

青 岛 科 技 大 学

专 科 毕 业 设 计

题 目 嵌入式智能家居控制终端的设计

指导教师 范玮

辅导教师

学生姓名 华政

学生学号 1808610605

信息科学技术学院 学院 计算机应用技术 专业 1866 班

2021 年 6 月 6 日

嵌入式智能家居控制终端的设计

摘 要

本文采用更科学的方法设计了一个嵌入式智能家居通用平台，利用网络通信技术、自动控制技术、安防技术、音视频技术，将家具和生活相关设施融为一体，提高家居的安全性、便捷性和舒适性，实现居住环境的环保节能，目的是加快智能家居产业的发展。

整个项目采用了大量的模块化设计思想，如存储模块、显示模块等，并采用 Git 来管理软硬件工程代码。现代集成开发软件大多内置了这一功能，便于多人协同开发。使用 CMake 构建软件项目，以便于跨平台软件移植。

在本课题中，我们详细讨论了常见嵌入式电子产品的设计过程，以及一些注意事项。对于这样一个平台，设计一个类似于软件包+桌面的软件运行结构是非常重要的。它类似于 DEB 和 APK，但要简单得多，本文中给出了其设计原型。

关键词：智能家居；Linux；嵌入式系统设计；图形用户界面；应用开发；电子产品项目。

DESIGN OF EMBEDDED SMART HOME CONTROL TERMINAL

ABSTRACT

In this paper, a more scientific method is used to design an embedded smart home general platform, which integrates furniture and living related facilities by using network communication technology, automatic control technology, security technology, audio and video technology, so as to improve home safety, convenience and comfort, and realize environmental protection and energy saving living environment, The purpose is to accelerate the development of smart home industry.

A large number of modular design ideas are used in the whole project, such as storage module, display module, and Git is used to manage the software and hardware engineering code. Most modern integrated development software has built-in this function, which is convenient for multi person collaborative development. CMake is used to build software projects to facilitate cross platform software transplantation.

In this project, we discussed in detail the design process of common embedded electronic products, as well as some matters needing attention. For such a platform, it is very important to design a software running structure similar to software package + desktop. It is similar to DEB and APK, but it is much simpler than that. The design prototype is given in this paper.

KEY WORDS: Smart home, Embedded system design, Operating system, Graphical user interface, Application development, Drive development, Embedded motherboard design, Electronic products project, Terminal

目录

前言.....	5
1 相关知识介绍.....	6
1.1 项目相关知识	6
1.2 开发环境.....	6
2 需求分析.....	8
2.1 项目需求分析	8
2.2 项目开发模型.....	8
3 系统概要设计.....	9
3.1 系统概要-软件方面	9
3.2 系统概要-硬件方面	9
4 系统详细设计.....	10
4.1 工程简述	10
4.2 供电 POWER SUPPLY	11
4.3 电源树 POWER TREE	18
4.4 CPU.....	19
4.5 外部 SPI FLASH.....	25
4.6 显示模块 DISPLAY	31
4.7 SD_EMMC.....	42
4.8 串口 UART	43
4.9 音频接口 AUDIO	43
5 系统测试.....	43
6 结论.....	44
参考文献.....	45
附录.....	47
致谢.....	52

前言

2020 年以后,各大厂商已开始密集布局智能家居,尽管从产业来看,还没有特别成功、特别能代表整个行业的案例显现,这预示着行业发展仍处于探索阶段,但越来越多的厂商开始介入和参与已使得外界意识到,智能家居未来已不可逆转。

对于嵌入式智能家居产品,我指的是硬件和软件。智能家居产品不仅要注重智能控制功能,还要有更强大的功能。它不仅应该有一个控制面板。即使它有一个分辨率很好的屏幕,它也应该具备一些电脑手机的功能,如游戏、多媒体等,另一方面,它可以大大缩短软件开发周期。可以运行 Linux 等软件的处理器越来越便宜。随着半导体制造技术的进步,本文所使用的处理器非常便宜。虽然升级处理器和外围设备会增加成本,但绝对值得。或许是时候让智能家居产品进军高性能处理领域了。

Linux 内核提供了强大的软件功能,例如 IPC 以及多线程等等,可以使得软件实现较为复杂的功能, Linux 内核允许动态静态加载驱动模块,使得系统可以支持多种多样的外设。在这种平台上进行开发无疑会变得十分方便。本文基本实现了一套完整的嵌入式软硬件平台,并且可进行拓展或裁剪,但大多数并未完善,需要时间来打磨和沉淀。

1 相关知识介绍

1.1 项目相关知识

- 1) U-Boot 的移植与适配
- 2) Linux Kernel 的移植与适配
- 3) 固件包的烧录
- 4) 嵌入式系统启动流程
- 5) Linux 下的驱动开发
- 6) LVGL 移植到 Linux Framebuffer
- 7) 基于 LVGL 的应用开发
- 8) 使用 CMake 进行 LVGL 的工程 Makefile 构建
- 9) MQTT 协议分析
- 10) 将设备连接至华为 IoTDA 云

1.2 开发环境

1.2.1 硬件环境

PC 机两台，开发板数块，各设备之间使用交换机组网。

936 焊台一部，WIFI 数字显微镜一台，PCB 恒温焊接加热台一部，热风枪焊台一部，锡膏钢网，890D 数字万用表一台，逻辑分析仪，3D 打印机一台。

1.2.2 软件环境

- 1) Windows 10、Ubuntu18.04 操作系统
- 2) Windows 下编程环境 Visual Studio + VS Code
- 3) Linux 应用及驱动开发环境 Source Insight
- 4) arm-linux-gnueabi 交叉编译工具链
- 5) make 工程编译链接工具
- 6) CMake 工程 Makefile 生成工具
- 7) docker 开发环境打包工具
- 8) Buildroot 嵌入式 Linux 系统编译工具
- 9) NFS 网络文件系统
- 10) VMware Workstation 虚拟机管理软件
- 11) MobaXterm 串口、SSH 调试工具
- 12) Filezilla SFTP 文件传输工具
- 13) Altium Designer PCB 工程开发软件

- 14) DigiPCBA AD 工程协同管理插件
- 15) Autodesk Fusion 360 PCB 外壳建模软件
- 16) Ultimaker Cura 3D 模型切片软件

Visual Studio 可使用 windows 接口进行应用开发,一些 GUI 库可以通过这些接口实现底层部分,在 Visual Studio 中创建项目,编译源程序,在 VS Code 中进行代码的编写工作,VS Code 提供了大量的插件使得开发效率提高,减轻一些编程工作。

Source Insight 常用于大型工程的管理,例如 Linux、U-Boot 这种大型项目,他可以快速查找变量及函数的定义,代码提示,文档比对,符号表等强大的功能,但是不能进行源码的编译。

Arm-linux-gnueabi 包含了一套 gcc 交叉编译工具链,常用于将 C、C++ 项目编译到 ARM 架构的嵌入式设备上。

Make 工具根据 Makefile 中的关系对源码执行操作,当工程结构变得愈来愈复杂时,需要经验丰富的工程师来进行编写。

CMake 工具根据工程下 CMakeLists.txt 中的内容对工程解析生成 Makefile,如今变得越来越常用,常用于跨平台的工程构建。

Docker 工具用来开发环境的导入和导出,其功能强大,可以让多个系统运行在容器之中,并且运行指定命令。

Buildroot 是一个简单、高效、易用的工具,可以通过交叉编译生成嵌入式 Linux 系统。NFS 网络文件系统可以使得本地计算机通过网络挂载其他设备上的文件系统。

Wmware Workstation 用来管理电脑上的虚拟机系统,其免去了在多台 PC 之前切换的烦恼。

Mobaxterm 是一个多功能的终端工具,他提供多种会话连接功能,SSH、VNC、SFTP、Serial 等,常用来连接虚拟机 shell 以及开发板串口。

Filezilla 使用 SFTP 协议在设备之间进行文件传输,其拥有直观的目录结构,方便物理机与虚拟机之间的文件传输。

Altium Designer 是一款专业的电子电路设计工具,这套软件通过把原理图设计、电路仿真、PCB 绘制编辑、拓扑逻辑自动布线、信号完整性分析和设计输出等技术的完美融合,为设计者提供了全新的设计解决方案,使设计者可以轻松进行设计,熟练使用这一软件使电路设计的质量和效率大大提高。

Autodesk Fusion 360 集成式 CAD、CAM、CAE 和 PCB 软件。Fusion 360 将设计、工程、电子产品和制造集成到单个软件平台。

Ultimaker Cura 是一款 3D 模型切片软件,切片完成之后导出 gcode 文件到 3D 打印机进行打印。

2 需求分析

2.1 项目需求分析

现阶段项目一般有如下几个方面的需求：

功能、性能、环境、开发进度、软件成本、资源使用、用户界面、安全保密、可靠性。

在这里只关注以下几个方面：

1) 功能：作为智能家居行业的一款产品，必然要体现其家居的功能，而本案例作为控制终端。则还要体现其控制的功能。对于家居而言，常用功能有例如，IP CAMERA 组成家庭摄像头，设备虚拟化界面。而作为智能家居产品中的控制终端，它是家庭中的管家，或者说大脑更加合适，它掌管着家庭中各个设备的数据走向，以及工作状态等。

3) 用户界面：界面设计应该符合现代审美标准，界面多使用高斯模糊及圆角特效，应用动画要流畅。

4) 可靠性：应用可开机自启动，应具备日志调试功能，代码易维护，且容易拓展。

2.2 项目开发模型

整个软件项目采用如图 2-1 模型进行开发：



图 2-1 项目开发模型

Fig.2-1 Charts Of Project development prototype

评估项目各个应用，策划应用涉及功能要求，以及是否有第三方开源项目支持，若有成熟项目，则优先采用。各应用开发周期不宜太长，最长不应超过 7 天。

在 Visual Studio 2019 中使用 Windows.h 中的接口实现与系统无关的界面设计（这部分 LVGL 官方已经给出了解决方案，可以直接用 Git 拉下来进行开发）。相比实机运行还是能省去不少的编译，拷贝，运行时间。而具体应用则要像采用 GTK 和 FLTK 的成熟 GUI 来实现具体功能。

将工程上传至 Linux 虚拟机中，使用 CMake 进行构建，交叉编译完成后，使用现有开发板挂载 NFS 网络文件系统运行可执行文件进行测试，项目小节完成推送到 Git 仓库。

硬件设计使用 Altium Designer 配合 DigiPCBA 完成。AD 中设计相关文件，使用 DigiPCBA 托管 PCB 项目，方便多设备开发以及多人合作，配合模块化设计，提高效率。

3 系统概要设计

3.1 系统概要-软件方面

软件方面，整个项目软件框图 3-1 如下：



图 3-1 项目软件框图

Fig.3-1 Charts of Project software block diagram

整个项目采用模块化的设计，分为：用户应用、系统应用、应用公共部分、UI_TOOLS 界面工具、UTILS 其他工具。各部分详细介绍如下：

- 1) 显示、输入管理器：使用管理器 Manager 机制来管理输入事件以及输出设备，增强程序可拓展性。
- 2) 应用公共部分：将应用的公共部分从中抽离出，作为单一应用执行，方便管理与配置。
- 3) UI_TOOLS：将用户界面自定义函数以及页面管理等单独处理做库函数。
- 4) UTILS：将系统库函数、LVGL 库、以及第三方库进行封装。
- 5) 用户应用：包含桌面、多媒体、游戏、语音助手等功能。

3.2 系统概要-硬件方面

PCB 工程结构如图 3-2 所示，按照模块化式的设计，功能划分明确，使整个工程一目了然。



图 3-2 项目硬件框图

Fig.3-2 Charts of Project hardware block diagram

4 系统详细设计

4.1 工程简述

笔者在撰写这篇文章时，整个 PCB 工程经过了多次修订，尽管如此，各模块核心作用并不会发生太大变化。

4.1.1 PCB 工程框图

这里使用了 Altium Designer 的页面符功能，通常在第一个原理图文件中将各模块功能细分，用信号线表示各模块之间的作用关系以及连接性，如图 4-1 所示：

中间部分为 CPU 模块，其周围则是各个外围功能模块，例如显示模块、flash 模块等。如此，在以后对功能进行拓展以及裁剪时将变得非常方便，同时，将该工程提供给其他工作人员审阅时，通过该文件也可以对整个工程做一个大体上的了解。

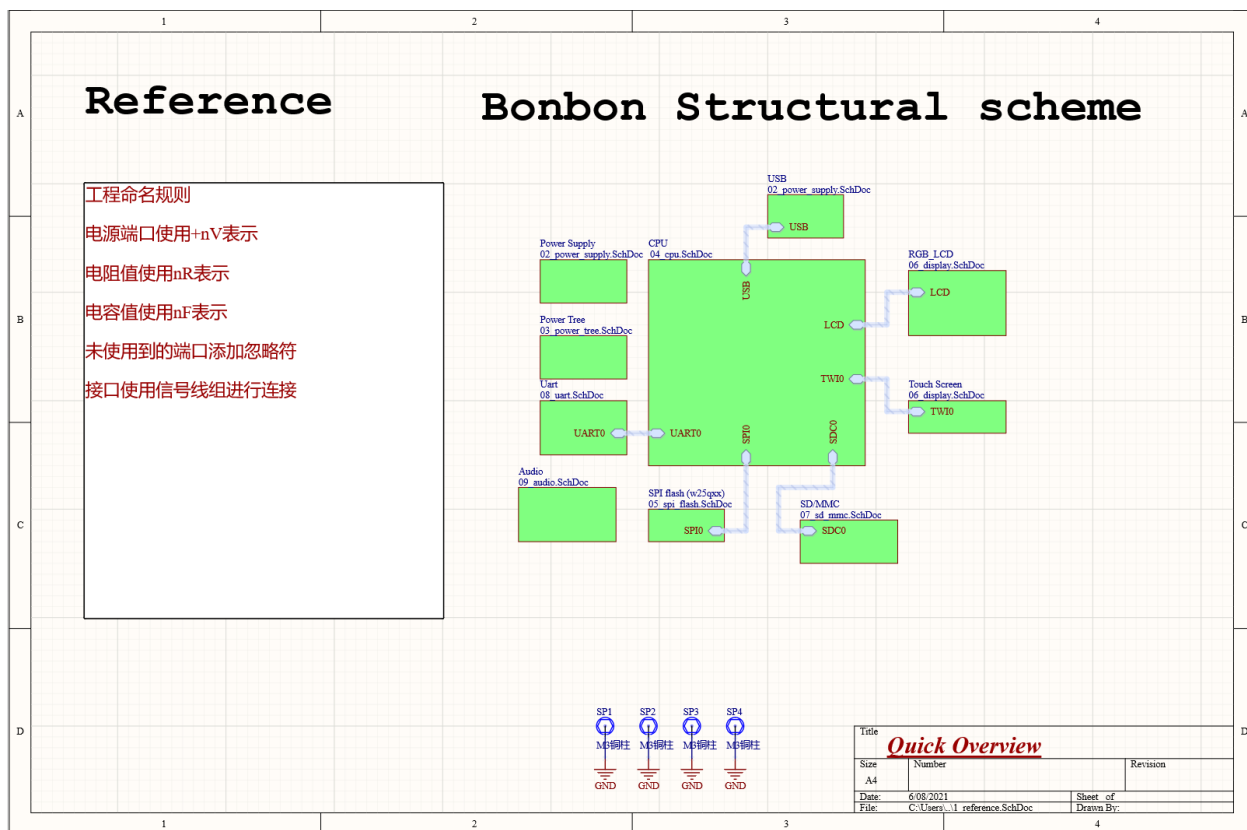


图 4-1 PCB 结构方案

Fig.4-1 Charts of PCB structural scheme

4.2 供电 Power Supply

对于供电部分，到目前为止我做了两版设计，在第一版中我使用 Micro USB 作为电源接口，使用一个 DC-DC 芯片 EA3036 对 CPU 所需要的其中 3 种电压值进行输出，如下图所示。但考虑到接口兼容性等一些问题的，我在第二版中使用了 USB Type-C 型作为电源接口，使用 3 路 JW5211 进行 3 种电压值的输出，尽管如此，其 BOM 成本以及焊接难度相比最初版本还是降低了不少。第一版如图 4-2 所示，第二版如图 4-3 所示。

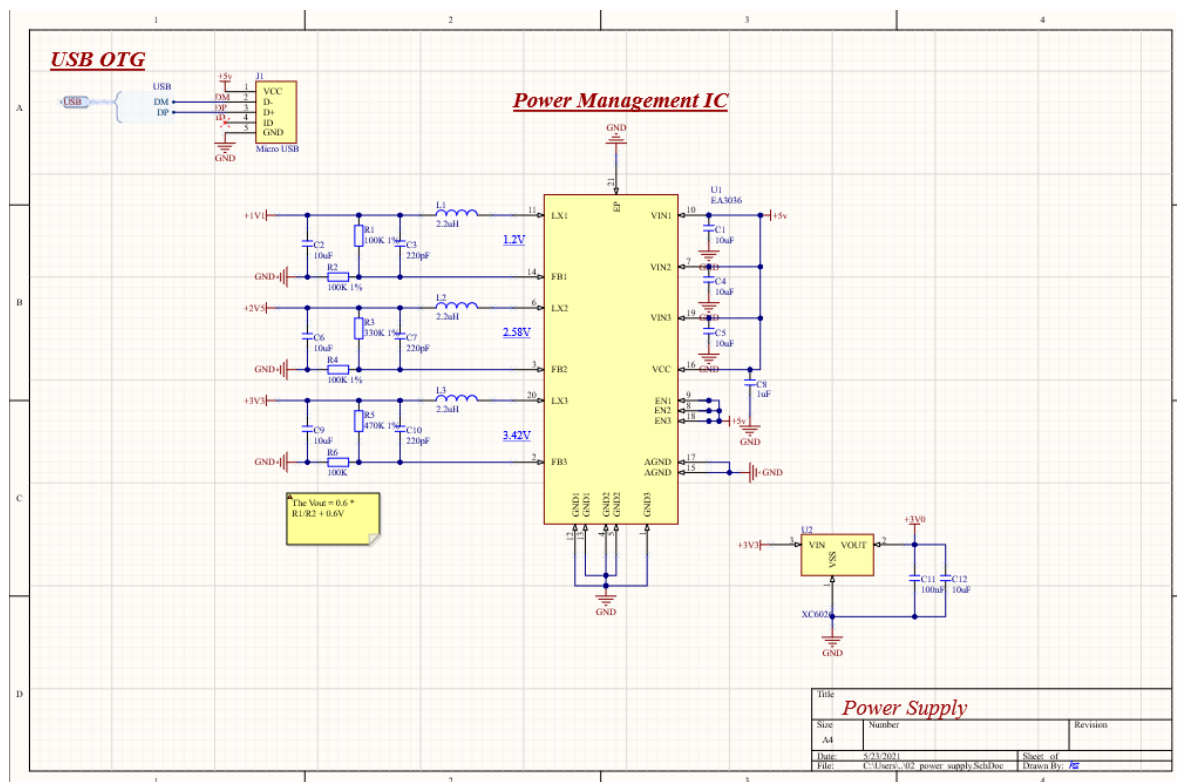


图 4-2 PCB 供电方案一
Fig.4-2 Charts of PCB power supply scheme I

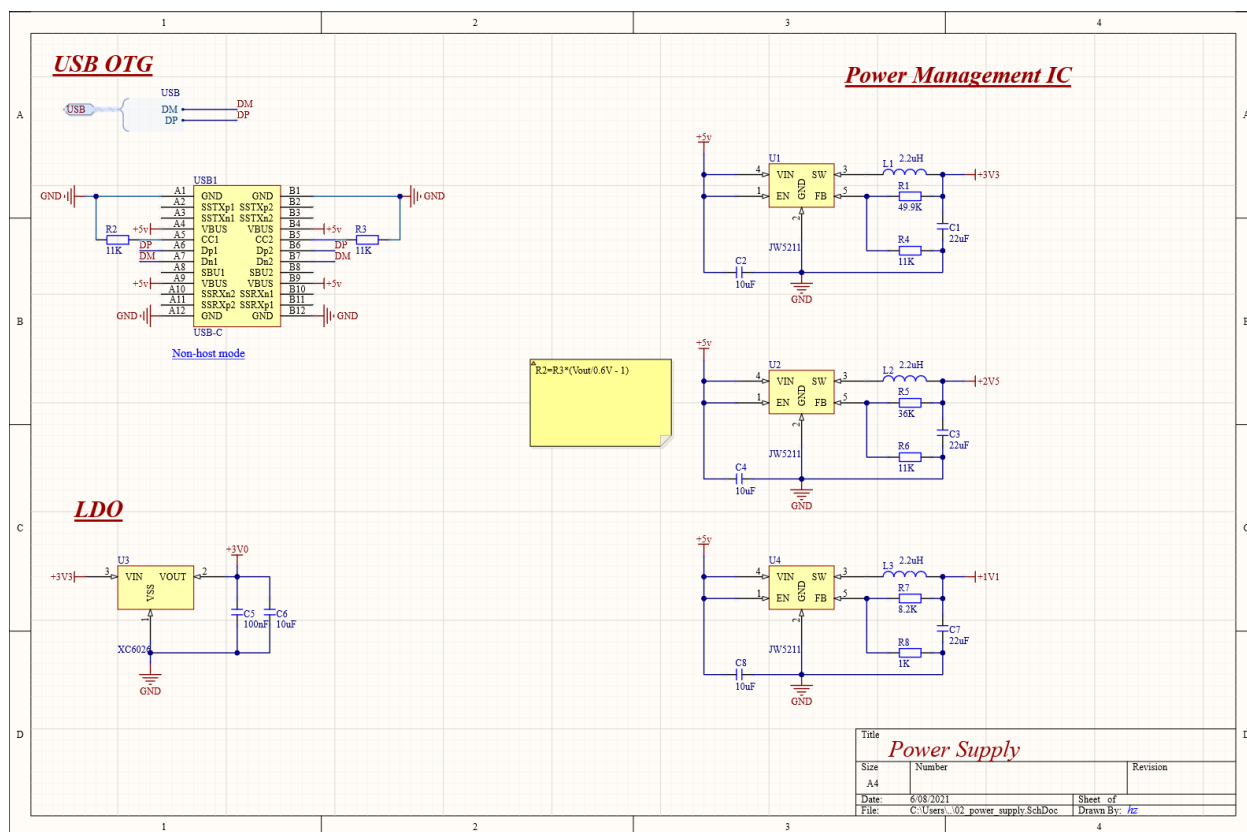


图 4-3 PCB 供电方案二
Fig.4-3 Charts of PCB power supply scheme 2

4.2.1 USB 接口

我使用 USB 接口作为+5V 电源输入源，将处理器 DM、DP 引脚连接至 USB 数据引脚上，方便后续使用 sunxi-tools 烧写固件。

USB-TypeC 接口定义如图 4-4 所示

A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1
GND	RX2+	RX2-	VBUS	SBU1	D-	D+	CC	VBUS	TX1-	TX1+	GND
GND	TX2+	TX2-	VBUS	VCONN			SBU2	VBUS	RX1-	RX1+	GND
B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12

图 4-4 USB TypeC 接口定义
Fig.4-4 Charts of Definition of USB typc interface

其中 D-、D+引脚用于兼容 USB 2.0，RXn、TXn 是高速数据传输线，VBUS 为电源总线，GND 为地线，如果要确定设备是主机或者从机，需要根据连接在 CC 和 VCONN 引脚上的电阻来确定，当 CC 和 VCONN 全部通过下拉电阻接地时，此时 USB TypeC 为从机接口，反之则为主机。

4.2.2 DC-DC EA3036 电源管理 IC

EA3036 是一款 3CH 电源管理 IC，适用于由一个锂离子电池或直流 5V 适配器供电的应用。输出电压范围为 0.6V 到 V_{in} ，最大持续负载电流 2A，应用场景有智能手机、IP Camera、数字相机等。芯片封装图如图 4-5 所示

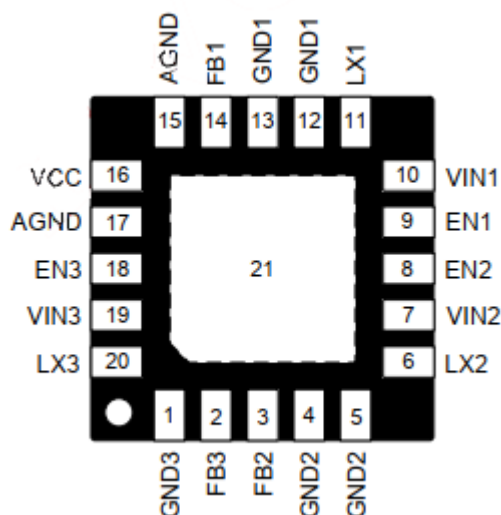


图 4-5 EA3036 引脚封装图

Fig.4-5 Charts of EA3036 pin package diagram

EA3036 拥有三路电压调节器，每个稳压器的输出电压都可以通过一个电阻分压器（例如 R1, R2）进行设置。调节器外围电路结构如图 4-6 所示：

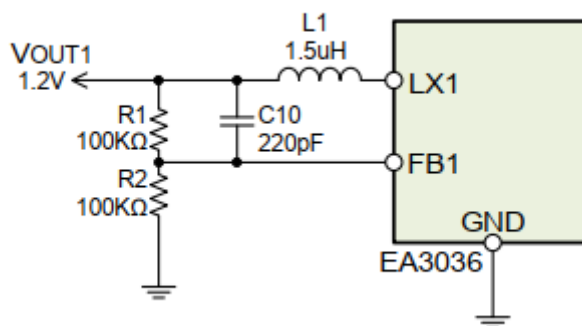


图 4-6 调节器外围电路结构

Fig.4-6 Charts of Peripheral circuit structure of regulator

通道输出电压计算公式：

$$V_{out} = 0.6 \times \frac{R1}{R2} + 0.6V \quad (4-1)$$

手册中列出了通用输出电压以及对应的 R1, R2 电阻值参考, 如图 4-7 所示

Output Voltage	R1 Resistance	R2 Resistance	Tolerance
3.3V	510K Ω	110K Ω	1%
1.8V	200K Ω	100K Ω	1%
1.5V	150K Ω	100K Ω	1%
1.2V	100K Ω	100K Ω	1%

图 4-7 常用电阻值

Fig.4-7 Charts of Common resistance value

F1C100S 芯片手册第 18 页 5.2 节推荐工作条件(Recommended Operating Conditions) 如图 4-8 所示

Symbol	Parameter	Min	Typ	Max	Unit
T _a	Ambient Operating Temperature[Commercial]	-20	-	85	°C
VCC-IO	Power Supply for I/O	3.0	3.3	3.6	V
AVCC	Power Supply for Codec	2.5	2.8	3.1	V
TV_AVCC	Power Supply for TV	3.0	3.3	3.6	V
VDD-CORE	Power Supply for Internal Digital Logic	1.0	1.1	1.2	V
UVCC	Power Supply for USB	3.0	3.3	3.6	V
VCC-DRAM	Power Supply for DDR1	2.3	2.5	2.7	V

图 4-8 推荐工作条件

Fig.4-8 Charts of Recommended working conditions

F1C100S 需要如表 4-1 参数的电压值

表 4-1 输入电压值

Tab.4-1 Tables of Input voltage value

电压类型	电压值
VCC-IO	3.3V
AVCC	2.5V~3.1V
TV_AVCC	3.3V
VDD-CORE	1.1V
UVCC	3.3V
VCC-DRAM	2.3~2.7V

由上述信息可以得知，EA3036 三路转换器分别需要输出 1.1V、2.6V、3.3V，3V 则由 XC6206302 芯片转换输出，根据公式

$$V_{out} = 0.6 \times \frac{R1}{R2} + 0.6V \quad (4-2)$$

计算出第一组 $R1=100K$ $R2=100K$ ，套入公式得 $V_{out}=1.2V < 1.21V$ ，在允许偏差范围之内

第二组 $R1=330K$ $R2=100K$ ，得 $V_{out}=2.58V$ 符合要求

第三组 $R1=470K$ $R2=100K$ ，得 $V_{out}=3.42 < 3.63V$ ，在允许偏差范围之内

根据 EA3036 芯片手册输出电感选择（Output Inductor Selection）小节说明，输出电感的选择主要取决于通过电感的纹波电流 ΔIL ，较大的 ΔIL 将导致较大的输出电压纹波和损耗，但用户可以使用较小的电感以节省成本和空间。相反，较大的电感可以得到较小的 ΔIL ，输出电压纹波和损耗越小，但这会增加空间和成本。电感大小计算公式：

$$L = \frac{V_{pwr} - V_{out}}{\Delta IL \times F_{sw}} \times \frac{V_{out}}{V_{pwr}} \quad (4-3)$$

对于大多数应用，1.0uH 至 2.2uH 电感器适用于 EA3036。

通过上述信息得知，虽然用电感越大越好，但是考虑成本以及空间，使用官方推荐值的最大值 2.2uH。

根据 EA3036 芯片手册 PCB 布局建议（PCB Layout Recommendations）小节说明，收集到以下信息：

- 1) 建议使用 4 层 PCB 布局，并在顶部放置 LX 平面和输出平面，将 Vin 平面放置在内层中。
- 2) 顶层 SMD 输入和输出电容器的地应该使用过孔连接到内部接地层和底部接地层。
- 3) 应当使用过孔将 AGND 直接连接到内部接地层。
- 4) 大电流走线应适当加宽。
- 5) 将输入电容尽可能的放置在靠近 VINx 引脚的位置，以降低噪声干涉。
- 6) 使反馈路径（Vout 到 FBx）远离噪声节点（例如 LXx）。LXx 是一个高电流噪声节点。通过使用短线和宽线来完成布局。
- 7) 顶层裸露焊盘接地层应连接到内部接地层通过使用多个过孔来改善散热性能，并形成底部接地层。

4.2.3 XC6206 系列——高精度，低功耗 LDO

XC6206 系列芯片在提供大电流的同时压降极小，XC6206 由限流电路，驱动晶体管，精密基准电压和误差校正电路组成。

输出电压在内部通过激光微调技术实现，它可以以 0.1V 的递增量在 1.2V 到 5V 调整。模块原理图如图 4-9 所示：

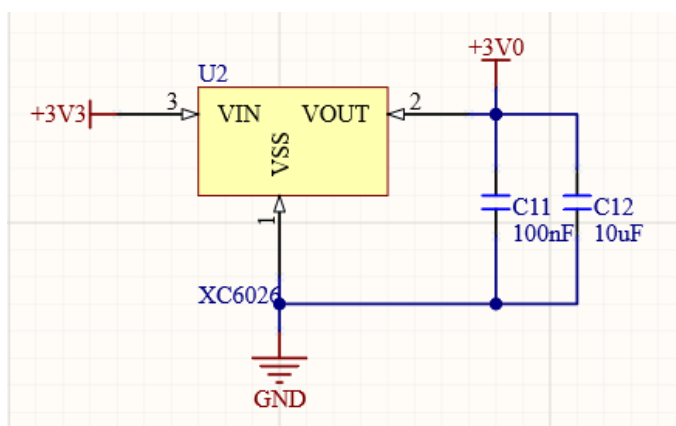


图 4-9 XC6206 原理图

Fig.4-9 Charts of Xc6206 schematic diagram

引脚定义如图 4-10 所示

■ PIN ASSIGNMENT

PIN NUMBER				PIN NAME	FUNCTIONS
SOT-23	SOT-89	USP-6B	TO-92		
1	1	2	1	Vss	Ground
3	2	4	2	VIN	Power Input
2	3	6	3	VOUT	Output
-	-	1, 3, 5	-	NC	No Connection

图 4-10 XC6206 引脚定义

Fig.4-10 Charts of Xc6206 pin definition

XC6026 系列拥有多种型号，例如 XC6206P122、XC6206P302 等，但我并未在芯片手册中发现有关命名规则的说明，根据图 4-11，不难推断出其命名规则。

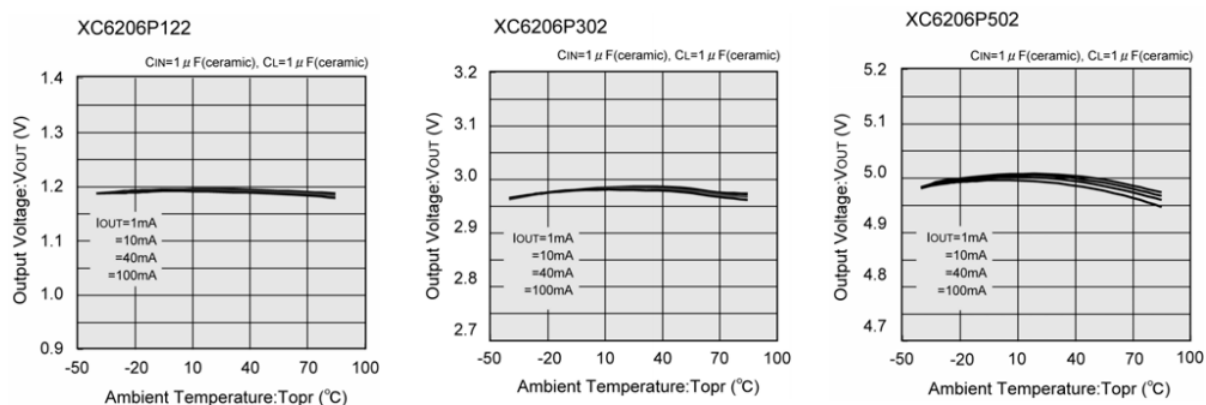


图 4-11 XC6206 温度电压曲线
Fig.4-11 Charts of Xc6206 temperature voltage curve

型号 P122 输出 1.2V 电压值，型号 P302 输出 3.0V 电压值，型号 P502 输出 5V 电压值
综上所述，XC6206P 之后 2 位数即输出电压值，这里需要 3V 电压，所以应该选择 XC6206302 型号。

4.3 电源树 Power Tree

随着产品集成度的提高，PCB 上各种型号芯片总功耗的增大，产品的电源供电设计变得越来越具有挑战性，如何快速把一个产品复杂的电源连接关系清晰地用图形显示出来，成为至关重要的问题。

模块原理图如图 4-12 所示

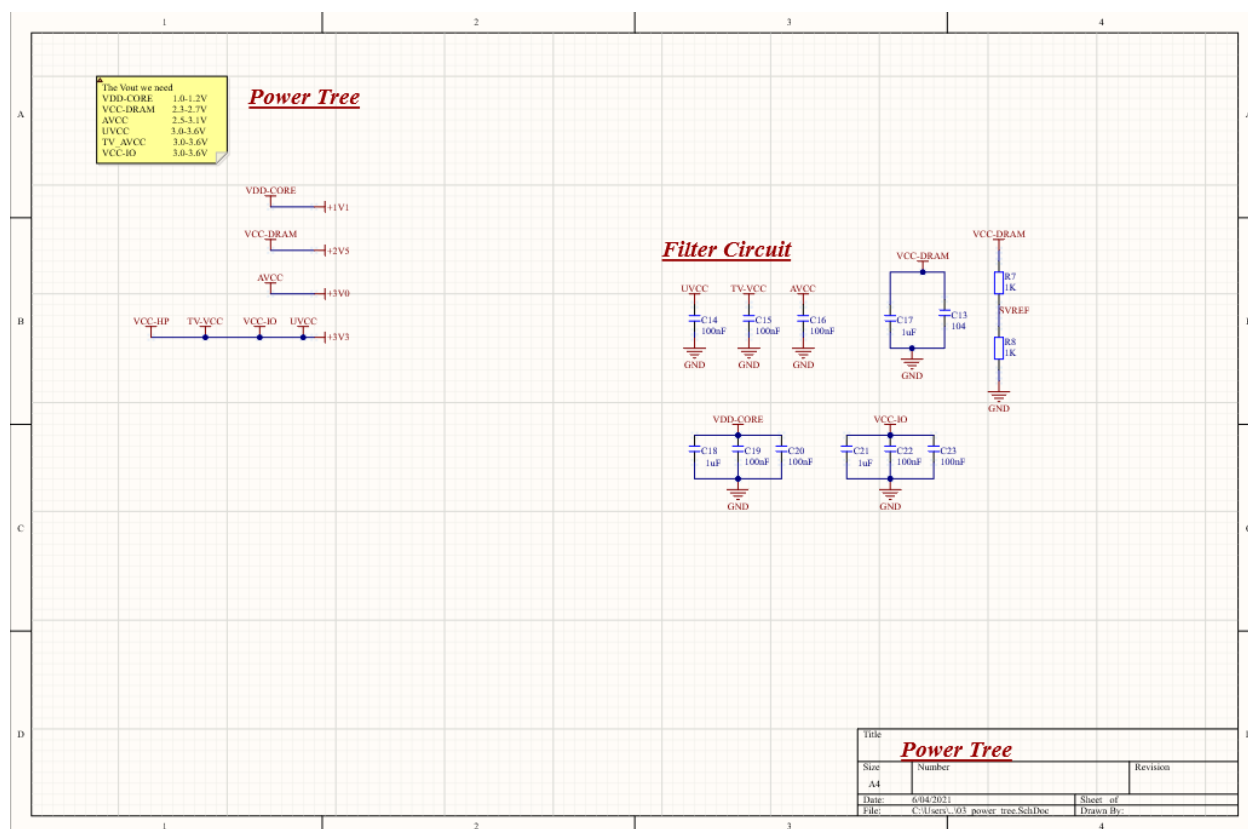


图 4-12 电源树原理图

Fig.4-12 Charts of Schematic diagram of power tree

在电路设计中，PCB 板第三层为 3.3V 电源层，因为板上大多数元件需要 3.3V 电源输入，将其作为单独电源层可以简化电源布线工作。另外在电路设计初期，要对 PCB 进行扇孔，其主要目的有两个，打孔占位，减少回流路径。打孔占位是为了防止不打孔导致后面走线的时候很密集无法打孔，增加回流路径。例如 GND 孔，就近扇孔可以做到缩短路径的目的。

4.4 CPU

模块原理图如图 4-13 所示

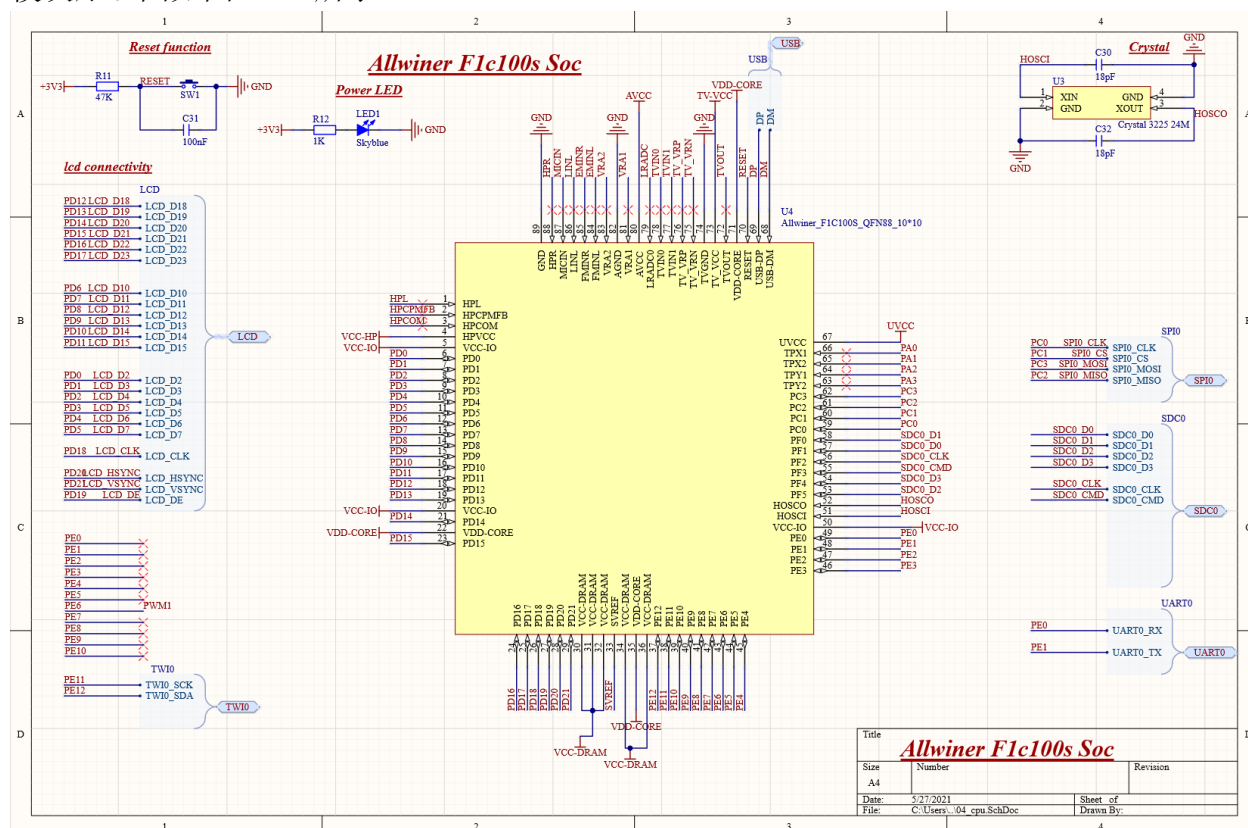


图 4-13 CPU 原理图

Fig.4-13Charts of Schematic diagram of power tree

4.4.1 CPU

对于 CPU 部分，我从 F1 系列中选择了两款芯片，在最初版本中使用的是 F1C100s，在后面的版本中随着软件体积的增大需要更换为 F1C200s，F1C200s 拥有更大的内存，可以实现更加复杂的软件功能。

F1C100s 处理器基于 ARM 9 CPU 架构功能集成度高，支持全高清视频回放，包括 H. 264、H. 263、MPEG 12/4 解码器。集成音频解码器和 I2S/PCM 接口为终端用户提供良好的音频体验，TV-IN 接口通过连接到摄像机等设备实现视频输入，TV-OUT 接口通过连接到电视设备实现视频输出。

为了降低 BOM 成本，F1C100s 内置 DDR1 内存，并配备了通用外设，例如 USB OTG, UART, SPI, TWI, TP, SD/MMC, CSI etc. F1C100s 完全支持 Android、Linux 等主流操作系统的各种应用。

4.4.2 芯片框图

F1C100s 芯片框图如图 4-14 所示，更多相关信息如图 4-15 所示

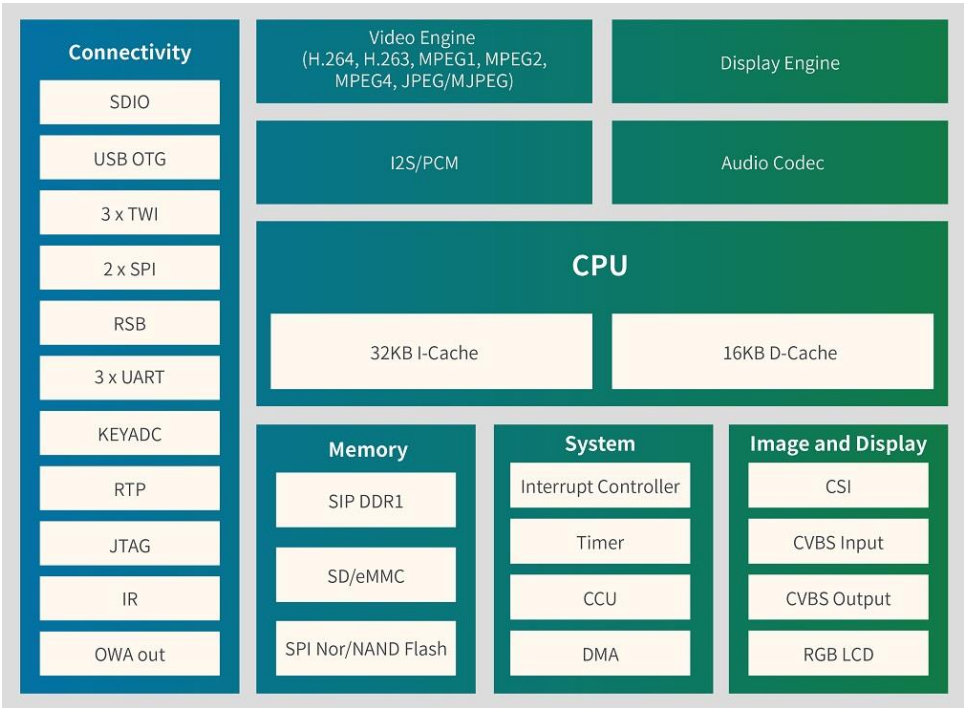


图 4-13 CPU 芯片框图
Fig.4-13 Charts of CPU chip diagram


<div><div>F1C100s</div><div><div>Manufacturer</div>Allwinner</div><div><div>CPU</div>ARM926EJ-S @ 533MHz</div><div><div>Memory</div>32MB Embedded DDR</div><div><div>GPU</div>Custom 2D-only</div><div><div>VPU</div>Unknown</div><div><div>Connectivity</div></div><div><div>Video</div>LCD, CVBS</div><div><div>Audio</div>DAC</div><div><div>Storage</div>SD/MMC (SD v2.0, eMMC V4.41)</div><div><div>USB</div>USB 2.0 OTG</div><div><div>Website</div>Product Page</div></div>

图 4-15 CPU 详细信息
4-15 Charts of CPU details

4.4.3 Boot 启动方式 (Boot type)

根据芯片手册可以得知 F1C100s 支持 3 种启动方式，分别如下：

- 1) 从 SPI Nor/Nand Flash 启动
- 2) 从 SD/TF 卡启动
- 3) 通过 USB 接口下载程序到内存中启动

对于启动方式的部分，sunxi 官网提供了大量的资料，以及篇幅颇长的说明，其中大部分都是对于 A 以及 H 系列而言的，但同样适用于 F 系列。所有已知的 Allwinner SoCs 都能从 SPI flash 启动，其拥有最低启动优先级，仅在 SD、Nand、eMMC 启动失败之后才会尝试从 SPI flash 启动。在削减物料成本的时候可以将引导加载固件存储在 SPI flash 中，不需要增加 Nand 以及 eMMC 存储设备。

SPI flash 的大小可以从 8Mb 到 128Mb 区间内选择，使用 8Mb 大小的 flash 芯片时，可以存储一个方便使用裁剪过的 BootLoader，并在此基础上添加一些高级功能，使其可以应对板上其他设备的初始化等工作，对于这点而言，U-Boot 就是最适合的选择了，同时这种方案也是我所选择使用的。而使用 128Mb 大小的 flash 芯片，通常可以将 boot loader、设备树文件、Linux 内核以及根文件系统放到其中，仅仅使用一块 flash 芯片作为整个软件系统的载体。

在这个项目中，我使用如图 4-16 所示这种比较常用的存储结构。

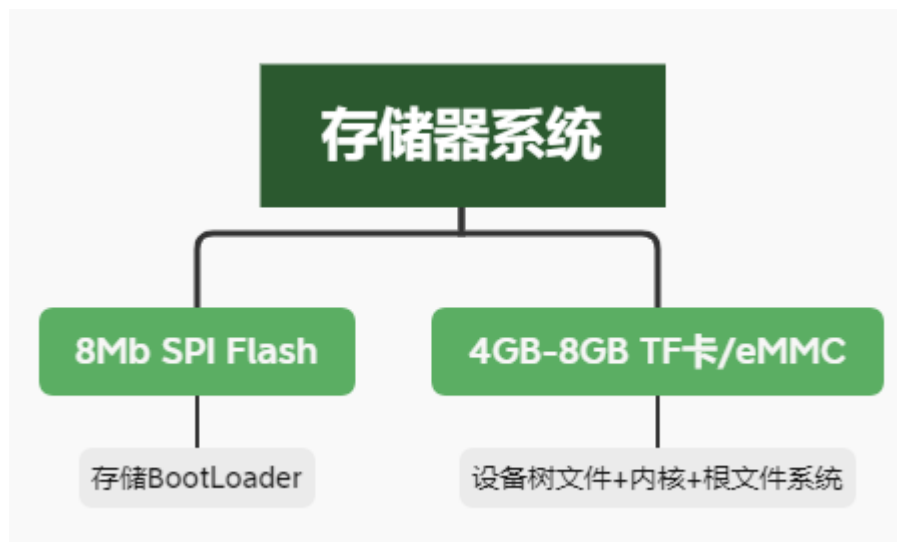


图 4-16 存储器结构

Fig.4-16 Charts of Memory structure

其实 8Mb flash 芯片完全可以省掉，将 BootLoader 也放入 TF/eMMC 中。按照官方的

做法，在进行系统级别更新时，无论是对于 SPI flash 还是 TF/eMMC，都要进行覆盖式的擦除和写入，例如在 SPI flash 上进行系统更新时，需要使用 dd 工具创建一个与 flash 大小相同的文件，然后分别按照各分区偏移值再次使用 dd 工具写入文件中，然后使用 sunxi-fel 工具对 flash 芯片进行烧写，对于 TF/eMMC 亦是如此。

我使用的方式存储结构就是如上图所示，我可以将 BootLoader 单独放在 flash 芯片中，将预处理过的 boot.cmd 文件放到 TF/eMMC 的第一分区分区中，将设备树文件和内核、根文件系统放到第二分区分区中，在引导阶段，BootLoader 根据 boot.cmd 中的配置信息，将设备树文件以及内核等读取到内存中，进行一系列操作后，引导内核启动。在需要进行系统更新时，例如更换外设，升级内核，只需要替换/boot 分区中的设备树或者内核文件即可。

4.4.4 接口介绍 (Connectivity)

作为一款智能家居物联网方向的嵌入式产品，其必然少不了大量的用户交互功能，我在设计这块板子的时候大体上用到了如下接口：

1) LCD 接口

显示器件作为是智能家居产品中最重要的其中一个组件，其承载了所有需要输出给用户的信息，无论是在系统的哪个阶段。在这里使用的是 TFT RGB 标准接口 40Pin LCD。

2) TWI 接口

在整个项目中，使用到 TWI 接口的是 NS2009 四线电阻式触摸屏驱动芯片，这种触摸屏需要贴在 LCD 屏幕上方，其尺寸要与 LCD 屏幕相当，通过 4 根排线连接到 LCD 屏幕触摸数据引脚，然后与 NS2009 输入 ADC 引脚连接，NS2009 收集到数据后，在 Linux 中可通过驱动使用中断或者查询方式收集 NS2009 上报的数据，更多 TWI 外设还在进一步的计划中。

3) SPI 接口

SPI 接口作为嵌入式系统中的常用接口，在这里一共用到了两组，其中一组即 SPI0 连接外部 flash，另外一组则是使用 SPI1 接口连接 ESP 模块，我从 ESP 文档中得知，ESP 模块可以工作在串口、SPI、SDIO 模式，让其工作于 SPI 或者 SDIO 模式，同时在 Linux 实现其设备驱动就可以当做 wifi 网卡使用，而更高级的 ESP 模块则又集成了蓝牙功能，可以实现更多更加复杂的软件功能。

4) SDC 接口

F1C100s 拥有两组 SDC 接口，其中 SDC0 是 4bit 接口，SDC1 是 1bit 接口。SDC 接口用于连接 SD/TF 卡以及 eMMC。我在最初版本的设计中使用了 SDC0 接口连接 TF 卡作为大容量存储设备，但是这种可卸载式的存储设备上在一等程度上影响了整个系统的运行稳定性，所以在后续的版本中应该修改为不可卸载式的 eMMC。

5) USB 接口

F1C100s SoC 提供了 DM、DP 接口，用于直接连接到外部 USB 接口，我在最初设计的版本中使用标准 Micro USB 2.0 接口，但在后来的版本中全面替换成了 USB Type-C 24pin 接口，无论是功能、载流和信号传输方面，其都要远优于 Micro USB 接口。

6) UART 接口

串口作为重要的调试工具，系统在测试阶段需要频繁的使用串口，使用串口进行调试时，系统往往不需要过多外设。串口在这个项目中并不是很重要的部分，在这里仅仅使用了一组 2x2 排针吧 UART0 接口引出，使用外部 USB 转串口设备和 CPU 进行串口的通信。

4.4.5 外部晶振时钟（Crystal）

晶振作为 CPU 的外部时钟输入源，也是整个系统的一个重要组件，根据 F1C100s 数据手册 19 页 5.4 节振荡器电气特性（Oscillator Electrical Characteristics）中的说明，一个频率为 24MHz 的晶振需要连接引脚 OSC24MI 和 OSC24MO 之间。并联在晶振上的电容大小范围在 5~7.5pF，典型大小为 6.5pF。

F1C100s 拥有强大的 CCU 即 Clock Control Unit，时钟控制单元作为嵌入式 MPU 中最终要的部分，其在 MPU 中扮演者心脏的身份，这个单元用来生成系统所需要的多种时钟信号。

在 F1C100s 中，CCU 提供寄存器来对 PLL 进行编程并控制大部分时钟的生成、分频、分配、同步和门控。CCU 输入信号参考自外部时钟源即 24MHz 晶振。CCU 的输出对象主要是系统中的其他模块。

在 ARM9 架构中，ARM 拥有类似于 PC 机上的南北桥式的外设总线设计，分别为 AHB、APB 总线，其中 AHB 总线类似于 PC 机中的“北桥”，该总线上连接的大都属于高速设备，其往往拥有很高的带宽以及频率，其接口例如 LCD、SRAM、CSI、SDC 等等，而 APB 总线上连接的大都属于低速设备，其接口例如 UART、TWI、INTC 等。如图 4-17 所示。

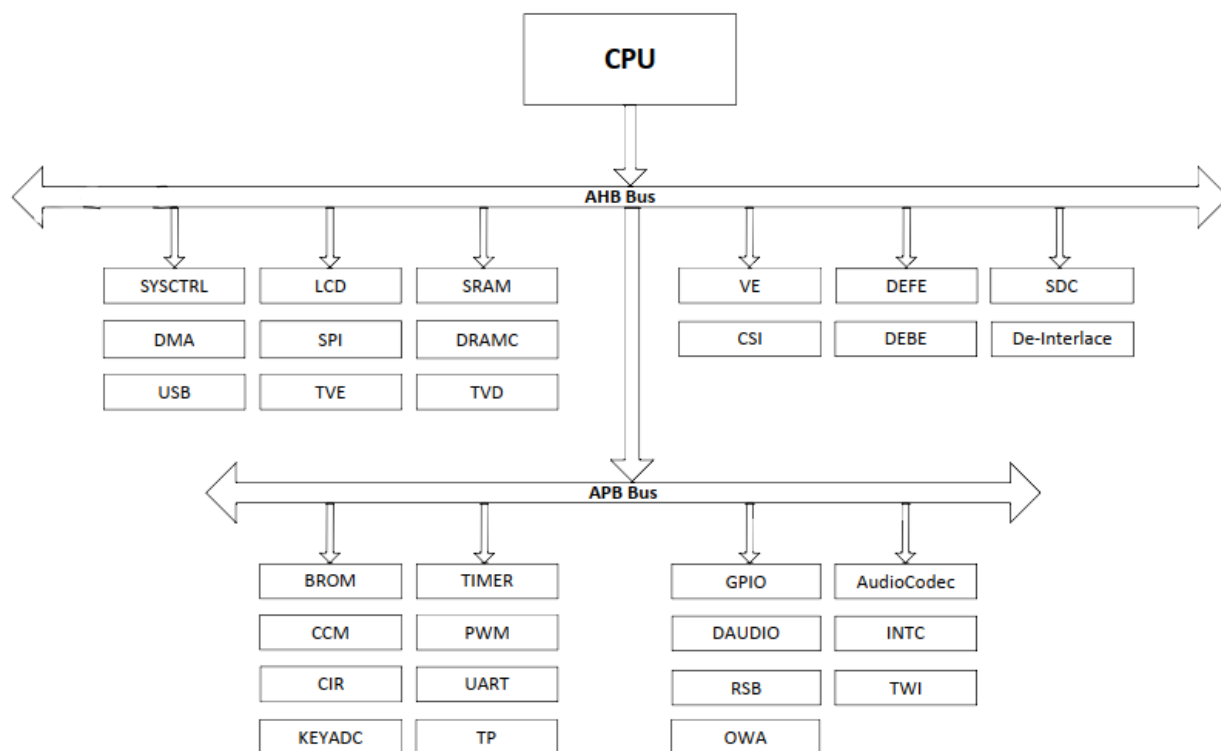


图 4-17 ARM9 总线架构

Fig.4-17 Charts of ARM9 bus architecture

4.5 外部 SPI Flash

在这个项目中 SPI Flash 的大体作用已经介绍的差不多了，所以有关在系统中 SPI 的作用就不在耗费篇幅了，这里只说明下 SPI 模块设计原理以及是如何工作的。

4.5.1 模块原理图

如图 4-18 所示。

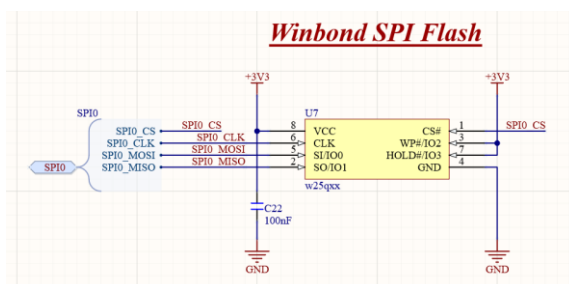


图 4-18 外部 SPI Flash 模块

Fig.4-18 Charts of External SPI flash module

4.5.2 模块命名问题

为什么模块要命名为 Winbond SPI Flash 呢？其实笔者学习外部 Flash 式第一块 Flash 芯片就是 64Mb 大小的华邦 w25q64，其作为外部 Flash 的典型器件，基本涵盖了外部 flash 的所有内容，本项目最初计划也是使用华邦的 flash 芯片，但是最近一段时间各种芯片涨价严重，华邦的芯片预算已经超过了原来的计划，所以，使用更加便宜的国产芯片来进行替换，成了大家最热门的话题。我比较看好的是国产芯天下厂家生产的 XT25F128B 闪存芯片，相比华邦 128Mb 闪存芯片，其虽在传输速度上略逊一筹，但是其功能在保持兼容性的情况下做了一些探索性的尝试，同时价格则相当乐观，样品价格仅 2.8¥一片。

4.5.3 SPI Flash 工作模式

我使用的是 Flash 芯片的标准 SPI 模式，但其提供了 Dual/Quad I/O SPI 模式，使用 Fast Read 指令提供更高的传输速率的情况下，则需要使用更多的 IO 接口，按照简单易用的原则，在这里使用标准 SPI 模式，暂时不使用 Dual 或是 Quad 模式 SPI。

4.5.4 SPI Flash 通讯模式

这里以读取 Flash 芯片为例，该 SPI Flash 在拉低 CS 引脚后会将指令码和地址存入移位寄存器中，并且在上升沿将其锁存，当所有地址位发送完毕后，地址相对应的数据字节将会从最高有效为（MSB）开始在每个 CLK 引脚的下降沿发送到 MISO 引脚。在每个数据字节发送完毕后，地址将自动递增，这意味着 Flash 芯片允许读取连续的数据流。

4.5.5 SPI Flash JEDEC ID

对 SPI flash 操作时, 通常先进行读 Flash JEDEC id 进行测试, 通常发送 9F 指令进行读取, SPI flash 的读取操作并不涉及 flash 寄存器的设置, 在发送完 9F 指令后, 连续读取 3 个字节, 第一个字节表示 Manufacture ID 厂商 ID, 第二个字节表示 Memory Type 闪存类型, 第三个字节表示 Capacity 容量。对于 XT25F128B 这款芯片, 其完整 JEDEC ID 为 0b, 40, 18。这里为什么要单独说一下读 JEDEC ID 操作呢? 原因是 u-boot 在使用 SPI Flash 启动的时候, 会先读取 flash 芯片的 JEDEC ID, 然后去跟 spi_flash_ids.c 文件中的预先定义好的 Flash JEDEC ID 进行逐一比对, 若无匹配项则 u-boot 终止启动。XT25F128B 作为一款新型 flash 芯片, 在目前版本的 flash_ids 中并没有进行定义, 这方面的内容我们留到 U-Boot 修改小节中继续讨论。

4.5.6 SPI Flash 飞线连接导致的问题

我在最初版本中为了方便烧写 u-boot 到 Flash 当中, 将 Flash 芯片的 1、4 引脚用拨片式开关连接了起来, 用来方便使 CPU 进入 FEL 模式, FEL 是一个包含在所有 Allwinner SoC 上的 BootROM 中的底层子程序, 它用于使用 USB 接口对 SoC 进行初始编程和恢复。这个细节导致了我后来在 uboot 源码上所做的修改一律不生效, 我将在 U-Boot 修改小节中继续讨论这个问题。

4.5.7 SPI Flash 操作原型

W25QXX 系列 flash 芯片拥有三个状态寄存器和配置寄存器, 用于控制设备状态等, 如果仅仅对 Flash 芯片进行读取操作, 是不需要修改任何寄存器信息的, 往往只需要发送需要读取的地址, 在发送完毕后, 读取返回的数据即可。

对于写操作, 通常使用指令 02h 即页编程指令, 该操作允许在 1 字节到 256 字节的长度上进行编程。在进行页编程指令之前需要执行写使能指令, 写使能指令是一个非常常用的指令, 通常需要在各种写操作之前执行, 因为这些操作在执行完毕后会自动将 WEL 置为 0, 而执行写使能指令会将 WEL 置为 1。

在 SPI Flash 初始化时, 往往需要进行寄存器的配置, 例如 SEC 等, 每个寄存器中如下位是可写的:

状态寄存器 1 中的 SEC、TB、BP[2:0]位

状态寄存器 2 中的 CMD、LB[3:1]、QE、SRL 位

状态寄存器 3 中的 DRV1, DRV0、WPS 位

其余位全部都是只读的, 并且不会受写寄存器指令影响。对状态寄存器的写访问由非易失性寄存器的 SRL 位的状态控制, 状态寄存器锁定位 SRL 是状态寄存器 (S8) 中的易失

性/非易失性，可读/可写位。当 SRL 置为 0 时，状态寄存器解锁。其他寄存器设置大同小异，不再叙述。

4.5.8 U-Boot 修改

在前面我们说到，因为华邦 Flash 闪存芯片价格过于昂贵，从而使用国产厂商芯天下生产的 XT25F128B 来代替。U-Boot 在尝试从 SPI Flash 启动时会读取 Flash 芯片的 JEDEC ID，并将其与数组 spi_flash_ids 中的 ID 逐一比较，若匹配失败，则不会进行设备驱动注册等操作，打断引导。

如果要通过 SPI Flash 启动，则需要移除 SD/TF 卡，因为 SPI 方式启动拥有最低的优先级，除非在 SD/TF 卡，eMMC 设备中找不到可以引导加载的镜像，才可以从 SPI Flash 上引导加载镜像。在确认板子可以从 SPI Flash 启动之后，进行下一步。设备首次上电，从串口接收到如图 4-19 信息。

```
SF: unrecognized JEDEC id bytes: 0b, 40, 18
Failed to initialize SPI flash at 0:0 (error -2)
No SPI flash selected. Please run `sf probe`
No SPI flash selected. Please run `sf probe`
=> 
```

图 4-19 首次上电 U-Boot 打印的部分信息

Fig.4-19 Charts of External SPI flash module

从 U-Boot 提供的启动信息可以得知，U-Boot 以 SPI 方式启动，卡在 SF: unrecognized JEDEC id bytes: 0b, 40, 18。表示未识别的 JEDEC id，初步分析问题原因是我这块板子上的 SPI Flash 的配置信息并没有添加到 U-Boot 中，导致 U-Boot 无法识别。

使用 grep 工具查找打印出这段信息的函数，输出信息如图 4-20 所示。

grep unrecognized * -nR (4)

```
hz@DESKTOP-UN1BE8G:/mnt/h/u-boot_lichee_nano$ grep unrecognized * -nR
cmd/spl.c:167: /* unrecognized command */
drivers/ddr/fsl/options.c:1181: printf("hwconfig has unrecognized parameter for ctrl_intlv.\n");
drivers/ddr/fsl/options.c:1209: printf("hwconfig has unrecognized parameter for bank_intlv.\n");
drivers/mtd/spi/spi_flash.c:919: printf("SF: unrecognized JEDEC id bytes: %02x, %02x, %02x\n",
drivers/usb/gadget/pxa25x_udc.c:1941: printf(" %s: unrecognized processor: %08x\n",
drivers/usb/gadget/usbstring.c:126: /* unrecognized: stall. */
drivers/usb/musb-new/musb_gadget_ep0.c:199: * USB_REQ_SET_CONFIGURATION, USB_REQ_SET_INTERFACE, unrecognized
drivers/usb/ulpi/ulpi.c:181: printf("ULPI: %s: unrecognized Serial Mode specified: %u\n",
drivers/video/vidconsole-uclass.c:292: debug("unrecognized escape sequence: %*s\n",
fs/ubifs/super.c:1234: ubifs_err(c, "unrecognized mount option \"%s\" or missing value",
include/cli.h:27: * -1 - not executed (unrecognized, bootd recursion or too many args)
include/cli.h:29: * considered unrecognized
tools/buildman/kconfiglib.py:817: _parse_error(line, "unrecognized construct",
tools/buildman/kconfiglib.py:1021: _parse_error(line, "unrecognized option", filename, lineno)
```

图 4-20 grep 查找相关字符串来源

Fig.4-20 Charts of Grep finds the source of related strings

可以看到 drivers/mtd/spi/spi_flash.c 文件中第 919 行打印出了该信息，打开文件查看，发现这三个 id 数值是从数组 id 中获取的，如图 4-21 所示

```

899 static const struct spi_flash_info *spi_flash_read_id(struct spi_flash *flash)
900 {
901     int tmp;
902     u8 id[SPI_FLASH_MAX_ID_LEN];
903     const struct spi_flash_info *info;
904
905     tmp = spi_flash_cmd(flash->spi, CMD_READ_ID, id, SPI_FLASH_MAX_ID_LEN);
906     if (tmp < 0) {
907         printf("SF: error %d reading JEDEC ID\n", tmp);
908         return ERR_PTR(tmp);
909     }
910
911     info = spi_flash_ids;
912     for (; info->name != NULL; info++) {
913         if (info->id_len) {
914             if (!memcmp(info->id, id, info->id_len))
915                 return info;
916         }
917     }
918
919     printf("i[TEST]SF: unrecognized JEDEC id bytes: %02x, %02x, %02x\n",
920           id[0], id[1], id[2]);
921     return ERR_PTR(-ENODEV);
922 }

```

图 4-21 相关字符串具体信息

Fig.4-21 Charts of Related string details

这里将 info 中的 id 注意与通过函数 spi_flash_cmd 获取到的 id 进行比较
本例中结果是比对失败，将 id 中的信息打印，如图 4-22 所示

```

info = spi_flash_ids;
for (; info->name != NULL; info++) {
    if (info->id_len) {
        if (!memcmp(info->id, id, info->id_len))
            return info;
    }
}

```

图 4-22 相关字符串具体信息

Fig.4-22 Charts of Related string details

使用 grep 工具查找 spi_flash_ids 的定义，输出信息如图 4-23 所示

```

doc/device-tree-bindings/mtd/spi/spi-flash.txt:9: drivers/mtd/spi/spi_flash_ids.c for the list of supported chips.
drivers/mtd/spi/Makefile:15:obj-$(CONFIG_SPI_FLASH) += sf_probe.o spi_flash.o spi_flash_ids.o sf.o
drivers/mtd/spi/sandbox.c:171: for (data = spi_flash_ids; data->name; data++) {
drivers/mtd/spi/sf_internal.h:137:extern const struct spi_flash_info spi_flash_ids[];
drivers/mtd/spi/spi_flash.c:911: info = spi_flash_ids;
drivers/mtd/spi/spi_flash_ids.c:46:const struct spi_flash_info spi_flash_ids[] = {

```

图 4-23 grep 查找 spi_flash_ids 定义来源

Fig.4-23 Charts of Grep find SPI_flash_IDS definition source

文件 drivers/mtd/spi/spi_flash_ids.c 中存放 spi flash JEDEC 信息，打开该文件查看，文件内容如图 4-24 所示，我们来到文件底部，仿照上方代码，定义一个 spi flash 信息。

最后修改设备树文件 arch/arm/dts/suniv-flc100s-licheepi-nano.dts 中的 spi0 节点中的 compatible 属性，如图 4-24 所示。


```
&spi0 {
    pinctrl-names = "default";
    pinctrl-0 = <&spi0_pins_a>;
    status = "okay";

    flash@0 {
        #address-cells = <1>;
        #size-cells = <1>;
        compatible = "winbond,xt25f128", "jedec,spi-nor";
        reg = <0>;
        spi-max-frequency = <40000000>;
    };
};
```

图 4-24 设备树节点

Fig.4-24 Charts of device tree node

修改完成后，编译测试，观察到 U-Boot 打印如图 4-25 所示信息，至此 SPI Flash 适配完成。

```
SF: Detected xt25f128 with page size 256 Bytes, erase size 4 KiB, total 16 MiB
*** Warning - bad CRC, using default environment

In:      serial@1c25000
Out:     serial@1c25000
Err:     serial@1c25000
Net:     No ethernet found.
starting USB...
No controllers found
Hit any key to stop autoboot: 0
Card did not respond to voltage select!
mmc_init: -95, time 23
starting USB...
No controllers found
USB is stopped. Please issue 'usb start' first.
starting USB...
No controllers found
No ethernet found.
```

图 4-25 启动成功打印信息

Fig.4-25 Charts of boot info

4.5.9 固件包的烧录

在进行固件包的烧录时，SoC 需要进入 FEL 模式，对于不同的 SoC 分别有不同的做法，我手上这块 F1C100s 则使用最通用的办法，当 SoC 检测不到任何的可加载的镜像时，就会自动进入 FEL 模式。

在本案例中，完全关闭电源，确认 SD 卡拔出或为空时，短接 SPI Flash 的 1、4 引脚，就可以使得 CPU 进入 FEL 模式。原理是这样的，短接 1、4 引脚使得 CS 片选信号被外部拉低，CPU 无法控制片选信号，这将会导致 SPI 接口传输的指令无法正常运行，因为 Flash 芯片在 CS 下降沿接收指令，在上升沿执行接收到的指令。所以 CPU 读取不到任何信息，系统检测不到任何的可加载镜像，所以自动进入 FEL 模式。

编译好的 u-boot 文件、dtb 设备树文件、zImage 内核文件、根文件系统需要打包成一个完整的文件方可进行烧录，在 linux 下，最常用的工具莫过于 dd 了。

使用如下指令创建烧录镜像文件：

```
dd if=/dev/zero of=flash.img bs=1M count=16
```

按照配置文件中的偏移地址信息将文件写入进 flash.img 中

```
dd if=u-boot.bin of=flash.img seek=0
```

```
dd if=sunxi.dtb of=flash.img bs=$((0x100000)) seek=1
```

```
dd if=zImage of=flash.img bs=$((0x110000)) seek=1
```

```
dd if=jffs2.img of=flash.img bs=$((0x610000)) seek=1
```

4.5.10 系统的启动流程

CPU 上电后会执行 BROM 中的程序，这也就是说 BROM 中的程序决定了整个系统的初步走向，CPU 根据 BROM 中代码，对各种存储设备中的内容进行尝试启动，优先级都是固定的，例如 Flash 为最低优先级，初步推测 BROM 这部分代码是固化在其中的，当然，SUNXI 官方并没有提供能修改这部分代码的工具，所以对于 BROM 我们不能做任何修改。

假设 CPU 成功在 SPI Flash 中找到可执行的镜像文件，执行引导加载程序，在本案例中为 U-Boot，推测 SPL 程序位于整个 u-boot 镜像的头部，它会将真正的 u-boot 拷贝到内存中继续执行。U-Boot 执行过程中对外设进行初始化，设置环境变量，通过 bootz 加载 zImage 内核镜像，内核读取环境变量参数，进行初始化操作，然后挂载根文件系统，系统启动完成。

4.6 显示模块 Display

该模块中包含 LCD 背光驱动电路, TFT RGB 40Pin 接口, NS2009 触摸驱动芯片

4.6.1 模块原理图

如图 4-26 所示

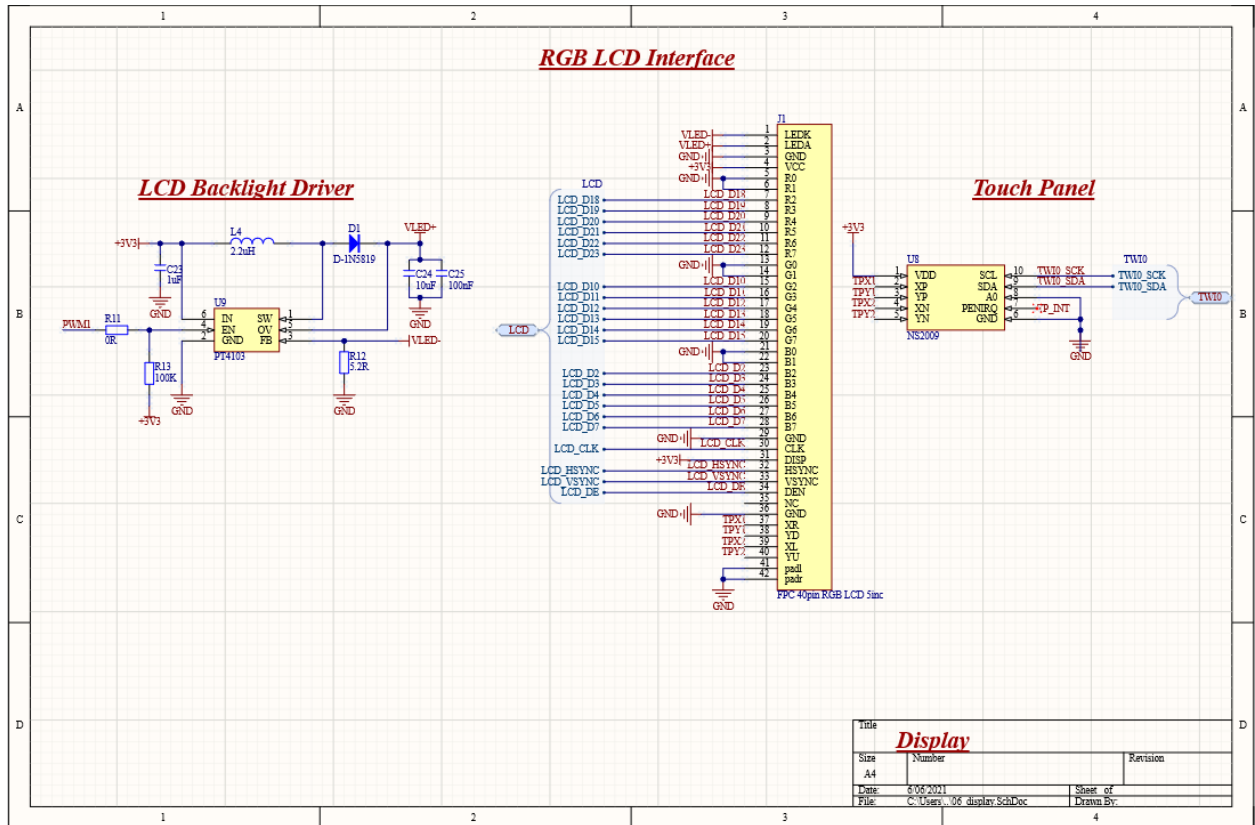


图 4-26 显示模块原理图

Fig.4-26 Charts of Schematic diagram of display module

4.6.2 LCD 背光驱动电路

PT4103 是一款升压 DC/DC 转换器, 被设计用单个恒流锂电池来驱动最多高达 8 个串联 LED, 在驱动 8 颗 LED 时要求输入电压为 3.6V, 串接 8 个 led 意味着输出电压要大于 8 个 LED 的压降才能工作, 典型的 LED 压降大约在 1.8V 左右, 甚至更高, 若为 1.8V, 则串联 8 个 LED 压降达到了 14.4V。在手册顶部概述部分说明: PT4103 使用单个锂电池驱动 8 个 LED 的原因是它直接调节输出电流, PT4103 是理想的 LED 驱动, LED 的发光强度与通过它们的电流大小成正比, 而不是端子之间的电压。

按照手册中推荐工作范围图表可以得知, V_{in} 输入电压范围要在控制在 2.5V~6V, V_{sw} 输出电压范围在 V_{in} ~28V 之间, 所以这块芯片理论上能最大驱动 20 多颗串联 LED。

在本案例中,我使用了两种型号的 LCD 显示器件,其中一种是尺寸大小为 5 寸的 TFT LCD,其分辨率大小为 800*480,背光类型为 12 颗并联 LED,背光驱动电压为 18V。另外一种尺寸大小为 4.3 寸的 TFT LCD,其分辨率大小为 480*272,背光类型为 7 颗并联 LED,背光驱动电压为 21V。两者都可搭配电阻式触摸屏,接口类型都为 40PIN 24 位 RGB 接口。

连接到芯片 FB 引脚的电阻用于控制 LED 电流大小,其计算公式为:

$$R_{FB} = \frac{104mV}{I_{LED}} \quad (4-5)$$

LED 的工作电流一般在 5mA 到 20mA 之间,在这里为了保险起见使用 20mA 电流供电,导入公式计算得 R_{FB} 大小为 5.2Ω 。

PT4103 芯片手册中,官方给出了三种调光电路,本案例使用的是如图 4-27 所示的电路,通过调整 PWM 波占空比,根据频闪间隔调整亮度。

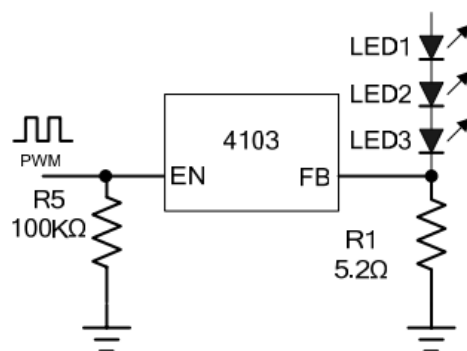


图 4-27 使用 PWM 信号控制 EN 使能端口

Fig.4-27 Charts of Using PWM signal to control en enable port

4.6.3 40PIN TFT LCD 24 位 RGB 接口

在上一小节中,我们说到在这个项目中使用了两种型号的 LCD 显示屏,分别为 4.3 寸和 5 寸,其两者都是 40Pin 接口,RGB 三组信号分别都有 8 根线组成,但是从原理图上看的话每组只用了 6 根线,原因是 F1C100s 的 LCD 数据线只有 18 根,所以每组信号线都裁减了两根,然后就是 CLK 线用来控制像素点刷新,HSYNC 线用来控制换行,VSYSNC 线用来归零位,DE 用来使能数据。

XR、YD、XL、YU 四根线用来连接电阻式触摸屏,其通过排线触点跟 LCD 排线连接在一起,触摸屏框体则粘贴在 LCD 屏幕区域正上方。

4.6.4 NS2009 电阻式触摸屏驱动芯片

NS2009 是使用 I^2C 接口通信的四线式电阻触摸屏驱动芯片,其内置一个 12 位分辨率的

AD 转换器，它通过 XP、YP、XN、YN 四个端口连接电阻式触摸屏，在本项目中占用了 CPU 的 TWIO 接口，A0 端口用来表示 $1I^2C$ 设备地址的最后一位，在这里连接到地，根据芯片手册中的说明，设备地址为 $1001000+R/W\#$ ，写指令时设备地址为 0x90，读数据时设备地址为 0x91。PENIRQ 即笔中断端口用来向 CPU 发出中断信号。官方建议，当处理器向 NS2009 发送命令时，屏蔽 PENIRQ 中断。

4.6.5 LVGL 的移植

LVGL (Light and Versatile Graphics Library) 是一个用于显示和触摸屏的轻量级嵌入式库，提供构建功能齐全的嵌入式 GUI 所需的一切。它拥有如图 4-28 所示特点：

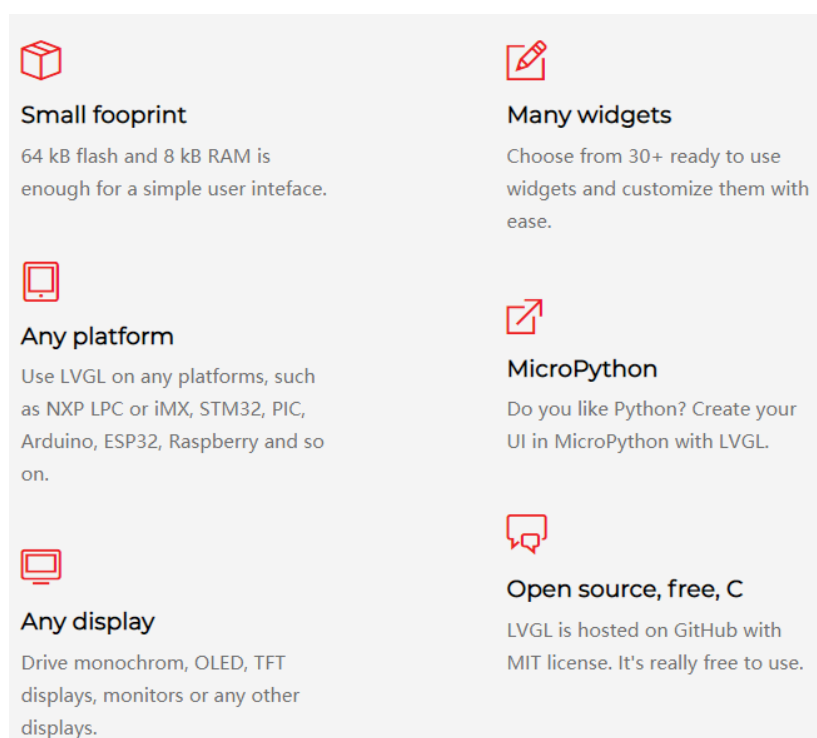


图 4-28 LVGL 特点

Fig.4-28 Charts of Lvgl features

我在开始这个项目的时候使用的是 lvgl V7 版本，在我写这篇论文的时候，lvgl 最新版已经是 V8 了，V8 版本优化了很多问题，提供了一些新的特性，我正积极的将应用迁移到 V8 版本中。下面的移植操作仅适用于 V7 版本，进行修改后可以适用于 V8 版本。

在本案例中，我将 LVGL 移植到 Linux framebuffer，经过初步分析，需要做如图 4-29 工作：



图 4-29 移植准备工作

Fig.4-29 Charts of Preparation for transplantation

Linux 内核提供了一个名为 fb 的设备节点，其由 framebuffer 帧缓冲驱动生成，用户可以通过打开该设备节点，通过一系列操作，可以使自定义内容显示到输出设备上，一般是一块 LCD 屏幕，利用这个特性我们可以完成 LVGL 的显示部分。

我实现了一个函数 `void my_fb_init(void)`

这个函数做了些什么？

- 1) 打开设备节点 `/dev/fb0`
- 2) 通过 `ioctl` 获取 `fb_var_screeninfo`
- 3) 计算行宽、单像素宽度、屏幕像素数量
- 4) 映射 framebuffer 到内存中
- 5) 清除整个屏幕

做完这些操作之后你就可以在 `my_disp_flush` 函数中使用 `memcpy` 函数操作 framebuffer 了。实现代码见附录。

输入部分我暂时只实现了单点触摸，日后可能会实现多点触摸，使用单点触摸会限制一些交互功能，比如双指点击事件，双指放大缩小等等。

我选用的平台有一块大小为 5 寸的电阻式触摸屏，分辨率为 800*480。Linux 操作系统中的设备驱动提供了一系列的输入事件(有关这部分，限于篇幅，不展开讨论)，而我的这块电阻式触摸屏大概有以下这么几种事件：

- 1) 同步事件 `EV_SYN`，用来间隔事件
- 2) 按键事件 `EV_KEY`
- 3) 压力值 `BTN_TOUCH`
- 4) 绝对位移事件 `EV_ABS`
- 5) 触点 ID `ABS_MT_TRACKING_ID`
- 6) X 坐标 `ABS_MT_POSITION_X` `ABS_X`
- Y 坐标 `ABS_MT_POSITION_Y` `ABS_Y`

使用异步通知方式读取输入事件时，需要提供一个信号处理函数，在本项目中名为 my_touchpad_sig_handler。在 main 函数中也不许要创建单独线程来读取输入事件，一切操作都由信号处理函数完成。首先在输入设备初始化函数中进行如图 4-30 中的设置

```
1  signal(SIGIO, my_touchpad_sig_handler);
2  fcntl(indev_info.tp_fd, F_SETOWN, getpid());
3  flags = fcntl(indev_info.tp_fd, F_GETFL);
4  fcntl(indev_info.tp_fd, F_SETFL, flags | FASYNC | O_NONBLOCK);
5  printf("Successfully run in async mode.\n");
```

图 4-30 开启驱动异步通知方式

Fig.4-30 Charts of Turn on driver asynchronous notification mode

我使用的信号处理函数，将事件筛选功能抽离出成为一个函数 my_touchpad_probe_event，相关代码见附录，信号处理函数如图 4-31 所示

```
void my_touchpad_sig_handler(int signal)
{
    while(read(indev_info.tp_fd, &indev_info.indev_event,
        sizeof(struct input_event)) > 0)
        my_touchpad_probe_event();
}
```

图 4-31 信号处理函数

Fig.4-31 Charts of Signal processing function

4.6.7 基于 LVGL 的应用开发

根据我使用的软件设计模型，一开始是在使用 Windows API 实现的 LVGL 工程上进行模拟开发的，然后交叉编译到现有平台上运行测试，该平台为百问网售卖的基于 NXP I.MX6ULL 的 100ask_imx6ull_pro 开发板，对于 LVGL 应用开发来说，它的一个亮点就是拥有一块大小为 7 寸，分辨率为 1024*600 的电容式触摸屏，并且支持多点触摸，尽管我在应用开发中还没有使用到多点触摸，毫无疑问，这块屏幕在各方面都是非常不错的。

无论是哪种 GUI 框架，编写 GUI 部分都是一个非常耗时的任务。LVGL 提供了一些常用的基本的控件和交互事件。控件例如按钮，输入框、列表、通知气泡等，事件有点击、长按、短按等，但是没有类似于全面屏手势、状态栏等这种高级 GUI 功能，官方的意思是让开发者自己编写这部分内容，这无疑大大增加了开发难度，任何一个毫不起眼的细节，都有可能需要几十行代码来实现。

尽管我在项目初期 4 月份的时候就开始练习编写 LVGL 应用了，但最后开发出原型的应用也仅仅只有 2 个，它们分别是自动售货机应用和应用桌面。

自动售货机应用

应用结构如下图 4-32 所示



图 4-31 自动售货机应用结构

Fig.4-31 Charts of Application structure of vending machine

在 Windows 上运行效果如图 4-32 所示。



图 4-32 自动售货机应用示例图

Fig.4-32 Charts of Application example of vending machine

实机运行效果如图 4-33 所示。

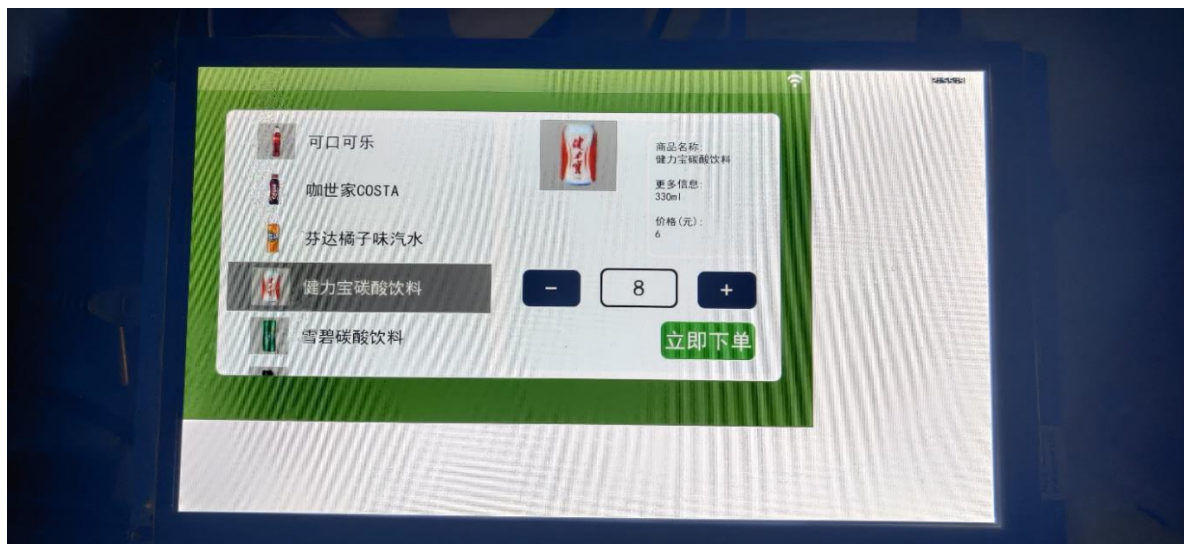


图 4-33 自动售货机应用实机运行图

Fig.4-32 Charts of Real machine operation diagram of vending machine application

应用桌面

应用桌面是本项目的核心软件，这里几乎涵盖了所有应用的入口，笔者在编写这个应用时在寻找一种可行的解决方案，即动态安装卸载软件，已安装的软件会在主界面中显示图标，比较可行的一种解决方案就是定义一种软件包，规定软件包中应该存放哪些文件，例如图片、音效、配置信息和可执行文件，目前正在实现这方面的功能。

应用结构如下图 4-34 所示。



图 4-34 应用桌面结构

Fig.4-32 Charts of Application desktop structure

在 Windows 上运行效果如下图 4-35 所示。

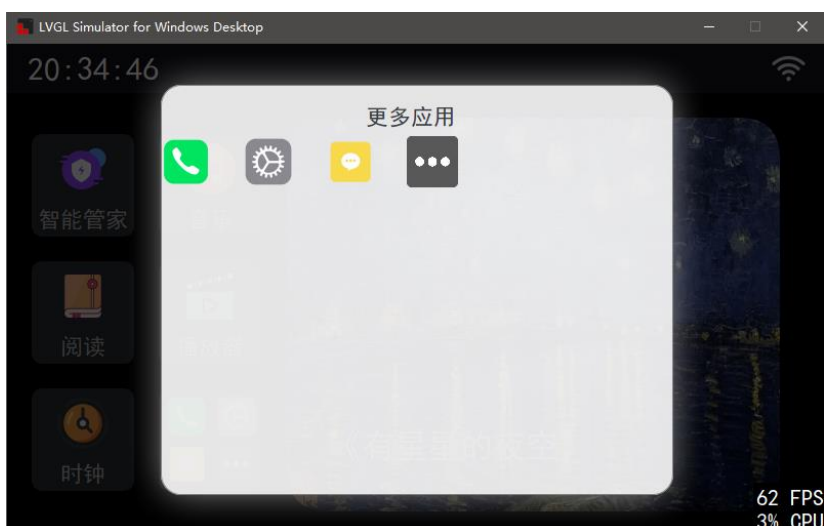


图 4-35 应用桌面运行示例

Fig.4-35 Charts of Application desktop running example

在实机上运行效果如下图 4-36 所示。

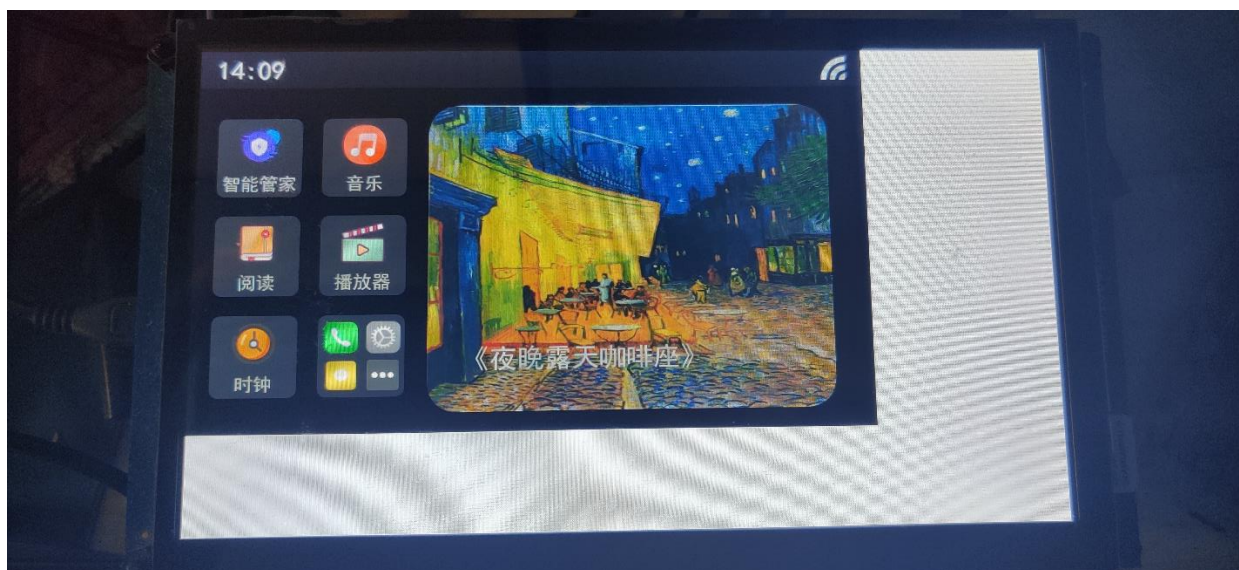


图 4-36 应用桌面实机运行效果

Fig.4-36 Charts of Application desktop real machine running effect

4.6.8 使用 CMake 进行 LVGL 的 Makefile 的构建

随着工程结构变得愈来愈复杂，手动编写 Makefile 来对工程进行编译变得非常困难，使用 CMake 工具来自动生成 Makefile 文件让这一过程变得无比轻松。使用 CMake 构建工程时，需要手动编写 CMakeLists.txt 文件，其中包含很多设置信息以及变量等，在本案例中，编写的 CMakeLists.txt 内容如下

```
cmake_minimum_required(VERSION 3.10)
project(lvgl)
set(CMAKE_C_STANDARD 11)#C11
set(CMAKE_CXX_STANDARD 17)#C17
set(CMAKE_CXX_STANDARD_REQUIRED ON)
INCLUDE_DIRECTORIES(${PROJECT_SOURCE_DIR})
include_directories(.)
file(GLOB_RECURSE INCLUDES "lv_drivers/*.h" "lv_examples/*.h" "lvgl/*.h" "/*.h" "FLT/*.h"
"my_apps/*.h")
file(GLOB_RECURSE SOURCES "lv_drivers/*.c" "lv_examples/*.c" "lvgl/*.c" "FLT/*.c"
"my_apps/*.c")
set(CMAKE_C_COMPILER
"/home/book/100ask_imx6ull-sdk/ToolChain/gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabi/bin/arm-li
nux-gnueabi-gcc")
set(CMAKE_BUILD_TYPE "Release")
set(WARINGS "-Wall -Wextra \
-Wshadow -Wundef -Wmaybe-uninitialized -Wmissing-prototypes
-Wno-discarded-qualifiers \
```

```

        -Wno-unused-function      -Wno-error=strict-prototypes      -Wpointer-arith
-fno-strict-aliasing -Wno-error=cpp -Wuninitialized \
        -Wno-unused-parameter -Wno-missing-field-initializers -Wno-format-nonliteral
-Wno-cast-qual -Wunreachable-code -Wno-switch-default \
        -Wreturn-type      -Wmultichar      -Wformat-security      -Wno-ignored-qualifiers
-Wno-error=pedantic -Wno-sign-compare -Wno-error=missing-prototypes -Wdouble-promotion -Wclobbered
-Wdeprecated \
        -Wempty-body -Wshift-negative-value -Wstack-usage=2048 \
        -Wtype-limits -Wsizeof-pointer-memaccess -Wpointer-arith")
SET(CMAKE_C_FLAGS_RELEASE "-O3 ${WARNINGS}")
SET(EXECUTABLE_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/bin)
add_executable(demo main.c ${SOURCES} ${INCLUDES})

```

编写完成 CMakeLists.txt 文件之后，在其所在目录执行 **cmake** . 后，cmake 将开始对工程进行 Makefile 的构建，命令执行成功，则 Makefile 将会生成在当前目录下，然后就可以执行 **make -jn** 对工程进行编译。另外，cmake 还有一些高级用法，例如传递头文件参数，不在本文讨论范围之内。

4.7 SD_eMMC

模块原理图如下图 4-37 所示

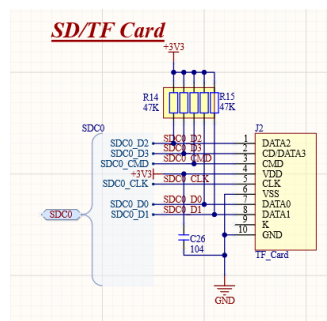


图 4-37 SD 卡模块原理图

Fig.4-37 Charts of Schematic diagram of SD card module

这里使用回弹式的标准 TF 卡座，使用了处理器的 SDC0 接口。有关 SDIO 协议的具体内容不在本文讨论范围之内。参考网上资料得知，当接口工作于 SD 模式时，为了避免引脚浮空，除 CLK 外所有引脚上拉到 3.3V，CLK，CMD 和 DAT 在插卡的状态都可能高阻。不能下拉的原因是 CMD，DAT 数据的起始标志是 0。

如果走线大于 $\frac{\text{波长}}{10}$ 就需要考虑阻抗匹配，在信号源端串接 33R 电阻，而 SD 卡信号速率最大 208M，所以走线不超过 15cm 就不需要串接匹配电阻。在本案例中，布线短于 15cm，使用 0R 电阻上拉至 3.3V 即可。

对于 eMMC，在后续的版本中为了系统稳定性需要使用 eMMC 替换 SD 卡，但是在目前版本内还没有添加 eMMC 芯片，不过按照接口 eMMC 接口 4.41 已经找到了比较合适的 eMMC 芯片，例如 SDIN5G2-4G，并且已经规划了设计方案。

4.8 串口 UART

模块原理图如下图 4-38 所示

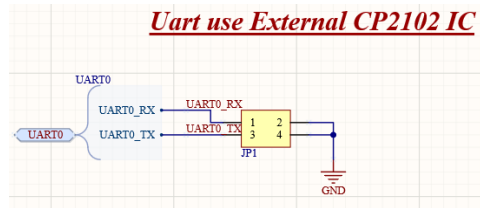


图 4-38 串口通信模块原理图

Fig.4-38 Charts of Schematic diagram of serial communication module

为了减少 BOM 成本，在本项目内并没有添加板载的 USB 转串口芯片，而是使用外部的 USB 转串口芯片 CP2102 来完成与处理器之间的通讯，所以用了一个 2x2 排针引出相关接口来使用。

4.9 音频接口 Audio

模块原理图如下图 4-39 所示

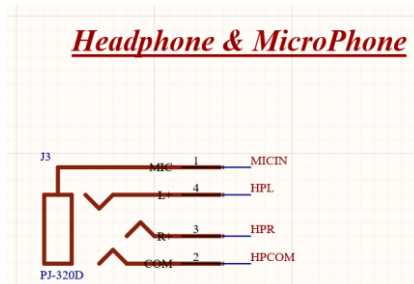


图 4-39 音频模块原理图

Fig.4-39 Charts of Schematic diagram of audio module

我使用的音频接口是 PJ-320D，需要配合四段式音频插头使用，其相比三段式插头额外增加了麦克风输入。MICIN 端口即麦克风信号输入，HPL 为音频左声道，HPR 为音频右声道，HPCOM 为公共地。

该模块需要相关驱动支持，我暂时还不确定官方 bsp 包中包含音频驱动，但是音频功能是本项目必不可少的一部分，如果没有提供音频驱动，则实现音频驱动是一个非常困难且棘手的问题。

5 系统测试

在最初版本的 PCB 中，因为设计方面错误的铺铜问题导致板上 IC 异常难手工焊接，我错误的将芯片引脚附近铺上了铜以及相同网络的 IC 引脚用铺铜连接，这是一个非常错误的做法，正确的做法应该是去除 IC 引脚附近的铜，原因则是其引脚附近铺铜会导致焊接时热量被快速散发向周围，导致融化的锡过快凝固，增加焊接难度，同时对相同网络引脚进行铺铜会导致连锡，与前者的问题相似，这样会导致该铜皮与其他铜皮割裂开，从而导致热量只集中在本区域而造成连锡，使得芯片引脚频繁连锡，电源电压无法正常输出，系统无法正常工作。

在最新的修订版本 2 中这些问题都得到了解决，对整个 PCB 进行了重新布线铺铜，并且做了一定优化，考虑到主、从机问题，在修订版本 2.1 中给 USB Type-C 接口添加了下拉电阻以让其被识别为从机。

对于软件工具库，一开始做界面设计时，不需要涉及系统库函数的调用，类似本地时间的获取等，所以在 Windows 上做的非常顺利，但后来设计这部分操作时找到了如下的解决方案，使用宏定义区分软件运行平台，包含不同的库文件，数据使用共性结构体保存，函数封装根据不同平台使用类似 WIN32 类似宏分别定义。测试运行效果良好，软件可以跨平台运行。

6 结论

本案例实现了作为嵌入式 Linux 设备应该拥有的基本功能，例如 LCD、接口、触摸屏、音频输入输出、存储设备等。

本项目优点是结构紧凑，可以通过螺柱将多块板子组合成集群，还有就是方便二次开发，工程可拓展性高。缺点就是目前版本无法进行网络通信，虽然 Sipeed 公司有推出的使用 SD 卡接口的 WIFI 模块，但我想把网络功能固定在板子上，还有就是还没有实现真正的控制设备功能。再就是应用开发还未完成，很多 APP 还处于开发阶段，计划使用 GTK 以及 FLTK 这种高级 GUI 库来编写更加高级的应用软件，未来的版本中可能还会加入娱乐游戏等等。

所以在未来版本中，将逐步完善这些功能，有网络、NFC、设备控制等等一些特别酷的功能，电子项目的成长就像是植物的生长过程，这是一个非常微妙的过程，硬件像是植物的根茎，随着一次次修订让其越来越稳定，而软件则像植物生长出的果实供人们食用，由此看来，本项目还有相当长的一段路要走。

参考文献

- [1] 韦东山. 嵌入式 Linux 应用开发完全手册[M]. 人民邮电出版社, 2008
- [2] JONATHAN CORBET, ALESSANDRO RUBINI & GREG KROAH-HARTMAN, Linux 设备驱动程序[M] 中国电力出版社 2006
- [3] Michael Kerrisk. Linux/UNIX 系统编程手册[M] (上/下册). 人民邮电出版社, 2014
- [4] Robert Love. Linux 内核设计与实现[M]. 人民邮电出版社 2011.6
- [5] Allwinner Technology, "F1C100s Datasheet," F1C100s datasheet[Z], Nov. 2015[Revised Nov.10 2015]
- [6] Allwinner Technology, "F1C600 User Manual," F1C600 datasheet[Z], Nov. 2015[Revised Nov.10 2015]
- [7] SanDisk Corporation, "iNAND Standard and Ultra e.MMC 4.41 I/F," SDIN5D2-4G-LT datasheet[Z], Mar. 2011
- [8] TORIEX, "Low ESR Cap. Compatible Positive Voltage Regulators", XC6206 Series datasheet[Z], 2013
- [9] PowTech, "White LED Step-Up Converter", PT4103 datasheet[Z], 2017
- [10] EVER ANALOG, "3CH Power Management IC", EA3036 datasheet[Z]
- [11] ShenZhen Nsiway Technology Co, "4-Wire Touch Screen Controller with I2C Interface", NS2009 datasheet[Z], 2011
- [12] Winbond, "SPI Flash 3V 128M-BIT SERIAL FLASH MEMORY WITH DUAL/QUAD SPI", W25Q128JV datasheet[Z], [Revised 2016]
- [13] NXP, "NTAG I2C - Energy harvesting NFC forum Type 2 Tag with field detection pin and I2C interface", NT3H1201 datasheet[Z]
- [14] 乐鑫科技, "高性能无线 SoC", ESP8089 datasheet[Z]
- [15] JoulWatt, "1.2A, 6V, 1.25MHz, 40uA IQ Synchronous Step-Down Converter", JW5211 datasheet[Z].
- [16] Allwinner Technology, "F1C200s Datasheet," F1C200s datasheet[Z], Nov 5 2019
- [17] 芯天下 XTX, "Quad IO Serial NOR Flash Datasheet", XT25F128B datasheet[Z], Apr 1 2021
- [18] Dongguan Hanbo Electronic Technology Co, "SD 卡座", SD-111 datasheet[Z].2019
- [19] devicetree.org, "Devicetree Specification Release v0.3" Devicetree Specification[Z], Feb 13 2020.
- [20] 乐鑫科技, "搭载 RISC-V 32 位单核处理器的极低功耗 SoC E" SP32-C3 datasheet[Z], 2021.
- [21] 安信可, ESP32-C3F 规格书[Z], 2021

附录

Alitium Designer 生成的预览 PCB 模型，如图 8-1 所示

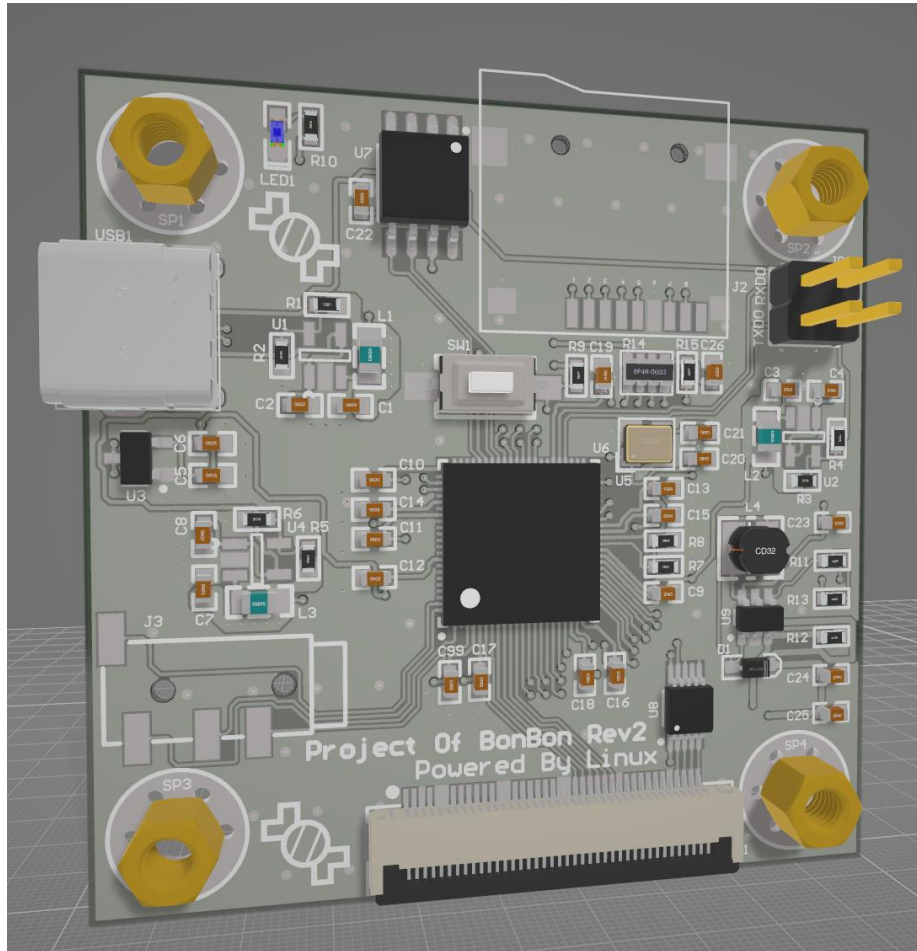


图 8-1 PCB 模型

Fig.8-1 Charts of PCB model

mkimage 工具指令格式

```
mkimage -C none -A arm -T script -d boot.cmd boot.scr
```

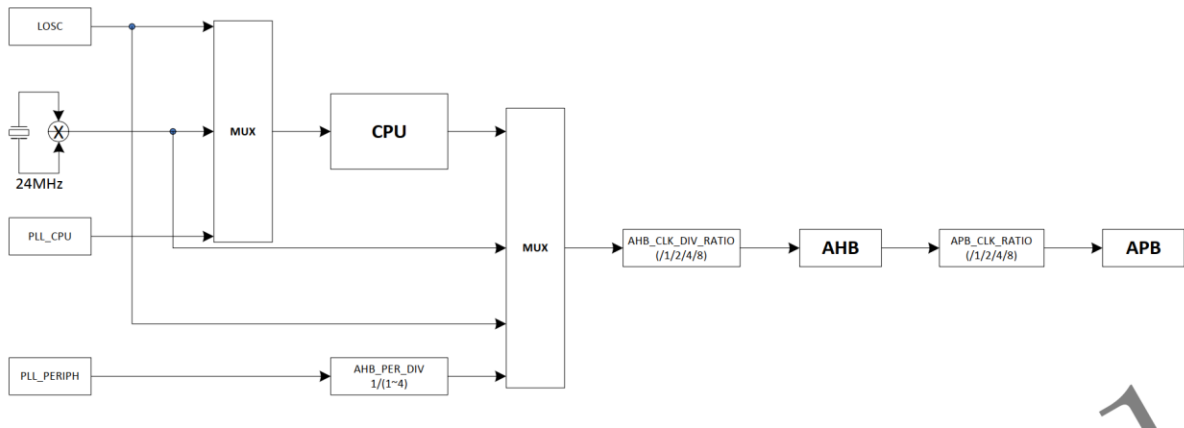
Boot.cmd 常见格式

```
setenv bootargs console=ttyS0,115200 root=/dev/mmcblk0p2 rootwait panic=10
load mmc 0:1 0x43000000 ${fdtfile} || load mmc 0:1 0x43000000 boot/${fdtfile}
load mmc 0:1 0x42000000 uImage || load mmc 0:1 0x42000000 boot/uImage
```

```
bootm 0x42000000 - 0x43000000
```

时钟树结构

3.2.3.2. Bus clock tree



函数 my_fb_init 源代码

```
void my_fb_init(void)
{
    fbdev_info.fd_fb = open(DEFAULT_LINUX_FB_PATH, O_RDWR);
    if(fbdev_info.fd_fb < 0)
    {
        handle_error("can not open framebuffer");
    }
    printf("Successfully opened framebuffer.\n");
    /* already get fd_fb */
    if(ioctl(fbdev_info.fd_fb, FBIOGET_VSCREENINFO, &fbdev_info.fb_var) < 0)
    {
        handle_error("can not ioctl");
    }
    if (ioctl(fbdev_info.fd_fb, FBIOGET_FSCREENINFO, &finfo))
    {
        handle_error("Error reading fixed information/n");
    }
    /* already get the var screen info */
    screen_info.width = fbdev_info.fb_var.xres;
    screen_info.height = fbdev_info.fb_var.yres;
    screen_info.bpp = fbdev_info.fb_var.bits_per_pixel;
    screen_info.line_width = fbdev_info.fb_var.xres * fbdev_info.fb_var.bits_per_pixel / 8;
```

```

screen_info.pixel_width = fbdev_info.fb_var.bits_per_pixel / 8;
screen_info.screen_size      =      fbdev_info.fb_var.xres      *      fbdev_info.fb_var.yres      *
fbdev_info.fb_var.bits_per_pixel / 8;

printf("screen info:\n Resolution:\t%d\t%d\n Bits per pixel:\t%d\n",
       screen_info.width,screen_info.height,screen_info.bpp);
printf("frame buffer size:%d\n", finfo.smem_len);

/* mmap the fb_base */
fbdev_info.fb_base = (unsigned char *)mmap(NULL, screen_info.screen_size, PROT_READ |
PROT_WRITE, MAP_SHARED, fbdev_info.fd_fb, 0);
if(fbdev_info.fb_base == (unsigned char *) -1)
{
    handle_error("can not mmap frame buffer");
}
/* already get the start addr of framebuffer */
printf("Successfully get the start address of framebuffer.\n");
memset(fbdev_info.fb_base, 0xff, screen_info.screen_size); /* clear the screen */
printf("Successfully clear the screen.\n");
}

```

my_disp_flush 函数源代码

```

void my_disp_flush(lv_disp_drv_t *disp, const lv_area_t *area, lv_color_t *color_p)
{
    int32_t x, y;
    for(y = area->y1; y <= area->y2; y++)
    {
        for(x = area->x1; x <= area->x2; x++)
        {
            memcpy(fbdev_info.fb_base + x * screen_info.pixel_width + y * screen_info.line_width,
                   &color_p->full, sizeof(lv_color_t));
            color_p++;
        }
    }

    if(ioctl(fbdev_info.fd_fb, FBIOPAN_DISPLAY, &fbdev_info.fb_var) < 0) {
        fprintf(stderr, "active fb swap failed\n");
    }
}

```

```

    }
    lv_disp_flush_ready(disg);
}

```

函数 my_touchpad_probe_event 源代码

```

void my_touchpad_probe_event(void)
{
    switch(indev_info.indev_event.type)
    {
        case EV_KEY:    /* Key event. Provide the pressure data of touchscreen*/
            if(indev_info.indev_event.code == BTN_TOUCH)        /* Screen touch event */
            {
                if(1 == indev_info.indev_event.value)            /* Touch down */
                {
                    indev_info.touchdown = true;
                }
                else if(0 == indev_info.indev_event.value)        /* Touch up */
                {
                    indev_info.touchdown = false;
                }
                else                                                /* Unexcepted data */
                {
                    goto touchdown_err;
                }
            }
            break;
        case EV_ABS:    /* Abs event. Provide the position data of touchscreen*/
            if(indev_info.indev_event.code == ABS_MT_POSITION_X)
            {
                indev_info.last_x = indev_info.indev_event.value;
            }
            if(indev_info.indev_event.code == ABS_MT_POSITION_Y)
            {
                indev_info.last_y = indev_info.indev_event.value;
            }
            break;
        default:
            break;
    }
    touchdown_err:    /* Do nothing. Just return and ready for next event come. */
    return;
}

```

函数 my_touchpad_thread 源代码

函数名已经更名为 my_touchpad_poll_handler

void my_touchpad_poll_handler(lv_task_t *task)

```
{
    (void)task;
    int len;
    len = poll(indev_info.mpollfd, indev_info.nfds, INPUT_SAMEPLING_TIME);
    if(len > 0)          /* There is data to read */
    {
        if(read(indev_info.tp_fd, &indev_info.indev_event,
                sizeof(indev_info.indev_event)) > 0)
        {
            my_touchpad_probe_event();
        }
    }
    else if(len == 0)     /* Time out */
    {
        /* Do nothing */
    }
    else                 /* Error */
    {
        handle_error("poll error!");
    }
touchdown_err:          /* Do nothing. Just return and ready for next event come. */
    return;
}
```

致谢

首先对百问网韦东山嵌入式 Linux 团队全体表示由衷的感谢，如果没有韦东山老师团队如此优秀的教程体系，我不会在嵌入式道路上走下去。我能看到我与这个项目度过的日日夜夜，与这个项目一起逐渐成长。

感谢我的亲人及朋友们的鼓励与支持。

最后感谢青岛科技大学高密校区提供的优秀的学习环境，以及大量的学习资源，感谢各位老师的细心指导。

华政
2021 年 6 月于青岛

青岛科技大学专科毕业设计（论文）综合评定意见表

毕业设计(论文)题目		嵌入式智能家居控制终端的设计	
学院(校区)	信息科学技术学院	综合成绩(Y)	
专 业	计算机应用技术	班 级	计专 1866
学生学号	1808610605	学生姓名	华政
指导教师评语			
	(满分 40 分)成绩(X ₁):	指导教师签名:	年 月 日
答辩组评语			
	(满分 60 分)成绩(X ₂):	答辩组组长签名:	年 月 日