KD485 Modbus Library

User Manual

Prepared for KK Systems by Cooke Technologies Ltd

KK Systems Ltd
Tates, London Road
Pyecombe, Brighton
BN45 7ED
Great Britain

tel +44 1273 857185
fax +44 1273 857186
email sales@kksystems.com
http://www.kksystems.com

Cooke Technologies

Cooke Technologies have over ten years experience in software design and system integration for many areas of industry and manufacture. Our key area of expertise is the design of software for control and measurement, including serial protocols. We also have wide experience in electronics and data communications.

Modbus Protocols: a De-Facto Standard

The Modbus protocols were devised by Modicon but have already become a de-facto standard in the process industry. Having been developed for a mission critical environment they include checksums, and resend mechanisms rarely found in less developed protocols. The documentation is readily available. This contrasts with many other protocols that are available only under confidentiality agreements and useage restrictions.

The concept is a set of addressable registers, that can hold, binary, numeric, or textual data. How these registers are assigned and used is left to the user, which means that the protocol can be used for virtually any application.

Both ASCII and binary protocols are documented, but the binary ones, called RTU, are best suited to modern technology.

A key feature of the protocol is the ability to read or write to a group of consecutive registers with a single command. Partitioning the registers in a sensible way is therefore an important aspect of writing a Modbus driver. Our Modbus library can be optimised by using KK Systems' hardware.

Many Modbus slaves can be connected to a single Modbus master, via an RS 422 connection. Thus there are two types of Modbus drivers. Control and measuring instruments

Introduction

The KD485 Modbus Library is a library of C functions designed to work with the KK Systems KD485 Universal Interface Converter. The functions can be used in user programs to implement conversions of proprietary protocols to and from the RTU mode Modicon Modbus protocol.

The library is supplied as an object file with which the user can link their code using the supplied batch file mkmbhex.bat. This adds a number of available functions over and above the in-built KD485 functions to make support of the Modbus protocol as straightforward as possible. Prototypes of all exported functions are included in the supplied modbus.h file.

Application Areas

The KD485 Modbus library is aimed at solving an increasingly common problem: connecting a SCADA package to equipment that uses a proprietary protocol. Typically, some special equipment needs to be made part of a factory or control system and this requires that it be interfaced to a SCADA package. Normally, to do this would require a suitable software driver.

However, the cost of writing a specialised software driver can be prohibitive. It is in these circumstances that a protocol converter can be an effective solution.

When this approach is taken it makes sense to convert to the Modicon Modbus protocol. This is rapidly becoming a de facto standard and virtually all SCADA packages support it.

The KD485-PROG programmable interface converter, together with the KD485 Modbus library, can be the basis of such a solution.

The KD485

The KD485 is a multi-mode interface converter manufactured by KK Systems Ltd. which contains two serial asynchronous ports. This library is designed for use with the KD485-PROG user programmable version. This version contains an EEPROM and a means of uploading user written programs into the EEPROM. The KD485 will then execute this user program on power up.

With a suitable user program, one port of the KD485 can be connected to the equipment and will communicate with it in the appropriate proprietary protocol; the other port can be connected to the SCADA system, where it will behave as the serial port of a Modbus slave device.

The KDCFG configuration utility provided with the KD485 is the most convenient way to upload programs. See the KDCFG help file for details on its operation.

Installation

The KD485 configuration utilities, Hi-Tech C compiler etc. should all be installed before this product. See the KD485 user manual for details, the C reference section in particular.

If you are using Windows, you can install all the library together with all the associated example files and test tools by running ‘install.exe’ from the disk. You should install everything to the same directory as the original KD485 material as the ‘mkmdhex.bat’ make file assumes that various tools will be found in the directory it is running from. The Windows installation also installs a Modbus driver needed for the test tools and makes some changes to the System.ini file to enable this driver.

If you are using DOS, you can install the library by copying the contents of the \dos directory to your hard drive. This should be copied to the same directory as the KD485 material for the same reason as the Windows version. The test tools are not included as they only run under Windows.

Typical application

A typical application for the KD485 Modbus Library would be to attach an instrument which does not use the Modbus protocol to a SCADA or similar system which does use it. A user program in the KD485, written with the aid of the library, would convert the instrument’s protocol to Modbus so that it could be used as a Modbus slave device by the system.

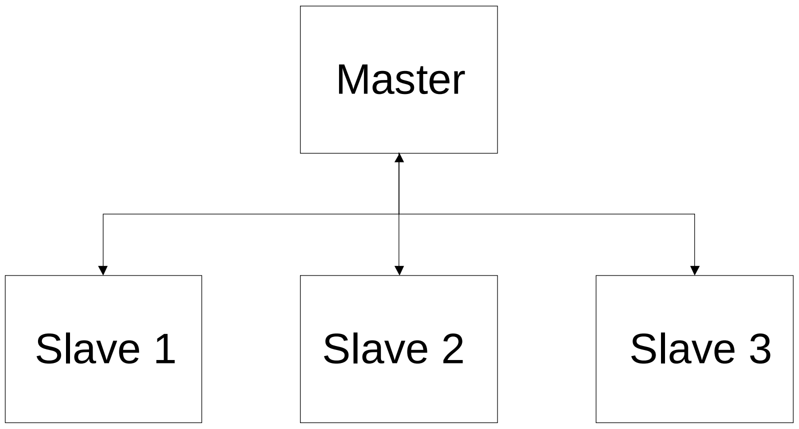
Another application might be for the purposes of concentration; the KD485 could be attached to a number of Modbus slave devices and act as the Master device. It could then emulate a single Slave device using the library and thus present all the various Slave devices to the system as one device.

Modbus Protocol Overview

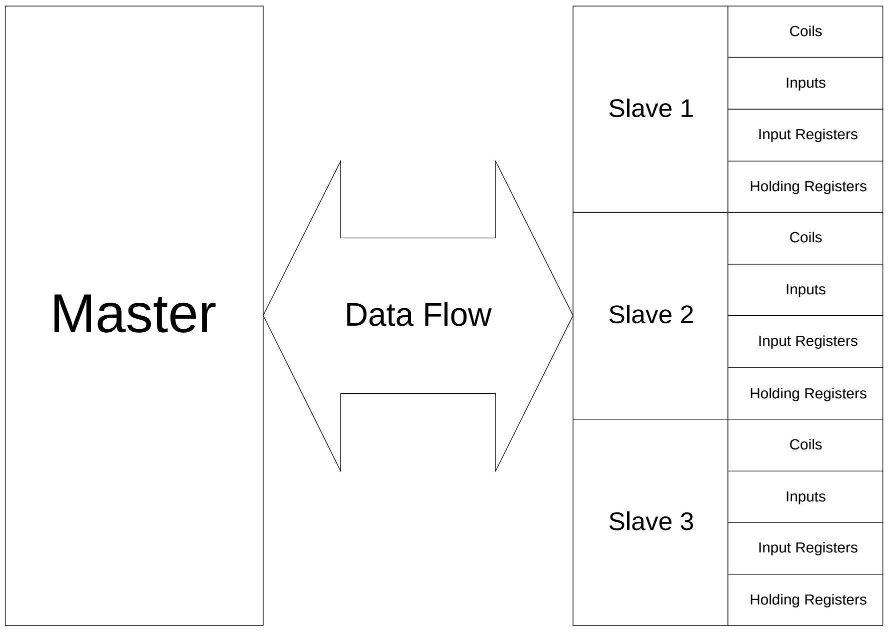
The Modbus protocol is supported by virtually all SCADA packages as a general purpose protocol for reading and writing values in external devices. It can operate in two modes, RTU and ASCII; only RTU mode is supported by this library.

In a Modbus system, there are two types of devices; Masters and Slaves. Only a Master device can initiate a transaction. Slave devices can transmit information only as a result of a Master device's request. A Modbus system can contain only one Master device. This library is designed to ease emulation of Modbus Slave devices.

Conceptually, the protocol treats each Slave device as if it contained a block of memory addresses, some of which are read only. These are called registers and can be accessed individually or in consecutive blocks.



Physical structure of a Modbus system - the Modbus Master is connected to one or more slaves via serial leads. Each Slave unit has a unique address.



Conceptual structure of a Modbus system - the Modbus Master can access any register or block of consecutive registers via read and/or write commands, treating them as the equivalent of memory addresses.

Each KD485 running the Modbus library is treated as a Slave device. The Modbus slave address and ID of the KD485 can be set using the library's 'SetSlaveAddress' and

'SetSlaveID' functions; otherwise it will default to the values in the KD485's internal settings. (See 'KD485 Configuration and its effect on the Modbus Library' for details.)

The protocol supports commands for accessing four basic variable (register) types plus a number of other commands for getting diagnostic and status information. The KD485 library supports all but Modicon device specific calls. A method is provided for allowing user defined responses for all Modbus commands that the library itself does not support if required.

The four types of variable that the Modbus protocol interfaces with are as follows:

- Coils - Read/Write single bit data. Numbered from 00001 to 10000.
- Inputs - Read only single bit data. Numbered from 10001 to 30000.
- Input Registers - Read only 16 bit data. Numbered from 30001 to 40000.
- Holding Registers - Read/Write 16 bit data. Numbered from 40001 upwards.

Modbus Commands

The Modbus protocol defines 24 commands. The following are implemented by the library:

- ReadCoilStatus (Code 1) - Reads a number of Coils.
- ReadInputStatus (Code 2) - Reads a number of Inputs.
- ReadHoldingRegisters (Code 3) - Reads a number of Holding Registers.
- ReadInputRegisters (Code 4) - Reads a number of Input Registers.
- ForceSingleCoil (Code 5) - Sets the state of a single Coil.
- PresetSingleRegister (Code 6) - Writes to a single Holding Register.
- ReadExceptionStatus (Code 7) - Reads a set of eight exception registers. By default the library returns a zero exception code unless the user has specified a set of exception registers.
- Diagnostic (Code 8) - Returns various diagnostic data.
- FetchCommEventCtr (Code 11) - Returns number of communication events.
- FetchCommEventLog (Code 12) - Returns a character array showing status of latest 64 events.
- ForceMultipleCoils (Code 15) - Writes to a number of Coils.
- PresetMultipleRegisters (Code 16) - Writes to a number of Holding Registers.
- ReportSlaveID (Code 17) - Returns device ID code. The KD485 returns a device ID of 10 by default though the user can set other values between 10 and 255
- ResetCommLink (Code 19) - Resets the communications link. In the KD485 it simply flushes the input buffer.
- MaskWrite4XRegister (Code 22) - Allows a logical AND and OR operation to be performed on a Holding Register.
- ReadWrite4XRegisters (Code 23) - Allows a combined read and write operation on different Holding Register sets in one operation.

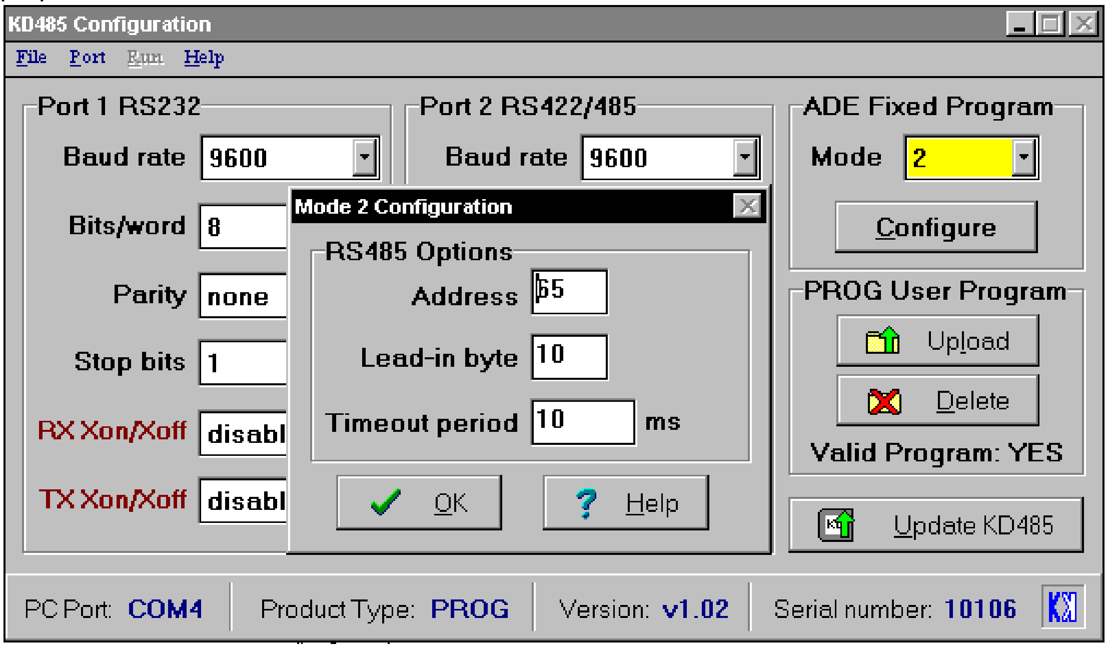
The following eight commands are not implemented by the library:

- Program484 (Code 9) - Modicon device specific call.

- Poll484 (Code 10) - Modicon device specific call.
- ProgramController (Code 13) - Modicon device specific call.
- PollController (Code 14) - Modicon device specific call.
- Program884 (Code 18) - Modicon device specific call.
- ReadGeneralReference(Code 20) - Modicon device specific call.
- WriteGeneralReference(Code 21) - Modicon device specific call.
- ReadFIFOQueue (Code 24) - Modicon device specific call.

KD485 Configuration and its effect on the Modbus Library

Certain parts of the KD485 configuration accessed through the KDCFG.exe configuration application can affect the operation of the Modbus library. This application sets up values in the KD485's EEPROM which the library can access and use. Specifically, it can be used to set the Modbus address and slave ID of the KD485 and also some of its timing properties.



Port 1 RS232
Baud rate 9600
Bits/word 8
Parity none
Stop bits 1
RX Xon/Xoff disabl
TX Xon/Xoff disabl

Port 2 RS422/485
Baud rate 9600

ADE Fixed Program
Mode 2

RS485 Options
Address 65
Lead-in byte 10
Timeout period 10 ms

PROG User Program
Valid Program: YES

PC Port: COM4 Product Type: PROG Version: v1.02 Serial number: 10106

The 'Address' and 'Lead-in byte' correspond to the Modbus slave address and slave ID respectively

First, run the KDCFG utility with the KD485 connected and in executive mode, as described in the KD485 user manual (page 16).
The Modbus address and slave ID can be set by changing the 'ADE Fixed Program' mode to '2' and pressing the 'Configure' button. The 'address' field then shown in the dialog box will be used by the library as the Modbus slave address unless it is overridden by a 'SetSlaveAddress' function call in the user program. This value can be edited from this dialog box. Similarly, the value marked 'Lead-in byte' will be used as the slave ID unless it is overridden by a 'SetSlaveID' function call.

KDCFG also contains a facility for changing the RS485 driver turn-off delays. This is obtained by changing the 'ADE Fixed Program' mode to '1' and pressing the 'Configure' button. These delays can affect the behaviour of a KD485. If a bus contention error is experienced, the delays may be too high and should be reduced. If the last character in a transmitted message is being cut off or garbled, the delays may be too low and should be increased.

The 'ADE Fixed Program Mode' should be reset to 'Off' and the 'Update KD485' button pressed after the appropriate changes have been made. The 'ADE Fixed Program Mode' must be set to 'Off' in order to run a user program.

Building the Application

Once the C file representing the user program has been written, it can be compiled and linked with the Modbus library by using the ‘Mkmbhex.bat’ batch file. (Note: This assumes you are using the standard Hi-Tech C Compiler described in the KD485 User Manual.) This takes as its one parameter the name of the C file without the extension. E.g., if the C file is ‘myprog.c’, then

```
mkmbhex myprog
```

- is the appropriate command.

If there are no compiler errors, this will produce a file named ‘myprog.hex’. This is the file which should be uploaded to the KD485 using the KDCFG configuration utility.

In KDCFG, the ‘ADE Fixed Program Mode’ should be set to ‘Off’ before pressing the ‘Upload’ button to upload the hex file into the KD485. You can then select the ‘Run’ menu option to run the user program immediately, or power down the KD485, in which case the program will execute on the next power up.

Modbus Library Function Descriptions

The general idea of the Modbus library is that it allows the user to declare their own arrays that will then be accessed by the library as though they were the Modbus coils, inputs and registers. Input and holding registers are accessed as integer arrays with one element per register and, although the coils and inputs are actually single bit data, the Modbus library uses character arrays, with one character per bit; this saves the user from having to “bit fiddle” to determine the contents.

In addition, user defined call-back functions can be specified that the library calls when access is needed to the arrays by the controlling Modbus master. For functions where the master writes to a set of data the user call-back is called after the data has been written, allowing the user to take appropriate action with the new data. For functions where the master reads a set of data the call-back is called before the data is read to allow the user to construct the data before it is returned. The function is passed two parameters, the start address of the registers, and the number of registers affected.

The user defined call-back functions return a boolean value. This should be FALSE if the library is still required to handle the command or TRUE if the call-back has already handled it, for example by returning an exception with the ‘ExceptResponse’ function.

Thus most of the function calls are those to set the arrays and call-backs. In addition to these calls the user also has the option of setting up a set of eight exception registers the contents of which are returned when the library receives a ReadExceptionStatus command and an option to patch a function to handle the Modbus commands which the library does not normally handle. To facilitate writing these response functions the library also exports the appropriate CRC calculation algorithm needed for the Modbus checksum. Finally a function is supplied to allow the user to set the device ID which is returned by the GetSlaveID command.

The following is a complete list of all exported functions and their purpose:

Typedefs for call-back functions supplied by user.

typedef bool (*FUNCPTR)(uint16 regstart, uint16 numregs);

typedef void (*UNKNOWNFUNCPTR)(uint16 funcnum, uint8 *mess, uint16 messlen);

- **int16 SetPort(int16 portnum)**

Defines which of the KD485 ports is to have the Modbus library attached. Must be 1 or 2. Returns 0 for success, -1 otherwise. This function should be called before the Start function, but after the KD485 function for setting the baudrate if used.

- **int16 SetSlaveAddress(int16 address)**

Set the Modbus slave address, the library will only respond to Modbus messages addressed to this number. In addition the library responds to all broadcast messages (those addressed to zero) as appropriate. If this function is not called the address will default to that set as the 485 address using the configuration utility in mode 2.

- **int16 SetSlaveID(int16 id)**

Set the Modbus slave id returned by function code 17 (ReportSlaveID) . If this function is not called the ID will default to that specified as the lead-in byte defined using the configuration utility in mode 2.

- **int16 SetCoils(uint8 *coilbuf, int16 numcoils)**

Specifies a character array that the Modbus library reads and writes as coils. If the coil array has not been set the library will respond to read or write access commands with an exception code of ILLEGAL DATA ADDRESS. Similarly if any attempt is made by the master to access coils beyond the maximum number specified an ILLEGAL DATA ADDRESS exception is returned.

- **int16 SetCoilReadCallback(FUNCPTR func)**

Sets a function that will be called whenever a Modbus ReadCoilStatus (code 1) command is decoded by the library. This function is called by the library before the data is read from the coil array and packed into the response message.

- **int16 SetCoilWriteCallback(FUNCPTR func)**

Sets a function that will be called whenever a Modbus ForceSingleCoil (code 5) or ForceMultipleCoils (code 15) command is decoded. This function is called after the data has been written to the coil array.

- **int16 SetInputs(uint8 *inpbuf, int16 numinputs)**

Specifies a character array that the Modbus library reads as Inputs. If the inputs array has not been set the library will respond to read or write access commands with an exception code of ILLEGAL DATA ADDRESS. Similarly if any attempt is made by the master to access inputs beyond the maximum number specified an ILLEGAL DATA ADDRESS exception is returned.

- **int16 SetInputsReadCallback(FUNCPTR func)**

Sets a function that will be called whenever a ReadInputStatus (code 2) command is decoded. This function is called by the library before the data is read from the inputs array and packed into the response message.

- **int16 SetInputRegs(int16 *regbuf, int16 numregs)**

Specifies a 16 bit integer array that the Modbus library reads as input registers. If the input register array has not been set the library will respond to read or write access commands with an exception code of ILLEGAL DATA ADDRESS. Similarly if any attempt is made by the master to access input registers beyond the maximum number specified an ILLEGAL DATA ADDRESS exception is returned.

- **int16 SetInRegsReadCallback(FUNCPTR func)**

  Sets a function that will be called whenever a ReadInputRegister (code 4) command is decoded**.** This function is called by the library before the data is read from the input register array and packed into the response message.

- **int16 SetHoldingRegs(int16 *regbuf, int16 numregs)**

  Specifies a 16 bit integer array that the Modbus library reads and writes as holding registers. If the holding register array has not been set the library will respond to read or write access commands with an exception code of ILLEGAL DATA ADDRESS. Similarly if any attempt is made by the master to access holding registers beyond the maximum number specified an ILLEGAL DATA ADDRESS exception is returned.

- **int16 SetHoldRegsReadCallback(FUNCPTR func)**

  Sets a function that will be called whenever a ReadHoldingRegister (code 3) or a ReadWrite4XRegisters (Code 23) command is decoded. This function is called by the library before the data is read from the input register array and packed into the response message.

- **int16 SetHoldRegsWriteCallback(FUNCPTR func)**

  Sets a function that will be called whenever a PresetSingleRegister (code 6), a PresetMultipleRegs (code 16), ReadWrite4XRegisters (Code 23) or a MaskWrite4Xregister (code 22) command is decoded. This function is called after the data has been written to the holding register array.

- **int16 SetExceptionRegisters(uint8 *exbuf)**

  Sets the 8 coils that are used to hold exception status. If the user does not specify this array the library responds to the ReadExceptionStatus (code 7) with a zero status. The length of the supplied character array must be at least eight.

- **int16 SetExReadCallback(FUNCPTR func)**

  Sets a function that will be called whenever a ReadExceptionStatus (code 7) command is decoded.

- **int16 SetUnknownCallback(UNKNOWNFUNCPTR func)**

  Sets a function to be called whenever an un-handled command is decoded. If this function is not patched the default action is to return an ILLEGAL FUNCTION exception. The supplied call-back is given the function number, a pointer to a character array containing the message and the length of the message. If this function is patched it is the users responsibility to build the appropriate response and return it as per the Modbus protocol definition. This function, if patched, will be called whenever a Modicon device specific command is decoded as well as command numbers over 24.

- **void ExceptResponse(int16 xcode)**

  Returns an exception response message. Codes are defined in modbus.h. If this function is called in a callback function return TRUE from the callback to stop the library from making any further response.

- **uint16 revcrcbuf(uint8 *buf, uint16 num, uint16 seed)**

  Checksum calculator, set seed to -1 for modbus.

- **int16 Start(void)**

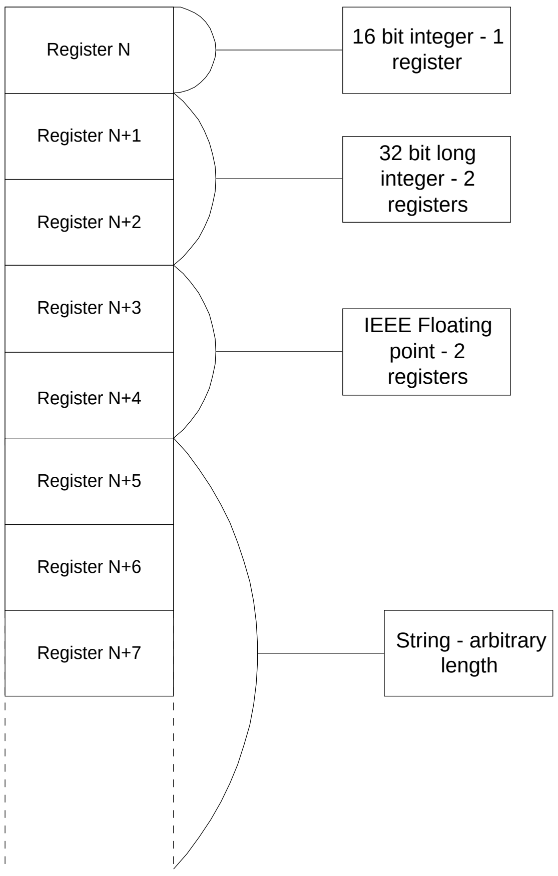
  Starts the Modbus library monitoring the specified communications port. Returns 0 for success, -1 if SetPort has not yet been called.

- **int16 Stop(void)**

  Stops the Modbus library monitoring the specified port.

Modbus Data Types

Modbus holding and input registers are 16 bit values which can represent four different data types: the 16 bit integer itself, a 32 bit long integer, a string, or a floating point number. The long integer and the floating point types are both stored in two registers per value. The string is stored as two characters per register.


Register N
Register N+1
Register N+2
Register N+3
Register N+4
Register N+5
Register N+6
Register N+7
16 bit integer - 1 register
32 bit long integer - 2 registers
IEEE Floating point - 2 registers
String - arbitrary length

Floating point numbers are held in 32-bit IEEE floating point format. Each value has two registers assigned to it; the eight most significant bits represent the exponent and the other 23 bits (plus one assumed bit) represent the mantissa and sign of the value.

Header File modbus.h

```
/*
      Filename : Modbus.h
      Author : Stephen Constable
      Copyright 1996 Cooke Technologies Ltd.
*/

/* Exception codes used with ExceptResponse */
#define ILLEGAL_FUNCTION 1
#define ILLEGAL_DATA_ADDRESS 2
#define ILLEGAL_DATA_VALUE 3
#define SLAVE_DEVICE_FAILURE 4
#define ACKNOWLEDGE 5
#define SLAVE_DEVICE_BUSY 6
```

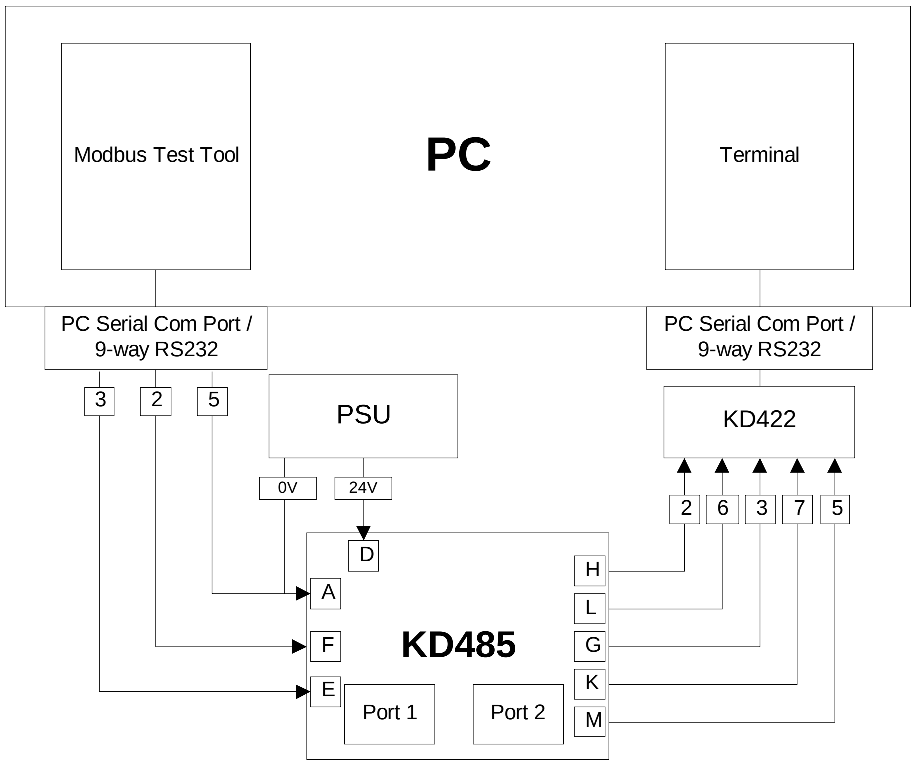
```
      #define NEGATIVE_ACKNOWLEDGE 7
      #define MEMORY_PARITY_ERROR 8
      /* Boolean defines */
      #define TRUE 1
      #define FALSE 0

      /* Common typedefs */
      typedef unsigned char uint8;
      typedef char int8;
      typedef short int16;
      typedef unsigned short uint16;
      typedef long int32;
      typedef unsigned long uint32;
      typedef int16 bool;

      /* Typedef for callback functions supplied by user
            If the callback returns an exception response (i.e.
handles
            the reply itself) then the callback should return TRUE
which
            will prevent the library from taking any more action.
Otherwise
            return FALSE.
      */
      typedef bool (*FUNCPTR)(uint16 regstart, uint16 numregs);


      /* Unknown callbacks MUST respond to the message themselves if
a response
            is required.
            The parameters supplied give access to the complete
message so
            the user can do whatever is necessary with it*/
      typedef void (*UNKNOWNFUNCPTR)(uint16 funcnum, uint8 *mess,
uint16 messlen);


      /* Defines which of the KD485 ports is to have the Modbus
library attached */
      int16 SetPort(int16 portnum);

      /* Set the modbus slave address 1-255 */
      int16 SetSlaveAddress(int16 address);

      /* Set the modbus slave id (0-255) returned by function code 17
(ReportSlaveID) */
      int16 SetSlaveID(int16 id);

      /* Specifies a character array that the Modbus library reads
and writes as coils. */
      int16 SetCoils(uint8 *coilbuf, int16 numcoils);

      /* Sets a function that will be called whenever a Modbus
ReadCoilStatus (code 1) command is decoded by the library. */
      int16 SetCoilReadCallback(FUNCPTR func);

      /*  Sets a function that will be called whenever a Modbus
ForceSingleCoil (code 5) or
             ForceMultipleCoils (code 15) command is decoded. */
      int16 SetCoilWriteCallback(FUNCPTR func);

      /* Specifies a character array that the Modbus library reads as
Inputs. */
      int16 SetInputs(uint8 *inpbuf, int16 numinputs);

      /* Sets a function that will be called whenever a
ReadInputStatus (code 2) command is decoded. */
```

```
      int16 SetInputsReadCallback(FUNCPTR func);

      /* Specifies a 16 bit integer array that the Modbus library
reads as input registers. */
      int16 SetInputRegs(int16 *regbuf, int16 numregs);

      /* Sets a function that will be called whenever a
ReadInputRegister (code 4) command is decoded. */
      int16 SetInRegsReadCallback(FUNCPTR func);

      /* Specifies a 16 bit integer array that the Modbus library
reads and writes as holding registers. */
      int16 SetHoldingRegs(int16 *regbuf, int16 numregs);

      /* Sets a function that will be called whenever a
ReadHoldingRegister (code 3) command is decoded. */
      int16 SetHoldRegsReadCallback(FUNCPTR func);

      /* Sets a function that will be called whenever a
PresetSingleRegister (code 6), a PresetMultipleRegs (code 16)
            or a MaskWrite4Xregister (code 22) command is decoded. */
      int16 SetHoldRegsWriteCallback(FUNCPTR func);

      /* Sets the 8 coils that are used to hold exception status. */
      int16 SetExceptionRegisters(uint8 *exbuf);

      /* Sets a function that will be called whenever a
ReadExceptionStatus (code 7) command is decoded. */
      int16 SetExReadCallback(FUNCPTR func);

      /* Sets a function to be called whenever an unknown command is
decoded.
            If this function is not patched the default action is to
return
            an ILLEGAL FUNCTION exception. */
      int16 SetUnknownCallback(UNKNOWNFUNCPTR func);

      /* General error function, return TRUE from callbacks if used
*/
      void ExceptResponse(int16 xcode);

      /* Checksum calculator, set seed to -1 for modbus */
      uint16 revcrcbuf(uint8 *buf, uint16 num, uint16 seed);

      /* Starts the Modbus library monitoring the specified
communications port. */
      int16 Start(void);

      /* Stops the Modbus library monitoring the specified port. */
      int16 Stop(void);
```

Example 1: Echo

File: Example1.C

This is a simple example which provides no protocol translation; it simply demonstrates how to set up the Modbus library and use callbacks to respond to incoming Modbus commands. If the KD485 is connected to a Modbus master (for example, the Test Tool provided) on one port this example will output an ASCII string representing the commands received through the other port. This could then be displayed by connecting a PC running a terminal program to the port.



Wiring Diagram for the KD485 running Example 1

This example sets the slave address of the instrument to be 100 decimal.

Use the batch file mkmbhex.bat to compile and load the resulting hex file in the usual way. See ‘Building the Application’ for details.

Example 2: Protocol conversion - Internal Database; Immediate access

File: Example2.C

This example is a demonstration of protocol conversion which converts the proprietary protocol of the provided software dummy instrument to Modbus, thus making it appear as a Modbus slave device. Values can then be written to or read from the instrument using a Modbus master device, e.g. the provided Test Tool.

The code polls the dummy instrument continuously using its proprietary protocol and stores the data internally in the arrays representing the Modbus registers. The data is forwarded to Modbus automatically by the library when an appropriate command is received. Writing to the registers is handled by setting flags in the write callbacks, with the actual writing to the instrument handled in the poll loop.

This example sets the slave address of the instrument to be 100 decimal.

The disadvantage of this approach over that taken in Example 3 is that the data returned to Modbus is the data obtained on the last poll loop, which may not be current (depending on the time it takes the loop to poll all the needed data values). It is also harder to handle exceptions; this example ignores exceptions, but to handle them properly would involve setting flags in the poll loop if there was a problem with the instrument or its data. The exception could then be raised in the appropriate callback when Modbus requested the data.

The principle advantage of this approach is that the poll loop can run continuously updating the registers, so any Modbus data request can be performed instantly by the library. Thus even if the instrument is slow to respond there should never be a problem with Modbus timing out.

Example 3: Protocol conversion - Commands relayed; Slow

File: Example3.C

This example is a demonstration of protocol conversion. As in the previous example, it converts the protocol of the dummy instrument to Modbus and vice versa.

This example interprets incoming Modbus commands and requests the data from or writes data directly to the instrument. This is handled in callback routines. The Modbus protocol is thus converted to the proprietary protocol of the software instrument and the proprietary protocol back to Modbus. All of the instrument’s values can then be read as Modbus registers.

This example does not set the slave address of the instrument; the slave address is therefore taken from the KD485’s configuration.(See ‘KD485 Configuration and its effect on the Modbus Library’).

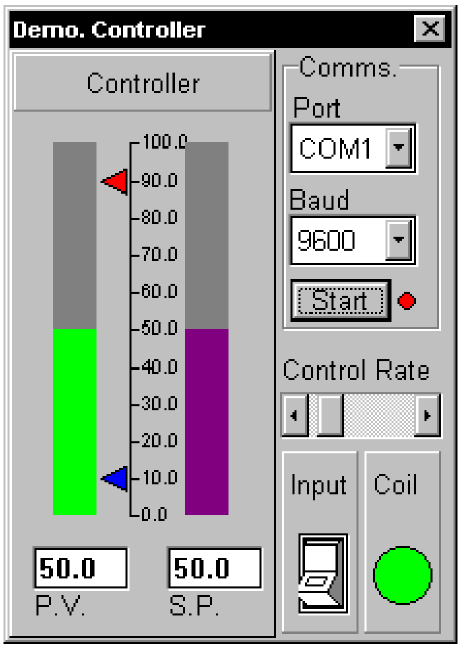
The disadvantage of this type of protocol conversion is that the KD485 must retrieve its data from the instrument and reply within the Modbus timeout interval. This may not be possible if the instrument is slow to respond. The advantage of this method is that if data is available in the time allowed it will be the current data value rather than that obtained during the last poll, as it would be in an asynchronous conversion. This type of conversion also allows easier handling of exceptions, as they can be directly generated in the callbacks.

Appendix A: The Dummy Instrument

Included in this package is a software dummy instrument for the example programs to communicate with. The application is in the file ‘Demoinst.exe’ and runs under Windows. It is installed with the Windows installation option.

The software represents a process controller type instrument with four associated values: a process value, a set point, the state of a switch and the state (colour) of an LED. The instrument is driven by a proprietary (non-Modbus) protocol described below. When the protocol is converted to Modbus, each of the four internal values can be associated with one of the four Modbus register types.

- The process value is read only and can be represented by an Input Register
- The set point is read/write and can be represented by a Holding Register
- The switch status is read only and can be represented by an Input
- The LED status is read/write and can be represented by a Coil



The dummy instrument

The instrument can be connected to a serial port by setting the ‘Port’ and ‘Baud’ to the desired values and pressing ‘Start’. The Baud rate should match that of the KD485 port the instrument is to be connected to; this can be seen and set in the KDCFG configuration utility. The Set Point can be edited in the ‘S.P.’ edit box. When the Set Point is changed, the dummy Process Value will adjust itself to equal the Set Point at a rate determined by the ‘Control Rate’ slider.
The ‘Input’ switch can be clicked to change it from off (0) to on (1).
The Coil LED colour can be either red (0) or green (1).

Instrument Protocol

This device has a simple proprietary protocol that defines the following commands:

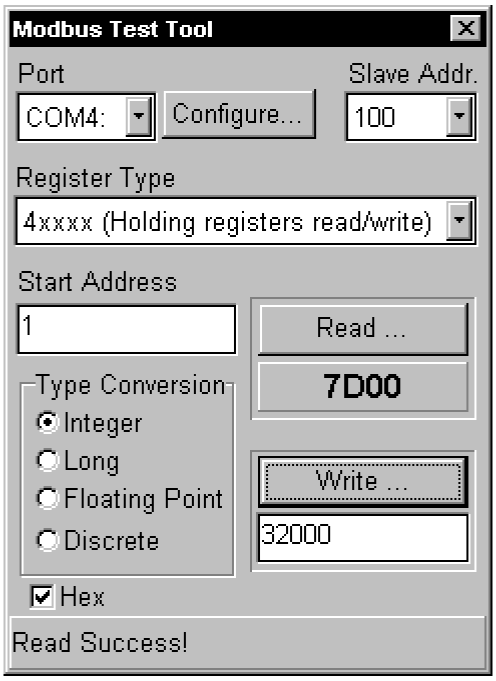
?1	: Return current Process Value as a string
?2n	: Set value of Set Point (holding register) determined by 'n' which is a number between 0-2^12 (0-4096) (instrument only displays values to 100)
?3	: Return value of Set Point as a string
?4	: Return current state of switch as a string

?5n : Set state of LED (coil) determined by 'n' which is 0 or 1
?6 : Return current state of LED as a string

All returned strings are assumed to be terminated by a carriage return/line feed pair. Examples 2 and 3 both convert this proprietary protocol so that the instrument can be treated as a Modbus slave device.

Appendix B: The Modbus Master Test Tool

Included in this package is a simple Modbus Master test tool for the example programs to communicate with. The application is in the file Mbustest.exe and runs under Windows. It is installed with the Windows installation option.



Port: COM4:
Slave Addr.: 100
Register Type: 4xxxx (Holding registers read/write)
Start Address: 1
7D00
Type Conversion: [✓ Integer] Long Floating Point Discrete
32000
[✓] Hex
Read Success!

The Modbus Master Test Tool

This tool is a simple Modbus Master device, capable of reading or writing one value at a time from a Modbus slave device and displaying it as one of four data types, in decimal or hex where appropriate.

The correct serial port and Modbus slave address for the required device should be entered in the two combo boxes at the top. The ‘Configure’ button can be pressed to set up the serial port parameters and the Modbus Protocol Tick and Timeout. The register type and start address should then be entered and the returned data type selected. The ‘Hex’ box should be checked to display integer and long type read results in hexadecimal.

For a read operation, you should then press the Read button; the result should appear in the panel below the button. For a write operation, enter the value in the box below the Write button and then press the button. If successful, the tool will read back the value just written and display it in the panel below the Read button.

If there are any communications problems the Status line at the bottom will report them. The most common causes of problems will be having the wrong port or slave address selected.

File List

After installation, you should have the following files:

Dos

Library
Modbus.obj

Make file
mkmbhex.bat

Link template - used by mkmbhex.bat
mblinkf.mkh

Header file
Modbus.h

Example Source
Example1.c
Example2.c
Example3.c

Example Hex
Example1.hex
Example2.hex
Example3.hex

Windows

As for Dos, plus:

Software Dummy Instrument
Demoinst.exe

Modbus Master Test Tool
Mbustest.exe

Test Tool Support Files (in Windows System directory)
Mbus.vbx
Modrtu.dll
Bivbx11.dll

Licence Agreement & Limited Warranty for the KD485 Modbus Library

This document constitutes a legal agreement between 'you,' (the 'Licensee') , and Cooke Technologies Limited (CTL). If you do not agree to the terms of this agreement please return the unopened disk package, and all other components of the product (including the documentation with which it was bundled), to the place from which you purchased it, for a full refund. Your attention is particularly drawn to Clause 9 (Your undertakings). To protect its legal rights, CTL is not selling any of its intellectual property rights, copyrights, or other rights in the software. Instead, CTL are granting you the right, by means of a software licence, to use the software. CTL retain all title & rights not expressly granted to you.

Cooke Technologies Ltd Licence Agreement (KD485 Modbus Library)
1. Grant of Licence. In consideration for the payment of the licence fee - which is part of the price you have paid or agreed to pay for this product - and your agreement to abide by the conditions of this licence and the limited warranty, CTL will grant you the non-exclusive right to use and display this copy of the software on a single computer at a single location so long as you comply with the terms of this licence. CTL retain all title and rights not expressly granted to the licensee.

2. Ownership of Software. You own the magnetic or other physical media on which this software is recorded, but an express condition of this licence is that CTL retain title and ownership of the software recorded on the original disk copy (ies) and all subsequent copies of the software, regardless of the form or media in or on which the original and other copies may exist. This licence is not a sale of the original software or any copy.

3. Copy Restrictions. This software and the accompanying documentation is copyrighted. Unauthorised copying of the documentation or software, including that which has been modified or merged with other software, is expressly forbidden. However, you may make two (2) copies of the software solely for back-up purposes. You must reproduce and include the copyright message on the back-up copies. You may be held legally responsible for any copyright infringement caused by your failure to abide by this licence.

4. Use Restriction. The software is licensed for use only on one computer at a time. You may not distribute copies of the software or documentation to others. You may not modify, adapt, translate, reverse engineer, decompile, disassemble or create derivative work based on the software, nor may you modify, adapt, or create derivative work based on the documentation without prior written permission from CTL.

5. Use with KD485. The library may be used to produce real-time software for the KD485. There is no restriction on the number of systems in which it is used.

6. Transfer Restrictions. This software is licensed to the original purchaser. This licence may be transferred or assigned to a new user, only if they also agree, in writing, to be bound by the terms of this licence and limited warranty.

7. Termination. This licence is effective until terminated, which will happen automatically, without notice from CTL, if you fail to comply with any provision of this licence. Within 14 days of termination you must destroy all copies of the software and its associated documentation (including any copies, complete, partial or modified) and inform CTL, in writing, that you have done so. On termination you shall pay CTL all costs and expenses, including legal and other fees, incurred in respect of the software, this licence or otherwise. Termination, howsoever or whenever occasioned, shall be subject to any rights and remedies that CTL may have under this licence or in law.

8. Miscellaneous. This license is governed by, and construed in accordance with, the provisions of English Law.

9. Update Policy. CTL may offer information on enhancements or modifications to the software at such cost as may be notified to you. CTL shall charge for any software services requested by you, including upgrades, which are not covered by this licence.

10. Your undertakings. You undertake to supervise and control the use of the software in accordance with the terms of this licence. You also agree to include our copyright notice on all copies made of the software, including partial copies or modified copies.

11. Our liability. Subject to the provisions of Clause 9 above, the software and its accompanying documentation are provided 'as is' without warranty of any kind, and are specifically not suitable for life-critical applications. The entire risk as to the results and performance of the software rest with you. CTL are not liable to you for any loss or damage whatsoever, howsoever caused, arising directly or indirectly, in connection with this licence, the software, its use or otherwise, except that which it is unlawful to exclude. CTL expressly exclude liability for consequential loss or damage which may arise in respect of the software, its use, or in respect of other equipment or property, or for loss of profit, business, revenue, goodwill or anticipated savings. CTL warrant you only that the disk on which the software is supplied is free from defects in materials and faulty workmanship, under normal use, for a period of 90 days from the date of purchase. CTL's sole liability in respect of any such defect is expressly limited to the replacement of the software and the medium on which it is supplied, and is not applicable if the failure is the result of accident or abuse. In the event that any provision contained in this licence shall be held to be invalid for any reason, and CTL become liable for loss or damage that it would otherwise have been lawful to exclude, such liability shall be subject to other provisions limiting CTL's liability to the total cost of the software (exclusive of VAT). CTL do not exclude liability for death or personal injury to the extent that the same arises as a result of CTL's negligence, or that of its employees, agents or authorised representatives. Nothing in this Clause (11) affects your statutory rights.

12. General. CTL are under no obligation to you in respect of anything which, apart from this provision, may constitute breach of this licence arising by reason of force majeure, namely, circumstances beyond the control of CTL, including industrial action of any sort.

Failure by CTL to enforce any of the provisions above shall not be construed to be a waiver of CTL's rights under this licence, nor shall it in any way affect the validity of the whole or any part of this licence . CTL reserves all rights not expressly granted to you. If any of this provisions shall be determined invalid, unlawful or unenforceable to any extent, then such provision shall be severed from the remainder, and these shall continue to be valid to the fullest extent permitted by law.

Acknowledgement
YOU ACKNOWLEDGE THAT YOU HAVE READ THIS LICENCE AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. You also agree that this licence supersedes all proposals or prior agreements, oral or written, and any other communications between the parties regarding the subject matter of this licence.

Trademark Acknowledgements

References

KD485 User Manual, KK Systems
Modicon Modbus Protocol Reference Guide (PI-MBUS-300 Rev. E, 1993)