Modbus-RTU Slave Lite Library User Guide

### Introduction

This library provides a source code level implementation of Modbus-RTU slave on serial line. Only a few driver level operations are needed to apply the library on specific target system, such as serial port operation, timer operation.

This is a lite library because only the most commonly used holding register area and corresponding function codes are supported.

Table 1.1 Implemented Modbus Functions

|  |  |  |
| --- | --- | --- |
| **Register Area** | **Function Code** | **Function Description** |
| Holding Registers | 0x03 | Read Holding Registers |
| 0x06 | Write Single Register |
| 0x10 | Write Multiple Registers |

### Porting

Before using this library on your platform, following several functions must be prepared at first.

#### Supporting Functions: Serial Port

**Comment:** Users can give function name at their options. There is no special restriction on function name so long as all the needed functions are rightly given and called. But all the function prototypes must be strictly abided.

/\*! \name funcPortInit

\* \brief Initialize physical port

\* \param baud: baud rate setting, (2400~115200 bps)

\* \param parity: parity check mode setting, ('N' 'E' or 'O')

\* \param frmSpace: min RTU frame space setting, unit: ms

\* \return TRUE-success

\* FALSE-failed (invalid arguments)

\*/

* **BOOL8 funcPortInit (uint32 baud, uint8 parity, uint16 frmSpace)**

/\*! \name funcGetRcvCnt

\* \brief Get count of received data

\* \return count of received data

\*/

* **uint16 funcGetRcvCnt (void)**

/\*! \name funcGetLastRcvTime

\* \brief Get time of last data received

\* \return time of last data received (ms)

\*/

* **uint32 funcGetLastRcvTime (void)**

/\*! \name funcRcvFrm

\* \brief Get received modbus frame out from port

\* \param pDes: destination to save received frame

\* \return length of received frame

\*/

* **uint16 funcRcvFrm (uint8 \*pDes)**

/\*! \name funcIsSndIdle

\* \brief Check whether port is idle for sending of modbus.

\* \return TRUE- port is idle for sending

\* FALSE- port is busy for sending

\* \comment: For half-duplex mode, sending is always idle while received

\* a frame, so can ignore check and always return TRUE here.

\*/

* **BOOL8 funcIsSndIdle (void)**

/\*! \name funcSndFrm

\* \brief Send modbus frame out

\* \param pFrm: pointer to the frame to be send out

\* \param size: length of frame

\* \return TRUE-sending successful,

FALSE-sending failed, buffer is full

\*/

* **BOOL8 funcSndFrm (uint8 \*pFrm, uint16 size)**

#### Supporting Functions: Timer

A global timer is needed to generate 1ms timing. Each received data must be labeled with time stamp. Following function must be provided to calculate the interval from reference time to current time.

**Comment:** User must strictly abide the function name as well as prototype.

/\*! \name GetInterval

\* \brief Get the interval from reference time to current time. (unit: ms)

\* \param prevTime: previous reference time (unit: ms)

\* \return the interval from reference time to current time (unit: ms)

\*/

* **uint32 GetInterval (uint32 prevTime)**

### Implementation

Follow steps below to implement Modbus slave.

**Step 1.** Add library source files into your project.

**Step 2.** Create a new file (e.g. “mbs1.c”) to define a Modbus instance like below.

MB\_RTU\_SLV mbs1; //Modbus structure

static uint8 sndBuf[SND\_BUF\_SIZE] = {0}; //send buffer for response frame

static uint16 mbReg[MB1\_REG\_SIZE] = {0}; //Holding register area

In head file (e.g. “mbs1.h”), define size of Modbus register area like below.

#define MB1\_REG\_SIZE 40 //total size of register area

#define MB1\_REG\_STADDR 0 //start Modbus address of whole register area

#define MB1\_RW\_REG\_STADDR 30 //start Modbus address of RW section

#define MB1\_RW\_REG\_SIZE 10 //size of RW section in register area

**Step 3.** Call initialization function like below to initialize Modbus slave.

MbsInit(&mbs1, //overall structure

9600, ‘N’, //port parameters

1, //slave device ID

mbReg, MB1\_REG\_SIZE, MB1\_REG\_STADDR, //register area

MB1\_RW\_REG\_STADDR, MB1\_RW\_REG\_SIZE, //RW section in register area

sndBuf, SND\_BUF\_SIZE, //send buffer

funcPortInit, //porting function…

funcGetRcvCnt,

funcGetLastRcvTime,

funcRcvFrm,

funcIsSndIdle,

funcSndFrm);

**Step 4.** Call Modbus polling service in main loop like below.

MbsSrv(&mbs1); //polling Modbus slave

### Configuration

Modbus register area can be configurated by macro defines as described in above.

### Interface

Four functions are provided for external access to Modbus register data.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name : ReadMbReg

Arguments : [in]pMbs: pointer to the modbus object

[in]index: data index in the modbus register array

Return : data to be read, if 'index' overrun, return 'FAIL'(0xFFFF)

Description : read a single modbus register.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

* **uint16 ReadMbReg (MB\_RTU\_SLV \*pMbs, uint8 index)**

{

if (index >= pMbs->regSize)

return FAIL;

return pMbs->pMbReg[index];

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name : WriteMbReg

Arguments : [in, out]pMbs: pointer to the modbus object

[in]index: data index in the modbus register array

[in]data: data to be written in

Return : TRUE-success

FALSE-failed, 'inedx' overrun

Description : write a single modbus register.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

* **uint8 WriteMbReg (MB\_RTU\_SLV \*pMbs, uint8 index, uint16 data)**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name : ReadMassMbReg

Arguments : [in]pMbs: pointer to the modbus object

[in]index: start data index in the modbus register array

[out]pDes: desitination point to save data to be read

[in]size: amount of words to be read

Return : If failed, return FALSE

If successful, return 'TRUE'

Description : read a serial mass of modbus registers.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

* **uint8 ReadMassMbReg (MB\_RTU\_SLV \*pMbs, uint8 index, uint16 \*pDes, uint8 size)**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name : WriteMassMbReg

Arguments : [in, out]pMbs: pointer to the modbus object

[in]index: start data index in the modbus register array

[in]pSrc: point to source data to be writen in

[in]size: amount of words to be read

Return : If failed, return FALSE

If successful, return 'TRUE'

Description : write a serial mass of modbus registers.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

* **uint8 WriteMassMbReg (MB\_RTU\_SLV \*pMbs, uint8 index, uint16 \*pSrc, uint8 size)**