, so users need to customize the operations corresponding to each data type according to their own data buffer structure.

All Modbus data processing callback interfaces are as follows:

|Interface |Function Description|

|:----- |:----|

|eMBMasterRegInputCB |Input register callback interface|

|eMBMasterRegHoldingCB |Holding register callback interface|

|eMBMasterRegCoilsCB |Coil callback interface|

|eMBMasterRegDiscreteCB |Discrete input callback interface|

> For the data buffer structure in the form of an array, the source code has already been transplanted and can be used directly. You can also use the [EasyDataManager](https://github.com/armink/EasyDataManager) library to use a linked list as a buffer. The library also supports event-driven, so that data changes can be automatically notified to the application layer.

### 2.2、Hardware

When porting the host part of the FreeModbus protocol stack, you need to modify the serial port and timer configuration in terms of hardware. The file is located under the port file, and the user needs to port and modify it according to their own CPU.

> Note: The protocol stack comes with STM32F103X migration files by default, users can refer to the migration

Here we mention the porting based on the operating system device driver framework. The later protocol stack will increase the porting of the [RT-Thread][1] built-in device driver framework. As long as it is an MCU supported by RT-Thread's BSP, users do not need to consider the underlying layer. The transplantation process reduces the cost of transplantation.

#### 2.2.1, serial port

The porting file related to the serial port is located in `FreeModbus\port\portserial\_m.c`. In this file, the user needs to modify the following interface methods

|Interface |Function Description|

|:----- |:----|

|vMBMasterPortSerialEnable |Enable and disable the sending and receiving functions of the serial port. If you use the 485 bus, you need to pay attention to the transceiver mode switch|

|vMBMasterPortClose |Close the serial port|

|xMBMasterPortSerialInit |Serial port initialization, if you use 485, the transceiver mode switch pin should also be initialized here|

|xMBMasterPortSerialPutByte |Serial port to send single byte data|

|xMBMasterPortSerialGetByte |Serial port receives single byte data|

|prvvUARTTxReadyISR |Serial port sending complete interrupt service program interface, according to the default method, directly reference `pxM

> It is also necessary to add the serial port service program of the CPU at the end of the file, and put the sending and receiving interrupt program interface in the above table into the corresponding interrupt service program.

#### 2.2.2、Timer

The migration file related to the timer is located in `FreeModbus\port\porttimer\_m.c`. In this file, the user needs to modify the following interface methods

|Interface |Function Description|

|:----- |:----|

|xMBMasterPortTimersInit |Timer initialization, backup the timer prescaler number and T3.5 time count value to `usPrescalerValue` and `usT35TimeOut50us` respectively|

|vMBMasterPortTimersT35Enable |Set the timer to start counting at T3.5 time|

|vMBMasterPortTimersConvertDelayEnable |Set the timer to start counting according to the conversion delay time of the broadcast frame|

|vMBMasterPortTimersRespondTimeoutEnable |Set the timer to start counting according to the response timeout time|

|vMBMasterPortTimersDisable |Disable the timer, the timer will stop counting|

|prvvTIMERExpiredISR |Timer interrupt service program interface, according to the default method, directly reference the `pxMBMasterPortCBTimerExpired` method|

> Note:

> 1. `usPrescalerValue` and `usT35TimeOut50us` are defined at the top of the file

> 2. The conversion delay time and response timeout time are in `FreeModbus\modbus\include\mbconfig.h`, which can be set by users according to the characteristics of their own system.

In addition to the above interface methods, users need to add the CPU's own timer interrupt service program at the end of the file, and put the timer interrupt service program interface in the above table into it.

## Three, API

The Modbus master is very different from the slave in the use process. The slave needs to passively wait for the request of the master, while the master actively sends out the request and receives and processes the response from the slave. When the host sends a broadcast request, the slave does not need to return a response, so the broadcast request is suitable for the master's write slave data command, not suitable for the read slave data command.

The return value format of all methods in the host request API is the same, and the meaning of the return value is as follows.

|Return Value |Description|

|:----- |:----|

|MB\_MRE\_NO\_ERR |Normal, no error|

|MB\_MRE\_NO\_REG |Register, coil or discrete input address error|

|MB\_MRE\_ILL\_ARG |The input parameter format is wrong|

|MB\_MRE\_REV\_DATA |Receive data error|

|MB\_MRE\_TIMEDOUT |Response timed out. The host did not receive the response from the slave within the set time. |

|MB\_MRE\_MASTER\_BUSY |The host is busy. During the set time, no request has not been sent. |

|MB\_MRE\_EXE\_FUN |After the host receives the response, an error occurs when executing the Modbus method (function). |

> All host request methods are \*\*thread safe\*\* and \*\*blocking mode\*\*. During use, as long as the host resource is not obtained within the set timeout period, it will return that the host is busy; if the host resource is obtained within the set timeout period, it must wait for the request result before returning.

### 3.1, write a single holding register

Write data to a holding register of the slave

```C

eMBMasterReqErrCode eMBMasterReqWriteHoldingRegister( UCHAR ucSndAddr,

USHORT usRegAddr,

USHORT usRegData,

LONG lTimeOut );

```

|Parameter |Description|

|:----- |:----|

|ucSndAddr |Requested slave address, 0 means broadcast. |

|usRegAddr |Write register address|

|usRegData |Write register data|

|lTimeOut |Request timeout time. To support permanent waiting, just use the permanent waiting parameter of the operating system. |

### 3.2, write multiple holding registers

Write data to multiple holding registers of the slave.

```C

eMBMasterReqErrCode eMBMasterReqWriteMultipleHoldingRegister( UCHAR ucSndAddr,

USHORT usRegAddr,

USHORT usNRegs,

USHORT \* pusDataBuffer,

LONG lTimeOut)

```

|Parameter |Description|

|:----- |:----|

|ucSndAddr |Requested slave address, 0 means broadcast. |

|usRegAddr |Start address of write register|

|usNRegs |Total number of write registers|

|pusDataBuffer |Write register data|

|lTimeOut |Request timeout time. To support permanent waiting, just use the permanent waiting parameter of the operating system. |

### 3.3, read multiple holding registers

Read data in multiple holding registers

```C

eMBMasterReqErrCode eMBMasterReqReadHoldingRegister( UCHAR ucSndAddr,

USHORT usRegAddr,

USHORT usNRegs,

LONG lTimeOut );

```

|Parameter |Description|

|:----- |:----|

|ucSndAddr |Requested slave address, 0 means broadcast. |

|usRegAddr |Read register address|

|usRegData |Number of read registers|

|lTimeOut |Request timeout time. To support permanent waiting, just use the permanent waiting parameter of the operating system. |

### 3.4、Read and write multiple holding registers

Read multiple registers first, and then write multiple registers.

```C

eMBMasterReqErrCode eMBMasterReqReadWriteMultipleHoldingRegister( UCHAR ucSndAddr,

USHORT usReadRegAddr,

USHORT usNReadRegs,

USHORT \* pusDataBuffer,

USHORT usWriteRegAddr,

USHORT usNWriteRegs,

LONG lTimeOut)

```

|Parameter |Description|

|:----- |:----|

|ucSndAddr |Requested slave address, 0 means broadcast. |

|usReadRegAddr |Read register address|

|usNReadRegs |Number of read registers|

|pusDataBuffer |Write register data|

|usWriteRegAddr |Address of write register|

|usNWriteRegs |Number of write registers|

|lTimeOut |Request timeout time. To support permanent waiting, just use the permanent waiting parameter of the operating system. |

3.5 Read multiple input registers

Read data in multiple input registers

*eMBMasterReqErrCode eMBMasterReqReadInputRegister*