TECHNICAL NOTE

TN262

Modbus Protocol (Serial and TCP) for Rabbit-Based Systems

As defined by the Modbus Organization, Inc., “Modbus is an application-layer messaging protocol that provides client/server communication among boards connected on different types of buses or networks. Modbus is the *de facto* industrial serial standard that enables automation boards to communicate.”

Dynamic C provides the libraries and sample programs to allow you to create a Modbus link using Rabbit-based single board computers (SBCs) and RabbitCore modules. Although the sample programs were initially developed around Rabbit’s BL2600 SBCs, they can be adapted for other Rabbit-based SBCs and core modules.

The Rabbit-based systems will work on a Modbus network containing non-Rabbit based products.

1.0 System and Software Requirements

The Dynamic C Modbus software was developed using the specifications published by the Modbus Organization on their Web site at www.modbus.org/specs.php. While the Modbus software will in principle work with all Rabbit-based SBCs and RabbitCore modules, keep the following things in mind.

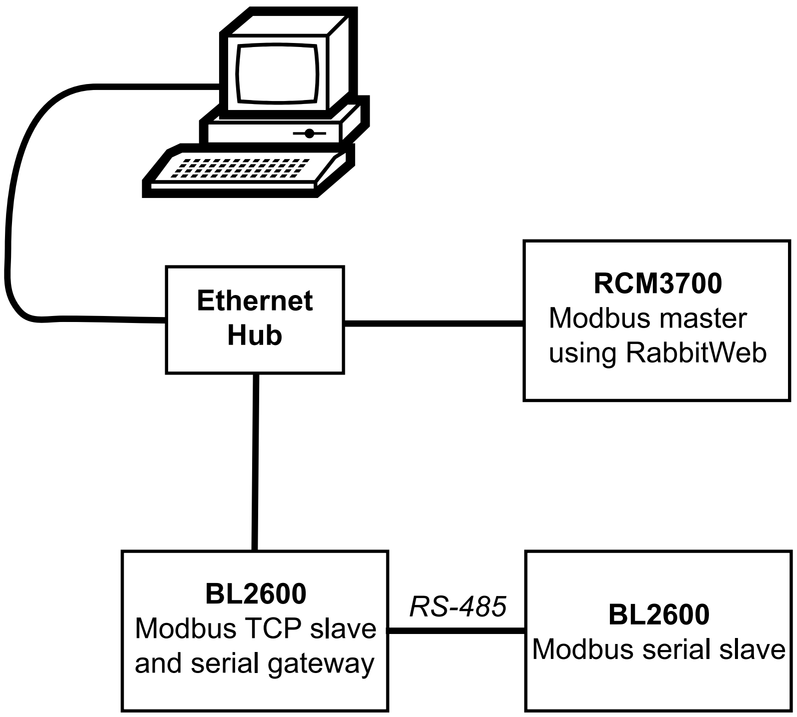
- To be used as a Modbus TCP master or Modbus slave and serial gateway, the SBC or RabbitCore module must have an Ethernet jack and must be connected to a customer-supplied Ethernet hub or router.

- You will have to make some small modifications to the Modbus Master sample program `Modbus_RabWeb.c` if you are using a Rabbit-based board other than the BL2600 for a Modbus slave. The edits requires the use of the Dynamic C RabbitWeb software.

2.0 Setup Instructions

Figure 1 illustrates the test configuration that was used to develop the samples and libraries.

Figure 1. Test Configuration Used to Develop Modbus Libraries


Ethernet Hub
RCM3700 Modbus master using RabbitWeb
BL2600 Modbus TCP slave and serial gateway
RS-485
BL2600 Modbus serial slave

While you are running sample programs on the BL2600s, the PC's Web browser is used to contact the RCM3700, which is running a RabbitWeb program specifically designed for the application. The Rabbit-Web program is discussed in Section 2.1.

The Web browser displays a page with four tabs, each representing a BL2600. The first tab also has an entry box for you to enter the IP address of the BL2600 used as the TCP slave and serial gateway. Each tab has a field for you to enter the Modbus address of the board for that tab. Default values are entered by the sample program running on the Modbus master. See Section 2.1.6 for further information on the browser interface.

2.1 Running the Sample Programs

All sample programs referenced in this document are located in the "\Samples\Modbus" folder where you installed Dynamic C. This section gives instructions for downloading the sample programs that will allow for the configuration shown in Figure 1. These sample programs are described in detail starting in Section 3.1.

2.1.1 Modbus Slaves

There are two types of Modbus slaves supported by the Modbus module: serial and gateway[1]. Both types are used in our example and each type requires a different application program. Follow these steps to download the appropriate sample program to the target board:

1. Connect the programming cable to the programming header on the Rabbit-based board. For the example in Figure 1, that would be a BL2600. Connect the other end of the programming cable to a free COM port on the PC running Dynamic C.
2. From Dynamic C, open

   ..\Samples\Modbus\Modbus_Serial_Slave.c. for a serial slave
   ..\Samples\Modbus\Modbus_Gateway_BL2600.c. for a gateway slave
3. Press F5 to compile and download the sample program to the Rabbit-based board.

No changes to the sample programs are needed if you are using BL2600s.

The default IP address of the Modbus gateway is 10.10.6.102. This IP address may be changed in `Modbus_Gateway_BL2600.c`. If you change the IP of the Modbus gateway there, you must also change it in the sample program that operates as the Modbus master. In addition to an IP address, the gateway slave has a Modbus address defined in `Modbus_Gateway_BL2600.c`. The default Modbus address for the gateway slave is 1. To change it, modify the definition of `MY_MODBUS_ADDRESS`.

Serial slaves also must have a unique Modbus address. The Modbus address for a serial slave is defined by the macro `MY_MODBUS_ADDRESS` in `Modbus_Serial_Slave.c`.

2.1.2 Modbus Master

The Modbus master in Figure 1 is the sample program `Modbus_RabWeb.c` running on an RCM3700. Using RabbitWeb, you can compile and download this source file to any RabbitCore module with Ethernet to create this Modbus master.

Follow the same three steps given above for slave devices in order to put `Modbus_RabWeb.c` on the target board. The default IP address for the Modbus master is 10.10.6.101. The default IP address for the Modbus gateway is also defined in `Modbus_RabWeb.c`.

1. For simplicity's sake, the slave that operates as a TCP/IP gateway and can also communicate serially with other Modbus slaves will be referred to as a "gateway slave." All other slaves are referred to by the interchangeable terms: "serial slave," "Modbus slave" or just plain "slave."

2.1.2.1 Modbus Master without BL2600s

The Modbus master used in this example is expecting its Modbus slaves, both gateway and serial, to be BL2600s. To use some other Rabbit-based board as a serial or gateway slave requires modification of `Modbus_RabWeb.c`.

Here are the editing steps to use `Modbus_RabWeb.c` with Rabbit-based slaves other than the BL2600.

1. `Modbus_RabWeb.c` is set up to communicate with two different slave boards, Type A and Type B. The I/O for both types of boards are set up for BL2600s in the sample program. If a BL2600 and another single-board computer, for example, a BL2100, is used, they will be different board types — the BL2600s are Type A boards and the BL2100s are Type B boards. Set the `DEV_TYPE_A` and the `DEV_TYPE_B` macros to reflect the I/O for the boards you are using.

```
#define DEV_TYPE_A 2      // Number of Type A boards
#define DEVADIO 16        // Device A max digital I/O
#define DEVAHCO 4         // Device A max high-current outputs
#define DEVADAC 4         // Device A max analog outputs
#define DEVAADC 8         // Device A max analog inputs

#define DEV_TYPE_B 2      // Number of Type B boards
#define DEVBDIO 16        // Device B max digital I/O
#define DEVBHCO 4         // Device B max high-current outputs
#define DEVBDAC 4         // Device B max analog outputs
#define DEVBADC 8         // Device B max analog inputs
```

NOTE: The same number of I/O must be specified for all the boards of that particular board type. For example, if one BL2600 does not use any analog outputs, but the other BL2600s do, you must specify “4” for the maximum number of analog outputs.

2. Change BL2600 in the following line to the model number of the Rabbit-based board you are using, e.g., BL2100. This line controls the tab labels displayed in the Web browser.

```
#web dev.tabs select("GATEWAY(A1)"=0,"BL2600(A2)","BL2600(B1)",
"BL2600(B2)")
```

3. Make the same types of changes in the remaining instances where “BL2600” is used, as in the following sample code comments.

```
switch ( dev.tabs )
{  // find the type of modbus device
   case 0:               // device type Gateway BL2600(A1)
   case 1:               // device type BL2600(A2)
   case 2:               // device type BL2600(B1)
   case 3:               // device type BL2600(B2)
```

2.1.3 Set IP Address of PC

Follow these instructions to set up your PC or notebook. Check with your administrator if you are unable to change the settings as described here since you may need administrator privileges. The instructions are specifically for Windows 2000, but the interface is similar for other versions of Windows.

> **TIP:** If you are using a PC that is already on a network, you will disconnect the PC from that network to run these sample programs. Write down the existing network settings before changing them so you can change them back.

1. Go to the control panel (Start > Settings > Control Panel), and then double-click the Network icon.
2. Select the network interface card used for the Ethernet interface you intend to use (e.g., TCP/IP Xircom Credit Card Network Adapter) and click on the “Properties” button. Depending on which version of Windows your PC is running, you may have to select the “Local Area Connection” first, and then click on the “Properties” button to bring up the Ethernet interface dialog. Then “Configure” your interface card for a “10Base-T Half-Duplex” or an “Auto-Negotiation” connection on the “Advanced” tab.

   > **NOTE:** Your network interface card will likely have a different name.

3. Now select the “IP Address” tab, and check “Specify an IP Address” or select TCP/IP and click on “Properties” to assign an IP address to your computer (this will disable “obtain an IP address automatically”):

   IP Address: 10.10.6.100
   Netmask: 255.255.255.0
   Default gateway: 10.10.6.1

4. Click <OK> or <Close> to exit the various dialog boxes.

2.1.4 Hardware Connections

Now you may connect the PC, Modbus master, and Modbus gateway to the Ethernet hub as shown in Figure 1. Finish by making a serial RS-485 connection of the Modbus slave to the Modbus gateway, and you will have your Modbus network running.

2.1.5 Summary of Default IP Addresses

- PC: 10.10.6.100
- Modbus Master: 10.10.6.101
- Modbus Gateway: 10.10.6.102

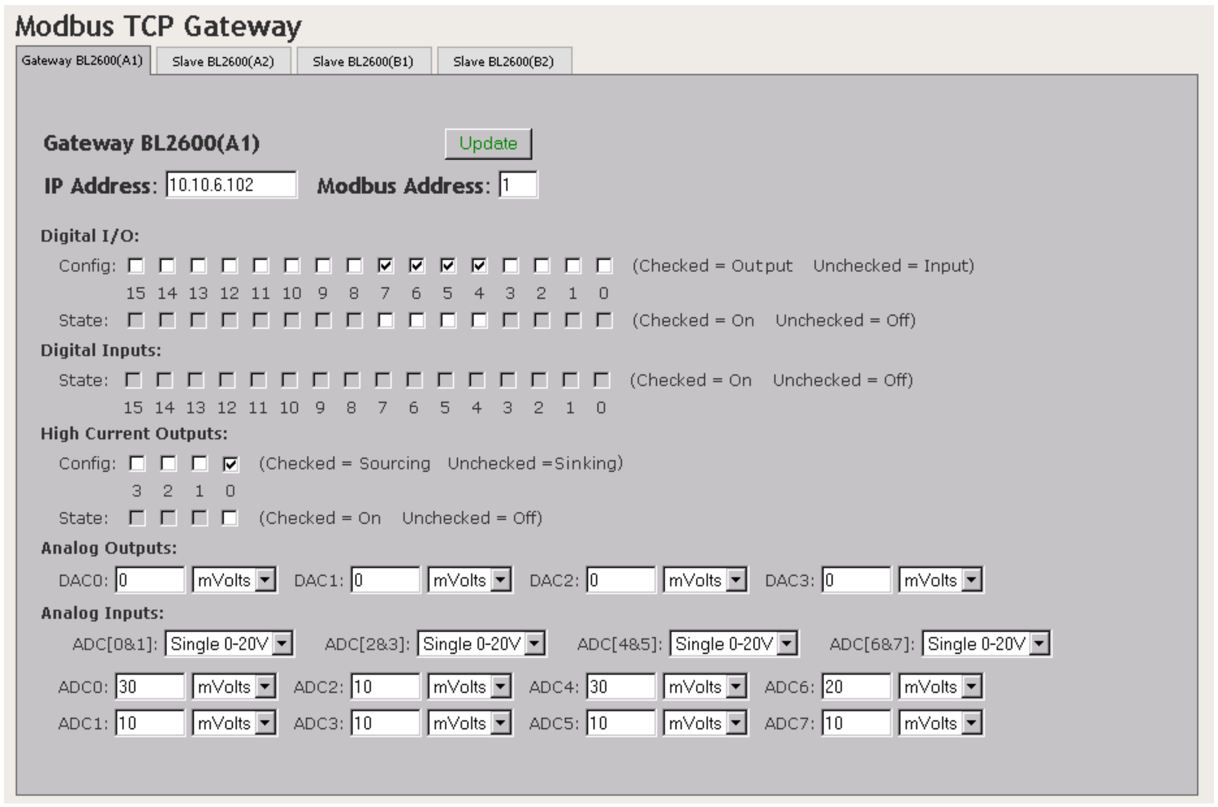
2.1.6 Using the Web Browser Interface

Enter 10.10.6.101 into your Web browser to access the Modbus master. If you changed the default IP address in `Modbus_RabWeb.c`, enter that IP address into the browser instead of 10.10.6.101.

The browser will display the page shown in Figure 2.

Use the tabs at the top of the screen to select the Modbus gateway or serial slave to configure.

Figure 2. Modbus Master RabbitWeb Page


Modbus TCP Gateway

Gateway BL2600(A1) | Slave BL2600(A2) | Slave BL2600(B1) | Slave BL2600(B2)

Gateway BL2600(A1) Update

IP Address: 10.10.6.102 **Modbus Address:** 1

Digital I/O:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Config:	☐	☐	☐	☐	☐	☐	☐	☐	☑	☑	☑	☑	☐	☐	☐	☐	(Checked = Output Unchecked = Input)
State:	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	(Checked = On Unchecked = Off)

Digital Inputs:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
State:	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	(Checked = On Unchecked = Off)

High Current Outputs:

	3	2	1	0	
Config:	☐	☐	☐	☑	(Checked = Sourcing Unchecked =Sinking)
State:	☐	☐	☐	☐	(Checked = On Unchecked = Off)

Analog Outputs:

DAC0: 0 mVolts DAC1: 0 mVolts DAC2: 0 mVolts DAC3: 0 mVolts

Analog Inputs:

ADC[0&1]: Single 0-20V ADC[2&3]: Single 0-20V ADC[4&5]: Single 0-20V ADC[6&7]: Single 0-20V

ADC0: 30 mVolts ADC2: 10 mVolts ADC4: 30 mVolts ADC6: 20 mVolts

ADC1: 10 mVolts ADC3: 10 mVolts ADC5: 10 mVolts ADC7: 10 mVolts

1. Enter the IP address (default 10.10.6.102) and the Modbus address (1) for the gateway slave.
2. Set the configurable I/O. Checked I/O are outputs, whose state can then be checked to be on by default or left unchecked to be off by default; unchecked I/O will be inputs.
3. The digital inputs are greyed out because no further configuration is available.
4. Set the configurable high-current outputs. Checked outputs will be sourcing; unchecked outputs will be sinking. Check their state to be on by default or leave the state unchecked for the output to remain off by default.

   NOTE: If you plan to switch a high-current output between sourcing and sinking operation, the high-current output must be turned "off" when changing its mode.

5. Set the analog outputs. They can be up to ±10 V DC or 4–20 mA.
6. Set the analog inputs. They can be up to 10 V DC single-ended voltage, up to ±10 V DC differential voltage, or 4–20 mA current.

Click the button labeled "Update" to send the configuration to the slave and to receive updated status information from the slave. You may then configure other serial slave boards by selecting other tabs.

3.1 Modbus Device Types

There are three types of Modbus devices supported by this software:

1. Modbus master
2. Modbus gateway slave
3. Modbus serial slave

The following sections discuss each of these Modbus devices in turn.

3.2 Modbus Master

The Dynamic C library `ModBus_Master.lib` implements the protocol for a Modbus master. The library is independant of the communication method. It is equally compatible with Ethernet and serial interfaces.

This library supports the Modbus function codes listed in Table 1.

Table 1. Modbus_Master.lib Support of Modbus Function Codes

Modbus Function Code	Description of Function Code (# of bits to change I/O state)	Corresponding Dynamic C Function
0x01	Read Coils (1 bit)	`MBM_ReadCoils`
0x03	Read Holding Registers (16 bit)	`MBM_ReadRegs`
0x04	Read Input Registers (16 bit)	`MBM_ReadInRegs`
0x05	Write Single Coil (1 bit)	`MBM_WriteCoil`
0x06	Write Single Register (16 bit)	`MBM_WriteReg`
0x0F	Write Multiple Coils (1 bit)	`MBM_WriteCoils`

3.2.1 Modbus Master Sample Programs

At the time of this writing, there are three Modbus master sample programs.

1. The Modbus master `Modbus_RabWeb.c` communicates using TCP over Ethernet. This Modbus master allows you to configure the I/O on the slave device.
2. The Modbus master `Modbus_Serial_Master.c` communicates over a serial connection. This sample program has limited functionality and is offered as a template for the user to create a Modbus master.
3. The Modbus master `Modbus_Master.exe` also communicates over a serial connection. It is meant to be used as a debugging utility. You will find it in a folder named “Modbus” located in the root directory where you installed Dynamic C. Look in the subfolder “Docs” for `Modbus_Master.pdf` for instructions on using this utility program.

3.2.2 Dynamic C API for Modbus Master

MBM_ReadCoils

```
int MBM_ReadCoils( int MB_address, int *Result, int Starting_Coil,
   int Nbr_of-Coils );
```

DESCRIPTION

Modbus function code = 0x01.

Reads the state of the specified coils, a.k.a., digital outputs. This function is not reentrant.

PARAMETERS

`MB_address`	Modbus address of the target board.
`Result`	Pointer to the starting address where to put the result; state of the coils: 1 = on 0 = off Each coil state will occupy one bit of the result with the first coil in bit 0, the next in bit 1, etc.
`Starting_Coil`	Starting coil number to read, relative to 1.
`Nbr_of-Coils`	Number of coils to read, maximum of 16.

RETURN VALUE

`MB_SUCCESS` = success
`MBM_INVALID_PARAMETER` = invalid parameter
`MBM_PACKET_ERROR` = packet error
`MB_BADADDRESS` = illegal channel

LIBRARY

`MODBUS_MASTER.LIB`

MBM_ReadRegs

```
int MBM_ReadRegs( int MB_address, int *Result, int Starting_Reg,
   int Nbr_of_Regs );
```

DESCRIPTION

Modbus function code = 0x03.

Reads the specified registers. This function is not reentrant.

PARAMETERS

`MB_address`	Modbus address of the target board
`Result`	Pointer to the starting address where to put the result
`Starting_Reg`	Starting register number, 1 relative to read
`Nbr_of-Regs`	Number of registers to read

RETURN VALUE

`MB_SUCCESS` = success
`MBM_INVALID_PARAMETER` = invalid parameter
`MBM_PACKET_ERROR` = packet error
`MB_BADADDRESS` = illegal channel

LIBRARY

`MODBUS_MASTER.LIB`

MBM_ReadInRegs

```
int MBM_ReadInRegs( int MB_address, int *Result, int Starting_Reg,
   int Nbr_of_Regs );
```

DESCRIPTION

Modbus function code = 0x04.

Reads the specified input registers. This function is not reentrant.

PARAMETERS

`MB_address`	Modbus address of the target board
`Result`	Starting address to put the results
`Starting_Reg`	Starting input register number, 1 relative, to read
`Nbr_of-Regs`	Number of registers to read

RETURN VALUE

`MB_SUCCESS` = success
`MBM_INVALID_PARAMETER` = invalid parameter
`MBM_PACKET_ERROR` = packet error
`MB_BADADDRESS` = illegal channel

LIBRARY

`MODBUS_MASTER.LIB`

`MBM_WriteCoil`

```
int MBM_WriteCoil( int MB_address, int CoilNbr, int CoilState );
```

DESCRIPTION

Modbus function code = 0x05.

This function writes a value to a single coil. This function is not reentrant.

PARAMETERS

`MB_address`	Modbus address of the target board
`CoilNbr`	Coil number
`CoilState`	Coil state

RETURN VALUE

`MB_SUCCESS` = success
`MBM_INVALID_PARAMETER` = invalid parameter
`MBM_PACKET_ERROR` = packet error
`MB_BADADDRESS` = illegal channel

LIBRARY

`MODBUS_MASTER.LIB`

MBM_WriteReg

```
int MBM_WriteReg( int MB_address, int RegNbr, int RegData );
```

DESCRIPTION

Modbus function code = 0x06.

This function writes a value to a single register. This function is not reentrant.

PARAMETERS

MB_address	Modbus address of the target board
RegNbr	Register number
RegData	Register data

RETURN VALUE

`MB_SUCCESS` = success
`MBM_INVALID_PARAMETER` = invalid parameter
`MBM_PACKET_ERROR` = packet error
`MB_BADADDRESS` = illegal channel

LIBRARY

`MODBUS_MASTER.LIB`

MBM_WriteCoils

```
int MBM_WriteCoils( int MB_address, int StartCoilNbr, int NbrCoils,
    int CoilStates );
```

DESCRIPTION

Modbus function code = 0x0F.

This function writes values to coils. This function is not reentrant.

PARAMETERS

`MB_address`	Modbus address of the target board
`StartCoilNbr`	Starting coil number
`NbrCoils`	Number of coils
`CoilStates`	Coil states, max. 16, with lowest coil number value in bit 0

RETURN VALUE

`MB_SUCCESS` = success
`MBM_INVALID_PARAMETER` = invalid parameter
`MBM_PACKET_ERROR` = packet error
`MB_BADADDRESS` = illegal channel

LIBRARY

`MODBUS_MASTER.LIB`

3.3 Modbus Slaves

As mentioned previously, there are two types of Modbus slaves supported by the software: serial and gateway. Both types need `ModBus_Slave.lib` and a board-specific library such as `Modbus_Slave_BL26xx.lib`. These two libraries implement the Modbus protocol for a slave.

The gateway slave requires the use of `MODBUS_SLAVE_TCP.LIB` in addition to `ModBus_Slave.lib` and a board-specific library such as `Modbus_Slave_BL26xx.lib`.

For slaves that are not BL2600s, a different board-specific library is required. If you have an LP3500 that you want to use as a serial slave, the library `Modbus_Slave_LP35xx.lib` is provided for you. If you want to use some other Rabbit-based board as a slave, you must provide a board-specific library similar to `Modbus_Slave_BL26xx.lib` and `Modbus_Slave_LP35xx.lib`.

Dynamic C libraries that support the Modbus protocol are in the folder "..\Lib\..\Modbus" where you installed Dynamic C.

The following table summarizes the above text:

Table 2. Modbus Slave Library Requirements

Modbus Slave Type	Dynamic C Library
Serial Slave	`ModBus_Slave.lib` Board-specific library (e.g., `Modbus_Slave_BL26xx.lib`)

Gateway Slave	`ModBus_Slave.lib`
	`ModBus_Slave_TCP.lib`
	Board-specific library (e.g., `Modbus_Slave_BL26xx.lib`)

The Modbus commands listed in Table 3 are supported by both `MODBUS_SLAVE.LIB` and `MODBUS_SLAVE_TCP.LIB`. The most significant byte is transmitted/received first (Big Endian) for 16-bit (2-byte) data.

Table 3. Modbus Commands Supported by Slaves

Modbus Function Code	Description of Function Code (# of bits to change I/O state)	Corresponding Dynamic C Function from Board-Specific Library
0x01	Read Coils (1 bit)	`mbsDigOutRd()`
0x02	Read Input Status (1 bit)	`mbsDigIn()`
0x03	Read Holding Registers (16 bit)	`mbsRegOutRd()`
0x04	Read Input Registers (16 bit)	`mbsRegIn()`
0x05	Write Single Coil (1 bit)	`mbsDigOut()`
0x06	Write Single Register (16 bit)	`mbsRegOut()`
0x0F	Write Multiple Coils (1 bit)	`mbsDigOut()`
0x10	Write Multiple Registers (16 bit)	`mbsRegOut()`
0x16	Mask Write Register (16 bit)	`mbsRegOut()` and `mbsRegOutRd()`
0x17	Read/Write Multiple Registers (16 bit)	`mbsRegOut()` and `mbsRegIn()`

3.3.1 Modbus Message Structure

Before looking at the libraries in more detail, we will look at the Modbus message. An understanding of its structure, for both serial and TCP/IP communication, will help make sense of the tasks performed by the libraries.

The format of a *Modbus Protocol Data Unit* (PDU) is:

Function Code	Data

Data values are handled most significant byte first.

The format of a *Modbus Application Data Unit* (ADU) is:

Additional Address	**PDU**	Error Check CRC

For a serial interface the *Additional Address* is a single byte with the target board’s Modbus address. The *CRC* is two bytes and is stored low byte first.

The format of a *Modbus TCP Application Protocol* (MBAP) packet is:

MBAP Header	**PDU**

The contents of the *MBAP Header* are:

- Transaction Identifier (2 bytes): A value that is incremented with each transaction. It is used to identify the response to the current transaction. The libraries do not currently use this value, although it is generated and incremented with each transaction.
- Protocol Identifier (2 bytes): Always 0.
- Length (2 bytes): Number of bytes following this element
- Unit Identifier (1 byte): Modbus board address.

3.3.2 MODBUS_SLAVE_TCP.LIB

This library receives the Modbus/TCP packet from a Modbus master, removes the MBAP header, and does one of the following steps.

1. If the message is for "this board," send the message to the `msExec()` function in the `MODBUS_SLAVE.LIB` library.
2. If the message is for a downstream unit, create and send the ADU, wait for the response, and send the response to the Modbus master.

3.3.2.1 Configuration Macros

The library `MODBUS_SLAVE_TCP.LIB` uses the following macros.

`INACTIVE_PERIOD`
Period of inactivity in seconds (default 5), before sending a `keepalive`, or 0 to turn off `keepalive`.

`KEEPALIVE_WAITTIME`
Number of seconds (default 3) to wait between `keepalives` after the first `keepalive` was sent.

`KEEPALIVE_NUMRETRYS`
Number of retries (default 3).

`MB_MAX_SKT`
Maximum number of socket connections (default 1).

`MODBUS_GATEWAY`
`#define` this macro only if this board is a gateway.

`TCPCONFIG`
Defining this macro to “0” means that the application does not use the TCP/IP configuration macro definitions from `TCP_CONFIG.LIB`.

`USE_ETHERNET`
This macro must be defined to “1” for stack support of the Ethernet interface.

`IFCONFIG_ETH0`
This macro is defined as follows to set the default IP address and netmask for the gateway slave before bringing up the interface.

```
IFCONFIG_ETH0 \
   IFS_IPADDR,aton ("10.10.6.101"), \
   IFS_NETMASK,aton("255.255.255.0"), \
   IFS_UP
```

`MODBUS_DEBUG`
This macro has two options:
- `nodebug`, the default, disallows library debugging
- `debug` allows library debugging

`MODBUS_DEBUG_PRINT`
This macro is a bit flag, defined as follows:

- All bits = 0 (default) = no printing of messages
- bit 0 = `1` = state machine transitions
- bit 1 *not implemented*.
- bit 2 = `1` = print TCP packets
- bit 3 = `1` = print serial packets

3.3.2.2 API Functions

`MODBUS_SLAVE_TCP.LIB` contains the following two API functions: `MODBUS_TCP_Init()` and `MODBUS_TCP_tick()`.

MODBUS_TCP_Init

```
void MODBUS_TCP_Init( unsigned wAddr, unsigned wPort );
```

DESCRIPTION

This function must be called ***one*** time only. It initializes the Modbus TCP system. It does ***not*** initialize the TCP/IP connection.

PARAMETERS

wAddr	The Modbus slave address
wPort	Modbus TCP port number; the standard Modbus port is 502

MODBUS_TCP_tick

```
void MODBUS_TCP_tick( void );
```

DESCRIPTION

This function call is a process ModBus TCP state handler. It must be called repeatedly, usually within a loop, by the program in order to ensure that the TCP/IP command packets get serviced properly. The function call causes `tcp_tick()` to execute, and services one socket each time it is called.

3.3.3 MODBUS_SLAVE.LIB

The library `MODBUS_SLAVE.LIB` parses the Modbus PDU and calls the appropriate function in the board-specific library. The API functions `MODBUS_Serial_tick()` and `MODBUS_CRC()` are defined in this library. None of the other functions in `MODBUS_SLAVE.LIB` are meant to be accessed directly by a slave application program.

MODBUS_Serial_tick

```
void MODBUS_Serial_tick( void );
```

DESCRIPTION

Checks for a command from a Modbus master. This function is called from a Modbus slave that is connected to a Modbus master or gateway slave via a serial port. If there is no such serial connection, this function is not needed.

This function requires the function calls `MODBUS_Serial_Rx()` and `MODBUS_Serial_Tx()` from the `MODBUS_SLAVE_BL26xx.LIB` library.

MODBUS_CRC

```
unsigned MODBUS_CRC( unsigned char *pcMess, unsigned wLen );
```

DESCRIPTION

Alternate cyclical redundancy check (CRC) calculation. To use this alternate CRC function you must insert:

```
#define USE_MODBUS_CRC
```

before the #use directives for the Modbus libraries.

PARAMETERS

pcMess	Address of bytes for CRC calculation
wLen	Number of bytes in paraameter1

RETURN VALUE

CRC value

3.3.4 Board-Specific Libraries

Each Modbus slave device must have an associated board-specific library that handles the I/O operations.

At the time of this writing, two board-specific libraries are available: `MODBUS_SLAVE_BL26xx.LIB` and `MODBUS_SLAVE_LP35xx.LIB`. As the names imply, these libraries are for the BL2600 series and LP3500 single-board computers.Taken together, these two libraries present an excellent template for creating board-specific Modbus libraries for other Rabbit-based boards.

The following functions must be defined in the board-specific library regardless of whether they are actually used.

- `mbsStart()` is called whenever a packet is received.
- `mbsDone()` is called whenever a packet is finished.

The following function descriptions are from `MODBUS_SLAVE_BL26xx.LIB`.

`mbsDigOutRd`

`int mbsDigOutRd( unsigned OutputNbr, int *pnState );`

DESCRIPTION

Modbus function code = 0x01.

Reads the specified output. This is slightly different than `mbsDigIn()` in that this function call returns a “1” if the output is on (low). It essentially returns the opposite of `mbsDigIn()`.

This function is not reentrant.

PARAMETERS

`OutputNbr`	Output number: 0..15
`pnState`	Pointer to destination variable

RETURN VALUE

`MB_SUCCESS` = success
`MB_BADADDR` = illegal channel
`MB_DEVNOTSET` = I/O not set as output

LIBRARY

`MODBUS_SLAVE_BL26XX.LIB`

mbsDigIn

```
int mbsDigIn( unsigned InputNbr, int *pnState );
```

DESCRIPTION

ModBus function code = 0x02.

Reads the specified input. This function is not reentrant.

PARAMETERS

`InputNbr`	Input number: 0..31 Inputs 0..15 are the DIO signals Inputs16..31 are DIN16..31
`pnState`	Pointer to destination variable; a “1” is returned if the input is high

RETURN VALUE

`MB_SUCCESS` = success
`MB_BADADDR` = illegal channel
`MB_DEVNOTSET` = I/O not set as input

LIBRARY

`MODBUS_SLAVE_BL26XX.LIB`

mbsRegOutRd

```
int mbsRegOutRd( unsigned OutRegNbr, unsigned *pwValue );
```

DESCRIPTION

ModBus function code = 0x03.

Reads an 8-bit output register. This function is not reentrant.

PARAMETERS

`OutRegNbr`	Register number: 0 = DIO 0..7 — read state of pins and invert (0 V = 1) 1 = DIO 8..15 — read state of pins and invert (0 V = 1) 2 = HOUT 0..3 — return state of shadow register Special registers See `mbsRegIn()`
`pwValue`	Pointer to destination variable; for each bit: 0 = output is off 1 = output is on

RETURN VALUE

`MB_SUCCESS` = success
`MB_BADADDR` = illegal channel
`MB_DEVNOTSET` = I/O not set as output

LIBRARY

`MODBUS_SLAVE_BL26xx.LIB`

mbsRegIn

`int mbsRegIn( unsigned InRegNbr, unsigned *pwValue );`

DESCRIPTION

ModBus function code = 0x04.

Reads an input register. This function is not reentrant.

PARAMETERS

`InRegNbr`	Register number: 0 = DIO 0..7 1 = DIO 8..15 2 = IN 16..23 3 = IN 24..31 Special registers: 1000 = DIO 0..15 configuration; see `digOutConfig()` 1001 = HOUT 0..3 configuration; see `digHoutConfig()` 3nnx = analog input, where nn = A/D channel 3nn0, 3nn1 = floating point (volts) 3nn1 = returns the MS word 3nn2 = Read the A/D converter and return the integer (millivolts) 3003 = Read the A/D converter and return the integer raw value See `mbsRegOut()` for gain code storage
`pwValue`	Pointer to destination variable

RETURN VALUE

`MB_SUCCESS` = success
`MB_BADADDR` = illegal channel
`MB_DEVNOTSET` = I/O not set as input

LIBRARY

`MODBUS_SLAVE_BL26xx.LIB`

mbsDigOut

```
int mbsdigOut( unsigned OutputNbr, int state );
```

DESCRIPTION

ModBus command = 0x05, 0x0F.

Turns the specified output on or off. This function is not reentrant.

PARAMETERS

`OutputNbr`	Output channel number: 0 ≤ channel ≤ 15: DIO 00..15 16 ≤ channel ≤19: HOUT 0..3 (high-current)
`state`	Output state: 0 = turn output off 1 = turn output on 0 ≤ channel ≤ 15 Connect the load to GND 16 ≤ channel ≤ 19 Sinking - connect the load to GND Sourcing - connect the load to +V

RETURN VALUE

`MB_SUCCESS` = success
`MB_BADADDR` = illegal channel
`MB_BADDATA` = illegal data value
`MB_DEVNOTSET` = I/O not set as output

LIBRARY

`MODBUS_SLAVE_BL26xx.LIB`

mbsRegOut

```
int mbsRegOut( unsigned OutRegNbr, unsigned wValue );
```

DESCRIPTION

Modbus function codes = 0x06, 0x10, 0x16 and 0x17.

Writes to an I/O register. This function is not reentrant.

PARAMETERS

`OutRegNbr` Register number:

- 0 = DIO 0..7
- 1 = DIO 8..15
- 2 = HOUT 0..3

Special Registers:

- 1000 = DIO 0..15 configuration — see `digOutConfig()`
- 1001 = HOUT 0..3 configuration — see `digHoutConfig()`
- 2nnx = Analog output where nn = D/A converter channel no.
  - 2nn0, 2nn1 = floating point volts; see `anaOutVolts()`
  - 2nn2 = integer millivolts; uses `anaOutVolts()`
  - 2nn3 = integer raw value; see `anaOut()`
  - 2nn9 = urns on D/A power: see `anaOutPwr()`
- 3nnx = A/D converter input where nn = A/D converter channel
  - 3nn8 = integer operating mode — see `anaInConfig()`
    - nn = channel pair 0–3
  - 3nn9 = integer range code used for `mbsRegIn()` 0–7
- 1999 = miscellaneous configuration bits

`wValue` Register value (each bit) for DIO 0..15

- 0 = turn output off
- 1 = turn output on

Register values for HOUT 0..3 (4 bits per output)

- HOUT 0 = bits 0..3, ...
- 0 = both transistors off = tri-state
- 1 = source (upper transistor on)
- 2 = sink (lower transistor on)
- All other values are illegal.

RETURN VALUE

`MB_SUCCESS` = success
`MB_BADADDR` = illegal channel
`MB_DEVNOTSET` = I/O not set as output

LIBRARY

`MODBUS_SLAVE_BL26xx.LIB`

The MODBUS_SLAVE_BL26xx.LIB library also contains functions for communicating with "downstream" Modbus slaves.

MODBUS_Serial_Init

int MODBUS_Serial_Init();

DESCRIPTION

Initializes the serial port for RS-485 operation. This function call requires the following macros.

MODBUS_BAUD[1]: baud rate
MODBUS_PORT: serial port on the slave device; e.g., on the BL2600 it is E.
SERIAL_MODE: mode to configure the serial port selected by MODBUS_PORT; e.g., on the BL2600 it is 1 or 3.

See the User's Manual for your board to determine the available serial ports and the mode that will configure the selected serial port for RS-485 communication.

This function calculates Serial_timeout, which is used by MODBUS_Serial_Rx() as the timeout between bytes once a byte has been received; the timeout is 5 byte times or 2 ms, whichever is greater.

This function is called by MODBUS_TCP_Init() if the macro MODBUS_GATEWAY is #defined.

This function is not reentrant.

RETURN VALUE

MB_SUCCESS = success
MB_BADDATA = if illegal SERIAL_MODE

LIBRARY

MODBUS_SLAVE_BL26xx.LIB

1. This macro was renamed from RS485_BAUD.

MODBUS_Serial_Tx

```
int MODBUS_Serial_Tx( char *Packet, int ByteCount );
```

DESCRIPTION

Transmits a Modbus packet to a “downstream” board. Calculates CRC and appends it to packet.

This function is not reentrant.

PARAMETERS

`Packet`	Pointer to packet. The packet must have a two-byte pad at the end for inclusion of the CRC word
`ByteCount`	Number of bytes in the packet

RETURN VALUE

`MB_SUCCESS` = success

LIBRARY

`MODBUS_SLAVE_BL26xx.LIB`

MODBUS_Serial_Rx

```
int MODBUS_Serial_Rx( char *DataAddress );
```

DESCRIPTION

Receives the response from the Modbus slave. The function stores the bytes in the global array `acMSReply` and uses the global variable `Serial_timeout`. It is the responsibility of the caller to handle a timeout if required.

This function is not reentrant.

PARAMETER

`DataAddress`	Address to put the data

RETURN VALUE

`0` = no message
`>0` = number of bytes with valid CRC
`MB_CRC_ERROR` = invalid CRC

LIBRARY

`MODBUS_SLAVE_BL26xx.LIB`