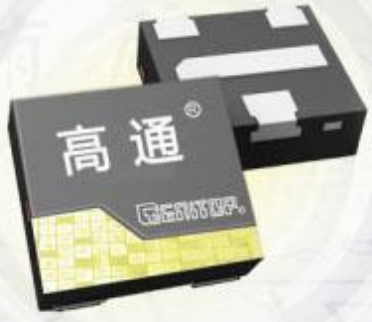
高通®字库
GENITOP®

高通字库芯片详细使用指南

V1.2

2025.6.3

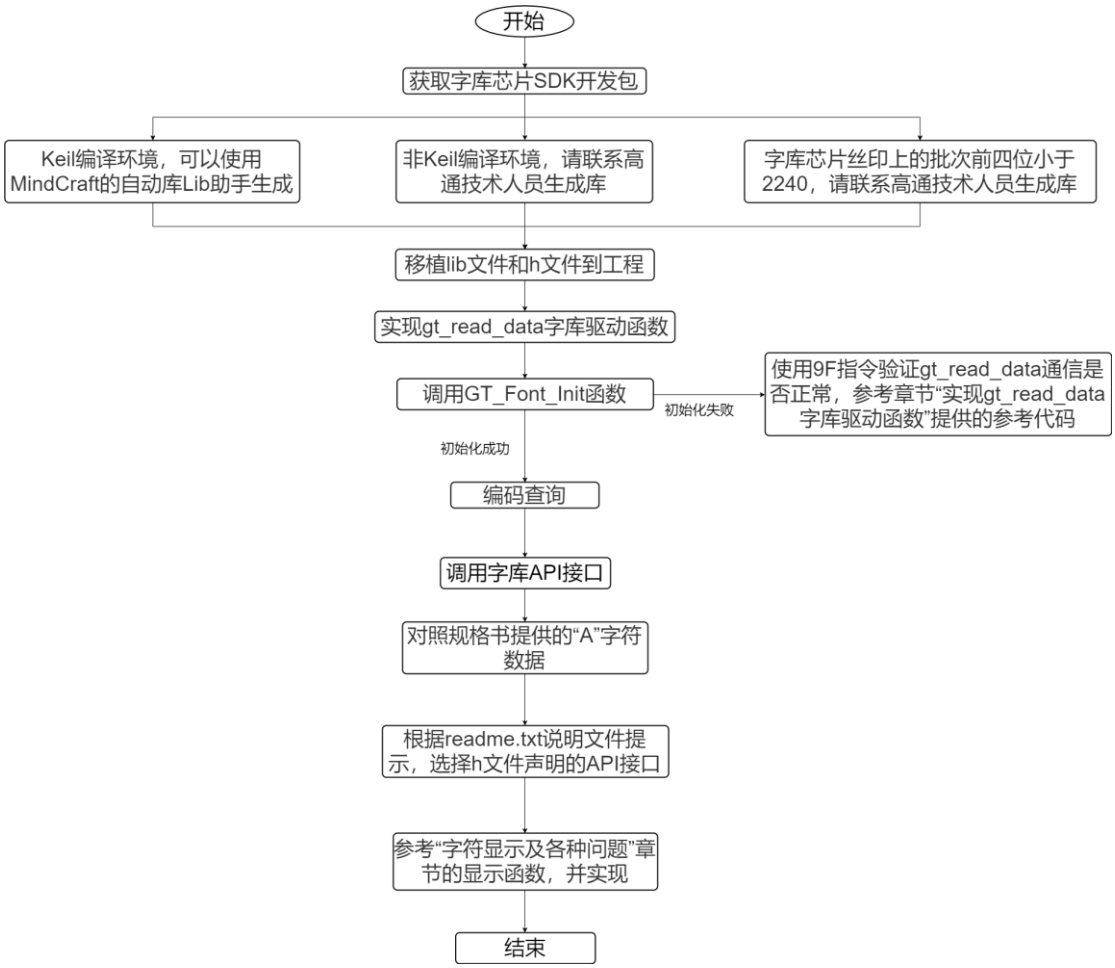


深圳高通半导体有限公司

版本修订记录

版本号	修改内容	日期	备注
V1.0	高通字库芯片详细使用指南制定	2025/5/9	
V1.1	1.附上字库使用流程图	2025/5/21	
	2.更新目录顺序，初始化字库后，先进行编码查询，最后再实现显示函数	2025/5/21	
	3.点阵 API 接口章节中加入了如何对比字符“A”的参考数据	2025/5/21	
	4.点阵 API 接口章节中加入中文，拉丁文编码查询方式	2025/5/21	
V1.2	5. 修改 3.1 章各个国家的编码查询表链接	2025/6/3	

字库驱动流程图


开始
获取字库芯片SDK开发包
Keil编译环境，可以使用MindCraft的自动库Lib助手生成
非Keil编译环境，请联系高通技术人员生成库
字库芯片丝印上的批次前四位小于2240，请联系高通技术人员生成库
移植lib文件和h文件到工程
实现gt_read_data字库驱动函数
调用GT_Font_Init函数
初始化失败
使用9F指令验证gt_read_data通信是否正常，参考章节“实现gt_read_data字库驱动函数”提供的参考代码
初始化成功
编码查询
调用字库API接口
对照规格书提供的“A”字符数据
根据readme.txt说明文件提示，选择h文件声明的API接口
参考“字符显示及各种问题”章节的显示函数，并实现
结束

深圳高通半导体有限公司

目录

1. 获取字库芯片 SDK 开发包

1.1. 字库芯片 SDK 开发包生成步骤

用户可以通过 https://www.hmi.gaotongfont.cn/gtzkxpkfzl 链接下载 MindCraft（高通智匠）。注意：如果用户使用的编译器不是 KEIL，或者芯片丝印上的批次前四位小于 2240 则无法使用自动库 Lib 助手生成的 SDK 库文件。请拨打 0755-83453881 联系业务或者技术人员，手动生成对应字库的 API 函数库。生成步骤如下：

步骤一：点击 MindCraft 菜单栏的“应用”选项，然后选择“自动库 Lib 助手”。

步骤二：按照图 1-2 中标注的顺序，逐个填写每个选项完成配置。注意选择编译器路径可以查看编译器属性中的目标，如图 1-1 所示：

图 1-1

将目标中的路径复制粘贴到图 1-2 中的“本地编译器路径”选项中。

图 1-2

步骤三：点击生成，会看到图 1-3 中的 5 个文件夹。

图 1-3

① 01_参考资料：里面包含多个国家的文字 Unicode 编码范围，技术支持的相关视频以及如何在 STM32 上使用高通字库的文档。

② 02_硬件资料：包含字库芯片的规格书，选型表。

③ 03_SPI 驱动：硬件 SPI，模拟 SPI 参考代码。

④ 04_字库 API 使用：提供字库的 lib 文件、h 文件、字库 readme 文件。

⑤ 05_屏幕显示资料：提供各种功能的代码。例如：横竖转换，获取字符实际宽度等。

1.2. 使用自动库 Lib 助手注意事项

获取字库芯片 SDK 开发包注意事项：

① 编译器路径不要带中文，最后选择的目标可以通过 keil 编译器属性查看目标。如下图 1-4 所示。

② 库文件生成路径不要带中文。

③ 工程参数一键导入不用勾选，默认为空即可。

④ 编译器内核版本根据开发者的工程设置选择。

⑤ 优化等级根据开发者的工程设置选择。

⑥ 开发者只需勾选“MicroLib”、“uC99”、“OneELF Section per Function”三个选项即可。如下图 1-5 红框所示。


keil5.exe - 快捷方式 属性
常规
快捷方式
兼容性
安全
详细信息
以前的版本
keil5.exe - 快捷方式
目标类型:
应用程序
目标位置:
UV4
此路径不能含有中文
目标(T):
C:\Keil_v5\UV4\UV4.exe
起始位置(S):
C:\Keil_v5\UV4
快捷键(K):
无
运行方式(R):
常规窗口
备注(O):
打开文件所在的位置(F)
更改图标(C)...
高级(D)...
确定
取消
应用(A)

图 1-4


ARM处理器参数:
编译器内核版本:
AC5
6、选择编译器的内核版本
MCU内核类型:
Cortex-M4
7、选择MCU的内核
优化等级:
Level 0(-O0)
8、根据需求选择优化等级
MicroLib
Strict ANSI C
Enum Container always int
uC99
uGnu
OneELF Section per Function
9、勾选这三个选项即可

图 1-5

2. 字库驱动

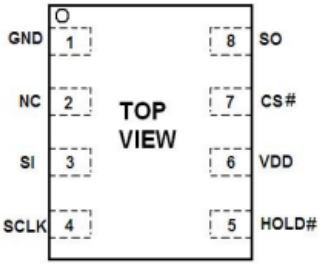
2. 1. 引脚描述及封装

打开“02_硬件资料”文件夹下的产品规格书，里面有可以查看到硬件的具体信息。

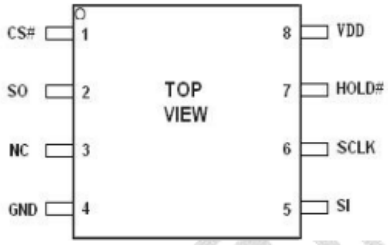
3 引脚描述与电路连接

3.1 引脚配置

DFN8 2X3



SOP8-B



3.2 引脚描述

DFN8 2X3

NO.	名称	I/O	描述
1	GND		地(Ground)
2	NC		悬空
3	SI	I	串行数据输入（Serial data input）
4	SCLK	I	串行时钟输入（Serial clock input）
5	HOLD#	I	总线挂起（Hold, to pause the device without）
6	VDD		电源(+ 3.3V Power Supply)
7	CS#	I	片选输入（Chip enable input）
8	SO	O	串行数据输出（Serial data output）

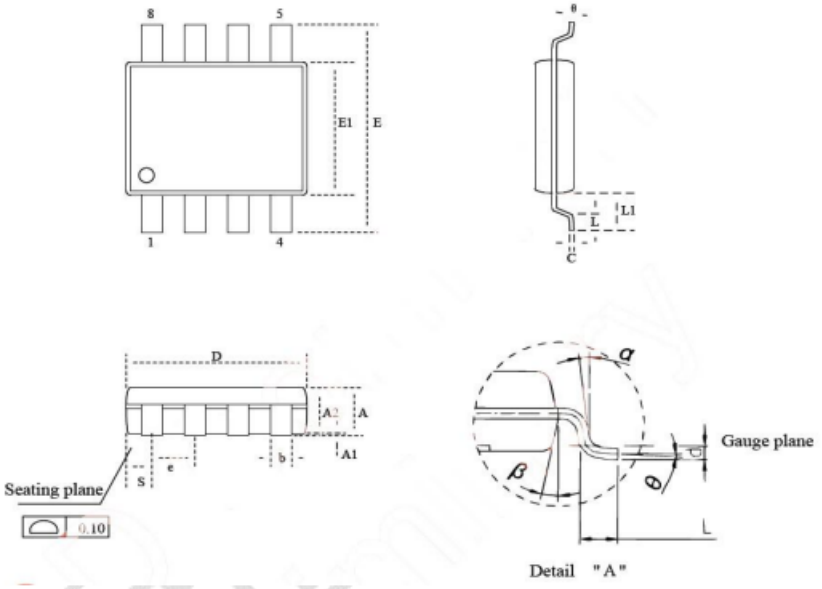
SOP8-B

NO.	名称	I/O	描述
1	CS#	I	片选输入（Chip enable input）
2	SO	O	串行数据输出（Serial data output）
3	NC		悬空
4	GND		地(Ground)
5	SI	I	串行数据输入（Serial data input）
6	SCLK	I	串行时钟输入（Serial clock input）
7	HOLD#	I	总线挂起（Hold, to pause the device without）
8	VDD		电源(+ 3.3V Power Supply)

6 封装尺寸

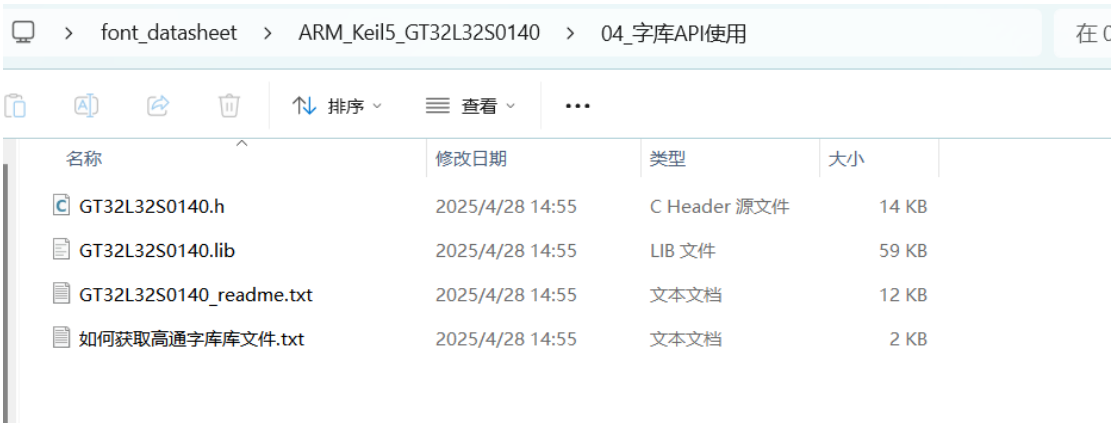
封 装 类 型	封装尺寸
SOP8-A	4.90mmX3.90mm（193milX154mil）

SOP8-A



		A	A1	A2	b	C	D	E	E1	e	L	L1	S	θ
Mm	Min.	-	0.10	1.35	0.36	0.15	4.77	5.80	3.60		0.46	0.65	0.41	0
	Norm.	-	0.15	1.45	0.41	0.20	4.90	5.99	3.90	1.27	0.66	1.05	0.54	5
	Max.	1.75	0.20	1.55	0.51	0.25	5.03	6.20	4.00		0.86	1.25	0.67	8
inch	Min.	-	0.004	0.053	0.014	0.006	0.188	0.228	0.150		0.018	0.033	0.016	0
	Norm.	-	0.006	0.057	0.016	0.008	0.193	0.236	0.154	0.050	0.026	0.041	0.021	5
	Max.	0.069	0.008	0.061	0.020	0.010	0.198	0.244	0.156		0.034	0.049	0.026	8

2.2. 添加 lib 文件和 h 文件到工程



打开文件夹“04_字库 API 使用”，将 lib 文件和 h 文件添加到工程中即可。

2.3. 实现 gt_read_data 字库驱动函数

实现 gt_read_data 之前，要保证 SPI 正常初始化。SPI 初始化例程可以在 MCU 芯片原厂的官网获取。下面的代码是 STM32F407 的 SPI 初始化代码，以供参考。

```
void SPI1_Init(void)
{
    GPIO_InitTypeDef  GPIO_InitStructure;
    SPI_InitTypeDef  SPI_InitStructure;
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);//使能 GPIOB 时钟
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1, ENABLE);//使能 SPI1 时钟
    //GPIOB 3,4,5 初始化设置
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3|GPIO_Pin_4|GPIO_Pin_5;//PB3~5 复用功能输出
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;//复用功能
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;//推挽输出
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;//100MHz
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;//上拉
    GPIO_Init(GPIOB, &GPIO_InitStructure);//初始化
    GPIO_PinAFConfig(GPIOB,GPIO_PinSource3,GPIO_AF_SPI1); //PB3 复用为 SPI1
    GPIO_PinAFConfig(GPIOB,GPIO_PinSource4,GPIO_AF_SPI1); //PB4 复用为 SPI1
    GPIO_PinAFConfig(GPIOB,GPIO_PinSource5,GPIO_AF_SPI1); //PB5 复用为 SPI1
    //这里只针对 SPI 口初始化
    RCC_APB2PeriphResetCmd(RCC_APB2Periph_SPI1,ENABLE);//复位 SPI1
    RCC_APB2PeriphResetCmd(RCC_APB2Periph_SPI1,DISABLE);//停止复位 SPI1
    SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;  //设置 SPI 单向或者双向的
数据模式:SPI 设置为双线双向全双工
    SPI_InitStructure.SPI_Mode = SPI_Mode_Master;       //设置 SPI 工作模式:设置为主 SPI
```

```
    SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;       //设置 SPI 的数据大小:SPI 发送接收 8 位帧结构
    SPI_InitStructure.SPI_CPOL = SPI_CPOL_High;     //串行同步时钟的空闲状态为高电平
    SPI_InitStructure.SPI_CPHA = SPI_CPHA_2Edge;    //串行同步时钟的第二个跳变沿（上升或下降）数据被采样
    SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;       //NSS 信号由硬件（NSS 管脚）还是软件（使用 SSI 位）管理:内部 NSS 信号有 SSI 位控制
    SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_2;      //定义波特率预分频的值:波特率预分频值为 256
    SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;  //指定数据传输从 MSB 位还是 LSB 位开始:数据传输从 MSB 位开始
    SPI_InitStructure.SPI_CRCPolynomial = 7;    //CRC 值计算的多项式
    SPI_Init(SPI1, &SPI_InitStructure);  //根据 SPI_InitStruct 中指定的参数初始化外设 SPIx 寄存器
    SPI_Cmd(SPI1, ENABLE); //使能 SPI 外设
    SPI1_ReadWriteByte(0xff);//启动传输
}
```

然后再实现 gt_read_data 函数，该函数的实现可以参考文件夹“03_SPI 驱动\gt_read_data.c”文件。将代码移植到工程中，并替换代码中的 CS 上拉和下拉以及 SPI 收发的代码，使用用户的代码。

```
/**
 * @brief 发送读取函数
 *
 * @param sendbuf 发送数据的buff
 * @param sendlen 发送数据长度
 * @param receivebuf 读取数据的buff
 * @param receivelen 读取数据长度
 */
unsigned char gt_read_data(unsigned char * sendbuf, unsigned char sendlen, unsigned char * receivebuf, unsigned int receivelen)
{
    unsigned int i;
    Rom_csL;   //拉低片选cs，选中flash      <-- 替换成用户的CS拉低
    for(i = 0; i < sendlen; i++)
    {
        SPI1_ReadWriteByte(sendbuf[i]); //发送数据      <-- 替换成用户的SPI读写函数，发送数据
    }
    for(i = 0; i < receivelen; i++)
    {
        receivebuf[i] = SPI1_ReadWriteByte(0x00);   //接收数据，保存到receivebuf[]      <-- 替换成用户的SPI读写函数，接发数据
    }
    Rom_csH;   //拉高片选cs，空闲      <-- 替换成用户的CS拉高
    return 1;
}
```

实现 gt_read_data 函数后，可以通过发 9F 指令测试 gt_read_data 通信是否正常。具体测试代码如下：

```
uint8_t tmp_buf[8] = {0};
tmp_buf[0] = 0x9F;
gt_read_data(tmp_buf, 1, tmp_buf, 8);
for(uint8_t i = 0; i < 8; i++)
{
printf("tmp_buf:%x\r\n", tmp_buf[i]);
}
```

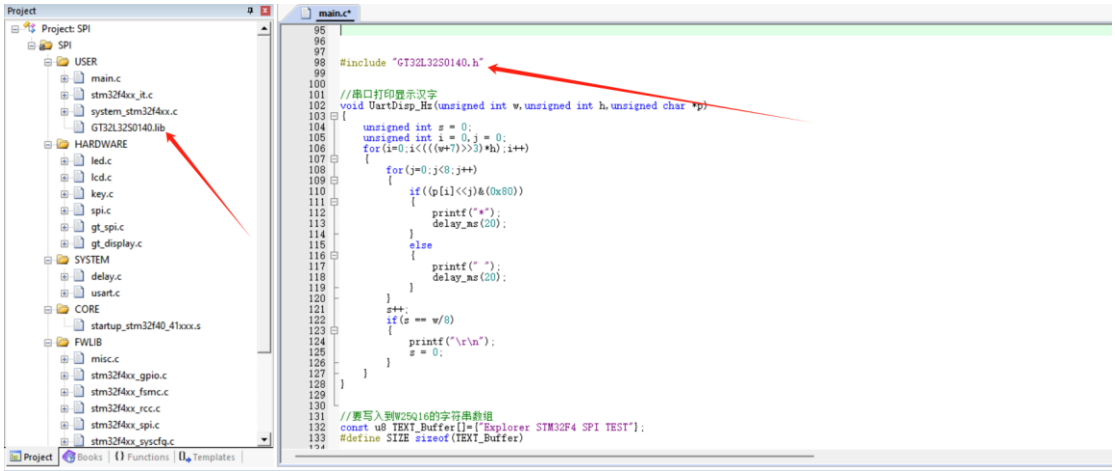
检查 tmp_buf 最后接收到的数据能否在图 2-1 的参考的数据中找到，如果找不到可以通过拨打 0755-83453881 联系业务或者技术人员。

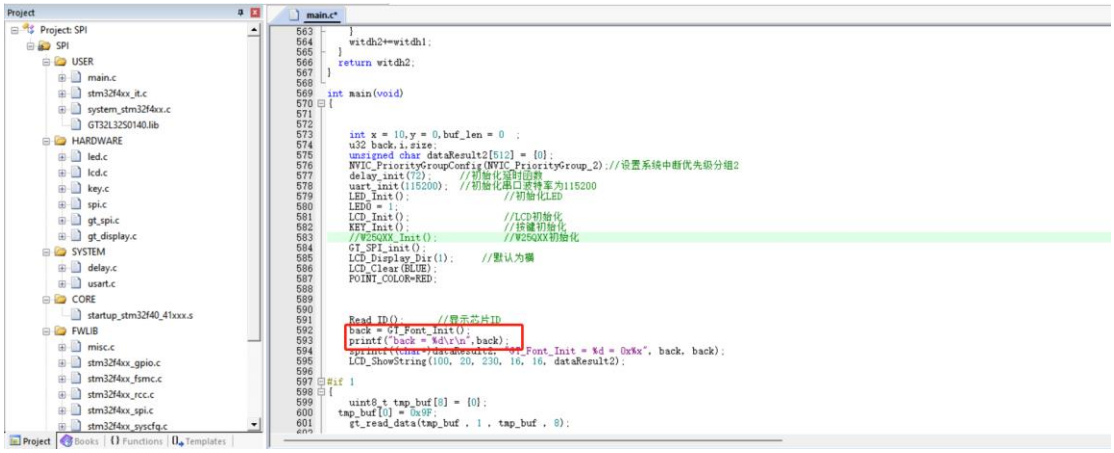
0x856013	0xC84013
0x856014	0xC84014
0x856015	0xC84015
0x856016	0xC84016
0x856017	0xC84017
0x856018	0xC84018
0x852018	
	0x684013
0x5E3213	0x684014
0x5E3214	0x684015
0x5E4015	0x684016
0x5E6015	0x684017
0x5E4016	0x684018
0x5E4017	
0x5E4018	0x204018
0x0B4013	0xEAD8C6
0x0B4014	0xEAD8C7
0x0B4015	0xEAD8C8
0x0B4016	0xEAD8C9
0x0B4017	0xEAD8CA
0x0B4018	0xEAD8CB

图 2-1

2.4. 调用 GT_Font_Init 函数

如果 gt_read_data 通信正常，那么可以开始添加 lib 文件以及 h 文件到工程中，并进行头文件的声明，然后调用 GT_Font_Init 进行字库初始化。注意：必须在 SPI 初始化后调用 GT_Font_Init 函数。

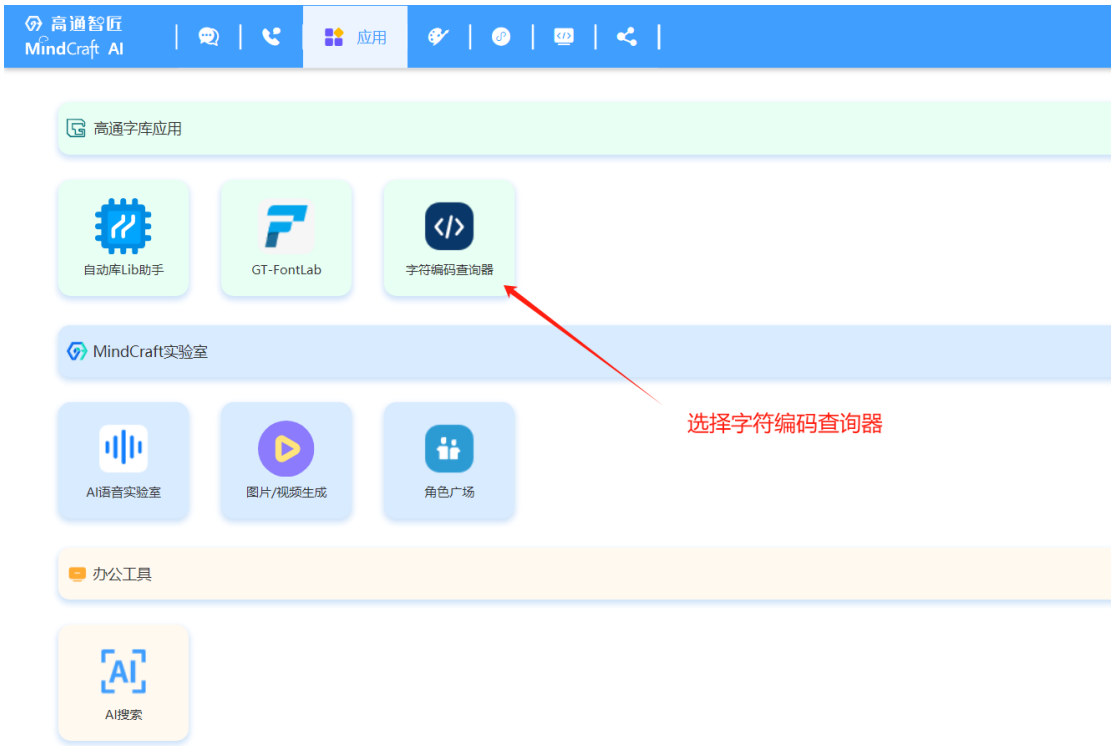




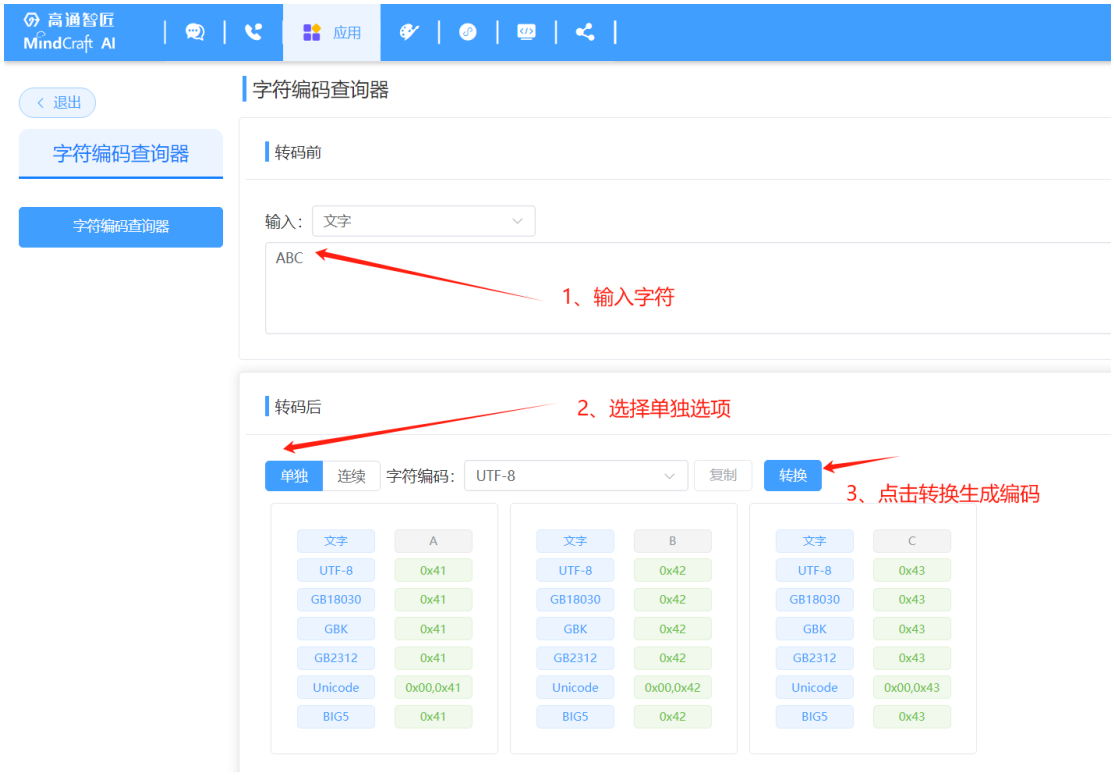
如果 GT_Font_Init 返回值大于 0 表示字库初始化成功，可以正常调用字库的 API 接口函数；如果返回值小于或等于 0，说明主控与字库芯片的通信仍存在问题。可以检查 SPI 的引脚初始化是否都上拉了；检查 SPI 的时钟频率是否过高，建议 SPI 时钟频率设置 45MHz 或以下。如果上述的操作都执行了，字库初始化还是返回小于或等于 0 的值，请到官网联系高通字库技术人员。联系方式 0755-83453881。

2.5. 编码查询

MindCraft（高通智匠）提供了 GB2312、GBK、GB18030、Unicode、UTF-8、BIG5 编码查询。查询步骤如下：



例如分别查询‘A’、‘B’、‘C’三个文字的编码，输入后点击转换即可看到它们的 GB2312、GBK、GB18030、Unicode、UTF-8、BIG5 编码。



3. API 接口说明及使用

3.1. 点阵字库 API 接口（此处以字库 GT30L24A3W 为例）

注意：中文使用 GB2312、GBK、GB18030 编码，日文使用 JIS0208 编码，韩文使用 KSC5601 编码，其他国家都是 Unicode 编码。以下是字符编码集的链接，可以查找。

https://www.mindcraft.com.cn/activity/client/decode

3.1.1. ASCII 的 API 调用

该函数使用的是 Unicode 编码调用 ascii 字符。获取到的字符数据会存储到用户定义的数组中。用户可以将数组里的数据与规格书中展示的参考数据进行对比是否相同。下文展示了 GT30L24A3W 规格书中展示的参考数据，如图所示：

```
/*************************************************************
函数用法：
    unsigned char DZ_Data[数组长度客户自定义];
    ASCII_GetData(0x41,ASCII_5X7,DZ_Data);       //读取 5X7 点阵 ASCII 编码 A 的点阵数据，并将点阵数据
存在 DZ_Data 数组中；数据长度为 8 BYTE
    ASCII_GetData(0x41,ASCII_7X8,DZ_Data);       //读取 7X8 点阵 ASCII 编码 A 的点阵数据，并将点阵数据
存在 DZ_Data 数组中；数据长度为 8 BYTE
```

```
    ASCII_GetData(0x41,ASCII_6X12,DZ_Data);     //读取 6X12 点阵 ASCII 编码 A 的点阵数据，并将点阵数据存在 DZ_Data 数组中；数据长度为 12 BYTE
    ASCII_GetData(0x41,ASCII_12_A,DZ_Data);     //读取 12X12 点阵 ASCII 编码 A 的点阵数据，并将点阵数据存在 DZ_Data 数组中；数据长度为 26 BYTE
    ASCII_GetData(0x41,ASCII_8X16,DZ_Data);     //读取 8X16 点阵 ASCII 编码 A 的点阵数据，并将点阵数据存在 DZ_Data 数组中；数据长度为 16 BYTE
    ASCII_GetData(0x41,ASCII_12X24_A,DZ_Data);  //读取 12X24 点阵 ASCII 编码 A 的点阵数据，并将点阵数据存在 DZ_Data 数组中；数据长度为 48 BYTE
    ASCII_GetData(0x41,ASCII_12X24_P,DZ_Data);  //读取 12X24 点阵 ASCII 编码 A 的点阵数据，并将点阵数据存在 DZ_Data 数组中；数据长度为 48 BYTE
    ASCII_GetData(0x41,ASCII_16_A,DZ_Data);     //读取 16X16 点阵 ASCII 编码 A 的点阵数据，并将点阵数据存在 DZ_Data 数组中；数据长度为 32 BYTE
    ASCII_GetData(0x41,ASCII_16X32,DZ_Data);        //读取 16X32 点阵 ASCII 编码 A 的点阵数据，并将点阵数据存在 DZ_Data 数组中；数据长度为 64 BYTE
    ASCII_GetData(0x41,ASCII_24_B,DZ_Data);         //读取 24X24 点阵 ASCII 编码 A 的点阵数据，并将点阵数据存在 DZ_Data 数组中；数据长度为 74 BYTE
    ASCII_GetData(0x41,ASCII_32_B,DZ_Data);         //读取 32X32 点阵 ASCII 编码 A 的点阵数据，并将点阵数据存在 DZ_Data 数组中；数据长度为 128 BYTE
*************************************************************/
unsigned char  ASCII_GetData(unsigned char ASCIICode,unsigned long ascii_kind,unsigned char* DZ_Data);
```

7 点阵数据验证（客户参考用）

客户将芯片内“A”的数据调出与以下进行对比。若一致，表示 SPI 驱动正常工作；若不一致，请重新编写驱动。

排置：W（横置横排）点阵大小 8X16
字母"A"
代码示例：ASCII_GetData(0X41, ASCII_8X16, DZ_Data)
点阵数据：00 00 10 38 6C C6 C6 FE C6 C6 C6 C6 00 00 00 00

3.1.2. 中文的 API 调用

使用中文的 API 之前，可以通过 MindCraft（高通智匠）的“字符编码查询器”中查询中文的 GB2312、GBK 以及 GB18030 编码。按照图 3-1 中的顺序操作即可。

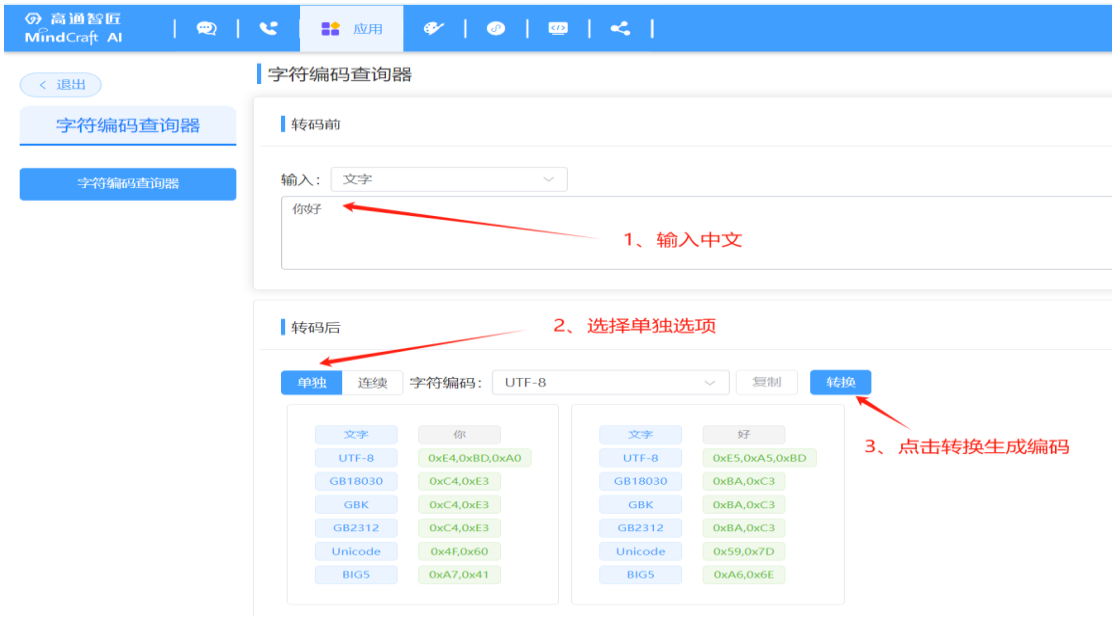


图 3-1

以下为具体的中文 API 函数。第一个参数表示编码的高字节，第二个参数表示编码的低字节。

```
/************************************************************
函数用法：
    unsigned char DZ_Data[数组长度客户自定义];
    gt_12_GetData(0xb0,0xa1,DZ_Data); //读取 12X12 点阵 汉字“啊”的点阵数据，并将点阵数据存在 DZ_Data
数组中；数据长度为 24 BYTE
************************************************************/
void gt_12_GetData (unsigned char MSB,unsigned char LSB,unsigned char *DZ_Data);
```

如果要调用的中文编码在 GBK 范围内，则要使用：

```
/************************************************************
函数用法：
    unsigned char DZ_Data[数组长度客户自定义];
    GBK_24_GetData(0xb0,0xa1,DZ_Data); //读取 24X24 点阵 汉字“啊”的点阵数据，并将点阵数据存在 DZ_Data
数组中；数据长度为 72 BYTE
************************************************************/
unsigned long GBK_24_GetData (unsigned char c1, unsigned char c2,unsigned char *DZ_Data) ;
```

3.1.3. 拉丁文的 API 调用

使用拉丁文的 API 函数前，可以通过 MindCraft（高通智匠）的“字符编码查询器”查询拉丁文的 Unicode 编码。按照图 3-2 中的顺序操作即可。

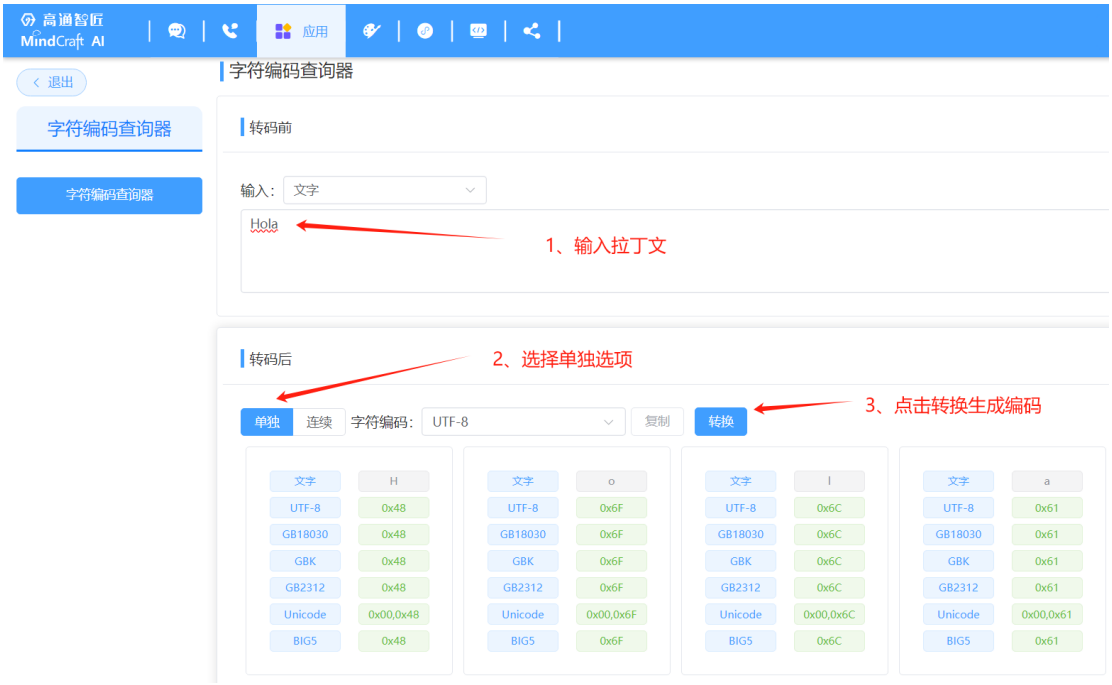


图 3-2

拉丁文适用大多数的欧美国家。例如：英国，法国，德国，西班牙，葡萄牙，荷兰等国家。该函数使用的是 Unicode 编码调用文字。如果要用到希腊或者东欧国家的文字，需要调用其他的 API 接口。此处调用的是拉丁文 8x16 字号大小，其他字号的调用方式同理。

```
/*************************************************************
函数用法：
    unsigned char DZ_Data[数组长度客户自定义];
    LATIN_8X16_GetData(0xa5,DZ_Data); //读取 8X16 点阵 拉丁文编码 0xa5 的点阵数据，并将点阵数据存在
DZ_Data 数组中；数据长度为 16 BYTE
*************************************************************/
unsigned long LATIN_8X16_GetData(unsigned int FontCode,unsigned char *DZ_Data);
```

3.1.4. 西里尔文的 API 调用

西里尔文适用部分东欧国家，例如俄罗斯，塞维利亚，乌克兰等国家。该函数使用的是 Unicode 编码调用文字。西里尔文的编码查询参考“2.5、编码查询”章节，获取 Unicode 编码。此处调用的是西里尔文 8x16 字号大小，其他字号的调用方式同理。

```
/*************************************************************
函数用法：
    unsigned char DZ_Data[数组长度客户自定义];
    CYRILLIC_8X16_GetData(0x405,DZ_Data); //读取 8X16 点阵西里尔文编码 0x405 的点阵数据，并将点阵数据
存在 DZ_Data 数组中；数据长度为 16 BYTE
*************************************************************/
unsigned long CYRILLIC_8X16_GetData(unsigned int FontCode,unsigned char *DZ_Data);
```

3.1.5. 希腊文的 API 调用

希腊文适用于希腊国家。该函数使用的是 Unicode 编码调用文字。希腊文的编码查询参考“2.5、编码查询”章节，获取 Unicode 编码。此处调用的是 8x16 字号大小的希腊文，其他字号大小调用方式同理。

```
/*************************************************************
函数用法：
    unsigned char DZ_Data[数组长度客户自定义];
    GREECE_8X16_GetData(0x375, DZ_Data);  //读取 8X16 点阵希腊文编码 0x375 的点阵数据，并将点阵数据存在 DZ_Data 数组中；数据长度为 16 BYTE
*************************************************************/
unsigned long GREECE_8X16_GetData(unsigned int FontCode, unsigned char *DZ_Data);
```

3.1.6. 希伯来的 API 调用

希伯来文适用于部分中东国家。该函数使用的是 Unicode 编码调用文字。希伯来文的编码查询参考“2.5、编码查询”章节，获取 Unicode 编码。此处调用的是 8x16 字号大小的希伯来文，其他字号大小调用方式同理。

```
/*************************************************************
函数用法：
    unsigned char DZ_Data[数组长度客户自定义];
    HEBREW_8X16_GetData(0x595, DZ_Data);  //读取 8X16 点阵希伯来文编码 0x375 的点阵数据，并将点阵数据存在 DZ_Data 数组中；数据长度为 16 BYTE
*************************************************************/
unsigned long HEBREW_8X16_GetData(unsigned int FontCode, unsigned char *DZ_Data);
```

3.1.7. 日文的 API 调用

该函数使用的是 JIS2080 编码调用文字。日文的编码查询可以通过下方链接查询对应的 JIS0208 编码：

https://download.gaotongfont.cn/GT_Font_Chip/Encoding_table/JISx0208.pdf

此处调用的是 16x16 字号大小的日文，其他字号大小调用方式同理。

```
/*************************************************************
函数用法：
    unsigned char DZ_Data[数组长度客户自定义];
    JIS0208_16X16_GetData(0x04, 0x01, DZ_Data); //读取 16X16 点阵 JIS-0208 编码 0x0401 的点阵数据，并将点阵数据存在 DZ_Data 数组中；数据长度为 32 BYTE
*************************************************************/
unsigned long JIS0208_16X16_GetData(unsigned char MSB, unsigned char LSB, unsigned char *DZ_Data);
```

3.1.8. 韩文的 API 调用

该函数使用的是 KSC 编码调用文字。韩文的编码查询可以通过下方链接查询对应的 KSC5601 编码：

https://download.gaotongfont.cn/GT_Font_Chip/Encoding_table/KSC5601.pdf

此处调用的是 16x16 字号大小的韩文，其他字号大小调用方式同理。

```
/*************************************************************
函数用法：
    unsigned char DZ_Data[数组长度客户自定义];
    KSC5601_F_16_GetData(0xb0, 0xA1, DZ_Data); //读取 16X16 点阵 KSC5601 编码 0xb0a1 的点阵数据，并将点阵数据存在 DZ_Data 数组中；数据长度为 32 BYTE
*************************************************************/
unsigned long KSC5601_F_16_GetData(unsigned char MSB, unsigned char LSB, unsigned char *DZ_Data);
```

3.1.9. 泰文的 API 调用

该函数使用的是 Unicode 编码调用泰文文字。泰文的编码查询参考“2.5、编码查询”章节，获取 Unicode 编码。此处调用的 16x24 字号大小的泰文，如果只是显示单个的泰文字符只调用 THAILAND_16X24_GetData 即可。如果要显示泰文的句子，则要使用泰文的组合算法。目前的组合算法只能显示 16x32 大小的泰文。具体代码如下：

```
/*************************************************************
函数用法：
    unsigned char DZ_Data[数组长度客户自定义];
    THAILAND_16X24_GetData(0xe05, DZ_Data);  //读取 16X24 点阵不等宽希腊文编码 0xe05 的点阵数据，并将点阵数据存在 DZ_Data 数组中；数据长度为 48 BYTE
*************************************************************/
unsigned long THAILAND_16X24_GetData(unsigned int FontCode, unsigned char *DZ_Data);
unsigned short
thai11[14]={0x0E41,0x0E2B,0x0E25,0x0E48,0x0E07,0x0E19,0x0E49,0x0E33,0x0E14,0x0E37,0x0E48,0x0E21};
unsigned char thai_dot_buf[4960],m,len;//该数组大小最好不要改变
unsigned int tcount,i;
unsigned char thai_len = 0;
thai_len = sizeof(thai11) / sizeof(thai11[0]);  //获取泰文字符个数
tcount=thai_buf_compose(thai11,thai_len,thai_dot_buf);//泰文组合
for (i=0;i<tcount;i++)
{
    Display_Y(thai_dot_buf+2+i*66, len, 100, 16, 32);//显示泰文
    m=get_width(thai_dot_buf+i*66+2,32,16);//获取显示的宽度，偏移到下次字符显示的起始点
    m=*(thai_dot_buf+1+i*66);
    len=len+m;
```

```
}
```

3.1.10. 阿拉伯文的 API 调用

该函数使用的是 Unicode 编码调用阿拉伯文文字。阿拉伯文的编码查询参考“2.5、编码查询”章节，获取 Unicode 编码。此处调用的是 16x16 字号大小的阿拉伯文，如果只是显示单个的阿拉伯字符，可以直接调用 ALB_16_GetData。如果要显示阿拉伯句子，则需要阿拉伯组合算法，具体的使用示例如下：

```
/*************************************************************
函数用法：
    unsigned char DZ_Data[数组长度客户自定义];
    ALB_16_GetData(0x632, DZ_Data); //读取 16X16 点阵阿拉伯文编码 0x632 的点阵数据，并将点阵数据存在
DZ_Data 数组中；数据长度为 32 BYTE
*************************************************************/
unsigned long ALB_16_GetData(unsigned int unicode_alb, unsigned char *DZ_Data);
#define ARABIC_UNICODE_SIZE 8   //定义数组长度
unsigned int ara[ARABIC_UNICODE_SIZE]={
0x0627,0x0644,0x062d,0x062f,0x064a,0x062f,0x064a,0x0629};//测试组合的 Unicode 编码
unsigned char dataResult[1024] = {0};   //最终接收组合字符数据 buff
int x_position = 10,y_position = 20;
ArBic_Convertor(ara, ARABIC_UNICODE_SIZE, dataResult, 16, ALB_16x16_NW , ALB_Type_W);   //W 排置
for( i = 0;i<ARABIC_UNICODE_SIZE;i++){
    //16x16 阿拉伯文不等宽，总 34 字节，前两字节宽度信息，后 32 字节为点阵数据
    Display_W(dataResult + 2 +  i * 34,x_position,y_position ,16,16); //函数参考资料文件
    x_position += dataResult[1 + i * 34];       //获取前两个字节宽度，x 游标后移准备显示下一个字符
    if(x_position > 230) {  //屏幕越界换到下一行进行显示
        x_position = 0;
        y_position += 20;
    }
}
```

3.1.11. 印地文的 API 调用

该函数使用的是 Unicode 编码调用印地文文字。印地文的编码查询参考“2.5、编码查询”章节，获取 Unicode 编码。此处调用的是 16x16 字号大小的印地文，如果只是显示单个的印地文字符，可以直接调用 Indic_16_GetData。如果要显示印地文句子，则需要使用组合算法，具体的使用示例如下：

```
/*************************************************************
函数用法：
    unsigned char DZ_Data[数组长度客户自定义];
    Indic_16_GetData(0x0905, DZ_Data); //读取 16X16 点阵阿拉伯文编码 0x0905 的点阵数据，并将点阵数据
存在 DZ_Data 数组中；数据长度为 32 BYTE
*************************************************************/
```

```
unsigned long Indic_16_GetData(unsigned int FontCode, unsigned char *DZ_Data);
/*
 * 印地文组合算法
 * 参数 ：unicode 输入的印地文 Unicode 编码，Nub 输入的编码数据量，DZ_Data 返回的数据缓存数组
 * 返回值：组合后的字符数量
 */
unsigned short Devanagari_Convertor(unsigned short *unicode,unsigned short Nub,unsigned char
*DZ_Data);
// 印地文测试示例
-----------------------------------------------------------------
unsigned char DataBuff[48*500];
void Devanagari_Test(void)
{
    unsigned short i,nub=0,x=0,y=0,w=0,h=0;
    unsigned short charColor=0xffff,bkColor=0x0000;
    unsigned char  codeNub=120;
    unsigned short unicode[]={
        0x0917,0x0941,0x0924,0x093E,0x0930,0x0947,0x0938,0x0020,0x0928,0x0947,0x002
0,0x091A,0x0947,0x0924,0x093E,0x0935,0x0928,0x0940,0x0020,0x0926,
        0x0940,0x0020,0x0915,0x093F,0x0020,0x092F,0x0926,0x093F,0x0020,0x0924,0x092
4,0x094D,0x0915,0x093E,0x0932,0x0020,0x0915,0x0926,0x092E,0x0020,
        0x0928,0x0939,0x0940,0x0902,0x0020,0x0909,0x0920,0x093E,0x090F,0x0020,0x091
7,0x090F,0x0020,0x0924,0x094B,0x0020,0x0938,0x093E,0x092B,0x0020,
        0x0939,0x0948,0x0020,0x0915,0x093F,0x0020,0x092D,0x0940,0x0937,0x0923,0x002
0,0x0935,0x0948,0x0936,0x094D,0x0935,0x093F,0x0915,0x0020,0x0916,
        0x093E,0x0926,0x094D,0x092F,0x093E,0x0928,0x094D,0x0928,0x0020,0x0906,0x092
A,0x093E,0x0924,0x0020,0x0938,0x094D,0x0925,0x093F,0x0924,0x093F,
        0x0020,0x0915,0x093E,0x0020,0x091C,0x094B,0x0916,0x093F,0x092E,0x0020,0x092
C,0x0922,0x093C,0x0020,0x0930,0x0939,0x093E,0x0020,0x0939,0x0948,
    };
    w=16;h=24;
    x=5;y+=30;
    nub = Devanagari_Convertor(unicode,codeNub,DataBuff);   //印地文组合
    for(i=0;i<nub;i++)
    {
        Display_Y( &DataBuff[i*48],x, y, w, h );    //显示组合后的印地文
        x += get_width_Y(&DataBuff[i*48],h,w);      //获取字符的宽度，偏移到下次字符显
示的起始点
        if(get_width_Y(&DataBuff[i*48],h,w)==0)//不存在的编码则空格跳过
            x+=5;
        if((x+w)>300){
            x=5;y+=h+1;
        }
    }
```

```
}
```

3.1.12. 高棉文的 API 调用

该函数使用的是 Unicode 编码调用高棉文文字。高棉文的编码查询参考“2.5、编码查询”章节，获取 Unicode 编码。此处调用的是 24x24 字号大小的高棉文，其他字号大小调用方式同理。

```
/*************************************************************
不等宽高棉文
函数用法：
    unsigned char DZ_Data[数组长度客户自定义];
    Khmer_24_GetData(0x19e0, DZ_Data); //读取 24X24 点阵不等宽高棉文编码 0x19e0 的点阵数据，并将点阵数据存在 DZ_Data 数组中；数据长度为 72 BYTE
*************************************************************/
unsigned long Khmer_24_GetData(unsigned int FontCode, unsigned char *DZ_Data);
```

3.2. 矢量字库 API 接口（此处以 GT5SLAD3BFA 为例）

获取矢量文字的接口函数有两种，一种是通过 get_font_st 函数获取矢量字符，第二种是通过 get_font 函数获取矢量字符。但是他们的作用是一样的，只是调用的方法有些不同。显示不同国家的文字，只需要根据提供的宏放到参数 sty 中即可。注意：调用中文用的是 GB 编码，调用其他外文用的是 Unicode 编码。

```
#define ZK_BUFFER_LEN   4608    //可修改大小，约等于 字号*字号/8.
typedef struct vecFont
{
    unsigned long fontCode;     //字符编码中文：GB18030，ASCII/外文：unicode
    unsigned char type;         //字体    @矢量公用部分
    unsigned char size;         //文字大小
    unsigned char thick;        //文字粗细
    unsigned char zkBuffer[ZK_BUFFER_LEN];    //数据存储
}VECFONT_ST;
unsigned int get_font_st(VECFONT_ST *font_st);
/**
  * @brief  矢量文字读取函数
  * @param  pBits : 数据存储
  * @param  sty   : VEC_SONG_STY;  VEC_FANG_STY; VEC_BLACK_STY; VEC_KAI_STY; VEC_FT_ASCII_STY; VEC_DZ_ASCII_STY; VEC_CH_ASCII_STY;
  * @param  fontCode：字符编码 ： 中文 GBK 编码；ASCII 码
  * @param  width : 文字宽度
  * @param  height  : 文字高度
  * @param  thick : 文字粗细
  * @retval 返回字符显示需要的字号大小
```

```
 */
unsigned int get_font(unsigned char *pBits,unsigned char sty,unsigned long fontCode,unsigned char
width,unsigned char height, unsigned char thick);
```

3.3. 灰度矢量字库 API 接口（此处以字库 GT5SLAD3BFA 为例）

参数的填写方式跟上述的矢量字库一致，函数会返回包含字符宽度和灰度阶级的数组首地址。实际的宽度数据就不需要额外的函数获取了。

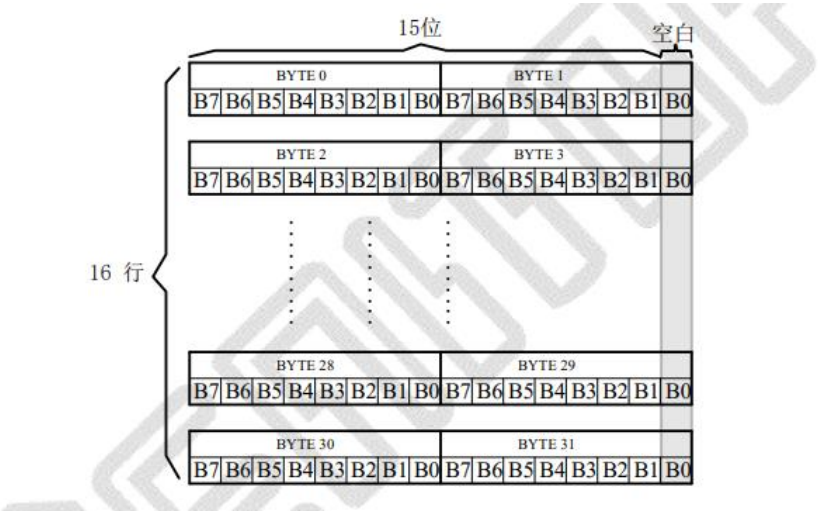
```
/**
 * @brief 灰度矢量文字读取函数
 * @param pBits    文字数据缓存
 * @param sty      文字字体选择  @矢量公用部分
 * @param fontCode 字符编码中文: GB18030, ASCII/外文: unicode
 * @param fontSize 文字大小
 * @param thick    文字粗细
 * @return unsigned int * ret_buff[0] 字符的显示宽度 , ret_buff[1] 字符的灰度阶级[1 阶/2 阶/3 阶/4
阶]
 */
unsigned int *get_Font_Gray(unsigned char *pBits, unsigned char sty, unsigned long fontCode,
unsigned char fontSize, unsigned char thick);
```

4. 字符显示及各种问题

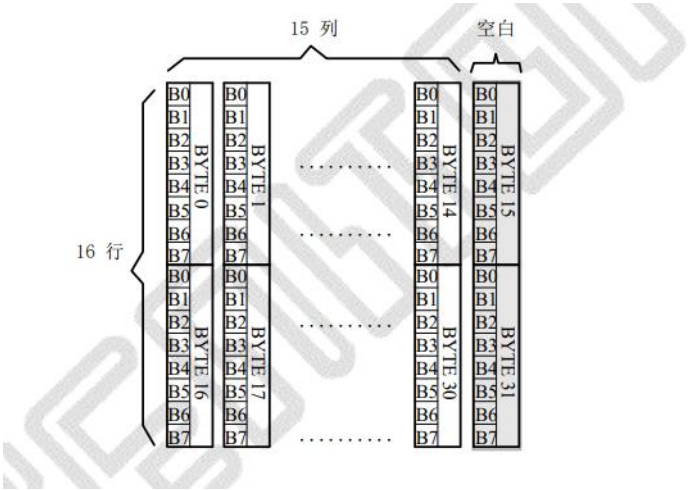
4.1. 字符排置解释（横置横排/竖置横排）

在显示字符之前，用户需确认使用的字库是横置横排还是竖置横排。如果字库型号尾部是‘Y’字样那么就是竖置横排；如果字库型号尾部是‘W’字样或者没有，那么就是横置横排。

横置横排：比如宽度是 16bit 位，高度是 16bit 位的字符。那么“横置”指的就是两个字节水平放置，即宽度。“横排”指的是每个字节从做到右横向放两个字节，满两个字节换行继续放，一共有 16 排，即高度。如下图所示。



竖置横排：比如宽度是 16bit 位，高度是 16bit 位的字符。那么“竖置”指的就是每个字节都是竖直放置，放 16 个字节，即宽度。“横排”指的从左到右放 16 个字节，满 16 个字节换行继续放，一共有 2 排，即高度。如下图所示。



4. 2. 显示函数（TFT 彩屏）

首先用户要确定使用的字库芯片是横置横排还是竖置横排。如果是竖置横排可以参考 Display_Y 函数，反之参考 Display_W 函数，这里两个是打点显示的代码，LCD_Display_Area 是区域刷新显示代码。以下是打点显示以及区域刷新显示代码。如果想快速显示可以使用区域刷新显示代码。

```
/**
 * @brief 竖置横排显示
 *
 * @param pBits 存放字符数据的数组
 * @param x 竖轴坐标
```

```
 * @param y 横轴坐标
 * @param widt 字符宽度
 * @param high 字符高度
 */
void Display_Y(unsigned char *pBits,unsigned int x,unsigned int y,unsigned int
widt,unsigned int high)
{
    unsigned int i,j,k,n;
    unsigned char temp;
    n = 0;
    for( i = 0;i < ((high+7)>> 3); i++)//遍历每一行
    {
        for( j = 0;j < widt;j++)//遍历每个字节
        {
            temp = pBits[n++];
            for(k = 0;k < 8;k++)//遍历每一位
            {
                if(((temp >> k)& 0x01) == 0 )
                {
                    /* 显示一个像素点 */
                    LCD_Fast_DrawPoint( x+j, y+k+(i*8), 0xFFFF);
                }
                else
                {
                    /* 显示一个像素点 */
                    LCD_Fast_DrawPoint( x+j, y+k+(i*8), 0x0000);
                }
            }
        }
    }
}
/**
 * @brief 横置横排显示
 *
 * @param pBits 存放字符数据的数组
 * @param x 竖轴坐标
 * @param y 横轴坐标
 * @param widt 字符宽度
 * @param high 字符高度
 */
void Display_W(unsigned char *pBits,unsigned int x,unsigned int y,unsigned int
widt,unsigned int high)
{
    unsigned int i,j,k,n;
```

```
    unsigned char temp;
    n = 0;
    for( i = 0;i < high; i++)//遍历每一行
    {
        for( j = 0;j < ((widt+7)>> 3);j++)//遍历每个字节
        {
            temp = pBits[n++];
            for(k = 0;k < 8;k++)
            {
                if(((temp << k)& 0x80) == 0 )//遍历每一位
                {
                    /* 显示一个像素点 */
                    LCD_Fast_DrawPoint( x+k+(j*8), y+i, 0xFFFF);
                }
                else
                {
                    /* 显示一个像素点 */
                    LCD_Fast_DrawPoint( x+k+(j*8), y+i, 0x0000);
                }
            }
        }
    }
}
/**
 * @brief 设置显示区域，快刷显示
 *
 * @param buff 存放字符数据的数组
 * @param x 竖轴坐标
 * @param y 横轴坐标
 * @param widt 字符宽度
 * @param high 字符高度
 */
void LCD_Display_Area(unsigned char* buff,unsigned int x,unsigned int y,unsigned int
widt,unsigned int high )
{
    uint16_t i,j,k;
    uint16_t da, color;
    unsigned char* p = buff;
    unsigned char c;
    uint32_t t, h = high;
    k=0;
    t = (widt+7)/8;
    LCD_Set_Window(x,y,(((widt+7)/8)*8),high);  //设置窗口
    LCD_WriteRAM_Prepare();     //开始写入 GRAM
```

```
    for(i=0;i<t*h;i++)
    {
        c=*p++;
        for(j=0;j<8;j++) //循环判断每一位是 1 还是 0
        {
            if(c&0x80)
                color=0x0000;
            else
                color=0xffff;
            c<<=1;
            //写 16 位数据
            LCD_WriteRAM(color);
        }
    }
}
```

灰度矢量的显示函数可以参考下文代码。重点关注设置显示区域的代码和设置写入 GRAM 寄存器指令的代码。最后通过 LCD_WR_DATA 将每个像素点的灰度数据写到 GRAM 中。注意：用户需根据使用的 LCD 进行代码调整。

```
#define DISMODE 1 // 1  刷图显示  0 打点显示
//描点函数
#if DISMODE == 0
static void Gui_DrawPoint(unsigned int x, unsigned int y, unsigned int Data)
{
  //显示一个点
  putPixel( x, y, Data);
}
#endif
/*-------------------------------------------------------------------------------------
 * 灰度数据显示函数 1 阶灰度/2 阶灰度/4 阶灰度
 * 参数 :
 * data 灰度数据;  x,y=显示起始坐标 ; w 宽度, h 高度,grade 灰度阶级[1 阶/2 阶/4 阶]
 * HB_par 1 白底黑字  0 黑底白字
 *-------------------------------------------------------------------------------------*/
void Gray_Display_hz(unsigned char *data,unsigned short x,unsigned short y,
  unsigned short w ,unsigned short h,unsigned char grade, unsigned char HB_par)
{
  unsigned int temp=0,gray,x_temp=x;
  unsigned int i=0,j=0,k=0,t;
  unsigned char c,c2,*p;
  unsigned long color8bit,color4bit,color3bit[8],color2bit,color;

  t=(w+7)/8*grade;//
  p=data;
    t=(w+7)/8*4;//
```

```
#if DISMODE
LCD_Set_Window(x,y,(((w+7)/8)*8),h);  //设置显示区域
LCD_WriteRAM_Prepare();   //设置写入 GRAM 寄存器指令
#endif
if(grade==2)  //2bits
{
  for(i=0;i<t*h;i++)
  {
    c=*p++;
    for(j=0;j<4;j++)
    {
      color2bit=(c>>6);//获取像素点的 2bit 颜色值
      if(HB_par==1)
        color2bit= (3-color2bit)*250/3;//白底黑字
      else
        color2bit= color2bit*250/3;//黑底白字
      gray=color2bit/8;
      color=(0x001f&gray)<<11;              //r-5
      color=color|(((0x003f)&(gray*2))<<5); //g-6
      color=color|(0x001f&gray);            //b-5
      temp=color;
      temp=temp;
      c<<=2;
      #if DISMODE
      //写 16 位数据
      LCD_WriteRAM(temp);
      #else
      //打点
      if(x<(x_temp+w))
      {
        Gui_DrawPoint(x,y,temp);
      }
      x++;
      if(x>=x_temp+(w+7)/8*8) {x=x_temp; y++;}
      #endif
    }
  }
}
else if(grade==3)//3bits
{
  for(i=0;i<t*h;i+=3)
  {
    c=*p; c2=*(p+1);
    color3bit[0]=(c>>5)&0x07;
```

```
    color3bit[1]=(c>>2)&0x07;
    color3bit[2]=((c<<1)|(c2>>7))&0x07;
    p++;
    c=*p; c2=*(p+1);
    color3bit[3]=(c>>4)&0x07;
    color3bit[4]=(c>>1)&0x07;
    color3bit[5]=((c<<2)|(c2>>6))&0x07;
    p++;
    c=*p;
    color3bit[6]=(c>>3)&0x07;
    color3bit[7]=(c>>0)&0x07;
    p++;
    for(j=0;j<8;j++)
    {
      if(HB_par==1)
      color3bit[j]= (7-color3bit[j])*255/7;//白底黑字
      else
      color3bit[j]=color3bit[j]*255/7;//黑底白字
      gray =color3bit[j]/8;
      color=(0x001f&gray)<<11;              //r-5
      color=color|(((0x003f)&(gray*2))<<5); //g-6
      color=color|(0x001f&gray);            //b-5
      temp =color;
      #if DISMODE
      //写 16 位数据
      LCD_WriteRAM(temp);
      #else
      //打点
      if(x<(x_temp+w))
      {
        Gui_DrawPoint(x,y,temp);
      }
      x++;
      if(x>=x_temp+(w+7)/8*8) {x=x_temp; y++;}
      #endif
    }
  }
}
else if(grade==4) //4bits
{
  for(i=0;i<t*h;i++)
  {
    c=*p++;
    for(j=0;j<2;j++)
```

```
    {
      color4bit=(c>>4);
      if(HB_par==1)
        color4bit= (15-color4bit)*255/15;//白底黑字
      else
        color4bit= color4bit*255/15;//黑底白字
      gray=color4bit/8;
      color=(0x001f&gray)<<11;        //r-5
      color=color|(((0x003f)&(gray*2))<<5); //g-6
      color=color|(0x001f&gray);        //b-5
      //printf("color = 0x%X\r\n",color4bit);
      temp=color;
      c<<=4;
      #if DISMODE
      //写 16 位数据
      LCD_WriteRAM(temp);
      #else
      //打点
      if(x<(x_temp+w)){
        Gui_DrawPoint(x,y,temp);
      }
      x++;
      if(x>=x_temp+(w+7)/8*8) {x=x_temp; y++;}
      #endif
    }
  }
}
else if(grade == 5 || grade == 6)
{
 for(i=0;i<t*h;i++)
  {
    c=*p++;
    for(j=0;j<2;j++)
    {
      color4bit=(c>>4);
      if(HB_par==1)
        color4bit= (15-color4bit)*255/15;//白底黑字
      else
        color4bit= color4bit*255/15;//黑底白字

      if(color4bit > 0x00 && color4bit < 0xFF && grade == 5)
        color4bit = color4bit - 0x11;
      else if(color4bit >= 0x11 && color4bit < 0xFF && grade == 6)
        color4bit =    (color4bit == 0x11) ? (color4bit - 0x11) : (color4bit - 0x22);
```

```
      gray=color4bit/8;
      color=(0x001f&gray)<<11;        //r-5
      color=color|(((0x003f)&(gray*2))<<5); //g-6
      color=color|(0x001f&gray);        //b-5
      //printf("color = 0x%X\r\n",color4bit);
      temp=color;
      c<<=4;
      #if DISMODE
      //写 16 位数据
      LCD_WriteRAM(temp);
      #else
      //打点
      if(x<(x_temp+w)){
        Gui_DrawPoint(x,y,temp);
      }
      x++;
      if(x>=x_temp+(w+7)/8*8) {x=x_temp; y++;}
      #endif
    }
  }
}
else  //1bits
{
  for(i=0;i<t*h;i++)
  {
    c=*p++;
    for(j=0;j<8;j++)
    {
      if(c&0x80) color=0x0000;
      else color=0xffff;
      c<<=1;
      #if DISMODE
      //写 16 位数据
      LCD_WriteRAM(color);
      #else
      //打点
      if(x<(x_temp+w))
      {
        if(color == 0x0000 && HB_par == 1)
        {
          Gui_DrawPoint(x,y,color); //打点
        }
        else if(HB_par == 0 && color == 0x0000)
```

```
        {
          Gui_DrawPoint(x,y,~color);  //打点
        }
       }
       x++;
       if(x>=x_temp+(w+7)/8*8) {x=x_temp; y++;}
       #endif
     }
    }
  }
}
```

4.3. 显示函数（STN 黑白屏）

```
/**
 * @brief 设置坐标
 *
 * @param page 竖轴起始坐标
 * @param column 横轴起始坐标
 */
void LCD_SetPos(uint8_t page, uint8_t column)
{
    column = column - 1; //我们平常所说的第 1 列，在 LCD 驱动 IC 里是第 0 列。所以在这里减去 1.
    page = page - 1;
    write_com(0xb0 + page); /*设置页地址。每页是 8 行。一个画面的 64 行被分成 8 个页。我们平常所说的第 1 页，在 LCD 驱动 IC 里是第 0 页，所以在这里减去 1*/
    write_com(column & 0x0f); //设置列地址的低 4 位
    column >>= 4;
    column = column | 0x10;
    write_com(column); //取高 4 位行地址
}

/**
 * @brief 显示 16x16 点阵图像、汉字、生僻字或 16x16
 *
 * @param y 横轴坐标
 * @param x 竖轴坐标
 * @param dp 存放字符数据的数组
 */
void display_graphic_16x16(uint8_t y, uint8_t x, unsigned char *dp)
{
    uint8_t i;
    LCD_SetPos(y, x); //发送地址，第一页
```

```
    for (i = 0; i < 16; i++) //第一页的数据 16 字节，列号自动累加
    {
        write_data(*dp); //写数据到 LCD，如果要反色显示，就在*P 前用~取反。
        dp++;
    }
    LCD_SetPos(y + 1, x); //发送地址，第二页
    for (i = 0; i < 16; i++) //第二页的数据 16 字节
    {
        write_data(*dp); //写数据到 LCD，如果要反色显示，就在*P 前用~取反。
        dp++;
    }
}
```

4. 4. 点阵字库的显示

通过上述的“4.1、点阵字库 API 接口”，能获取到文字的点阵数据。数据会存放在定义的数组里，那么接下来就是将数组里的数据显示到屏幕上。我们以显示 8x16 字号大小的 ASCII 字符为例：

```
char pBits[16] = {0};
ASCII_GetData(0x41, ASCII_8X16, pBits);
Display_W(pBits, 100, 100, 8, 16);
```

4. 5. 矢量字库的显示

通过上述的“4.2、矢量字库 API 接口”，以下面是两种矢量字库 API 接口的使用方式。

使用 get_font_st 获取矢量字符时，注意宏定义 ZK_BUFFER_LEN 的大小，这里决定了存放矢量字符数据的数组的大小，用户可以根据自己的需求进行改动。示例中调用的是 ASCII 中的‘A’字符，type 选的是 ASCII 圆角类型，size 字号大小是 16，粗细 thick 建议跟 size 一样，这里一样是 16，zkBuffer 就是用来存矢量字符数据的数组。get_font_st 返回值表示字符显示时需要的宽度。

```
VECFONT_ST font_st = {.fontCode = 0x41, .type = VEC_FT_ASCII_STY, .size = 16, .thick = 16, .zkBuffer
= {0}};
uint32_t width =  get_font_st(&font_st);
Display_W(font_st.zkBuffer, 100, 100, width, 16);
```

使用 get_font 获取矢量字符的方法与 get_font_st 类似，这里定义了一个 256 字节大小的数组。get_font 返回值表示字符显示时需要的宽度。

```
char pBits[256] = {0};
uint32_t width = get_font(pBits, VEC_FT_ASCII_STY, 0x41, 16, 16, 16);
Display_W(pBits, 100, 100, width, 16);
```

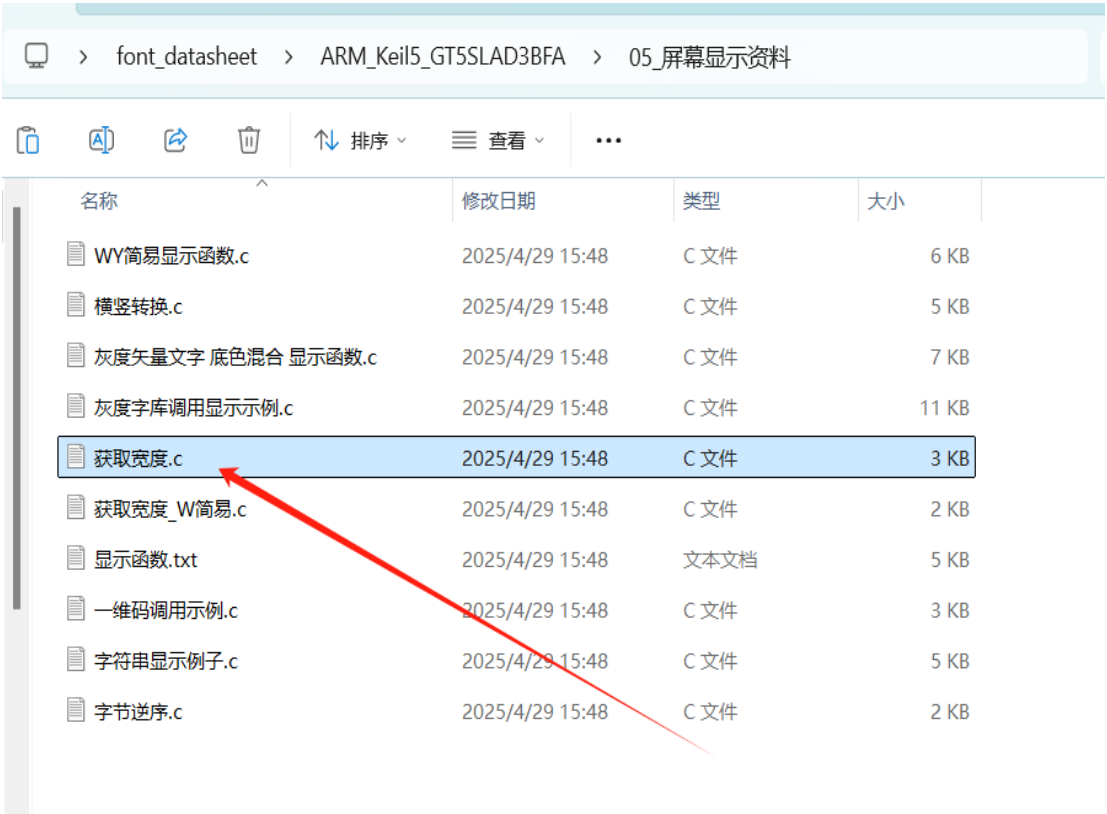
4. 6. 灰度矢量字库/灰度字库的显示

通过上述的“4.3、灰度矢量字库 API 接口”，使用 get_Font_Gray 函数获取灰度数据，并返回灰度字符的宽度数据和灰度值。显示的时候必须要用到这两个参数显示字符，否则无法正常显示。

```
unsigned int* width = NULL;
width = get_Font_Gray(pBits, VEC_SONG_STY, 0xB0A1, 24, 24); //获取灰度矢量字符
Gray_Display_hz(pBits,150,100, width[0] ,24, width[1], 1);  //width[0]：宽度数据，width[1]：灰度数据
```

4. 7. 获取点阵，矢量的实际宽度

获取实际的宽度主要针对于外文且大于 12 字号大小的字符。中文的宽度都是等宽。打开“05_屏幕显示资料\获取宽度.c”，直接拷贝获取宽度的代码。



具体的代码以及参数含义如下文所示：

```
/**
 * @brief Get the width object
 *
 * @param p 字符数据数组首地址
 * @param zfwidth 获取字符时填写的宽度
 * @param zfhigh 获取字符时填写的高度
 * @return unsigned int 实际宽度
```

```
 */
unsigned int get_width(unsigned char *p,unsigned int zfwidth,unsigned int zfhigh )
{
  unsigned char *q;
  unsigned int i,j,tem,tem1,witdh1=0,witdh2=0;
  q=p;
  for (i=0;i<zfwidth/16;i++)
  {
      tem=0;
    tem1=0;
    for (j=0;j<zfhigh;j++)
    {
       tem=(*(q+j*(zfwidth/8)+i*2)|(*(q+1+j*(zfwidth/8)+i*2))<<8);
       tem1=tem1|tem;
    }
    witdh1=0;
    for (j=0;j<16;j++)
    {
      if (((tem1<<j)&0x8000)==0x8000)
      {
        witdh1=j;
      }
    }
    witdh2+=witdh1;
  }
  return witdh2;
}
```

点阵获取实际宽度：

```
cchar pBits[16] = {0};
ASCII_GetData(0x41, ASCII_16_A, pBits); //按照 16x16 大小获取
int width = get_width(pBits, 16, 16);   //获取实际宽度
Display_W(pBits, 100, 100, width, 16);  //显示实际宽度
```

矢量获取实际宽度：

```
VECFONT_ST font_st = {.fontCode = 0x41, .type = VEC_FT_ASCII_STY, .size = 16, .thick = 16, .zkBuffer
= {0}};
get_font_st(&font_st);  //获取矢量字符数据
int width = get_width(font_st.zkBuffer, 16, 16);  //获取实际的字符宽度
Display_W(font_st.zkBuffer, 100, 100, width, 16);



char pBits[256] = {0};
get_font(pBits, VEC_FT_ASCII_STY, 0x41, 16, 16, 16);    //获取矢量字符数据
int width = get_width(pBits, 16, 16);  //获取实际的字符宽度
Display_W(pBits, 100, 100, width, 16);
```

4. 8. 字符显示的横竖转换问题

在上述我们有提到显示字符前，要确认使用的字库是横置横排还是竖置横排。比如字库里的字符都是横置横排的，但是显示的时候要用竖置横排的方式显示，那么我们就需要进行一次横竖转换。我们以 GT30L24A3W 为例。打开“05_屏幕显示资料\横竖转换.c”文件，具体代码如下文所示，可以直接拷贝到工程中。如果要实现横置横排转竖置横排，选择宏定义 HZ_MODE00 即可。注意：目前横竖转换的函数只能用在点阵字库。

```
#define  HZ_MODE00 0//竖置横排(Y)
#define  HZ_MODE01 1//竖置竖排(Z)
#define  HZ_MODE10 2//横置横排(W)
#define  HZ_MODE11 3//横置竖排(X)
#define  HZ_MODE4  4//Y-->W
//条件 putdata 的数据写入 X，Y 的 LCD 显示缓冲 RAM 中
/*
*getdate: 转换后的字形数据
*putdate: 转换签订字形数据
*width: 字宽
*high:  字高
*style: 转换功能的选择，  HZ_MODE00 0//竖置横排(Y)
                         HZ_MODE01 1//竖置竖排(Z)
                         HZ_MODE10 2//横置横排(W)
                         HZ_MODE11 3//横置竖排(X)
                         HZ_MODE4  4//Y-->W

*使用示例：lcdram_map(getdate,putdate,32,32,HZ_MODE4);//将 32x32 的字形数据由 Y 排置（竖置横排）转换成 W 排置（横置横排）保存在 getdate 数组中
*/
void lcdram_map( u8 *getdate,u8 *putdata, u8 width, u8 high, u8 style )
{
  u16 i,j,hbyte,wbyte;
  unsigned char i_8,j_8;
  wbyte = (width+7)/8;
  hbyte = (high+7)/8;
   //-------------------------------------------------------------------------
   // Y--> W; Y-->X; Y-->Z;
   //-------------------------------------------------------------------------
  if( style == HZ_MODE4 )   //竖置横排 转 横置横排 ( Y-->W )
  {
    for( i = 0; i < high; i++ )
        for( j = 0; j < width; j++ )
        {
            i_8 = i/8;
            if((*(putdata+i_8*width+j)&(0x01<<(i%8))) > 0)
```

```
                getdate[wbyte*i+j/8] |= (0x80>>(j%8));
            else
                getdate[wbyte*i+j/8] &= (~(0x80>>(j%8)));
        }
}
 //-------------------------------------------------------------------------
 // W--> Y;  W-->Z;W-->X;
 //-------------------------------------------------------------------------

if( style == HZ_MODE00 )  //竖置横排 (W--> Y)
{
    for( i = 0; i < high; i++ )
        for( j = 0; j < width; j++ )
        {
          i_8 = i/8;
          if((*(putdata+wbyte*i+j/8)&(0x80>>(j%8))) > 0)
              getdate[i_8*width+j] |= (0x01<<(i%8));
          else
              getdate[i_8*width+j] &= (~(0x01<<(i%8)));
        }
}
else if(style == HZ_MODE01)   //竖置竖排 (W-->Z)
{
  for( i = 0; i < high; i++ )
    for( j = 0; j < width; j++ )
    {
      if((*(putdata+wbyte*i+j/8)&(0x80>>(j%8))) > 0)
          getdate[j*hbyte+i/8] |= (0x01<<(i%8));
      else
          getdate[j*hbyte+i/8]  &= (~(0x01<<(i%8)));
    }
}
else if(style == HZ_MODE11)//横置竖排 (W-->X)
{
  for( i = 0; i < high; i++ )
    for( j = 0; j < width; j++ )
    {
      j_8 = j/8;
      if((*(putdata+wbyte*i+j/8)&(0x80>>(j%8))) > 0)
          getdate[j_8*high+i] |= (0x80>>(j%8));
      else
          getdate[j_8*high+i] &= (~(0x80>>(j%8)));
    }
}
```

```
  else if(style == HZ_MODE10)//横置横排 做镜像(W-->W')
  {
    for( i = 0; i < high; i++ )
      for( j = 0; j < width; j++ )
      {
        if((*(putdata+wbyte*i+j/8)&(0x80>>(j%8)))>0 )
            *(getdate+wbyte*i+(width-j)/8) |=(0x80>>((width-j)%8));
        else
            *(getdate+wbyte*i+(width-j)/8) &=~(0x80>>((width-j)%8));
      }
  }
}
```

转换代码的使用方法示例如下：

提前定义两个同样大小的数组，先使用 pBits 存放从字库芯片里读取到的数据。然后使用 get_pBits 在 lcdram_map 中获取转换后的数据，接着再通过竖置横排的显示函数进行显示即可。

```
char pBits[16] = {0};
char get_pBits[16] = {0};
ASCII_GetData(0x41, ASCII_16_A, pBits); //按照 16x16 大小获取
void lcdram_map(get_pBits, pBits, 16, 16, HZ_MODE00);  //开始转换
Display_Y(pBits, 100, 100, 16, 16);  //显示实际宽度
```

4. 9. 一维码获取及显示

打开“05_屏幕显示资料\一维码调用示例.c”，将代码拷贝到工程中即可。首先用户需要实现 SPI_Address 和 r_dat_bat 两个函数，参考代码如下。SPI_Address 函数中，用户只需将发送字节的函数替换成自己的 SPI 发送函数即可，r_dat_bat 函数的实现同理，并将 CS 拉高和拉低的代码替换成自己的即可。具体的一维码代码接口及调用示例如下。

```
void SPI_Address(unsigned char AddH,unsigned char AddM,unsigned char AddL)
{
    Send_Byte(AddH);    //更换成开发者的 spi 发送字节函数；
    Send_Byte(AddM);
    Send_Byte(AddL);
}
/*****************************************************************************/
//                  Get N bytes sub-pro  (STM8,STM32, 51)                    //
/*****************************************************************************/
//客户自己实现，从 address 地址读取 len 个字节的数据并存入到 pBuff 数组当中
unsigned char r_dat_bat(unsigned long address,unsigned long DataLen,unsigned char *pBuff)
{
    unsigned long i;
  unsigned char addrHigh;
  unsigned char addrMid;
```

```
  unsigned char addrLow;
  addrHigh=address>>16;
  addrMid=address>>8;
  addrLow=(unsigned char)address;
    Rom_csL;          //更换成开发者的 cs 拉低，片选选中字库芯片
    Send_Byte(0x03);    //更换成开发者的 spi 发送字节函数；普通读取首先送 0X03,然后发送地址高八位
addrHigh，中八位 addrMid，低八位 addrLow。
    SPI_Address(addrHigh,addrMid,addrLow);
    for(i=0;i<DataLen;i++)
    *(pBuff+i)=Get_Byte();
    Rom_csH;        //这里更换成开发者的 cs 拉高，片选选中字库芯片
    return 0;
}
/**
 * @brief 条形码 EAN13 ：将数组条形码转为对应条形码图形地址
 * @param bar_num_p 条形码 EAN13 数字数组指针，数组包含 13 个数字。
 * @return unsigned long * 数组条形码[13]对应条形码图形地址
 */
unsigned long *BAR_CODE_EAN13(unsigned char *bar_num_p);
/* code_128_type */
#define CODE_128_A         (1)
#define CODE_128_B         (2)
#define CODE_128_C         (3)
/**
 * @brief GB/T 18347-2001(CODE128)条形码：将数组条形码转为对应条形码图形地址
 * @param bar_num_p 条形码数字数组指针，bar_num_p[4]数组包含 4 个条形码 ASCII 符(数组取值为 0~94)。
 * @param bar_num_len 条形码数字数组 的长度
 * @param code_type 起始符 编码格式类型 @ref code_128_type
 * @return unsigned long * 对应条形码图形地址
 */
unsigned long *BAR_CODE128(unsigned char *bar_num_p, unsigned char bar_num_len, unsigned char
code_type);

/************************************
arr[] : 13 个数组
************************************/
void  BAR_codeEAN13(unsigned int x, unsigned int y, unsigned char arr[])
{
  unsigned char buff_temp[60];
  unsigned long *tmp ;
  unsigned char i = 0;

  tmp = BAR_CODE_EAN13(arr);
  for(i = 0;i < 13;i++)
```

```
  {
    memset(buff_temp,0,sizeof(buff_temp));
    r_dat_bat(*tmp, 54, buff_temp);
    DisZK_DZ_W(x,y,16,28,BLACK,WHITE,&buff_temp[2],1);
    x+=buff_temp[1];
    tmp++;
  }
}
/*************************************
ascii_arr[] : 字符数组 ASCII
arr_len:  字符数组的长度
1、Code 39 码只接受如下 43 个有效输入字符：
2、26 个大写字母(A – Z);
3、十个数字(0 – 9)：
连接号(-),句号(.),空格,美圆符号($),斜扛(/),加号(+)以及百分号(%)。其余的输入将被忽略。
*************************************/
void BAR_codeEAN39(unsigned int x, unsigned int y, unsigned char ascii_arr[], unsigned int arr_len)
{
  unsigned char buff_tmp[50], i;

  for(i=0; i<arr_len; i++) {
    BAR_CODE39(ascii_arr[i], buff_tmp);
    DisZK_DZ_W(x,y,16,22,BLACK,WHITE,&buff_tmp[0],1);
    x+=11;
  }
}
/*************************************
arr[]: 条形码编码 整数
arr_len: 数组长度
flag: 起始符有 3 种模式
  当 flag=1 时为 Code-128-A;
  当 flag=2 时为 Code-128-B;
  当 flag=3 时为 Code-128-C;
Code128 编码表参考: https://blog.csdn.net/rodulf/article/details/51276820
*************************************/
void BAR_codeEAN128(unsigned int x, unsigned y, unsigned char arr[], unsigned int arr_len, unsigned
char flag) {
  unsigned char buff_tmp[60];
  unsigned long *tmp;
  unsigned char i=0;
  const unsigned char verify = 3;

  tmp = BAR_CODE128(arr, arr_len, flag);
```

```
  for(i=0; i<arr_len+verify; i++) {
    memset(buff_tmp, 0, sizeof(buff_tmp));
    r_dat_bat(*tmp, 40, buff_tmp);
    DisZK_DZ_W(x, y, 16, 20, BLACK, WHITE, &buff_tmp[0], 1);  //显示函数，根据实际修改，参考显示函数文件
    x+=11;  //可修改
    tmp++;
  }
}
```

一维码的显示函数可以参考“05_屏幕显示资料\显示函数.txt”中的代码。具体代码如下：

```
/*横置横排打点函数 ----------------------------------------------------------------*/
/*
  Xpos,Ypos：显示的起点坐标
  data：显示的数据
  charColor：字体颜色
  bkColor：背景颜色
  sizeType：放大倍数，默认为1
*/
void WriteData( uint16_t Xpos, uint16_t Ypos, uint8_t data, uint16_t charColor, uint16_t bkColor,u8 sizeType)
{
    uint16_t j,i;
    unsigned char count=0;
    for( j=0; j<8; j++ )
    {
        if( ((data >> (7-j))&0x01)== 0x01 ){
            for(count=0;count<sizeType;count++){
              for(i=0;i<sizeType;i++){
                LCD_SetPoint( Xpos + sizeType*j+count, Ypos+i, charColor ); //修改此函数
              }
            }
        }
        else{
            for(count=0;count<sizeType;count++){
              for(i=0;i<sizeType;i++){
                LCD_SetPoint( Xpos + sizeType*j+count, Ypos+i, bkColor ); //修改此函数
              }
            }
        }
    }
}
/*竖置横排打点函数 ----------------------------------------------------------------*/
```

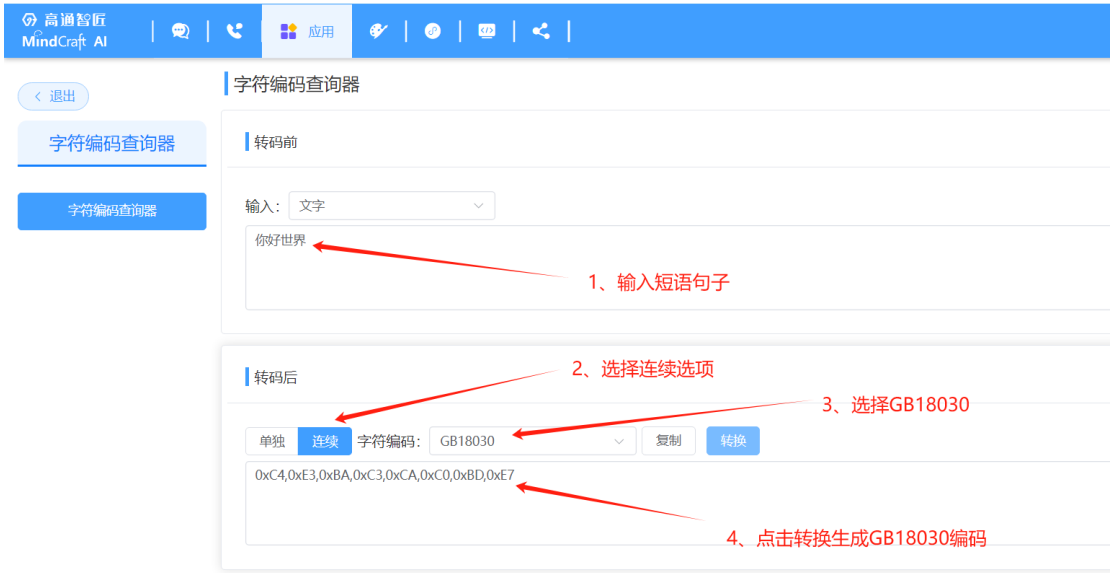
```
/*
  Xpos,Ypos：显示的起点坐标
  data：显示的数据
  charColor：字体颜色
  bkColor：背景颜色
  sizeType：放大倍数，默认为1
*/
void WriteDataY( uint16_t Xpos, uint16_t Ypos, uint8_t data, uint16_t charColor, uint16_t bkColor,u8
sizeType)
{
    uint16_t i,j;
    unsigned char count = 0;
    for( j = 0; j < 8; j++ )
    {
        if( ((data >> (7-j))&0x01)== 0x01 ){
          for(count = 0;count<sizeType;count++){
            for(i = 0;i < sizeType;i++)
            {
                LCD_SetPoint( Xpos+i, Ypos- sizeType*j+count, charColor );  //修改此函数
            }
          }
        }
        else{
          for(count=0;count<sizeType;count++){
            for(i=0;i<sizeType;i++)
            {
              LCD_SetPoint( Xpos+i, Ypos- sizeType*j+count, bkColor );  //修改此函数
            }
          }
        }
    }
}
/******* 竖置横排显示函数 ***********/
/*
  Xpos,Ypos：显示的起点坐标
  W：字体显示的宽
  H：字体显示的高
  charColor：字体颜色
  bkColor：背景颜色
  DZ_Data：显示的字体数据
  sizeType：放大倍数，默认为1
*/
void DisZK_DZ_Y(uint16_t Xpos, uint16_t Ypos, uint16_t W,uint16_t H, uint16_t charColor, uint16_t
bkColor,uint8_t*DZ_Data,u8 sizeType)
```

```
{
  static u16 Vertical,Horizontal;
  u32 bit = 0;
  u8 temp = 0;
  Vertical = Ypos + 7*sizeType;
  Horizontal = Xpos;
  for(bit = 0;bit < ((H+7)/8*W);bit++) //data sizeof (byte)
  {
        if((bit%W==0)&&(bit>0))//W/8 sizeof
        {
            temp += 8 * sizeType;
            Horizontal = Xpos;
            Vertical = Ypos + 7*sizeType + temp;
            //Vertical = Ypos + 7*sizeType + temp;
        }
        else if(bit > 0)
        {
            Horizontal += sizeType;
            Vertical = Ypos+7*sizeType + temp;
        }
        WriteDataY(Horizontal,Vertical,DZ_Data[bit],charColor,bkColor,sizeType);
  }
}
/********º 横置横排显示函数***********/
/*
  Xpos,Ypos：显示的起点坐标
  W：字体显示的宽
  H：字体显示的高
  charColor：字体颜色
  bkColor：背景颜色
  DZ_Data：显示的字体数据
  sizeType：放大倍数，默认为 1
*/
void DisZK_DZ_W(uint16_t Xpos, uint16_t Ypos, uint16_t W,uint16_t H, uint16_t charColor, uint16_t
bkColor,uint8_t*DZ_Data,u8 sizeType)
{
  static u16 Vertical,Horizontal;
  u32 bit=0;
  Vertical=Ypos;
  Horizontal=Xpos;
  for(bit=0;bit<((W+7)/8*H);bit++) //data sizeof (byte)
  {
        if((bit%((W+7)/8)==0)&&(bit>0))//W/8 sizeof
        {
```

```
            Vertical+=sizeType;
            Horizontal=Xpos;
        }
        else if(bit>0)
          Horizontal+=sizeType*8;
        WriteData(Horizontal,Vertical,DZ_Data[bit],charColor,bkColor,sizeType);
    }
}
/*使用示例 */
unsigned char code[13]={6,9,0,1,6,8,7,5,8,2,4,4,2};
unsigned char code1[13]={6,9,3,8,7,0,9,3,0,0,4,4,9};
unsigned char code3[13] = {16,33,45,31,27,26,40};
BAR_codeEAN13(0, 0, code);
BAR_codeEAN13(0, 0, code1);
BAR_codeEAN128(0, 200, code3, 7, 1);
```

4. 10. 字符串显示

首先定义一个数组，用来存放字符的编码。字符的编码查询可以使用高通的MindCraft（高通智匠）应用软件中的“字符编码查询器”中进行编码查询。例如查询“我爱中国”：



大扩号内，从左到右，每两个字节表示一个字符。

```
uint16_t buff[] = {0xCED2,0xB0AE,0xD6D0,0xB9FA};
uint8_t msb = 0, lsb = 0, xpos = 100, ypos = 100;
for(int i = 0; i < sizeof(buff)/sizeof(buff[0]); i++)
```

```
{
    msb = buff[i]>>8;
    lsb = buff[i]&0xff;
    gt_16_GetData(msb, lsb, pBits);
    Display_W(pBits, xpos, ypos, 16, 16);
    xpos += 16;
}
```

注意：显示字符串不推荐在定义的数组里存放文字。例如：

```
unsigned char buff = "我爱中国";
```

因为高通字库的中文使用的是 GB 编码格式，但有可能用户使用的编译器在编译的过程中可能会将这些中文编译成 Unicode 编码或者 UTF-8 编码。最后导致无法正常显示。

4. 11. 字节逆序

如果用户使用的 MCU 存储的方式是大端的存储方式（数据的低位存储在地址的高位），那么需要提前进行一次转换。转换函数可以在“05_屏幕显示资料\字节逆序.c”可以查找到。具体代码如下：

```
void byte_cov(unsigned char *p,unsigned long len)
{
    unsigned long i;
    unsigned char tem;
    for (i=0;i<len;i++)
    {
        tem=p[i];
        p[i]=((tem & 0x01)<<7)|((tem & 0x02)<<5)|((tem & 0x04)<<3)|((tem & 0x08)<<1)  \
            |((tem & 0x10)>>1)|((tem & 0x20)>>3)|((tem & 0x40)>>5)|((tem & 0x80)>>7);
    }
}
```

深圳高通半导体有限公司