STM32 的硬件 IIC 很好用

——cuyebiren

——意法半导体 STM32/STM8 技术社区

都说 STM32 的 IIC 有 BUG，不好用，都用 IO 模拟，但我想说 STM32 的 硬件 IIC 很好用！用 IO 模拟的话，Keil 的优化等级要设为 Level 0，这样的话代码量势必要变大，而且也不能用中断、DMA 等方式，操作方式单一。

在此，本人基于 STM32CubeMx 生成初始化代码工程，参考 ST 官方列程和 正点原子的列程 ，以及 AT24C02 的 Datasheet ，编写本 IIC_AT24CXX 列程。

首先，说一下，HAL 库的外设驱动是比较完整和封装比较彻底的，使用它，我们不用再写一些如 IIC 读写过程等过程操作函数，直接调用 HAL 库函数即可。

CubeMx 的详细配置过程请参考：（神器）STM32CubeMx 使用详解

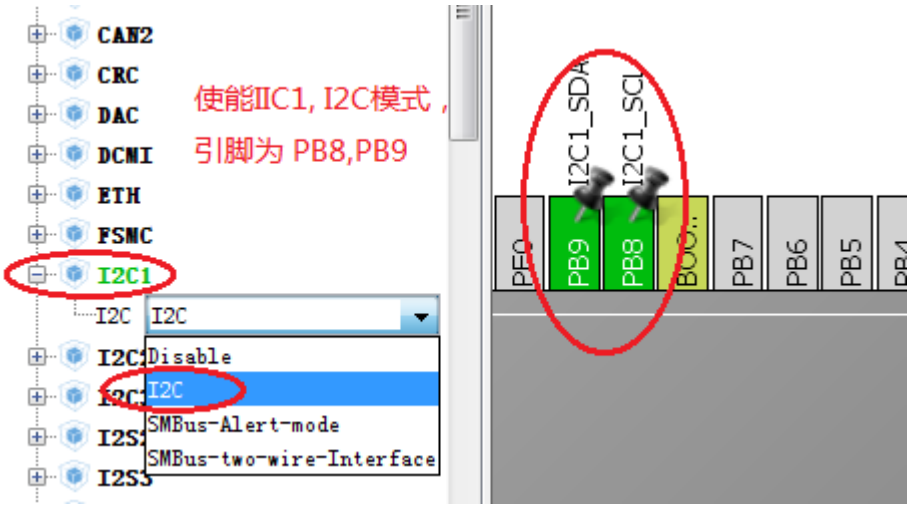
http://www.stmcu.org/module/forum/forum.php?mod=viewthread&tid=608654&fromuid=3135760

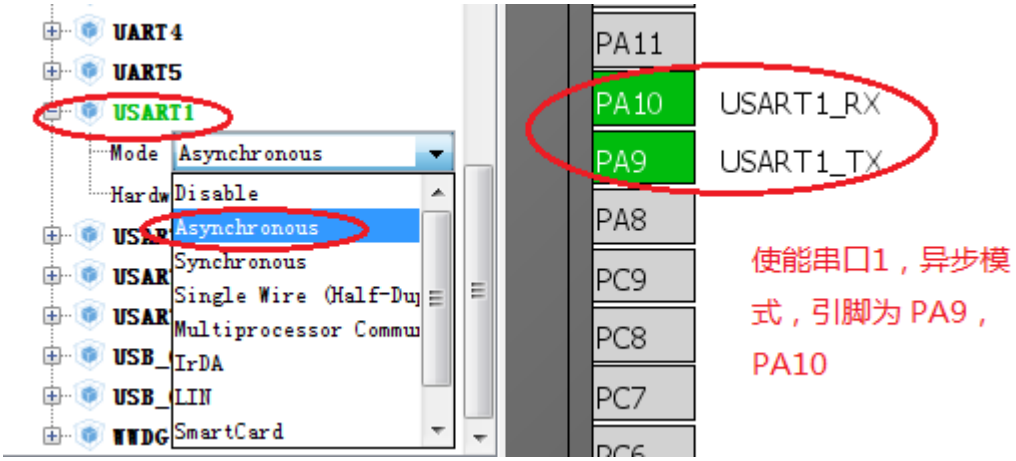
(出处: 意法半导体 STM32/STM8 技术社区)

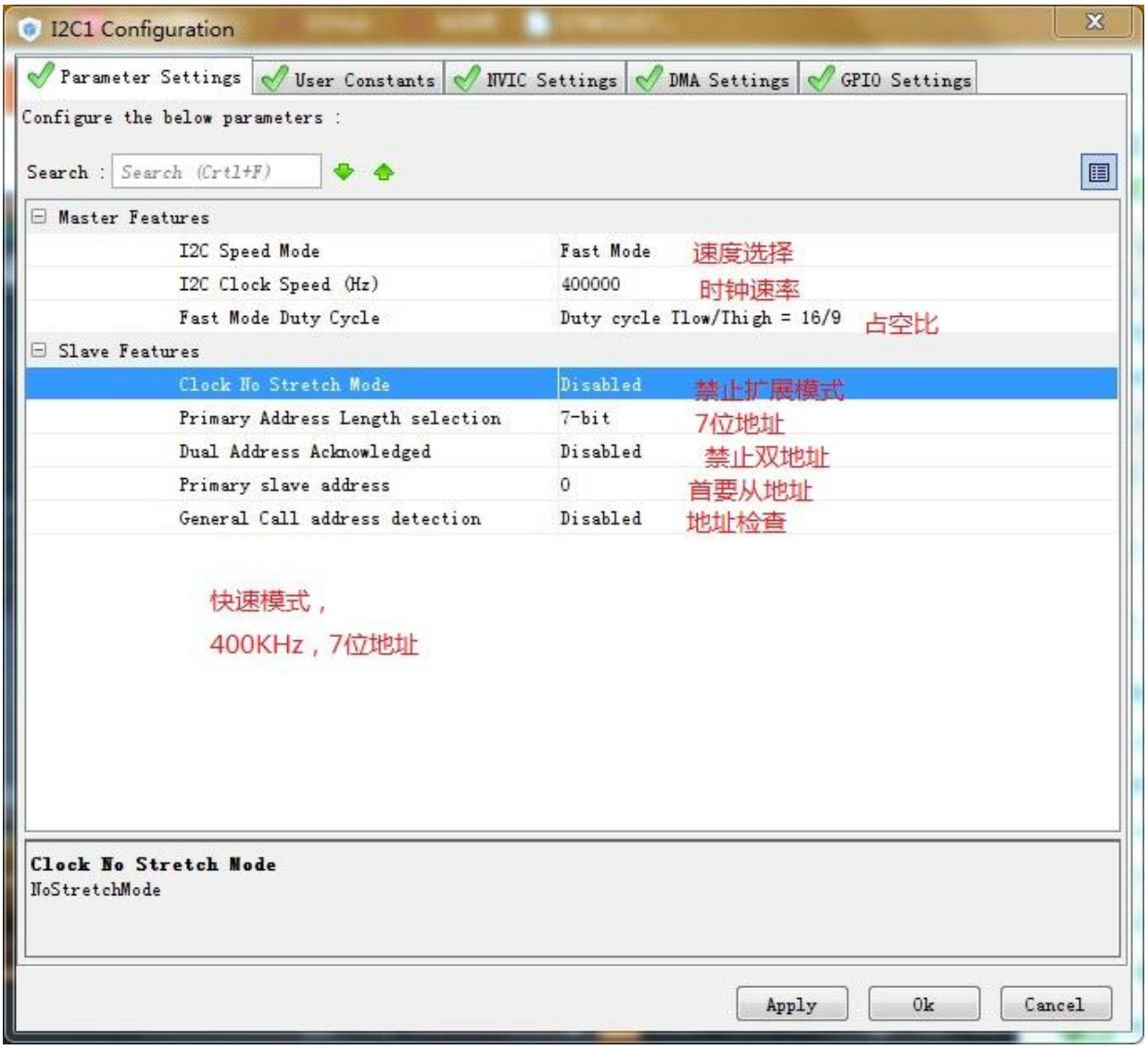
本人开发板的 IIC 引脚兼容正点原子的开发板。

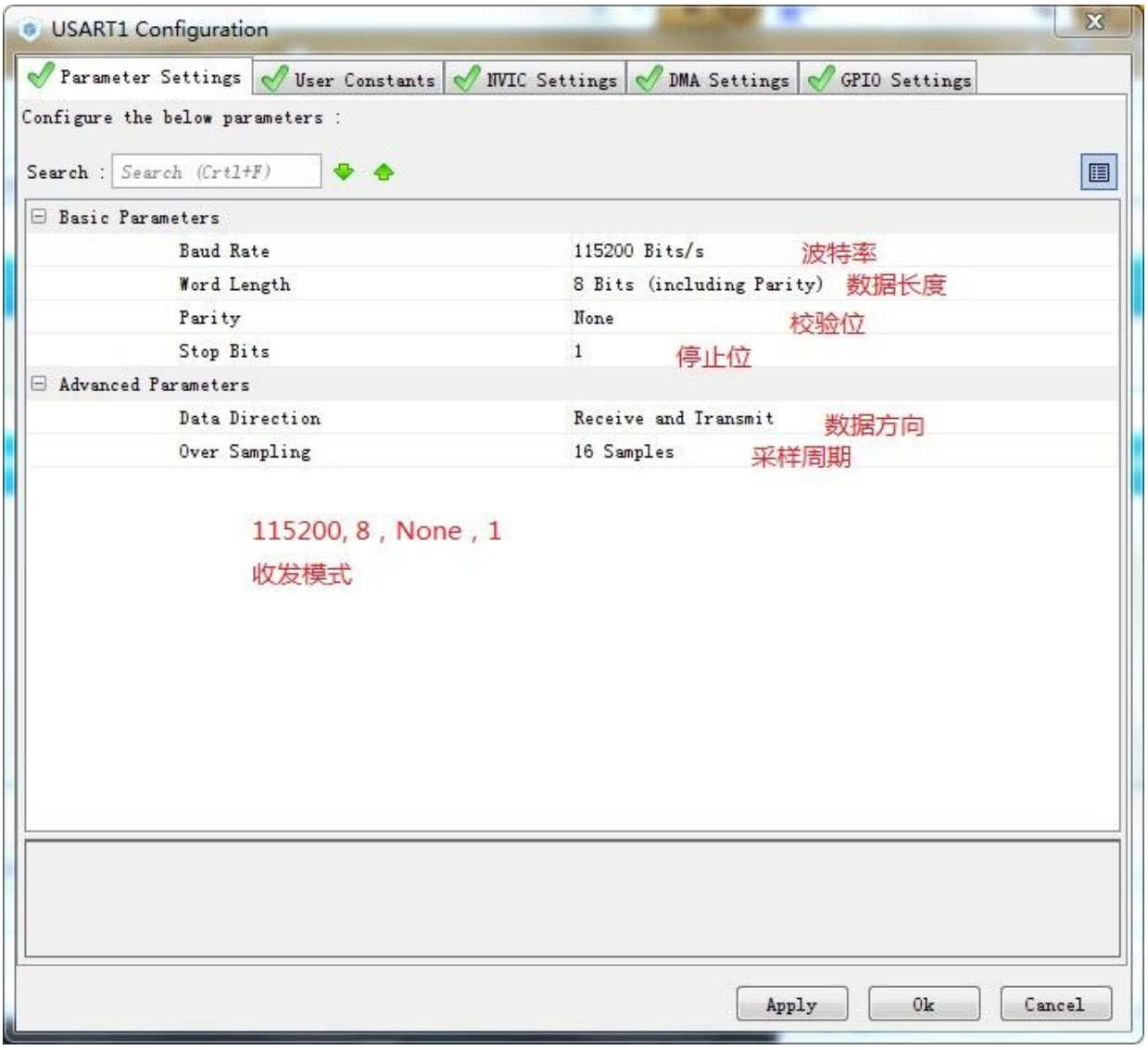
下面，开始介绍用 IIC 读写 AT24C02 的教程。

一、CubeMx 工程配置。

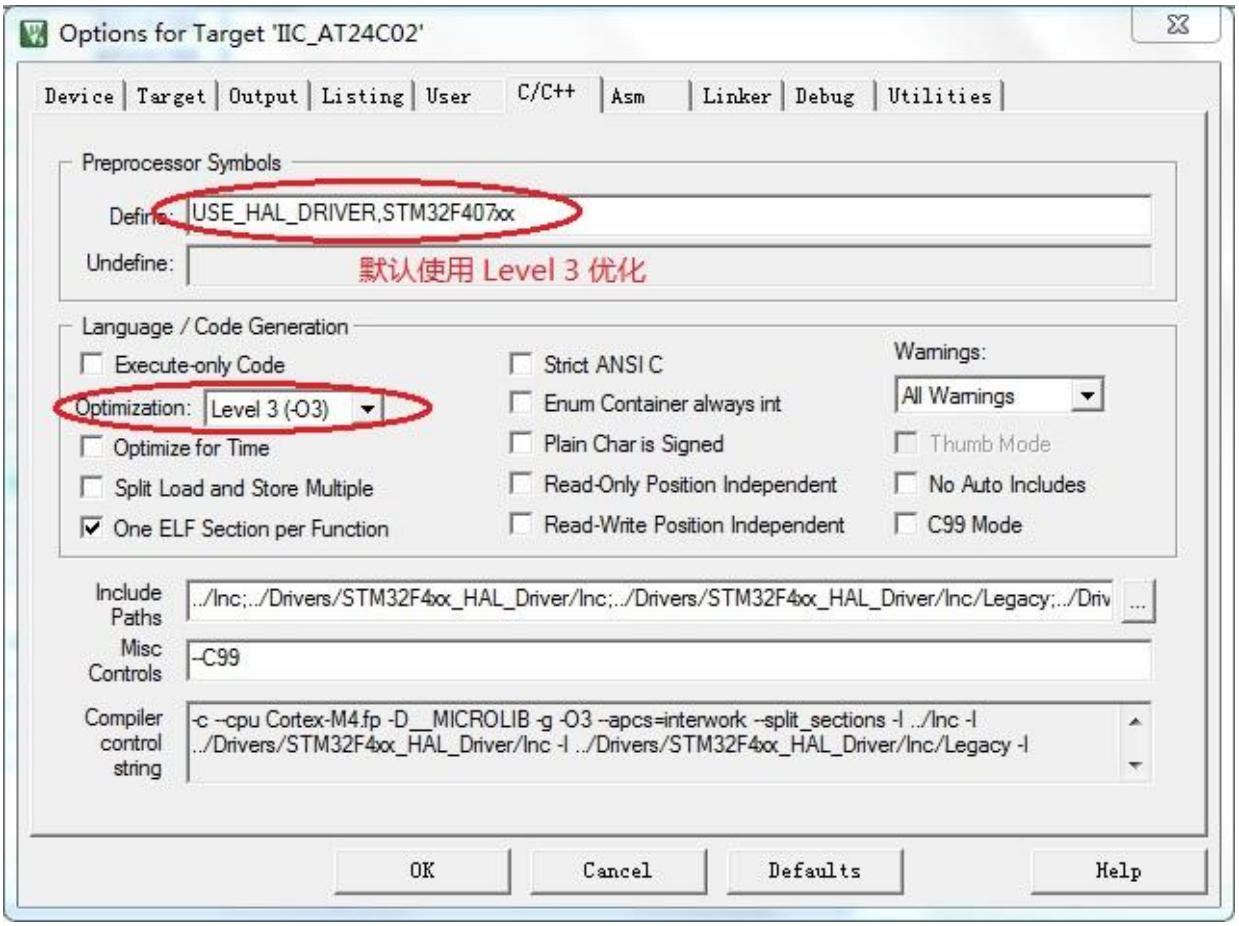


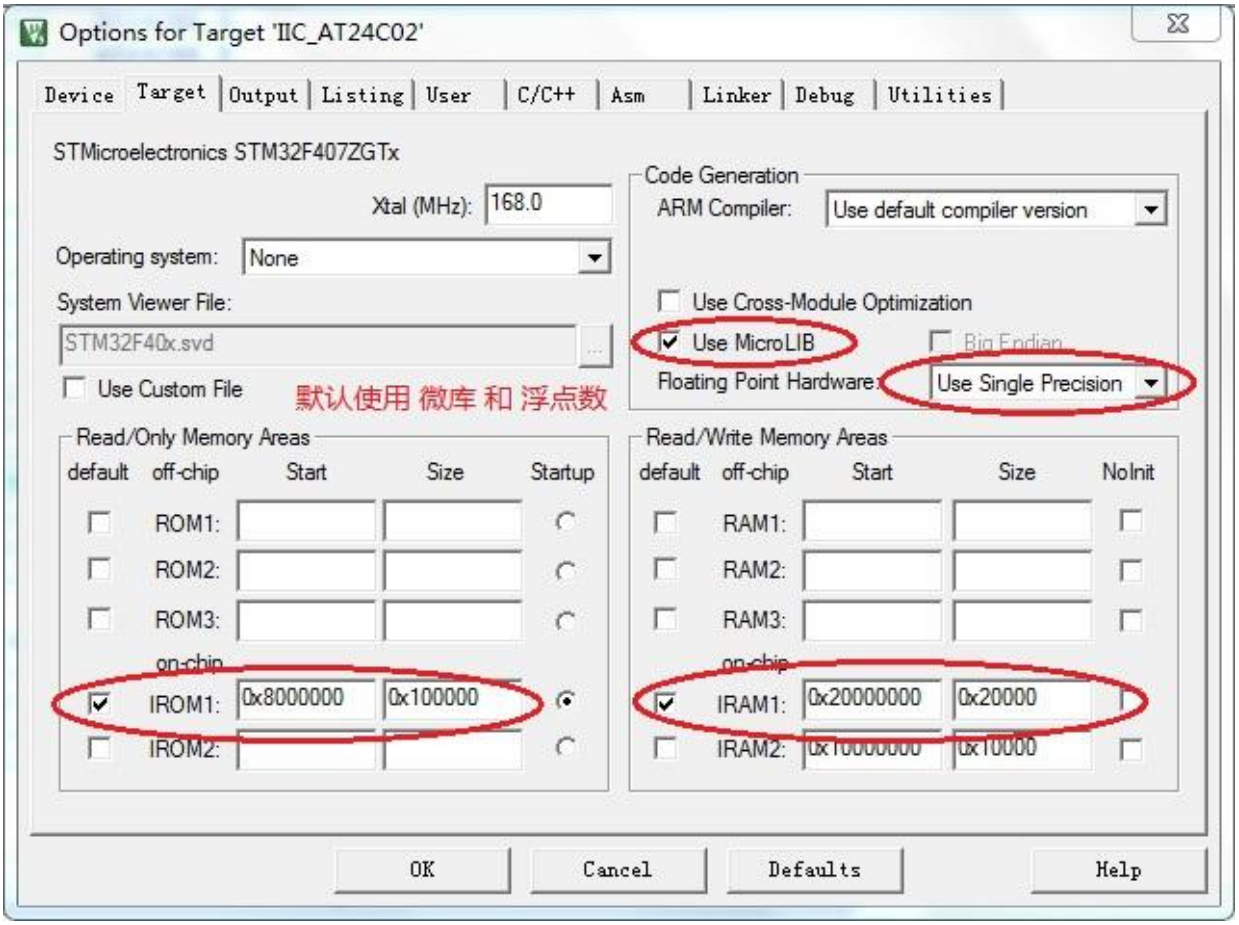


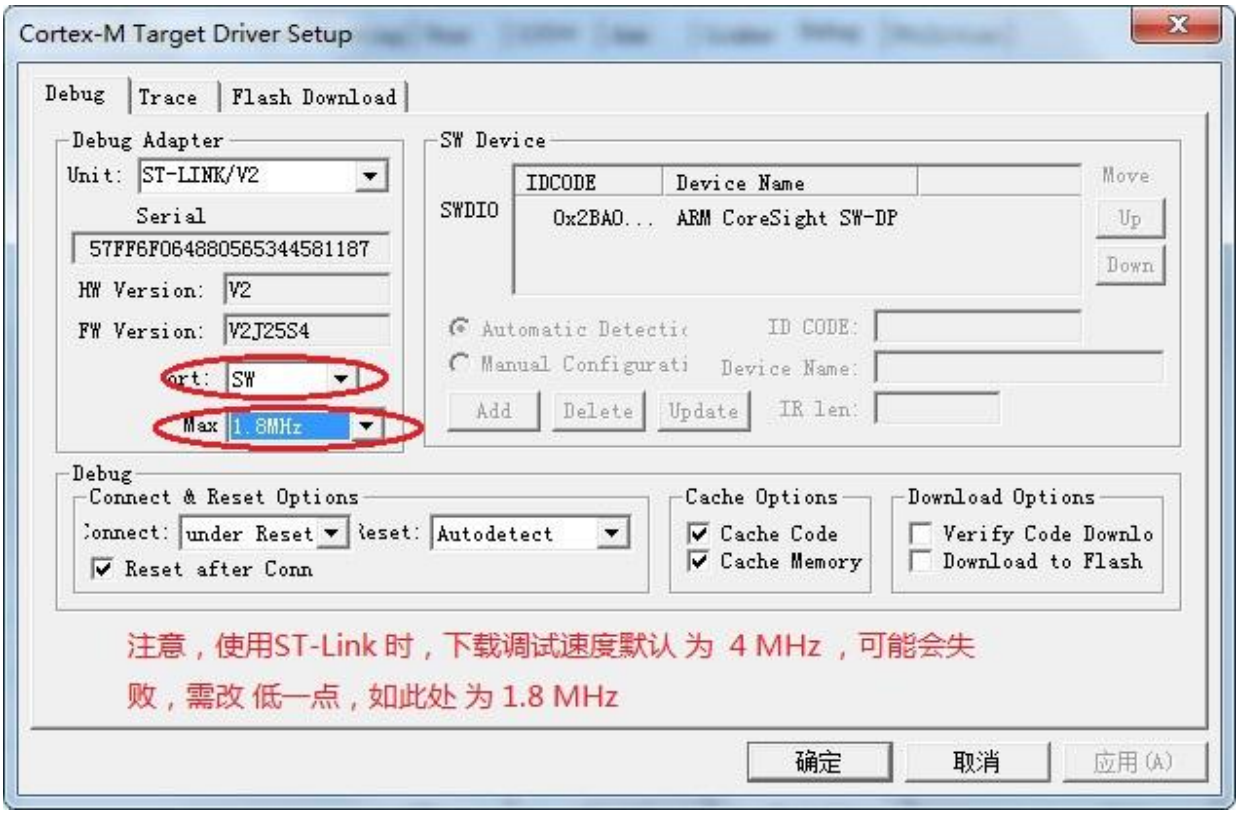

I2C1 Configuration
Parameter Settings
User Constants
NVIC Settings
DMA Settings
GPIO Settings
Configure the below parameters :
Search :
Search (Crtl+F)
Master Features
I2C Speed Mode
Fast Mode
速度选择
I2C Clock Speed (Hz)
400000
时钟速率
Fast Mode Duty Cycle
Duty cycle Tlow/Thigh = 16/9
占空比
Slave Features
Clock No Stretch Mode
Disabled
禁止扩展模式
Primary Address Length selection
7-bit
7位地址
Dual Address Acknowledged
Disabled
禁止双地址
Primary slave address
0
首要从地址
General Call address detection
Disabled
地址检查
快速模式，
400KHz，7位地址
Clock No Stretch Mode
NoStretchMode
Apply
Ok
Cancel


USART1 Configuration
Parameter Settings
User Constants
NVIC Settings
DMA Settings
GPIO Settings
Configure the below parameters :
Search :
Search (Crtl+F)
Basic Parameters
Baud Rate
115200 Bits/s
波特率
Word Length
8 Bits (including Parity)
数据长度
Parity
None
校验位
Stop Bits
1
停止位
Advanced Parameters
Data Direction
Receive and Transmit
数据方向
Over Sampling
16 Samples
采样周期
115200, 8 , None , 1
收发模式
Apply
Ok
Cancel

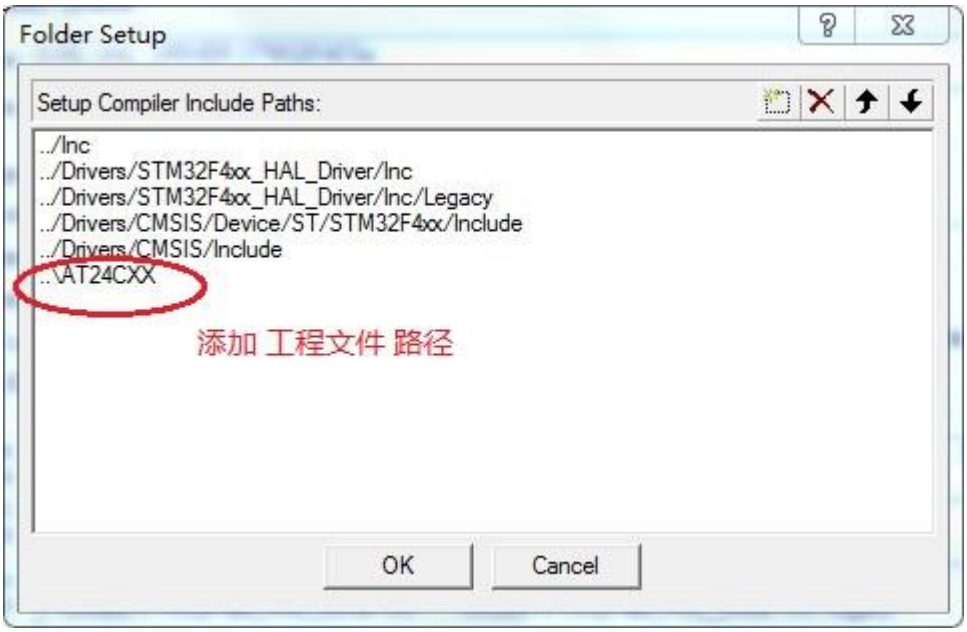
二、Cube 生成的 Keil 工程的配置选项详解。

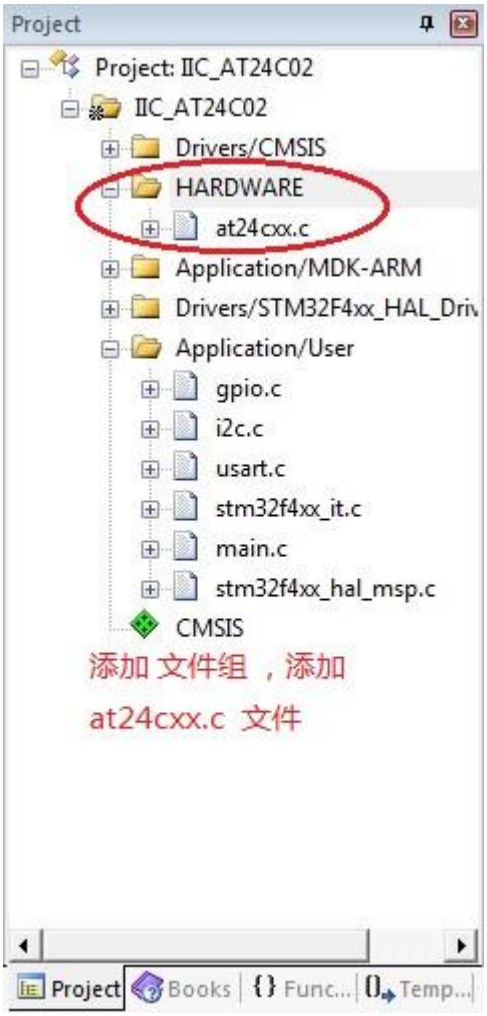



Options for Target 'IIC_AT24C02'
Device
Target
Output
Listing
User
C/C++
Asm
Linker
Debug
Utilities
STMicroelectronics STM32F407ZGTx
Xtal (MHz): 168.0
Operating system: None
System Viewer File:
STM32F40x.svd
Use Custom File
默认使用 微库 和 浮点数
Code Generation
ARM Compiler: Use default compiler version
Use Cross-Module Optimization
Use MicroLIB
Big Endian
Floating Point Hardware: Use Single Precision
Read/Only Memory Areas
default off-chip Start Size Startup
ROM1:
ROM2:
ROM3:
on-chip
IROM1: 0x8000000 0x100000
IROM2:
Read/Write Memory Areas
default off-chip Start Size NoInit
RAM1:
RAM2:
RAM3:
on-chip
IRAM1: 0x20000000 0x20000
IRAM2: 0x10000000 0x10000
OK
Cancel
Defaults
Help


Cortex-M Target Driver Setup
Debug
Trace
Flash Download
Debug Adapter
Unit: ST-LINK/V2
Serial
57FF6F064880565344581187
HW Version: V2
FW Version: V2J25S4
ort: SW
Max 1.8MHz
SW Device
IDCODE Device Name
SWDIO 0x2BA0... ARM CoreSight SW-DP
Move
Up
Down
Automatic Detecti
Manual Configurati
ID CODE:
Device Name:
Add
Delete
Update
IR len:
Debug
Connect & Reset Options
Connect: under Reset
Reset: Autodetect
Reset after Conn
Cache Options
Cache Code
Cache Memory
Download Options
Verify Code Downlo
Download to Flash
注意，使用ST-Link 时，下载调试速度默认 为 4 MHz ，可能会失
败，需改 低一点，如此处 为 1.8 MHz
确定
取消
应用 (A)

三、添加文件。


Folder Setup
Setup Compiler Include Paths:
../Inc
../Drivers/STM32F4xx_HAL_Driver/Inc
../Drivers/STM32F4xx_HAL_Driver/Inc/Legacy
../Drivers/CMSIS/Device/ST/STM32F4xx/Include
../Drivers/CMSIS/Include
..\AT24CXX
添加 工程文件 路径
OK
Cancel


Project
Project: IIC_AT24C02
IIC_AT24C02
Drivers/CMSIS
HARDWARE
at24cxx.c
Application/MDK-ARM
Drivers/STM32F4xx_HAL_Driv
Application/User
gpio.c
i2c.c
usart.c
stm32f4xx_it.c
main.c
stm32f4xx_hal_msp.c
CMSIS
添加 文件组 ，添加
at24cxx.c 文件
Project
Books
Func...
Temp...

四、文件分析。

at24cxx.h

```
#include "i2c.h"                    头文件

#define AT24C01   127   //PAGE_SIZE 8 byte
#define AT24C02   255   //PAGE_SIZE 8 byte
#define AT24C04   511   //PAGE_SIZE 16  byte
#define AT24C08   1023  //PAGE_SIZE 16  byte
#define AT24C16   2047  //PAGE_SIZE 16  byte
#define AT24C32   4095
#define AT24C64   8191
#define AT24C128  16383
#define AT24C256  32767
//开发板使用的是24c02，所以定义EE_TYPE为AT24C02
#define AT24CXX_TYPE                    AT24C02
#define AT24CXX_ADDRESS                 0XA0 器件地址
#define AT24CXX_PAGE_SIZE               8    页大小

#if (AT24CXX_TYPE < AT24C04)
#define AT24CXX_MEMADD_SIZE             I2C_MEMADD_SIZE_8BIT
#else
#define AT24CXX_MEMADD_SIZE             I2C_MEMADD_SIZE_16BIT
#endif
        器件 内部存储器 大小

void AT24CXX_Init(void);
void AT24CXX_WriteByte(uint8_t Reg, uint8_t Value);
uint8_t AT24CXX_ReadByte(uint8_t Reg);
HAL_StatusTypeDef AT24CXX_PageWrite(uint16_t MemAddress, uint8_t* pBuffer, uint32_t BufferSize);
HAL_StatusTypeDef AT24CXX_WriteData(uint16_t MemAddress, uint8_t* pBuffer, uint32_t BufferSize);
HAL_StatusTypeDef AT24CXX_ReadData(uint16_t MemAddress, uint8_t* pBuffer, uint32_t BufferSize);
HAL_StatusTypeDef AT24CXX_IsDeviceReady(uint32_t Trials);

uint8_t AT24CXX_Check(void);
```

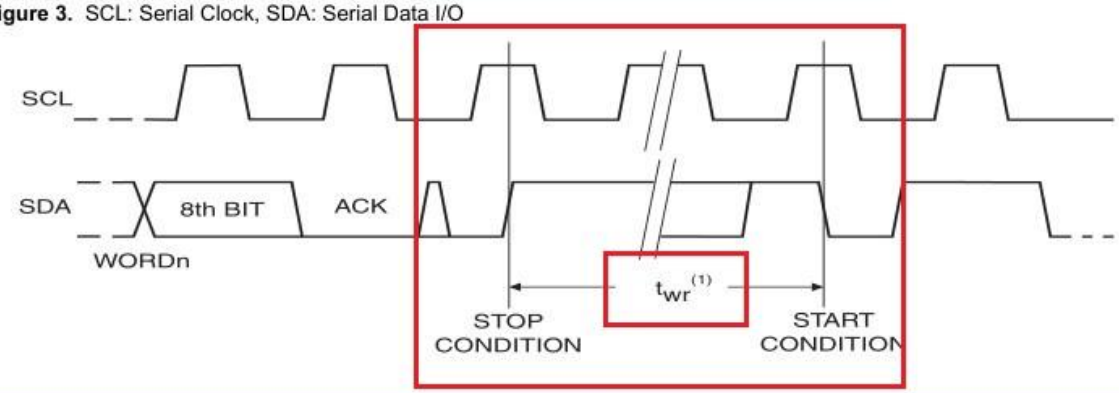
```
/**
  * @brief  Reads data from I2C EEPROM driver in using DMA channel.
  * @param  MemAddress: memory address
  * @param  pBuffer: Pointer to data buffer
  * @param  BufferSize: Amount of data to be read
  * @retval HAL status
  */                          连续 读 函数
HAL_StatusTypeDef AT24CXX_ReadData(uint16_t MemAddress, uint8_t* pBuffer, uint32_t BufferSize)
{
  return (I2Cx_ReadMultiple(AT24CXX_ADDRESS, MemAddress, AT24CXX_MEMADD_SIZE, pBuffer, BufferSize));
}
```

Symbol	Parameter	AT24C01A/02/04/08A/16A		Units
		Min	Max	
f_{SCL}	Clock Frequency, SCL		400	kHz
t_{LOW}	Clock Pulse Width Low	1.2		µs
t_{HIGH}	Clock Pulse Width High	0.6		µs
t_I	Noise Suppression Time(1)		50	ns
t_{AA}	Clock Low to Data Out Valid	0.1	0.9	µs
t_{BUF}	Time the bus must be free before a new transmission can start(2)	1.2		µs
$t_{HD.STA}$	Start Hold Time	0.6		µs
$t_{SU.STA}$	Start Set-up Time	0.6		µs
$t_{HD.DAT}$	Data In Hold Time	0		µs
$t_{SU.DAT}$	Data In Set-up Time	100		ns
t_R	Inputs Rise Time(2)		300	ns
t_F	Inputs Fall Time(2)		300	ns
$t_{SU.STO}$	Stop Set-up Time	0.6		µs
t_{DH}	Data Out Hold Time	50		ns
t_{WR}	Write Cycle Time		5	ms
Endurance(2)	5.0V, 25°C, Page Mode	1M		Write Cycles

Write Cycle Timing

每次 写 停止 到 下次 启动条件 之间 必须 延时

Figure 3. SCL: Serial Clock, SDA: Serial Data I/O



Note: 1. The write cycle time t_{WR} is the time from a valid stop condition of a write sequence to the end of the internal clear/write cycle.

```
/** 描述   : 在EEPROM的一个写循环中可以写多个字节，但一次写入的字节数
  *          不能超过EEPROM页的大小。AT24C04每页有16个字节。
  * 输入   : -pBuffer 缓冲区指针
  *          -MemAddress 接收数据的EEPROM的地址
  *          -BufferSize 要写入EEPROM的字节数        按页 写 函数
  */
HAL_StatusTypeDef AT24CXX_PageWrite(uint16_t MemAddress, uint8_t* pBuffer, uint32_t BufferSize)
{
  HAL_StatusTypeDef status = HAL_OK;

  status = I2Cx_WriteMultiple(AT24CXX_ADDRESS,MemAddress,AT24CXX_MEMADD_SIZE, pBuffer,BufferSize);

  AT24CXX_DELAY_MS(5);       /*  延时是必须的，否则通信失败  */

  return status;
}
```

此处 延时 必须 有

```
153 /**
154   * @brief  Camera writes single data.
155   * @param  Reg: Reg address
156   * @param  Value: Data to be written
157   */
158 void AT24CXX_WriteByte(uint8_t Reg, uint8_t Value)
159 {
160   I2Cx_Write(AT24CXX_ADDRESS, Reg, AT24CXX_MEMADD_SIZE, Value);
161
162   AT24CXX_DELAY_MS(5);    /*  延时是必须的，否则通信失败  */
163 }
164
165 /**
166   * @brief  Camera reads single data.
167   * @param  Reg: Reg address
168   * @retval Read data
169   */
170 uint8_t AT24CXX_ReadByte(uint8_t Reg)
171 {
172   return I2Cx_Read(AT24CXX_ADDRESS, Reg, AT24CXX_MEMADD_SIZE);
173 }
```

重新封装 函数

单字节 读 写 函数

此处 延时 必须有

```
56 static void I2Cx_Write(uint8_t Addr, uint8_t Reg, uint16_t MemAddSize, uint8_t Value)
57 {
58   HAL_StatusTypeDef status = HAL_OK;
59
60   status = HAL_I2C_Mem_Write(&heval_I2c, Addr, (uint16_t)Reg, MemAddSize, &Value, 1, I2C_TIMEOUT);
61
62   /* Check the communication status */
63   if(status != HAL_OK)
64   {
65     /* I2C error occured */
66     I2Cx_Error(Addr);
67   }
68 }
```

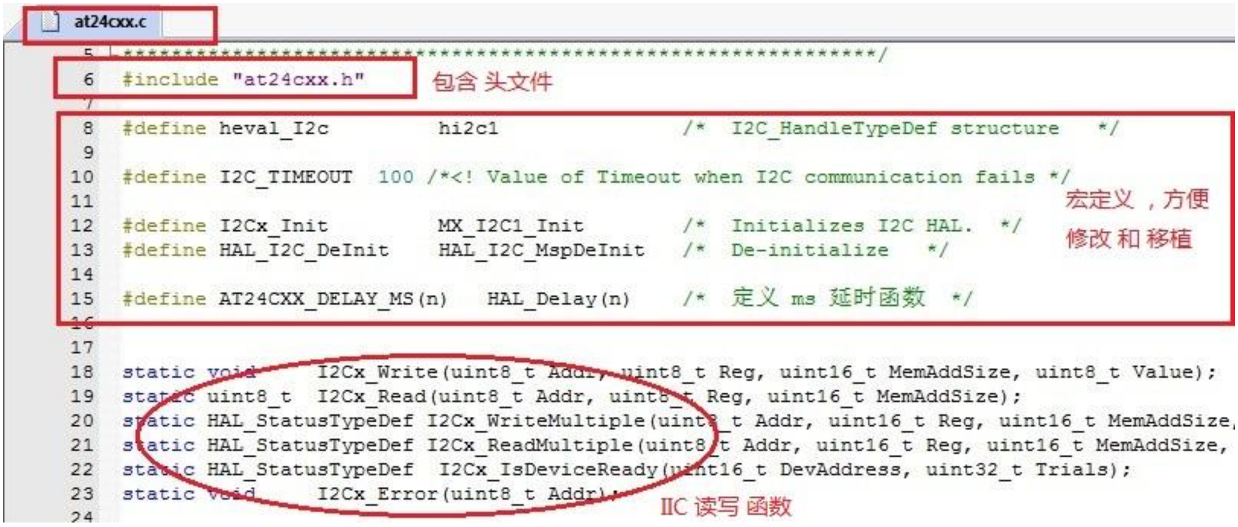
重新封装 函数

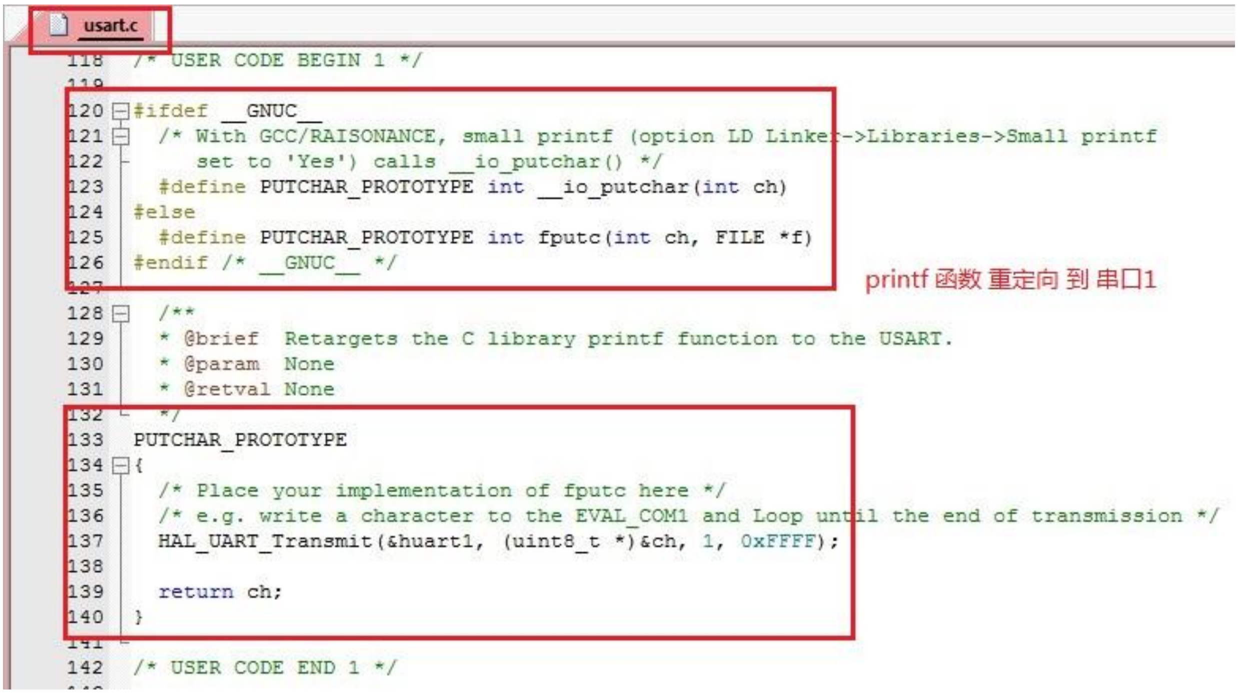
```
31   */
32 static uint8_t I2Cx_Read(uint8_t Addr, uint8_t Reg, uint16_t MemAddSize)
33 {
34   HAL_StatusTypeDef status = HAL_OK;
35   uint8_t Value = 0;
36
37   status = HAL_I2C_Mem_Read(&heval_I2c, Addr, Reg, MemAddSize, &Value, 1, I2C_TIMEOUT);
38
39   /* Check the communication status */
40   if(status != HAL_OK)
41   {
42     /* Execute user timeout callback */
43     I2Cx_Error(Addr);
44   }
45
46   return Value;
47 }
```

重新封装 函数

```
494 /** @addtogroup I2C_Exported_Functions_Group2
495   * @{
496   */
497 /* I/O operation functions  *****************************************************/
498 /******* Blocking mode: Polling */
499 HAL_StatusTypeDef HAL_I2C_Master_Transmit(I2C_HandleTypeDef *hi2c, uint16_t DevAddres
500 HAL_StatusTypeDef HAL_I2C_Master_Receive(I2C_HandleTypeDef *hi2c, uint16_t DevAddress
501 HAL_StatusTypeDef HAL_I2C_Slave_Transmit(I2C_HandleTypeDef *hi2c, uint8_t *pData, uin
502 HAL_StatusTypeDef HAL_I2C_Slave_Receive(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint
503 HAL_StatusTypeDef HAL_I2C_Mem_Write(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uin
504 HAL_StatusTypeDef HAL_I2C_Mem_Read(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint
505 HAL_StatusTypeDef HAL_I2C_IsDeviceReady(I2C_HandleTypeDef *hi2c, uint16_t DevAddress,
506
```

IIC 读写 存储器 的 HAL 库 函数


at24cxx.c
```c
/*****************************************************************/
#include "at24cxx.h"

#define heval_I2c          hi2c1               /*  I2C_HandleTypeDef structure   */

#define I2C_TIMEOUT  100 /*<! Value of Timeout when I2C communication fails */

#define I2Cx_Init          MX_I2C1_Init        /*  Initializes I2C HAL.  */
#define HAL_I2C_DeInit     HAL_I2C_MspDeInit   /*  De-initialize   */

#define AT24CXX_DELAY_MS(n)    HAL_Delay(n)    /*  定义 ms 延时函数  */


static void        I2Cx_Write(uint8_t Addr, uint8_t Reg, uint16_t MemAddSize, uint8_t Value);
static uint8_t  I2Cx_Read(uint8_t Addr, uint8_t Reg, uint16_t MemAddSize);
static HAL_StatusTypeDef I2Cx_WriteMultiple(uint8_t Addr, uint16_t Reg, uint16_t MemAddSize,
static HAL_StatusTypeDef I2Cx_ReadMultiple(uint8_t Addr, uint16_t Reg, uint16_t MemAddSize,
static HAL_StatusTypeDef  I2Cx_IsDeviceReady(uint16_t DevAddress, uint32_t Trials);
static void        I2Cx_Error(uint8_t Addr);
```



usart.c
```c
/* USER CODE BEGIN 1 */

#ifdef __GNUC__
  /* With GCC/RAISONANCE, small printf (option LD Linker->Libraries->Small printf
     set to 'Yes') calls __io_putchar() */
  #define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#else
  #define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif /* __GNUC__ */

  /**
  * @brief  Retargets the C library printf function to the USART.
  * @param  None
  * @retval None
  */
PUTCHAR_PROTOTYPE
{
  /* Place your implementation of fputc here */
  /* e.g. write a character to the EVAL_COM1 and Loop until the end of transmission */
  HAL_UART_Transmit(&huart1, (uint8_t *)&ch, 1, 0xFFFF);

  return ch;
}

/* USER CODE END 1 */
```

五、编写测试文件。

main.c | at24cxx.c | at24cxx.h

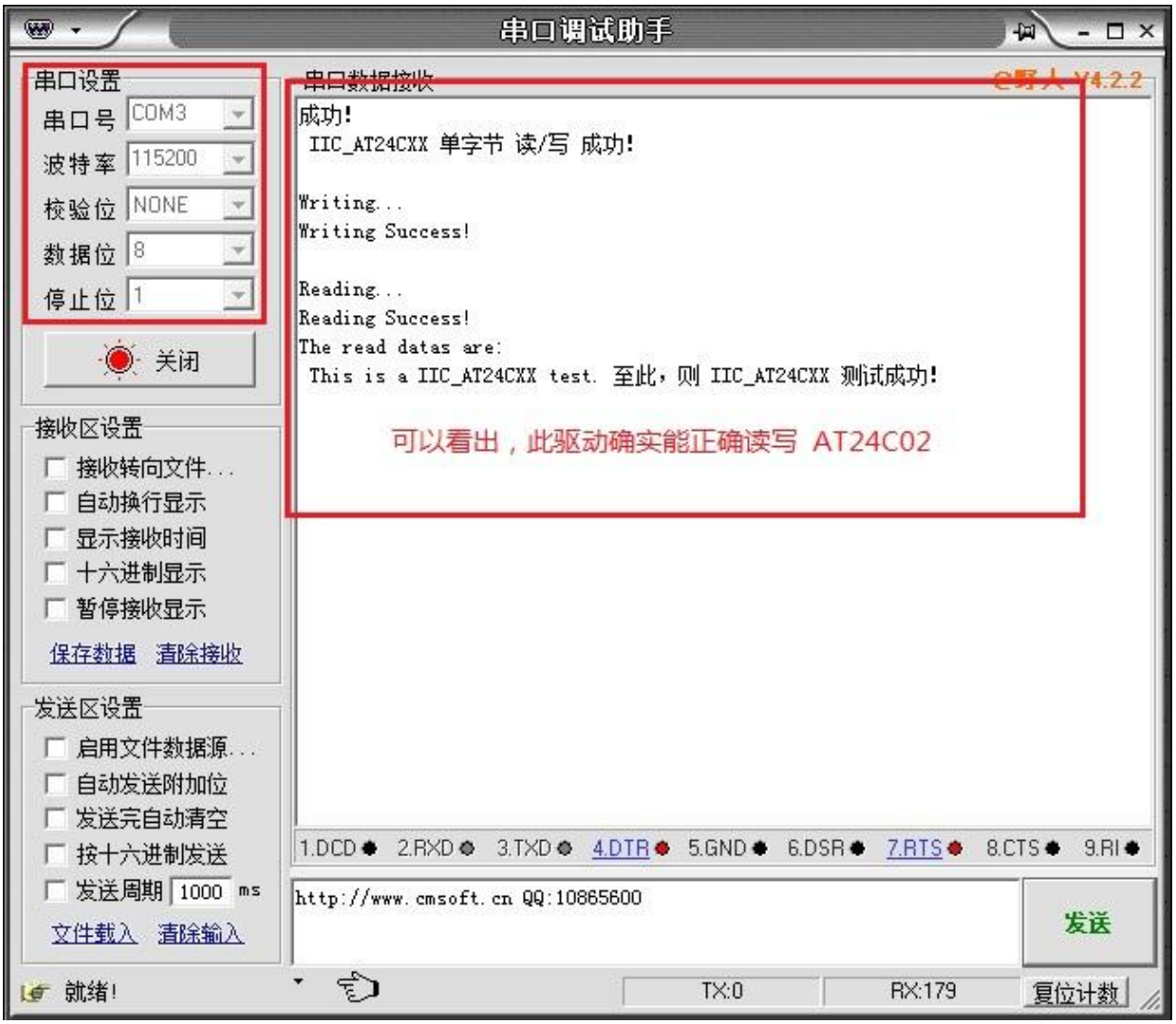
```
/* USER CODE BEGIN Includes */
#include "at24cxx.h"
/* USER CODE END Includes */

/* Private variables ---------------------------------------------------------*/

/* USER CODE BEGIN PV */
/* Private variables ---------------------------------------------------------*/

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
void Error_Handler(void);

/* USER CODE BEGIN PFP */
/* Private function prototypes -----------------------------------------------*/

/* USER CODE END PFP */

/* USER CODE BEGIN 0 */
uint8_t writebuf[]={"This is a IIC_AT24CXX test. 至此，则 IIC_AT24CXX 测试成功！"};
#define SIZE  sizeof(writebuf)
/* USER CODE END 0 */

int main(void)
{

  /* USER CODE BEGIN 1 */
  uint8_t readbuf[SIZE];
  /* USER CODE END 1 */
```

包含 头文件

```
  /* USER CODE BEGIN 2 */

  while(AT24CXX_Check())
  {
    printf("失败！\n IIC_AT24CXX 单字节 读/写 失败！\n");
    HAL_Delay(500);
  }
  printf("成功！\n IIC_AT24CXX 单字节 读/写 成功！\n\n");

  printf("Writing...\n");
  AT24CXX_WriteData(10, writebuf, SIZE);
  printf("Writing Success!\n\n");

  printf("Reading...\n");
  AT24CXX_ReadData(10, readbuf, SIZE);
  printf("Reading Success!\n");

  printf("The read datas are:\n %s\n",readbuf);

  /* USER CODE END 2 */
```

检测 是否 存在 AT24C02

六、测试结果。



七、总结。

STM32 的硬件 IIC 能够正确读写 AT24C02，使用 HAL 库只需要两个 API 函数，而且我们也不用关心具体的实现过程，十分方便。

学习 STM32 的最好方法就是学习官方列程。