

## 二维码开源库 ZBar-吐槽篇

前不久在网上看到一篇文章《QR-Decoder-OV5640 二维码识别》，是某开发板的教程。记得对应的开发板以前购买过，当初只是为了看 OV5640 的 JPG 的输出效果，结果由于公司奇葩的漫长采购流程，开发板还没到，方案早就改了。所以开发板到手后就立马压箱底了，吃灰吃了一年多。最近又闲得快失业了，对 QR 码有点兴趣，于是翻箱倒柜找到开发板，上网下载了对应例程，打开例程后，果然跟教程说的一样，只有库文件，没有 QR 解码库的源码，用了开源代码还打包成库，而且用了哪个解码库也只字未提，呵呵。教程原话如下：

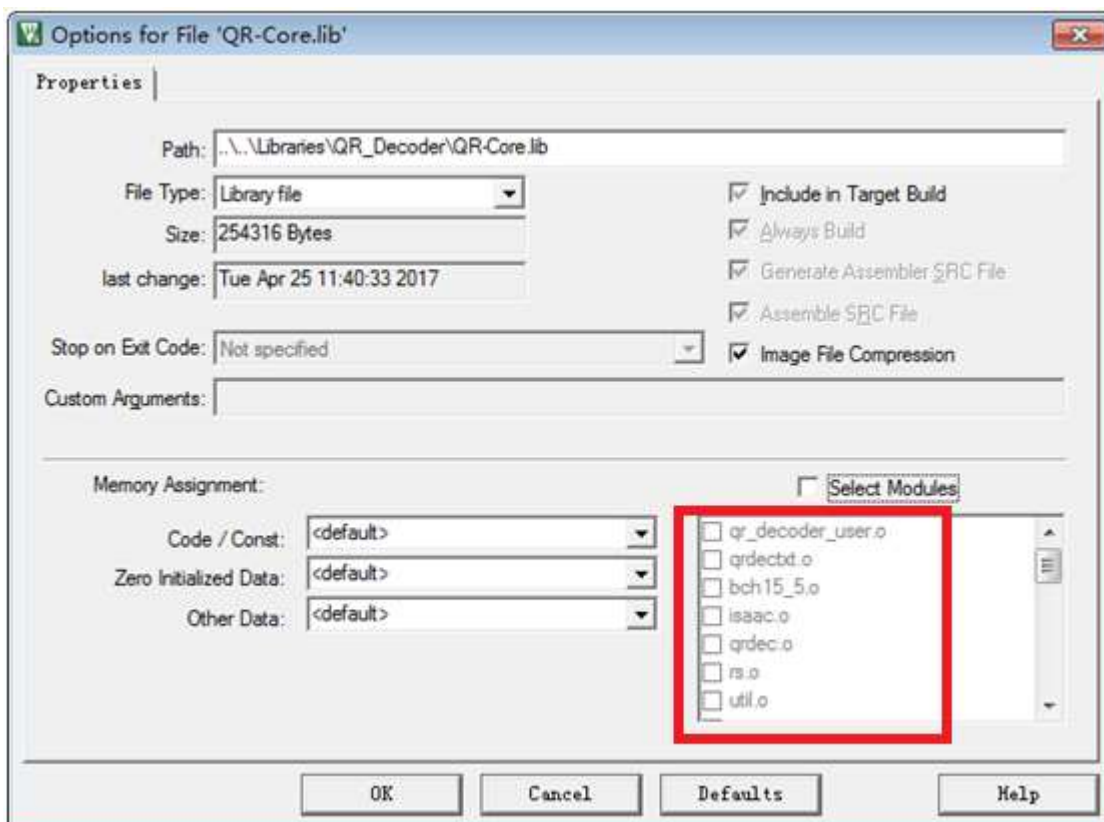
### 50.5.1 QR-Code 解码库特点

QR-Code 解码库是秉火专门针对 STM32F429 移植的一个的条码解码库，因为其结构复杂，移植过程繁琐，所以打包为一个解码库，提供接口方便用户直接调用，提高开发的效率。其主要特点如下：

- ☐ 条码种类：支持常用 QR-Code、EAN、UPC
- ☐ 扫描速度：400 毫秒
- ☐ 扫描英文：250 个字符
- ☐ 扫描中文：90 中文字符，UTF-8 编码格式(需上位机支持)
- ☐ 多码扫描：支持多个二维码同时解码，同时输出结果

百度了一下，也只能用度娘了。当前比较流行的解码库是 Zxing 和 ZBar，这里就不介绍了，自行了解。看了例程里面的解码库，如下图，很明显是用了 ZBar，源文件文件名和 ZBar 的源码一模一样。

(<http://zbar.sourceforge.net/>)



又搜索了一番，貌似找不到比较完善在 Windows 下编译和使用 ZBar 的记录，更不用说把 Zbar 移植到 ARM 之类的芯片。所以打算记录一下 ZBar 编译移植和使用过程，分享给有需要的人。

吐槽完了，先挖两个坑，一共分两部分：

一：ZBar 在 window 下的编译和使用

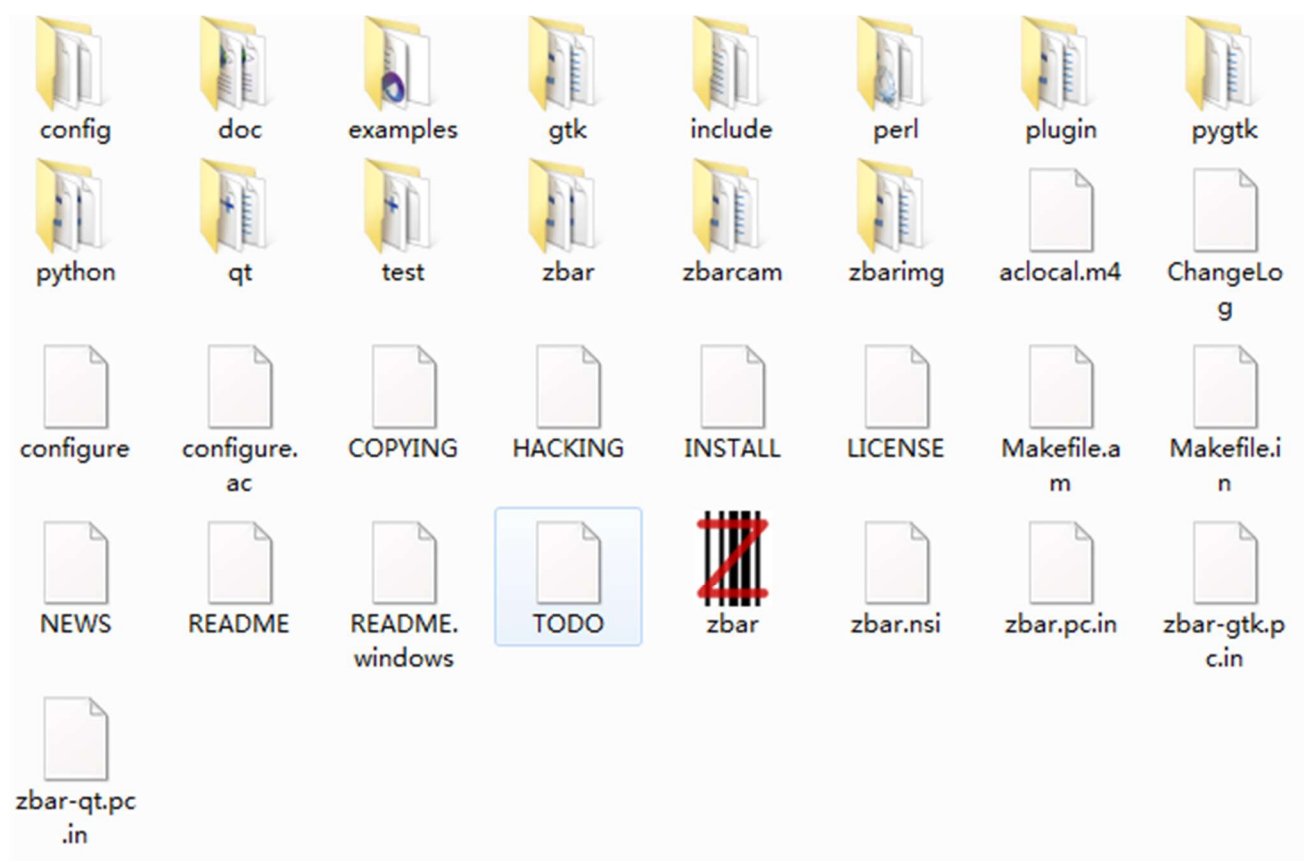
二：ZBar 的移植，基于 STM32F4

## 二维码开源库 ZBar-windows 下编译和使用

### 源码

下载最新 Zbar 源码 (<http://zbar.sourceforge.net/>)，网站的 WIKI 是空白的，所以只能在源码包里找使用说明了，很遗憾 Windows 下怎么编译没说明，只是说明了 Windows 安装包的使用（可能看得不够仔细，没找到），源码包里面有个 VS 的工程同样是用库文件的。网上搜了一下，也都是使用官网 Windows 安装包，然后调用安装目录的库文件的例程，直接使用源码的找不到。倒是刚好找到了一个 STM32F4 的版本

(<http://www.openedv.com/forum.php?mod=viewthread&tid=82582&page=1>)，按照其说明貌似是有问题，不能使用的，但能编译，至少还有些参考价值，比如需要使用到源码包里面的那些文件。源码目录如下，需要使用的文件，基本在 zbar 文件夹内。



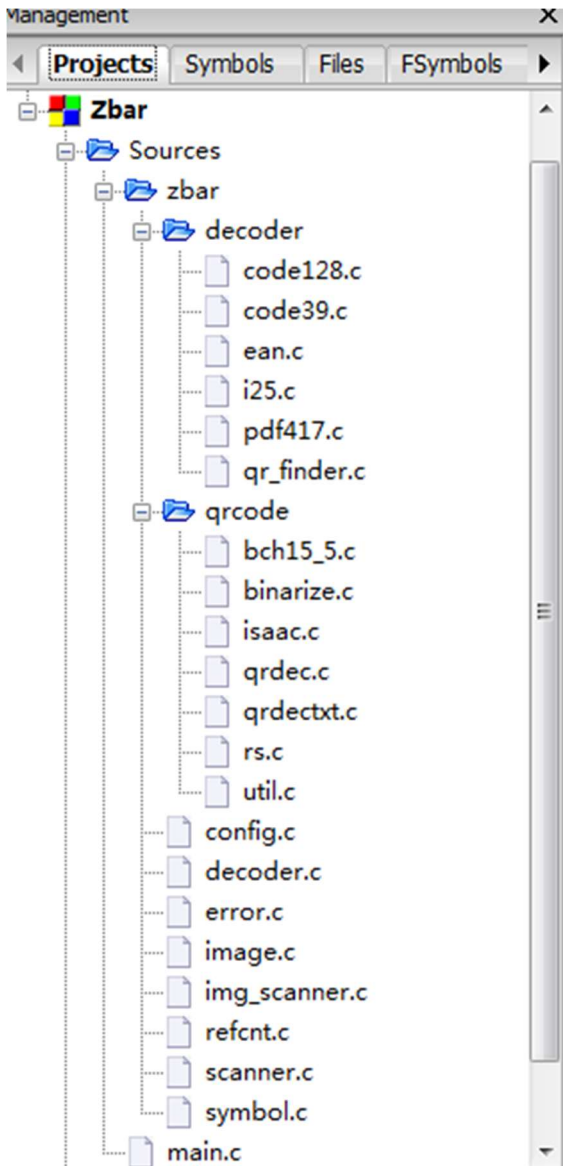
### 移植过程

开始时使用的 IDE 是 VS，由于平时会写一些上位机和小工具，所以第一时间想到的是 VS。但经过一番折腾还是投降了。因为 Zbar 是用 C99 写的，还有用了不少 GCC 的语法，使用 VC 编译不了。于是下载了 CodeBlock，下载是 codeblocks-17.12mingw-setup.exe，带 mingw，包含 GCC 和 GDB，链接 <http://www.codeblocks.org/downloads/26>。

以下是移植记录：

- 1: 新建 Console 工程
- 2: 复制 zbar-0.10\include\zbar.h 到 zbar-0.10\zbar\, 并且把 zbar 文件夹复制到工程目录
- 3: 添加 zbar-0.10\zbar\decoder 和 zbar-0.10\zbar\qrcode 目录下的源码
- 4: 添加 zbar-0.10\zbar\目录下的 config.c,decoder.c,error.c,image.c, img\_scanner.c, refcnt.c, scanner.c, symbol.c

工程结构如下:



- 5: 添加相关头文件路径, 编译报错

Zbar\zbar\config.c|24|fatal error: config.h: No such file or directory|

搜遍整个源码目录也没看到有 config.h。查看了源码包根目录下 INSTALL 文件, config.h 应该是由 autoconf 工具生成的, 但这东西又是 linux 或 unix 下的, 于是拷贝 ZBar 到 Linux 下, 按照 INSTALL 的说明生成 config.h, 再添加回 windows 下的工程目录,



```
1 /* include/config.h.  Generated from config.h.in by configure.  */
2 /* include/config.h.in.  Generated from configure.ac by autoheader.  */
3
4 /* whether to build support for Code 128 symbology */
5 #define ENABLE_CODE128 1
6
```

```

7 /* whether to build support for Code 39 symbology */
8 #define ENABLE_CODE39 1
9
10 /* whether to build support for EAN symbologies */
11 #define ENABLE_EAN 1
12
13 /* whether to build support for Interleaved 2 of 5 symbology */
14 #define ENABLE_I25 1
15
16 /* whether to build support for PDF417 symbology */
17 #define ENABLE_PDF417
18
19 /* whether to build support for QR Code */
20 #define ENABLE_QRCODE 1
21
22 /* Program major version (before the '.') as a number */
23 #define ZBAR_VERSION_MAJOR 0
24
25 /* Program minor version (after '.') as a number */
26 #define ZBAR_VERSION_MINOR 10

```



编译。

报错

Zbar\zbar\qrcode\qrdectx.c|9|fatal error: iconv.h: No such file or directory|

Iconv 同样是 linux 下的东西，Windows 编译 Linux 下的源码就是苦逼。

下载并安装 libiconv-1.9.2-1.exe (<http://gnuwin32.sourceforge.net/packages.html>)

将安装目录下的 iconv.h, libcharset.h, localcharset.h, libiconv2.dll 拷贝工程目录下，并设置相关头文件和 lib 路径

重新编译后

报错 Zbar\zbar\image.c|217|error: expected ')' before 'PRIx32'|

PRIx32 是在 inttypes.h 定义的，包含该头文件后再编译，终于不报错了。

6: 编译通过后，那怎么使用呢？查看 zbar-0.10\examples 目录下文件，里面是一些使用例程，最终挑选了 zbar-0.10\examples\scan\_image.c，将其内容复制粘贴到 main.c, 重新编译

报错

Zbar\main.c|3|fatal error: png.h: No such file or directory|

下载并安装 libpng-1.2.37-setup.exe (<http://gnuwin32.sourceforge.net/packages.html>)

将安装目录下的 png.h, pngconf.h, libpng12.dll 拷贝工程目录下，并设置相关头文件和 lib 路径

重新编译

报错

png\include\png.h|477|fatal error: zlib.h: No such file or directory|

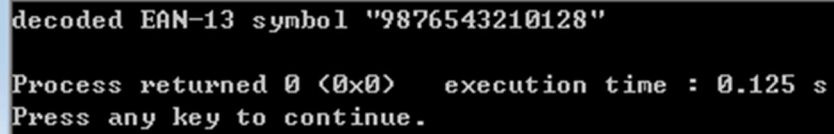
下载并安装 zlib-1.2.3.exe (<http://gnuwin32.sourceforge.net/packages.html>)

将安装目录下的 zconf.h, zlib.h, zlib1.dll 拷贝工程目录下，并设置相关头文件和 lib 路径

重新编译。

通过。

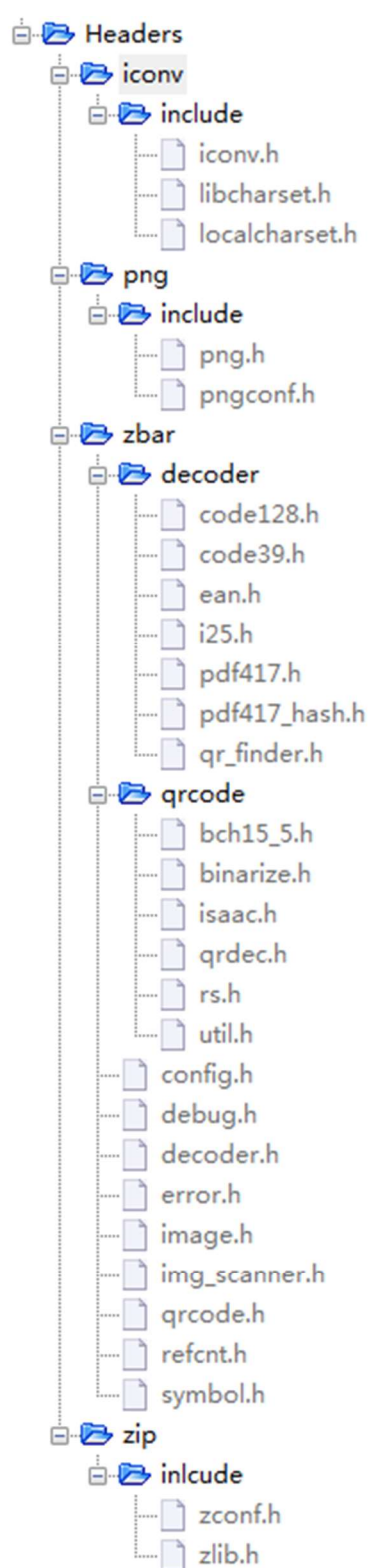
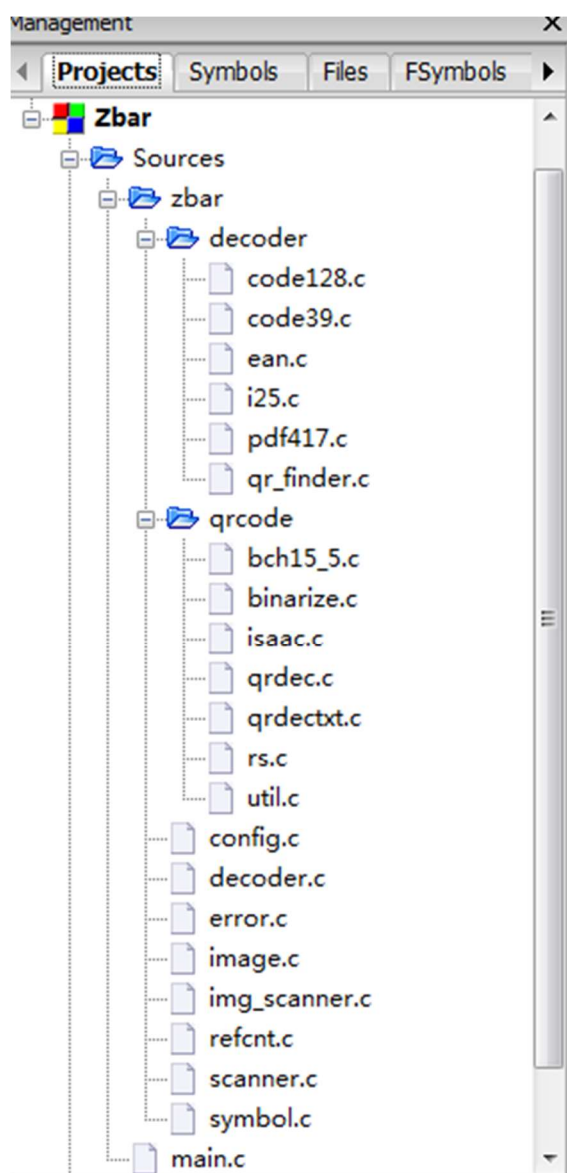
7: 将 zbar-0.10\examples\barcode.png 复制到工程根目录, 对 main.c 做一下修改  
屏蔽 if(argc < 2) return(1);  
get\_data(argv[1], &width, &height, &raw);改为 get\_data("barcode.png", &width, &height, &raw);  
重新编译运行后, 窗口输出如下:



```
decoded EAN-13 symbol "9876543210128"  
  
Process returned 0 (0x0)   execution time : 0.125 s  
Press any key to continue.
```

到此, ZBar 基本能正常运行了。

最终的工程结构如下:



分类: Zbar

# 二维码开源库 ZBar-实现中文解码

## 中文乱码

上篇《ZBar-windows 下编译和使用》已经成功解析了条形码，但目标是二维码，经测试二维码中文会出现乱码。

下图二维码的内容是“http123 测试 456”，解析后的内容为“http123 嫵嫵嫵 456”



搜索了一下关键词，解决方案如下 <http://blog.csdn.net/zizi7/article/details/51880129>

修改文件 zbar/qrcode/qrdectx.c:

```
1 latin1_cd=iconv_open("GB18030", "UTF-8");
2 /*But this one is often used, as well.*/
3 sjis_cd=iconv_open("GB2312", "UTF-8");
4 /*This is a trivial conversion just to check validity without extra code.*/
5 utf8_cd=iconv_open("UTF-8", "UTF-8");
6 ...
7 enc_list[1]=sjis_cd;
8 enc_list[0]=latin1_cd;
9 enc_list[2]=utf8_cd;
```

重新编译运行后，正确输出“http123 测试 456”

```
decoded QR-Code symbol "http123测试456"
Process returned 0 (0x0)   execution time : 0.088 s
Press any key to continue.
```

## 自己实现中文解码

ZBar 解析后的字符原始输出是 UTF-8 格式，然后使用了 iconv 将其转换为相应的字符编码，但最终目标是移植到 STM32F4 上，如果要直接输出中文编码，有几种方案：



- 1. 把 iconv 移植到 STM32F4 上
- 2. 自己实现 UTF8-8 转中文编码
- 3. 把编码工作交给上位机

字符集和编码格式

搜索了一下字符编码规则，觉得方案 2 比方案 1，3 更容易实现。这里先简单介绍下与本文相关的字符集和编码格式。

1. Unicode

Unicode 是字符集，也叫万国码，顾名思义就是包含所有国家的文字。

2. GB18030

GB18030 是中文字符集，可以认为是 Unicode 的一个子集，还有其他 GBXXXXX 的中文字符集，他们的关系简单来说就是包含的中文字符个数不一样。简单起见，这里只是使用 2 个字节的 GB18030，一共 20902 个汉字，也基本覆盖常见的汉字了。书读得少，4 个字节的汉字也没认识几个。

3. UTF-8

UTF-8 是 Unicode 字符集的一种编码格式，还有其他 UTF-16，UTF-32，ZBar 使用了 UTF-8。

UTF-8 最大的一个特点，就是它是一种变长的编码方式。它可以使用 1~6 个字节表示一个符号，根据不同的符号而变化字节长度。

UTF-8 的编码规则很简单，只有二条：

- 1）对于单字节的符号，字节的第一位设为 0，后面 7 位为这个符号的 unicode 码。因此对于英语字母，UTF-8 编码和 ASCII 码是相同的。
- 2）对于 n 字节的符号（n>1），第一个字节的前 n 位都设为 1，第 n+1 位设为 0，后面字节的前两位一律设为 10。剩下的没有提及的二进制位，全部为这个符号的 unicode 码。  
(具体例子可以参考 <http://blog.csdn.net/xiaolei1021/article/details/52093706>)

下表总结了编码规则，字母 x 表示可用编码的位。

n	Unicode rang(hex)	UTF-8 encode(bin)
1	0000 0000 - 0000 007F	0xxxxxxx
2	0000 0080 - 0000 07FF	110xxxxx 10xxxxxx
3	0000 0800 - 0000 FFFF	1110xxxx 10xxxxxx 10xxxxxx
4	0001 0000 - 0010 FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
5	0020 0000 - 03FF FFFF	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
6	0400 0000 - 7FFF FFFF	1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

4. GB18030，Unicode，UTF-8 的关系

GB18030-2005	Unicode 5.0	对应的 Unicode 编码	UTF-8 编码字节数
CJK 统一汉字的 20902 汉字	CJK 统一汉字的 20902 汉字	0x4E00-0x9FA5	2
CJK 统一汉字扩充 A 的 6582 汉字	CJK 统一汉字扩充 A 的 6582 汉字	0x3400-0x4DB5	4
CJK 统一汉字扩充 B 的 42711 汉字	CJK 统一汉字扩充 B 的 42711 汉字	0x20000-0x2A6D6	4

UTF-8 转 GB18030 实现

了解相关字符集和编码格式，可以开始写转换代码了。



1. 需要一个 GB18030 字符集，其实就是一个数组，实现代码如下（VS 下编译）  
调用 UnicodeToGB18030Table 函数生成一个 GB18030 字符集数组。

```
1 char* UnicodeToGB18030String(const wchar_t* unicode_str)
2 {
3     UINT code_page = 54936 ; //GB2312 :936   GB18030: 54936
4     int len=WideCharToMultiByte(code_page,0,unicode_str,-1,NULL,0,NULL,NULL);
5     char* buf=new char[len+1];
6     WideCharToMultiByte(code_page,0,unicode_str,-1,buf,len,NULL,NULL);
7     buf[len]=0;
8     return buf;
9 }
10 int UnicodeToGB18030Table(void)
11 {
12     FILE *table;
13     wchar_t unicode[]={0x4E00,0};
14     char* gb18030;
15     int cnt=0;
16     table = fopen("unicode_to_gb18030_table.c","w");
17     if(table == NULL)
18     {
19         printf("can not open unicode_to_gb18030_table.c\n");
20         system("pause");
21         exit(0);
22     }
23     fprintf(table, "%s", "const char unicode_to_gb18030_table1[] = {\n");
24     for(unicode[0]=0x4E00; unicode[0]<=0x9FA5; unicode[0]++)
25     {
26         gb18030 = UnicodeToGB18030String(unicode);
27
28         if(unicode[0]==0x9FA5)
29         {
30             fprintf(table, "0x%X,0x%X ", (UINT8)gb18030[0], (UINT8)gb18030[1]);
31         }
32         else
33         {
34             fprintf(table, "0x%X,0x%X, ", (UINT8)gb18030[0], (UINT8)gb18030[1]);
35         }
36
37         cnt ++;
38         if(cnt == 16)
39         {
40             cnt = 0;
41             fprintf(table, "\n");
42         }
43     }
44
45     fprintf(table, "\n};");
```

```

46     fclose(table);
47 }

```

## 2. 通过查表，将 UTF-8 转为 GB18030

```

1 int zbar_utf8_to_gbl8030 (uint8_t* utf8_code, uint32_t utf8_len, uint8_t* gbl8030)
2 {
3     uint8_t utf8_bytes[3]; //该数组最大为 6 个字节，但这里只考虑 3 个字节的中文编码
4     uint32_t i = 0, j = 0;
5     uint16_t unicode_value;
6     uint8_t* unicode = gbl8030;
7
8     for(i=0; i< utf8_len; i+=3) {
9         utf8_bytes[0] = utf8_code[i+0] & 0x0F;
10        utf8_bytes[1] = utf8_code[i+1] & 0x3F;
11        utf8_bytes[2] = utf8_code[i+2] & 0x3F;
12
13        unicode[j] = (utf8_bytes[1] >> 2) | ((utf8_bytes[0]) << 4);
14        unicode[j+1] = utf8_bytes[2] | ((utf8_bytes[1] & 0x03) << 6);
15        unicode_value = (unicode[j]<<8) + unicode[j+1];
16        if(unicode_value>=0x4E00) {
17            gbl8030[j] = unicode_to_gbl8030_table1[(unicode_value-0x4E00)*2];
18            gbl8030[j+1] = unicode_to_gbl8030_table1[(unicode_value-0x4E00)*2 + 1];
19        }
20        j += 2;
21    }
22    return 0;
23 }

```

中文字符集和编码转码函数有了，下一步就是替换 ZBar 源码的编码转换部分。

删掉 zbar/qrcode/qrdectx.c 中 iconv 相关的代码，将 zbar\_utf8\_to\_gbl8030 函数加入 qr\_code\_data\_list\_extract\_text 函数中：

```

1 int qr_code_data_list_extract_text(const qr_code_data_list *_qrlist,
2                                   zbar_image_scanner_t *iscn,
3                                   zbar_image_t *img)
4 {
5     ....
6     case QR_MODE_BYTE: {
7         int gbl8030_cnt = zbar_utf8_to_gbl8030(entry->payload.data.buf,
8         entry->payload.data.len, sa_text+sa_ncontext);
9         sa_ncontext += gbl8030_cnt;
10    }
11    break;
12    ....

```

12 }



重新编译运行后

正确输出"http123 测试 456"

```
decoded QR-Code symbol "http123测试456"
```

```
Process returned 0 (0x0)   execution time : 0.088 s  
Press any key to continue.
```

坐等下班，回家过年.....

# 二维码开源库 ZBar-MDK STM32F429 移植

前两篇文章已经实现 ZBar 在 Windows 平台下的编译和使用，本文将介绍如何把 ZBar 移植到 STM32F429，IDE 使用 MDK。

## 1. MDK 工程设置

(1) 不勾选 Use MicroLIB，使用 ISO C。

如 MDK 帮助文档关于 MicroLIB 的介绍，故在硬件资源允许的情况还是优先使用 ISO C。

### About microlib

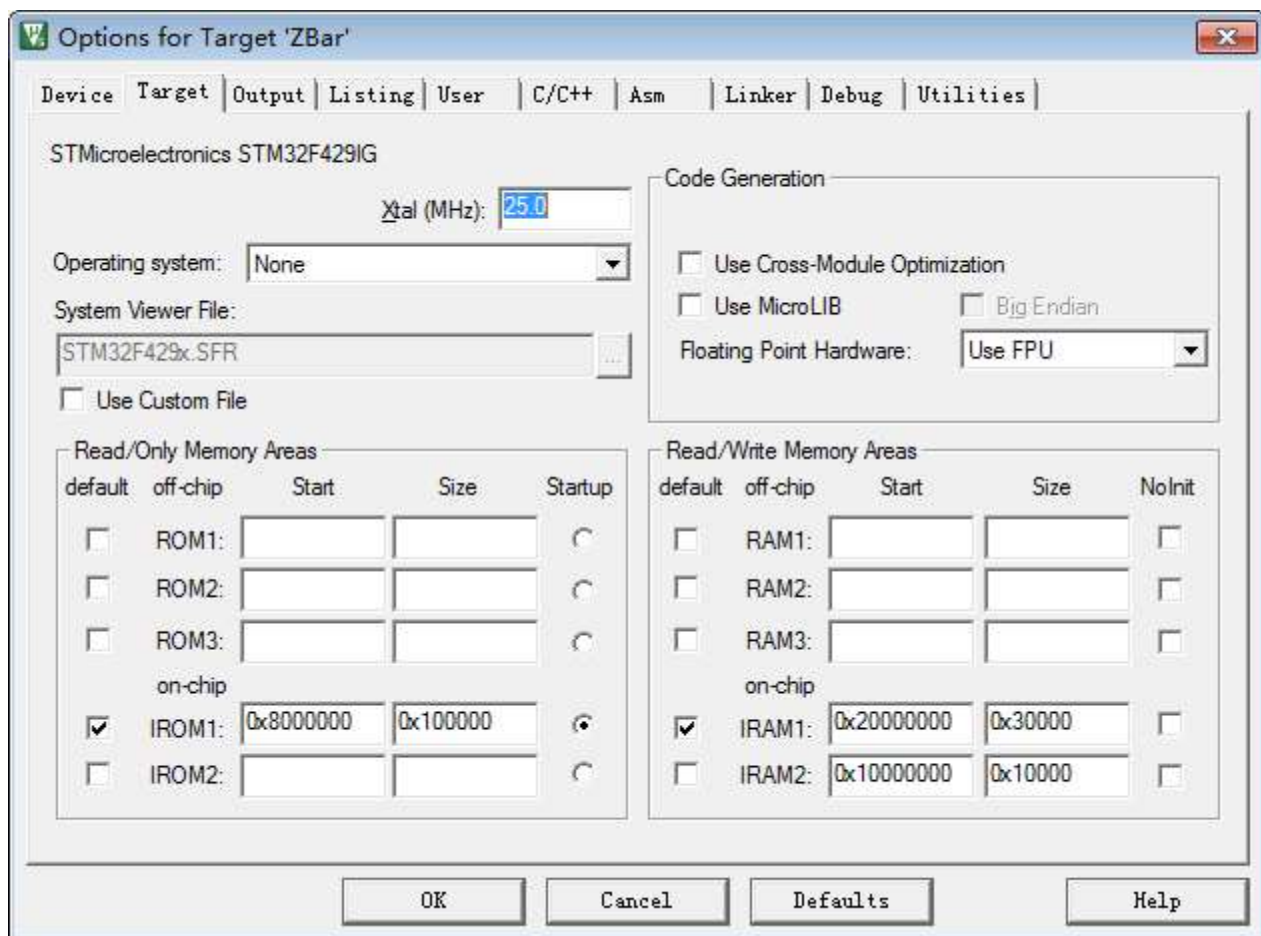
#### 2.1 About microlib

Microlib is an alternative library to the default C library. It is intended for use with deeply embedded applications that must fit into extremely small memory footprints.

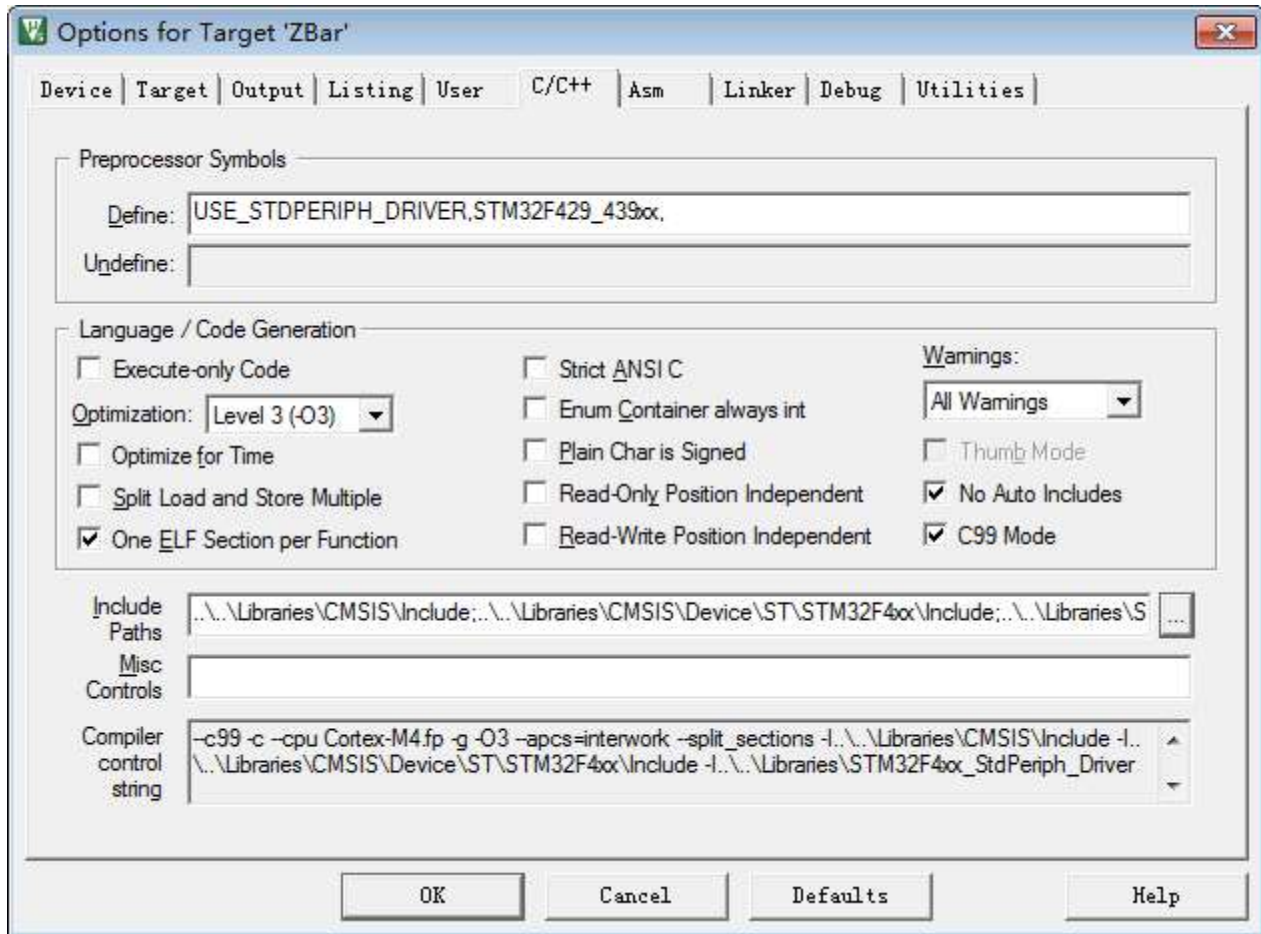
These applications do not run under an operating system.

#### Note

Microlib does not attempt to be an ISO C-compliant library. Microlib is highly optimized for small code size. It has less functionality than the default C library and some ISO C features are completely missing. Some library functions are also slower.



(2) 勾选 C99 Mode，因为 ZBar 源码是基于 C99 的



(3) 不勾选 Use Memory Layout From Target Dialog, 使用自定义 Scatter File

Options for Target 'ZBar'

Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

☐ Use Memory Layout from Target Dialog

☐ Make RW Sections Position Independent

☐ Make RO Sections Position Independent

☐ Don't Search Standard Libraries

☒ Report 'might fail' Conditions as Errors

X/O Base:

R/O Base:

R/W Base:

disable Warnings:

Scatter File:  ... Edit...

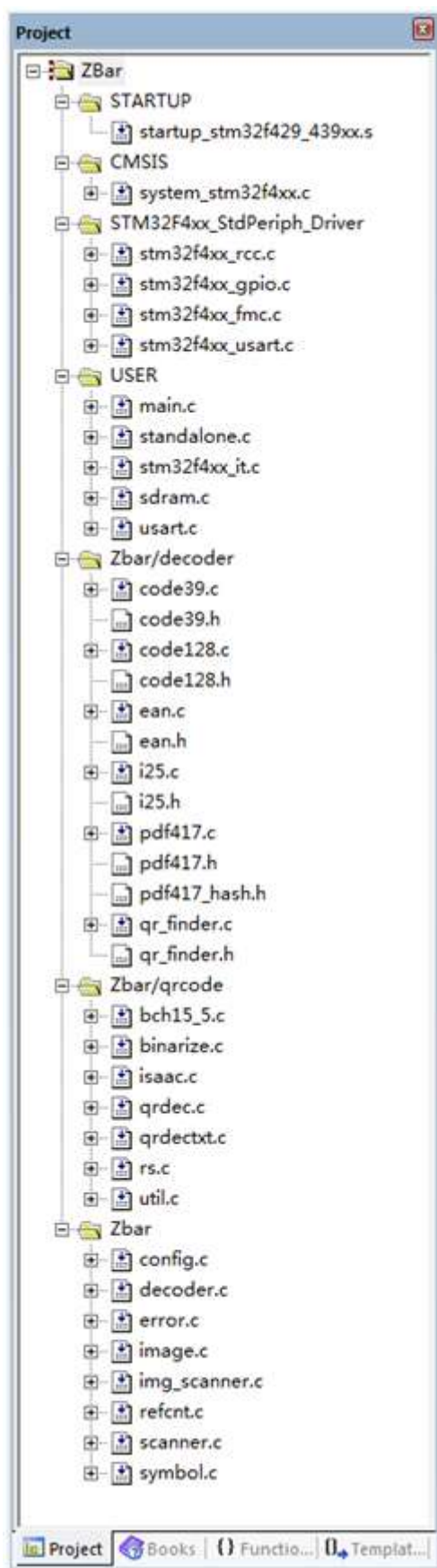
Misc controls:

Linker control string:

OK Cancel Defaults Help



## 2.工程目录结构



### 3. ZBar 源码修改地方

由于 MDK 使用的编译器不是 GCC，故不能支持一些特殊 GCC 语法和相应的 C 库函数。

(1) error.c 和 error.h 文件中 strdup 函数报错。

处理方法：屏蔽该函数相关代码或者自己实现 strdup 函数

(2) 屏蔽 img\_scanner.c 文件中的

```
.....
//#include <unistd.h>
//#include <time.h> /* clock_gettime */
//#include <sys/time.h> /* gettimeofday */
.....
// struct timeval abstime;
// gettimeofday(&abstime, NULL);
// iscn->time = (abstime.tv_sec * 1000) + ((abstime.tv_usec / 500) + 1) / 2;
.....
```

(3) GCC 默认 void\* 为 char\*，MDK 的编译器对 void 指针的加减操作当作错误处理

处理方法：把 void 指针强制转换为 char 指针

### 4. 系统存储空间分配

对于 STM32 来说 ZBar 对 Heap 消耗较大，经测试需要 2M 左右的 Heap，具体 ZBar 对 RAM 的要求暂时没深究。

Scatter file 如下

```
LR_IROM1 0x08000000 0x00100000 { ; load region size_region
    ER_IROM1 0x08000000 0x00100000 { ; load address = execution address
        *.o (RESET, +First)
        *(InRoot$$Sections)
        .ANY (+RO)
    }

    RW_IRAM_DATA 0x20000000 0x00030000 { ; RW data
        .ANY (+RW +ZI)
    }

    ARM_LIB_STACK 0x20030000 EMPTY -(0x20030000 - ImageLimit(RW_IRAM_DATA)) ; Stack
    region growing down
    { }

    RW_ERAM 0xD0000000 0x00800000 { ;Extern SDRAM
        main.o(+RW +ZI)
```

```

}

ARM_LIB_HEAP +0 EMPTY (0xD0800000 - ImageLimit(RW_ERAM)) ; Heap region growing up
{ }
}

```

## 5. 修改默认的启动文件 **startup\_stm32f429\_439xx.s**

(1) **Stack** 和 **Heap** 已经在 **Scatter** 文件中指定，故屏蔽 **startup\_stm32f429\_439xx.s** 中 **Stack** 和 **Heap** 的相关代码。

```

;Stack_Size      EQU      0x00000400

;AREA            STACK, NOINIT, READWRITE, ALIGN=3
;Stack_Mem       SPACE    Stack_Size
;__initial_sp


;; <h> Heap Configuration
;;   <o>   Heap Size (in Bytes) <0x0-0xFFFFFFFF:8>
;;   </h>

;Heap_Size       EQU      0x00000200

;AREA            HEAP, NOINIT, READWRITE, ALIGN=3
;__heap_base
;Heap_Mem        SPACE    Heap_Size
;__heap_limit

.....

;IF              :DEF:__MICROLIB

;EXPORT          __initial_sp
;EXPORT          __heap_base
;EXPORT          __heap_limit

;ELSE

;IMPORT          __use_two_region_memory
;EXPORT          __user_initial_stackheap

;__user_initial_stackheap

;LDR             R0, = Heap_Mem
;LDR             R1, =(Stack_Mem + Stack_Size)

```

```

;LDR    R2, = (Heap_Mem + Heap_Size)
;LDR    R3, = Stack_Mem
;BX     LR

;ALIGN

;ENDIF

```

(2) 把 **Scatter** 文件指定的 **Stack** 基地址赋给向量表的首地址

```

.....
__Vectors      IMPORT | Image$$ARM_LIB_STACK$$Base |
of Stack      DCD     | Image$$ARM_LIB_STACK$$Base |; __initial_sp           ; Top

```

(3) 因为 **Heap** 分配在外部 **SDRAM** 中，故在进入 **\_\_main** 前需初始化 **SDRAM**

```

.....
IMPORT SystemInit
IMPORT SDRAM_Init
IMPORT __main

LDR    R0, =SystemInit
BLX    R0
LDR    R0, =SDRAM_Init
BLX    R0
LDR    R0, =__main
BX     R0
ENDP

```

## 6.测试代码

之前 **Windows** 平台下的测试例程是直接读取 **PNG** 格式的二维码，在 **STM32F429** 平台下没有移植文件系统和 **PNG** 解码库，为了简单起见，直接把待

测试的二维码图片转换为二值化数组，然后把该数组传给 **ZBar**。

```

#include "type_define.h"
#include "usart.h"
#include "sdrum.h"
#include "zbar.h"
#include "image.h"
//280X280 的二维码图片灰度值数组
const uint8_t image_data_buf[] = {

```

```

.....
};
int Zbar_Test(void* raw, int width, int height)
{
    zbar_image_scanner_t *scanner = NULL;
    /* create a reader */
    scanner = zbar_image_scanner_create();

    /* configure the reader */
    zbar_image_scanner_set_config(scanner, 0, ZBAR_CFG_ENABLE, 1);

    /* obtain image data */
    //int width = 0, height = 0;
    //void *raw = NULL;
    //get_data("barcode.png", &width, &height, &raw);

    /* wrap image data */
    zbar_image_t *image = zbar_image_create();
    zbar_image_set_format(image, *(int*)"Y800");
    zbar_image_set_size(image, width, height);
    zbar_image_set_data(image, raw, width * height, zbar_image_free_data);

    /* scan the image for barcodes */
    int n = zbar_scan_image(scanner, image);
    printf("n = %d\r\n", n);
    /* extract results */
    const zbar_symbol_t *symbol = zbar_image_first_symbol(image);
    for(; symbol; symbol = zbar_symbol_next(symbol)) {
        /* do something useful with results */
        zbar_symbol_type_t typ = zbar_symbol_get_type(symbol);
        const char *data = zbar_symbol_get_data(symbol);
        printf("decoded %s symbol \"%s\"\r\n",
            zbar_get_symbol_name(typ), data);
        printf("len = %d\r\n", strlen(data));
        for(int k=0; k<strlen(data); k++)
        {
            printf("%X ", (uint8_t)data[k]);
        }
    }

    /* clean up */
    zbar_image_destroy(image);
    zbar_image_scanner_destroy(scanner);

    return(0);
}

int main(void)
{

```

```

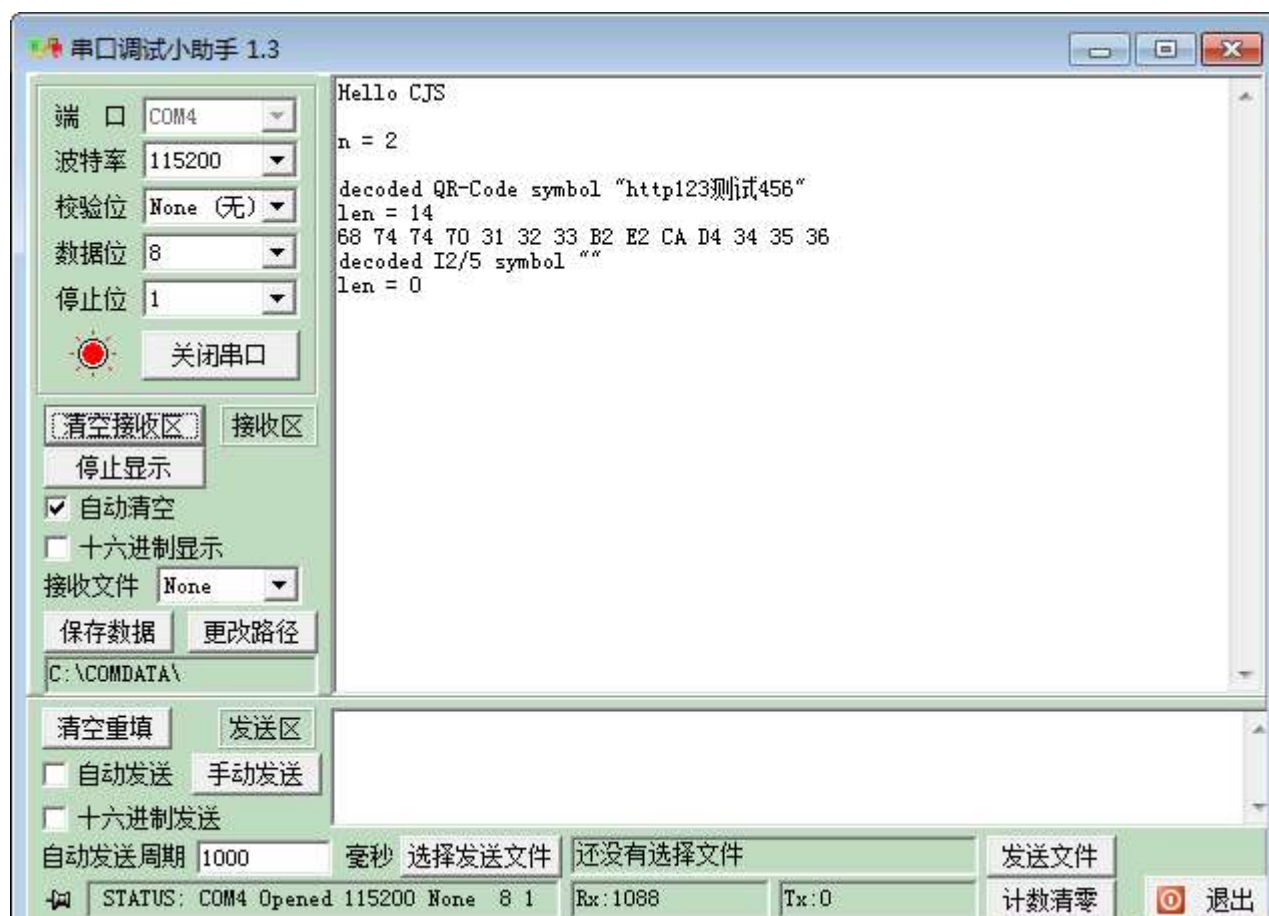
Usart_Init();
printf("Hello CJS\r\n");
Zbar_Test((void* )image_data_buf,280,280);
while(1)
{

}
return 0;
}

```

## 7.运行结果

编译下载到 STM32F429 后，运行结果如下



到此，ZBar 已经能正常在 STM32F429 上运行了，但有点小问题，如上图，会检测到 decoded I2/5 symbol "", 原因未知。

分类: [Zbar](#)