Quick Start for Porting Libmodbus

V1.00.001

Publication Release Date: Sep. 2020

Support Chips:

W55FA Series

Support Platforms:

Non-OS_Keil

The information in this document is subject to change without notice.

The Nuvoton Technology Corp. shall not be liable for technical or editorial errors or omissions contained herein; nor for incidental or consequential damages resulting from the furnishing, performance, or use of this material.

Table of Contents

1. Introduction...4

1.1. How to work Linux under N9H30..4

1.2. Command vs API function 5

1.3. History 5

1.4. Changed Code..5

2. Revision History...10

1. Introduction

1.1. How to work Linux under N9H30.

Libmodbus is a modbus library for Linux, Mac OS X, and Win32. It is a popular modbus system follows modbus specification. The latest version is 3.1.6 and is maintained.

Current libmodbus supports the standard command of modbus protocol. About the standard command, please refer to the document https://modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf. For more details. If user sends the non-standard command, feedback will exports the error meesages.

For embedded Linux under N9H30, please follow the document “N9H30 Linux BSP User Manual EN.pdf” to set the RS485 hardware for UART6. After building the binary image file “n9h30image”, please run the following command to test the libmodbus sample for PZEM-003

1. “tar zxvf libmodbus-3.1.6_nuvoton.tar.gz”
2. “./configure --host=arm-linux --enable-static --prefix=/opt/libmodbus/install”
3. “make”
4. “make install”
5. Copy the static library "libmodbus.a" from the path /opt/libmodbus/install/lib into the subfolder tests”,
6. Build the tests "demo_pzem.c" within the subfolder tests, by using the following command "arm-linux-gcc demo_pzem.c -o demo_pzem -I /opt/libmodbus/install/include/modbus -L . -lmodbus" to obtain the binary file "demo_pzem".
7. Run "./demo_pzem" under N9H30 board.

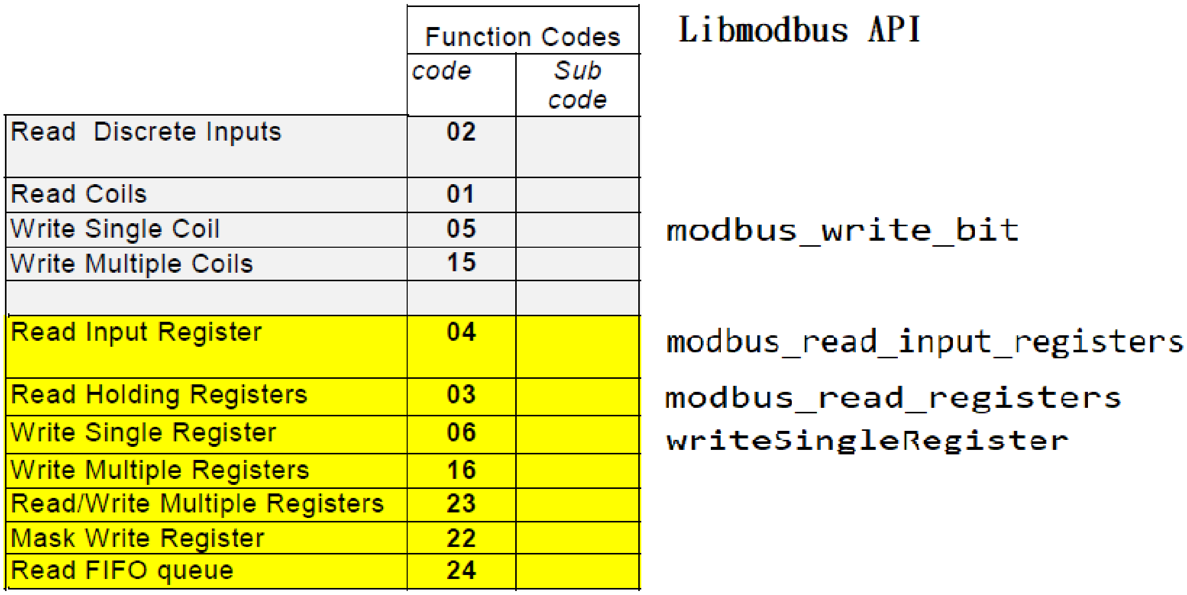
If user would like to run the thermal sensor, please do the following.

8. Build the tests "demo_thermal.c" within the subfolder tests, by using the following command "arm-linux-gcc demo_thermal.c -o demo_thermal -I /opt/libmodbus/install/include/modbus -L . -lmodbus" to obtain the binary file "demo_thermal".
9. Run "./demo_thermal" under N9H30 board.

Within the sample code, there is one statement “modbus_set_debug”, if user would like to see the modbus status, please set the statement “modbus_set_debug(ctx,TRUE)”, then there are some modbus messages to be displayed. If user does not understand the Modbus status, please set “modbus_set_debug(ctx, FALSE)”.

1.2. Command vs API function

Currently the sample code of the files demo_thermal.c and demo_pzem.c uses some APIs of libmodbus as follows. Please refer to the document in the website https://modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf



	Function Codes		Libmodbus API
	code	Sub code	
Read Discrete Inputs	02		
Read Coils	01		
Write Single Coil	05		modbus_write_bit
Write Multiple Coils	15		
Read Input Register	04		modbus_read_input_registers
Read Holding Registers	03		modbus_read_registers
Write Single Register	06		writeSingleRegister
Write Multiple Registers	16		
Read/Write Multiple Registers	23		
Mask Write Register	22		
Read FIFO queue	24		

1.3. History

In the version, we change the code in order to support the vendor command for Modbus, and fix the issue of RS485 for N9H30.

1.4. Changed Code.

We adds two files demo_pezm,c and demo_thermal.c, and change the both files Modbus_rtu.c and Modbus.c. Within the file modbus-rtu.c, we add some statements within the function "modbus_rtu_set_serial_mode" as follows, in order to fix the failure of RS486 for N9H30.

```
        if (mode == MODBUS_RTU_RS485) {
            // Get
            if (ioctl(ctx->s, TIOCGRS485, &rs485conf) < 0) {
                return -1;
            }
            // Set
//printf("Setting the para\n");
            rs485conf.flags |= SER_RS485_ENABLED;
#if 1
           rs485conf.flags |= SER_RS485_RTS_ON_SEND;
            rs485conf.flags &= ~(SER_RS485_RTS_AFTER_SEND);
           rs485conf.delay_rts_after_send = 0x80;
#endif
            if (ioctl(ctx->s, TIOCSRS485, &rs485conf) < 0) {
                return -1;
            }

            ctx_rtu->serial_mode = MODBUS_RTU_RS485;
            return 0;
        } else if (mode == MODBUS_RTU_RS232) {
```

PZEM-003 has some vendor commands that libmodbus does not support, so we need to add the function "modbus_vendor_access" within the file modbus.c as follows

```
// Ray added for vendor command
int modbus_vendor_access(modbus_t *ctx, uint8_t *req, uint8_t req_length, uint8_t *dest, int8_t diff)
{
    int rc;
    uint8_t rsp[MAX_MESSAGE_LENGTH];

    rc = send_msg(ctx, req, req_length);
    if (rc > 0) {
        int i;

        rc = _modbus_vendor_receive_msg(ctx, rsp, MSG_CONFIRMATION,req_length+2+diff);
        if (rc == -1)
            return -1;

       for (i = 0; i < rc; i++) {
            /* shift reg hi_byte to temp OR with lo_byte */
            dest[i] = rsp[i];
          }
    }

    return rc;
}

int _modbus_vendor_receive_msg(modbus_t *ctx, uint8_t *msg, msg_type_t msg_type, uint8_t length)
{
    int rc;
    fd_set rset;
    struct timeval tv;
struct timeval *p_tv;
int length_to_read;
int msg_length = 0;


if (ctx->debug) {
        if (msg_type == MSG_INDICATION) {
                printf("Waiting for an indication...\n");
        } else {
                printf("Waiting for a confirmation...\n");
        }
}

/* Add a file descriptor to the set */
FD_ZERO(&rset);
FD_SET(ctx->s, &rset);


length_to_read = length;

if (msg_type == MSG_INDICATION) {
        /* Wait for a message, we don't know when the message will be
          * received */
        if (ctx->indication_timeout.tv_sec == 0 && ctx->indication_timeout.tv_usec == 0) {
                /* By default, the indication timeout isn't set */
                p_tv = NULL;
        } else {
                /* Wait for an indication (name of a received request by a server, see schema) */
                tv.tv_sec = ctx->indication_timeout.tv_sec;
                tv.tv_usec = ctx->indication_timeout.tv_usec;
                p_tv = &tv;
        }
} else {
        tv.tv_sec = ctx->response_timeout.tv_sec;
        tv.tv_usec = ctx->response_timeout.tv_usec;
        p_tv = &tv;
}

while (length_to_read != 0) {
        rc = ctx->backend->select(ctx, &rset, p_tv, length_to_read);
        if (rc == -1) {
                _error_print(ctx, "select");
                if (ctx->error_recovery & MODBUS_ERROR_RECOVERY_LINK) {
                        int saved_errno = errno;

                        if (errno == ETIMEDOUT) {
                                _sleep_response_timeout(ctx);
                                modbus_flush(ctx);
                        } else if (errno == EBADF) {
                                modbus_close(ctx);
                                modbus_connect(ctx);
                        }
                        errno = saved_errno;
        }
        return -1;
    }

    rc = ctx->backend->recv(ctx, msg + msg_length, length_to_read);
    if (rc == 0) {
        errno = ECONNRESET;
        rc = -1;
    }

    if (rc == -1) {
        _error_print(ctx, "read");
        if ((ctx->error_recovery & MODBUS_ERROR_RECOVERY_LINK) &&
            (errno == ECONNRESET || errno == ECONNREFUSED ||
             errno == EBADF)) {
            int saved_errno = errno;
            modbus_close(ctx);
            modbus_connect(ctx);
            /* Could be removed by previous calls */
            errno = saved_errno;
        }
        return -1;
    }

    /* Display the hex code of each character received */
    if (ctx->debug) {
        int i;
        for (i=0; i < rc; i++)
            printf("<%.2X>", msg[msg_length + i]);
    }

    /* Sums bytes received */
    msg_length += rc;
    /* Computes remaining bytes */
    length_to_read -= rc;

    if (length_to_read > 0 &&
        (ctx->byte_timeout.tv_sec > 0 || ctx->byte_timeout.tv_usec > 0)) {
        /* If there is no character in the buffer, the allowed timeout
           interval between two consecutive bytes is defined by
           byte_timeout */
        tv.tv_sec = ctx->byte_timeout.tv_sec;
        tv.tv_usec = ctx->byte_timeout.tv_usec;
        p_tv = &tv;
    }
    /* else timeout isn't set again, the full response must be read before
       expiration of response timeout (for CONFIRMATION only) */
}

if (ctx->debug)
    printf("\n");

return (msg_length-2);
```

```
}
```

2. Revision History

Version	Date	Description
V1.00.001	Sep. 30, 2020	● Created

Important Notice

Nuvoton products are not designed, intended, authorized or warranted for use as components in equipment or systems intended for surgical implantation, atomic energy control instruments, aircraft or spacecraft instruments, transportation instruments, traffic signal instruments, combustion control instruments, or for any other applications intended to support or sustain life. Furthermore, Nuvoton products are not intended for applications whereby failure could result or lead to personal injury, death or severe property or environmental damage.

Nuvoton customers using or selling these products for such applications do so at their own risk and agree to fully indemnify Nuvoton for any damages resulting from their improper use or sales.