MB9BFx10_AN706-00074-1v0-E

32-BIT MICROCONTROLLER
FM Family Application note

MODBUS TCP on FM Family Microcontrollers

Revision History

Rev	Date	Remark
0.1	Aug. 16, 2013	CNo, First Edition

This document contains 19 pages.

Table of Contents

Table of Contents 3
Target products 4
1. Preface 5
2. Introduction to MODBUS 6
1.1 Some Background Information 6
1.2 The MODBUS Technology 7
1.3 MODBUS data model 9
1.3.1 Number format 9
1.3.2 Register mapping 9
2 FreeMODBUS on FM3 11
2.1 FreeMODBUS 11
2.2 FreeMODBUS on FM3 11
3 How to Use the MODBUS TCP Demonstration on FM3 15
3.1 Modpoll 15
3.2 Further PC master software options 16
4 Final thoughts regarding MODBUS TCP 17
5 Additional Information 18

Target products

This application note is described about below products;

(TYPE2)

Series	Product Number (not included Package suffix)
MB9BF210	MB9BF216, MB9BF217, MB9BF218
MB9BF610	MB9BF616, MB9BF617, MB9BF618
MB9BFD10	MB9BFD16, MB9BFD17, MB9BFD18

1. Preface

Spansion's microcontroller line FM Family consists of more than 600 derivatives. The high performance line is designed with industrial automation requirements in mind.

Besides controlling sensors and actuators, industrial automation systems need standard communication interfaces to ensure interoperability of equipment from different vendors. FM Family controllers support many of those interfaces, including UART, LIN, CAN, USB and Ethernet.

Since the 2000s, Ethernet based automation protocols began to supersede the older serial or CAN based fieldbus protocols as it provides high bandwidth, high noise immunity, and long cable lengths.

This application note explains how one of the most prevalent Ethernet based automation protocols, called MODBUS TCP, can be used on the Spansion FM3 microcontroller MB9BFD18T.

The software example is based on the free open-source software package FreeMODBUS and uses the free open-source TCP/IP stack LwIP (lightweight IP) underneath.

The project was developed on the starterkit SK-FM3-176PMC-FA but can easily be adapted to other hardware platforms with FM3 or FM4 microcontrollers that feature an Ethernet port, as MODBUS TCP does not depend on any special hardware features.

2. Introduction to MODBUS

THIS CHAPTER EXPLAINS THE MOST IMPORTANT PROPERTIES OF MODBUS

1.1 Some Background Information

MODBUS TCP is based on the MODBUS fieldbus, and was published in 1979 by Modicon, now Schneider Electric. It is a relatively simple industrial communication protocol to connect Modicon programmable logic controllers (PLCs) with compatible automation-related equipment like sensors and actuators. "*It is a de facto standard, truly open and the most widely used network protocol in the industrial manufacturing environment. It has been implemented by hundreds of vendors on thousands of different devices to transfer discrete/analog I/O and register data between control devices. It's a lingua franca or common denominator between different manufacturers. One report called it the "de facto standard in multi-vendor integration". Industry analysts have reported over 7 million Modbus nodes in North America and Europe alone.*"[1]

It is free from royalties and the specification is available free of charge. However to use the official, trademarked logo, membership of the Modbus Organization is compulsory.

Although originally designed for asynchronous serial links like EIA/TIA-232 and EIA/TIA-485 (MODBUS ASCII/RTU[2]), being an application protocol on OSI layer 7 made it independent from the actual physical media. Subsequently it has been adapted to HDLC on optical fiber (MODBUS+) and TCP/IP (MODBUS TCP).

In 2002 the MODBUS TCP specification was submitted to the IETF[3], which makes it an official Internet standard that uses the well-known TCP port 502.

There is a vendor association, called *Modbus Organization*, which seeks to promote the MODBUS protocol and to drive future development. Its website can be visited at http://modbus.org.

[1] http://www.modbus.org/faq.php

[2] There are two formats: RTU (compact binary) and ASCII (human readable). The functionality is identical.

[3] https://datatracker.ietf.org/doc/draft-dube-modbus-applproto/

1.2 The MODBUS Technology

The MODBUS specification distinguishes two types of message elements. The PDU (Protocol Data Unit) consists of the function code and as much data as needed for the specific function. The PDU is the unchanging core element of MODBUS and independent from all underlying protocols, which simplifies the implementation of gateway devices.

The PDU combined with protocol specific meta information forms the ADU (Application Data Unit). Serial MODBUS implementations need a leading address field and a trailing CRC:

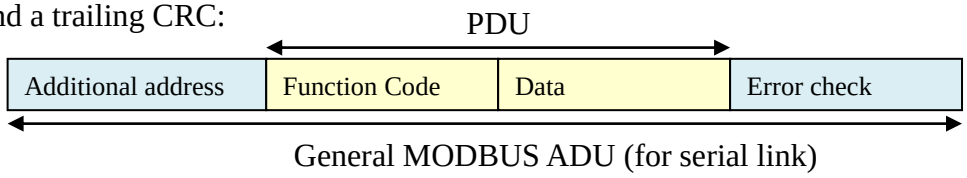

Figure 1 Structure of a MODBUS message

Because TCP over IP provides both functions with a reliable virtual channel, error checking and addressing are not part of the MODBUS TCP specific ADU. The MBAP header (MODBUS Application Protocol Header) basically contains data to simplify bridging with serial MODBUS variants:

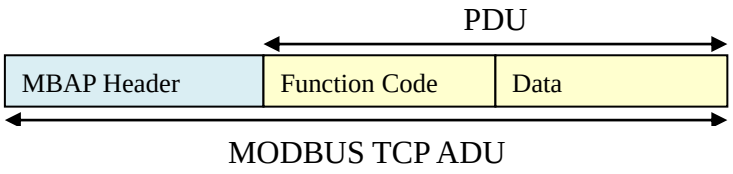

Figure 2 Structure of a MODBUS TCP message

The function code defines the action to be performed by the slave/server. If an action needs any data, for example to write a value into a register, it must be provided in the data field.

MODBUS communication always follows the same pattern:

1. the master sends a request
2. the addressed slave sends a response

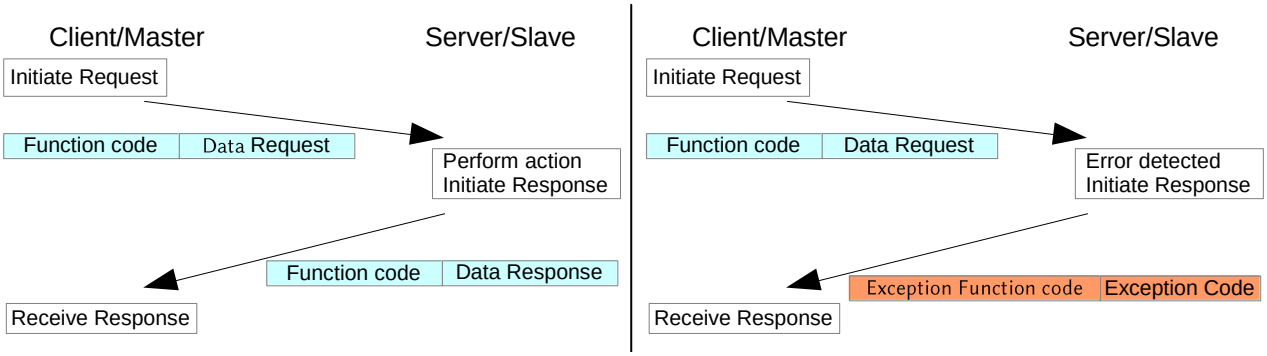

Figure 3 MODBUS communication

A request consists of the function code, which determines the desired action and a data field if required. A reply mirrors the function code and appends the requested data. If an error occurred, an error code is given back instead. Error codes simply are function codes but with the most significant bit set (<Function Code> OR <0x80>). Some examples are given in following table:

Function Code	Response	Exception Response
0x01: Read Coils	0x01, 1 byte Data	0x81, 1 Byte exception code
0x02: Read Discrete Inputs	0x02, 1 byte Data	0x82, 1 Byte exception code
0x06: Write Single Register	0x06, 2b register address, 2b register value	0x86, 1 Byte exception code

MODBUS TCP is defined as a client/server system, which is actually nothing else than a master/slave system, but the first term is used because TCP/IP is a peer-to-peer network without fixed roles.

1.3 MODBUS data model

1.3.1 Number format

Numbers are generally represented as 16 bit integers in big-endian format. That means the most significant byte is sent first like this:

0x11223344 → 0x11, 0x22, 0x33, 0x44

1.3.2 Register mapping

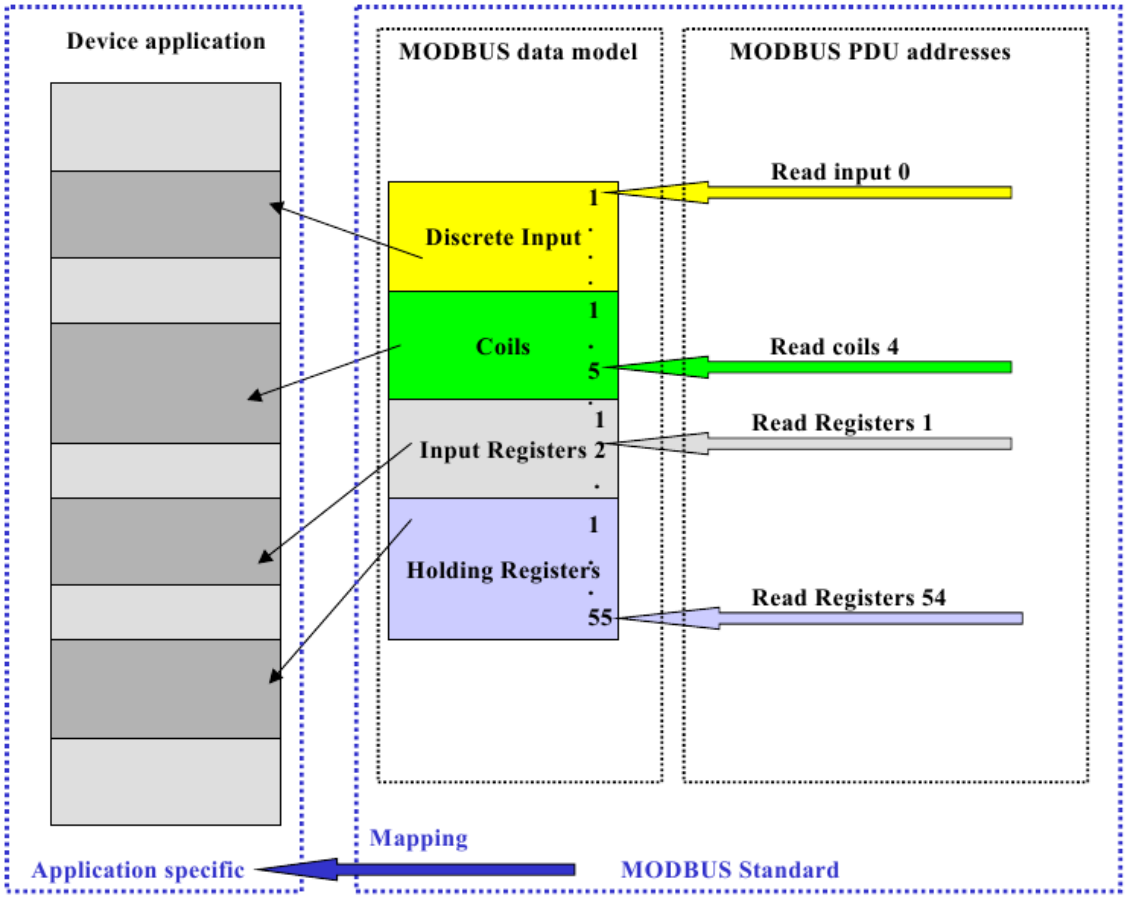

Figure 4 MODBUS Addressing Model [MB2012, page 8][4]

A MODBUS slave device appears to the master like a memory, which consists of four separate segments. The function of the MODBUS memory registers are determined by their address.

[4] The Modbus Organization 2012: "MODBUS Application Protocol Specification V1.1b3", http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf

1. Read Discrete Input

The 1xxxx addresses accommodate single-bit input channels like switches, reed contacts, light curtains and so forth.

2. Read/Write Discrete Output (Coils)

In the 0xxxx address range are single-bit signals to control the output level of “binary” mode devices like relays or lamps.

3. Read Input Registers

16-bit read-only registers in the 3xxx address range that are normally used to represent analog input data like temperatures.

4. Read/Write Output or Holding Registers

Read/write registers, which can be used to hold 16-bit numbers. They can represent analog output values like voltage levels, motor speed, valve positions and so forth. Other common use cases are configuration information or scratchpad memory in a PLC. Holding registers reside in the 4xxx address range

The function of certain memory positions differs for every device, thus if an automation component has to be replaced by a similar model from another vendor, the control unit has to be reprogrammed according to the manual.

2 FreeMODBUS on FM3

THIS CHAPTER INTRODUCES FREEMODBUS AND GIVES SOME DETAILS REGARDING THE FM3 PORT

2.1 FreeMODBUS

FreeMODBUS is an open-source software project that implements MODBUS ASCII/RTU as well as MODBUS TCP. It is written by Christian Walter available under the free BSD license at http://www.freemodbus.org/.

This software example contains an adaption of the MODBUS TCP part to Spansion and is based on the latest release, version 1.5.0 from June 6th 2010.

2.2 FreeMODBUS on FM3

FreeMODBUS' source code files are organized in a logical, modular fashion. All platform independent, protocol related functions are contained in a directory called mb, which has to be added to the project's source tree. As underlying TCP/IP stack, the open source software LwIP (Lightweight IP) is used.

Platform dependent adaptions have to be provided in four files:

- mbconfig.h: allows to disable MODBUS features in order to save memory.
- portevent.h: here an event queue has to be implemented. If an RTOS is used, its mechanism can be utilized.
- portother.h: used to implement a logging mechanism. In our case printf to the serial interface is used.
- porttcp.h: here the interface between FreeMODBUS and the underlying TCP/IP stack is realized.

The FreeMODBUS server is integrated in the common software framework in tasks.c by adding a task, which periodically polls the MODBUS server. If it is not running already, it is started again.

```
/**

*********************************************************************************
 ** \brief Handle MODBUS TCP communication

*********************************************************************************
/
void Task_ModbusTcpCommunication(void)
{
  static boolean_t bMbInitialized = FALSE;

  if (!bMbInitialized)
  {
    if( eMBTCPInit( MB_TCP_PORT_USE_DEFAULT ) != MB_ENOERR )
    {
      printf("%s: can't initialize modbus stack!\r\n", PROG );
    }
    else if( eMBEnable(  ) != MB_ENOERR )
    {
      printf("%s: can't enable modbus stack!\r\n", PROG );
    }
    else
    {
      bMbInitialized = TRUE;
    }
  }
  else
  {
    if (eMBPoll()!= MB_ENOERR )
    {
      /* An error occurred. Maybe we can restart. */
      eMBDisable();
      eMBClose();
      bMbInitialized = FALSE;
    }
  }
}
```

The MODBUS data model is implemented at two places.

A set of preprocessor definitions defines the registers' logical addresses and their number:

```
#define PROG "FreeModbus"
#define REG_INPUT_START 1000
#define REG_INPUT_NREGS 40
#define REG_HOLDING_START 2000
#define REG_HOLDING_NREGS 130
```

A set of arrays act as actual implementations then:

```
static USHORT usRegInputStart = REG_INPUT_START;
static USHORT usRegInputBuf[REG_INPUT_NREGS];
static USHORT usRegHoldingStart = REG_HOLDING_START;
static USHORT usRegHoldingBuf[REG_HOLDING_NREGS];
```

The MODBUS server function eMBPoll() processes every incoming MODBUS TCP message and acts by invoking following callback functions:

- eMBRegInputCB(UCHAR * pucRegBuffer, USHORT usAddress, USHORT usNRegs)
- eMBRegHoldingCB(UCHAR * pucRegBuffer, USHORT usAddress, USHORT usNRegs, eMBRegisterMode eMode)
- eMBRegCoilsCB(UCHAR * pucRegBuffer, USHORT usAddress, USHORT usNCoils, eMBRegisterMode eMode)
- eMBRegDiscreteCB(UCHAR * pucRegBuffer, USHORT usAddress, USHORT usNDiscrete)

The data mapping between board support framework and FreeMODBUS is done by implementing the respective callback functions like this:

```
eMBErrorCode
eMBRegInputCB( UCHAR * pucRegBuffer, USHORT usAddress, USHORT usNRegs )
{
  eMBErrorCode eStatus = MB_ENOERR;
  int iRegIndex;

  if( ( usAddress >= REG_INPUT_START )
    && ( usAddress + usNRegs <= REG_INPUT_START + REG_INPUT_NREGS ) )
  {
    iRegIndex = ( int )( usAddress - usRegInputStart );
    while( usNRegs > 0 )
    {
      *pucRegBuffer++ = ( unsigned char )( usRegInputBuf[iRegIndex] >> 8 );
      *pucRegBuffer++ = ( unsigned char )( usRegInputBuf[iRegIndex] & 0xFF );
      iRegIndex++;
      usNRegs--;
    }
  }
  else
  {
  eStatus = MB_ENOREG;
  }
  return eStatus;
}
```

In this case the code above copies data from aforementioned array into internal registers.

It makes sense to define another task to fill usRegInputBuf[] with valid data. Exactly this is done in following example:

```
void Task_LinkProcessData(void)
{
  // input process data (from masters point of view)
  usRegInputBuf[0] = u32Heartbeat;
  usRegInputBuf[2] = u16ADCData;
  usRegInputBuf[4] = u16DataIn;

  // output process data (from masters point of view)
  u16DataOut = usRegHoldingBuf[ 4];
  u8LedRed   = usRegHoldingBuf[ 8];
  u8LedGreen = usRegHoldingBuf[12];
  u8LedBlue  = usRegHoldingBuf[16];
}
```

Here input register data is mapped to global system variables as well as holding register data.

3 How to Use the MODBUS TCP Demonstration on FM3

THIS CHAPTER INTRODUCES SOME PC SOFTWARE TO INTERACT WITH FREEMODBUS ON SPANSION FM3

In order to test MODBUS slaves, there are several master implementations available on the Internet. Below is a small selection of useful software packages for that purpose, however there are many more.

3.1 Modpoll

Modpoll is a freeware command line application for Linux, Solaris, QNX Neutrino and Microsoft Windows that simulates a MODBUS master. It can be used to set or read-out register values on a MODBUS slave. Further information is available at http://www.modbusdriver.com/modpoll.html.

It has been successfully tested with the FreeMODBUS implementation on SK-FM3-176PMC-FA as shown in Figure 67.

There are some variables defined in tasks.c to simulate process data, which can also be accessed on the evaluation board vie the HMI. Function Task_LinkProcessData(void) regularly copies values between modbus address space and them.

For testing purposes all process variables can be read and written with following command line arguments:

command	**variable**
modpoll -m tcp -r 1000 -t 3:int -c3 192.168.1.20	Read out every second u32Heartbeat, u16ADCData, u16DataIn
modpoll -m tcp -r 2004 -t 4 192.168.1.20 7092	u16DataOut set to 7092
modpoll -m tcp -r 2008 -t 4 192.168.1.20 10	u8LedRed set to 10
modpoll -m tcp -r 2012 -t 4 192.168.1.20 0	u8LedGreen set to 0
modpoll -m tcp -r 2016 -t 4 192.168.1.20 7	u8LedBlue set to 7

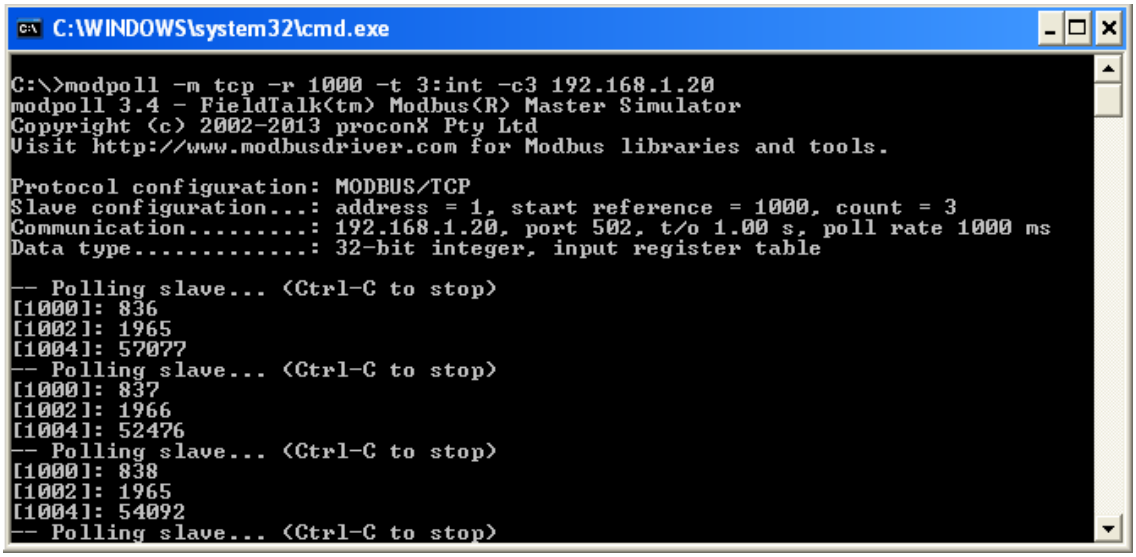

Figure 5 Modpoll demonstrating FreeMODBUS on FM3

3.2 Further PC master software options

Modpoll serves well for testing while developing the slave's firmware, it is rather unsuited in actual real-world use. However as MODBUS TCP is used in areas such as building automation, networking hardware power control and so forth, several options have been made that can act as a MODBUS TCP master and many are even free open-source software. Therefore MODBUS TCP is easy to integrate into most IT infrastructure.

"Although most system administrators will find little need for a Modbus server on any modern hardware, they may find the need to query devices on their network for status (PDU, PDR, UPS, etc)." [Collins2013] [5]

[5] Galen Collins, pymodbus project page at github, 2013, https://github.com/bashwork/pymodbus

There are libraries for a range of programming languages available, such as

- C: http://www.libmodbus.org
- Python: https://github.com/bashwork/pymodbus
- Java: http://sourceforge.net/projects/j2mod/
- Ruby: http://joachimwuttke.de/techblog/modbus.html
- Perl: https://github.com/sourceperl/MBclient
- FreePascal/Lazarus: http://mbutils.sourceforge.net/
- …

Beside libraries there exist also some interesting free projects that seek to implement complete SCADA solutions such as

- MBLogic, web based HMI: http://mblogic.sourceforge.net/
- UniGO, small SCADA for Android: http://www.unigo.se/

Beside these free projects MODBUS is of course supported by many commercial solutions as well.

4 Some notes regarding MODBUS TCP

MODBUS TCP offers advantages and disadvantages:

- Because it is one of the oldest industrial protocols, it is well established and the most commonly deployed protocol in existing SCADA systems. In many cases it is something like the least common denominator.
- It is simple, inexpensive and well supported, also with many free open source options.
- As MODBUS originally was not intended to be connected to a worldwide computer network, no security features are integrated into the protocol, which makes it vulnerable against sabotage [Bristow2011][6].
- For applications like wastewater treatment, building automation or traffic control systems, real-time behavior in the millisecond domain is not necessary, although

[6] See e.g. Mark Bristow's speech at DEFCON 16 conference: ModScan: A SCADA MODBUS Network Scanner,
http://www.defcon.org/html/defcon-16/dc-16-speakers.html#Bristow or
http://www.youtube.com/watch?v=z14tgdvZf_E

operation over large distances is important. Here MODBUS TCP is popular and inexpensive standard IT infrastructure reduces costs.

- The only supported data types are single bits and 16 bit integers. They may be interpreted differently, however both communication partners have to be programmed accordingly.
- Register meaning is not stored in the devices but must be obtained from documentation.

5 Additional Information

More information about Spansion's Microcontrollers can be found on the Internet at http://www.spansion.com/products/microcontrollers/Pages/default.aspx

The software example related to this application note is:
mb9bf61xt_ethernet_lwip_modbustcp