目录

[第1章 自动点胶机应用方案 1](#_Toc535172853)

[1.1 产品概述 1](#_Toc535172854)

[1.2 方案介绍 1](#_Toc535172855)

[1.2.1 功能框图 1](#_Toc535172856)

[1.2.2 资源需求 1](#_Toc535172857)

[1.2.3 优势特点 1](#_Toc535172858)

[1.2.4 推荐器件 2](#_Toc535172859)

[1.3 参考设计 2](#_Toc535172860)

[1.3.1 矩阵键盘 2](#_Toc535172861)

[1.3.2 SPI NOR Flash存储器 4](#_Toc535172862)

[1.3.3 带缓冲区的UART接口 7](#_Toc535172863)

[第2章 出租车打印机应用方案 11](#_Toc535172864)

[2.1 产品概述 11](#_Toc535172865)

[2.2 方案介绍 11](#_Toc535172866)

[2.2.1 功能框图 11](#_Toc535172867)

[2.2.2 资源需求 11](#_Toc535172868)

[2.2.3 优势特点 12](#_Toc535172869)

[2.2.4 推荐器件 12](#_Toc535172870)

[2.3 参考设计 12](#_Toc535172871)

[2.3.1 梭式点阵打印机介绍 12](#_Toc535172872)

[2.3.2 梭式点阵打印机组件 14](#_Toc535172873)

[2.3.3 带缓冲区的UART接口 18](#_Toc535172874)

[第3章 读卡应用方案 19](#_Toc535172875)

[3.1 产品概述 19](#_Toc535172876)

[3.2 方案介绍 19](#_Toc535172877)

[3.2.1 功能框图 19](#_Toc535172878)

[3.2.2 资源需求 19](#_Toc535172879)

[3.2.3 优势特点 19](#_Toc535172880)

[3.2.4 推荐器件 20](#_Toc535172881)

[3.3 参考设计 20](#_Toc535172882)

[3.3.1 读卡电路设计 20](#_Toc535172883)

[3.3.2 读卡通信设计 21](#_Toc535172884)

[3.3.3 设备控制类接口函数 22](#_Toc535172885)

[3.3.4 操作接口函数 23](#_Toc535172886)

[3.3.5 密钥和权限控制 25](#_Toc535172887)

[第4章 智能门锁应用方案 28](#_Toc535172888)

[4.1 产品介绍 28](#_Toc535172889)

[4.2 方案介绍 28](#_Toc535172890)

[4.2.1 功能框图 28](#_Toc535172891)

[4.2.2 资源需求 28](#_Toc535172892)

[4.2.3 优势特点 29](#_Toc535172893)

[4.2.4 推荐器件 29](#_Toc535172894)

[4.3 参考设计 29](#_Toc535172895)

[4.3.1 电路设计 29](#_Toc535172896)

[4.3.2 RTC实时时钟 29](#_Toc535172897)

[4.3.3 EEPROM储存器 32](#_Toc535172898)

[4.3.4 52810 BLE组件 32](#_Toc535172899)

[第5章 待续 33](#_Toc535172900)

# 自动点胶机应用方案



图1.1 产品展示

## 产品概述

自动点胶机/焊接/螺丝控制器，主要是给外部的电机机构或者电机平台提供控制信号，有序的控制多个电机，从而实现自动点胶、自动焊接或者自动拧螺丝的功能。

产品实物如图1.1所示，分为主控板卡和手持示教器两部分。驱动控制算法由主控板卡完成，主控板卡通过UART接口与手持示教器进行通信，实时获取手持示教器的运行状态，从而做出相应的控制功能。手持示教器中的主控采用ZLG116N32A设计。

## 方案介绍

### 功能框图

自动点胶机手持示教器功能框图如图1.2所示，自动点胶机手持示教器通过5V输入电压供电，经过LDO转化为3.3V给MCU供电，外置1个LED状态指示灯和1个蜂鸣器，支持48个按键（14路I/O提供6×8矩阵式按键检测），外扩4MByte SPI NOR Flash用于存储控制相关数据。手持示教器通过UART转化为RS232电平串口与主控板卡通讯，以保证更远的通信距离。

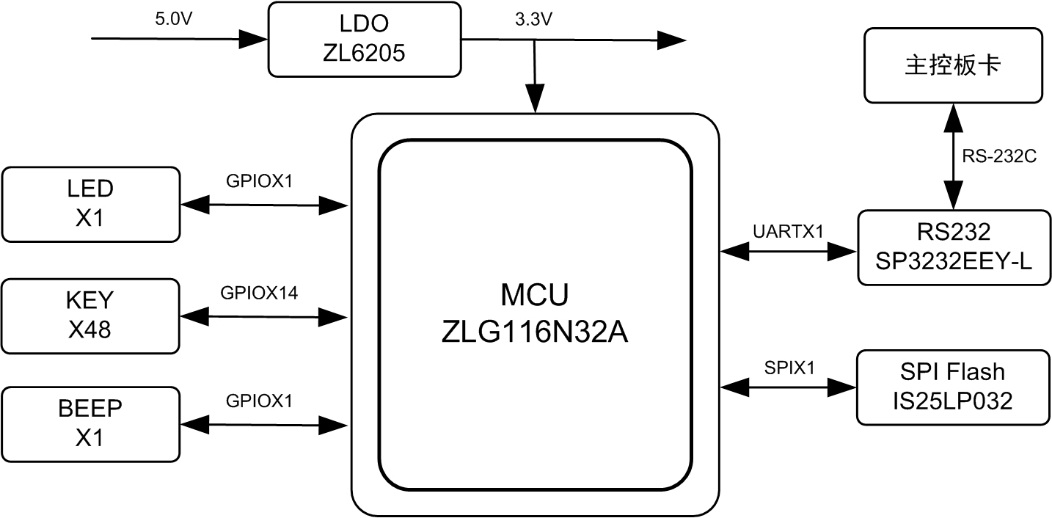


图1.2 功能框图

### 资源需求

* 1路I/O实现LED状态显示；
* 1路I/O实现BEEP发音提示；
* 14路I/O提供6×8矩阵式按键检测实现48个按键输入；
* 1路SPI接口用于外扩SPI NOR Flash；
* 1路UART接口转化为RS232电平串口用于与主控板卡通信。

### 优势特点

对比自动点胶机手持示教器同类产品现用芯片方案，有如下优势：

* MCU资源丰富（64kB Flash / 8K SRAM）、开发更为灵活，超高性价比；
* 完善齐全的DEMO软件和详尽的技术文档，帮助客户快速完成产品开发；
* 提供矩阵键盘、SPI NOR Flash存储器驱动、带缓冲区的UART接口通讯等丰富的组件软件，极大简化客户的开发，更专注于核心应用软件的设计；
* 采用AMetal软件架构，真正实现跨平台移植，帮助客户快速完成产品升级换代。

### 推荐器件

ZLG提供点胶机手持示教器全套BOM解决方案，全套BOM打包，一站式采购，降低整体成本。主控芯片采用ZLG IoT MCU ZLG116N32A设计，详细器件推荐如表1.1所示。

表1.1 推荐器件

|  |  |  |  |
| --- | --- | --- | --- |
| 产品名称 | 型号 | 厂家 | 器件特点 |
| MCU | ZLG116N32A | ZLG | Cortex-M0内核，64kB Flash / 8K SRAM；  运行频率高达48MHz；  支持宽电压输入2.0~5.5V；  多路UART、SPI、I2C等外设接口。 |
| Flash | IS25LP032 | ISSI | 数据存储、低功耗、低电压。 |
| 串口通信 | SP3232EEY-L | EXAR | 支持波特率高达200K；  支持3.3~5.0V电压范围；  两线制异步串行通信。 |
| LDO | ZL6205 | ZLG | 成本低、噪音小、静态电流小。 |

## 参考设计

### 矩阵键盘

本方案采用6×8矩阵按键设计，仅用14个I/O就能实现48个按键检测，如图1.3所示，行线为6，列线为8，行线为输出，列线为输入。

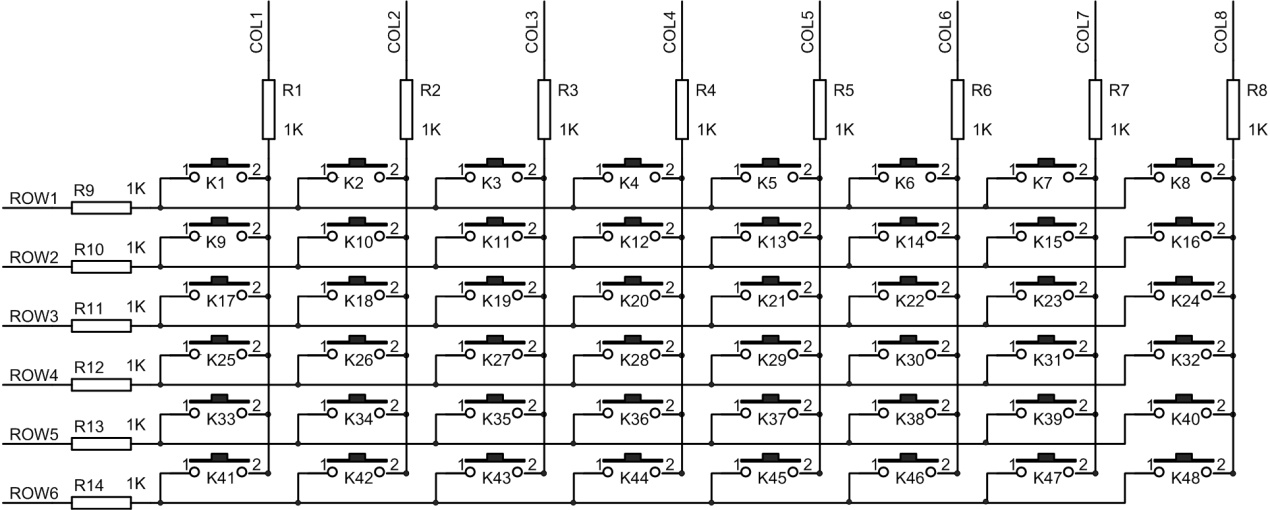


图1.3 矩阵键盘电路

矩阵按键可以提高I/O的使用效率，但是要区分和判断按键动作的方法却比较复杂。每次扫描一行，扫描该行时，对应行线输出为低电平，其余行线输出为高电平，然后读取所有列线的电平，若有列线读到低电平，则表明该行与读到低电平的列对应的交叉点有按键按下。逐列扫描法恰好相反，其列线为输出，行线为输入，但基本原理还是一样的。

AMetal已经集成了矩阵按键组件软件，客户只需配置与矩阵键盘相关的信息即可，操作非常简单。在am\_key\_matrix\_gpio.h文件中，定义了与矩阵键盘相关配置函数结构体，详见程序清单1.1。

程序清单1.1 矩阵键盘配置信息定义

typedef struct am\_key\_matrix\_gpio\_info {

am\_key\_matrix\_base\_info\_t base\_info; // 矩阵键盘基础信息

const int \*p\_pins\_row; // 行线引脚

const int \*p\_pins\_col; // 列线引脚

} am\_key\_matrix\_gpio\_info\_t;

/\*\*

\* \brief 矩阵键盘基础信息

\*/

typedef struct am\_key\_matrix\_base\_info {

int row; // 行数目

int col; // 列数目

const int \*p\_codes; // 各个按键对应的编码，按行的顺序依次对应

am\_bool\_t active\_low; // 按键按下后是否为低电平

uint8\_t scan\_mode; // 扫描方式 （按行扫描或按列扫描）

} am\_key\_matrix\_base\_info\_t;

在am\_key\_matrix\_gpio\_info成员中包含了GPIO驱动矩阵键盘的所有信息，包含了矩阵键盘的基础信息，如矩阵键盘的行数和列数、各按键对应的编码、按键扫描时间及扫描方式等，在am\_key\_matrix\_gpio.c文件中进行赋值，对应信息配置如下：

* \_\_g\_key\_pins\_row指向存放矩阵键盘行线对应引脚号的数组，在此填入行引脚；

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

按键GPIO行线引脚

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static const int \_\_g\_key\_pins\_row[] = {PIOB\_5, PIOB\_4, PIOB\_3,

PIOB\_2, PIOB\_1, PIOB\_0};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

* \_\_g\_key\_pins\_col指向存放矩阵键盘列线对应引脚号的数组，在此填入列引脚；

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

按键GPIO列线引脚

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static const int \_\_g\_key\_pins\_col[] = {PIOA\_15, PIOA\_14, PIOA\_13, PIOA\_12,

PIOA\_11, PIOA\_10, PIOA\_09, PIOA\_08};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

* \_\_g\_key\_codes指向按键编码数组，指定了各按键对应的编码，在此填入按键编码；

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

按编码信息

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static const int \_\_g\_key\_codes[] = {

KEY\_00, KEY\_01,KEY\_02, KEY\_03,KEY\_04, KEY\_05,KEY\_06, KEY\_07,

KEY\_10, KEY\_11,KEY\_12, KEY\_13,KEY\_14, KEY\_15,KEY\_16, KEY\_17,

KEY\_20, KEY\_21,KEY\_22, KEY\_23,KEY\_24, KEY\_25,KEY\_26, KEY\_27,

KEY\_30, KEY\_31,KEY\_32, KEY\_33,KEY\_34, KEY\_35,KEY\_36, KEY\_37,

KEY\_40, KEY\_41,KEY\_42, KEY\_43,KEY\_44, KEY\_45,KEY\_46, KEY\_47,

KEY\_50, KEY\_51,KEY\_52, KEY\_53,KEY\_54, KEY\_55,KEY\_56, KEY\_57,

};

* scan\_interval\_ms指定了按键扫描的时间间隔（单位：毫秒），即每隔该段时间执行一次按键检测，检测是否有按键事件发生（按键按下或按键释放），该值一般设置为5 ms，在结构体中直接赋值即可，如程序清单1.2所示；

程序清单1.2 按键实例化函数

1 int am\_miniport\_key\_inst\_init (void)

2 {

3 static am\_key\_matrix\_gpio\_softimer\_t miniport\_key;

4 static const am\_key\_matrix\_gpio\_softimer\_info\_t miniport\_key\_info = {

5 {

6 {

7 6, // 6行按键

8 8, // 8列按键

9 \_\_g\_key\_codes, // 各按键对应的编码

10 AM\_TRUE, // 按键低电平视为按下

11 AM\_KEY\_MATRIX\_SCAN\_MODE\_COL, // 扫描方式，按列扫描

12 },

13 \_\_g\_key\_pins\_row,

14 \_\_g\_key\_pins\_col,

15 },

16 5, // 扫描时间间隔，5ms

17 };

18 return am\_key\_matrix\_gpio\_softimer\_init(&miniport\_key, &miniport\_key\_info);

19 }

更多矩阵按键组件的使用请参考《面向AMetal框架与接口的编程》第4.3章节。

### SPI NOR Flash存储器

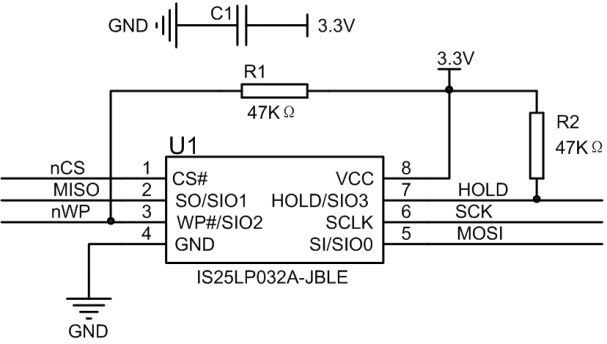


图1.4 SPI NOR Flash电路原理图

本方案需要外扩一个4Mbyte SPI NOR Flash用于存储控制相关数据，采用ISSI的IS25LP032设计。

IS25LP032的通信接口为标准4线SPI接口（支持模式0和模式3），即CS、MOSI、MISO、CLK，详见图1.4。其中，CS（#1）、SO（#2）、SI（#5）、SCLK（#6）分别为SPI的CS、MISO、MOSI和CLK信号引脚。特别地，WP（#3）用于写保护，HOLD（#7）用于暂停数据传输。一般来说，这两个引脚不会使用，可通过上拉电阻上拉至高电平。

AMetal提供了支持常见的IS25LP064、 MX25L8006、MX25L1606……等系列SPI NOR Flash器件的驱动函数，SPI NOR Flash比较特殊，在写入数据前必须确保相应的地址单元已经被擦除，因此除初始化、读写函数外，还有一个擦除函数，其接口函数详见表1.2。

表1.2 IS25xx接口函数

|  |  |
| --- | --- |
| 函数原型 | 功能简介 |
| am\_is25xx\_handle\_t is25xx\_handle = **am\_is25xx\_inst\_init();** | 实例初始化 |
| int **am\_is25xx\_erase**(  am\_is25xx\_handle\_t handle,  uint32\_t addr,  uint32\_t len); | 擦除 |
| int **am\_is25xx\_write**(  am\_is25xx\_handle\_t handle,  uint32\_t addr,  uint8\_t \*p\_buf,  uint32\_t len); | 写入数据 |
| int **am\_is25xx\_read**(  am\_is25xx\_handle\_t handle,  uint32\_t addr,  uint8\_t \*p\_buf,  uint32\_t len); | 读取数据 |

各API的返回值含义都是相同的：AM\_OK表示成功，负值表示失败，失败原因可根据

具体的值查看am\_errno.h文件中相对应的宏定义。正值的含义由各API自行定义，无特殊说明时，表明不会返回正值。

1. 擦除

擦除就是将数据全部重置为0xFF，即所有存储单元的位设置为1。擦除操作并不能直接擦除某个单一地址单元，擦除的最小单元为扇区，即每次只能擦除单个或多个扇区。擦除一段地址空间的函数原型如程序清单1.3所示。

程序清单1.3 擦除函数

int am\_is25xx\_erase(am\_is25xx\_handle\_thandle,uint32\_taddr, uint32\_t len);

其中，handle为IS25LP032的实例句柄，addr为待擦除区域的首地址，由于擦除的最小单元为扇区，因此该地址必须为某扇区的起始地址0x000000（0）、0x001000（4096）、0x002000（2×4096）……同时，擦除长度必须为扇区大小的整数倍。

如果返回AM\_OK，说明擦除成功，反之失败。假定需要从0x001000地址开始，连续擦除2个扇区，范例程序详见程序清单1.4。

程序清单1.4 擦除范例程序

1 am\_is25xx\_erase(is25xx\_handle, 0x001000, 2 \* 4096); // 擦除两个扇区

0x001000 ~ 0x3FFF空间被擦除了，即可向该段地址空间内写入数据。

1. 写入数据

在写入数据前，需确保写入地址已被擦除。即将需要变为0的位清0，但写入操作无法将0变为1。比如，写入数据0x55就是将bit1、bit3、bit5、bit7清0，其余位的值保持不变。若存储的数据已经是0x55，再写入0xAA（写入0xAA实际上就是将bit0、bit2、bit4、bit6清0，其余位不变），则最终存储的数据将变为0x00，而不是后面再写入的0xAA。因此为了保证正常写入数据，写入数据前必须确保相应的地址段已经被擦除了。

从指定的起始地址开始写入一段数据的函数原型如程序清单1.5所示。

程序清单1.5 写入函数原型

int am\_is25xx\_write(

am\_is25xx\_handle\_t handle, // is25xx实例句柄

uint32\_t addr, // 写入数据的起始地址

uint8\_t \*p\_buf, // 写入数据缓冲区

uint32\_t len); // 写入数据的长度

如果返回AM\_OK，说明写入数据成功，反之失败。假定从0x001000地址开始，连续写入128字节数据，范例程序详见程序清单1.6。

程序清单1.6 写入数据范例程序

1 uint8\_t buf[128];

2 int i;

3 for (i = 0 ; i < 128; i++) buf[i] = i; // 装载数据

4 am\_is25xx\_erase(is25xx\_handle, 0x001000, 4096); // 擦除一个扇区

5 am\_is25xx\_write(is25xx\_handle, 0x001000, buf, 128); // 写入128字节数据

虽然只写入了128字节数据，但由于擦除的最小单元为扇区，因此擦除了4096字节（一个扇区）。已经擦除的区域后续可以直接写入数据，而不必再次擦除，比如，紧接着写入128字节数据后的地址，再写入128字节数据，详见程序清单1.7。

程序清单1.7 写入数据范例程序

1 am\_is25xx\_write(is25xx\_handle, 0x001000 + 128, buf, 128); // 再写入128字节数据

若需要再次从0x001000地址连续写入128字节数据，由于之前已经写入过数据，因此必须重新擦除后方可再次写入。

1. 读取数据

从指定的起始地址开始读取一段数据的函数原型如程序清单1.8所示。

程序清单1.8 读取函数

int am\_is25xx\_read(

am\_is25xx\_handle\_t handle, // is25xx实例句柄

uint32\_t addr, // 读取数据的起始地址

uint8\_t \*p\_buf, // 读取数据的缓冲区

uint32\_t len); // 读取数据的长度

如果返回值为AM\_OK，则说明读取成功，反之失败。假定从0x001000地址开始，连续读取128字节数据，详见程序清单1.9。

程序清单1.9 读取数据范例程序

1 uint8\_tdata[128];

2 am\_is25xx\_read(is25xx\_handle, 0x001000, buf, 128); // 读取128字节数据

由于函数中的参数为IS25LP032的实例handle，与IS25LP032器件具有依赖关系，因此无法实现跨平台调用。对此，AMetal将其进行抽象为一个读写IS25LP032的MTD（Memory Technology Device），使之与器件无关，实现跨平台调用。

另外，此前的接口需要在每次写入数据前，确保相应的存储空间已经被擦除，则势必会给编程带来很大的麻烦。与此同时，由于IS25LP032的某一地址段擦除次数超过10万次的上限，则在相应段地址空间存储数据将不再可靠。假设将用户数据存放到0x001000~0x001FFF连续的4K地址中，则每次更新这些数据都要重新擦除该地址段。而其它存储空间完全没有使用过，IS25LP032的使用寿命大打折扣。为了延长flash的使用寿命， AMetal提供了FTL（Flash Translation Layer）通用接口供用户使用，在实际写入时，将数据写入到擦除次数最少的区域。

关于MTD和FTL详细说明和使用请参考《面向AMetal框架与接口的编程》第5.2章节。

### 带缓冲区的UART接口

手持示教器通过UART转化为RS232电平串口与主控板通信。由于查询模式会阻塞整个应用，因此在实际应用中几乎都使用中断模式。但在中断模式下，UART每收到一个数据都会调用回调函数，如果将数据的处理放在回调函数中，很有可能因当前数据的处理还未结束而丢失下一个数据。

基于此，AMetal提供了一组带缓冲区的UART通用接口，详见表1.3，其实现是在UART中断接收与应用程序之间，增加一个接收缓冲区。当串口收到数据时，将数据存放在缓冲区中，应用程序直接访问缓冲区即可。

对于UART发送，虽然不存在丢失数据的问题，但为了便于开发应用程序，避免在UART中断模式下的回调函数接口中一次发送单个数据，同样提供了带缓冲区的UART发送函数。当应用程序发送数据时，将发送数据存放在发送缓冲区中，串口在发送空闲时提取发送缓冲区中的数据进行发送。

表1.3 带缓冲区的UART通用接口函数（am\_uart\_rngbuf.h）

|  |  |
| --- | --- |
| 函数原型 | 功能简介 |
| am\_uart\_rngbuf\_handle\_t am\_uart\_rngbuf\_init(  am\_uart\_rngbuf\_dev\_t \*p\_dev,  am\_uart\_handle\_t handle,  uint8\_t \*p\_rxbuf,  uint32\_t rxbuf\_size,  uint8\_t \*p\_txbuf,  uint32\_t txbuf\_size); | 初始化 |
| int am\_uart\_rngbuf\_send(  am\_uart\_rngbuf\_handle\_t handle,  const uint8\_t \*p\_txbuf,  uint32\_t nbytes); | 发送数据 |
| int am\_uart\_rngbuf\_receive(  am\_uart\_rngbuf\_handle\_t handle,  uint8\_t \*p\_rxbuf,  uint32\_t nbytes); | 接收数据 |
| int am\_uart\_rngbuf\_ioctl(  am\_uart\_rngbuf\_handle\_t handle,  int request,  void \*p\_arg); | 控制函数 |

#### 初始化

指定关联的串口外设（相应串口的实例句柄handle），以及用于发送和接收的数据缓冲区，初始化一个带缓冲区的串口实例，其函数原型如程序清单1.10所示。

程序清单1.10 串口初始化函数原型

am\_uart\_rngbuf\_handle\_t am\_uart\_rngbuf\_init(

am\_uart\_rngbuf\_dev\_t \*p\_dev, // 带缓冲区的UART设备

am\_uart\_handle\_t handle, // UART实例句柄handle

char \*p\_rxbuf, // 接收数据缓冲区

uint32\_t rxbuf\_size, // 接收数据缓冲区的大小

char \*p\_txbuf, // 发送数据缓冲区

uint32\_t txbuf\_size); // 发送数据缓冲区的大小

其中， p\_dev为指向am\_uart\_rngbuf\_dev\_t类型的带缓冲区的串口实例指针，在使用时，只需要定义一个am\_uart\_rngbuf\_dev\_t类型（am\_uart\_rngbuf.h）的实例即可：

am\_uart\_rngbuf\_dev\_t g\_uart0\_rngbuf\_dev;

其中，g\_uart0\_rngbuf\_dev为用户自定义的实例，其地址作为p\_dev的实参传递。handle为 UART实例句柄，用于指定该带缓冲区的串口实际关联的串口。p\_rxbuf和rxbuf\_size用于指定接收缓冲区及其大小，p\_txbuf和txbuf\_size用于指定发送缓冲区及其大小。

函数的返回值为带缓冲区串口的实例句柄，可用作其它通用接口函数中handle参数的实参。其类型am\_uart\_rngbuf\_handle\_t（am\_uart\_rngbuf.h）定义如下：

typedef struct am\_uart\_rngbuf\_dev \* am\_uart\_rngbuf\_handle\_t;

如果返回值为NULL，表明初始化失败，初始化函数使用范例详见程序清单1.11。

程序清单1.11 am\_uart\_rngbuf\_init()范例程序

1 static uint8\_t uart\_rxbuf[128]; // 定义用于接收数据的缓冲区，大小为128

2 static uint8\_t uart\_txbuf[128]; // 定义用于发送数据的缓冲区，大小为128

3 am\_uart\_rngbuf\_dev\_t g\_uart\_rngbuf\_dev;

4 am\_uart\_rngbuf\_handle\_t g\_uart\_rngbuf\_handle;

5

6 g\_uart\_rngbuf\_handle = am\_uart\_rngbuf\_init(

7 &g\_uart\_rngbuf\_dev,

8 uart\_handle, // UART实例句柄handle

9 uart\_rxbuf, // 用于接收数据的缓冲区

10 128, // 接收缓冲区大小为128

11 uart\_txbuf, // 用于发送数据的缓冲区

12 128); // 发送缓冲区大小为128

虽然程序将缓冲区的大小设置为128，但实际上缓冲区的大小应根据实际情况确定。若接收数据的缓冲区过小，则可能在接收缓冲区满后又接收新的数据发生溢出而丢失数据。若发送缓冲区过大，则在发送数据时很可能因为发送缓冲区已满需要等待，直至发送缓冲区有空闲空间而造成等待过程。

#### 发送数据

发送数据就是将数据存放到am\_uart\_rngbuf\_init()指定的发送缓冲区中，串口可以进行数据发送时（发送空闲），从发送缓冲区中提取需要发送的数据进行发送。其函数原型如程序清单1.12所示。

程序清单1.12 发送函数原型

int am\_uart\_rngbuf\_send(

am\_uart\_rngbuf\_handle\_t handle, // 带缓冲区的串口实例句柄

const uint8\_t \*p\_txbuf, // 应用程序发送数据缓冲区

uint32\_t nbytes); // 发送数据的个数

该函数将数据成功存放到发送缓冲区后返回，返回值为成功写入的数据个数。比如，发送一个字符串“Hello World!”，详见程序清单1.13。

程序清单1.13 am\_uart\_rngbuf\_send()范例程序

1 uint8\_t str[] = "Hello World!";

2 am\_uart\_rngbuf\_send(g\_uart0\_rngbuf\_handle, str, sizeof(str)); // 发送字符串"Hello World!"

注意，当该函数返回时，数据仅仅只是存放到了发送缓冲区中，并不代表已经成功地将数据发送出去了。

#### 接收数据

接收数据就是从am\_uart\_rngbuf\_init()指定的接收缓冲区中提取接收到的数据，其函数原型如程序清单1.14所示。

程序清单1.14 接收函数

int am\_uart\_rngbuf\_receive(

am\_uart\_rngbuf\_handle\_t handle, // 带缓冲区的串口实例句柄

uint8\_t \*p\_rxbuf, // 应用程序接收数据缓冲区

uint32\_t nbytes); // 接收数据的个数

该函数返回值为成功读取数据的个数，使用范例详见程序清单1.15。

程序清单1.15 am\_uart\_rngbuf\_receive()范例程序

1 uint8\_t rxbuf[10];

2 am\_uart\_rngbuf\_receive(g\_uart0\_rngbuf\_handle, rxbuf,10);// 接收10个数据

#### 控制函数

与UART控制函数类似，用于完成一些基本的控制操作。其函数原型如程序清单1.16所示。

程序清单1.16 控制函数

int am\_uart\_rngbuf\_ioctl(

am\_uart\_rngbuf\_handle\_t handle, // 带缓冲区的串口实例句柄

int request, // 控制命令

void \*p\_arg); // 对应命令的参数

“控制命令”和“对应命令的参数”，与UART控制函数am\_uart\_ioctl()的含义类似。带缓冲区的UART可以看作是在UART基础上的一个扩展，因此绝大部分UART控制函数的命令均可直接使用。

更多串口带缓冲组件的使用请参考《面向AMetal框架与接口的编程》第4.8章节。

# 出租车打印机应用方案



图2.1 产品展示

## 产品概述

出租车打印机如图2.1所示，出租车打印机属于梭式点阵打印机，是利用机械和电路驱动打印针撞击色带和打印介质，进而打印出点阵，通过打印的字符或组成的图形来完成打印，出租车打印机具有结构简单、技术成熟、性价比高、消耗费用低等特点。

主机通过UART和MCU通信，MCU控制打印机完成信息打印。本出租车打印机方案的主控采用ZLG217P64A设计。

## 方案介绍

### 功能框图

出租车打印机功能框图如图2.2所示，打印机机头使用5V供电，通过LDO将5V电压转化为3.3V给系统供电，出租车打印机外置2个LED状态指示灯，1路I/O驱动蜂鸣器作为打印提示。UART转化为RS232电平串口与主机通讯，获取打印数据。5路I/O通过逻辑器控制打印机机头工作，其中1路用于控制电机走纸，4路用于控制4个打印针工作。2路I/O连接打印机机头，其中1路检测打印复位信号，1路检测打印机脉冲信号。

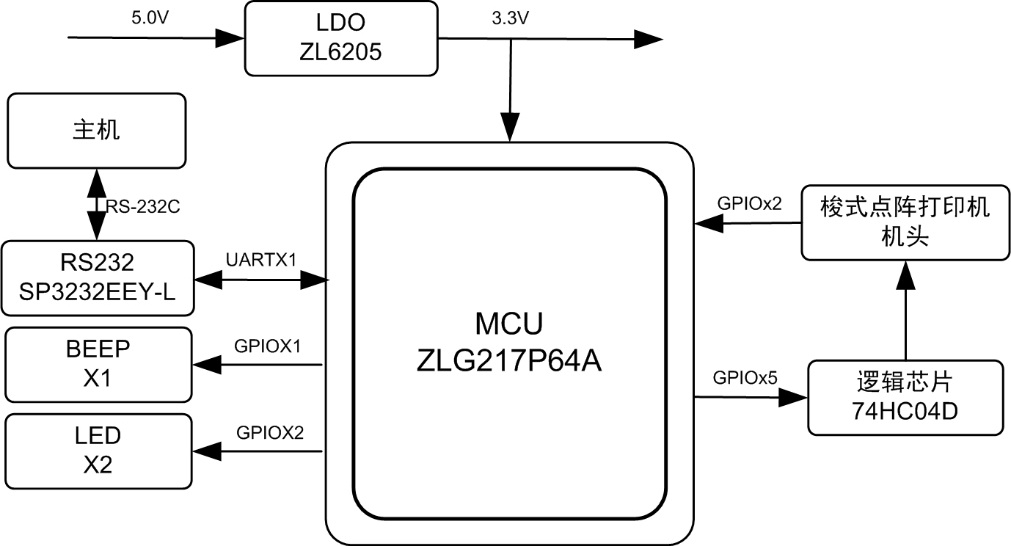


图2.2 功能框图

### 资源需求

* 2路I/O实现LED状态显示；
* 1路I/O用于驱动蜂鸣器；
* 5路I/O通过逻辑器控制打印机机头，其中4路控制打印针，1路控制电机；
* 2路I/O连接打印机机头，其中1路连接用于检测复位信号，1路检测打印机脉冲信号；
* 1路UART接口转化为RS232电平串口用于与主机通信。

### 优势特点

对比出租车打印机同类产品现用芯片方案，有如下优势：

* MCU资源丰富（128kB Flash / 20K SRAM）、开发更为灵活，超高性价比；
* 完善齐全的DEMO软件和详尽的技术文档，帮助客户快速完成产品开发；
* 提供梭式点阵打印机驱动组件、带缓冲区的UART接口通讯等丰富的组件软件，极大简化客户的开发，更专注于核心应用软件的设计；
* 采用AMetal软件架构，真正实现跨平台移植，帮助客户快速完成产品升级换代。

### 推荐器件

ZLG提供出租车打印机全套BOM解决方案，全套BOM打包，一站式采购，降低整体成本。主控芯片采用ZLG IoT MCU ZLG217P64A设计，详细器件推荐如表2.1示。

表2.1 推荐器件

|  |  |  |  |
| --- | --- | --- | --- |
| 产品名称 | 型号 | 厂家 | 器件特点 |
| MCU | ZLG217P64A | ZLG | Cortex-M3内核，20K SRAM/128kB Flash；  单指令周期32位硬件乘法器；  运行频率高达96MHz；  支持宽电压输入2.0-5.5V；  多路SPI I2C等外设接口。 |
| 逻辑器 | 74HC04D | NXP | 成本低、支持多达6路独立通道。 |
| 串口通信 | SP3232EEY-L | EXAR | 支持波特率高达200K；  支持3.3~5.0V电压范围；  两线制异步串行通信。 |
| LDO | ZL6205 | ZLG | 成本低、噪音小、静态电流小。 |

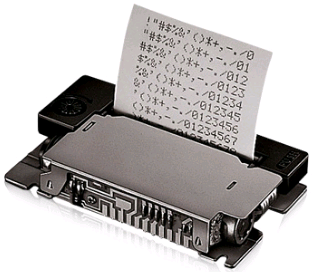


图2.3 M-150II梭式点阵打印机

## 参考设计

### 梭式点阵打印机介绍

1. M-150II打印机

本方案采用爱普生微型打印机中的M-150II梭式点阵打印机，打印机体积小巧且高度可靠，其重量约为60g，但性能依然很强。因为体积小巧，所以M-150II满足各种小型设备的打印需求，包括从手持终端到笔记本电脑以及小型测量仪器等。由于运行所需电量较小，这款打印机可以选择采用电池供电，产品如图2.3所示。

1. 工作原理

M-150II梭式点阵打印机，由打印头、电机、定时和复位检测器等组成。其中打印头由4个水平放置的打印电磁阀（A.B.C.D）组成。打印头在打印状态下从左侧向右移动，移动量为每个打印电磁阀24个点。当打印头移动时，通过逐个驱动打印电磁阀打点形成一条点线，每个点线的总点数为96个点（24点×4个打印电磁阀），当打印头从右侧返回左侧时，纸张自动送入0.35mm（一个间距）重复此点线的打印，如图2.4所示。

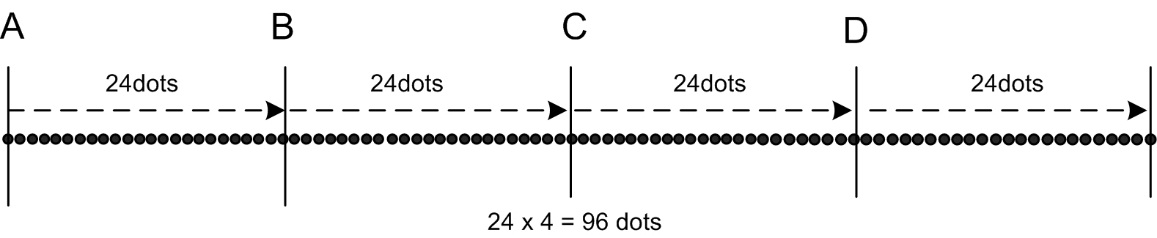


图2.4 一条点线

电机采用直流有刷电机，当打印机处于待机状态（即非打印状态）时，电机处于暂停状态。定时检测器(Timing Detector)是与电机直接连接的转速计发生器。检测器每个点线产生168个输出信号，其中96个输出信号对应打印头的点位置，72个输出信号对应打印头返回。这些输出信号按脉冲排列的波形，用作定时脉冲，如图2.5所示。

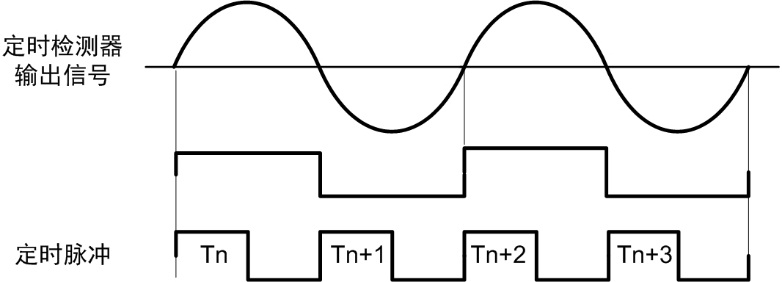


图2.5 定时脉冲

复位检测器(Reset Detector)具有簧片开关，每条点线都会产生。在每次打印周期中点位置的标准位置，复位检测器输出的信号用作复位，时序图如图2.6所示。

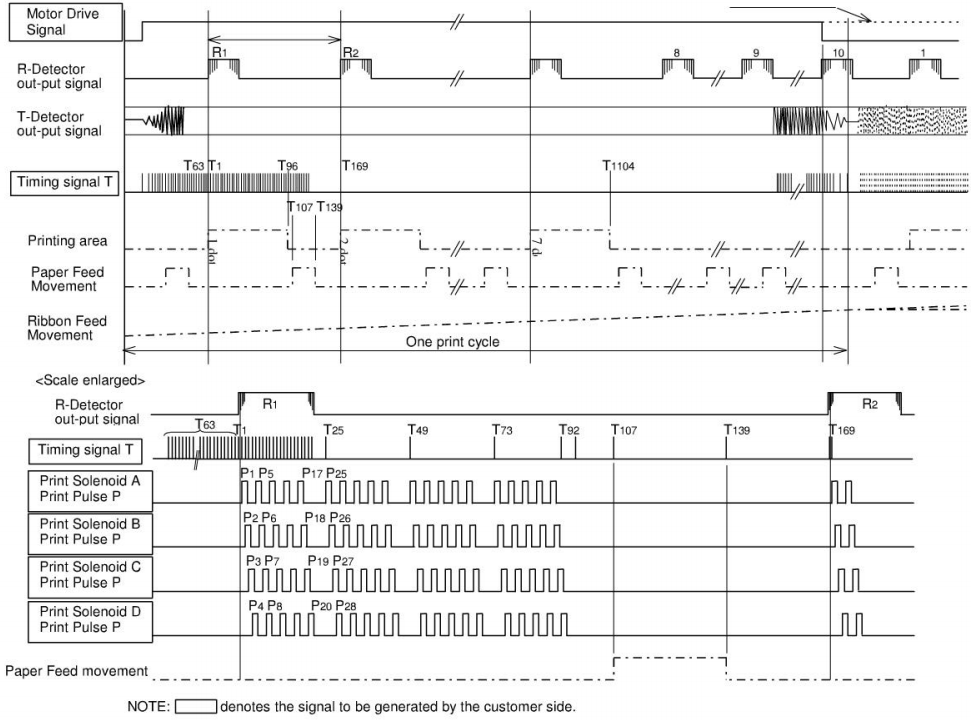


图2.6 时序图

### 梭式点阵打印机组件

AMetal已经集成了M-150II打印机组件软件，客户只需配置与打印机相关的信息即可，操作非常简单。在am\_printer\_m150.h文件中，定义了与M-150II打印机设备相关信息结构体，详见程序清单 2.1。

程序清单 2.1 打印机配置信息定义

1 typedef struct am\_printer\_m150\_info {

2 int mot\_pin; // 电机控制引脚

3 int tir\_pin; // 时钟脉冲输入引脚

4 int rst\_pin; // 复位信号检测引脚

5 int print\_a\_pin; // 电磁阀A控制引脚

6 int print\_b\_pin; // 电磁阀B控制引脚

7 int print\_c\_pin; // 电磁阀C控制引脚

8 int print\_d\_pin; // 电磁阀D控制引脚

9 uint8\_t \*p\_addr\_buf; // 指向字库数组，存储字符相对地址

10 } am\_printer\_m150\_info\_t;

在am\_printer\_m150\_info结构体成员中包含了驱动M-150II打印机的所有信息，包括电机控制引脚、时钟脉冲输入引脚、复位信号检测引脚、ABCD电磁阀控制引脚等，在am\_hwconf\_printer\_m150.c中进行赋值，对应信息配置示例如程序清单2.2所示。

程序清单2.2 打印机配置信息示例

1 static const am\_printer\_m150\_info\_t \_\_g\_printer\_m150\_info = {

2 PIOB\_15, // 电机控制引脚

3 PIOB\_14, // 时钟脉冲输入引脚

4 PIOB\_13, // 复位信号检测引脚

5 PIOC\_6, // 电磁阀A控制引脚

6 PIOC\_7, // 电磁阀B控制引脚

7 PIOC\_8, // 电磁阀C控制引脚

8 PIOC\_9, // 电磁阀D控制引脚

9 \_\_g\_add\_buf, // 指向字符地址缓存

10 };

配置ZLG217P64A的PIOB\_15引脚控制电机，PIOB\_14引脚输入时钟脉冲，PIOB\_13引脚检测复位信号，PIOC\_6.7.8.9引脚分别控制电磁阀A.B.C.D。

AMetal提供了M-150II打印机的驱动函数，其接口函数详见表2.2。

表2.2 M-150II接口函数

|  |  |
| --- | --- |
| 函数原型 | 功能简介 |
| am\_printer \_m150\_handle\_t handle =  am\_printer\_m150\_inst\_init(); | 实例初始化 |
| void am\_printer\_m150\_print\_line\_char(  am\_m150\_handle\_t handle,  unsigned char \*data\_buf); | 打印一行字符 |
| void am\_printer\_m150\_print\_line\_ chinese (  am\_m150\_handle\_t handle,  unsigned char \*data\_buf); | 打印一行汉字 |

其中handle为服务句柄，即为初始化M-150II打印机获取的句柄，databuf为指向字符或汉字数据相对地址的指针。

1. 打印一行字符

字符是由5x7的点阵组成，将一个打印电磁阀可打印的24个点分成四个相等的部分，并且一个部分中的6个点用作一列，即用于打印的5个点和用于列空间的一个点。因此一个点线由96个点形成，其被分成16个部分并且可以通过在送纸方向上重复7次获得5×7点阵的字符，如图2.7所示。

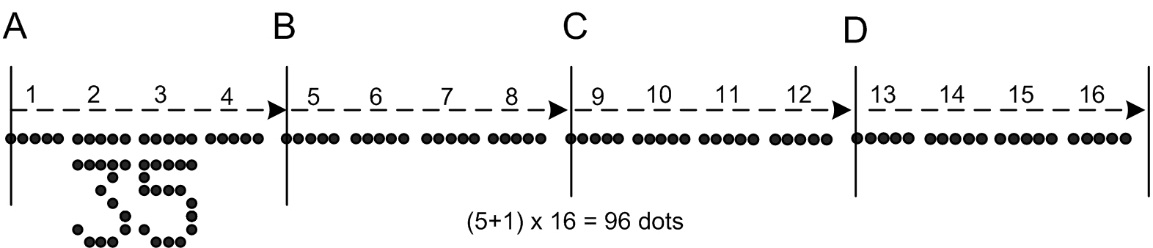


图2.7 5x7点阵字符-35

以打印一行16个字符中的第2、3个字符‘35’为例：

* 字符‘3’在字库中的编码为：0x07, 0xef, 0xdf, 0xcf, 0xf7, 0x77, 0x8f,
* 字符‘5’在字库中的编码为：0x07, 0x7f, 0x0f, 0xf7, 0xf7, 0x77, 0x8f,

字符转换过程详见表2.3。

表2.3 字库字符转换示例

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字符‘3’ | 二进制 | 0的位 | 字符‘5’ | 二进制 | 0的位 |
| 0x07  0xef  0xdf  0xcf  0xf7  0x77  0x8f | 00000111‬  11101111‬  11011111‬  11101111‬  11110111‬  01110111  10001111 | 00000\_\_\_‬  \_\_\_0\_\_\_\_‬  \_\_0\_\_\_\_\_  \_\_\_0\_\_\_\_  \_\_\_\_0\_\_\_  0\_\_\_0\_\_\_  \_000\_\_\_\_ | 0x07  0x7f  0x0f  0xf7  0xf7  0x77  0x8f | 00000111  01111111  00001111  11110111  11110111  01110111  10001111 | 00000\_\_\_  0\_\_\_\_\_\_\_  0000\_\_\_\_  \_\_\_\_0\_\_\_  \_\_\_\_0\_\_\_  0\_\_\_0\_\_\_  \_000\_\_\_\_ |

打印一行字符的函数定义如程序清单2.3所示。

程序清单2.3 打印字符函数定义

1 void am\_printer\_m150\_print\_line\_char(am\_m150\_handle\_t handle,

2 unsigned char\* data\_buf)

3 {

4 unsigned char i;

5 am\_m150\_dev\_t \*p\_m150\_dev = handle;

6 // 连接时钟脉冲引脚中断服务函数

7 am\_gpio\_trigger\_connect(p\_m150\_dev->p\_info->tir\_pin,

8 \_\_gpio\_isr\_handler,

9 p\_m150\_dev);

10 // 配置时钟脉冲引脚中断触发方式

11 am\_gpio\_trigger\_cfg(p\_m150\_dev->p\_info->tir\_pin,

12 AM\_GPIO\_TRIGGER\_BOTH\_EDGES);

13 // 寻找第一行地址

14 \_\_addr\_search(0, p\_m150\_dev->p\_info->p\_addr\_buf, data\_buf);

15 am\_gpio\_set(p\_m150\_dev->p\_info->mot\_pin, 0); // 打开电机

16 am\_udelay (10);

17 for (i = 1; i <= 7; i++) { // 循环打印7行

18 \_\_print\_point\_line(p\_m150\_dev); // 打印一行墨点

19 if(i < 7){

20 // 寻找下一行地址

21 \_\_addr\_search(i, p\_m150\_dev->p\_info->p\_addr\_buf, data\_buf);

22 }

23 }

24 \_\_print\_end(p\_m150\_dev); //打印结束

25 }

1. 打印一行汉字

汉字是由11x12的点阵组成，将一个打印电磁阀可打印的24个点分成2个相等的部分，并且一个部分中的12个点用作一列，即用于打印的11个点和用于列空间的一个点。因此一个点线由96个点形成，其被分成8个部分并且可以通过在送纸方向上重复12次获得11×12点阵的汉字，如图2.8所示。

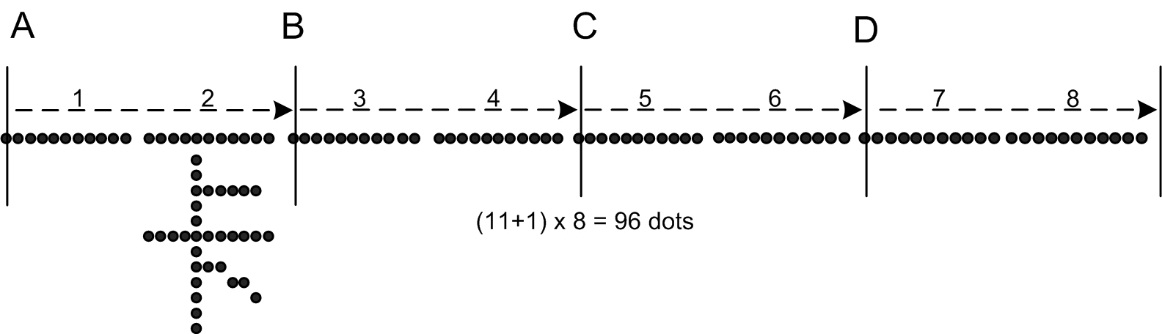


图2.8 11x12点阵汉字-卡

以打印一行8个汉字中的第2个汉字‘卡’为例，汉字‘卡’在字库中的编码为：

0x08,0x08,0x0f,0x08,0x08,0xff,0x08,0x0e,0x09,0x08,0x08,0x08,

0x00,0x00,0xc0,0x00,0x00,0xe0,0x00,0x00,0x80,0x40,0x00,0x00,

汉字转换过程详见表2.4。

表2.4 字库汉字转换示例

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 汉字‘卡’ | | 二进制 | | 1的位 | |
| 0x08,  0x08,  0x0f,  0x08,  0x08,  0xff,  0x08,  0x0e,  0x09,  0x08,  0x08,  0x08, | 0x00,  0x00,  0xc0,  0x00,  0x00,  0xe0,  0x00,  0x00,  0x80,  0x40,  0x00,  0x00, | 00001000  00001000  00001111  00001000  00001000  11111111  00001000  00001110  00001001  00001000  00001000  00001000 | 00000000  00000000  11000000  00000000  00000000  11100000  00000000  00000000  10000000  01000000  00000000  00000000 | \_\_\_\_1\_\_\_  \_\_\_\_1\_\_\_  \_\_\_\_1111  \_\_\_\_1\_\_\_  \_\_\_\_1\_\_\_  11111111  \_\_\_\_1\_\_\_  \_\_\_\_111\_  \_\_\_\_1\_\_1  \_\_\_\_1\_\_\_  \_\_\_\_1\_\_\_  \_\_\_\_1\_\_\_ | \_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_  11\_\_\_\_\_\_  \_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_  111\_\_\_\_\_  \_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_  1\_\_\_\_\_\_\_  \_1\_\_\_\_\_\_  \_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_ |

打印一行汉字的函数定义如程序清单2.4所示。

程序清单2.4 打印汉字函数定义

1 void am\_printer\_m150\_print\_line\_chinese(am\_m150\_handle\_t handle,

2 unsigned char\* data\_buf)

3 {

4 unsigned char i;

5 am\_m150\_dev\_t \*p\_m150\_dev = handle;

6 // 连接时钟脉冲引脚中断服务函数

7 am\_gpio\_trigger\_connect(p\_m150\_dev->p\_info->tir\_pin,

8 \_\_gpio\_isr\_handler,

9 p\_m150\_dev);

10 // 配置时钟脉冲引脚中断触发方式

11 am\_gpio\_trigger\_cfg(p\_m150\_dev->p\_info->tir\_pin,

12 AM\_GPIO\_TRIGGER\_BOTH\_EDGES);

13 // 寻找第一行的地址

14 \_\_addr\_search\_chinese(0, p\_m150\_dev->p\_info->p\_addr\_buf, data\_buf);

15 am\_gpio\_set(p\_m150\_dev->p\_info->mot\_pin, 0); // 打开电机

16 am\_udelay (10);

17 for (i = 1; i <= 12; i++) { // 循环打印12行

18 \_\_print\_point\_line(p\_m150\_dev); // 打印一行墨点

19 if(i < 12){

20 // 寻找下一行汉字的地址

21 \_\_addr\_search\_chinese(i, p\_m150\_dev->p\_info->p\_addr\_buf, data\_buf);

22 }

23 }

24 \_\_print\_end(p\_m150\_dev); // 打印结束

25 }

### 带缓冲区的UART接口

出租车打印机通过UART转化为RS232电平串口与主机通信。由于查询模式会阻塞整个应用，因此在实际应用中几乎都使用中断模式。但在中断模式下，UART每收到一个数据都会调用回调函数，如果将数据的处理放在回调函数中，很有可能因当前数据的处理还未结束而丢失下一个数据。带缓冲的UART接口组件使用请参考第1.3.3章节。

# 读卡应用方案



图3.1 产品展示

## 产品概述

读卡方案，产品实物如图3.1所示。 采用FM17520作为读卡芯片，通过SPI与主控芯片通信，读卡距离达5cm，支持多种卡片。

采用非接触式读卡安全性较高，使用方便。非接触式读卡是目前主流的读卡方案，应用广泛，支持多种应用场合，可以存储用户信息，离线管理数据和上传等。读卡方案中的主控采用ZLG116N32A设计。

## 方案介绍

### 功能框图

读卡功能框图如图3.2所示，读卡系统采用DC12V供电，经过LDO转化为3.3V和5.0V给MCU和外设器件供电，外置两个LED状态指示灯。通过SPI和读卡芯片通信，从而对卡片进行读写。

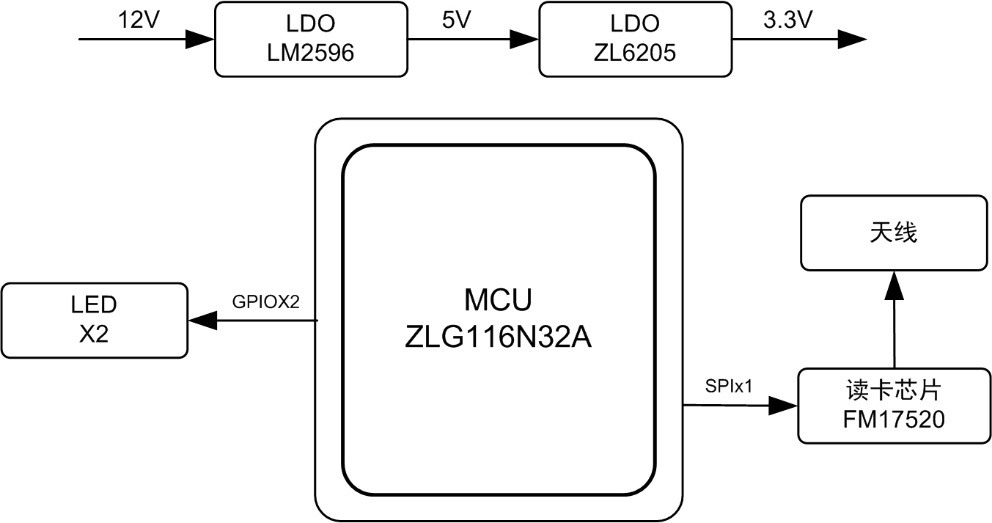


图3.2 功能框图

### 资源需求

* 2路I/O实现LED状态显示；
* 1路SPI接口用于和FM17520通信。

### 优势特点

对比读卡系统同类产品现用芯片方案，有如下优势：

* MCU资源丰富（64kB Flash / 8K SRAM）、开发更为灵活，超高性价比；
* 完善齐全的DEMO软件和详尽的技术文档，帮助客户快速完成产品开发；
* 提供SPI驱动通讯组件等丰富的组件软件，极大简化客户的开发，更专注于核心应用软件的设计；
* 采用AMetal软件架构，真正实现跨平台移植，帮助客户快速完成产品升级换代。

### 推荐器件

ZLG提供读卡系统全套BOM解决方案，全套BOM打包，一站式采购，降低整体成本。主控芯片采用ZLG IoT MCU ZLG116N32A设计，详细器件推荐如表3.1所示。

表3.1 推荐器件

|  |  |  |  |
| --- | --- | --- | --- |
| 产品名称 | 型号 | 厂家 | 器件特点 |
| MCU | ZLG116N32A | ZLG | Cortex-M0内核，64kB Flash / 8K SRAM；  运行频率高达48MHz；  支持宽电压输入2.0~5.5V；  多路UART、SPI、I2C等外设接口。 |
| 读卡芯片 | FM17520 | 复旦微 | 非接触式读卡、低电压、长距离。 |
| LDO | ZL6205 | ZLG | 成本低、噪音小、静态电流小。 |
| LDO | LM2596 | TI | 成本低、噪音小、静态电流小。 |

## 参考设计

### 读卡电路设计

读卡芯片FM17520内部集成了强大的内部电路，外部电路设计通常比较简单，主要组成为：供电电路、通信接口电路、天线电路和振荡电路。其中天线电路设计由为重要，读卡器天线电路主要分成四个部分：EMC滤波、匹配电路、天线和接收电路，如图3.3所示。

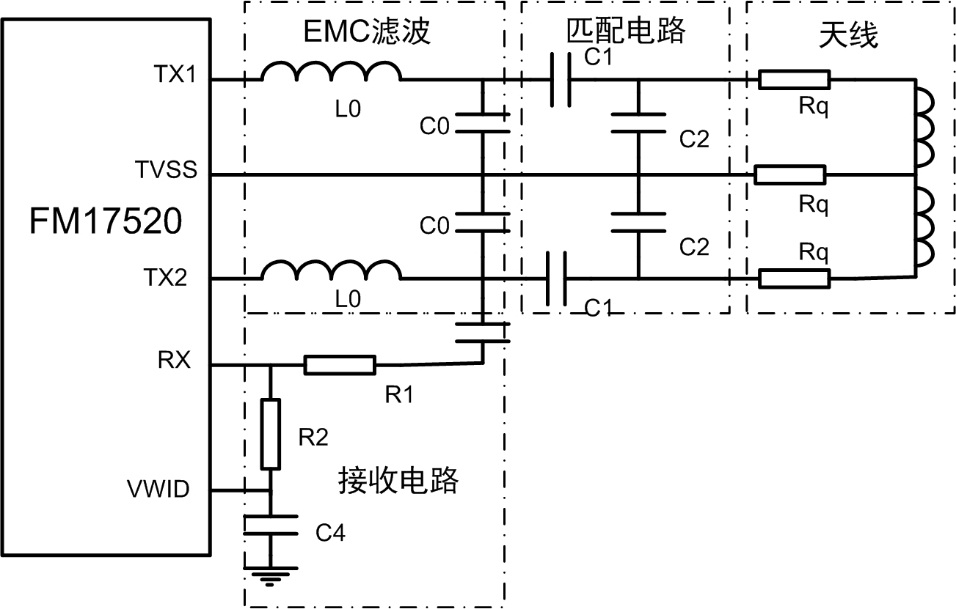


图3.3 读卡电路

滤波电路主要用于滤去高于13.56MHz的衍生谐波，由L0和C0组成的低通滤波器，滤波器截止频率设计在14MHz以上。这里推荐680nH电感和180pF电容或者1μH电感和120pF电容。这两组组合使得匹配网络既不偏感性也不偏容性，设计合理。

匹配电路用于调节发射负载和谐振频率。电路由电容C1和C2组成。射频电路发射功率一般受芯片内阻抗和外阻抗影响，当芯片内阻抗和外阻抗一致时，发射功率最大。C1是负载电容，天线电感量越大，C1取值越小。电容C2是谐振电容，通常设计由两个电容并联。C2取值和天线电感量直接相关，使得谐振频率在13.56MHz。

天线设计由Q值电阻Rq (通常1Ω或者0Ω)和印制PCB线路组成。 接收电路由R1、R2和C3、C4组成，其中C3=102pF、C4=104pF。R1和R2组成分压电路，使得RX端接收正弦波幅度电压在1.5V-3V之间。

### 读卡通信设计

Ametal已经提供了FM175XX系列的驱动函数，在使用之前，必须先完成初始化，初始化函数详见表3.2。

表3.2 FM175xx初始化接口函数

|  |  |
| --- | --- |
| 函数原型 | 功能简介 |
| uint8\_t am\_fm175xx\_init (am\_fm175xx\_dev\_t \*p\_dev,  am\_spi\_handle\_t spi\_handle,  const am\_fm175xx\_devinfo\_t \*p\_devinfo); | FM175xx初始化，得到  FM175xx设备信息 |

1. 初始化

初始化意在获取FM175xx实例句柄（handle），该实例句柄将作为其他功能接口函数handle的实参。其函数初始化原型为：

uint8\_t am\_fm175xx\_init (am\_fm175xx\_dev\_t \*p\_dev,

am\_spi\_handle\_t spi\_handle,

const am\_fm175xx\_devinfo\_t \*p\_devinfo);

* p\_dev为指向am\_fm175xx \_dev\_t类型实例的指针；
* p\_devinfo为指向am\_ fm175xx \_devinfo\_t类型实例信息的指针；
* spi\_handle为获取SPI服务的实例化句柄。

1. 实例信息

实例信息主要描述了FM175xx的相关信息，包括SPI设备信息、软件定时器、超时计数器、保存读卡芯片协议、命令信息、天线状态、掉电标志等信息。其类型am\_fm175xx\_dev\_t定义（am\_fm175xx.h）如下：

/\*\*

\* \brief FM175XX 设备定义

\*/

typedef struct am\_fm175xx\_dev {

am\_spi\_device\_t spi\_dev; // SPI 设备

am\_softimer\_t timer; //软件定时器，用于超时

volatile uint32\_t tmo\_cnt; //超时计数器

am\_fm175xx\_prot\_type\_t iso\_type; //保存读卡芯片协议

am\_fm175xx\_cmd\_info\_t cmd\_info; //命令信息

volatile uint8\_t tx\_state; //天线状态

volatile am\_bool\_t power\_down; // 掉电标志

am\_fm175xx\_tpcl\_prot\_para\_t cur\_prot\_para; //T=CL通信协议参数

const am\_fm175xx\_devinfo\_t \*p\_devinfo; //设备信息

void (\*lpcd\_int\_cb)(void \*p\_arg);

void \*p\_lpcd\_cb\_arg;

} am\_fm175xx\_dev\_t;

### 设备控制类接口函数

FM17520支持多种IC卡，比如，Mifare S50/S70、ISO7816-3、ISO14443（PICC）、PLUS CPU卡等，每种卡都有对应的命令。命令与接口函数基本上是一一对应的关系，AMetal提供了标准接口函数，与具体卡片没有直接关系，直接作用于FM17520，获取相应的设备信息、通信加密、设置防碰撞及卡请求模式等。

表3.3 FM17520设备控制接口函数

|  |  |
| --- | --- |
| 函数原型 | 功能简介 |
| uint8\_t am\_fm175xx\_crypto1 (am\_fm175xx\_dev\_t \*p\_dev,  uint8\_t mode,  const uint8\_t p\_key[6],  const uint8\_t p\_uid[4],  uint8\_t nblock); | 设置通信加密 |
| uint8\_t am\_fm175xx\_picca\_anticoll (am\_fm175xx\_dev\_t \*p\_dev,  uint8\_t anticoll\_level,  uint8\_t \*p\_uid,  uint8\_t \*p\_real\_uid\_len); | 设置防碰撞等级 |
| uint8\_t am\_fm175xx\_picca\_request (am\_fm175xx\_dev\_t \*p\_dev,  uint8\_t req\_mode,  uint8\_t p\_atq[2]); | 卡请求模式 |

* 设置通信加密

该函数意在设置通信加密类型，卡片内存储的数据均是加密的，必须验证成功后才能读写数据。验证就是将用户提供的密钥与卡片内部存储的密钥对比，只有相同才认为验证成功。设置密钥类型，主要有密钥A、密钥B，可设置为外部输入的密钥验证，或使用内部E2的密钥验证。使用内部密钥时，第一字节为密钥存放扇区。其函数定义如下：

uint8\_t am\_fm175xx\_crypto1 (am\_fm175xx\_dev\_t \*p\_dev,

uint8\_t mode,

const uint8\_t p\_key[6],

const uint8\_t p\_uid[4],

uint8\_t nblock);

IC卡密钥类型定义如下：

#define AM\_FM175XX\_IC\_KEY\_TYPE\_A 0x60 /\*\*< \brief 类型 A \*/

#define AM\_FM175XX\_IC\_KEY\_TYPE\_B 0x61 /\*\*< \brief 类型 B \*/

注：之所以存在两类密钥，是由于实际卡片中往往存在两类密钥，两类密钥可以更加方便地进行权限管理，比如，TypeA验证成功后只能读，而TypeB只有验证成功后才能写入，但权限可以自定义设置。

* 设置防碰撞等级

设置防碰撞等级，符合ISO14443A标准卡的序列号都是全球唯一的，正是这种唯一性，才能实现防碰撞的算法逻辑，当若干卡同时在天线感应区内，则这个函数能够找到一张序列号较大的卡来操作。该函数需要执行一次请求命令，并返回请求成功，才能执行防碰撞操作，否则返回错误。

uint8\_t am\_fm175xx\_picca\_anticoll (am\_fm175xx\_dev\_t \*p\_dev,

uint8\_t anticoll\_level,

uint8\_t \*p\_uid,

uint8\_t \*p\_real\_uid\_len);

防碰撞等级设置有如下三级设置：

#define AM\_FM175XX\_PICCA\_ANTICOLL\_1 0x93 /\*\*< \brief 第一级防碰撞 \*/

#define AM\_FM175XX\_PICCA\_ANTICOLL\_2 0x95 /\*\*< \brief 第二级防碰撞 \*/

#define AM\_FM175XX\_PICCA\_ANTICOLL\_3 0x97 /\*\*< \brief 第三级防碰撞 \*/

防碰撞等级设置参考卡的序列号长度，目前主流卡的序列号长度有三种，4字节、7字节和10字节，4字节选择第一级防碰撞即可得到完整的序列号，7字节使用第二等级防碰撞可得到完整序列号，前一级多得到的序列号的最低字节为级联标志0x88，在序列号内，只有3字节可用，后一级选择能得到4字节序列号，两者按顺序连接为7字节序列号，10字节以此类推。

* 卡请求模式设置

卡进入天线后，从射频场中获取能量，从而得电复位，复位后卡处于IDLE模式，用两种请求模式的任一种请求时，此时的卡均能响应，若对某一张卡成功挂起，则进入Halt模式，此时卡只响应ALL（0x52）模式的请求，除非将卡离开天线感应区后再进入。

uint8\_t am\_fm175xx\_picca\_request (am\_fm175xx\_dev\_t \*p\_dev,

uint8\_t req\_mode,

uint8\_t p\_atq[2]);

卡请求模式主要有IDLE和ALL两种，如下所示：

#define AM\_FM175XX\_PICCA\_REQ\_IDLE 0x26 /\*\*< \brief IDLE模式，请求空闲的卡 \*/

#define AM\_FM175XX\_PICCA\_REQ\_ALL 0x52 /\*\*< \brief ALL模式，请求所有的卡 \*/

### 操作接口函数

Mifare卡是一种符合ISO14443标准的A型卡，其接口函数详见表3.4。

表3.4 读卡操作接口函数

|  |  |  |
| --- | --- | --- |
| 接口类型 | 函数原型 | 功能简介 |
| 卡片自动  检测接口函数 | uint8\_t am\_fm175xx\_picca\_authent (am\_fm175xx\_dev\_t \*p\_dev,  uint8\_t key\_type,  const uint8\_t p\_uid[4],  const uint8\_t p\_key[6],  uint8\_t nblock); | 密钥验证 |
| 读卡操作接口函数 | uint8\_t am\_fm175xx\_picca\_read (am\_fm175xx\_dev\_t \*p\_dev,  uint8\_t nblock,  uint8\_t p\_buf[16]); | 卡读数据 |
| uint8\_t am\_fm175xx\_picca\_write (am\_fm175xx\_dev\_t \*p\_dev,  uint8\_t nblock,  const uint8\_t p\_buf[16]); | 卡写数据 |
| uint8\_t am\_fm175xx\_picca\_val\_set (am\_fm175xx\_dev\_t \*p\_dev,  uint8\_t nblock,  int32\_t value); | 卡值块写值 |
| uint8\_t am\_fm175xx\_picca\_val\_get (am\_fm175xx\_dev\_t \*p\_dev,  uint8\_t nblock,  int32\_t \*p\_value); | 卡值块读取 |

经常使用的公交卡、房卡、水卡和饭卡等均是Mifare卡。比如，S50和S70，它们的区别在于容量的不同。S50为1Kbyte，共16个扇区，每个扇区4块，每块16字节。S70为4K byte，共40个扇区，前32个扇区每个扇区4块，每块16字节，后8个扇区每个扇区16块，每块16字节。

* 密钥验证

由于绝大部分卡片在检测到时，都要先读取一块数据，因此可以将读取数据作为自动检测的一个附加功能。即在检测到卡片时，自动读取1块（16字节）数据。由于读取数据前均需要验证，这就需要在启动自动检测时，指定密钥验证相关的信息。将传入的密钥与卡的密钥进行验证，对应的卡的序列号有4字节和7字节之分，对于7字节的卡，只需将卡号的高4字节，即第二防碰撞等级得到的序列号作为验证的卡号即可。

每张卡片都具有一个唯一序列号，即UID。所有卡片的UID都是不相同的。卡的序列号长度有三种：4字节、7字节和10字节。uid\_len表明了读取到的UID的长度，uid[4]中存放了读取到的UID（字节数）。

uint8\_t am\_fm175xx\_picca\_authent (am\_fm175xx\_dev\_t \*p\_dev,

uint8\_t key\_type, //密钥类型

const uint8\_t p\_uid[4], //卡序列号，4字节

const uint8\_t p\_key[6], //密钥，6字节

uint8\_t nblock); //需验证卡块号，与卡有关

* 卡读数据

验证成功后，才能读相应的块数据，所验证的块号与读块号必须在同一个扇区内，Mifare1卡从块号0开始，按顺序每4个块1个扇区，若要对一张卡中的多个扇区进行操作，在对某一个扇区操作完成后，必须进行一条读命令才能对另一个扇区直接进行验证命令，否则必须从请求开始操作，对于PLUS CPU卡，若对下一个读扇区的密钥和当前扇区的密钥相同，则不需要再次验证密钥，直接读即可。

对应的密钥正确，验证成功，将读取启动自动检测时信息结构体的nblock成员指定的块（由信息结构体的nblock指定）的数据。读取的数据存放在p\_buf[16]数组中。

uint8\_t am\_fm175xx\_picca\_read (am\_fm175xx\_dev\_t \*p\_dev,

uint8\_t nblock, //读取数据的块号

uint8\_t p\_buf[16]); //存放读取数据，16bytes

* 卡写数据

对卡内某一块进行验证成功后，即可对同一个扇区的各个块进行写操作（只要访问条件允许），其中包括位于扇区尾的密码块，这是更改密码的唯一方法，对于PLUS CPU卡等级2、3的AES密钥则是其他位置修改密钥，写入数据缓冲区，大小必须为16。

uint8\_t am\_fm175xx\_picca\_write (am\_fm175xx\_dev\_t \*p\_dev,

uint8\_t nblock, //读取数据的块号

const uint8\_t p\_buf[16]); //写入缓冲区，大小必须为16

* 卡块值写操作

对Mifare卡块值的设置，其中，nblock指定写入的块号，value为指向写入数据的值，缓冲区大小为16字节。对卡内某一块进行验证成功后，并且访问条件允许，才能进行该写值操作。

uint8\_t am\_fm175xx\_picca\_val\_set (am\_fm175xx\_dev\_t \*p\_dev,

uint8\_t nblock, //块值地址

int32\_t value); //写入值

* 卡块值获取

对Mifare卡块值的读取，若验证成功，则开始读写已验证的块。读写数据都是以块为单位的，其大小为16字节，指定读取数据的值块地址，nblock指定本次验证的块号，可以使用该函数读取数值块的值。对卡内某一块进行验证成功后，并且访问条件允许，才能进行读值操作。

uint8\_t am\_fm175xx\_picca\_val\_get (am\_fm175xx\_dev\_t \*p\_dev,

uint8\_t nblock, //块值地址

int32\_t \*p\_value); //获取值指针

### 密钥和权限控制

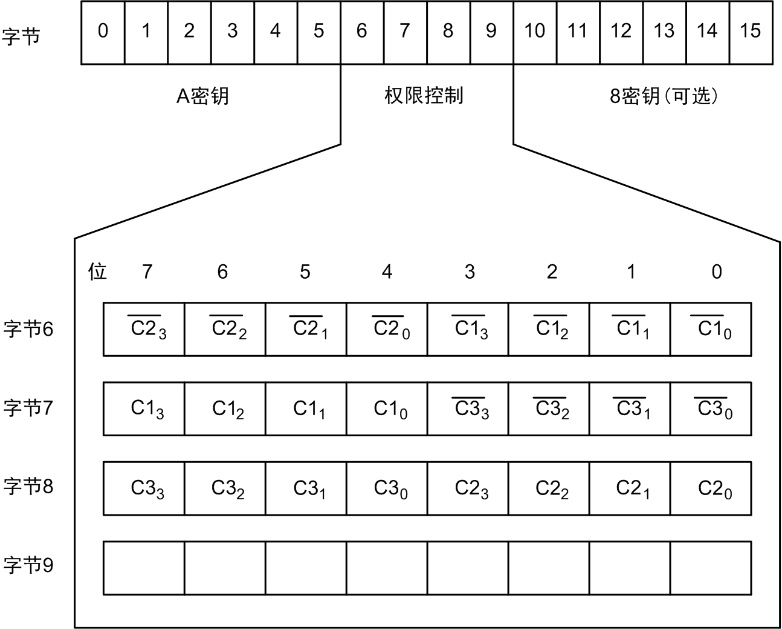


图3.4 尾块格式

Mifare S50/S70卡的初始密钥全为0xFF，显然，对于实际产品来讲，希望能够更改其密钥为其它值。由于存在密钥A和密钥B，可以对每个密钥设定不一样的权限，如验证密钥A后仅只读，验证密钥B后可写。下面以Mifare S50为例，介绍密钥和权限控制的修改方法。

密钥和权限控制是针对扇区而言的，即一个扇区的密钥是相同的，不同扇区的密钥可以不同。S50共计16个扇区，每个扇区4块，每块16字节，前3块为普通的数据块，最后一块（尾块）为密钥和权限控制块。对最后一块存储的数据进行修改，即可完成密钥和权限控制的修改。操作最后一块的存储数据时要格外小心，数据稍有错误，就可能导致扇区被锁死。尾块的前6字节为A密钥，后6字节为B密钥，中间4字节用于权限控制，详见图3.4。

如需修改密钥和控制权限，重点在理解字节6、7和8（字节9是一个普通的数据）的含义。3个字节共计24位，每6位（分别为C1、C2、C3、、、）控制扇区中的一个块，刚好可以控制4个块，图中的下标0、1、2、3对应块0、1、2、3。如 C10表示块0的C1控制位。同样的标志位，上方带横线的位必须与不带横线的位的值相反，即如果C10的值为1，则的值就必须为0。

由于存在此关系，因此实际控制位的含义仅通过C1、C2、C3三个位即可确定。控制尾块和数据块的控制位含义是不同的。对于尾块，其控制了密钥A、密钥B以及控制区域的访问权限。控制位的含义详见表3.5。

表3.5 尾块控制位含义

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 控制位 | | | 访问权限 | | | | | |
| 密钥A区域 | | 控制区域 | | 密钥B区域 | |
| C1 | C2 | C3 | 读 | 写 | 读 | 写 | 读 | 写 |
| 0 | 0 | 0 | × | KeyA | KeyA | × | KeyA | KeyA |
| 0 | 1 | 0 | × | × | KeyA | × | KeyA | × |
| 1 | 0 | 0 | × | KeyB | KeyA | B | × | × | KeyB |
| 1 | 1 | 0 | × | × | KeyA | B | × | × | × |
| 0 | 0 | 1 | × | KeyA | KeyA | KeyA | KeyA | KeyA |
| 0 | 1 | 1 | × | KeyB | KeyA | B | KeyB | × | KeyB |
| 1 | 0 | 1 | × | × | KeyA | B | KeyB | × | × |
| 1 | 1 | 1 | × | × | KeyA | B | × | × | × |

表中，“×”表示任何情况下都无权限，“KeyA”表示通过密钥A验证后可以取得权限，KeyB表示通过密钥B验证后可以取得权限，“KeyA | B”表示通过密钥A或者密钥B验证后均可取得权限。由此可见，密钥A的安全性很高，任何情况下都无法读出。特殊情况下，当C1C2C3的值为000、010或001时，验证密钥A后即可读取密钥B区域的数据。

无论什么情况，验证密钥A后，均可获得控制区域的读权限。通过读取控制区域，可以知道当前C1、C2、C3的值，以判断需要验证哪个密钥后可以获得密钥区域或控制区域的写权限，进而修改密钥和控制区域的值。比如，当前的C1、C2、C3的值为0、1、1，为了修改密钥A，则需要先验证密钥B，验证密钥B后，即可对尾块进行写入，写入时其它数据保持不变，仅修改前6字节（KeyA区域）的值即可完全对密钥A的修改。

对于数据块，C1、C2、C3控制了对块中存储数据的操作权限，详见表3.6。

表3.6 数据块控制位含义

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 控制位 | | | 数据访问权限 | | | |
| C1 | C2 | C3 | 读 | 写 | 加值操作 | 减值操作 |
| 0 | 0 |  | KeyA | B | KeyA | B | KeyA | B | KeyA | B |
| 0 | 1 | 0 | KeyA | B | × | × | × |
| 1 | 0 | 0 | KeyA | B | KeyB | × | × |
| 1 | 1 | 0 | KeyA | B | KeyB | KeyB | KeyA | B |
| 0 | 0 | 1 | KeyA | B | × | × | KeyA | B |
| 0 | 1 | 1 | KeyB | KeyB | × | × |
| 1 | 0 | 1 | KeyB | × | × | × |
| 1 | 1 | 1 | × | × | × | × |

同样，表中“×”表示任何情况下都无权限，“KeyA”表示通过密钥A验证后可以取得权限，KeyB表示通过密钥B验证后可以取得权限，“KeyA | B”表示通过密钥A或者密钥B验证后均可取得权限。

加值操作（相当于充值）和减值操作（相当于消费）是对块中存放的值进行增加和减少操作，加值和减值均有对应的命令可以直接使用。例如，当前块1的C1、C2、C3控制位的值为0、0、0（默认值），只要密钥A或密钥B验证通过后，均可取得数据块的读、写、加值、减值的权限。可以根据实际需要，修改尾块中相应控制位的值（修改时，需确保具有写入控制区域的权限），以对数据进行保护。

需要注意的是，凡是表中标识验证密钥B后可以取得权限的，在特殊情况下验证密钥B后可能并不能取得权限。在介绍尾块控制位含义时，当C1、C2、C3的值为000、010或001时，KeyB区域将可能被读取，详见表3.5。这些情况下，由于密钥B可能被读取，为了确保安全，此时密钥B验证将无效，即使密钥B验证通过，同样无法取得相应的权限。

更多读卡功能组件的使用请参考《面向AMetal框架与接口的编程》第5.4章节。

# 智能门锁应用方案

## 产品介绍



图4.1 产品展示

智能门锁产品实物如图4.1所示，广泛应用于银行、政府部门，以及酒店、学校宿舍等。智能门锁区别于传统机械锁，在用户安全性、识别、管理性方面更具优势。

智能门锁内部集成非接触式读卡模块、蓝牙模块、NB模块，支持刷卡、手机APP蓝牙控制、远程控制多种开门方式。本智能门锁方案主控采用KL16Z128，通过SPI接口与读卡芯片FM17550通信，实现非接触式读卡，通过UART接口与BLE ZLG52810、NB ZM7100M通信，实现APP开锁、远程控制及管理。

## 方案介绍

### 功能框图

智能门锁功能框图如图4.2所示，智能门锁通过7.8V锂电池供电，经过LDO转化为3.3V给MCU供电，外置2个LED状态指示灯和1个蜂鸣器，支持触摸按键，通过4个I/O控制门锁电机，外扩EEPROM用于存储用户数据，RTC用于时间管理。主控通过UART接口与BLE模块、NB无线模块通信，用于实现手机APP开锁、远程控制，方便用户管理和使用。

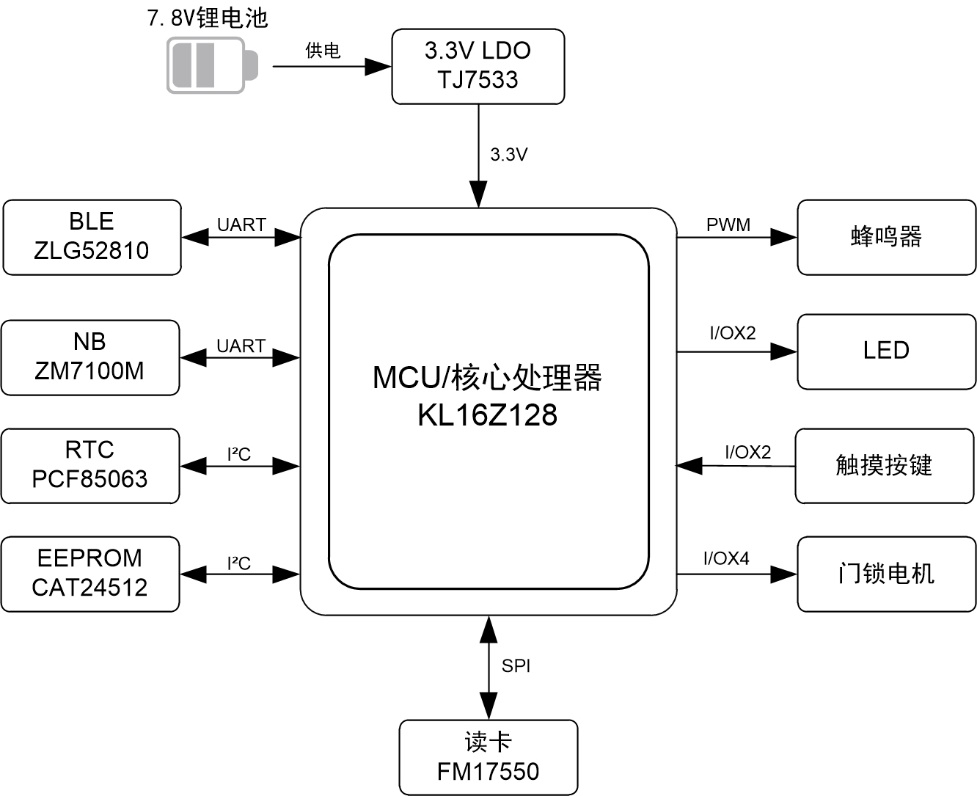


图4.2 功能框图

### 资源需求

* 2路I/O实现LED状态显示；
* 1路I/O实现蜂鸣器发音提示；
* 2路I/O实现触摸按键检测输入；
* 4路I/O用于控制门锁电机；
* 1路SPI接口用于与读卡芯片FM17550通信；
* 1路I2C接口用于外扩EEPPROM；
* 1路I2C接口用于读取RTC时钟；
* 1路UART接口用于与NB ZM7100M模块通信；
* 1路UART接口用于与BLE ZLG52810模块通信。

### 优势特点

对比智能门锁同类产品现用芯片方案，有如下优势：

* MCU资源丰富（128KB Flash / 16K SRAM）、开发更为灵活，超高性价比；
* 完善齐全的DEMO软件和详尽的技术文档，帮助客户快速完成产品开发；
* 提供SPI读卡驱动、EEPPROM驱动、RTC驱动、带缓冲区的UART接口通讯等丰富的组件软件，极大简化客户的开发，更专注于核心应用软件的设计；
* 采用AMetal软件架构，真正实现跨平台移植，帮助客户快速完成产品升级换代。

### 推荐器件

ZLG提供智能门锁全套BOM解决方案，全套BOM打包，一站式采购，降低整体成本。主控芯片采用NXP低功耗芯片KL16Z128设计，详细器件推荐如表1.1所示。

表4.1 推荐器件

|  |  |  |  |
| --- | --- | --- | --- |
| 产品名称 | 型号 | 厂家 | 器件特点 |
| MCU | KL16Z128 | NXP | Cortex-M0+内核，128 KB Flash / 16K SRAM；  运行频率高达48MHz；  支持宽电压输入1.71–3.6 V；  低功耗，低功耗停止模式典型电流2.71µA；  多路UART、SPI、I2C等外设接口。 |
| 读卡芯片 | FM17550 | 复旦微 | 非接触式读卡、低电压、长距离。 |
| LDO | TJ7333 | MPS | 成本低、噪音小、静态电流小。 |
| EEPROM | CAT24512 | 安森美 | 宽电压输入1.7~5.5V、低功耗、静态电流小。 |
| RTC | PCF85063 | NXP | 成本低、低功耗、静态电流小。 |
| BLE | ZLG52810 | ZLG | 成本低、低功耗、静态电流小。 |
| NB | ZM7100M | ZLG | 成本低、低功耗、静态电流小。 |

## 参考设计

### 电路设计

### RTC实时时钟

本智能门锁方案采用NXP半导体公司的PCF85063，是一款低功耗实时时钟芯片，它提供了实时时间的设置与获取、闹钟、可编程时钟输出、中断输出等功能。

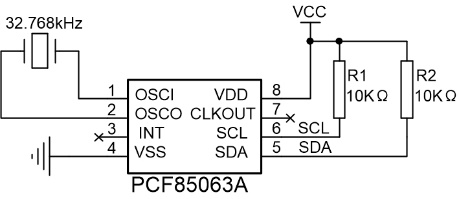


图4.3 PCF85063电路原理图

PCF85063电路如图4.1,其中SCL和SDA为I2C接口引脚，VDD和VSS分别为电源和地；OSCI和OSCO为32.768KHz的晶振连接引脚，作为PCF85063的时钟源；CLKOUT为时钟信号输出引脚，供外部电路使用；INT为中断引脚，主要用于定时、闹钟等功能。PCF85063的7位I2C从机地址为0x51。

AMetal提供了PCF85063组件及RTC通用接口，PCF85063初始化完成后，即可调用通用接口设置时间和获取时间，其接口函数详见表4.2。

表4.2 RTC接口函数

|  |  |
| --- | --- |
| 函数原型 | 功能简介 |
| am\_rtc\_handle\_t am\_microport\_rtc\_std\_inst\_init (void); | 初始化 |
| am\_static\_inline int am\_rtc\_time\_set (  am\_rtc\_handle\_t handle,  am\_tm\_t \*p\_tm); | 设置时间 |
| am\_static\_inline int am\_rtc\_time\_get(  am\_rtc\_handle\_t handle,  am\_tm\_t \*p\_tm); | 获取时间 |

#### 初始化

在使用PCF85063前，必须调用初始化函数完成PCF85063的初始化操作，以获取对应的操作句柄，进而才能使用PCF85063的各种功能。PCF85063的实例初始化函数原型（am\_hwconf\_microport\_rtc.h）为：

am\_rtc\_handle\_t am\_microport\_rtc\_std\_inst\_init (void);

使用无参数的PCF85063实例初始化函数，即可获取RTC实例句柄，进而通过RTC通用接口使用PCF85063的各种功能。

#### 设置时间

该函数意在设置RTC的当前时间，其函数原型为：

am\_static\_inline int am\_rtc\_time\_set (am\_rtc\_handle\_t handle, am\_tm\_t \*p\_tm);

其中，handle为PCF85063的实例句柄，p\_tm为待设置的时间值的指针。返回AM\_OK，表示设置成功，反之失败。其类型am\_tm\_t是在am\_time.h中定义的细分时间的结构体类型，用于表示年/月/日/时/分/秒等信息，结构体原型为：

typedef struct am\_tm {

int tm\_sec; // 秒，0 ~ 59

int tm\_min; // 分，0 ~ 59

int tm\_hour; // 小时， 0 ~ 23

int tm\_mday; // 日期， 1 ~ 31

int tm\_mon; // 月份， 1 ~ 12

int tm\_year; // 年

int tm\_wday; // 星期

int tm\_yday; // 天数

int tm\_isdst; // 夏令时

} am\_tm\_t;

其中，tm\_year表示年，1900年至今的年数，其实际年为该值加上1900。tm\_wday表示星期，0~6分别对应星期日~星期六。tm\_yday表示1月1日以来的天数（0~365），0对应1月1日。tm\_isdst表示夏令时，夏季将调快1个小时。如果不使用，可设置为-1。设置当前时间的程序详见程序清单4.1，星期等附加的一些信息无需用户设置，主要便于在获取时间是获得更多的信息。

程序清单4.1 设置时间范例程序

1 am\_local am\_tm\_t \_\_g\_current\_time = {

2 55, // 秒

3 59, // 分

4 11, // 小时

5 9, // 日期

6 1-1, // 月份

7 2019-1900, // 年

8 0, // 星期

9 0, // 天数

10 -1 // 夏令时

11 };

12 am\_rtc\_time\_set(rtc\_handle, &\_\_g\_current\_time);

#### 获取时间

该函数意在设置RTC的当前时间，其函数原型为：

am\_static\_inline int am\_rtc\_time\_get (am\_rtc\_handle\_t handle, am\_tm\_t \*p\_tm);

其中，handle为PCF85063的实例句柄，p\_tm为指向时间值的指针。返回AM\_OK，表示获取时间成功，反之失败。获取时间的程序详见程序清单4.2。

程序清单4.2 获取时间范例程序

1 am\_tm\_t time;

2 am\_rtc\_time\_get(rtc\_handle, &time);

关于AMetal RTC实时时钟组件、通用接口的实现和使用请参考《面向AMetal框架与接口的编程》第6.3章节。

### EEPROM储存器

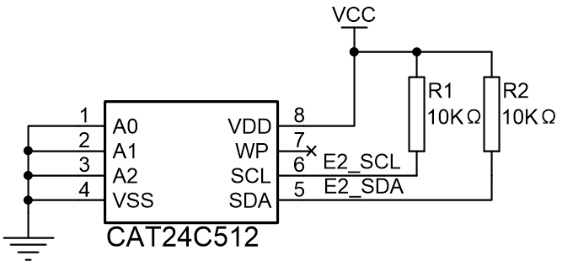


图4.4 CAT24C512电路原理图

本智能门锁方案需要外扩EEPROM用于存储用户数据，采用安森美的CAT24C512。CAT24C512总容量为512K（512\*1024）bits，即65536（512\*1024/8）字节。每个字节对应一个储存地址，因此其储存数据地址范围为0x0000~0xFFFF。CAT24C512页（page）的大小为128字节，分512页。支持按字节读写和按页读写，按页读写一次可高达128字节。

CAT24C512的通信接口为标准的I2C接口，仅需SDA和SCL两根信号线，其电路原理图如图4.4。

### 52810 BLE组件

# 待续