基于 STM32F10X 系列通信控制 AQMD 驱动器应用示例

目　录

1. 概要说明。

本文是基于STM32F10X系列单片机通过调用modbusmasterRTU.lib库函数采用485通信方式对 AQMD 电机驱动器进行的运动控制的应用示例。整个工程文件中包括了STM32F10X 系列的官方库文件，modbus 协议库函数和运动控制示例相关代码文件。本示例代码和文档可作为通过 STM32 单片机控制 AQMD 电机驱动器的开发参考手册使用。

2. 主要代码（目录）文件功能说明（STM32F10X库除外）。

2.1 modbusmasterRTU 库文件。（注意此部分文件函数名不可更改）
ModbusCommon.h----------modbus 通信相关宏定义，错误码定义等头文件；
ModbusMaster.h----------modbus 协议主站通信主任务相关函数头文件；
Typedef.h----------库函数数据类型定义头文件；
ModbusmasterRTU.lib----------modbusmasterRTU 静态库文件；
callbackfunction.h----------modbusmasterRTU 库与底层驱动接口回调函数头文件。
2.2 基于 STM32 的底层驱动设置与接口回调函数实现文件。
SystemTimer.c----------系统时间主定时器和中断相关接口函数实现；
Usart.c----------串口初始化与配置相关驱动接口函数实现；
SystemInit.c----------系统时钟初始化和底层驱动初始化函数。
2.3 485 通信控制驱动器示例程序文件。
Comunication.c----------通信主任务函数；
Main.c----------工程主函数，驱动器控制示例相关函数。

3. 变量类型说明。

示例代码中使用的变量类型都是基于 stm32f10x.h 库文件中定义的变量类型。

```
Typedef int32_t    s32;
typedef int16_t s16;
typedef int8_t    s8;

typedef const int32_t sc32;    /*!< Read Only */
typedef const int16_t sc16;    /*!< Read Only */
typedef const int8_t sc8;      /*!< Read Only */

typedef __IO int32_t    vs32;
typedef __IO int16_t    vs16;
typedef __IO int8_t      vs8;

typedef __I int32_t vsc32;    /*!< Read Only */
typedef __I int16_t vsc16;    /*!< Read Only */
typedef __I int8_t vsc8;      /*!< Read Only */

typedef uint32_t    u32;
typedef uint16_t u16;
typedef uint8_t    u8;
```

```
typedef const uint32_t uc32;    /*!< Read Only */
typedef const uint16_t uc16;    /*!< Read Only */
typedef const uint8_t uc8;      /*!< Read Only */

typedef __IO uint32_t    vu32;
typedef __IO uint16_t vu16;
typedef __IO uint8_t    vu8;

typedef __I uint32_t vuc32;    /*!< Read Only */
typedef __I uint16_t vuc16;    /*!< Read Only */
typedef __I uint8_t vuc8;      /*!< Read Only */

typedef enum {FALSE = 0, TRUE = !FALSE} bool;
```

4. 相关代码文件与库函数功能说明。

4.1 Comunication.c文件与函数。

Table1． Comunication.c 文件与函数

函数名	描述
COM_Task	串口通信主任务函数

4.1.1 函数COM_Task

Table2. 函数 COM_Task

函数名	COM_Task
函数原型	void COM_Task (void)
功能描述	Modbus 主站协议通信主任务函数
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	main

4.2 Main.c文件与函数。

Table3． Main.c 文件与函数

函数名	描述
DemoBLS	无刷系列控制演示函数
DemoNS	有刷系列控制演示函数
main	主函数

4.2.1 函数DemoBLS

Table4. 函数 DemoBLS

函数名	DemoBLS
函数原型	void DemoBLS(void)
功能描述	演示控制无刷系列驱动器，IN2 下降沿停车，IN3 下降沿切换转动方向

输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	main

4.2.2 函数DemoNS

Table5. 函数 DemoNS

函数名	DemoNS
函数原型	void DemoNS(void)
功能描述	演示控制有刷系列驱动器，AI1 高电平停车，AI2 高电平切换转动方向
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	main

4.2.3 函数main

Table6. 函数 main

函数名	main
函数原型	int main(void)
功能描述	主函数
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	系统

4.3 Modbuscommon.h文件与函数。

Table7． 错误码定义与描述。

错误码	描述
MB_ERR_NONE=0x00U	无错误
MB_ERR_ILLEGAL_FUNCCODE=0x01U	无效功能码
MB_ERR_ILLEGAL_DATAADDR=0x02U	无效数据地址
MB_ERR_ILLEGAL_DATAVALUE=0x03U	无效数据值
MB_ERR_SLAVE_MACH_FAULT=0x04U	从站设备故障
MB_ERR_DELAY_PROCESS=0x05U	请求已被确认，但需要较长时间来处理
MB_ERR_SALVE_BUSY=0x06U	从站忙
MB_ERR_SAVE_PARITY_ERR=0x08U	存储校验错误
MB_ERR_UNAPPLICABLE_GATEWAY=0x0AU	不可用的网关
MB_ERR_GATEWAY_DEVICE_RESPONS_FAIL=0x0BU	网关目标设备响应失败
MB_EXTEND_ERR_FORBID_OPER=0x40U	扩展错误，禁止操作

MB_EXTEND_ERR_HAVENOT_LEARN_PHASE=0x60U	扩展错误，尚未学习电机相序
MB_EXTEND_ERR_UNDEFINE_ERR=0xFFU	其他未定义错误
MB_LOCAL_ERR_SLAVE_ADDR_ERR=0xE0U	本地错误，从站地址错误
MB_LOCAL_ERR_INVALID_BUF_DATA=0xE1U	本地错误，无效缓冲区数据
MB_LOCAL_ERR_RESPONSES_TIMEOUT=0xE2U	本地错误，从站响应超时
MB_LOCAL_ERR_ILLEGAL_FUNCTION_ADDR=0xE3U	本地错误，无效功能函数地址
MB_LOCAL_ERR_ILLEGAL_DATAREG_ADDR=0xE4U	本地错误，无效数据寄存器地址
MB_LOCAL_ERR_SEND_BUFFER_IS_FULL=0xE5U	本地错误，发送缓冲区已满
MB_LOCAL_ERR_TRANSIT_FAIL=0xE6U	本地错误，数据传送失败
MB_LOCAL_ERR_RETURNDATA_ERR=0xE7U	本地错误，返回数据错误

4.4 ModbusMaster.h文件与函数。

Table8． ModbusMaster.h 文件与函数

函数名	描述
MB_ReadHoldReg	0x03 功能码，读保持寄存器功能函数
MB_WirteSingleReg	0x06 功能码，写单个寄存器功能函数
MB_WirteMultReg	0x10 功能码，写多个连续寄存器功能函数
MB_GetRecData	获取从站返回的 16 位格式数据
MB_GetBufAddr	获取存放从站返回数据的首地址和有效数据个数
MB_GetMasterEvent	获取主站当前通信事务状态
MB_SetMasterEvent	设置主站当前通信事务状态
MB_SetMasterErrCode	设置主站通信错误码值
MB_GetMasterErrCode	获取主站通信错误码值
MB_SetMasterMode	设置主站通信工作模式
MB_GetMasterMode	获取主站通信工作模式
MB_MasterTask	主站 modbus 通信主任务函数

4.4.1 函数MB_ReadHoldReg

Table9. 函数 MB_ReadHoldReg

函数名	MB_ReadHoldReg
函数原型	bool MB_ReadHoldReg(u16 uiSlaveAddr,u16 uiRegStartAddr,u16 uiRegCnt)
功能描述	0x03 功能码，读保持寄存器功能函数
输入参数	uiSlaveAddr：从站地址， uiRegStartAddr：起始寄存器地址， uiRegCnt：读取寄存器数量。
输出参数	无
返回值	发送成功返回 true 否则返回 false
先决条件	无
被调用函数	DemoBLS,DemoNS

4.4.2 函数MB_WirteSingleReg

Table10. 函数 MB_WirteSingleReg

函数名	MB_WirteSingleReg

函数原型	bool MB_WirteSingleReg(u16 uiSlaveAddr,u16 uiRegAddr,u16 uiData)
功能描述	0x06 功能码，写单个寄存器功能函数
输入参数	uiSlaveAddr：从站地址， uiRegAddr：写寄存器地址， uiData：待写入的值。
输出参数	无
返回值	发送成功返回 true 否则返回 false
先决条件	无
被调用函数	DemoBLS,DemoNS

4.4.3 函数MB_WirteMultReg

Table11. 函数 MB_WirteMultReg

函数名	MB_WirteMultReg
函数原型	bool MB_WirteMultReg(u16 uiSlaveAddr,u16 uiRegStartAddr,u16 uiRegCnt,u16 uiDataCnt,u16 *puiDataReg)
功能描述	0x10 功能码，写多个连续寄存器功能函数
输入参数	uiSlaveAddr：从站地址， uiRegStartAddr：写寄存器起始地址， uiRegCnt：写寄存器个数， uiDataCnt：待写数据个数（一般 uiDataCnt=2* uiRegCnt）， puiDataReg：待写数据的首地址。
输出参数	无
返回值	发送成功返回 true 否则返回 false
先决条件	无
被调用函数	保留函数（需要写多个寄存器值时使用）

4.4.4 函数MB_GetRecData

Table12. 函数 MB_GetRecData

函数名	MB_GetRecData
函数原型	void MB_GetRecData(u16 *puiDataReg,u16 uiRegStartAddr,u16 uiRegCnt)
功能描述	获取从站返回的 16 位格式的数据
输入参数	puiDataReg：用于存放获取到的数据的首地址， uiRegStartAddr：目标源数据的相对首地址的偏移量， uiRegCnt：需要获取的数据个数。
输出参数	puiDataReg[uiRegStartAddr]：获取的数据依次存放在此地址下面
返回值	无
先决条件	无
被调用函数	DemoBLS,DemoNS

4.4.5 函数MB_GetBufAddr

Table13. 函数 MB_GetBufAddr

函数名	MB_GetBufAddr
函数原型	void MB_GetBufAddr(u16 **ppuiBuf,u16 *puiBufCnt)

功能描述	获取存放从站返回数据的首地址和有效数据个数
输入参数	ppuiBuf：用于存放获取到的首地址指针， puiBufCnt：用于存放获取到的数据个数的地址。
输出参数	ppuiBuf：获取到存放返回数据的首地址， puiBufCnt：获取到的有效数据个数。
返回值	无
先决条件	无
被调用函数	保留函数

4.4.6 函数MB_GetMasterEvent

Table14. 函数 MB_GetMasterEvent

函数名	MB_GetMasterEvent
函数原型	MBEVENT MB_GetMasterEvent(void)
功能描述	获取主站当前通信事务状态
输入参数	无
输出参数	无
返回值	当前主站通信事务状态 typedef enum __MBEVENT { MM_EVENT_NONE=0,//空闲，此状态下可接受数据发送操作 MM_EVENT_REC_COMP,//串口数据接收完毕，对数据进行帧分析处理 MM_EVENT_SND_READY,//需发送数据装入缓冲区完毕，启动串口发送数据 MM_EVENT_SND_SENDING,//串口正在发送缓冲区数据 MM_EVENT_SND_COMP//缓冲区数据发送完毕，切换发送状态 }MBEVENT
先决条件	无
被调用函数	DemoBLS,DemoNS，MBMR_ResponsesTimeoutISR

4.4.7 函数MB_SetMasterEvent

Table15. 函数 MB_SetMasterEvent

函数名	MB_SetMasterEvent
函数原型	void MB_SetMasterEvent(MBEVENT Evnet)
功能描述	设置主站当前通信事务状态
输入参数	Evnet：需要设置的通信事务状态
输出参数	无
返回值	无
先决条件	无
被调用函数	MBMR_SendFrame，MBMR_timer2T35ISR，MBMR_ResponsesTimeoutISR

4.4.8 函数MB_SetMasterErrCode

Table16. 函数 MB_SetMasterErrCode

函数名	MB_SetMasterErrCode

函数原型	void MB_SetMasterErrCode(MBERRCODE Errcode)
功能描述	设置主站通信错误码值
输入参数	Errcode：需要设置的错误码值
输出参数	无
返回值	无
先决条件	无
被调用函数	MBMR_SendFrame，MBMR_timer2T35ISR，MBMR_ResponsesTimeoutISR

4.4.9 函数MB_GetMasterErrCode

Table17. 函数 MB_GetMasterErrCode

函数名	MB_GetMasterErrCode
函数原型	MBERRCODE MB_GetMasterErrCode(void)
功能描述	获取主站当前通信错误码值
输入参数	无
输出参数	无
返回值	MBERRCODE（见 table7 错误码定义）
先决条件	无
被调用函数	DemoBLS,DemoNS

4.4.10 函数MB_SetMasterMode

Table18. 函数 MB_SetMasterMode

函数名	MB_SetMasterMode
函数原型	void MB_SetMasterMode(MODEBUSMODE Mmode)
功能描述	设置主站通信工作模式
输入参数	Mmode：工作模式。 Typedef enum __MODEBUSMODE { MB_MODE_NOMAL=0,//普通主从工作模式 MB_MODE_TRANSIT//透传工作模式（当前库不支持此模式） }MODEBUSMODE;
输出参数	无
返回值	无
先决条件	无
被调用函数	保留函数

4.4.11 函数MB_GetMasterMode

Table19. 函数 MB_GetMasterMode

函数名	MB_GetMasterMode
函数原型	MODEBUSMODE MB_GetMasterMode(void)
功能描述	获取主站通信工作模式
输入参数	无
输出参数	无
返回值	主站当前通信工作模式

先决条件	无
被调用函数	保留函数

4.4.12 函数MB_MasterTask

Table20. 函数 MB_MasterTask

函数名	MB_MasterTask
函数原型	void MB_MasterTask(void)
功能描述	主站 modbus 通信主任务函数
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	COM_Task

4.5 SystemInit.c文件与函数。

Table21． SystemInit.c 文件与函数

函数名	描述
System_Init	系统时钟与外围设备初始化

4.5.1 函数System_Init

Table22 函数 System_Init

函数名	System_Init
函数原型	void System_Init(void)
功能描述	系统时钟与外围设备初始化
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	Main

4.6 SystemTimer.c文件与函数。

Table23． SystemTimer.c 文件与函数

函数名	描述
Timer1_Init	TIM1 定时器配置初始化
Timer1_Enable	TIM1 使能
Timer1_Disable	TIM1 取消使能
SYST_GetSystemTime	获取系统时间（从启动到获取当前时刻经历的时间，单位毫秒）
TIM1_UP_IRQHandler	TIM1 中断入口函数

4.6.1 函数Timer1_Init

Table24 函数 Timer1_Init

函数名	Timer1_Init

函数原型	void Timer1_Init(void)
功能描述	TIM1 定时器配置初始化
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	System_Init

4.6.2 函数Timer1_Enable

Table25 函数 Timer1_Enable

函数名	Timer1_Enable
函数原型	void Timer1_Enable(void)
功能描述	TIM1 使能
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	TIM1_Configuration

4.6.3 函数Timer1_Disable

Table26 函数 Timer1_Disable

函数名	Timer1_Disable
函数原型	void Timer1_Disable(void)
功能描述	TIM1 取消使能
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	无

4.6.4 函数SYST_GetSystemTime

Table27 函数 SYST_GetSystemTime

函数名	SYST_GetSystemTime
函数原型	u32 SYST_GetSystemTime(void)
功能描述	获取系统时间（从系统启动到获取当前时刻所经历的时间，单位为毫秒）
输入参数	无
输出参数	无
返回值	从系统启动到获取当前时刻所经历的时间
先决条件	无
被调用函数	无

4.6.5 函数TIM1_UP_IRQHandler

Table28 函数 TIM1_UP_IRQHandler

函数名	TIM1_UP_IRQHandler
函数原型	void TIM1_UP_IRQHandler(void)
功能描述	TIM1 中断入口函数
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	系统定时器中断

4.7 Usart.c文件与函数。

Table29． Usart.c 文件与函数

函数名	描述
USART1_ConfigurationParam	串口 1 配置初始化
USART1_RXTXSwitch	串口 1 发送接收功能切换
USART1_RecData	串口 1 接收数据
USART1_SendData	串口 1 发送数据
USART1_IRQHandler	串口 1 中断入口函数

4.7.1 函数USART1_ConfigurationParam

Table30 函数 USART1_ConfigurationParam

函数名	USART1_ConfigurationParam
函数原型	void USART1_ConfigurationParam(u32 baudrate,u16 databit,u16 stopbit,u16 parity)
功能描述	串口 1 配置初始化
输入参数	Baudrate：波特率， Databit：数据位数， Stopbit：停止位， Parity：校验方式。
输出参数	无
返回值	无
先决条件	无
被调用函数	COM_Init

4.7.2 函数USART1_RXTXSwitch

Table31 函数 USART1_RXTXSwitch

函数名	USART1_RXTXSwitch
函数原型	Void USART1_RXTXSwitch(bool rxsta,bool txsta)
功能描述	串口 1 发送接收功能切换
输入参数	rxsta：接收使能， txsta：发送使能。
输出参数	无
返回值	无
先决条件	无

被调用函数	MBMR_SendFrame，MBMR_timer2T35ISR，MBMR_ResponsesTimeoutISR，MbSndReady，MB_SetMasterMode

4.7.3 函数USART1_RecData

Table32 函数 USART1_RecData

函数名	USART1_RecData
函数原型	u16 USART1_RecData(void)
功能描述	从串口 1 接收数据
输入参数	无
输出参数	无
返回值	接收到的数据
先决条件	无
被调用函数	MBMR_RecFrame

4.7.4 函数USART1_SendData

Table33 函数 USART1_SendData

函数名	USART1_SendData
函数原型	void USART1_SendData(u16 data)
功能描述	从串口 1 发送数据
输入参数	Data：待发送的数据
输出参数	无
返回值	无
先决条件	无
被调用函数	MBMR_SendFrame

4.7.5 函数USART1_IRQHandler

Table34 函数 USART1_IRQHandler

函数名	USART1_IRQHandler
函数原型	void USART1_IRQHandler(void)
功能描述	串口 1 中断入口函数
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用函数	系统串口中断

5. 调用modbusmasterRTU.lib函数库示例。

该应用示例是基于 STM32 单片机和 MDK 编译平台进行的应用开发。该示例分为无刷 DemoBLS 和有刷 DemoNS 两个部分控制示例。

1、无刷驱动应用控制示例功能说明：

无刷示例通过设置通信参数为波特率 115200，无校验加 2 停止。实现驱动器在 IN2 端口接收到下降沿信号时执行停车操作，在 IN3 端口接收到下降沿时切换转动方向，转速采用占空比控制的方式，在程序里事先设定完成。

无刷驱动器相关的状态和控制寄存器地址说明如下：

寄存器地址（16 进制）	**描述**	**说明**
0x40	停止命令	0：正常停止，1：紧急停止，2：自由停止
0x42	设置占空比	-1000~1000，数值乘以 0.1 为目标占空比
0x7028	IN2 边沿触发	0：下降沿，1：上升沿
0x702D	IN3 边沿触发	0：下降沿，1：上升沿

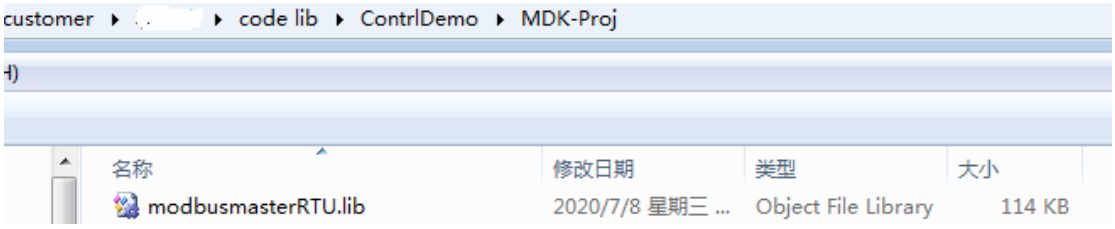
2、有刷驱动应用控制示例功能说明：
有刷示例通过设置通信参数为波特率 115200，无校验加 2 停止。实现驱动器在 AI1 端口接收到高电平信号时执行停车操作，在 AI2 端口接收到高电平信号时切换转动方向，转速采用占空比控制的方式，在程序里事先设定完成。

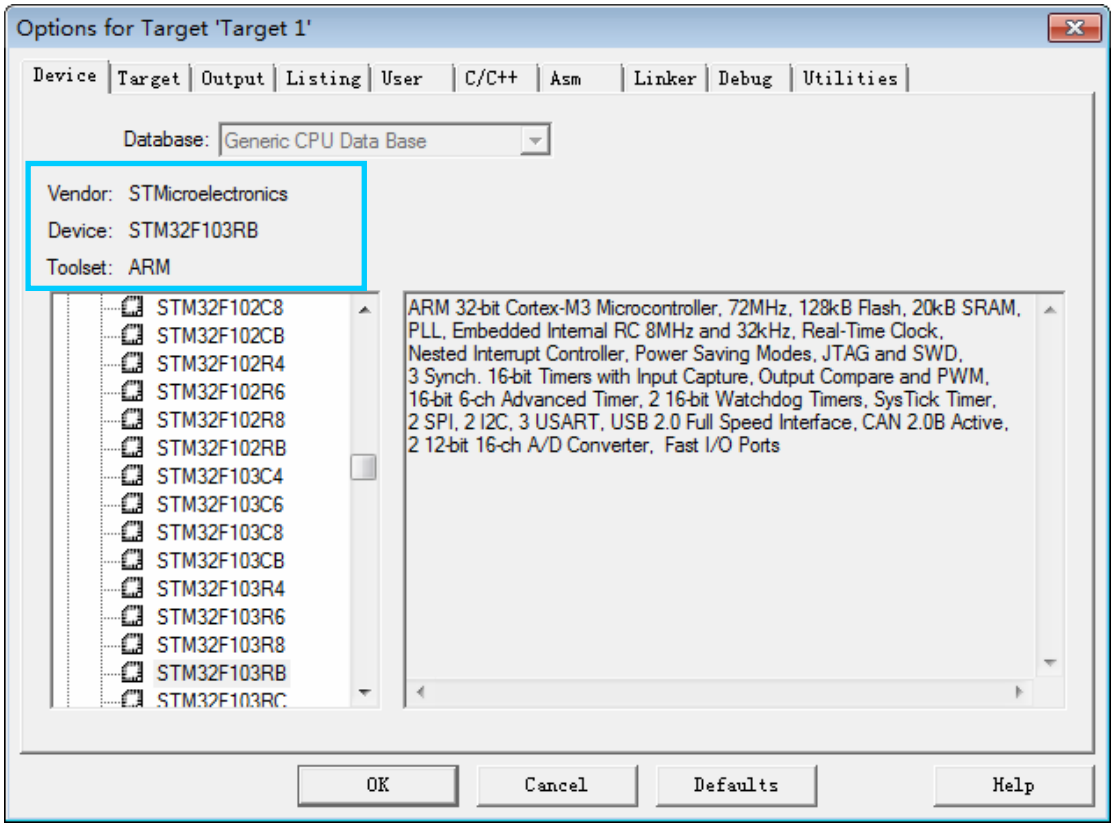
有刷驱动器相关的状态和控制寄存器地址说明如下：

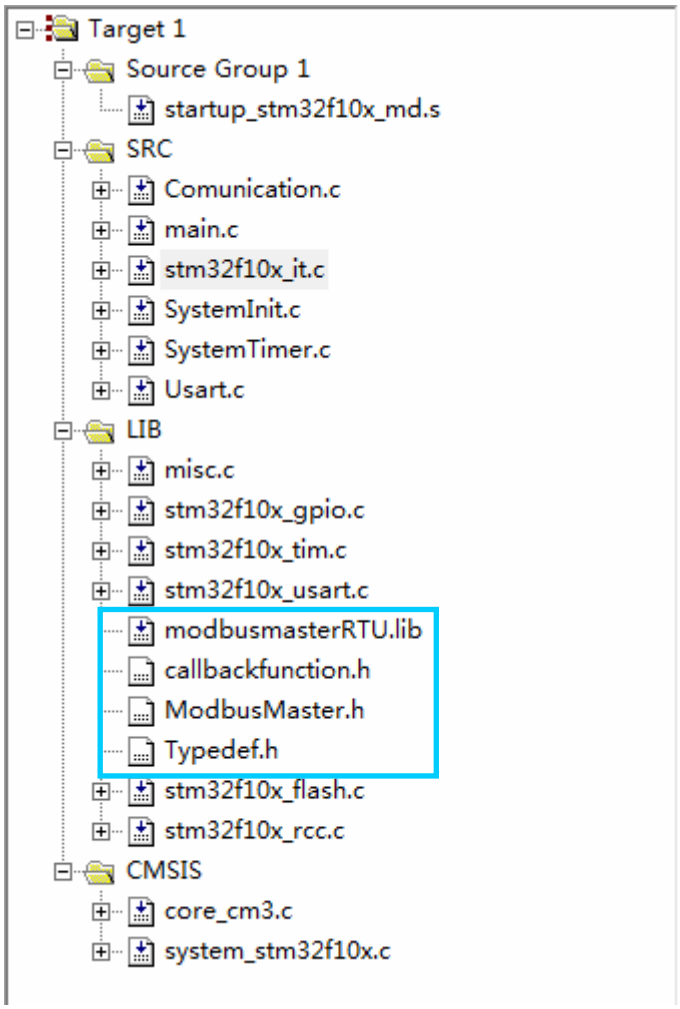
寄存器地址（16 进制）	**描述**	**说明**
0x40	设置占空比	-1000~1000，数值乘以 0.1 为目标占空比
0x14	AI1 端口电压	0~1000，数值乘以 0.01 为电压值，单位为 V
0x15	AI2 端口电压	0~1000，数值乘以 0.01 为电压值，单位为 V

3、控制代码实现分为以下几个步骤：
1）添加 modbus 协议库文件；
2）基于 STM32 平台对串口驱动和系统定时器等底层驱动函数进行编写实现；
3）添加 main 主函数文件，实现 485 通信控制应用示例代码。

下面按步骤进行详细说明。
3-1 添加 modbus 协议库文件。
3-1-1 打开 keil uVision4 软件，根据使用的 stm32 单片机型号建立相应的工程文件，将库文件复制到工程目录下，然后在项目工程文件夹内添加 modbusmasterRTU 库文件。




Options for Target 'Target 1'
Device
Target
Output
Listing
User
C/C++
Asm
Linker
Debug
Utilities
Database: Generic CPU Data Base
Vendor: STMicroelectronics
Device: STM32F103RB
Toolset: ARM
STM32F102C8
STM32F102CB
STM32F102R4
STM32F102R6
STM32F102R8
STM32F102RB
STM32F103C4
STM32F103C6
STM32F103C8
STM32F103CB
STM32F103R4
STM32F103R6
STM32F103R8
STM32F103RB
STM32F103RC
ARM 32-bit Cortex-M3 Microcontroller, 72MHz, 128kB Flash, 20kB SRAM,
PLL, Embedded Internal RC 8MHz and 32kHz, Real-Time Clock,
Nested Interrupt Controller, Power Saving Modes, JTAG and SWD,
3 Synch. 16-bit Timers with Input Capture, Output Compare and PWM,
16-bit 6-ch Advanced Timer, 2 16-bit Watchdog Timers, SysTick Timer,
2 SPI, 2 I2C, 3 USART, USB 2.0 Full Speed Interface, CAN 2.0B Active,
2 12-bit 16-ch A/D Converter, Fast I/O Ports
OK
Cancel
Defaults
Help


Target 1
Source Group 1
startup_stm32f10x_md.s
SRC
Comunication.c
main.c
stm32f10x_it.c
SystemInit.c
SystemTimer.c
Usart.c
LIB
misc.c
stm32f10x_gpio.c
stm32f10x_tim.c
stm32f10x_usart.c
modbusmasterRTU.lib
callbackfunction.h
ModbusMaster.h
Typedef.h
stm32f10x_flash.c
stm32f10x_rcc.c
CMSIS
core_cm3.c
system_stm32f10x.c

3-2 基于 STM32 平台对串口驱动和系统定时器等底层驱动函数进行编写实现：

3-2-1 usart.c 串口驱动接口函数。

```
#include "Usart.h"
#include "MBMasterRtu.h"
#include "callbackfunction.h"

#define     DE1_H          GPIO_SetBits(GPIOA, GPIO_Pin_8)             // PA8 高电平
#define     DE1_L          GPIO_ResetBits(GPIOA, GPIO_Pin_8)     // PA8 低电平

#define     USART1_RX_EN      DE1_H

#define     USART1_TX_EN      DE1_L

static u32 BaudRate[8]={9600,19200,38400,56000,57600,115200,128000,256000};
static u16 ParityrAra[4]={USART_Parity_No,USART_Parity_Odd,USART_Parity_Even,USART_Parity_No};
static u16 DataBit[2]={USART_WordLength_9b,USART_WordLength_8b};
static u16 StopBit[2]={USART_StopBits_2,USART_StopBits_1};

static void USART1_GPIOConfiguration(void);

void COM_Init(void)
{
      USART1_ConfigurationParam(BaudRate[5]\
                                          ,DataBit[1]\
                                          ,StopBit[0]\
                                          ,ParityrAra[0]);

}

/*
*********************************************************************************
** 函数名称 ：  USART1_GPIOConfiguration(void)
** 函数功能 ：  端口初始化
** 输    入    ：  无
** 输    出    ：  无
** 返    回    ：  无
*********************************************************************************
*/
static void USART1_GPIOConfiguration(void)
{
      GPIO_InitTypeDef GPIO_InitStructure;
      RCC_APB2PeriphClockCmd( RCC_APB2Periph_GPIOA , ENABLE   );
      /* Configure USART1 Tx (PA.09) as alternate function push-pull */
      GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;                              // 选中管脚 9
```

```
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;         // 复用推挽输出
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;            // 最高输出速率 50MHz
    GPIO_Init(GPIOA, &GPIO_InitStructure);                  // 选择 A 端口

    /* Configure USART1 Rx (PA.10) as input floating */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;                  //选中管脚 10
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;       //浮空输入
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);                  //选择 A 端口

    /* Configure DE1 */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8 ;
    GPIO_InitStructure.GPIO_Mode =      GPIO_Mode_Out_PP;       // 推挽输出
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;       // 最高输出速率 50MHz
    GPIO_Init(GPIOA, &GPIO_InitStructure);                  // 选择 C 端口

    USART1_TX_EN ;
}

/*
*********************************************************************************
** 函数名称 ： USART1_ConfigurationParam(u32 baudrate,u16 databit,u16 stopbit,u16 parity)
** 函数功能 ： 串口 1 初始化
** 输    入    ： 无
** 输    出    ： 无
** 返    回    ： 无
*********************************************************************************
*/
void USART1_ConfigurationParam(u32 baudrate,u16 databit,u16 stopbit,u16 parity)

{

    NVIC_InitTypeDef NVIC_InitStructure;
    USART_InitTypeDef USART_InitStructure;
    USART_ClockInitTypeDef  USART_ClockInitStructure;

    USART1_GPIOConfiguration();

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE  );



    USART_ClockInitStructure.USART_Clock = USART_Clock_Disable;             // 时钟低电平活动
    USART_ClockInitStructure.USART_CPOL = USART_CPOL_Low;                   // 时钟低电平
```

```
    USART_ClockInitStructure.USART_CPHA = USART_CPHA_2Edge;                    // 时钟第二个边沿进行数据捕获
    USART_ClockInitStructure.USART_LastBit = USART_LastBit_Disable;        // 最后一位数据的时钟脉冲不从 SCLK 输出
    /* Configure the USART1 synchronous paramters */
    USART_ClockInit(USART1, &USART_ClockInitStructure);

    USART_InitStructure.USART_BaudRate =baudrate;                                    // 波特率为：115200
    USART_InitStructure.USART_WordLength =databit;                    // 8 位数据
    USART_InitStructure.USART_StopBits = stopbit;                    // 在帧结尾传输 1 个停止位
    USART_InitStructure.USART_Parity =parity;                    // 奇偶失能


    USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;   // 硬件流控制失能

    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;          // 发送使能+接收使能
    /* Configure USART1 basic and asynchronous paramters */
    USART_Init(USART1, &USART_InitStructure);

    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);
    NVIC_InitStructure.NVIC_IRQChannel=USART1_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority=0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority=1;
    NVIC_InitStructure.NVIC_IRQChannelCmd=ENABLE;
    NVIC_Init(&NVIC_InitStructure);

      /* Enable USART1 */
    USART_ClearFlag(USART1, USART_IT_RXNE);                    //清中断，以免一启用中断后立即产生中断
    USART_ITConfig(USART1,USART_IT_RXNE, ENABLE);          //使能 USART1 中断源
    USART_ITConfig(USART1,USART_IT_TC,ENABLE);
    USART_Cmd(USART1, ENABLE);

}


u16 USART1_RecData(void)
{

    return USART1->DR;
```

```
}

void USART1_SendData(u16 data)
{

    USART1_TX_EN;
    USART_SendData(USART1,data);

}
//控制 485 芯片接收和发送使能切换
void USART1_RXTXSwitch(bool rxsta,bool txsta)
{
    if(rxsta)
        {
            USART_ITConfig(USART1,USART_IT_RXNE, ENABLE);          //使能 USART1 中断源
            USART1_RX_EN;
        }
    else
        {
            USART_ITConfig(USART1,USART_IT_RXNE, DISABLE);         //取消使能 USART1 中断
源
            USART1_TX_EN;
        }

    if(txsta)
        {
            USART_ITConfig(USART1,USART_IT_TC, ENABLE);        //使能 USART1 中断源
            USART1_TX_EN;
        }
    else
        {
            USART_ITConfig(USART1,USART_IT_TC, DISABLE);           //取消使能 USART1 中断
源
            USART1_RX_EN;
        }

}

/*
*********************************************************************************
** 函数名称 ： USART1_IRQHandler(void)
** 函数功能 ： 串口 1 中断处理函数
** 输       入 ： 无
** 输       出 ： 无
```

```
** 返       回 ： 无
*******************************************************************************
*/
void USART1_IRQHandler(void)
{
      if(USART_GetITStatus(USART1,USART_IT_RXNE)!=RESET)
      {
            USART_ClearITPendingBit(USART1,USART_IT_RXNE);
            USART_ClearFlag(USART1, USART_IT_RXNE);
            MBMR_RecFrame();

      }

      if(USART_GetITStatus(USART1,USART_IT_TC)!=RESET)
      {
            USART_ClearITPendingBit(USART1,USART_IT_TC);
            USART_ClearFlag(USART1, USART_IT_TC);
            MBMR_SendFrame();
      }

}
```

3-2-2 systemtimer.c 系统软定时器驱动接口函数。

```
#include "SystemTimer.h"
#include "callbackfunction.h"



static u32 g_uiSYSTIME=0;

static void TIM1_Configuration(void);

static void TIM1_Configuration(void)
{

  TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
  NVIC_InitTypeDef NVIC_InitStructure;

  RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1, ENABLE );
   /* Time Base configuration */
  TIM_DeInit(TIM1);

  TIM_TimeBaseStructure.TIM_Prescaler = 71;                    //设置预分频器分频系数 71，即 APB2=72M,
TIM1_CLK=72/72=1MHz ，
                                                               //它的取值必须在 0x0000 和 0xFFFF 之间
```

```
  TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up; // 向上计数模式
  TIM_TimeBaseStructure.TIM_Period = 1000;                    // 1ms 定时，计数器向上计数到 1000 后产生更新事件，计数值归零
  TIM_TimeBaseStructure.TIM_ClockDivision = 0x0;              //设置了定时器时钟分割，
  TIM_TimeBaseStructure.TIM_RepetitionCounter = 0x0;          // 设置了周期计数器值，它的取值必须在 0x00 和 0xFF 之间。

  TIM_TimeBaseInit(TIM1,&TIM_TimeBaseStructure);              // 根据 TIM_TimeBaseInitStruct 中指定的参数初始化 TIMx 的时间基数单位

  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);

  NVIC_InitStructure.NVIC_IRQChannel =TIM1_UP_IRQn;     //更新事件
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;    //抢占优先级 0
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;           //响应优先级 2
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;              //允许中断
  NVIC_Init(&NVIC_InitStructure);

  TIMER_ArrayInit();
  Timer1_Enable();



}
void Timer1_Enable(void)
{

      TIM_ClearFlag(TIM1, TIM_FLAG_Update);       //清中断，以免一启用中断后立即产生中断
      TIM_SetCounter(TIM1,0);
      TIM_ITConfig(TIM1, TIM_IT_Update, ENABLE); //使能 TIM1 中断源

      TIM_Cmd(TIM1, ENABLE);
}
void Timer1_Disable(void)
{
      TIM_ClearFlag(TIM1, TIM_FLAG_Update);       //清中断，以免一启用中断后立即产生中断
      TIM_SetCounter(TIM1,0);
      TIM_ITConfig(TIM1, TIM_IT_Update, DISABLE); //使能 TIM1 中断源

      TIM_Cmd(TIM1, DISABLE);
}

void Timer1_Init(void)
{
      TIM1_Configuration();
```

```
}


u32 SYST_GetSystemTime(void)
{
	return g_uiSYSTIME;
}

void TIM1_UP_IRQHandler(void)
{
	TIM_ClearITPendingBit(TIM1, TIM_FLAG_Update); //清中断
	g_uiSYSTIME++;
	TIMER_Servic_Timer1_ISR();


}
extern void TIM1_INTDisable(void)
{
	TIM_ITConfig(TIM1, TIM_IT_Update, DISABLE);
}
extern void TIM1_INTEnable(void)
{
	TIM_ITConfig(TIM1, TIM_IT_Update, ENABLE);
}
```

3-2-3 systeminit.c 系统时钟初始化与接口初始化函数。

```
#include "SystemInit.h"
#include "comunication.h"
#include "callbackfunction.h"

void RCC_Configuration(void);

void RCC_Configuration(void)
{
	ErrorStatus HSEStartUpStatus;
  /* RCC system reset(for debug purpose) */
  RCC_DeInit();

  /* Enable HSE */
  RCC_HSEConfig(RCC_HSE_ON);

  /* Wait till HSE is ready */
  HSEStartUpStatus = RCC_WaitForHSEStartUp();
```

```
  if(HSEStartUpStatus == SUCCESS)
  {
    /* HCLK = SYSCLK */
    RCC_HCLKConfig(RCC_SYSCLK_Div1);

    /* PCLK2 = HCLK */
    RCC_PCLK2Config(RCC_HCLK_Div1);

    /* PCLK1 = HCLK/2 */
    RCC_PCLK1Config(RCC_HCLK_Div2);

    /* Flash 2 wait state */
    FLASH_SetLatency(FLASH_Latency_2);
    /* Enable Prefetch Buffer */
    FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);

    /* PLLCLK = 8MHz * 9 = 72 MHz */
    RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);

    /* Enable PLL */
    RCC_PLLCmd(ENABLE);

    /* Wait till PLL is ready */
    while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET)
    {
    }

    /* Select PLL as system clock source */
    RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);

    /* Wait till PLL is used as system clock source */
    while(RCC_GetSYSCLKSource() != 0x08)
    {
    }
  }
}
void System_Init(void)
{
      RCC_Configuration();
      Modbus_Init();
}
```

3-3 添加 main 主函数文件，实现 485 通信控制应用示例代码。

3-3-1 建立一个 comunication.c 通信文件，调用 modbus 库的主任务函数，该函数最后会被 main 函数调用，成为系统的一个常规任务函数。

```
#include "Comunication.h"
#include "ModbusMaster.h"

void COM_Task(void)
{

	MB_MasterTask();

}
```

3-3-2 添加 main.c 主任务函数文件。实现通过 485 通信方式对电机驱动器进行寄存器数据读取和写操作。

```
#include "stm32f10x.h"
#include "ModbusMaster.h"
#include "Comunication.h"
#include "SystemInit.h"

#define DEMOBLS		1
#define DEMONS		1

//无刷电机演示，通信波特率 115200，IN2 下降沿停车，IN3 下降沿切换转动方向
static void DemoBLS(void)
{

	static int state=2;
	u16 tempdata[6]={1,1,1,1,1,1};
	static int value=500;//value 大于零正转，小于零反转
	u16 pwm=(0xffff)&value;
		switch(state)
			{
				case 0:
					if(MB_GetMasterEvent()==MM_EVENT_NONE)//检查是否为空闲状态
						{

							if(MB_WirteSingleReg(1,0x42,pwm))//发送转动指令
								{
									state++;
								}


						}
```

```
                    break;
                case 1:
                    if(MB_GetMasterEvent()==MM_EVENT_NONE)
                        {
                            if(MB_GetMasterErrCode()==MB_ERR_NONE)//发送是否成功
                                {
                                    state++;
                                }
                            else
                                {

                                    state--;
                                }

                        }
                    break;
                case 2:
                    if(MB_GetMasterEvent()==MM_EVENT_NONE)
                        {
                            if(MB_ReadHoldReg(1,0x7028,6))//读取 IN2 ,IN3 边沿寄存器
                                {
                                    state++;
                                }
                        }
                    break;
                case 3:
                    if(MB_GetMasterEvent()==MM_EVENT_NONE)
                        {
                            if(MB_GetMasterErrCode()==MB_ERR_NONE)
                                {
                                    MB_GetRecData(tempdata,0,6);//获取返回数据
                                    if(tempdata[0]==0)//IN2 下降沿
                                        {
                                            MB_WirteSingleReg(1,0x40,0);//发送停止
指令

                                            state--;
                                        }
                                    else if(tempdata[5]==0)//IN3 下降沿切换
                                        {
                                            value=value*(-1);//切换转动方向
                                            state=0;
                                        }
                                    else
```

```
                                                {
                                                    state--;
                                                }
                                            }
                                        else
                                            {
                                                state--;
                                            }
                                    }
                                break;
                            default:
                                break;
                        }
}



//有刷电机演示，通信波特率 115200，AI1 高电平停车，AI2 高电平切换转动方向

static void DemoNS(void)
{

    static int state=2;
    u16 tempdata[2]={1,1};
    static int value=500;//value 大于零正转，小于零反转
    u16 pwm=(0xffff)&value;
            switch(state)
                {
                    case 0:
                        if(MB_GetMasterEvent()==MM_EVENT_NONE)
                            {

                                if(MB_WirteSingleReg(1,0x40,pwm))//发送转动指令
                                    {
                                        state++;
                                    }


                            }

                        break;
                    case 1:
                        if(MB_GetMasterEvent()==MM_EVENT_NONE)
```

```
                {
                    if(MB_GetMasterErrCode()==MB_ERR_NONE)
                        {
                            state++;
                        }
                    else
                        {

                            state--;
                        }

                }
            break;
        case 2:
            if(MB_GetMasterEvent()==MM_EVENT_NONE)
                {
                    if(MB_ReadHoldReg(1,0x14,2))//读取 AI1,AI2 电压寄存器
                        {
                            state++;
                        }
                }
            break;
        case 3:
            if(MB_GetMasterEvent()==MM_EVENT_NONE)
                {
                    if(MB_GetMasterErrCode()==MB_ERR_NONE)
                        {
                            MB_GetRecData(tempdata,0,2);//获取返回数据
                            if(tempdata[0]>=300)//AI1 电压大于 3000mv 停止
                                {
                                    value=0;
                                    state=0;
                                }
                            else if(tempdata[1]>=300)//AI2 电压大于 3000mv 切换
                                {
                                    if(value==0)
                                        {
                                            value=500;
                                        }
                                    else
                                        {
                                            value=value*(-1);//切换转动方
向

                                        }
```

```
                                                    state=0;
                                                }
                                            else
                                                {
                                                    state--;
                                                }
                                        }
                                    else
                                        {
                                            state--;
                                        }
                                }
                            break;

                        default:
                            break;
                    }
}


int main(void)
{
    System_Init();
    while(1)
    {
        #if DEMOBLS
        DemoBLS();
        #else
        DemoNS();
        #endif
        COM_Task();

    }

}
```

End