第3章

基于 STM32 的智能锁的设计与实现

3.1 功能分析

3.1.1 智能锁设计的意义

随着智能时代的来临，智能家居产品开始取代传统家庭设备。在智能家居系列产品中，智能锁既是用户安全的智能硬件，也是家居安全的管理入口，还是智慧家庭的服务应用入口。如今，随着消费水平与生活品质的不断提升，人们每天进行的开门与关门的动作都和锁有着密不可分的关联，越来越多的智能锁进入人们的视野。作为第一道安全防线，智能锁所起的作用不言而喻。智能锁的出现不仅解决了忘带钥匙的烦恼，同时也给人们带来了更加安全与便捷的生活享受。

3.1.2 需求分析

一般的智能锁可以分为以下几类。

密码锁：通过密码开锁。密码锁有两种，一种是拨盘式的机械密码锁，另一种是键盘密码锁，分为按键和触屏两种。

IC 卡锁：又分为接触式 IC 卡锁与非接触式 IC 卡锁。接触式 IC 卡锁需要将 IC 卡插入锁缝，与芯片锁的内部读卡器接触；非接触式 IC 卡锁只需要将 IC 卡与锁的读卡器靠近即可。

还有指纹识别锁、人脸识别锁等。

3.1.3 具体功能

智能锁可以实现以下功能。

1. 指纹采集和识别

通过指纹读取器对指纹实现采集或者识别，当处于识别模式时，可以根据对比结果决定是否开门，并将结果显示在 OLED 上。

2. 开门功能

通过指纹和密码的解锁判断是否执行开门功能，开门模块可以选择继电器和电机这两种方式。

3. 摄像头抓拍功能

用户可以在门内选择是否启用摄像头，启用时可以点击屏幕进行拍照。在进行解锁操作时，如果密码或指纹错误超过两次，则摄像头自动进行抓拍，并将抓拍结果保存在存储卡中。

4. 远程查看记录和设置临时密码

通过 Wi-Fi 模组连接路由器并接入互联网，连接到云服务器，将开门情况上传至云端，通过云端设置临时密码。

5. 密码管理功能

方便管理员管理指纹和开门密码，管理员可以增加和删除指纹或修改开门密码。

3.2 总体设计

3.2.1 系统设计

智能锁使用 STM32 作为主控芯片，使用指纹模块、按键模块以及人机交互模块作为基础，并结合嵌入式技术和无线传输等技术，实现了从云端采集到开门情况记录的智能锁系统。通过指纹锁对指纹进行采集和识别，将采集到的结果上传至主控芯片，主控芯片根据其采集的结果决定执行状态。当解锁达到一定错误次数时，通过摄像头实时抓拍门外环境，并将解锁结果上传至云端，以提醒用户。

本设计主要分为硬件设计和软件设计两个部分。其中，硬件部分由 STM32、指纹模块、按键模块、人机交互模块、Wi-Fi 模组、继电器、电机驱动、摄像头和 TF 存储卡组成；软件部分主要由 μC/OS Ⅲ实时操作系统、STemWin 界面管理系统以及指纹模块、人机交互模块、摄像头模块、TF 存储卡、Wi-Fi 模组等驱动程序构成；最后根据这些模块的驱动程序编写应用程序，从而实现智能锁的全部功能。系统的整体设计架构如图 3-1 所示。

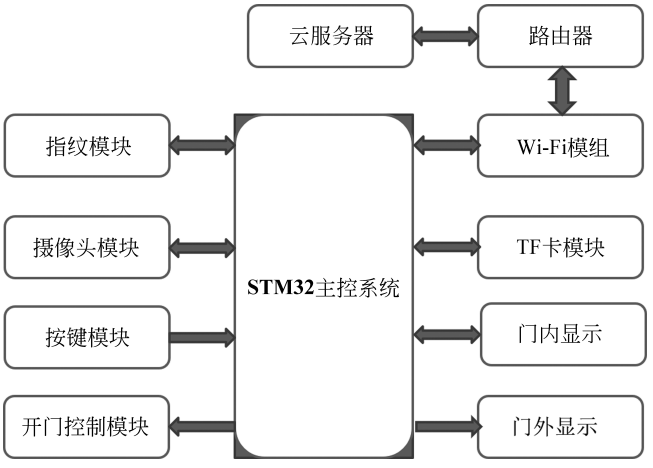

图 3-1 系统整体结构

3.2.2　系统硬件选型

1. 主控芯片的选择

选择 STM32 F103ZET6 作为主控芯片，STM32 F103ZET6 是一款高性能的 MCU，其凭借产品线的多样化、极高的性价比、简单易用的库开发方式迅速在众多 Cortex-M3 MCU 中脱颖而出，STM32 具有多种外设，如 SPI、CAN、I2C、USART 和 FSMC 等。其中，STM32 F103ZET6 的 RAM 为 64KB，Flash 为 512KB，主频高达 7MHz，支持 FSMC 内存扩展接口，特别是其拥有的 DMA 控制器可以将 CPU 从繁忙的数据传输中解脱出来。

2. 指纹模块的选择

首先，指纹模块应该具有指纹图像采集、计算、存储、对比等处理功能。一个高性能的指纹模块对指纹的采集、对比等的处理速度应该比较快，这也能提高系统的反应速度。一个高性能的指纹模块可以提高信息的安全性，且具有低功耗、体积小等特点。AS608 模块配备了串口接口，是一种光学传感器，在开发时无须研究复杂的图像处理及指纹识别算法，只需要通过简单的串口遵循通信协议即可控制模块。同时，AS608 模块的指纹适应性好，无论手指干湿都能准确识别指纹，其指纹识别算法优异，识别准确。AS608 模块的功能丰富，只需要通过串口使用不同的命令即可控制本模块，大幅降低了开发难度。

3. 摄像头的选择

野火 OV7725 模块是一颗 1/4 英寸的 CMOS VGA(640×480)图像传感器，摄像头模块采用该 OV7725 传感器作为核心部件，集成有源晶振和 FIFO(AL422B)，这样便对 MCU 的读速度没有了硬性要求，可以方便各种 MUC 使用。该摄像头还具有以下特点：集成有源晶振；无须外部提供时钟；集成 FIFO 芯片(AL422B)；支持 VGA、QVGA 等各种格式；支持 RGB、YCbCr 等格式输出；具有自动曝光、自动白平衡、自动消除灯光条纹等自动图像控制功能。OV7725 的这些优点完全可以满足本设计的要求，因此本设计采用 OV7725 作为图像采集部分。

4. Wi-Fi 模组的选择

本设计需要将开门记录上传至云服务器，并通过云服务器设置一些智能锁的参数，因此需要一个可以联网的模块，这里选择 Wi-Fi 作为联网模块。本设计选择 ATK-ESP8266 模组作为 Wi-Fi 传输模块，这是因为该模组内置了 TCP/IP 协议栈，能够实现串口与 Wi-Fi 之间的转换，支持串口转 Wi-Fi STA、串口转 AP 和 Wi-Fi STA＋Wi-Fi AP 的模式，从而可以快速构建串口至 Wi-Fi 的数据传输方案，方便设备使用互联网传输数据，本模组只需一条指令即可连接原子云，然后进行数据传输。

5. 人机交互模块的选择

由于本设计使用了摄像头，为了能够在门内实时显示图像，因此必须使用一个高速度的、支持 RBG 显示的屏幕。因此，本设计选择 TFT LCD 电阻触摸屏作为门内显示屏幕。

3.3 硬件设计

硬件电路是整个智能锁系统运行的基础。根据需求，本设计主要由以下模块组成：STM32 最小系统、指纹模块、人机交互模块、Wi-Fi 模组、摄像头模块、存储模块和继电器模块等。整体电路图使用绘制原理图和 PCB 板，总体系统的原理图见附录 A。

STM32 作为整个智能锁系统的主控芯片，它不仅需要完成各种信息的处理，还需要在操作系统下调度各个模块协调工作，如指纹、摄像头、Wi-Fi 等模块。

指纹模块主要完成指纹信息的采集、识别、存储等功能，并将处理结果通过串口上传至主控芯片，其余操作则交由主控芯片完成。

Wi-Fi 模块需要连接路由器和云服务器，为将解锁记录上传至云服务器提供基础，实现远程监控和控制。

摄像头需要完成图像的实时采集，将采集到的图像上传至主控芯片，并实时显示在 LCD 上，同时自动或者手动地进行拍照。

存储模块用来保存 LCD 和 OLED 所使用的字库以及开门密码和管理员密码等参数。

人机交互模块主要由显示屏和按键组成，本设计的显示模块又分为门内显示和门外显示，门内显示可以调整设置，门外显示用于指示解锁状态。

继电器用来控制需要使用高电压工作的器件功能的通断。

3.3.1 STM32 最小系统

STM32 最小系统主要包括 STM32 F103ZET6、晶振电路和复位电路，其最小系统结构如图 3-2 所示。

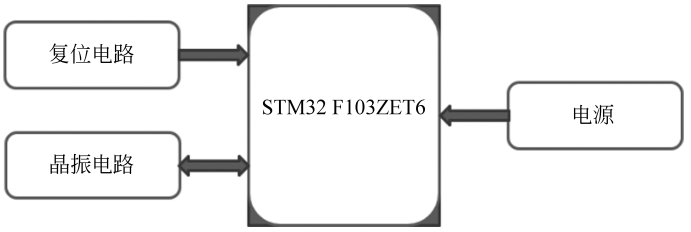

图 3-2　STM32 最小系统结构

3.3.2 指纹模块 AS608

AS608 指纹模块中内置了 DSP 运算单元，该模块集成了指纹采集、识别、存储等功能。在需要指纹模块工作时，只需要主控芯片通过串口发送相应的指令即可，如采集指纹图像、生成指纹特征、与库进行对比等。当指纹模块执行完相应操作后，会通过串口将执行结果上传至主控芯片。

该指纹模块除了具有串口通信使用的引脚以外，还有一个中断引脚，其功能是当指纹

模块检测到指纹头上有手指按下时，通过中断引脚输出一个高电平，在没有手指按下时则为低电平。中断引脚可以提高系统的反应速度，也不用在没有手指按下的情况下使用扫描的方式确认是否有指纹。模块接口的详细介绍如表 3-1 所示。

表 3-1　AS608 引脚说明

引脚编号	引　脚　名	引脚说明
1	VDD	供电引脚
2	INT	中断引脚
3	GND	地
4	TX	串行数据输出
5	RX	串行数据输入
6	VDD	触摸供电脚

3.3.3　显示设计

使用 TFT 3.2 英寸 LCD 电阻触摸屏作为门内人机交互模块，该显示屏使用 8080 并口进行像素点数据传输，由于 8080 并口的通信协议和 FSMC（可变静态存储控制器）的通信协议相似，因此可以使用 FSMC 模拟 8080 并口。设计电路时需要将显示屏和 STM32 FSMC 外设的对应引脚连接起来。读触摸屏的触摸位置时使用的是 SPI，所以在设计电路时需要将触摸屏的 SPI 和 STM32 的 SPI 引脚对应连接。显示屏幕引脚的具体说明如表 3-2 所示。

表 3-2　LCD 引脚说明

引脚编号	引　脚　名	引脚说明
1	GND	地
2	RES	复位
3	D15	数据 15
4	D14	数据 14
5	D13	数据 13
6	D12	数据 12
7	D11	数据 11
8	D10	数据 10
9	D09	数据 9
10	D08	数据 8
11	D07	数据 7
12	D06	数据 6
13	D05	数据 5

续表

引脚编号	引脚名	引脚说明
14	D04	数据 4
15	D03	数据 3
16	D02	数据 2
17	D01	数据 1
18	D00	数据 0
19	RD	读使能
20	WR	写使能
21	RS	数据命令选择
22	CS	片选
23	IO1	电阻屏时钟：SPI_CLK
24	IO2	电阻屏片选：SPI_CS
25	IO3	电阻屏输入：SPI_MOSI
26	IO4	电阻屏输出：SPI_MISO
27	IO5	电阻屏中断：INT
28	BK	背光，低电平亮
29	NC	悬空
30	NC	悬空
31	3V3	电源
32	GND	地

使用 OLED 作为门外显示模块，OLED 具有 SPI 和 I^2C 两种通信模式，本设计中选择 I^2C 通信模式的 OLED。由于 I^2C 是半双工的，通信时使用时钟线和数据线，数据线需要在输入和输出中切换，因此在设计电路时需要在该引脚上连接一个上拉电阻，这样可以在不改变 STM32 引脚配置的情况下直接使用输入和输出两种模式。OLED 接口引脚的具体说明如表 3-3 所示。

表 3-3　OLED 引脚说明

引脚编号	引脚名	说　明
1	VDD	供电引脚
2	GND	地
3	SCL	时钟线
4	SDA	数据线

3.3.4　按键设计

1. 独立按键

在智能锁系统中，一共使用了两个独立按键，分别是门铃和“一键开门”按键。在按键两端并联 0.1μF 的电容用于消抖。两个按键连接至不同的引脚号，以便使用中断控制。两个按键的电路图如图 3-3 所示。

图 3-3　独立按键

2. 矩阵按键

矩阵按键用于用户在门外输入密码，其主要功能是输入字符 0～9、删除最后一个字符以及确认密码，所以使用 3×4 的矩阵键盘。为了降低控制成本，本设计使用现成的矩阵按键模块以减小 PCB 板的面积，其接口电路如图 3-4 所示。

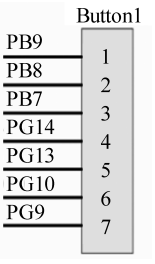

图 3-4　矩阵按键接口

3.3.5　Wi-Fi 模组设计

Wi-Fi 模组用于连接智能锁系统至云服务器，以达到远程监控和远程控制的目的。使用 AKT-ESP8266 模组，该模组使用的固件可以通过指令直接连接原子云。该模组有 USART、SPI 等通信接口，这里使用 USART 接口，其他通信引脚不连接主控引脚，悬空即可。模组中的 RESET 引脚为低电平有效，所以使用一个电阻上拉至 VDD。模组引脚说明如表 3-4 所示。

表 3-4　ESP8226 模组引脚说明

引脚编号	引　脚　名	说　　明
1	VDD	供电引脚
16	GND	地
7	RX	串行数据输入
8	TX	串行数据输出

3.3.6　摄像头和 TF 卡设计

摄像头采用 OV7725 模块，该摄像头使用 SCCB 通信协议配置其工作模式和各项配置，而像素点的传输则使用并口通信。该摄像头用于实时采集门外环境，通过主控芯片的控制将采集到的图像保存到 FIFO 中供主控芯片读取，以实现图像的实时显示和拍照。拍照时会把照片保存在 TF 存储卡中。摄像头的引脚说明如表 3-5 所示。

表 3-5　OV7725 引脚说明

引脚编号	引　脚　名	引脚说明
1	3V3	电源
2	D0	数据 0
3	OE	FIFO,输出使能
4	D1	数据 1
5	RRST	FIFI,读复位
6	D2	数据 2
7	RCLK	FIFO,读时钟
8	D3	数据 3
9	NC	不用接,悬空即可
10	D4	数据 4
11	VSYNC	场中断
12	D5	数据 5
13	WRST	FIFO,写复位
14	D6	数据 6
15	WEN	FIFO,写使能
16	D7	数据 7
17	HR	不用接,悬空即可
18	SCL	SCCB,时钟
19	GND	地
20	SDA	SCCB,数据

TF 存储卡有 SPI 和 SDIO 两种通信协议,这里使用具有 SDIO 协议的接口,这样能够加快对 TF 存储卡的读写速度。TF 存储卡的引脚说明如表 3-6 所示。

表 3-6　TF 存储卡引脚说明

引脚编号	引　脚　名	说　　明
1	DATA2	数据引脚 2
2	DATA3	数据引脚 3
3	CMD	命令引脚
4	VDD	电源
5	CLK	串行时钟
6	GND	地
7	DATA0	数据引脚 0
8	DATA1	数据引脚 1

3.3.7　存储器设计

由于 STM32 内部的 FLASH 只有 512KB，而 STemWin 要使用的字体却很大，512KB 完全不够使用，因此这里选择外扩 SPI FLASH，选择的型号是 W25Q128。该 FLASH 的内存共有 16MB，可以满足本设计的要求。该 FLASH 是通过 SPI 与主控芯片进行数据传输的，因此在设计中应将 FLASH 的引脚与 STM32 的 SPI 引脚对应连接。FLASH 引脚的说明如表 3-7 所示。

表 3-7　W25Q128 引脚说明

引脚编号	引　脚　名	引脚说明
1	CS	片选输入
2	SO	数据输出
4	GND	地
5	SI	数据输入
6	CLK	串行时钟
8	VCC	电源

3.3.8　开关门执行电路设计

考虑到现在市面上的智能锁一般有继电器控制和电机控制这两种方式，因此本设计预留两个开门接口，分别是继电器驱动和电机驱动。

1. 继电器驱动电路

由于继电器一般需要驱动较大电压，因此这里选择使用光耦进行隔离，并增加一个 LED 指示灯作为开门指示灯。设计电路图如图 3-5 所示。

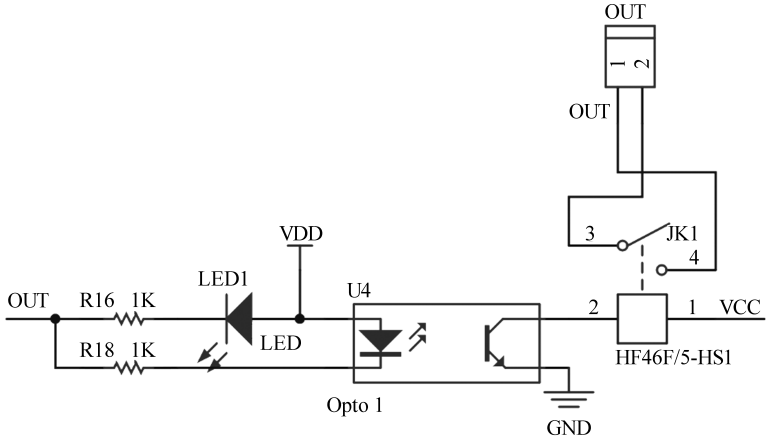

图 3-5　继电器模块

2. 电机驱动电路

现在市面上的电机驱动多数使用步进电机，因此这里使用 UNL2803 作为驱动电路，最多可以驱动四线制的步进电机。UNL2803 的引脚说明如表 3-8 所示。

表 3-8　UNL2803 引脚说明

引脚编号	引　脚　名	引脚说明
1	IN1	1 通道输入管脚
2	IN2	2 通道输入管脚
3	IN3	3 通道输入管脚
4	IN4	4 通道输入管脚
18	OUT1	1 通道输出管脚
17	OUT2	2 通道输出管脚
16	OUT3	3 通道输出管脚
15	OUT4	4 通道输出管脚
10	VCC	电源
9	GND	地

3.3.9　门铃设计

门铃使用一个有源蜂鸣器，使用 S8050 三极管蜂鸣器的驱动放大电流，电路图如图 3-6 所示。

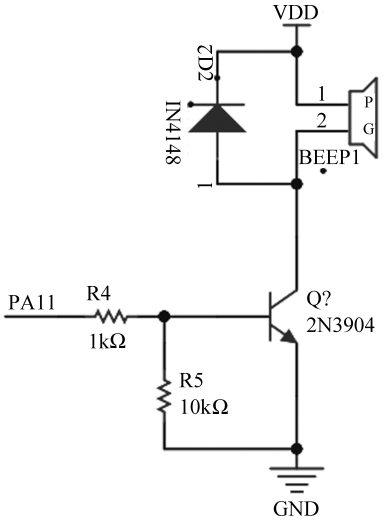

图 3-6　门铃电路

3.3.10　实时时钟设计

STM32 内部虽然有 RTC 外设，但是经测试发现，经过较长一段时间后，还是会出现误差，因此选择一个精度更高的实时时钟芯片及 DS1302。该时钟芯片的接口兼容 SPI 接口，但其通信协议并不是标准的 SPI 总线协议，因此其通信过程需要通过软件编写，只需要在主控芯片上选择空脚连接即可，使用一个纽扣电池在外部电源断电时给实时时钟芯片供电，由于该芯片的 I/O 脚既作为输出，又作为输入，因此在该引脚上增加一个电阻，将电压上拉到 VOD。该芯片引脚的说明如表 3-9 所示。

表 3-9　DS1302 引脚说明

引脚编号	引　脚　名	引脚说明
1	VCC2	电源 2
2	X1	晶振引脚
3	X2	晶振引脚
4	GND	地
5	RST	复位
6	I/O	数据输入输出脚
7	SCLK	串行时钟脚

3.4　软件设计

本设计搭建了基于 STM32 的嵌入式开发系统，移植了 μC/OS Ⅲ操作系统和 STemWin 图形界面管理工具。使用操作系统可以实现多任务管理；使用 STemWin 可以进行界面管理和设计，以大幅降低界面的设计和管理难度。由于本设计使用了 μC/OS Ⅲ和 STemWin，因此软件部分的设计思路和实现方法将会按照 μC/OS Ⅲ创建的任务进行分类介绍。系统各任务的状态如图 3-7 所示。

3.4.1　μC/OS Ⅲ简介

μC/OS Ⅲ是一个实时操作系统，支持多任务管理，任务数量在理论上可以有无数个，主要受限于运行平台的 RAM，不同优先级的任务是抢占式的，即高优先级的就绪任务可以抢占低优先级任务对 CPU 的控制权，对于同优先级的任务，则使用时间片轮转调度，这个时间可以由用户自己设置。μC/OS Ⅲ有很多服务，如信号量、消息队列、互斥信号量等，在这里就不一一介绍了，仅介绍本设计使用到的消息队列和任务信号量。

消息队列用于在任务之间同步消息，通过消息队列服务，任务或中断服务程序可以将消息放入消息队列。同样，一个或多个任务可以从消息队列中获得消息。当有多个消息发送到消息队列时，通常将先进入消息队列的消息传给任务，也就是说，任务先得到的是最先进入消息队列的消息，即“先进先出”原则(FIFO)，但是 μC/OS 也支持“后进先出”原

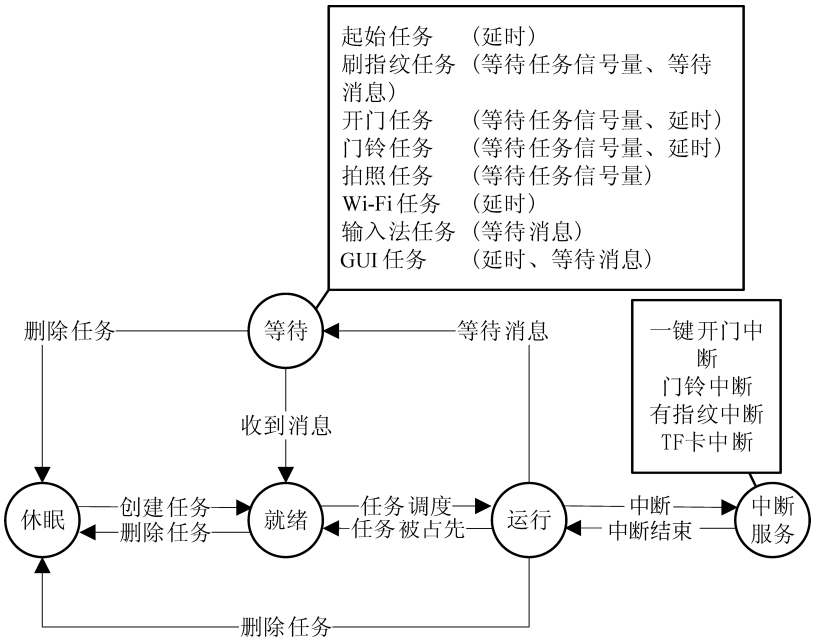

图 3-7 各任务的状态

则(LIFO)。

信号量(Semaphore)是一种实现任务之间通信的机制,可以实现任务之间同步或临界资源的互斥访问,常用于协助一组相互竞争的任务访问临界资源。在多任务系统中,各任务之间需要通过同步或互斥实现临界资源的保护,信号量功能可以为用户提供这方面的支持。μC/OS Ⅲ还提供任务信号量这个功能,每个任务都有一个 32 位(用户可以自定义位宽,使用 32 位的 CPU,此处就是 32 位)的信号量值 SemCtr,这个信号量值是在任务控制块中包含的,是任务独有的一个信号量通知值。在大多数情况下,任务信号量可以替代内核对象的二值信号量、计数信号量等,本设计只使用了任务信号量。

μC/OS Ⅲ的主要特点如下。

公开源代码:μC/OS Ⅲ的源代码全部开源,使得操作系统变得更加透明、易用和扩展。

可移植性好:μC/OS Ⅲ的大部分代码使用 C 语言编写,其可移植性强。与硬件相关的代码则是使用汇编语言编写的。μC/OS Ⅲ可以在大部分 8 位、16 位和 32 位的 MUC 上运行。

可固化:开发者可以通过编译、连接、下载和固化等手段使 μC/OS Ⅲ成为成品的一部分。

可裁剪:μC/OS Ⅲ提供多种服务,如信号量、消息队列、事件和互斥信号量等。服务开发者可以通过宏定义决定禁用还是启用,通过禁用不需要的服务可以减少产品对存储空间的需求。

多任务：理论上可以支持无数个任务，在同一优先级下可以有多个任务，同一优先级的任务是通过时间片轮转调度的；对于不同优先级的任务，则运行就绪任务中优先级最高的任务。

可确定性：全部 μC/OS Ⅲ的函数调用与系统服务的执行时间是确定的，这个时间不受应用程序数量的影响。

任务栈：每个任务都有自己独立的栈。通过不同的任务对栈的需求进行不同的分配，这样可以减少应用程序对 RAM 的需求。

系统服务：μC/OS Ⅲ提供了多种服务，如信号量、消息队列、事件、互斥信号量、任务信号量和软件定时器等。

中断管理：中断可以使正在运行的任务被挂起，转而执行中断服务函数，支持开关中断、恢复中断、中断使能、中断屏蔽、中断嵌套、中断延迟发布。

3.4.2　STemWin 简介

STemWin 是 Segger 公司针对嵌入式平台开发的稳定、高效的图形软件库，适用于任何图形 LCD 的操作应用，并可以输出高质量的无锯齿文字和图形，通过调用 emWin 提供的函数接口，开发嵌入式图形界面应用将变得简单而快捷。STemWin 是由 Segger 公司授权给 ST(意法半导体)公司的。在使用 ST 公司的芯片时，可以免费使用 STemWin。STemWin 只能在 ST 公司的芯片上运行的，在使用前需要进行芯片的 CRC 校验，否侧 STemWin 无法运行。使用 STemWin 创建的 GUI 桌面窗口如图 3-8 所示。

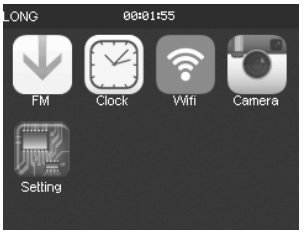

图 3-8　GUI 桌面

3.4.3　起始任务

在 main 中创建一个起始任务，然后在起始任务中创建消息队列、指纹任务、刷指纹任务、开门任务、门铃任务和拍照任务等。在任务执行体中执行触摸扫描任务和密码开锁扫描任务，任务在每次执行完成后延时 15ms 后再次执行。该任务是优先级最高的任务，其中密码开锁扫描任务是矩阵按键的扫描任务，用于开门密码的输入以及对密码的正确与否进行判断。密码分为常用密码和临时密码，确认密码后会将结果分为常用密码和临时密码上传至云端。临时密码在使用完成后会立即删除，只有一次开门机会。关于云端和临时密码的设置，请参考 Wi-Fi 任务。任务流程图如图 3-9 所示。

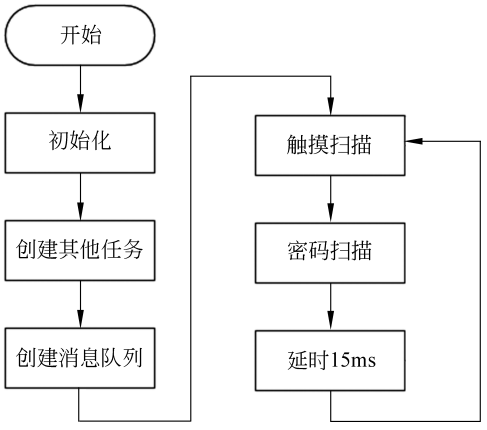

图 3-9　起始任务流程图

以下是矩阵按键的扫描函数的源代码。

```
/*****************************
  * @brief  扫描矩阵按键
  * @param  无
  * @retval 键盘输入结果
  *****************************/
u8 Matrix_Buttons_Scan(void)
{
    u8 xC =0;
    u8 yC =0;
    u16 DR =0;
    static u8 Matrix_Buttons[4][3];
    u8 NUM[4][3] ={{1, 2, 3},{4, 5, 6},{7, 8, 9},{10, 11, 12}};
    for(xC=0; xC<3; xC++)
    {
        DR =Line1_GPIO_PORT->ODR;
        DR =DR & (~(7 <<7));
        Line1_GPIO_PORT->ODR =DR | (1 <<(7 +xC));
        DR =Read_PIN();
        if ((DR >>9) & 1)
        {
            Matrix_Buttons[0][xC]++;
}
        if ((DR >>10) & 1)
        {
Matrix_Buttons[1][xC]++;
}
        if ((DR >>13) & 1)
        {
Matrix_Buttons[2][xC]++;
}
        if ((DR >>14) & 1)
        {
Matrix_Buttons[3][xC]++;
        }
    }
    for(yC=0; yC<4; yC++)
    {
        for(xC=0; xC<3; xC++)
        {
            if (Matrix_Buttons[yC][xC] >10)
            {
                Matrix_Buttons[yC][xC] =0;
                return NUM[yC][xC];
```

```
            }
        }
    }
    return 0;
}
```

3.4.4　刷指纹任务

指纹模块上有一个中断引脚，当指纹头上有手指按下时，中断引脚会输出一个高电平，在没有指纹时则为低电平。把主控芯片连接的引脚设置为中断模式，中断触发为上升沿触发。在有中断产生时，在中断服务函数中给刷指纹任务发送一个任务信号量，刷指纹任务会在进入任务时获取任务信号，如果获取不到，则一直等待，直到获取到任务信号量为止。当指纹任务在获取该信号量时，就会发送相应的指令给指纹模块，让指纹模块对图像进行采集和对比，最后指纹模块会把指纹模块的对比结果上传至主控芯片，主控芯片将对比结果通过 OLED 显示出来并上传至云端。如果指纹对比发送一个任务信号量给开门任务，则开门任务执行开门操作。指纹模块和主控芯片的通信使用的是串口，串口的配置参数为：波特率为 57 600，一个起始位，一个停止位，没有奇偶校验位，8 位数据位。刷指纹时会调用 void press_FR(void)函数。任务流程图如图 3-10 所示。

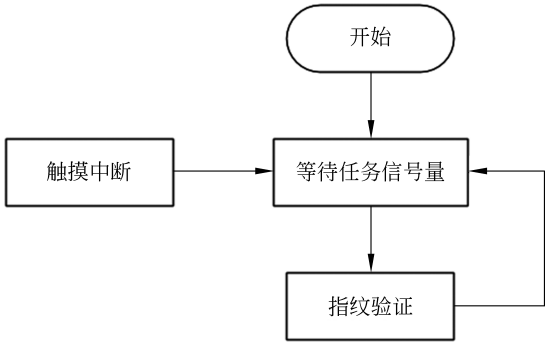

图 3-10　刷指纹任务流程图

```
/*********************************************************************************** *
 * 描述:刷指纹实现
 * 参数:无
 * 返回:无
*********************************************************************************** */
u8 ERR_Count =0;                          //错误计数
void press_FR(void)
{
    OS_ERR err;
    struct rtc_time time;
SearchResult seach;
```

```
u8 ensure;
    /* 获取当前时间 */
    RTC_TimeCovr(&time);
    ensure=PS_GetImage();
    if(ensure==0x00)                    //获取图像成功
    {
        ensure=PS_GenChar(CharBuffer1);
        if(ensure==0x00)                //生成特征成功
        {
            ensure=PS_HighSpeedSearch(CharBuffer1,0,300,&seach);

            if(ensure==0x00)            //搜索成功
            {
                ERR_Count =0;           //错误计数清零
/* 上报成功开锁的指纹编号和时间 */
                printf("指纹: %d 成功开锁-%d_%d_%d_%d\n",
seach.pageID,
                       time.tm_mon,
time.tm_mday,
time.tm_hour,
time.tm_min);
              /* 发送任务信号量给开门任务 */
                OSTaskSemPost(&AppTaskLockTCB,
                              (OS_OPT   )OS_OPT_POST_NONE,
                              (OS_ERR   *)&err);
            }
            else
            {
                ERR_Count++;            //错误计数
/* 上报指纹开锁失败时间和次数 */
                printf("指纹开锁第%d次失败-%d_%d_%d_%d\n",
ERR_Count, time.tm_mon,
time.tm_mday,
time.tm_hour,
time.tm_min);
                ShowErrMessage();       //显示错误信息

                if (ERR_Count >=ERR_MAX)
                {
                    OSTaskSemPost(&AppTaskShotTCB,
                                  (OS_OPT   )OS_OPT_POST_NONE,
                                  (OS_ERR   *)&err);
                }
```

```
            }
        }
else
        {
            ERR_Count++;                //错误计数
/* 上报指纹开锁失败时间和次数 */
printf("指纹开锁第%d次失败-%d_%d_%d_%d\n",
ERR_Count,
time.tm_mon,
time.tm_mday,
time.tm_hour,
time.tm_min);
            ShowErrMessage();           //显示错误信息

if (ERR_Count >=ERR_MAX)
            {
                OSTaskSemPost(&AppTaskShotTCB,
                          (OS_OPT   )OS_OPT_POST_NONE,
                          (OS_ERR   *)&err);
            }
        }
    }
    else
    {
        ERR_Count++;                    //错误计数
        /* 上报指纹开锁失败时间和次数 */
        printf("指纹开锁第%d次失败-%d_%d_%d_%d\n",
ERR_Count,
time.tm_mon,
time.tm_mday,
time.tm_hour,
time.tm_min);
        ShowErrMessage();               //显示错误信息
        if (ERR_Count >=ERR_MAX)
        {
            OSTaskSemPost(&AppTaskShotTCB,
                (OS_OPT   )OS_OPT_POST_NONE,
                (OS_ERR   *)&err);
        }
    }
}
```

3.4.5 开门任务

刷指纹任务会在进入任务时获取任务信号，该信号量有三个来源，分别是刷指纹任务、一键开门和密码开门。如果获取不到，则一直等待，直到获取任务信号量为止。当开门任务获取该信号量时，就会执行开锁函数，并在 OLED 上显示开门成功的提示。在执行完开锁函数后，该任务会延时 3s，然后执行关门任务，最后进入任务信号量的等待。任务流程图如图 3-11 所示。

3.4.6 门铃任务

门铃使用一个按键进行输入，当按键被按下时会产生一个中断，在中断服务函数中会给门铃任务发送一个任务信号量，门铃任务在获取到该信号量后，会调用开蜂鸣器的函数，蜂鸣器就是门铃。在调用蜂鸣器后，会延时 3s 再次调用关蜂鸣器的函数，即门铃会响 3s。任务流程图如图 3-12 所示。

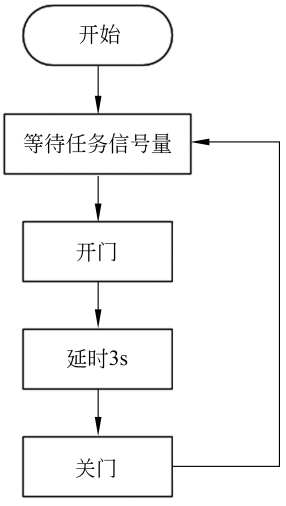

图 3-11 开门任务流程图

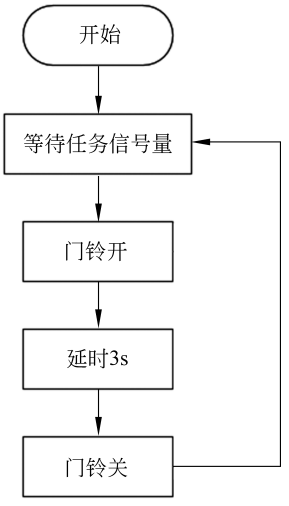

图 3-12 门铃任务流程图

以下是门铃开关的源代码。

```
macBEEP_ON();                    //蜂鸣器开
    macBEEP_OFF();               //蜂鸣器关
```

3.4.7 拍照任务

进入拍照任务后会获取任务信号量，当没有获取信号量时会一直等待，直到获取信号量。获取信号量后，拍照任务会调用 RTC_TimeCovr()函数的当前时间，并将照片的名字按如下规则命名：月_日_时_分_编号.bmp。bmp 类型的文件数据有文件头和数据两种，文件头包含文件大小、位图宽和位图高等数据，数据则是像素点的具体值，像素格式是 RGB565 格式。由于拍摄的照片将会保存在 TF 存储卡中，因此本设计还移植了文件系

统,使用文件系统的方式把图片保存到 TF 存储卡中。任务流程图如图 3-13 所示。

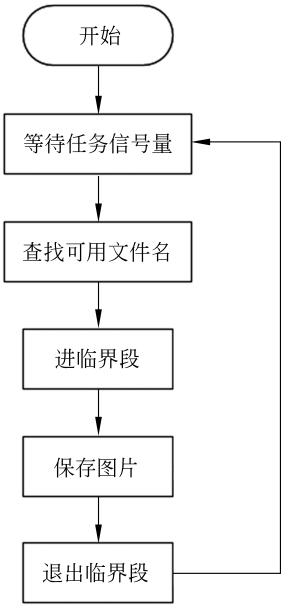

图 3-13　拍照任务流程图

以下是拍照任务的源代码。

```
static void AppTaskShot(void * p_arg)
{
    OS_ERR  err;
    CPU_TS  Ts;
    uint8_t i;
    char buf[20];
    struct rtc_time time;
    (void)p_arg;
    OS_DEBUG_FUNC();
    while(1)
    {
    //阻塞任务,直到拍照通知
        OSTaskSemPend((OS_TICK )0,                          //一直等待,直到有信号量
                      (OS_OPT )OS_OPT_PEND_BLOCKING,        //没有信号量可用就等待
                      (CPU_TS *)&Ts,           (OS_ERR *) &err);
        if (ERR_Count >=ERR_MAX)
        {
            Camera_Display();
        }
        /* 获取当前时间 */
```

```
        RTC_TimeCovr(&time);
        i =0;
        while(i<0xff)
        {
sprintf((char*)buf,
"1:/%d_%d_%d_%d_%d.bmp",
time.tm_mon,
time.tm_mday,
time.tm_hour,
time.tm_min,i);

            result=f_open(&file,(const TCHAR*)buf,FA_READ);

            if(result==FR_NO_FILE)break;
            else
                f_close(&file);
            i++;
        }
        /* 创建截图 */
        /* 向 TF 卡绘制 BMP 图片 */
        OSSchedLock(&err);
        Screen_Shot(0, 0, 320, 240, buf);
OSSchedUnlock(&err);

        /* 创建完成后关闭 FILE */
        if (ERR_Count >=ERR_MAX)
        {
            WM_DeleteWindow(hWinCamera);
        }
    }
}
```

3.4.8 Wi-Fi 任务

Wi-Fi 任务主要负责 Wi-Fi 的连接状态的检测以及对云端发送来的数据进行处理。

Wi-Fi 模组在没有连接 Wi-Fi 时会自动连接 Wi-Fi，当检测到连接 Wi-Fi 后，会发送连接云的指令，直到连接云成功为止。由于在 Wi-Fi 模组长时间不给云发送数据时云会断开与 Wi-Fi 模组的连接，因此需要在每隔一段时间就发送一个心跳包，心跳包的规定如下：每隔一分钟发送一次心跳包，内容是 HeartbeatPacket。

当 Wi-Fi 接收到数据时，该模组会对数据进行判断，如果数据是“SetTP******”(*为数字)，则说明是临时密码；当接收到该指令时，会设置一个临时密码，并记录当前时间戳，当密码在 5min 内没有被使用时，该临时密码会自动失效；该临时密码在使用后会立