基于STM32CubeF4移植FreeModbus到STM32F411VET6 (https://www.dhlx.wang/STM32F411VET6/Porting_FreeModbu

内容

>>原创文章，欢迎转载转载请注明：自转载帝鸿龙曦 (http://www.dhlx.wang/)，谢谢！

首先，本人需要声明一下，本文所移植FreeModbus的过程仅能用作参考，虽然FreeModbus的库已经用在工程很多年，但是由于移植的代码，本人经验等问题，所以本人不建议您直接用在工程中，但是可以作为工程测试，调试和个人DIY所用。

一，移植准备程序工具状语从句：¶

我在此次移植过程中，主要用到以下几个程序与工具：

- 32F411EDISCOVERY (http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/LN1848/PF260946?s_searchtype=partnumber)
- FreeModbus v1.5 (http://www.freemodbus.org/)
- Modbus Poll v6.3.0 (http://www.modbustools.com/)
- STM32CubeMX_V4.8.0 (http://www.st.com/web/catalog/tools/FM147/CL1794/SC961/SS1533/PF259242?sc=stm32cube)
- STM32Cube_FW_F4_V1.6.0 (http://www.st.com/web/en/catalog/tools/PF259243)
- Keil_MDK_ARM_V5.15.0 (https://www.keil.com/download/product/)

在移植过程中，主要用到芯片的外设有USART1（PB6，PB7）和TIMER11，请您根据实际情况进行设置。其中TIMER的设置，您需要设置时钟的信号基准为50US，时钟产生中断的时间为1750uS（串口波特率大于19200时，在mbrtu.c中有具体的描述），也就是35个基准时间。

二，需要拷贝产品的文件到工程目录下¶

1，解压缩通过下载得到的freemodbus-v1.5.zip。

2，按照下列的文件列表拷贝相应的文件到对应的目录（没有路径，需要自己建），并在keil中设置好相应的头文件引用路径和源码文件的引用。

拷贝文件到工程文件夹示例：

```
工程文件夹
|.mxproject
|<工程名> .ioc
|
├─驱动//底层驱动文件夹
├─Inc//头文件存放位置
||gpio.h
||stm32f4xx_hal_conf.h
||stm32f4xx_it.h
||tim.h
||usart.h
||
|└─Modbus// Modbus需要用到的头文件
|├─modbus_driver// Modbus函数等头文件
|||├─包含// Modbus通用函数头文件
|||mb.h
|||mbconfig.h
|||mbframe.h
|||mbfunc.h
|||mbport.h
|||mbroto.h
|||mbutils.h
|||
||└─rtu// Modbus RTU函数头文件
||mbcrc.h
||mbartu.h
||
|└─modbus_user// Modbus用户配置头文件
|port.h
|
├─MDK-ARM
|BPMER_USART_TEST.uvprojx
|
└─Src
    |gpio.c
    |main.c
    |stm32f4xx_hal_msp.c
    |stm32f4xx_it.c
    |tim.c
    |usart.c
    |
    └─Modbus// Modbus需要用到的源码文件
        ├─modbus_driver
        ||mb.// Modbus基本函数和设置源码文件
        ||
        |├─functions// Modbus通用函数源码文件
        ||mbfunccoils.c
        ||mbfuncdiag.c
        ||mbfuncdisc.c
        ||mbfuncholding.c
        ||mbfuncinput.c
        ||mbfuncother.c
        ||mbutils.c
        ||
        |└─rtu// Modbus RTU函数源码文件
        |mbcrc.c
        |mbrtu.c
        |
        └─modbus_user// Modbus用户配置源码文件
                portevent.c
                portserial.c
                porttimer.c
```

三，搞定定时器¶

定时器的移植过程中，应该算是比较简单的了，需要简单的设置一下初始化的返回值，完整写出开启计时器的函数，停止计时器的函数，并对定时器回掉函数进行设置下，就OK了。

porttimer.c移植后的代码如下：

```
/ * ----------------------平台包括---------------------- ----- * /
#include “port.h”
#include “stm32f4xx_hal.h”
#include “tim.h”
/ * ---------------------- Modbus包括---------------------- ------- * /
#include “mb.h”
#include “mbport.h”

/ * ----------------------静态函数---------------------- ------ * /
/ * ----------------------私人定义--------------- ---------------- * /
#ifndef Modbus_TimHandle
    #define Modbus_TimHandle htim11
#endif
/ * ---------------------- - 开始实现------------------------- * /
BOOL
xMBPortTimersInit （ USHORT  usTim1Timerout50us  ）
{
    / *系统初始化已经完成定时器的初始化，故事在此不在进行初始化* /
    return  TRUE ;
}


inline  void
vMBPortTimersEnable （  ）
{
    / *启用定时器，超时传递给xMBPortTimersInit（）* /
    / *禁用定时器和复位计数器* /
    HAL_TIM_Base_Stop_IT （&Modbus_TimHandle ）;
    __HAL_TIM_SET_COUNTER （&Modbus_TimHandle ， 0x0000 ）;

    / *启用Timer * /
    HAL_TIM_Base_Start_IT （&Modbus_TimHandle ）;
}

inline  void
vMBPortTimersDisable （  ）
{
    / *禁用任何挂起的计时器。* /
    HAL_TIM_Base_Stop_IT （&Modbus_TimHandle ）;
}

/ *创建一个在计时器到期时调用的ISR。
然后，
此函数*必须调用pxMBPortCBTimerExpired（）以通知协议栈*计时器已过期。
* /
void  prvvTIMERExpiredISR （ void  ）
{
     （ void  ）pxMBPortCBTimerExpired （）;
}
```

stm32f4xx_it.c移植后的部分代码如下所示：

```
/ *包括------------------------------------------- ------------------ * /
#include “stm32f4xx_hal.h”
#include “stm32f4xx.h”
#include “stm32f4xx_it.h”

/ *用户代码开始0 * /

#include “mb.h”
#include “mbport.h”

extern  void  prvvUARTTxReadyISR （void ）;
extern  void  prvvUARTRxISR （void ）;

extern  void  prvvTIMERExpiredISR （ void  ）;
/ * USER CODE END 0 * /
......  ......  ......
/ * USER CODE BEGIN 1 * /
void  HAL_TIM_PeriodElapsedCallback （TIM_HandleTypeDef  * htim ）
{
    / *注意：此函数不应修改，当需要回调时，
             __ HAL_TIM_PeriodElapsedCallback可以在用户文件
    * /
    prvvTIMERExpiredISR （ ）中实现;
}
/ * USER CODE END 1 * /
```

四，串口搞定¶

串口比较难搞一些了，需要设置中断处理函数，串口初始化函数，串口输入输出函数，串口中断服务函数。

portserial.c移植后的代码如下所示：

```
#include  “port.h”
#include  “usart.h”
#include  “stm32f4xx_hal.h”
#include  “stm32f4xx_it.h”

五  / * ---------------------- Modbus包括---------------------- ------- * /
#include  “mb.h”
#include  “mbport.h”

/ * ----------------------静态函数---------------------- ------ * /
void  prvvUARTTxReadyISR  （ void  ）;
void  prvvUARTRxISR  （ void  ）;
/ * ----------------------私人定义---------------------- -------- * /
#ifndef Modbus_UartHandle
    #define Modbus_UartHandle huart1
#endif

/ * ----------------------开始实施---------------------- -  * /

void
vMBPortSerialEnable  （ BOOL  xRxEnable ，  BOOL  xTxEnable  ）
{
    / *如果xRXEnable启用串行接收中断。如果xTxENable使能
     *发送器空中断。
     * /
    if  （ xRxEnable  ）
    {
        / *启用UART数据寄存器不为空中断* /
        __HAL_UART_ENABLE_IT  （&Modbus_UartHandle ，  UART_IT_RXNE ）;
    }
三  else
十  {
        / *禁用UART数据寄存器不为空中断* /
        __HAL_UART_DISABLE_IT  （&Modbus_UartHandle ，  UART_IT_RXNE ）;
    }
    if  （ xTxEnable  ）
    {
        / *使能UART发送数据寄存器空中断* /
        __HAL_UART_ENABLE_IT  （&Modbus_UartHandle ，  UART_IT_TXE ）;
        prvvUARTTxReadyISR  （）;
    }
    else
    {
        / *禁用UART发送数据寄存器空中断* /
        __HAL_UART_DISABLE_IT  （&Modbus_UartHandle ，  UART_IT_TXE ）;
        / *使能UART发送完成中断* /
        __HAL_UART_ENABLE_IT  （&Modbus_UartHandle ，  UART_IT_TC）;
    }

}

BOOL
xMBPortSerialInit  （ UCHAR  ucPORT ，  ULONG  ulBaudRate ，  UCHAR  ucDataBits ，  eMBParity  eParity  ）
{
    / *由于已经在系统初始化中已经完成串口初始化，故在此不需要再一次进行初始化* /
    return  TRUE ;
}

BOOL
xMBPortSerialPutByte  （ CHAR  ucByte  ）
{
    / *在UART发送缓冲区中放入一个字节。
     如果已
     调用
    pxMBFrameCBTransmitterEmpty（），则此函数由协议栈调用* 。* / 如果（HAL_UART_Transmit  （&Modbus_UartHandle  ，（uint8_t  * ）&ucByte ，1 ，0×
01 ）  =!  HAL_OK  ）
    {
        返回 FALSE  ;
    }
    else
    {
        return  TRUE ;
    }
}

BOOL
xMBPortSerialGetByte  （ CHAR  *  pucByte  ）
```

```
{
    / *返回UART接收缓冲区中的字节。调用
     pxMBFrameCBByteReceived（）后，协议栈
将调用此函数*。      * /
    如果（HAL_UART_Receive  （&Modbus_UartHandle  ，（uint8_t  * ）pucByte ，1 ，0×01 ）！=  HAL_OK  ）
    {
        返回 FALSE  ;
    }
    else
    {
        return  TRUE ;
    }

}

/ *
为目标处理器
的发送缓冲区空中断*（或等效的）创建中断处理程序。然后，此函数应调用pxMBFrameCBTransmitterEmpty（），
该函数告诉协议栈*可以发送新字符。然后协议栈将调用
* xMBPortSerialPutByte（）来发送字符。
* /
void  prvvUARTTxReadyISR （ void  ）
{
    pxMBFrameCBTransmitterEmpty （  ）;
}

/ *为目标
*处理器
的接收中断创建一个中断处理程序。然后，此函数应调用pxMBFrameCBByteReceived（）。然后*协议栈将调用xMBPortSerialGetByte（）来检索
*字符。
* /
void  prvvUARTRxISR （ void  ）
{
    pxMBFrameCBByteReceived （  ）;
}

```

stm32f4xx_it.c移植后的部分代码如下所示：

```
/ **
* @brief此函数处理USART1全局中断。
* /
void  USART1_IRQHandler （void ）
{
  / * USER CODE BEGIN USART1_IRQn 0 * /
    if （__HAL_UART_GET_IT_SOURCE （&huart1 ， UART_IT_RXNE ）！=  RESET ）
    {
        prvvUARTRxISR （）; //接收中断处理函数
    }

    if （__HAL_UART_GET_IT_SOURCE （&huart1 ， UART_IT_TXE ）！=  RESET ）
    {
        prvvUARTTxReadyISR （）; //发送完成终端处理函数
    }

    HAL_NVIC_ClearPendingIRQ （USART1_IRQn ）;
  / * USER CODE END USART1_IRQn 0 * /
  HAL_UART_IRQHandler （&huart1 ）;
  / *用户代码开始USART1_IRQn 1 * /

  / * USER CODE END USART1_IRQn 1 * /
}
```

五，其他搞定配置¶

1，搞定开关中断操作

port.h移植后的部分代码如下所示：

```
/ *禁用全部中断* /
#define ENTER_CRITICAL_SECTION（）__ disable_irq（）
/ *开启全部中断* /
#define EXIT_CRITICAL_SECTION（）__ enable_irq（）
```

2，搞定系统配置

mb.c移植后的部分代码如下所示：

```
/ *设置启用Modbus RTU模式，减少编译后的代码量* /
#if MB_RTU_ENABLED == 1
    #include “mbrtu.h”
＃万一
五

/ *减少编译后代码错误以及修改，将判定值由1改为0，并修改MB_ASCII_ENABLED的定义为0，
想要了解原因，请自行修改编译查看* /
#if MB_ASCII_ENABLED == 1
    #include “mbascii。 H”
＃万一

/ *减少编译后代码大小，将判定值由0改为1，并修改MB_TCP_ENABLED的定义为0，想要了解
原因，请自行修改编译查看* /
#if MB_TCP_ENABLED == 1
    #include “mbtcp.h”
＃万一
```

3，搞定寄存器地址差一的问题

需要在源码中注释掉所有的 `usRegAddress++;` 。

4，搞定KEIL编译过程中出现的各种警告

mb.h移植后的部分代码如下所示：

```
extern  void
xMBUtilSetBits （ UCHAR  *  ucByteBuf ， USHORT  usBitOffset ， UCHAR  ucNBits ， \
                 UCHAR  ucValue  ）;

五
extern  UCHAR
xMBUtilGetBits （ UCHAR  *  ucByteBuf ， USHORT  usBitOffset ， UCHAR  ucNBits  ）;
```

mbrtu.c移植后的部分代码如下所示：

```
eMBErrorCode
eMBRTUReceive （ UCHAR  *  pucRcvAddress ， UCHAR  **  pucFrame ， USHORT  *  pusLength  ）
{
    / *处女座心理，必须要求无错误，无警告，可过PC-Lint，目前也没有发现这个定义的用处* /
五   BOOL xFrameReceived = FALSE;
    eMBErrorCode     eStatus  =  MB_ENOERR ;

    ENTER_CRITICAL_SECTION （  ）;
    assert （ usRcvBufferPos  <  MB_SER_PDU_SIZE_MAX  ）;

    / *长度和CRC校验* /
    if （ （ usRcvBufferPos  > =  MB_SER_PDU_SIZE_MIN  ）
        &&  （ usMBCRC16 （ （ UCHAR  *  ） ucRTUBuf ， usRcvBufferPos  ） ==  0  ） ）
    {
         / *保存地址字段。所有帧都传递给上层
         *，如果在那里使用了框架则做出决定。
         * /
        * pucRcvAddress  =  ucRTUBuf [ MB_SER_PDU_ADDR_OFF ];

        / * Modbus-PDU的总长度是Modbus-Serial-Line-PDU减去
         *地址字段的大小和CRC校验和。
         * /
        * pusLength  =  （ USHORT  ）（ usRcvBufferPos  -  MB_SER_PDU_PDU_OFF  -  MB_SER_PDU_SIZE_CRC  ）;

        / *将Modbus PDU的开头返回给调用者。* /
        * pucFrame  =  （ UCHAR  *  ） & ucRTUBuf [ MB_SER_PDU_PDU_OFF ];
// xFrameReceived = TRUE;
    }
    else
三十 {
        eStatus  =  MB_EIO ;
    }

    EXIT_CRITICAL_SECTION （  ）;
    返回 eStatus ;
}
```

5，未知原因的添加代码

这部分代码没有明白，为什么需要添加，有些没有研究透，希望有大神可以指点。

mbrtu.c移植后的部分代码如下所示：

```
eMBErrorCode
eMBRTUSend ( UCHAR ucSlaveAddress , const UCHAR * pucFrame , USHORT usLength )
{
    eMBErrorCode     eStatus = MB_ENOERR ;
    USHORT           usCRC16 ;

    ENTER_CRITICAL_SECTION ( ) ;

    / *检查接收器是否仍处于空闲状态。如果不是，我们在哪里
     *慢处理接收到的帧并且主设备
     在网络上
发送另一个*帧。我们必须中止发送帧。      * /
    if ( eRcvState == STATE_RX_IDLE )
    {
        / * Modbus-PDU之前的第一个字节是从地址。* /
        pucSndBufferCur = ( UCHAR * ) pucFrame - 1 ;
        usSndBufferCount = 1 ;

        / *现在将Modbus-PDU复制到Modbus-Serial-Line-PDU中。* /
        pucSndBufferCur [ MB_SER_PDU_ADDR_OFF ] = ucSlaveAddress ;
        usSndBufferCount + = usLength ;

        / *计算Modbus-Serial-Line-PDU的CRC16校验和。* /
        usCRC16 = usMBCRC16 ( ( UCHAR * ) pucSndBufferCur , usSndBufferCount ) ;
        ucRTUBuf [ usSndBufferCount ++ ] = ( UCHAR ) ( usCRC16 & 0xFF ) ;
        ucRTUBuf [ usSndBufferCount ++ ] = ( UCHAR ) ( usCRC16 >> 8 ) ;

        / *激活发射器。* /
        eSndState = STATE_TX_XMIT ;

        / *插入代码启动第一次发送，这样才可以进入发送完成中断* /
        xMBPortSerialPutByte ( ( CHAR ) * pucSndBufferCur ) ;
        pucSndBufferCur ++ ;
        usSndBufferCount - ;

        vMBPortSerialEnable ( FALSE , TRUE ) ;
    }
    else
    {
        eStatus = MB_EIO ;
    }
    EXIT_CRITICAL_SECTION ( ) ;
    返回 eStatus ;
}
```

六，主搞定函数¶

主函数需要设置数组和处理函数，无其他设置，我放上我移植的代码以供参考。

main.c中移植后的代码如下所示：

```
/ *包括---------------------------------------------- -------------- * /
#include “stm32f4xx_hal.h”
#include “adc.h”
#include “tim.h”
#include “usart.h”
#include “gpio.h”

/ * USER CODE BEGIN包含* /

#include “mb.h”
#include “mbport.h”

/ * USER CODE END包含* /

/ *私有变量---------------------------------------------- ------ * /

/ *用户代码开始PV * /

/ *线圈状态寄存器* /
#define REG_COILS_START 0x0000
#define REG_COILS_SIZE 8

/ *线圈状态输入寄存器* /
#define REG_DISCRETE_START 0x0000
#define REG_DISCRETE_SIZE 8

/ *保持
寄存器
* / #define REG_HOLDING_START 0x0000 #define REG_HOLDING_NREGS 10
```

```
三十
/ *输入
寄存器
* / #define REG_INPUT_START 0x0000 #define REG_INPUT_NREGS 1

/ * USER CODE END PV * /

/ *私有函数原型---------------------------------------- * /
void  SystemClock_Config （void ）;

/ *用户代码开始PFP * /

/ * USER CODE END PFP * /

/ *用户代码开始0 * /

/ *定义线圈状态寄存器的地址起始值和存储数组* /
uint8_t    ucRegCoilsStart  =  REG_HOLDING_START ;
uint8_t    ucRegCoilsBuf [ REG_COILS_SIZE  /  8 ];

/ *定义线圈输入状态寄存器的地址起始值和存储数组* /
uint8_t    ucRegDiscreteStart  =  REG_HOLDING_START ;
uint8_t    ucRegDiscreteBuf [ REG_DISCRETE_SIZE  /  8 ];

/ *定义保持寄存器的地址起始值和存储数组* /
uint16_t    usRegHoldingStart  =  REG_HOLDING_START ;
uint16_t    usRegHoldingBuf [ REG_HOLDING_NREGS ]  =  { 1 ，2 ，3 ，4 ，5 ，6 ，7 ，8 ，9 ，0 };

/ *定义输入寄存器的地址起始值和存储数组* /
uint16_t    usRegInputStart  =  REG_INPUT_START ;
uint16_t    usRegInputBuf [ REG_INPUT_NREGS ];

/ * USER CODE END 0 * /

int  main （void ）
{

    / *用户代码开始1 * /

    / *用户代码结束1 * /

    / * MCU配置---------------------------------------------- --- * /

    / *重置所有外围设备，初始化Flash界面和Systick。* /
    HAL_Init （）;

    / *配置系统时钟* /
    SystemClock_Config （）;

    / *初始化所有已配置的外设* /
    MX_GPIO_Init （）;
    MX_ADC1_Init （）;
    MX_TIM1_Init （）;
    MX_TIM4_Init （）;
    MX_TIM11_Init （）;
    MX_USART1_UART_Init （）;

    / *用户代码开始2 * /

    / *设置Modnbus以RTU模式运行，从机ID为0×01，串口为默认串口，串口波特率为115200，无奇偶校验* /
    eMBInit （MB_RTU ，  0×01 ，  1 ，  115200 ，  MB_PAR_NONE ）;

    / *启用Modbus协议栈。* /
    eMBEnable （）;
    / *用户代码结束2 * /

    / *无限循环* /
    / * USER CODE BEGIN WHILE * /
    while  （1 ）
    {
        （ void  ）eMBPoll （）;
        __HAL_TIM_SetCompare （&htim4 ，TIM_CHANNEL_1 ，usRegHoldingBuf [ 0 ]）;
        __HAL_TIM_SetCompare （&htim4 ，TIM_CHANNEL_2 ，usRegHoldingBuf [ 1 ]）;
        __HAL_TIM_SetCompare （&htim4 ，TIM_CHANNEL_3 ，usRegHoldingBuf [ 2 ]）;
        __HAL_TIM_SetCompare （&htim4 ，TIM_CHANNEL_4 ，usRegHoldingBuf [ 3 ]）;
        HAL_TIM_PWM_Start （&htim4 ，  TIM_CHANNEL_1 ）;
        HAL_TIM_PWM_Start （&htim4 ，  TIM_CHANNEL_2 ）;
        HAL_TIM_PWM_Start （&htim4 ，  TIM_CHANNEL_3 ）;
        HAL_TIM_PWM_Start （&htim4 ，  TIM_CHANNEL_4 ）;

    / *用户代码结束时* /
```

```
111
112     / *用户代码开始3 * /
113
114     }
115     / * USER CODE END 3 * /
116
117 }
118
119 / **系统时钟配置
120 * /
121 void  SystemClock_Config （void ）
122 {
123
124   RCC_OscInitTypeDef  RCC_OscInitStruct ;
125   RCC_ClkInitTypeDef  RCC_ClkInitStruct ;
126
127   __PWR_CLK_ENABLE （）;
128
129   __HAL_PWR_VOLTAGESCALING_CONFIG （PWR_REGULATOR_VOLTAGE_SCALE2 ）;
130
131   RCC_OscInitStruct 。OscillatorType  =  RCC_OSCILLATORTYPE_HSI ;
132   RCC_OscInitStruct 。HSIState  =  RCC_HSI_ON ;
133   RCC_OscInitStruct 。HSICalibrationValue  =  16 ;
134   RCC_OscInitStruct 。PLL 。PLLState  =  RCC_PLL_ON ;
135   RCC_OscInitStruct 。PLL 。PLLSource  =  RCC_PLLSOURCE_HSI ;
136   RCC_OscInitStruct 。PLL 。PLLM  =  8 ;
137   RCC_OscInitStruct 。PLL 。PLLN  =  200 ;
138   RCC_OscInitStruct 。PLL 。PLLP  =  RCC_PLLP_DIV4 ;
139   RCC_OscInitStruct 。PLL 。PLLQ  =  4 ;
140   HAL_RCC_OscConfig （＆RCC_OscInitStruct ）;
141
142   RCC_ClkInitStruct 。ClockType  =  RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_PCLK1 ;
143   RCC_ClkInitStruct 。SYSCLKSource  =  RCC_SYSCLKSOURCE_PLLCLK ;
144   RCC_ClkInitStruct 。AHBCLKDivider  =  RCC_SYSCLK_DIV1 ;
145   RCC_ClkInitStruct 。APB1CLKDivider  =  RCC_HCLK_DIV2 ;
146   RCC_ClkInitStruct 。APB2CLKDivider  =  RCC_HCLK_DIV1 ;
147   HAL_RCC_ClockConfig （＆RCC_ClkInitStruct ，  FLASH_LATENCY_3 ）;
148
149   HAL_SYSTICK_Config （HAL_RCC_GetHCLKFreq （）/ 1000 ）;
150
151   HAL_SYSTICK_CLKSourceConfig （SYSTICK_CLKSOURCE_HCLK ）;
152
153 }
154
155 / *用户代码开始4 * /
156
157 / **
158 * @brief输入寄存器处理函数，输入寄存器可读，但不可写。
159 * @param pucRegBuffer返回数据指针
160 * usAddress寄存器起始地址
161 * usNRegs寄存器长度
162 * @retval eStatus寄存器状态
163 * /
164
165 eMBErrorCode
166 eMBRegInputCB （ UCHAR  *  pucRegBuffer ，  USHORT  usAddress ，  USHORT  usNRegs  ）
167 {
168     eMBErrorCode     eStatus  =  MB_ENOERR ;
169     int16_t             iRegIndex ;
170
171 //用作例子
172     usRegInputBuf [ 0 ]  =  0x11 ;
173 //例子结束
174
175     //查询是否在寄存器范围内
176     //为了避免警告，修改为有符号整数
177     if （ （（int16_t ） usAddress  > =  REG_INPUT_START  ） \
178          &&  （ usAddress  +  usNRegs  <=  REG_INPUT_START  +  REG_INPUT_NREGS  ） ）
179     {
180         //获得操作
181         偏移量，本次操作起始地址 - 输入寄存器的初始地址iRegIndex  =  （ int16_t  ）（ usAddress  -  usRegInputStart  ）;
182         //逐个赋值
183         while （ usNRegs  >  0  ）
184         {
185             //赋值高字节
186             * pucRegBuffer ++ =  （ uint8_t  ）（ usRegInputBuf [ iRegIndex ]  >>  8  ）;
187             //赋值低字节
188             * pucRegBuffer ++  =  （ uint8_t  ）（ usRegInputBuf [ iRegIndex ]  ＆ 0xFF  ）;
189             //
190             偏移量增加iRegIndex ++ ;
191             //被操作
```

```
            寄存器数量递减usNRegs - ;
        }
    }
    else
    {
        //返回错误状态，
        寄存器 数量不对eStatus =  MB_ENOREG ;
    }

    返回 eStatus ;
}

/ **
* @brief保持寄存器处理函数，保持寄存器可读，可读可写
* @param pucRegBuffer读操作时 - 返回数据指针，写操作时 - 输入数据指针
* usAddress寄存器起始地址
* usNRegs寄存器长度
* eMode操作方式，读或者写
* @retval eStatus寄存器状态
* /
eMBErrorCode
eMBRegHoldingCB （ UCHAR  *  pucRegBuffer ， USHORT  usAddress ， USHORT  usNRegs ，
eMBRegisterMode  eMode  ）
{
    //错误状态
    eMBErrorCode  eStatus  =  MB_ENOERR ;
    //
    偏移量int16_t  iRegIndex ;

    //判断寄存器是不是在范围内
    if （ （ （int16_t ）usAddress  > =  REG_HOLDING_START  ） \
     &&  （ usAddress  +  usNRegs  <=  REG_HOLDING_START  +  REG_HOLDING_NREGS  ） ）
    {
        //计算
        偏移 量iRegIndex =  （ int16_t  ）（ usAddress  -  REG_HOLDING_START  ） ;

        switch  （ eMode  ）
        {
            //读处理函数
            case  MB_REG_READ :
            while （ usNRegs  >  0  ）
            {
                * pucRegBuffer ++  =  （ uint8_t  ）（ usRegHoldingBuf [ iRegIndex ]  >>  8  ）;
                * pucRegBuffer ++  =  （ uint8_t  ）（ usRegHoldingBuf [ iRegIndex ]  & 0xFF  ）;
                iRegIndex ++ ;
                usNRegs - ;
            }
            打破;

            //写处理函数
            case  MB_REG_WRITE :
            while （ usNRegs  >  0  ）
            {
                usRegHoldingBuf [ iRegIndex ]  =  * pucRegBuffer ++  <<  8 ;
                usRegHoldingBuf [ iRegIndex ]  | =  * pucRegBuffer ++ ;
                iRegIndex ++ ;
                usNRegs - ;
            }
            打破;
        }
    }
    else
    {
        //返回错误状态
        eStatus  =  MB_ENOREG ;
    }

    返回 eStatus ;
}


/ **
* @brief线圈寄存器处理函数，线圈寄存器可读，可读可写
* @param pucRegBuffer读操作---返回数据指针，写操作 - 返回数据指针
* usAddress寄存器起始地址
* usNRegs寄存器长度
* eMode操作方式，读或者写
* @retval eStatus寄存器状态
* /
eMBErrorCode
eMBRegCoilsCB （ UCHAR  *  pucRegBuffer ， USHORT  usAddress ， USHORT  usNCoils ，
```

```
eMBRegisterMode  eMode  )
{
    //错误状态
    eMBErrorCode  eStatus  =  MB_ENOERR ;
    //寄存器个数
    int16_t  iNCoils  =  ( int16_t  )usNCoils ;
    //
    寄存器偏移 量int16_t usBitOffset ;

    //检查寄存器是否在指定范围内
    if  (  (  (int16_t )usAddress  > =  REG_COILS_START  ) &&
     ( usAddress  +  usNCoils  <=  REG_COILS_START  +  REG_COILS_SIZE  ) )
    {
        //计算
        寄存器偏移量usBitOffset  =  ( int16_t  ) ( usAddress  -  REG_COILS_START  ) ;
        switch  ( eMode  )
        {
            //读操作
            案例 MB_REG_READ :
                while  ( iNCoils  >  0  )
                {
                    * pucRegBuffer ++  =  xMBUtilGetBits  ( ucRegCoilsBuf ,  usBitOffset ,
                     ( uint8_t  ) ( iNCoils  >  8  ?  8  :  iNCoils  ) );
                    iNCoils  -  =  8 ;
                    usBitOffset  + =  8 ;
                }
            打破;

            //写操作
            案例 MB_REG_WRITE :
            while  ( iNCoils  >  0  )
                {
                    xMBUtilSetBits  ( ucRegCoilsBuf ,  usBitOffset ,
                     ( uint8_t  ) ( iNCoils  >  8  ?  8  :  iNCoils  ),
                    * pucRegBuffer ++  ) ;
                    iNCoils  -  =  8 ;
                }
            打破;
        }

    }
    else
    {
        eStatus  =  MB_ENOREG ;
    }
    return  eStatus ;
}

/ **
* @brief开关输入寄存器处理函数，开关输入寄存器，可读
* @param pucRegBuffer读操作---返回数据指针，写操作 - 返回数据指针
* usAddress寄存器起始地址
* usNRegs寄存器长度
* eMode操作方式，读或者写
* @retval eStatus寄存器状态
* /
eMBErrorCode
eMBRegDiscreteCB  ( UCHAR  *  pucRegBuffer ,  USHORT  usAddress ,  USHORT  usNDiscrete  )
{
    //错误状态
    eMBErrorCode  eStatus  =  MB_ENOERR ;
    //操作寄存器个数
    int16_t  iNDiscrete  =  ( int16_t  )usNDiscrete ;
    //偏移量
    uint16_t usBitOffset ;

    //判断寄存器时候再制定范围内
    if  (  (  (int16_t )usAddress  > =  REG_DISCRETE_START  ) &&
     ( usAddress  +  usNDiscrete  <=  REG_DISCRETE_START  +  REG_DISCRETE_SIZE  ) )
    {
        //获得
        偏移量usBitOffset  =  ( uint16_t  ) ( usAddress  -  REG_DISCRETE_START  ) ;

        while  ( iNDiscrete  >  0  )
        {
            * pucRegBuffer ++  =  xMBUtilGetBits  ( ucRegDiscreteBuf ,  usBitOffset ,
             ( uint8_t ) ( iNDiscrete  >  8  ?  8  :  iNDiscrete  ) );
            iNDiscrete  -  =  8 ;
            usBitOffset  + =  8 ;
        }

```

```
    }
    else
    {
        eStatus  =  MB_ENOREG ;
    }
    return  eStatus ;
}

/ *用户代码结束4 * /

#ifdef USE_FULL_ASSERT

/ **
   * @brief报告
   发生assert_param错误
的源文件名和源行号*。    * @param文件：指向源文件名的指针
   * @param行：assert_param错误行源编号
   * @retval无
   * /
void  assert_failed （uint8_t *  文件， uint32_t  行）
{
  / * USER CODE BEGIN 6 * /
  / *用户可以添加他自己的实现来报告文件名和行号，
    例如：printf（“错误的参数值：文件%s在行%d \ r \ n”，文件，行）* /
  / * USER CODE END 6 * /

}

＃万一

/ **
  * @}
  * /

/ **
  * @}
* /

/ **************（C）版权所有STMicroelectronics *****文件结尾******** /
```

七，结语¶

本次移植过程也是个人摸索得出的，如果有不尽完善的地方，欢迎您提出，我进行验证更正。

请问您是否有什么想法？或者您认为哪里有不对？有一些地方您没看明白？当然您可也以在下面留下您的见解。

0 Comments (https://www.dhlx.wang/STM32F411VET6/Porting_FreeModbus_to_STM32F411VET6_based_on_STM32CubeF4.html#disqus_thread)

- « 使用UART (https://www.dhlx.wang/STM32F411VET6/Porting_AN4657_STM32Cube_IAP_using_UART_to_STM32F411VET6.html)移植AN4657到STM32F411VET6 STM32Cube IAP (https://www.dhlx.wang/STM32F411VET6/Porting_AN4657_STM32Cube_IAP_using_UART_to_STM32F411VET6.html)
- 雷米“心理罪”书评系列轮回，抉择 (https://www.dhlx.wang/Book_Review/Psychological_Crime.html) »

发布时间

2015年7月6日

最近更新时间

2015年7月6日

类别

STM32F411VET6 (https://www.dhlx.wang/categories.html#stm32f411vet6-ref)

标签

- FreeModbus 1 (https://www.dhlx.wang/tags.html#freemodbus-ref)
- STM32CubeF4 2 (https://www.dhlx.wang/tags.html#stm32cubef4-ref)
- STM32F411VET6 3 (https://www.dhlx.wang/tags.html#stm32f411vet6-ref)

保持联系

(https://github.com/dihonglongxi) (https://weibo.com/510838401) (mailto:gsq510838401@gmail.com)

- 由Pelican (http://getpelican.com/)提供技术支持。主题：优雅 (http://oncrashreboot.com/pelican-elegant)的塔尔哈曼苏尔 (http://oncrashreboot.com)