



## Porting Modbus Master in M031/M032

Example Code Introduction for 32-bit NuMicro® Family

### Information

Application	This example code demonstrates how to porting Modbus master in the M031/M032 series microcontroller. The Modbus RTU is an open serial protocol derived from the Master/Slave architecture.	
BSP Version	M031 Series CMSIS BSP V3.01.001	
Hardware	NuMaker-M032SE Ver1.3	

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com



#### 1 Overview

The sample code demonstrates using the UART interface and implementing Modbus master in RTU mode. The Modbus RTU is an open serial protocol derived from the Master/Slave architecture. Modbus RTU messages are the simple 16-bit structure with CRC (Cyclic-Redundant-Checksum).

Due to the simplicity of these messages that can ensure reliability, the basic 16-bit Modbus RTU register structure can be used to pack in floating point, tables, ASCII text, queues, and other unrelated data. The protocol primarily uses an RS-232 or RS-485 serial interface for communication.

User can download Jinqi's Modbus master library from <a href="http://github.com/LittleBigQi/ModBus-Master">http://github.com/LittleBigQi/ModBus-Master</a>.

### 1.1 How to Porting Modbus Master

The Modbus needs porting timer and UART function for Modbus master.

The timer initial function sets timer0 in periodic mode, and enables interrupt and 1000 interrupt tick per second.

```
void Timer_Init(void)
{
   /* Open Timer0 in periodic mode, enable interrupt and 1000 interrupt tick per second */
   TIMER_Open(TIMER0, TIMER_PERIODIC_MODE, 1000);

   /* Enable Timer0 interrupt */
   TIMER_EnableInt(TIMER0);

   /* Enable Timer0 NVIC */
   NVIC_EnableIRQ(TMR0_IRQn);

   /* Start Timer0 counting */
   TIMER_Start(TIMER0);
}
```

The timer0 interrupt hander increases u32\_TimeTick variable for every 1 ms.

```
volatile uint32_t u32_TimeTick=0;
void TMR0_IRQHandler(void)
{
    if(TIMER_GetIntFlag(TIMER0) == 1)
    {
        /* Clear Timer0 time-out interrupt flag */
        TIMER_ClearIntFlag(TIMER0);
```



```
u32_TimeTick++;
}
}
```

The sample provider timer HAL(Hardware Abstract Layer) interface gets time tick and delay function.

```
uint32_t HAL_GetTick(void)
{
   return u32_TimeTick;
}

//This function provides accurate delay (in milliseconds) based on variable incremented.
void HAL_Delay(uint32_t Delay)
{
   uint32_t tickstart = HAL_GetTick();
   uint32_t wait = Delay;

   while((HAL_GetTick() - tickstart) < wait)
   {
   }
}</pre>
```

The UART initial function sets 57600 baud rate and RDA interrupt function.

```
void UART0_Init(void)
{
    /* Reset UART0 */
    SYS_ResetModule(UART0_RST);

    /* Configure UART0 and set UART0 baud rate */
    UART_Open(UART0, 57600);

    /* Enable UART RDA Interrupt */
    UART0->INTEN |= UART_INTEN_RDAIEN_Msk;

    /* Enable UART02 NVIC */
    NVIC_EnableIRQ(UART02_IRQn);
}
```



When Modbus slave sends UART data after the UART interrupt occurs, the UART interrupt function checks UART's RDA interrupt flag and executes Modbus\_Master\_Rece\_Handler function.

```
void UART02_IRQHandler(void)
{
    uint32_t u32IntSts = UART0->INTSTS;
    if(u32IntSts & UART_INTSTS_RDAINT_Msk)
    {
        Modbus_Master_Rece_Handler();
    }
}
```

The Modbus master needs porting Modbus\_Master\_Write and Modbus\_Master\_GetByte function.

```
uint8_t Modbus_Master_Write(uint8_t *buf,uint8_t length)
{
    UART_Write(UART0, (uint8_t *)buf,length);
    return HAL_OK;
}

uint8_t Modbus_Master_GetByte(uint8_t *getbyte)
{
    if(UART_Read(UART0,(uint8_t *)getbyte,1)==1)
        return HAL_OK;
    else
        return HAL_ERROR;
}
```

In the main loop routine, user can call ModbusMaster\_readInputRegisters to send slave ID and read input register command.

The Modbus Master will get two data from slave.

```
ModbusMaster_begin();
while (1)
{
    uint8_t result;
    result = ModbusMaster_readInputRegisters(0x01,0x2, 2);
    //master send readInputRegisters command to slave
    if (result == 0x00)
    {
        //user can read input register from adress 0x02 and get 2 byte context
        Input_Result[0] = ModbusMaster_getResponseBuffer(0x00);
```



```
Input_Result[1] = ModbusMaster_getResponseBuffer(0x01);
}
HAL_Delay(1000);
}
```

#### 1.2 How to Test Modbus Master

Follow steps below to test the Modbus master:

1. Download "Modbus Slave" form https://www.modbustools.com/download.html.



Figure 1-1 Modbus Slave



2. Install "Modbus Slave" first, and select "Slave definition" in Setup menu, as shown in Figure 1-2.

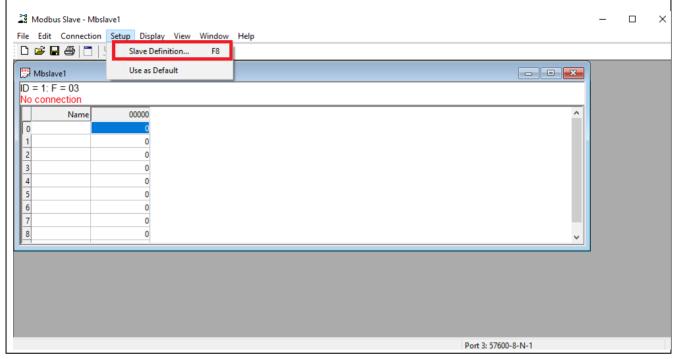


Figure 1-2 Slave Definition in Setup Menu

3. Set the value of Slave ID as "1" and select "04 Input Register (3X)" in Function, as shown in Figure 1-3.

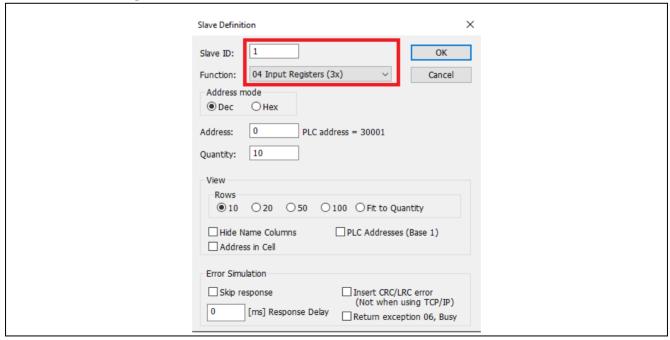


Figure 1-3 Salve Definition Setting



4. Select "Connect" in Connection menu, as shown in Figure 1-4.

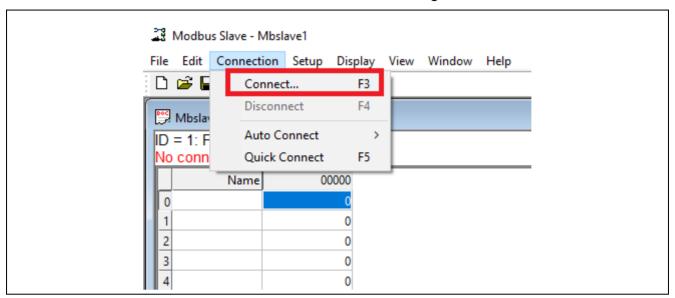


Figure 1-4 Connect in Connection Menu



5. Select "Serial Port" and set "57600 Baud", "8 Data bits", "None parity", "1 Stop Bit" and "RTU" mode, as shown in Figure 1-5.

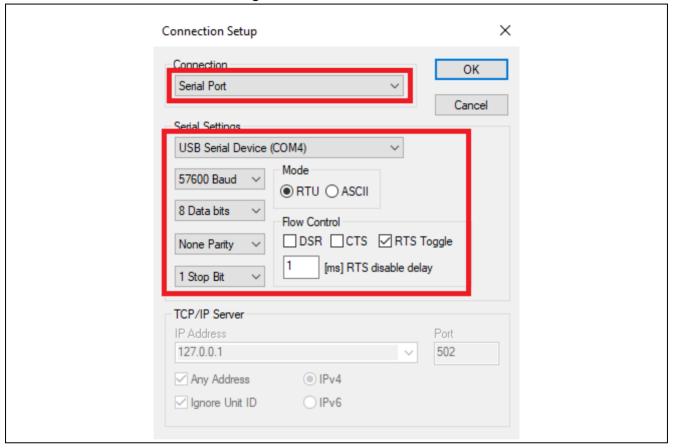


Figure 1-5 Com Port Setup

6. Select "Communication" in Display menu, as shown in Figure 1-6.

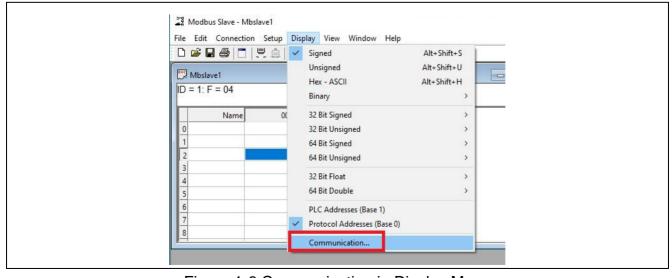


Figure 1-6 Communication in Display Menu



Mbslave1 - -ID = 1: F = 04 Communication Traffic 00000 Name 0 Exit Stop Clear 1 2 Tx:000191-01 04 04 00 05 00 06 6B 87 Rx:000192-01 04 00 02 00 02 D0 0B 3 Tx:000193-01 04 04 00 05 00 06 6B 87 4 Rx:000194-01 04 00 02 00 02 D0 0B 5 Tx:000195-01 04 04 00 05 00 06 6B 87 6 Rx:000196-01 04 00 02 00 02 D0 0B 7 Tx:000197-01 04 04 00 05 00 06 6B 87 8 Rx:000198-01 04 00 02 00 02 D0 0B Tx:000199-01 04 04 00 05 00 06 6B 87 Rx:000200-01 04 00 02 00 02 D0 0B Tx:000201-01 04 04 00 05 00 06 6B 87 Rx:000202-01 04 00 02 00 02 D0 0B Tx:000203-01 04 04 00 05 00 06 6B 87

7. The Modbus slave will send "5" and "6" value in data register, as shown in Figure 1-7.

Figure 1-7 Input Register

8. The procedure will read the input buffer from ModbusMaster\_getResponseBuffer function in main loop.

Figure 1-8 Reading Input Register

#### 1.3 Demo Result

For the demo result, refer to Figure 1-7 and Figure 1-8 Reading Input Register.



### 2 Software and Hardware Requirements

### 2.1 Software Requirements

- BSP version
  - ♦ M031 Series CMSIS BSP V3.01.001
- IDE version
  - ♦ Keil uVersion 5.26

### 2.2 Hardware Requirements

- Circuit components
  - ♦ NuMaker-M032 Ver1.3
  - ◆ The VCOM switch to on to enable VCOM function.
- Pin Connect

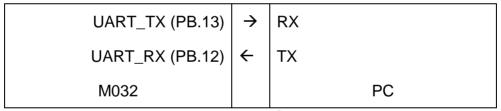


Figure 2-1 Pin Connect



# **3 Directory Information**

The directory structure is shown below.

The allegary ellactare is ellettir below		
C EC_M032_Modbus_Master_V1.00		
	Sample code header and source files	
CMSIS	Cortex® Microcontroller Software Interface Standard (CMSIS) by Arm® Corp.	
Device	CMSIS compliant device header file	
	All peripheral driver header and source files	
	Source file of example code	

Figure 3-1 Directory Structure



### 4 Example Code Execution

- Browse the sample code folder as described in the Directory Information section and double-click Modbus\_Master.uvprojx.
- 2. Enter Keil compile mode.
  - Build
  - Download
  - Start/Stop debug session
- 3. Enter debug mode.
  - Run



# **5 Revision History**

Date	Revision	Description
2021.11.10	1.00	1. Initially issued.



#### **Important Notice**

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.

All the trademarks of products and companies mentioned in this datasheet belong to their respective owners