

AN1616B ATK-QR 二维码&条码识别库使用说明

二维码、条形码相信大家耳熟能详了，条码技术已经广泛应用于我们生活当中。微信扫二维码、支付宝二维码付款、商品条形码、超市储物柜条形码等。本应用文档（AN1616B，对应**阿波罗 STM32F429 开发板扩展实验 SE01**）将教大家如何在 ALIENTEK 阿波罗 STM32F4\F7 开发板上使用 AKT-OV5640 摄像头模块完成二维码、一维条形码扫描识别实验。本章分为如下几部分：

- 1， 一维条形码、二维码简介
- 2， 硬件连接
- 3， 软件实现
- 4， 验证

1、一维条形码、二维码简介

1.1 一维条形码

一维条形码是将宽度不等的多个黑条和空白，按照一定的编码规则排列，用以表达一组信息的图形标识符。条形码可以标出物品的生产国、制造厂家、商品名称、生产日期、图书分类号、邮件起止地点、类别、日期等许多信息。常用码制的包括 EAN 码、UPC 码、39 码、交叉 25 码、128 码、93 码及 Codabar 库德巴码等。识别原理：将条形码条符宽度、间隔空间等空间信息转换成二进制码流，然后再编码即可。

1.2 二维码

二维码是用某种特定的几何图形按一定规律在平面（二维方向上）分布的黑白相间的图形记录数据符号信息的。在代码编制上巧妙地利用构成计算机内部逻辑基础的“0”、“1”比特流的概念，使用若干个与二进制相对应的几何形体来表示文字数值信息。常用的码制有：Data Matrix, Maxi Code, Aztec, QR Code, Vericode, PDF417, Ultracode, Code 49, Code 16K 等。二维码的优点：二维码存储的数据量更大；可以包含数字、字符，及中文文本等混合内容且有一定的容错性（在部分损坏以后可以正常读取，比如在二维码中间插入一张小图片）。本实验识别的二维码类型是我们最常用的和见到的 QR 二维码，我们常见的微信二维码、支付宝二维码、商品二维码均是 QR 二维码。图 1.1 为 QR 二维码构造框图。

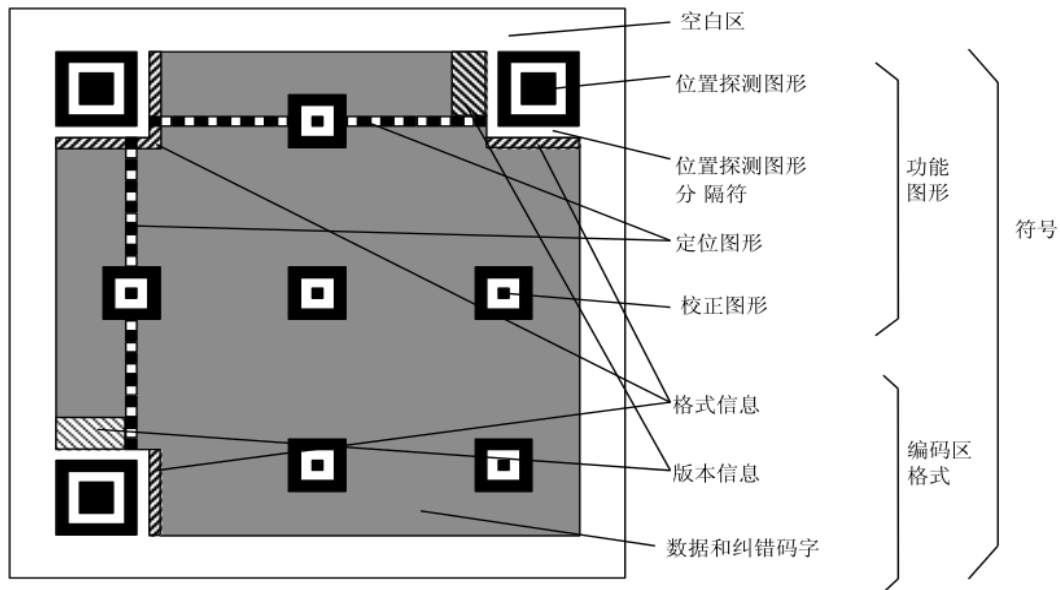


图 1.1 QR 二维码构造框图

1. 位置探测图形、位置探测图形分隔符：用于对二维码的定位，对每个 QR 码来说，位置都是固定存在的，只是大小规格会有所差异；这些黑白间隔的矩形块很容易进行图像处理的检测。

2. 校正图形：根据尺寸的不同，校正图形的个数也不同。校正图形主要用于 QR 码形状的矫正，尤其是当 QR 码印刷在不平坦的面上，或者拍照时候发生畸变等。

3. 定位图形：这些小的黑白相间的格子就好像坐标轴，在二维码上定义了网格。

4. 格式信息：表示该二维码的纠错级别，分为 L、M、Q、H；

5. 数据区域：使用黑白的二进制网格编码内容。8 个格子可以编码一个字节。

6. 版本信息：即二维码的规格，QR 码符号共有 40 种规格的矩阵（一般为黑白色），从 21x21（版本 1），到 177x177（版本 40），每一版本符号比前一版本 每边增加 4 个模块。

7. 纠错码字：用于修正二维码损坏带来的错误。

了解了 QR 二维码的构造，下面我们来了解编码及解码原理，QR 二维码的编码流程如下图 1.2.1 所示。QR 二维码的解码流程如下图 1.2.2 所示。

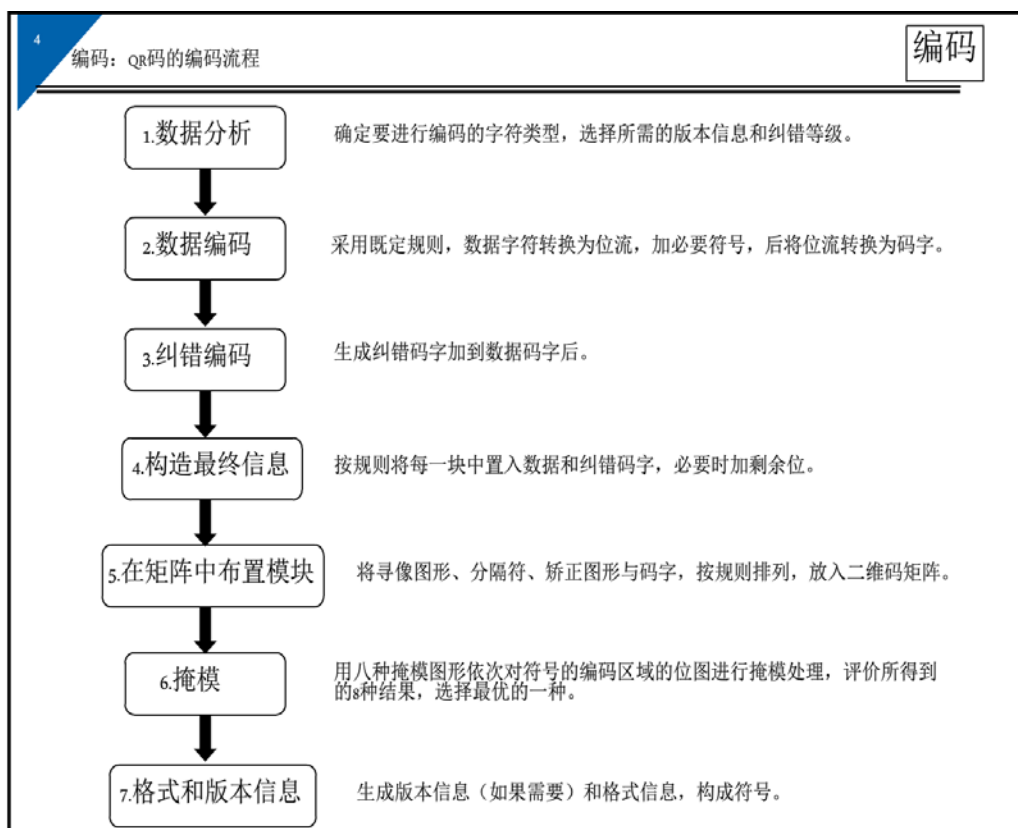


图 1.2.1 QR 二维码编码流程

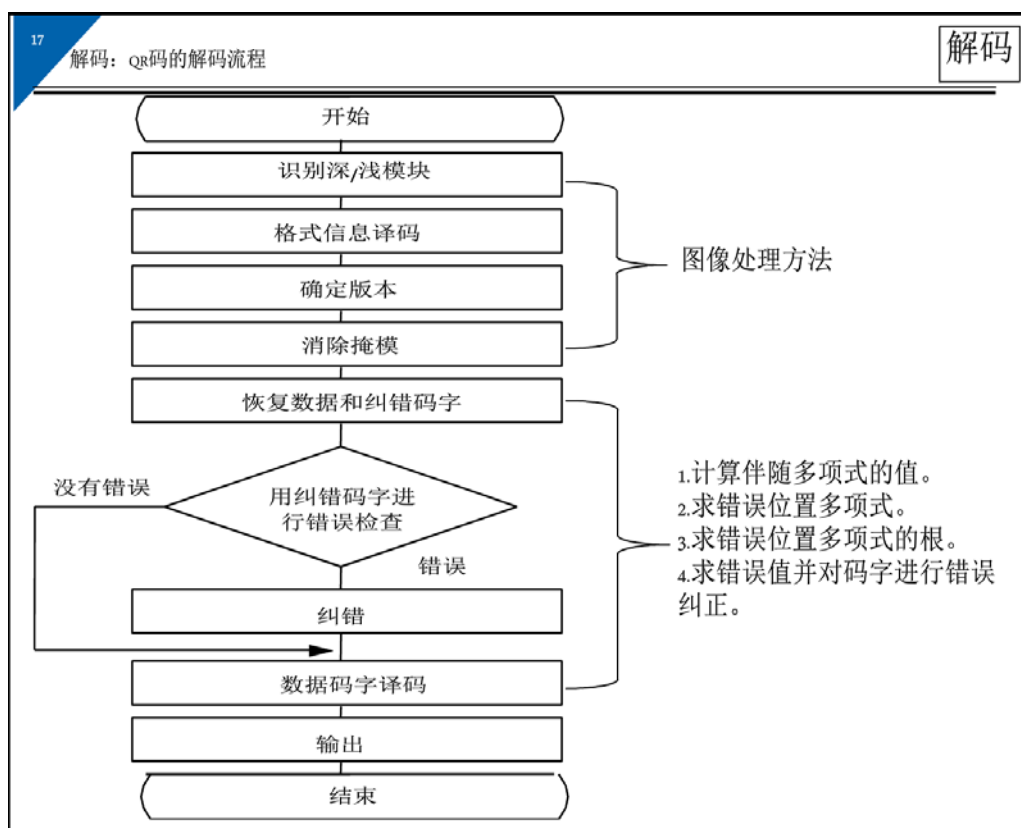


图 1.2.2 QR 二维码解码流程

二维码的编码及解码流程具体实现的步骤和方法是比较复杂,涉及到了 QR 二维码的标准规则及相对应的算法,知识点很多,大家感兴趣的话可以自己多去百度了解。以下两个文档分别是 QR 二维码编码解码标准、二维码编解码及系统实现。

1. 《QRCode-编码解码标准.pdf》,在: ATK-QR 二维码&条形码识别库→3, 二维码识别参考资料 文件夹。
2. 《二维条码的编解码及系统实现.pdf》,在: ATK-QR 二维码&条形码识别库→3, 二维码识别参考资料 文件夹。

以上说了很多关于一维条形码、二维码的知识是为了帮助大家了解关于它们的基础知识,实际上我们并不需要去研究太多,因为市面上已经有成熟的条码编解码库了,比如手机常用的 ZBar 和 ZXing 库。这里,我们 ALIENTEK 提供了一个基于 ZBar 的解码库,并封装成了 lib 方便大家使用(因为是由朋友移植的,他不愿意公开源代码,所以无法提供源码)。

2、硬件连接

本实验使用到的硬件资源有:

- 1, ATK-OV5640 摄像头模块。
- 2, LCD 模块 (RGB 屏、MCU 屏均可)。
- 3, LED (DS0/DS1)。
- 4, 按键 (KEY0\KEY1\KEY2\KEY_UP)。
- 5, IO 扩展芯片 PCF8574。
- 6, 蜂鸣器。

3、软件实现

本实验在阿波罗 STM32F4\F7 开发板的照相机实验基础上进行修改,去除了不需要的 PICTURE 和 USMART,并在工程路径中新建了 ATKQR 文件,里面包含了我们提供的 ATK_QR_V1.4.lib、atk_qrdecode.c 和 atk_qrdecode.h。ATK_QR_V1.4.lib 功能如下:

- 1, 支持 QR 二维码识别。(包括常见的 UTF8 编码格式和特殊的 GBK 编码格式)。
- 2, 支持 EAN 码、39 码、交叉 25 码、UPC 码、128 码等编码方式的条码识别。
- 3, 支持 UTF8-OEM 转换输出(需要客户自己提供转换码表, OEM 一般指 GBK 编码)。
- 4, 内存占用: 6K (算法内存) + bmp_heigh* bmp_width (字节, 8 位灰度图像内存)。

首先我们看一下 atk_qrdecode.h:

```
#define ATK_QR_UTF82OEM_SUPPORT    1
#define ATK_QR_GBK_SUPPORT        1
//返回值定义
#define ATK_QR_OK                    0           //正常
#define ATK_QR_MEM_ERR              1           //内存错误
#define ATK_QR_OTHER_ERR           2           //其他错误
#define ATK_QR_RECO_ERR            3           //无法识别

u8 atk_qr_init(void);                      //初始化识别库
//QR 解码函数
u8 atk_qr_decode(u16 bmp_width,u16 bmp_heigh,u8 *bmp,u8 btype,u8* result);
void atk_qr_destroy(void);                 //结束识别
void atk_qr_memset(void *p,u8 c,u32 len);  //内存设置函数
void *atk_qr_malloc(u32 size);            //动态申请内存
```

```
void *atk_qr_realloc(void *ptr,u32 size);           //内存重申请函数
void atk_qr_free(void *ptr);                       //动态释放内存
void atk_qr_memcpy(void *des,void *src,u32 n);     //内存复制函数
u16 atk_qr_convert(u16 unicode);                   //unicode 转 oem
```

ATK_QR_UTF82OEM_SUPPORT: 定义是否支持将 UTF8 编码转换为 OEM 编码(中文一般是指 GBK 编码), 定义为: 1 则输出结果为 GBK 编码方式的字符串; 定义为: 0 则输出的是原始字符串(未做编码转换, 可能是 UTF8, 也可能是 GBK 取决于二维码的编码方式)。

ATK_QR_GBK_SUPPORT: 定义在 **ATK_QR_UTF82OEM_SUPPORT==1** 的时候, 是否支持识别 GBK 编码的二维码, 如果定义为 1, 则程序会先判断是 UTF8 还是 GBK, 根据识别结果再做转换。如果定义为 0 则只识别 UTF8 编码方式的二维码。不识别 GBK 编码的二维码。

注意:

- 1, 当 **ATK_QR_UTF82OEM_SUPPORT==0** 的时候, 该宏定义不起作用。
- 2, 当发现有识别错误(UTF8 识别成 GBK 了)的时候, 可设置 **ATK_QR_GBK_SUPPORT==0**, 以更好的支持 UTF8。常见的 QR 二维码格式大多数为 UTF8 格式。
- 3, 这里的 GBK 并不是所有 GBK 都支持, 仅支持 GB2312 编码的识别。

接下来, 我们重点介绍一下: `u8 atk_qr_decode(u16 bmp_width,u16 bmp_height,u8 *bmp,u8 btype,u8* result)` 这个函数, 它的参数说明:

img_width, img_height: 输入图像的宽度和高度。

imgbuf: 图像缓存区(8 位灰度图像, 不是 RGB565!!!!)

btype: 0, 识别二维码。

- 1, 识别 CODE128 条码。
- 2, 识别 CODE39 条码。
- 3, 识别 I25 条码。
- 4, 识别 EAN13 条码。

result: 识别结果缓冲区。如果 `result[0]==0`, 则说明未识别到任何数据, 否则就是识别到的数据(字符串)。

返回值: **ATK_QR_OK**, 识别完成, 返回其他是相对应错误代码(以上返回值定义)。

提示: 如果需要对所有支持的编码进行识别, 则轮流设置 **btype** 为 0~4 即可实现。

本 LIB 移植步骤:

- 1, 实现 `atk_qrdecode.c` 里面的所有函数。
- 2, 堆栈(Stack_Size)设置为 0X1000 或以上, 在 `startup_stm32f40_41xx.s` 中设置。

本 LIB 使用步骤:

- 1, 调用 `atk_qr_init` 函数, 初始化识别程序, 返回值为 **ATK_QR_OK**, 则初始化成功。
- 2, 调用 `atk_qr_decode` 函数, 给定参数, 对图像进行识别。
- 3, 如果需要不停的识别, 则重复第 2 个步骤即可。
- 5, 调用 `atk_qr_destroy` 函数, 结束识别, 释放所有内存结束识别。

ATK_QR_V1.4.lib 移植及使用步骤在 `atk_qrdecode.c` 注释中都有说明, 具体过程, 见 `main.c` 中的 `qr_decode()` 函数。下面来看一下 `qr_decode()` 这个函数:

```
//imagewidth=imageheight: <=240;大于 240 时,是 240 的整数倍
//imagebuf:RGB 图像数据缓冲区
void qr_decode(u16 imagewidth,u16 *imagebuf)
{
```

```

static u8 bartype=0;
u8 *bmp;
u8 *result=NULL;
u16 Color;
u16 i,j;
u16 qr_img_width=0;           //输入识别器的图像宽度,最大不超过 240!
u8 qr_img_scale=0;           //压缩比例因子
if(imagewidth>240)
{
    if(imagewidth%240)return ; //不是 240 的倍数,直接退出
    qr_img_width=240;
    qr_img_scale=imagewidth/qr_img_width;
}
else
{
    qr_img_width=imagewidth;
    qr_img_scale=1;
}
result=mymalloc(SRAMIN,1536); //申请识别结果存放内存
//CCM 管理内存为 60K, 这里最大可申请 240*240=56K
bmp=mymalloc(SRAMCCM,qr_img_width*qr_img_width);
memset(bmp,0,qr_img_width*qr_img_width);
if(lcdltdc.pwidth==0) //MCU 屏,无需镜像
{
    for(i=0;i<qr_img_width;i++)
    {
        for(j=0;j<qr_img_width;j++) //将 RGB565 图片转成灰度
        {
            //按照 qr_img_scale 压缩成 240*240
            Color=(imagebuf+((i*imagewidth)+j)*qr_img_scale);
            *(bmp+i*qr_img_width+j)=(((Color&0xF800)>>8)*76+
            ((Color&0x7E0)>>3)*150+((Color&0x001F)<<3)*30)>>8;
        }
    }
}
else //RGB 屏,需要镜像
{
    for(i=0;i<qr_img_width;i++)
    {
        for(j=0;j<qr_img_width;j++) //将 RGB565 图片转成灰度
        {
            //按照 qr_img_scale 压缩成 240*240
            Color=(imagebuf+((i*imagewidth)+qr_img_width-j-1)*qr_img_scale);
            *(bmp+i*qr_img_width+j)=(((Color&0xF800)>>8)*76
            + ((Color&0x7E0)>>3)*150+((Color&0x001F)<<3)*30)>>8;
        }
    }
}

```

```

    }
}
//识别灰度图片（注意：单次耗时约 0.2S）
atk_qr_decode(qr_img_width,qr_img_width,bmp,bartype,result);
if(result[0]==0)    //没有识别出来
{
    bartype++;
    if(bartype>=5)bartype=0;
}
else if(result[0]!=0) //识别出来了，显示结果
{
    PCF8574_WriteBit(BEEP_IO,0); //打开蜂鸣器
    delay_ms(100);
    PCF8574_WriteBit(BEEP_IO,1);
    POINT_COLOR=BLUE;
    LCD_Fill(0,(lcddev.height+qr_image_width)/2+20,lcddev.width,lcddev.height,BL
        ACK);
    Show_Str( 0,(lcddev.height+qr_image_width)/2+20,lcddev.width,
        (lcddev.height-qr_image_width)/2-20,(u8*)result,16,0
    ); //LCD 显示识别结果
    printf("\r\nresult:\r\n%s\r\n",result); //串口打印识别结果
}
myfree(SRAMCCM,bmp); //释放灰度图 bmp 内存
myfree(SRAMIN,result); //释放识别结果
}

```

函数参数 imagewidth 是图像尺寸，*imagebuf 为 RGB565 图像数据。图像 RGB565 数据需转换成 8 位的灰度数据传入 atk_qr_decode() 才能被识别。我们定义了图像尺寸 imagewidth 高度等于宽度且必须是 240 的倍数，这是因为我们将 8 位灰度图像申请内存放在 CCM 里使得算法运算速度更快一些，且 CCM 只管理 60K 内存（在 malloc.h 中），所以最终将输入识别的图像压缩成 240*240=56K 的灰度图像。大家也可以将内存申请只外部 SDRAM 这样就可以随意定义 imagewidth 的大小了，只是图像的处理速度会慢一些。在函数里定义 static u8 bartype=0，使用 static 定义相当于静态全局变量，不会每次进入函数时 bartype=0；当没识别到数据时 bartype++，这样就可以实现二维码、CODE128、CODE39、I25、EAN13 轮流识别了。当 result[0]!=0 时即识别到了，蜂鸣器“滴”一声提示，并将识别结果显示在 LCD 上及发送给串口打印。

下面我们来看一下 main.c 中的 qr_dcmi_rx_callback 函数和 main 函数：

```

u16 qr_image_width;           //输入识别图像的宽度（长度=宽度）
u8  readok=0;                 //采集完一帧数据标识
u32 *dcmi_line_buf[2];        //摄像头采用一行一行读取,定义行缓存
u16 *rgb_data_buf;             //RGB565 帧缓存 buf
u16 dcmi_curline=0;            //摄像头输出数据,当前行编号
//摄像头数据 DMA 接收完成中断回调函数
void qr_dcmi_rx_callback(void)
{

```



```
u32 *pbuf;
u16 i;
//将 rgb_data_buf 地址偏移赋值给 pbuf
pbuf=(u32*)(rgb_data_buf+dcmi_curline*qr_image_width);
if(DMA2_Stream1->CR&(1<<19)) //DMA 使用 buf1,读取 buf0
{
    for(i=0;i<qr_image_width/2;i++)
    {
        pbuf[i]=dcmi_line_buf[0][i];
    }
}else //DMA 使用 buf0,读取 buf1
{
    for(i=0;i<qr_image_width/2;i++)
    {
        pbuf[i]=dcmi_line_buf[1][i];
    }
}
dcmi_curline++;
}
int main(void)
{
    float fac;
    u8 key;
    u8 i;

    Stm32_Clock_Init(360,25,2,8); //设置时钟,180Mhz
    delay_init(180); //初始化延时函数
    uart_init(90,115200); //初始化串口波特率为 115200
    LED_Init(); //初始化与 LED 连接的硬件接口
    SDRAM_Init(); //初始化 SDRAM
    LCD_Init(); //初始化 LCD
    KEY_Init(); //初始化按键
    PCF8574_Init(); //初始化 PCF8574
    OV5640_Init(); //初始化 OV5640
    W25QXX_Init(); //初始化 W25Q256
    my_mem_init(SRAMIN); //初始化内部内存池
    my_mem_init(SRAMEX); //初始化外部内存池
    my_mem_init(SRAMCCM); //初始化 CCM 内存池
    POINT_COLOR=RED;
    LCD_Clear(BLACK);
    while(font_init()) //检查字库
    {
        LCD_ShowString(30,50,200,16,16,(u8*)"Font Error!");
        delay_ms(200);
    }
}
```



```
LCD_Fill(30,50,240,66,WHITE);//清除显示
delay_ms(200);
}
Show_Str_Mid(0,0,(u8*)"阿波罗 STM32F4/F7 开发板",16,lcddev.width);
Show_Str_Mid(0,20,(u8*)"二维码/条形码识别实验",16,lcddev.width);
while(OV5640_Init())//初始化 OV5640
{
    Show_Str(30,190,240,16,(u8*)"OV5640 错误!",16,0);
    delay_ms(200);
    LCD_Fill(30,190,239,206,WHITE);
    delay_ms(200);
}
//自动对焦初始化
OV5640_RGB565_Mode();    //RGB565 模式
OV5640_Focus_Init();
OV5640_Light_Mode(0);    //自动模式
OV5640_Color_Saturation(3);    //色彩饱和度 0
OV5640_Brightness(4);    //亮度 0
OV5640_Contrast(3);    //对比度 0
OV5640_Sharpness(33);    //自动锐度
OV5640_Focus_Constant();    //启动持续对焦
DCMI_Init();    //DCMI 配置
qr_image_width=lcddev.width;
//这里 qr_image_width 设置为 240 的倍数
if(qr_image_width>480)qr_image_width=480;
if(qr_image_width==320)qr_image_width=240;
Show_Str(0,(lcddev.height+qr_image_width)/2+4,240,16,(u8*)"识别结果: ",16,1);
//为行缓存接收申请内存
dcmi_line_buf[0]=mymalloc(SRAMIN,qr_image_width*2);
//为行缓存接收申请内存
dcmi_line_buf[1]=mymalloc(SRAMIN,qr_image_width*2);
//为 rgb 帧缓存申请内存
rgb_data_buf=mymalloc(SRAMEX,qr_image_width*qr_image_width*2);
dcmi_rx_callback=qr_dcmi_rx_callback;    //DMA 数据接收中断回调函数
DCMI_DMA_Init((u32)dcmi_line_buf[0],(u32)dcmi_line_buf[1],qr_image_width/2,1,1
);    //DCMI DMA 配置
fac=800/qr_image_width;    //得到比例因子
OV5640_OutSize_Set((1280-fac*qr_image_width)/2,(800-fac*qr_image_width)/2,qr_i
mage_width,qr_image_width);
DCMI_Start();    //启动传输
printf("SRAM IN:%d\r\n",my_mem_perused(SRAMIN));
printf("SRAM EX:%d\r\n",my_mem_perused(SRAMEX));
printf("SRAM CCM:%d\r\n",my_mem_perused(SRAMCCM));
```

```

atk_qr_init();//为算法申请内存

printf("1SRAM IN:%d\r\n",my_mem_perused(SRAMIN));
printf("1SRAM EX:%d\r\n",my_mem_perused(SRAMEX));
printf("1SRAM CCM:%d\r\n",my_mem_perused(SRAMCCM));
while(1)
{
    key=KEY_Scan(0);//不支持连接
    if(key)
    {
        //按 KEY0、KEY1、KEYUP 手动单次自动对焦
        OV5640_Focus_Single();
        if(key==KEY2_PRES)break; //按 KEY2 结束识别
    }
    if(readok==1) //采集到了一帧图像
    {
        readok=0;
        qr_show_image((lcddev.width-qr_image_width)/2,(lcddev.height-qr_image_width)/2,qr_image_width,qr_image_width,rgb_data_buf);
        qr_decode(qr_image_width,rgb_data_buf);//识别图像
    }
    i++;
    if(i==20)//DS0 闪烁.
    {
        i=0;
        LED0=!LED0;
    }
}
atk_qr_destroy();//释放算法内存
printf("3SRAM IN:%d\r\n",my_mem_perused(SRAMIN));
printf("3SRAM EX:%d\r\n",my_mem_perused(SRAMEX));
printf("3SRAM CCM:%d\r\n",my_mem_perused(SRAMCCM));
while(1)
{
    LED0=!LED0;
    delay_ms(200);
}
}

```

qr_dcml_rx_callback 函数是 DMA 中断回调函数,当 DMA 传输完成一行图像数据则中断,并调用此函数。函数里先判断当前 DMA 传输使用的 buf0 还是 buf1, 如果是 buf0 说明 buf1 已满则读取 buf1 的数据, 并复制给 rgb_data_buf。

Main 函数首先是初始化时钟、外设、硬件等, 然后初始化 DCMI, 配置 DMA 双缓冲模式来接收 DCMI 采集的图像数据。这里我们配置 OV5640 为自动连续对焦模式, 并根据 lcddev.width (由 lcd 分辨率确定) 来设置 OV5640 输出图像尺寸为 240*240 或 480*480 (根

据屏幕分辨率，自动设置)。因为 DMA 双缓冲模式需要两个 buf (缓冲区) 轮流接收数据，所以我们申请了 dcmi_line_buf[0]、[1] 内存来接收图像的行数据。另外我们还申请了 rgb_data_buf 内存，用于存放一帧 RGB565 图像数据。while 循环里主要是按键扫描 KEY_Scan(); 和判断是否采集到了一帧图像 if(readok==1)，如果采集到了一帧图像则先显示了图像 qr_show_image() 然后调用 qr_decode() 函数进行识别。注意：readok=1 的设置，是在 dcmi.c 的 DCMI_IRQHandler 函数里面设置的。

4、验证

在代码编译成功之后，我们下载代码到 ALIENTEK 阿波罗 STM32F4/F7 开发板上。然后将二维码图像、一维条码图像放在摄像头前面（提示：可以网上百度二维码、一维码生成器生成各种格式的图像），接着等待 OV5640 自动对焦或者手动按（KEY0\KEY1\KEY_UP）任一按键执行一次对焦直至图像清晰。如果识别到了，蜂鸣器会“滴滴”提示，（只要不是很复杂的二维码可以很快被识别到）并在“识别结果：”下方显示。识别二维码结果如图 4.1 所示。识别一维条码如图 4.2 所示。识别二维码串口打印结果如图 4.3 所示。（提示：本实验使用 4.3 寸 LCD 可能显示位置不足，可在串口助手 XCOM 查看全部结果）。



图 4.1 识别二维码



图 4.2 识别一维条码

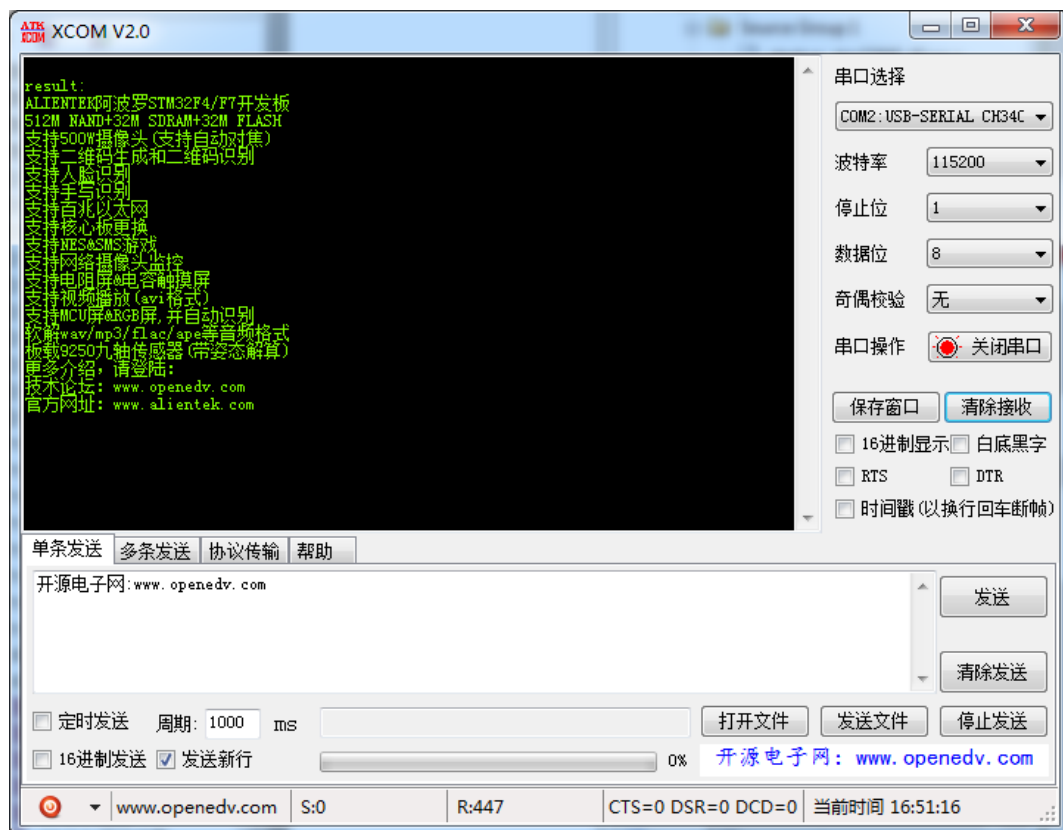


图 4.3 识别二维码输出结果

正点原子@ALIENTEK

公司网址: www.alientek.com

技术论坛: www.openedv.com

电话: 020-38271790

传真: 020-36773971

