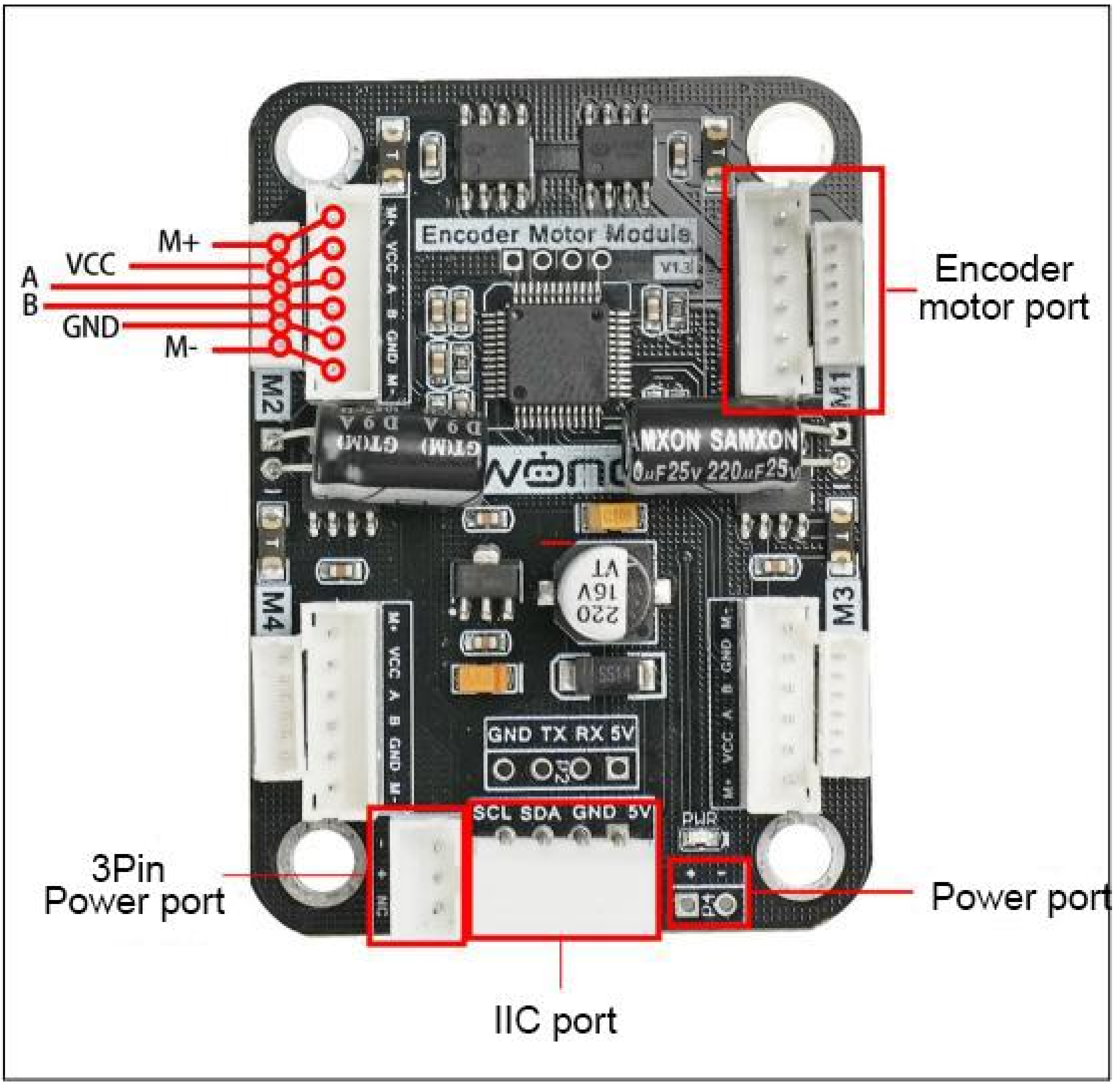
Encoder Motor Driver Module User Manual

1. Overview

1.1 Module Introduction

This motor drive module can be used with a microcontroller to drive TT motors or magnetic encoder motors. Its voltage range is DC 3V-12V, depending on the voltage applied to the motor. The interface distribution is shown in the figure below.



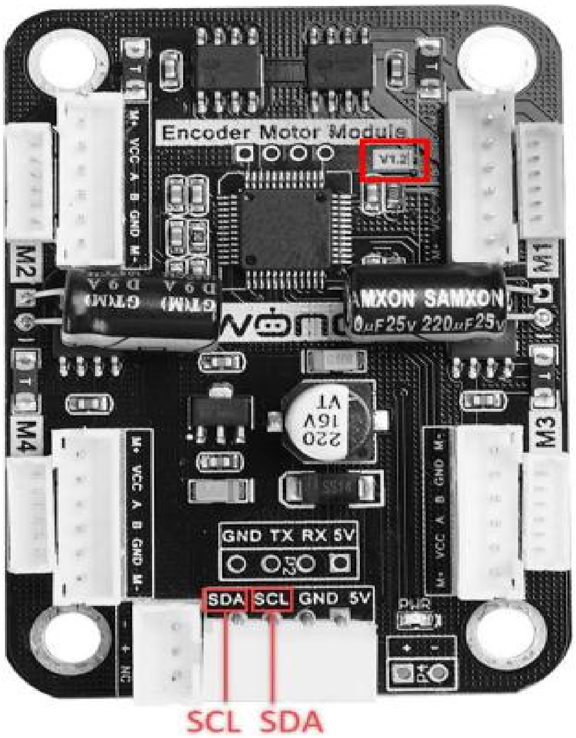
Interface Type	Interface	Instruction
Encoder motor Interface	GND	Hall sensor power supply negative terminal
	A	Phase A pulse signal output terminal
	B	Phase B pulse signal output terminal
	VCC	Hall sensor power supply positive terminal
	M+	Motor power supply positive terminal
	M-	Motor power supply negative terminal
	Note: 1. The voltage between VCC and GND depends on the voltage applied to microcontroller, which uses 3.3V or 5V in general. 2. When the spindle rotates clockwise, the output pulse of channel A is ahead of channel B. Conversely, when the spindle rotates counterclockwise, the signal of channel A is behind that of channel B. 3. The voltage between M+ and M- depends on the voltage applied to the motor.	
	SCL	Clock wire
	SDA	bi-directional data bus

IIC Port	GND	Power ground cable
	5V	5V DC output
3Pin Power Port	-	Power negative pole
	+	Power positive pole
	nc	-
Power Port	+	Power positive pole
	-	Power negative pole

Note:

1) The voltage of power supply is low using IIC port, which may lead to instability in the motor drive functionality.Therefore, it is recommended to use an independent power supply of 6V-9V.

2) For version 1.2, there is a silk printing error on the SDA and SCL ports ((subsequent versions have been corrected). The correct position of SCL and SDA are marked in the below figure.

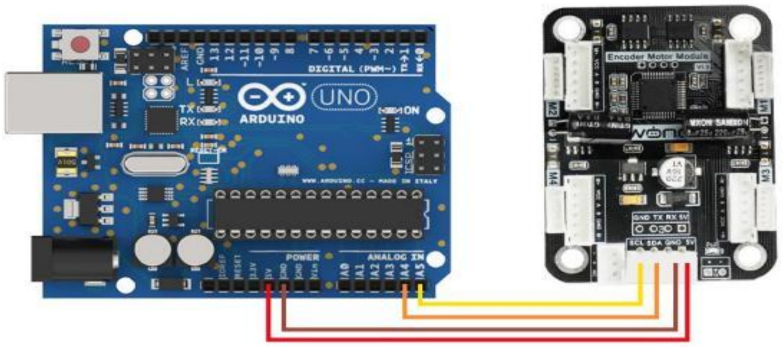


1.2 Wiring Instruction

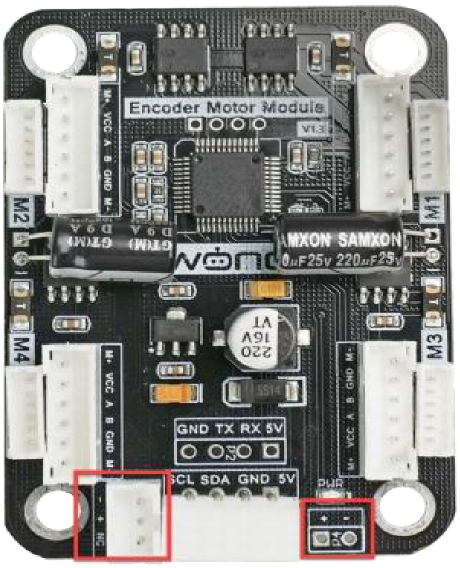
1.2.1 Power Supply

As an example of using I2C communication with an Arduino UNO board. The method of power supply and corresponding wiring is shown as follow:

1) Supply power through IIC interface. Correspondingly connect the 5V pin on Arduino UNO board to 5V port on motor driver board, GND to GND, A4 to SCL, A5 to SDA with male-to-female dupond wire, as the figure shown below:

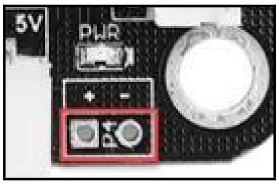


2) Use an independent power supply.You can connect to any corresponding port indicated by red markings in the below figure to supply power to the motor driver.

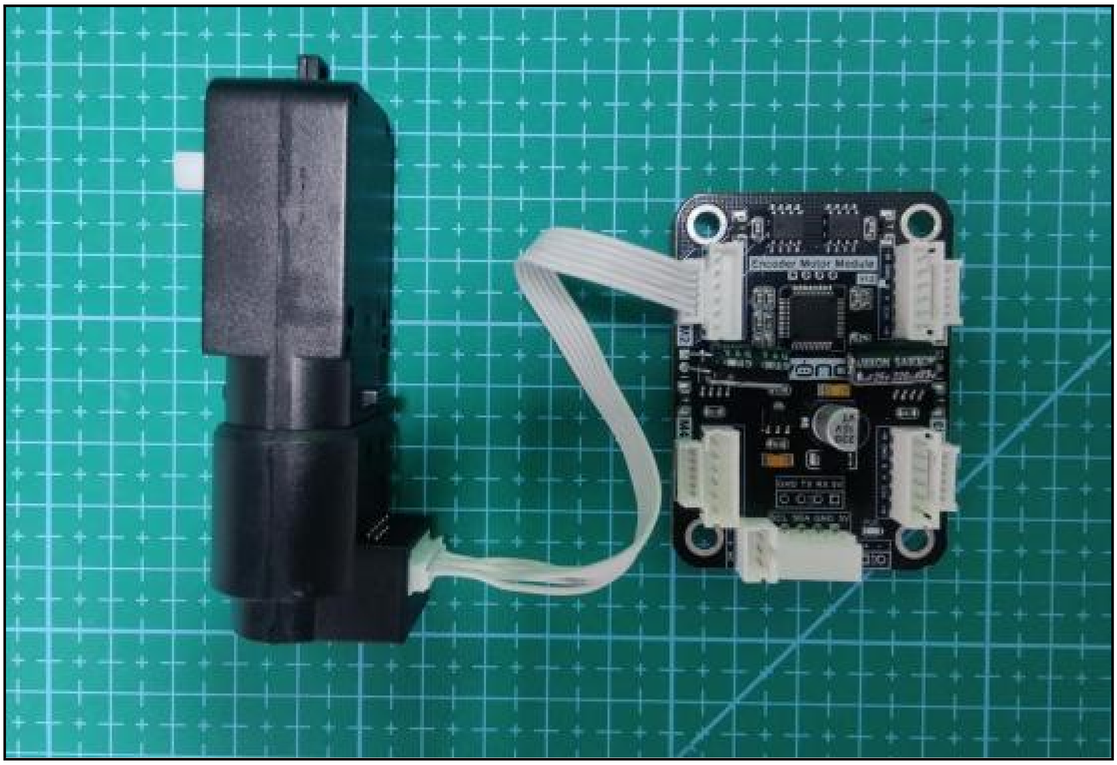


3) Wiring instruction:

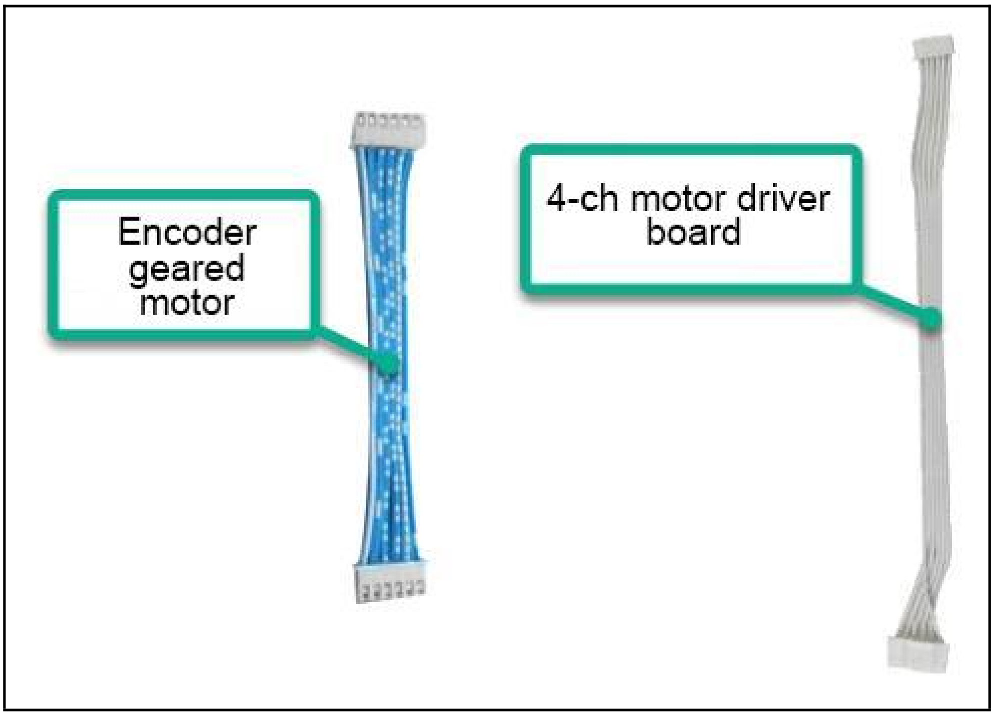
When using the power interface for power supply, it is necessary to solder the pins before proceeding with the power supply. The soldering position is shown as follow:



1.2.2 Motor Wiring



When you purchase the encoder motor and motor driver separately, you will get two different wires, as the figure shown below:



When connecting with encoder motor, you must use the white wire. Do not use the blue one, it will burn out motors.

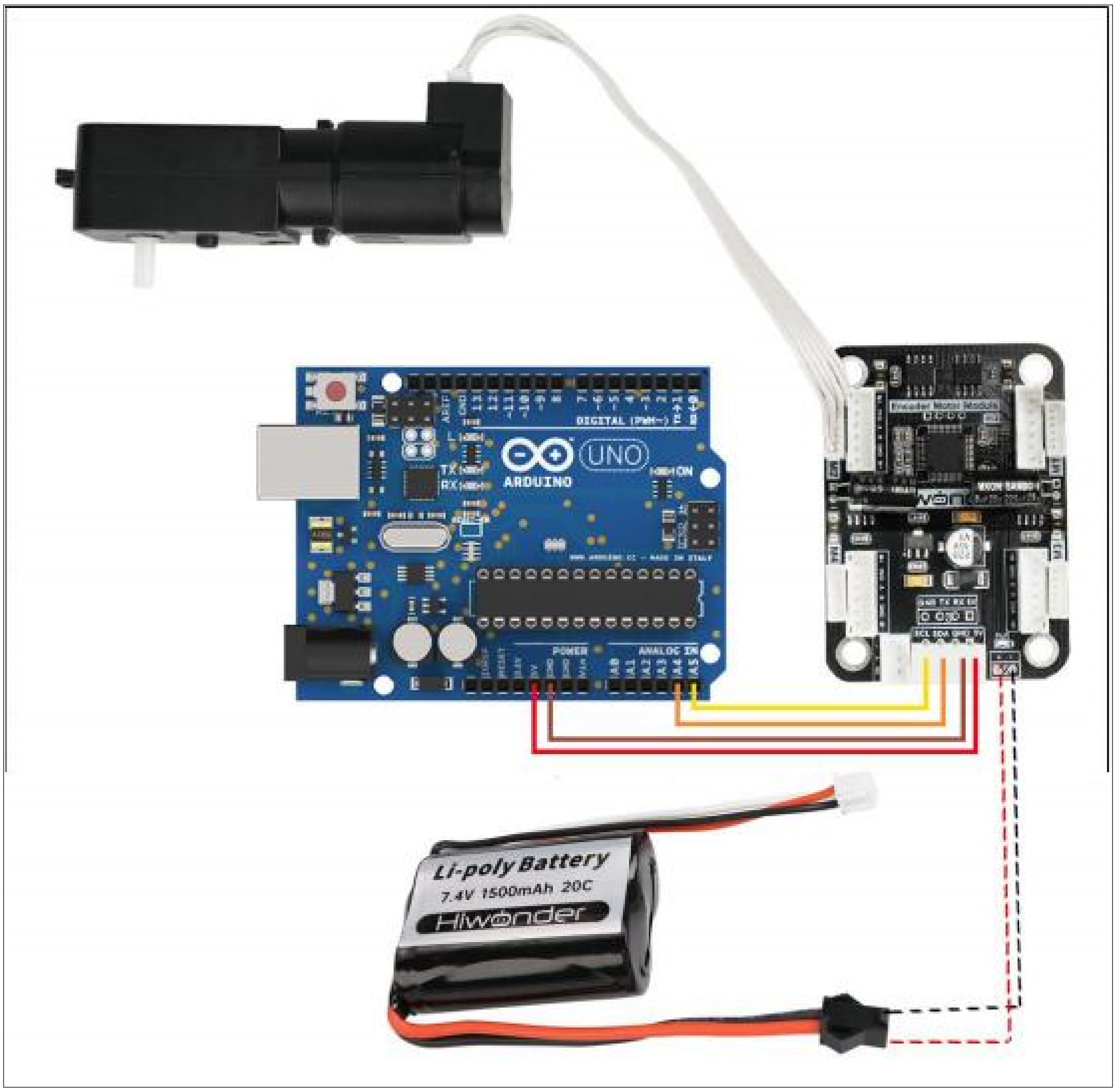
2. Hands on Project

2.1 Arduino Development

2.1.1 Preparation

1) Hardware

The part will use Arduino UNO board to control the circuit as an example. The specific wiring refer to the following diagram:



2) Software

Please refer to “**Arduino development environment**”to install and debug Arduino software.

Note:

1) It is required to solder the pins onto the power port. Then use male-to-female cable to connect battery. “5V” is connected to “5V” and “GND” to “GND”.

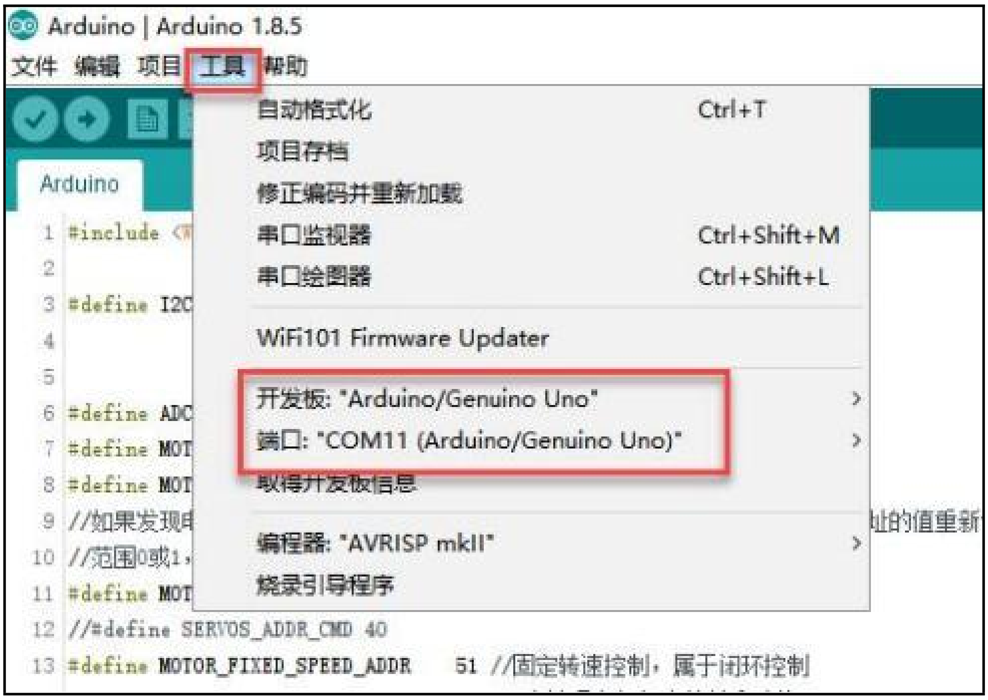
2) This wiring diagram is only for your reference. You can refer to this method to connect multiple motors.

2.1.2 Project Process

1) Connect Arduino UNO development board to your computer using USB cable.

2) Open Arduino program in folder.

3) Select the correct development board and corresponding port. (Here uses UNO corresponding to port 11. Please select the corresponding board model and connection port according to the actual situation.)



开发板: "Arduino/Genuino Uno"

端口: "COM11 (Arduino/Genuino Uno)"

```
#include <W
#define I2C
#define ADC
#define MOT
#define MOT
//如果发现电
//范围0或1，
#define MOT
//#define SERVOS_ADDR_CMD 40
#define MOTOR_FIXED_SPEED_ADDR     51 //固定转速控制，属于闭环控制
```

4) Click on ➔ to upload program to Arduino UNO board.

2.1.3 Project Outcome

The motor rotating forwards, then reverses in a continuous loop.

2.1.4 Sample Code

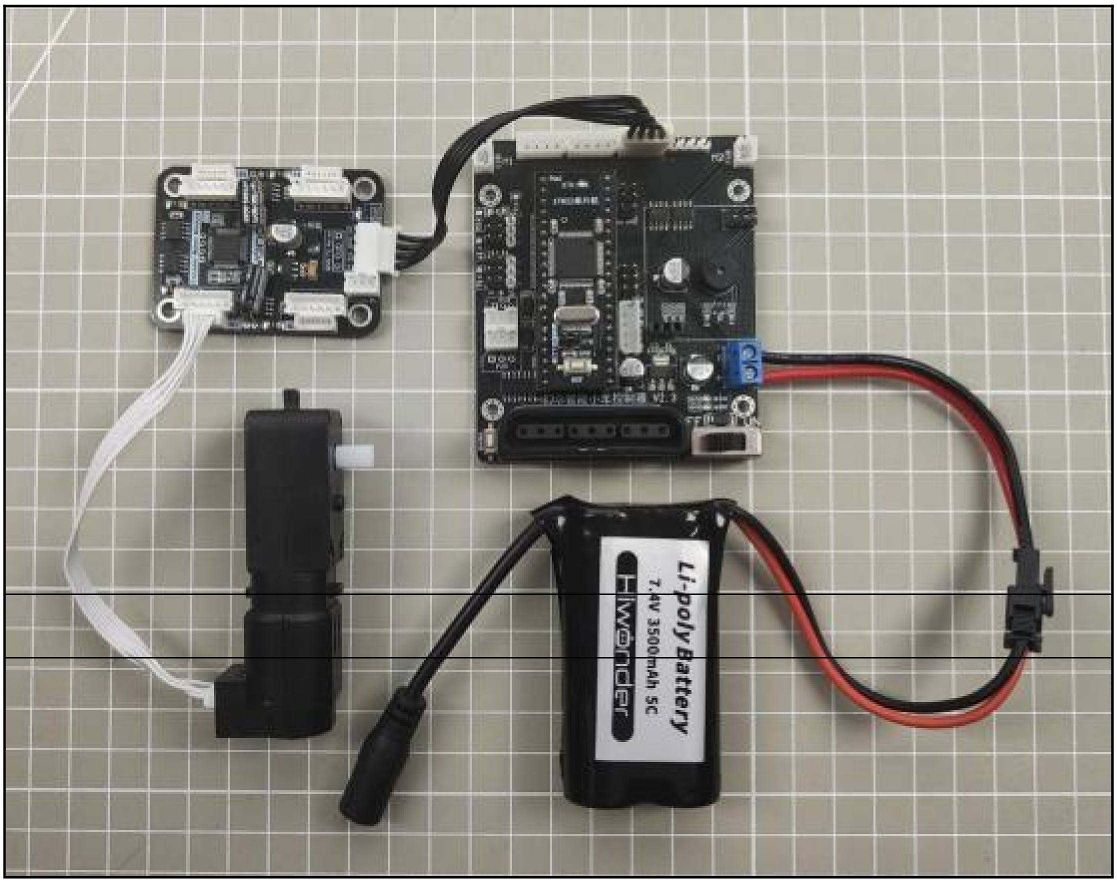
Please refer to “**4. Appendix-> 4.2 Arduino Sample Code**.”

2.2 STM32 Development

2.2.1 Preparation

1) Hardware

An encoder motor is connected to the motor driver board. Then IIC communication will be established through an STM32 microcontroller.



Note: Only a single motor is connected in the above picture, but you can refer to the same method for multiple motors.’’

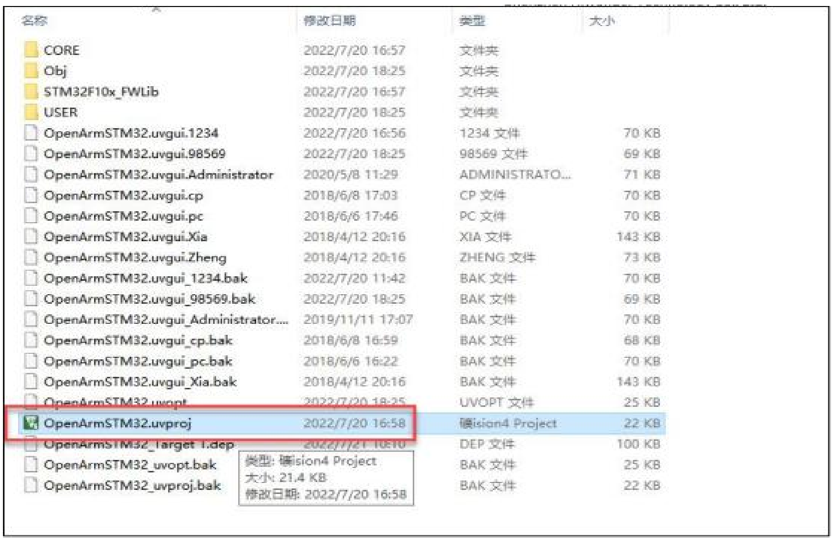
2) Software

Refer to the operation steps in “**Build STM32 Development Environment**” to install and debug the Keil tool.

2.2.2 Project Process

1) Connect the STM microcontroller to any USB port on your computer through USB-to-TTL converter.

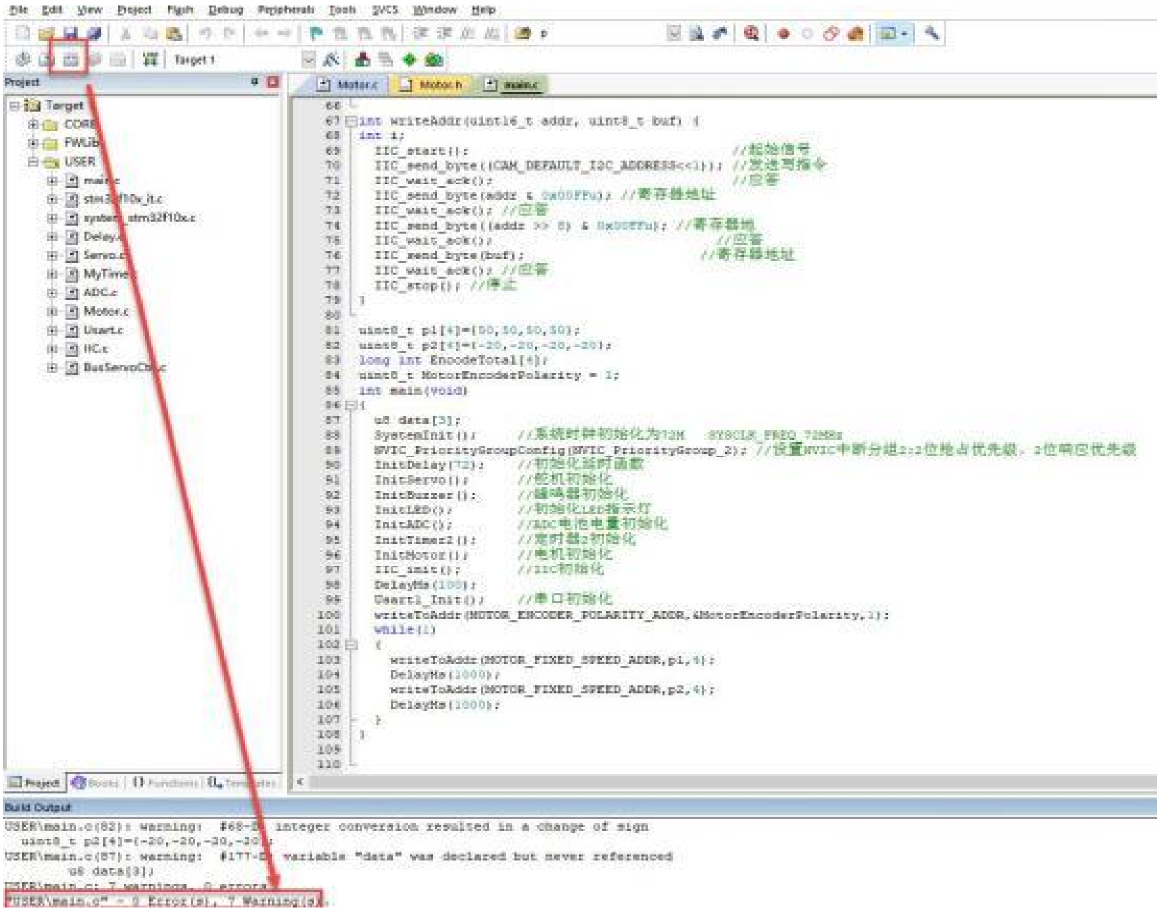
2) Open the STM32 program in folder.



名称	修改日期	类型	大小
CORE	2022/7/20 16:57	文件夹	
Obj	2022/7/20 18:25	文件夹	
STM32F10x_FWLib	2022/7/20 16:57	文件夹	
USER	2022/7/20 18:25	文件夹	
OpenArmSTM32.uvgui.1234	2022/7/20 16:56	1234 文件	70 KB
OpenArmSTM32.uvgui.98569	2022/7/20 18:25	98569 文件	69 KB
OpenArmSTM32.uvgui.Administrator	2020/5/8 11:29	ADMINISTRATO...	71 KB
OpenArmSTM32.uvgui.cp	2018/6/8 17:03	CP 文件	70 KB
OpenArmSTM32.uvgui.pc	2018/6/6 17:46	PC 文件	70 KB
OpenArmSTM32.uvgui.Xia	2018/4/12 20:16	XIA 文件	143 KB
OpenArmSTM32.uvgui.Zheng	2018/4/12 20:16	ZHENG 文件	73 KB
OpenArmSTM32.uvgui_1234.bak	2022/7/20 11:42	BAK 文件	70 KB
OpenArmSTM32.uvgui_98569.bak	2022/7/20 18:25	BAK 文件	69 KB
OpenArmSTM32.uvgui_Administrator....	2019/11/11 17:07	BAK 文件	70 KB
OpenArmSTM32.uvgui_cp.bak	2018/6/8 16:59	BAK 文件	68 KB
OpenArmSTM32.uvgui_pc.bak	2018/6/6 16:22	BAK 文件	70 KB
OpenArmSTM32.uvgui_Xia.bak	2018/4/12 20:16	BAK 文件	143 KB
OpenArmSTM32.uvopt	2022/7/20 18:25	UVOPT 文件	25 KB
OpenArmSTM32.uvproj	2022/7/20 16:58	礦ision4 Project	22 KB
OpenArmSTM32_Target 1.dep	2022/7/21 10:10	DEP 文件	100 KB
OpenArmSTM32_uvopt.bak		BAK 文件	25 KB
OpenArmSTM32_uvproj.bak		BAK 文件	22 KB

类型: 礦ision4 Project
大小: 21.4 KB
修改日期: 2022/7/20 16:58

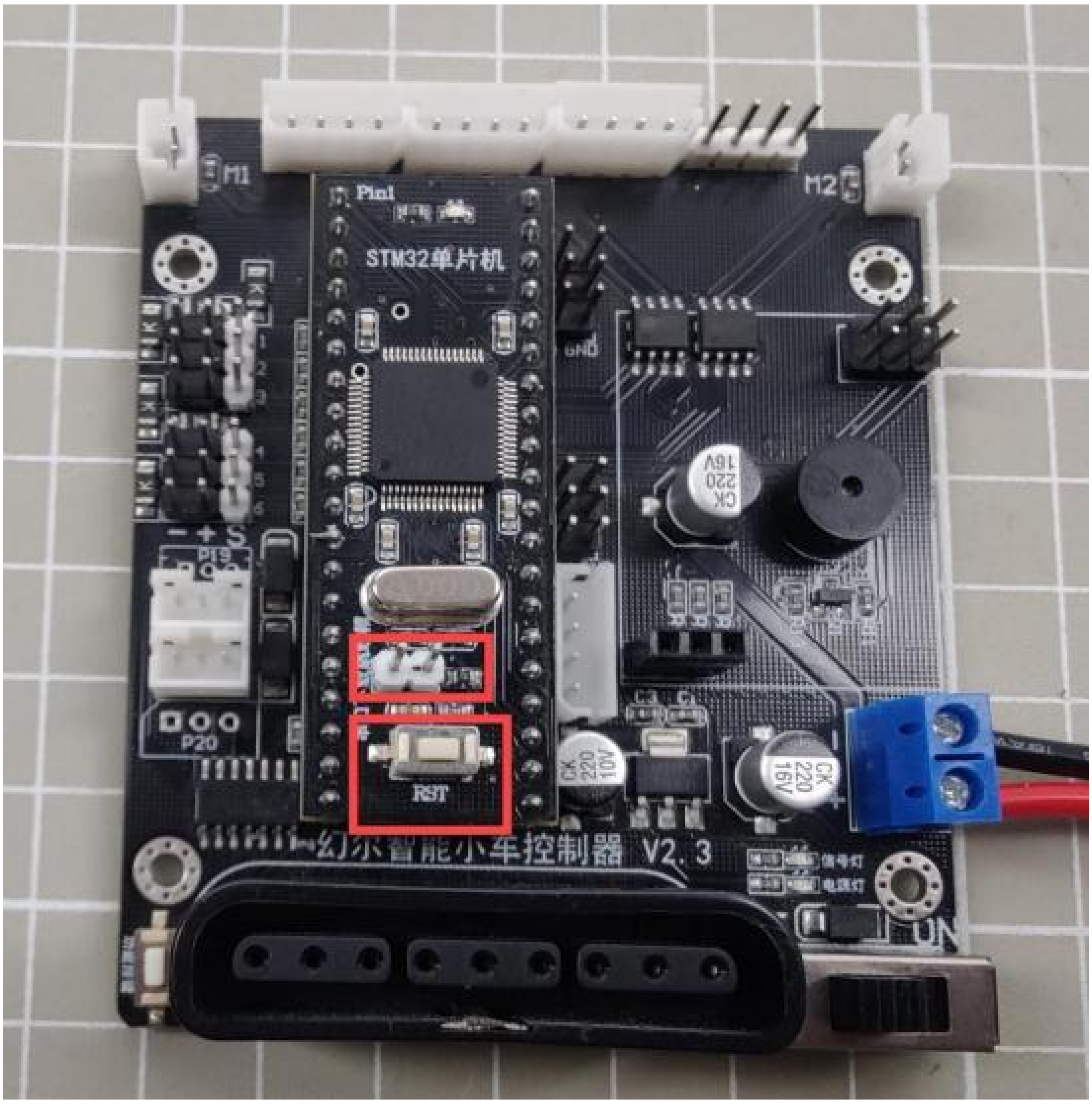
3) Compile all the code to generate an executable file.



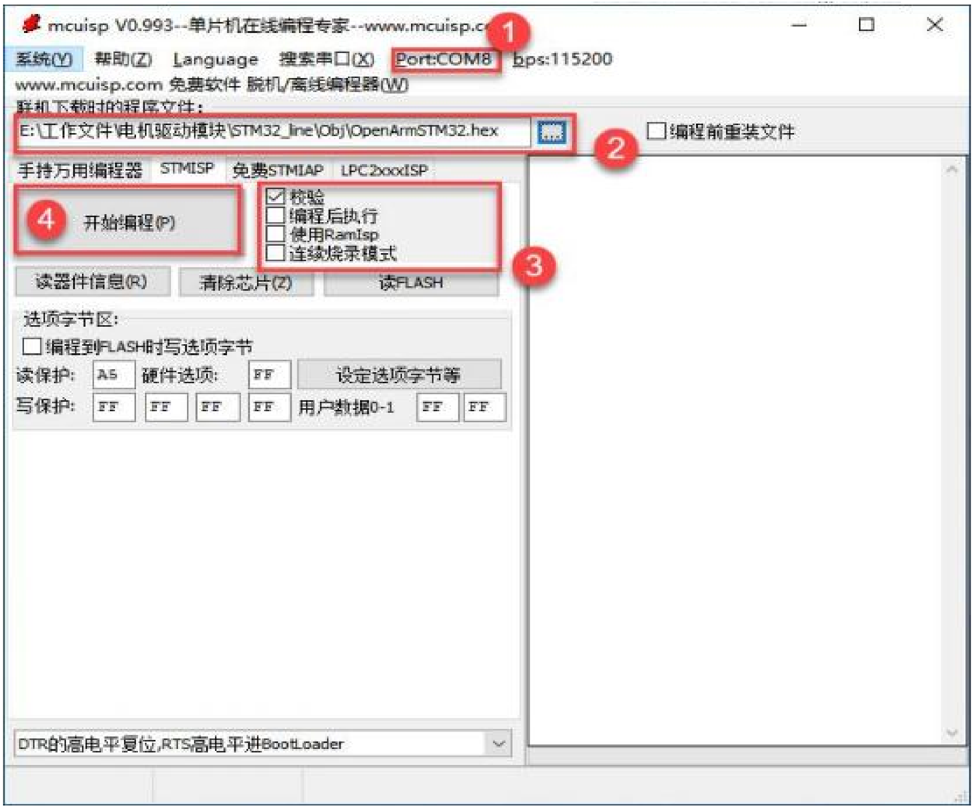
```
int writeAddr(uint16_t addr, uint8_t buf) {
int i;
  IIC_start();                                    //起始信号
  IIC_send_byte((CAM_DEFAULT_I2C_ADDRESS<<1)); //发送写指令
  IIC_wait_ack();                                 //应答
  IIC_send_byte(addr & 0x00FFu); //寄存器地址
  IIC_wait_ack(); //应答
  IIC_send_byte((addr >> 8) & 0x00FFu); //寄存器地
  IIC_wait_ack();                              //应答
  IIC_send_byte(buf);                        //寄存器地址
  IIC_wait_ack(); //应答
  IIC_stop(); //停止
}

uint8_t p1[4]={50,50,50,50};
uint8_t p2[4]={-20,-20,-20,-20};
long int EncodeTotal[4];
uint8_t MotorEncoderPolarity = 1;
int main(void)
{
  u8 data[3];
  SystemInit();       //系统时钟初始化为72M   SYSCLK_FREQ_72MHz
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2); //设置NVIC中断分组2:2位抢占优先级，2位响应优先级
  InitDelay(72);      //初始化延时函数
  InitServo();        //舵机初始化
  InitBuzzer();       //蜂鸣器初始化
  InitLED();          //初始化LED指示灯
  InitADC();          //ADC电池电量初始化
  InitTimer2();       //定时器2初始化
  InitMotor();        //电机初始化
  IIC_init();         //IIC初始化
  DelayMs(100);
  Usart1_Init();      //串口初始化
  writeToAddr(MOTOR_ENCODER_POLARITY_ADDR,&MotorEncoderPolarity,1);
  while(1)
  {
    writeToAddr(MOTOR_FIXED_SPEED_ADDR,p1,4);
    DelayMs(1000);
    writeToAddr(MOTOR_FIXED_SPEED_ADDR,p2,4);
    DelayMs(1000);
  }
}
```

```
USER\main.c(82): warning:  #68-D: integer conversion resulted in a change of sign
  uint8_t p2[4]={-20,-20,-20,-20};
USER\main.c(87): warning:  #177-D: variable "data" was declared but never referenced
        u8 data[3];
USER\main.c: 2 warnings, 0 errors
"USER\main.o" - 0 Error(s), 2 Warning(s).
```

4) Remove the jumper cap on the STM32 control board, and the push the RST (reset) button.



5) Open the program burning tool and select the correct port. Keep the same baud rate and download the program into the development board. The specific information refer to the below picture:


mcuisp V0.993--单片机在线编程专家--www.mcuisp.com
系统(Y) 帮助(Z) Language 搜索串口(X) Port:COM8 bps:115200
www.mcuisp.com 免费软件 脱机/高线编程器(W)
手持万用编程器 STMISP 免费STMIAP LPC2xxxISP
读器件信息(R) 清除芯片(Z) 读FLASH
1
2
3
4

联机下载时的程序文件：

E:\工作文件\电机驱动模块\STM32_line\Obj\OpenArmSTM32.hex

□编程前重装文件

开始编程(P)

☑校验
□编程后执行
□使用RamIsp
□连续烧录模式

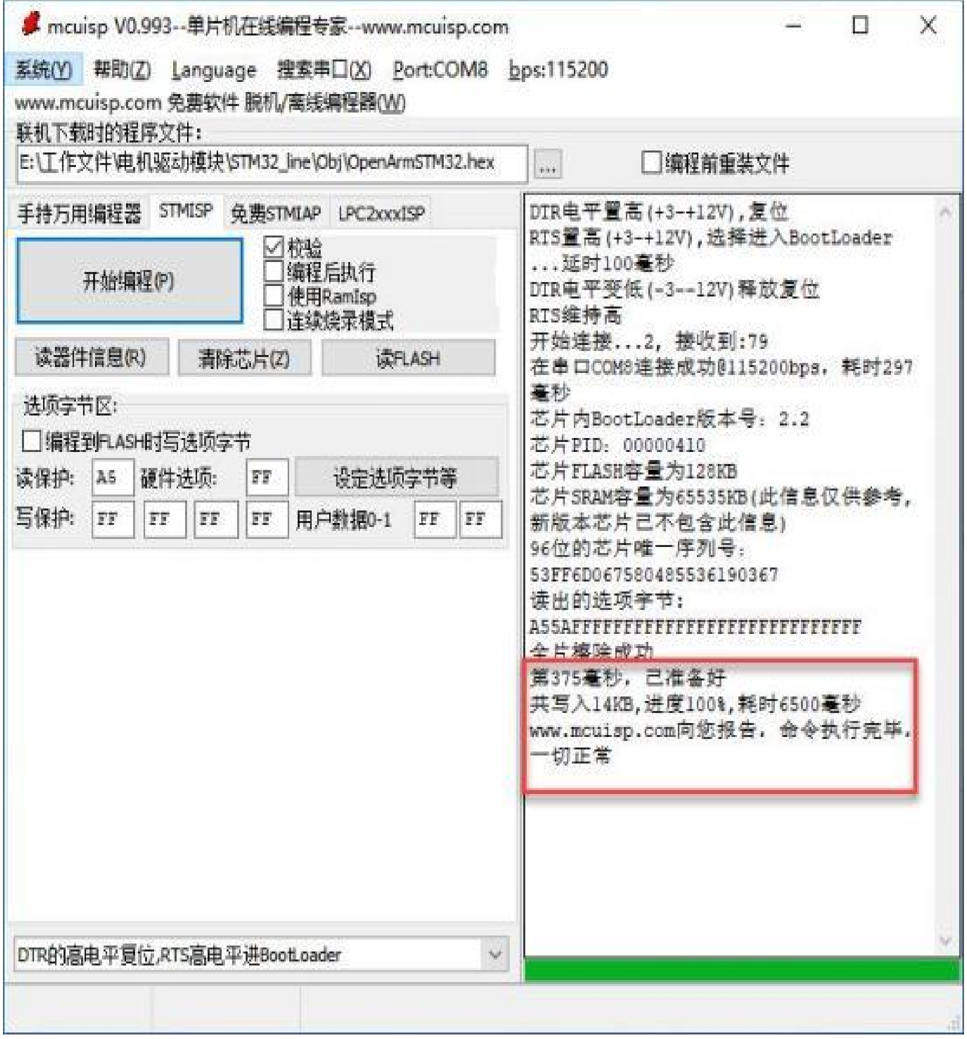
选项字节区：

□编程到FLASH时写选项字节

读保护：A5 硬件选项：FF 设定选项字节等

写保护：FF FF FF FF 用户数据0-1 FF FF

DTR的高电平复位,RTS高电平进BootLoader


mcuisp V0.993--单片机在线编程专家--www.mcuisp.com
系统(Y) 帮助(Z) Language 搜索串口(X) Port:COM8 bps:115200
www.mcuisp.com 免费软件 脱机/高线编程器(W)
手持万用编程器 STMISP 免费STMIAP LPC2xxxISP
读器件信息(R) 清除芯片(Z) 读FLASH

联机下载时的程序文件：

E:\工作文件\电机驱动模块\STM32_line\Obj\OpenArmSTM32.hex

□编程前重装文件

开始编程(P)

☑校验
□编程后执行
□使用RamIsp
□连续烧录模式

选项字节区：

□编程到FLASH时写选项字节

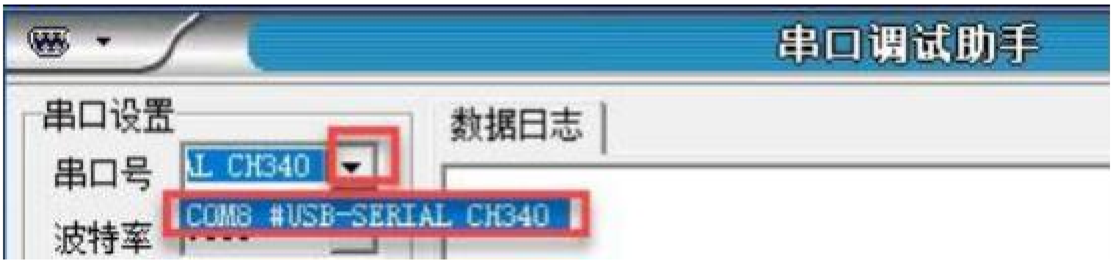
读保护：A5 硬件选项：FF 设定选项字节等

写保护：FF FF FF FF 用户数据0-1 FF FF

DTR的高电平复位,RTS高电平进BootLoader

```
DTR电平置高(+3-+12V),复位
RTS置高(+3-+12V),选择进入BootLoader
...延时100毫秒
DTR电平变低(-3--12V)释放复位
RTS维持高
开始连接...2, 接收到:79
在串口COM8连接成功@115200bps, 耗时297
毫秒
芯片内BootLoader版本号: 2.2
芯片PID: 00000410
芯片FLASH容量为128KB
芯片SRAM容量为65535KB(此信息仅供参考,
新版本芯片已不包含此信息)
96位的芯片唯一序列号:
53FF6D067580485536190367
读出的选项字节:
A55AFFFFFFFFFFFFFFFFFFFFFFFFFFFF
全片擦除成功
第375毫秒, 已准备好
共写入14KB,进度100%,耗时6500毫秒
www.mcuisp.com向您报告, 命令执行完毕,
一切正常
```

6) After the burning is finished, double click the executable file UartAssist.exe in "UartAssist.exe" folder. Open the serial debug assistant tool (Select the serial number "CH340" and baud rate "9600".) You can refer to the below figures for parameter adjustment.



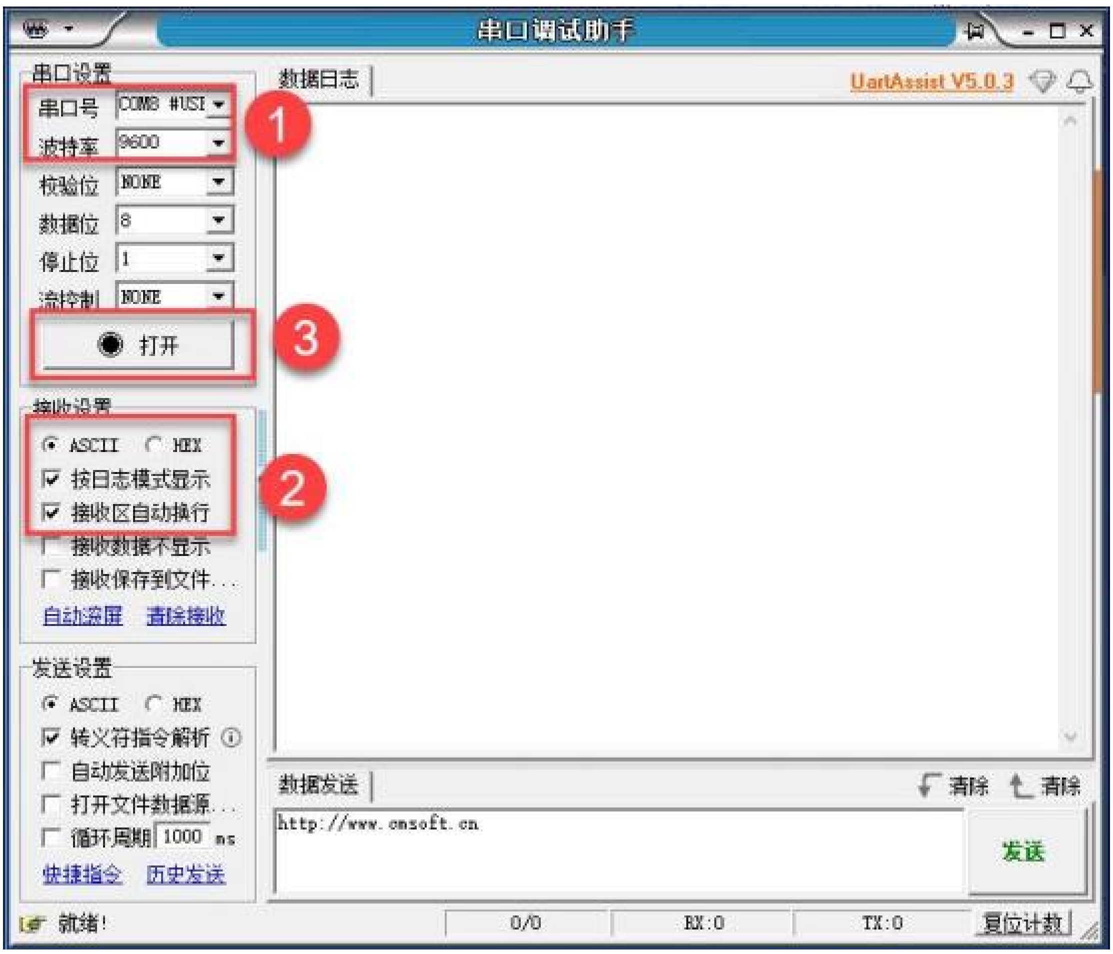
串口调试助手

串口设置

串口号 L CH340

COM8 #USB-SERIAL CH340

波特率



串口调试助手

串口设置

串口号 COM8 #USI

波特率 9600

校验位 NONE

数据位 8

停止位 1

流控制 NONE

● 打开

接收设置

● ASCII ○ HEX

☑ 按日志模式显示

☑ 接收区自动换行

□ 接收数据不显示

□ 接收保存到文件...

发送设置

● ASCII ○ HEX

☑ 转义符指令解析

□ 自动发送附加位

□ 打开文件数据源...

□ 循环周期 1000 ms

数据发送

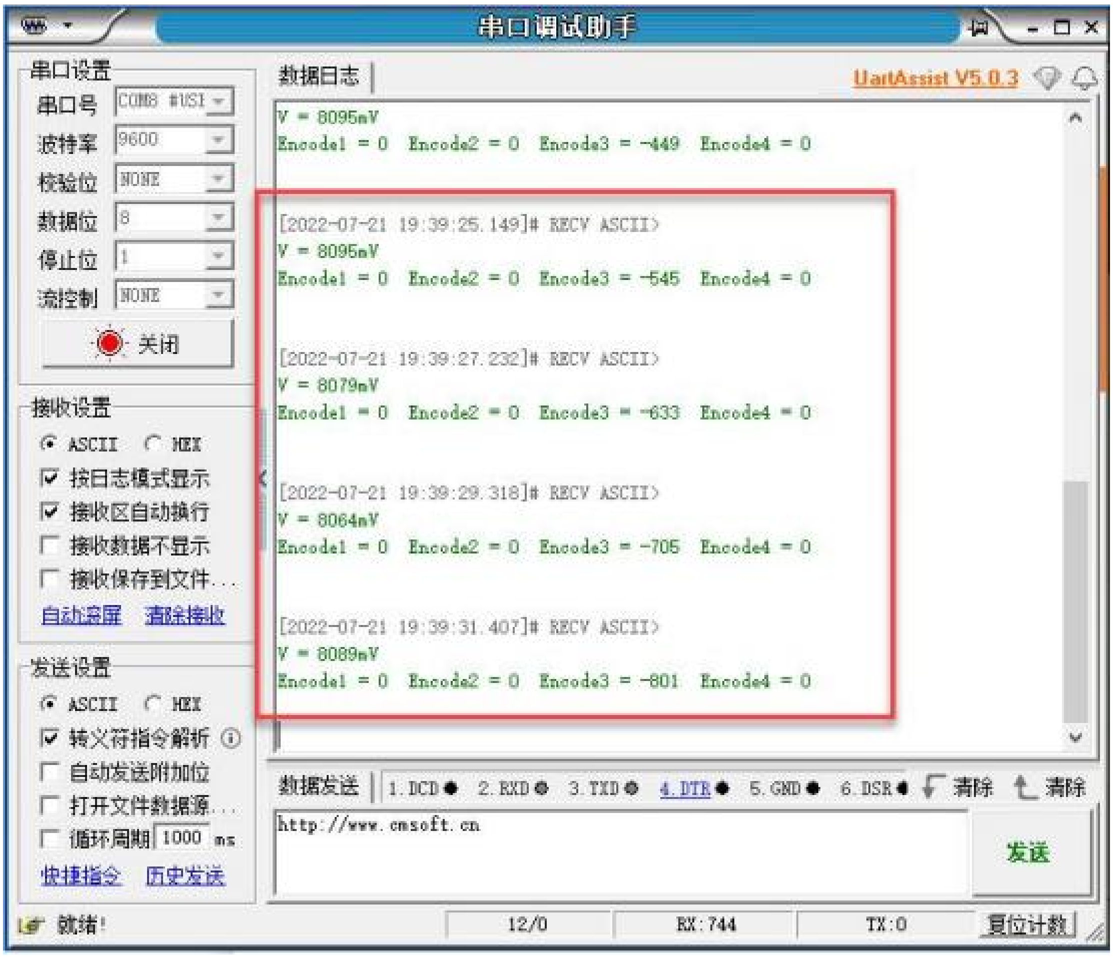
http://www.cmsoft.cn

就绪! 0/0 RX:0 TX:0

7) Insert the jumper cap, then push the RST button to execute the program.

2.2.3 Project Outcome

The motor will rotate forwards first, and then reverse in a continuous loop. Also, the serial debug tool will print the voltage value and Endoce3 information (the relative position of M3 motor).



串口号 COM8 #USI
波特率 9600
校验位 NONE
数据位 8
停止位 1
流控制 NONE

```
V = 8095mV
Encode1 = 0  Encode2 = 0  Encode3 = -449  Encode4 = 0

[2022-07-21 19:39:25.149]# RECV ASCII>
V = 8095mV
Encode1 = 0  Encode2 = 0  Encode3 = -545  Encode4 = 0

[2022-07-21 19:39:27.232]# RECV ASCII>
V = 8079mV
Encode1 = 0  Encode2 = 0  Encode3 = -633  Encode4 = 0

[2022-07-21 19:39:29.318]# RECV ASCII>
V = 8064mV
Encode1 = 0  Encode2 = 0  Encode3 = -705  Encode4 = 0

[2022-07-21 19:39:31.407]# RECV ASCII>
V = 8089mV
Encode1 = 0  Encode2 = 0  Encode3 = -801  Encode4 = 0
```

http://www.cmsoft.cn

就绪! 12/0 RX:744 TX:0

2.2.4 Sample Code

Please refer to "**4.Appendix/4.2 STM32 Sample Code**."

3. Register

The relevant register instruction refers to the following table:

Register Name	**Address**	**Function**
ADC_ BAR _ADDR	0x00	ADC battery sampling
MOTOR_ TYPE_ ADDR	0x20	Encoder motor type settings
MOTOR_ ENCODER_ POLARITY_ ADDR	0x21	Encoder direction polarity settings
MOTOR_FIXED _PWM_ADDR	0x31	Fixed PWM control. It is open-loop control and its range is -100 to 100
MOTOR_FIXED_SPEED_ADDR	0x51	Fixed speed control. It is closed-loop control
MOTOR_ENCODER _TOTAT_ADDR	0x60	The total pulse values for the four encoded motors.

4. Source Code

4.1 Arudino Sample Code

```
#include <Wire.h>

#define I2C_ADDR            0x34//I2C address

#define ADC_BAT_ADDR                         0 // voltage address

#define MOTOR_TYPE_ADDR                      20 // set the motor type

#define MOTOR_ENCODER_POLARITY_ADDR     21 //Set the encoder direction polarity

//If you find that the motor speed is out of control, you can modify the address value.

//0 or 1,  0 by default



#define MOTOR_FIXED_PWM_ADDR          31 //Fixed PWM control, it is open-loop
                                      control
//#define SERVOS_ADDR_CMD 40

#define MOTOR_FIXED_SPEED_ADDR       51 //Fixed speed control, it is closed-loop
                                     control

#define MOTOR_ENCODER_TOTAL_ADDR   60 // the total pulse value of 4 encoder
motors
```

```
//If the number of pulses per revolution of the motor (U) and the diameter of the wheel
(D) are known, then the distance traveled by each wheel can be determined through
pulse counting.

//For example, if you read the total number of pulses (P) from motor 1, then the distance
traveled can be calculated using the formula: (P/U) * (3.14159 * D)

//For different motors, you can test the number of pulses per revolution (U) by manually
rotating them for 10 revolutions and reading the pulse count. Then, take the average
value to determine it.


//The specific value of the motor type

#define MOTOR_TYPE_WITHOUT_ENCODER           0 //the magnetic ring generates
44 pulses per revolution, combined with a gear reduction ratio of 90  Default

#define MOTOR_TYPE_TT                        1   //TT  encoder motor

#define MOTOR_TYPE_N20                       2 //N20  encoder motor

#define MOTOR_TYPE_JGB37_520_ 12V_ 110RPM    3 //the magnetic ring generates 44
pulses per revolution, combined with a gear reduction ratio of 90  Default



u8 data[20]; //temperately store the data read through I2C

bool WireWriteByte(uint8_t val)

{

    Wire.beginTransmission(I2C_ADDR);

    Wire.write(val);

    if( Wire.endTransmission() != 0 ) {

        return false;

    }
    return true;

}

//Write the data into address (reg: address val: data content len:data length)
bool WireWriteDataArray(   uint8_t reg,uint8_t *val,unsigned int len)

{

    unsigned int i;



    Wire.beginTransmission(I2C_ADDR);

    Wire.write(reg);

    for(i = 0; i < len; i++) {

        Wire.write(val[i]);

    }

    if( Wire.endTransmission() != 0 ) {

        return false;

    }



    return true;

}

//Read the data from an address (reg:address  val: data content)

bool WireReadDataByte(uint8_t reg, uint8_t &val)

{

    if (!WireWriteByte(reg)) {
        return false;
    }

    Wire.requestFrom(I2C_ADDR, 1);
    while (Wire.available()) {
        val = Wire.read();
    }

    return true;
}
//Read data of a specified length from an address (reg: address val: data content  len: data length)
int WireReadDataArray(   uint8_t reg, uint8_t *val, unsigned int len)
{
    unsigned char i = 0;

    /* Indicate which register we want to read from */
    if (!WireWriteByte(reg)) {
        return -1;
    }

    /* Read block data */
    Wire.requestFrom(I2C_ADDR, len);
```

```
        while (Wire.available()) {
                if (i >= len) {
                        return - 1;
                }
                val[i] = Wire.read();
                i++;
        }

        return i;
}
int serial_putc( char c, struct __file * )
{
    Serial.write( c );
    return c;
}
void printf_begin(void)
{
    fdevopen( &serial_putc, 0 );
}

uint8_t MotorType = MOTOR_TYPE_TT; //Set the motor type

uint8_t MotorEncoderPolarity = 1;
```

```
void setup()

{

Wire.begin(); //Take Arduino UNO as example. I2C port is A4 (SCL), A5 (CLK) Serial.

Begin (9600); // Initialize the serial and set the baud rate as 9600



   printf_begin(); //
                      printf  output initialization
   delay(200);

   WireWriteDataArray(MOTOR_TYPE_ADDR,&MotorType, 1);// Write the type number into
the motor type address

   delay(5);

   WireWriteDataArray(MOTOR_ENCODER_POLARITY_ADDR,&MotorEncoderPolarity, 1);



}

/*Use an array to pass motor speed, positive values indicate forward speed, and
negative values indicate reverse speed."
Let's illustrate the example for p1 and p2: p1: Four motors move forward at a speed of
50. p2: Four motors move backward at a speed of 20.

int8_t p1[4]={50,50,50,50};

int8_t p2[4]={-50,-50,-50,-50};

int8_t s1[4]={2,2,2,2};

int8_t s2[4]={-2,-2,-2,-2};

//int8_t s1[4]={2,20, 10,30};

//int8_t s2[4]={-20,-20,-30,-9};

int32_t EncodeTotal[4];//This is used to temporarily store the accumulated rotation value
of the motor. The value increases when rotating in the forward direction and decreases
when rotating in the reverse direction.
```

```
void loop()
{
   u16 v; //This is used to temporarily store the voltage value.
   WireReadDataArray(ADC_BAT_ADDR,data,2);//Read the stored voltage
      stored in voltage address
   v = data[0]+ (data[ 1]<<8); //Cconvert the voltage into mV
   Serial.print("V = ");Serial.print(v);Serial.println("mV          "); //Print the voltage value


   WireReadDataArray(MOTOR_ENCODER_TOTAL_ADDR,(uint8_t*)EncodeTotal, 16);// Read
the accumulated rotation value of the motor
    /*Print the accumulated rotation value of the four motors*/
   printf("Encode1  =  %ld     Encode2  =  %ld     Encode3  =  %ld     Encode4  =  %ld     \r\n",
EncodeTotal[0], EncodeTotal[1], EncodeTotal[2], EncodeTotal[3]);
/*Write the motor's rotation direction and speed into the motor speed address:
WireWriteDataArray (speed control address, motor speed array, number of motors） */
   WireWriteDataArray(MOTOR_FIXED_SPEED_ADDR,p1,4);
//    WireWriteDataArray(MOTOR_FIXED_PWM_ADDR,p1,4);
   WireWriteDataArray(MOTOR_FIXED_SPEED_ADDR,s1,4);
   delay(700);


//    WireWriteDataArray(MOTOR_FIXED_PWM_ADDR,p2,4);
   WireWriteDataArray(MOTOR_FIXED_SPEED_ADDR,s2,4);
   delay(700);
}
```

4.2 STM32 Sample Code

```
/*****************************************************

//     Company:ShenZhen Hiwonder Technology Company                               //

*****************************************************/

#include "include.h"

#define CAM_DEFAULT_I2C_ADDRESS          (0x34)        //I2C  address
#define MOTOR_TYPE_ADDR                   20           // Motor type and set the
                                                       register address
#define MOTOR_FIXED_SPEED_ADDR             51          //Speed register address. It
Closed-loop control                                    is closed-loop control
#define MOTOR_ENCODER_POLARITY_ADDR        21          // Motor encoder direction
                                                       polarity address. It is open-
                                                       closed control
#define MOTOR_FIXED_PWM_ADDR               31          //Fixed PWM control
                                                       address. It is open-loop
                                                       control
#define MOTOR_ENCODER_TOTAL_ADDR                60     //The individual pulse value
                                                       of 4 encode motors
#define ADC_BAT_ADDR                     0
                                                       //Voltage address
```

```
uint8_t data[3];                                                                        //
This is used to temporarily store the ADC data.

//Loop through and send the data in an array. ( addr: address  buf: data content  leng：
data length)
Send a da

uint8_t I2C_Write_Len(uint8_t Reg,uint8_t *Buf,uint8_t Len)//The data of I2C

{

    uint8_t i;

    IIC_start();

                        //

After the start signal, a 7-bit slave address followed by a 1-bit direction bit must be sent,
with '0' indicating the master sends data and '1' indicating the master receives data.
    IIC_send_byte((CAM_DEFAULT_I2C_ADDRESS << 1) | 0);    //Sent the device

address and write into command.

    if(IIC_wait_ack() == 1)


    //Await respond. If it is failure, send a stop signal and return 1
    {

        IIC_stop();
        return 1;

    }

    IIC_send_byte(Reg);

            //Send the register address

    if(IIC_wait_ack() == 1)

    //Await respond. If it is failure, send a stop signal and return 1


    {

        IIC_stop();

        return 1;

    }

    for(i =0;i<Len;i++)
          //Write data for len iterations.
    {

        IIC_send_byte(Buf[i]);

          //Send the 8-bit data of the i-th position

        if(IIC_wait_ack() == 1)

    //Await respond. If it is failure, send a stop signal and return 1
        {

            IIC_stop();

            return 1;
        }

    }

    IIC_stop();

                    //Send a stop signal
    return 0;

                              //Returning 0 confirm a successful sending
}


//read 1 byte of data and return the read data (Reg: address)

                              uint8_t I2C_Read(uint8_t Reg)

{

    uint8_t Dat;

    IIC_start();

              //After the start signal, a 7-bit slave address followed by a 1-bit direction bit
must be sent, with '0' indicating the master sends data and '1' indicating the master
receives data

    IIC_send_byte((CAM_DEFAULT_I2C_ADDRESS << 1) | 0);//Send device address +
    write command.

    IIC_wait_ack();

              //Await respond. If it is failure, send a stop signal and return 1

    IIC_send_byte(Reg);

          //Send the register address

    IIC_wait_ack();

              //Await respond. If it is failure, send a stop signal and return 1

    IIC_start();

              //the starting signal

    IIC_send_byte((CAM_DEFAULT_I2C_ADDRESS << 1) | 1);//Send device address +
    write command.


    IIC_wait_ack();

              //Await respond. If it is failure, send a stop signal and return 1

    Dat = IIC_read_byte(0);                                                    //
```

```
Temporarily store the read data in Dat.

    IIC_stop();
                    //Send a stop signal to stop reading.

    return Dat;

                    //Read successfully and return data
}


//
  Read the data of multiple bytes (Reg: address But: data content Len: data length)


int readFromAddr(uint 16_t addr,   uint8_t *buf, uint 16_t leng) {

  int len = 0,i;



  for (i = 0; i < leng; i++) {

    buf[i] = I2C_Read(addr++); //Read a byte

      ++len;

    }


  return len;

}

//The specific address of the motor type

#define MOTOR_TYPE_WITHOUT_ENCODER    0          //The motor with encoder. The
                                      magnetic ring generates 44 pulses per
                                      revolution, combined with a gear
#define MOTOR_TYPE_TT                 reduction ratio of 90  Default
```

```
#define MOTOR_TYPE_N20
                                              1          //TT  encoder motor

                                              2          //N20 encoder motor
```

```
#define MOTOR_TYPE_JGB37_520_ 12V_ 110RPM      3         //The magnetic ring
generates 44 pulses per revolution, combined with a gear reduction ratio of 90  Default



/*Use an array to pass motor speed, positive values indicate forward speed, and
negative values indicate reverse speed

Let's illustrate the example for p1 and p2: p1:          // motor speed
Four motors move forward at a speed of 50. p2:
Four motors move backward at a speed of 20.

uint8_t p1[4]={10, 10, 10, 10};                       //Motor speed settings


uint8_t p2[4]={- 10,- 10,- 10,- 10};

uint8_t MotorEncoderPolarity = 1;
                                                      //Motor polarity control variable
long int EncodeTotal[4];
                                                      // This is used to temporarily store the
                                                      accumulated rotation value of the
                                                      motor. The value increases when
                                                      rotating in the forward direction and
                                                      decreases when rotating in the
                                                      reverse direction.



uint8_t MotorType = MOTOR_TYPE_TT;                    //set the motor type


int main(void)

{
    int v;
//this is used to peremptorily store the voltage value


    SystemInit();

                //The system clock is initialized to 72MHz        SYSCLK_FREQ_72MHz


NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);              //Set NVIC interrupt grouping
as 2:2, which means 2 bits for preemption priority and 2 bits for subpriority.

    InitDelay(72);

                      //Initialize the latency function
```

```
InitTimer2();

                    //Timer 2 Initialization

IIC_init();

                          //IIC  Initialization

Usart 1_Init();

             //Serial initialization

InitLED();


             //Initialize LED indicator



DelayMs(200);

I2C_Write_Len(MOTOR_TYPE_TT,&MotorType, 1);          // Write the motor type
                                                     number into the motor
                                                     type address

DelayMs(5);

I2C_Write_Len(MOTOR_ENCODER_POLARITY_ADDR,&MotorEncoderPolarity, 1);

// Set the motor polarity settings
DelayMs(5);

while( 1)


{

      readFromAddr(ADC_BAT_ADDR,data,2);                  //Read the voltage of
                                                          the motor

      v = data[0] + (data[ 1]<<8);                        // Convert voltage
```

```
printf("V = ");        printf("%d",v);        printf("mV\n"); //Print the voltage

readFromAddr(MOTOR_ENCODER_TOTAL_ADDR,(uint8_t*)EncodeTotal, 16);


//Read the absolute position of the motor
```

```
        /*Print the absolute position of four motors*/

        printf("Encode1 = %ld    Encode2 = %ld    Encode3 = %ld    Encode4 = %ld    \r\n",
EncodeTotal[0], EncodeTotal[1], EncodeTotal[2], EncodeTotal[3]);

        /*Write the motor rotation direction and speed into the motor speed
address: WireWriteDataArray (speed control address, motor speed array, number of
motors)*/

        I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p1,4);            //Control the
                                                               motor speed
        DelayMs( 1000);

        I2C_Write_Len(MOTOR_FIXED_SPEED_ADDR,p2,4);
                                                               //Control the
        DelayMs( 1000);
                                                               motor speed
    }

}
```