**青 岛 科 技 大 学**

**专 科 毕 业 设计（论文）**

嵌入式智能家居控制终端的设计

**题 目 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Design of embedded smart home control terminal

范玮

**指导教师\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

范玮

**辅导教师\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

华政

**学生姓名\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**学生学号\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1808610605

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_学院 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_专业\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_班**

1866

信息科学技术学院

计算机应用技术

20

5

2021

**\_\_\_\_\_\_年 \_\_\_月 \_\_\_日**

嵌入式智能家居控制终端的设计(三号，黑体)

摘 要

(三号，黑体，“摘要”两字中间插入两个半角空格)

智能家居是在互联网影响之下物联化的体现，互联网以及通信技术的高速发展则不断催化着这一过程。

本文利用一种较为科学的设计方法，设计出一款嵌入式智能家居平台，其拥有便于维护拓展，快速应用的特点。其目的为了加速智能家居行业发展，让每一位感兴趣的开发者都能快速融入智能家居行业。（小四，宋体）

关键词：（四号，黑体）智能家居；嵌入式系统设计；操作系统；图形用户界面；应用开发；驱动开发；嵌入式主板设计；电子产品项目；终端机（小四，宋体）

**Design of embedded smart home control terminal**

**ABSTRACT**

Smart home is the embodiment of IOT under the influence of Internet, and the rapid development of Internet and communication technology constantly catalyzes this process.

In this paper, a more scientific design method is used to design an embedded smart home platform, which has the characteristics of easy maintenance and rapid application. Its purpose is to accelerate the development of smart home industry, so that every interested developer can quickly integrate into the smart home industry.

**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 1](#_Toc511741486)

[1相关知识介绍 2](#_Toc511741487)

1.1 软件方面相关知识...........................................................................................................2

1.2 硬件方面相关知识.............................................................................................................2

[2需求分析 4](#_Toc511741492)

[3系统概要设计 4](#_Toc511741492)

[4系统详细设计 4](#_Toc511741492)

[5系统测试 4](#_Toc511741492)

[6结论 4](#_Toc511741492)

[参考文献 6](#_Toc511741496)

[附录 7](#_Toc511741497)

[致谢 8](#_Toc511741498)

# 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。

# 1相关知识介绍

## 1.1软件方面相关知识

对于软件方面，从底层往上层介绍：

①移植U-Boot

②移植Linux内核以及设备树文件的修改

在两本部分皆在全志科技提供的Bsp文件基础上进行二次开发。

③使用buildroot构建jffs2最小根文件系统

④编写Linux触摸屏驱动

基于ns2009触摸芯片以及4.3寸或5寸四线电阻式触摸屏。

基于esp8089 spi模式的无线网卡驱动。

⑤移植LVGL

嵌入式跨平台的轻量级开源图形库。

⑥基于Linux及LVGL的应用开发

涉及GUI设计，网络通信，多线程，进程通信等linux应用开发常用技术。

⑦MQTT通信协议的应用

⑧华为云物联网平台设备接入（IoTDA）

## 1.2 硬件方面相关知识

对于硬件方面：

①原理图的绘制采用树状结构——页面符来绘制，结构清晰分明，方便管理拓展。

②PCB绘制即一些常用技术，设计完成后对pcb进行2x2拼板。

原理图库及PCB库基于器件数据手册绘制。

# 2需求分析

## 2.1 项目需求分析

现阶段项目一般有这几个方面的需求：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：功能、性能、用户界面、以及可靠性。

①功能需求：可视化智能家居控制、多媒体功能、娱乐游戏、语音助手、自由软件市场、手机配套App、Web端管理页面、以及可OTA升级固件。

②性能需求：要求界面刷新流畅，网络响应快速，用户交互反馈及时。

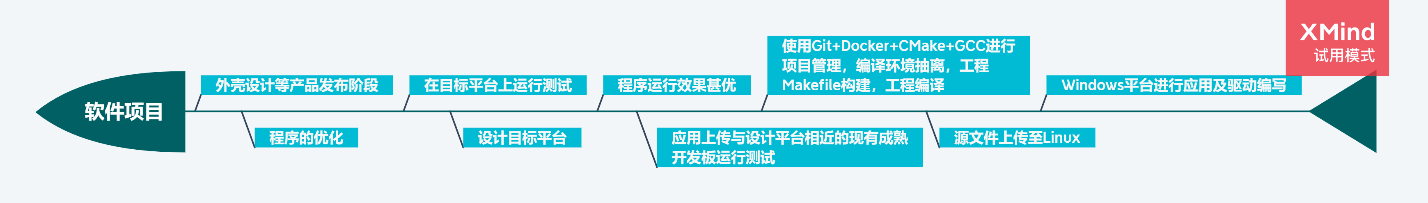
③用户界面需求：界面设计符合现代风格

④可靠性：应用要具有健壮性的特点，应用可自启动等，代码易维护，且容易拓展。

**BootLoader**使用U-Boot，**固件烧录**使用sunxi-tools[[1]](#footnote-1)，**操作系统**基于Linux，而**图形用户界面**使用LVGL配合Linux fbdev完成。**Linux驱动**涉及ESP8089-SPI网卡驱动，NS2009四线电阻式触摸驱动，可能会涉及帧缓冲以及音频驱动[[2]](#footnote-2)。

## 2.2 项目开发模型

整个软件项目采用如下模型进行开发：



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

## 3.2 系统概要-软件方面

软件方面，整个项目软件框图如下:



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，方便管理与配置。

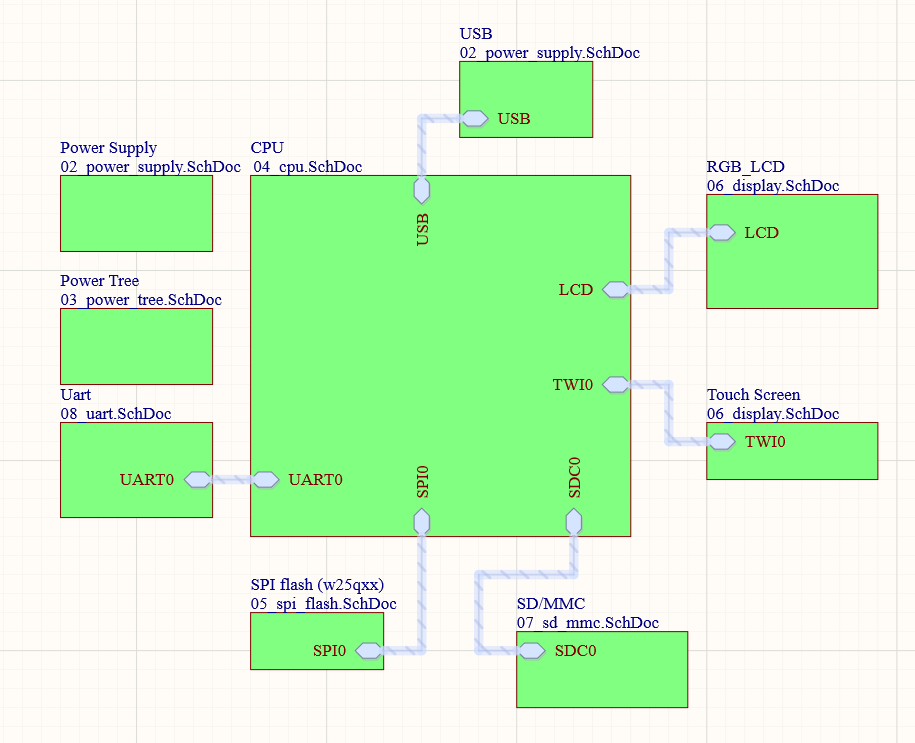
③UI\_TOOLS：将用户界面自定义函数以及页面管理等单独处理做库函数。

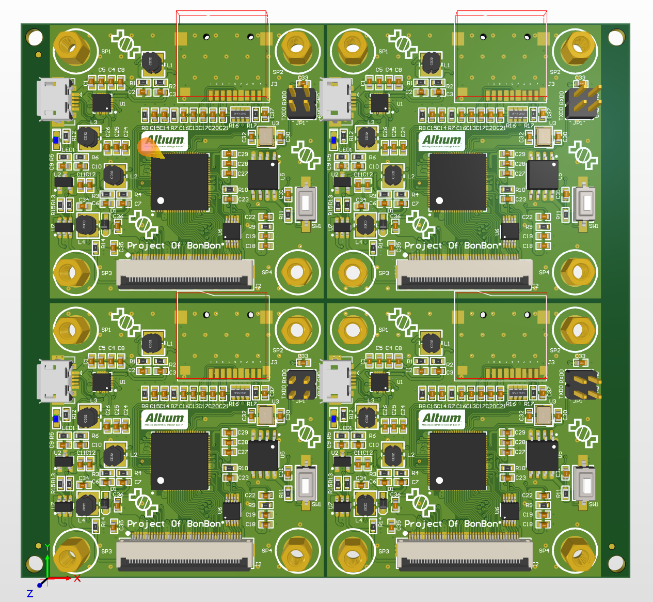
④UTILS：将系统库函数以及第三方软件进行封装。

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

硬件方面，整个硬件框图如下：





整个硬件框架采用模块化式的设计，方便修改拓展。硬件方面大致分为以下几个模块：

USB、供电、电源树、CPU、LCD及触摸屏、Flash芯片、TF卡或eMMC、串口。

4系统详细设计

## 硬件部分

4.1 PCB工程

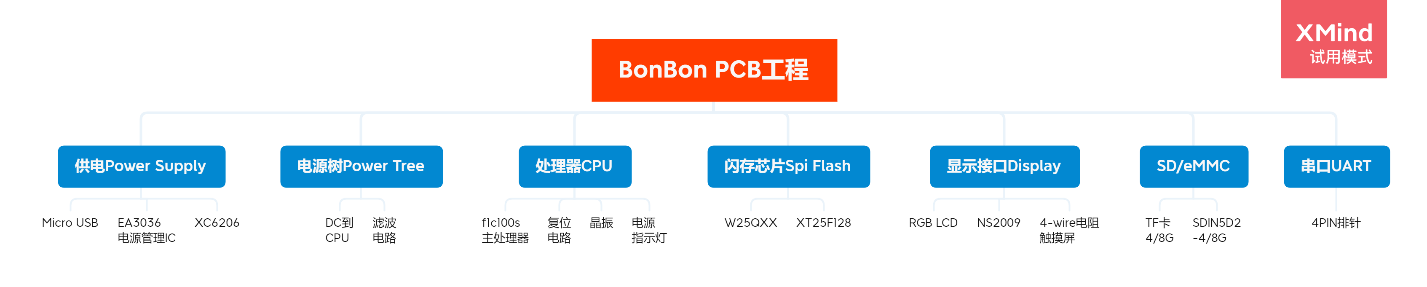
笔者在撰写这篇文章时，整个PCB工程经过了多次修订，尽管如此，各模块核心作用并不会发生太大变化，在这里仅仅展示最初版本的PCB工程。

### 4.1.1工程简述

在这里我对整个PCB工程做一个大体上的叙述。

### 4.1.2 PCB工程结构

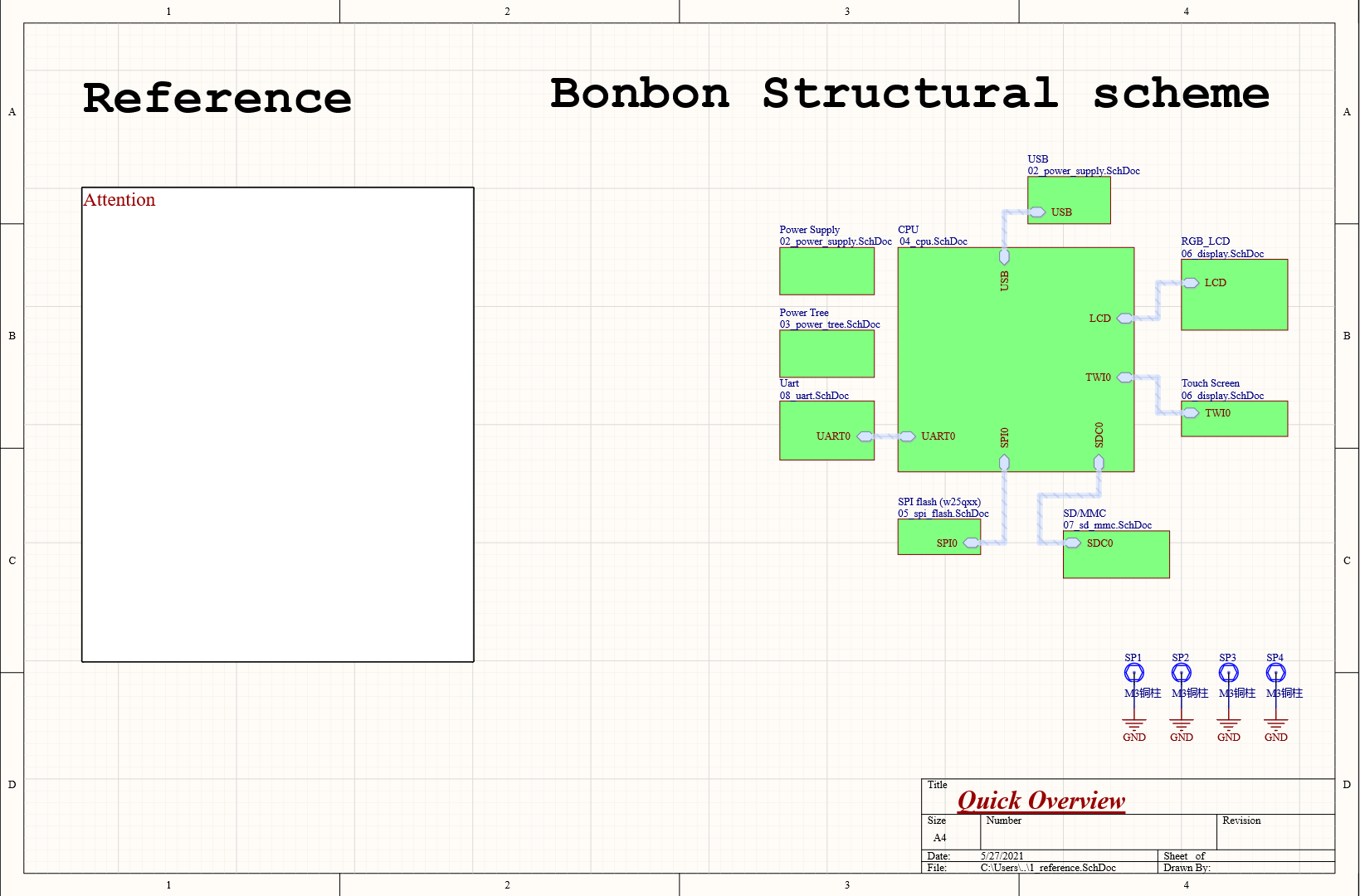
PCB工程结构如下图所示，按照模块化式的设计，功能划分明确，使整个工程一目了然。



### 4.1.3 PCB工程框图

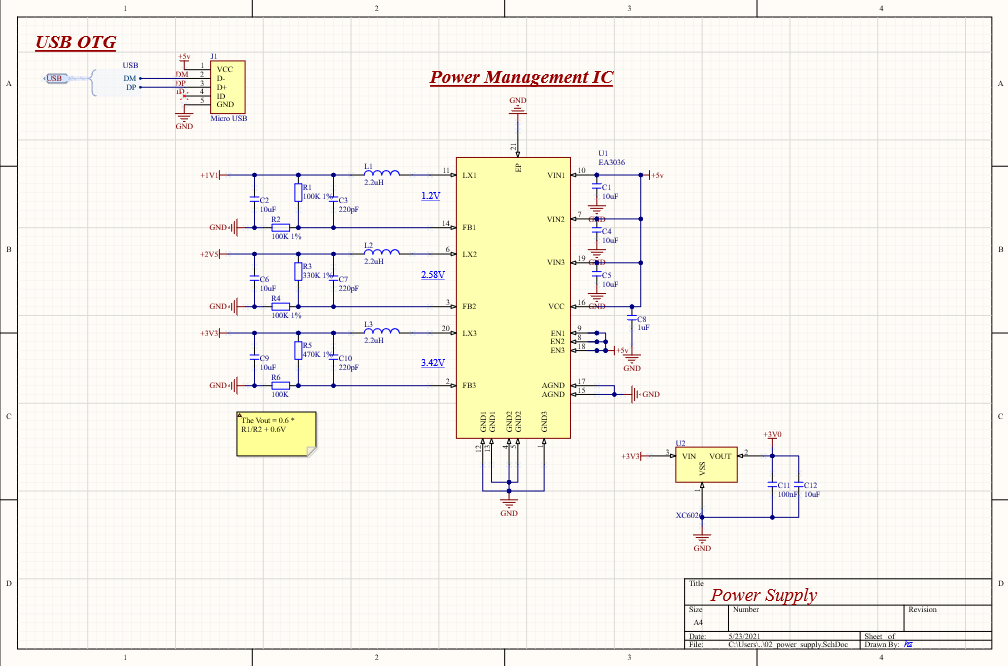
这里使用了Altium Designer的页面符功能，通常在第一个原理图文件中将各模块功能细分，用信号线表示各模块之间的作用关系以及连接性，例如下图：

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。



## 4.2供电Power Supply

对于供电部分，到目前为止我做了两版设计，在最初版本中我使用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，我在后来的版本中使用了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。（价格方面：EA3036平均单价0.97￥，JW5211平均单价0.15￥；封装方面：EA3036为3m x 3m QFN20封装，JW5211为SOT23-5封装）若无特殊说明，各模块原理图皆为最初版本。

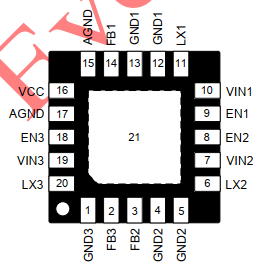


### 4.2.1 Micro USB

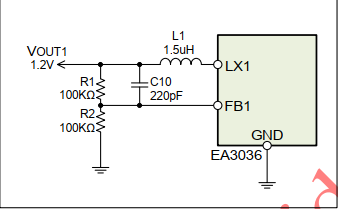
下图为供电Power Supply原理图，分为DC-DC以及LDO

我使用Micro USB接口作为+5V电源输入源，全志科技F1C100S芯片DM、DP引脚连接至USB DM、DP上，方便后续使用sunxi-tools烧写固件。

### 4.2.2 DC-DC EA3036电源管理IC

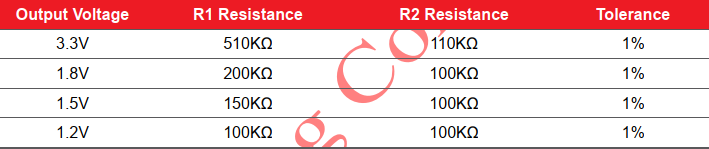
EA3036是一款3CH电源管理IC，适用于由一个锂离子电池或直流5V适配器供电的应用。输出电压范围为0.6V到Vin，最大持续负载电流2A，应用场景有智能手机、IP Camera、数字相机等。如下所示为QFN封装图：

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下图所示：

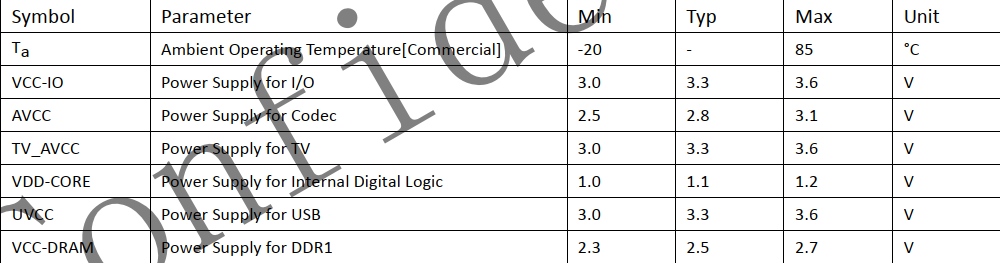


通道输出电压计算公式：

下表列出了通用输出电压以及对应的R1，R2电阻值参考。



根据全志F1C100S芯片手册第18页5.2节推荐工作条件（Recommended Operating Conditions）表格



F1C100S需要以下参数的电压值

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

AVCC取3.0V

VCC-DRAM取2.6V

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

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

第二组R1=330K R2=100K，套入公式得Vout=2.58V 符合要求

第三组R1=470K R2=100K，套入公式得Vout=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量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：

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

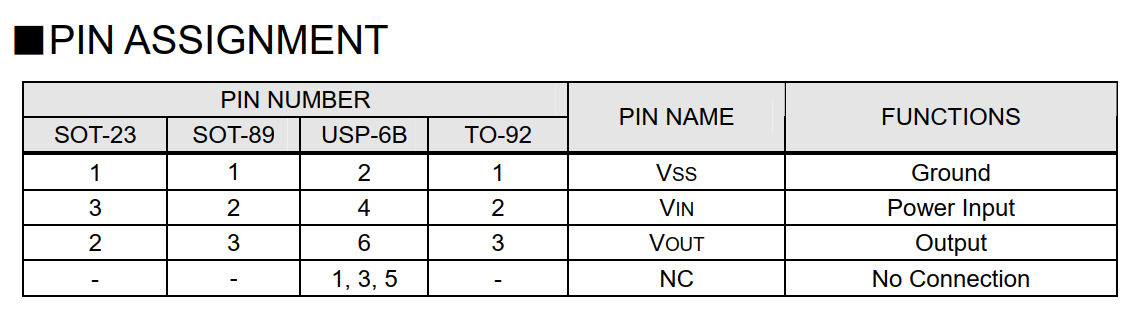
XC6206系列芯片在提供大电流的同时压降极小，XC6206由限流电路，驱动晶体管，精密基准电压和误差矫正电路组成。

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。

模块原理图如下：

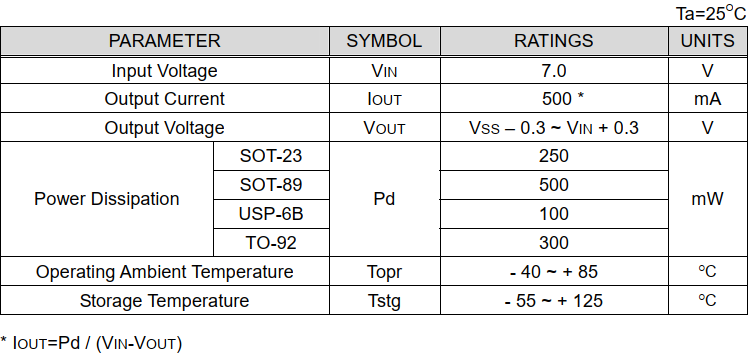
### 

引脚定义

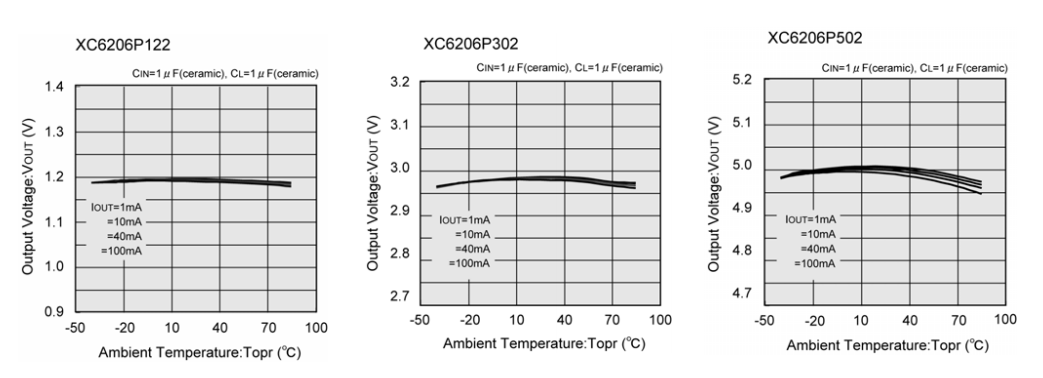


**电气特性**

这里列举极限参数



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，但是根据手册中如下图表，不难推断出其命名规则。



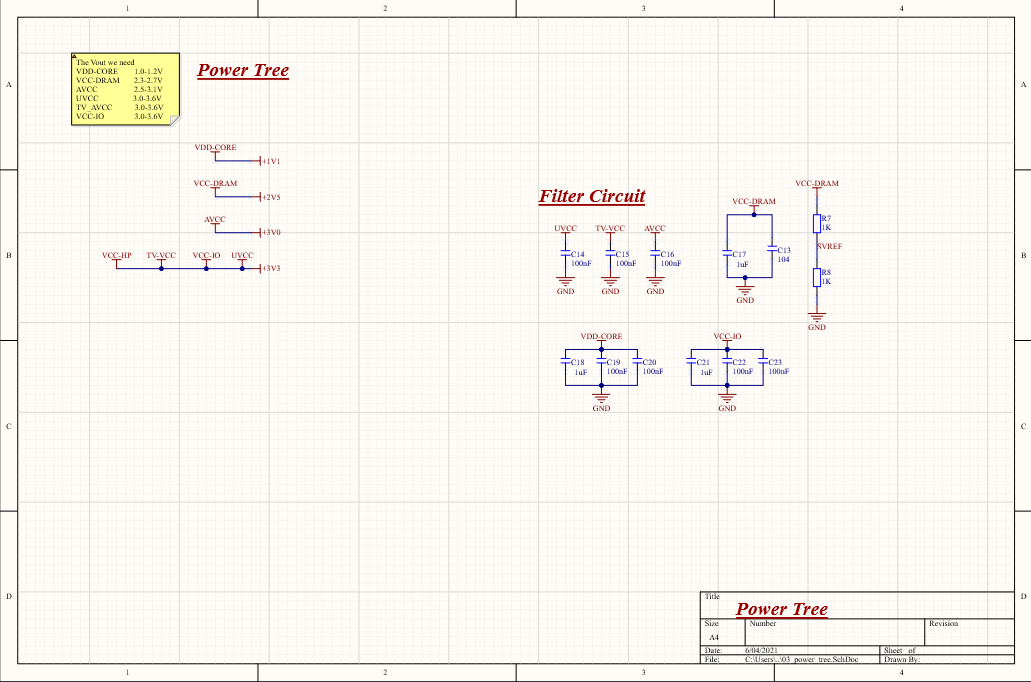
型号P122输出1.2V电压值，型号P302输出3.0V电压值，型号P502输出5V电压值

综上所述，XC6206P之后两位数字即输出电压值Vout，这里需要3V电压，所以应该选择XC6206302型号

丝印65Z5，实物图如下：，SOT-23样品单价0.1元

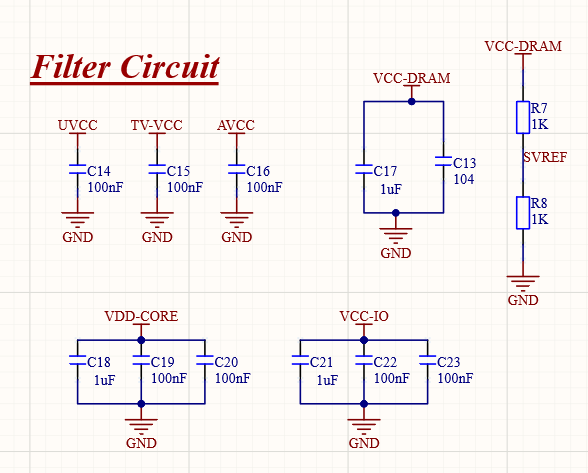
## 4.3 电源树 Power Tree

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。

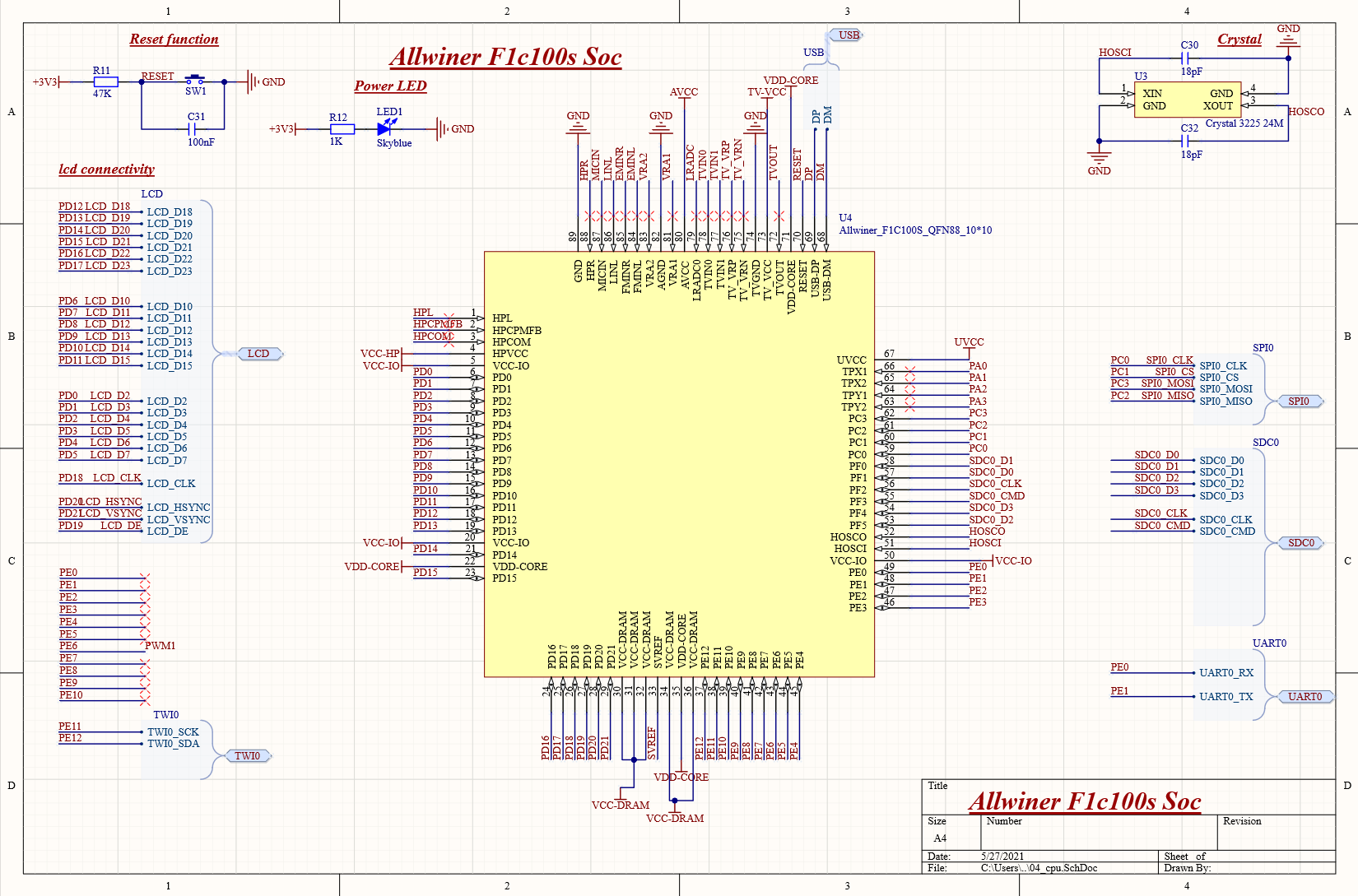


### 4.3.1 滤波电路

用于滤去整流输出电压中的纹波



## 4.4 CPU



### 4.4.1 CPU

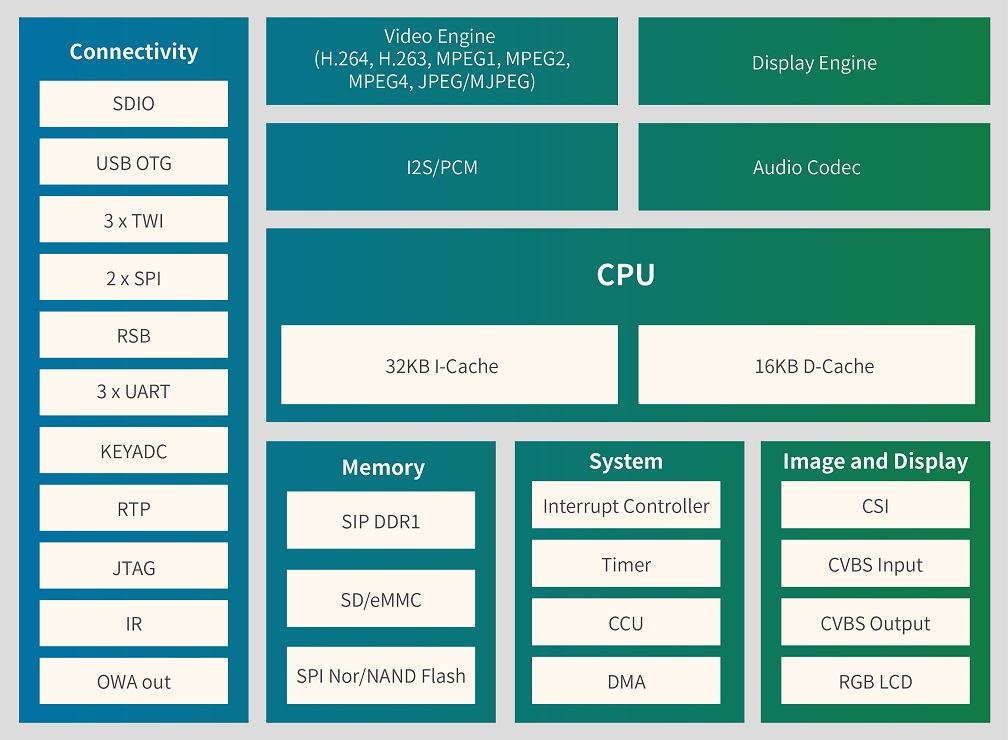
对于CPU部分，我从F1系列中选择了两款芯片，在最初版本中使用的是F1C100s，在后面的版本中随着软件体积的增大需要更换为F1C200s[[3]](#footnote-3)，F1C200s拥有更大的内存，可以实现更加复杂的软件功能。

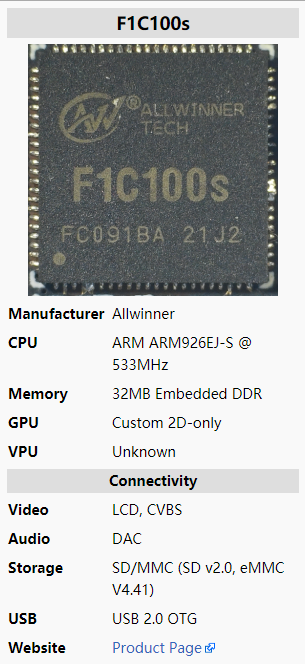
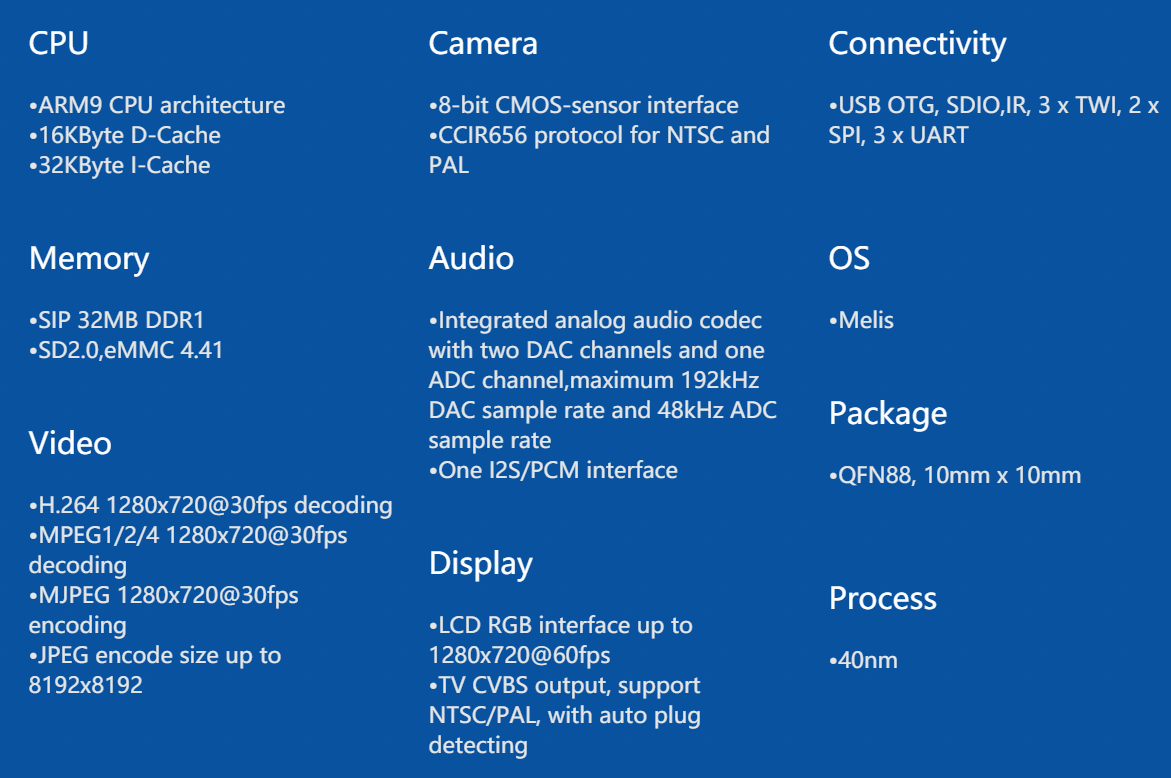
**芯片介绍**

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



### 4.4.3 Boot启动方式（Boot type）

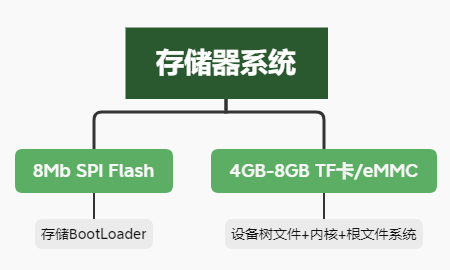
根据芯片手册可以得知F1C100s支持3种启动方式，分别如下：

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[[4]](#footnote-4)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Allwiner 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bootloader、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，我使用如下比较常用的结构的存储结构。



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[[5]](#footnote-5)过的boot.cmd[[6]](#footnote-6)文件放到TF/eMMC的第一分区分区中，将设备树文件和内核放到根文件系统/boot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。

1. TWI[[7]](#footnote-7)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。

1. 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，让其工作与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。

1. 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程度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。

1. 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。

1. 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。

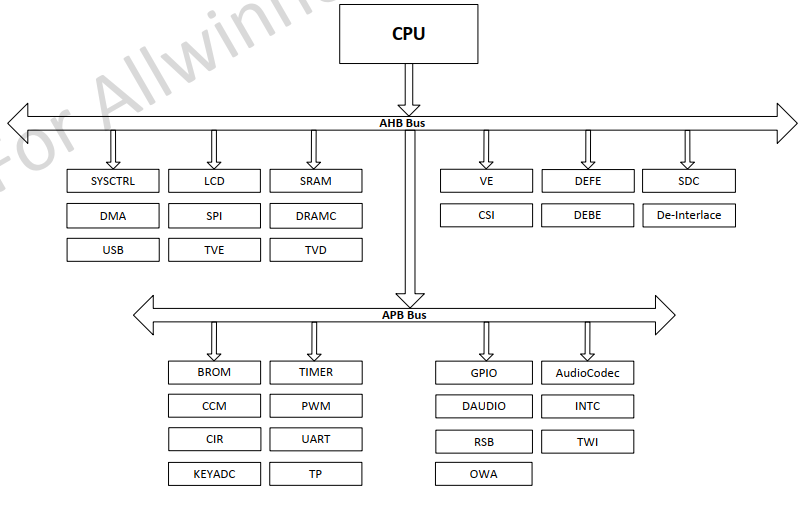
### 4.4.5 外部晶振时钟（Crystal）

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的晶振需要连接咱引脚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。如下图所示。

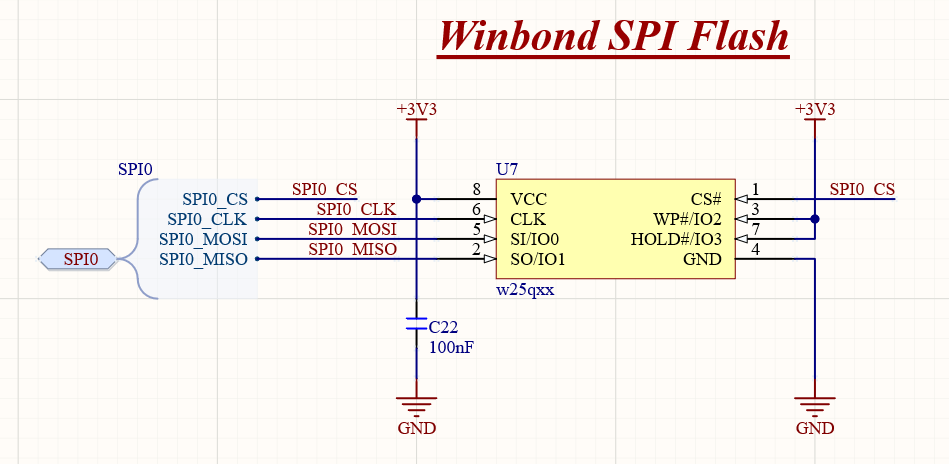


对于时钟树具体结构，就不放在正文中了，可参考附录。

## 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



### 4.5.2 模块命名问题

为什么模块要模块命名为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[[8]](#footnote-8)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提供更高的传输速率的情况下，则需要使用更多的IO接口，秉着简单易用的原则，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，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软件方面继续讨论。

### 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[[9]](#footnote-9)模式。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，为什么呢？我将在后边软件方面继续讨论这个问题。

### 4.5.6 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[[10]](#footnote-10)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

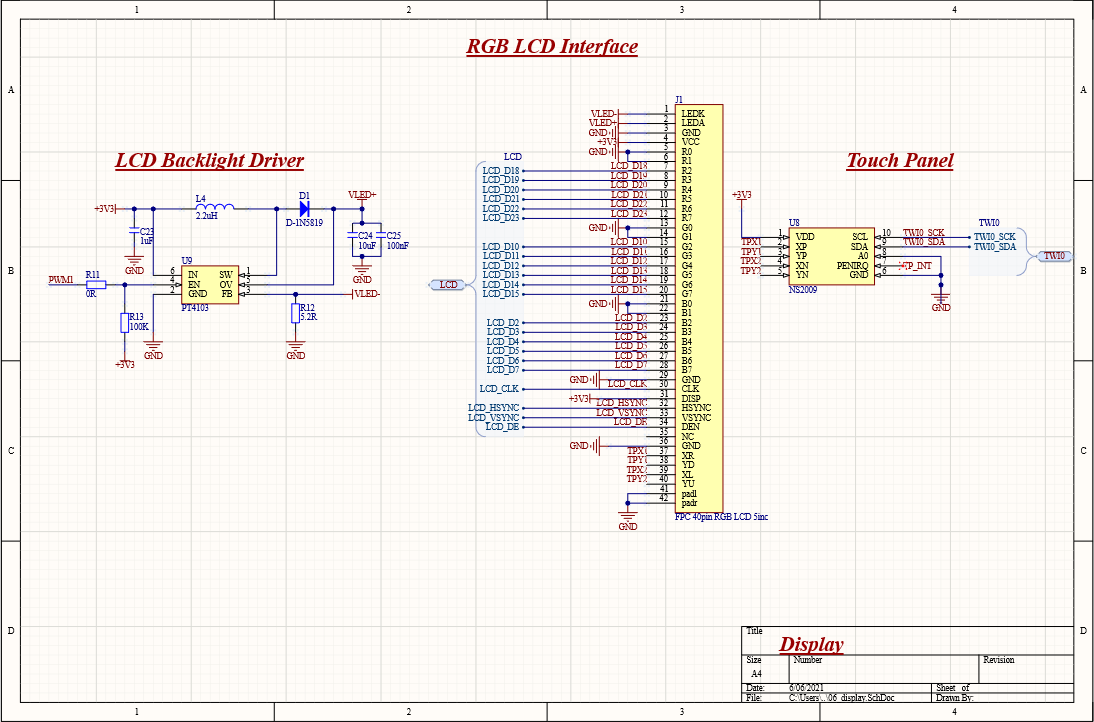
其余位全部都是只读的，并且不会受写寄存器指令影响。对状态寄存器的写访问由非易失性寄存器的SRL位的状态控制，状态寄存器锁定位SRL是状态寄存器（S8[[11]](#footnote-11)）中的易失性/非易失性，可读/可写位。当SRL置为0时，状态寄存器解锁。其他寄存器设置大同小异，不再叙述。

，

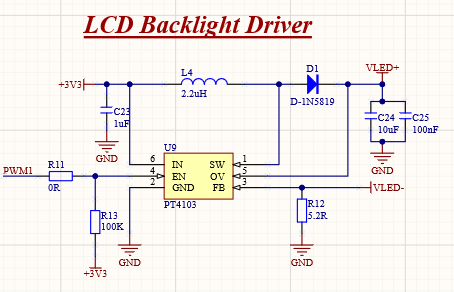
## 4.6 显示模块Display

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



### 4.6.2 LCD背光驱动电路



**芯片介绍**

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，Vin输入电压范围要在控制在2.5V~6V，Vsw输出电压范围在Vin~28V之间，所以这块芯片理论上能最大驱动20多颗串联LED。

**模块说明**

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。另外一种是尺寸大小为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。

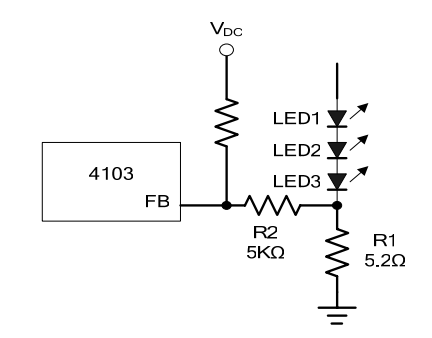
连接到芯片FB引脚的电阻用于控制LED电流大小，其计算公式为：

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，导入公式计算得大小为5.2Ω。

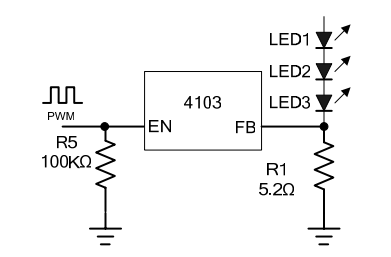
**调光控制**

PT4103芯片手册中，官方给出了三种调光电路，分别是：

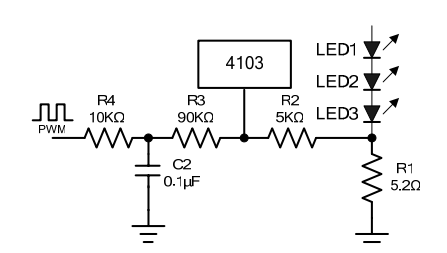
1. 使用直流电压



1. 使用PWM信号控制EN使能端口

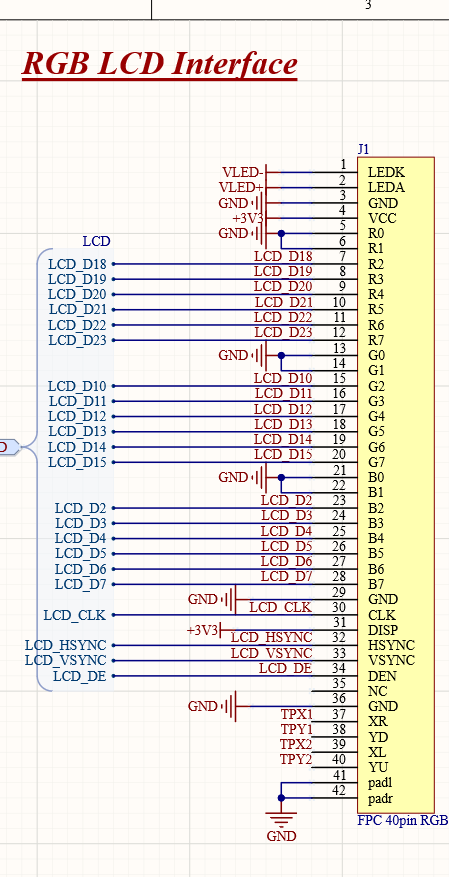


1. 使用滤波后的PWM信号



在这里使用的是第二种调光控制电路，即通过PWN信号控制EN使能端口进行背光强度的调整，其原理是这样的，通过调整PWM波占空比，根据频闪间隔调整亮度。

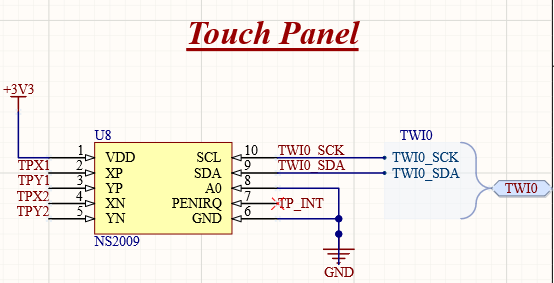
### 4.6.3 40PIN TFT LCD 24位RGB接口



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，VSYNC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

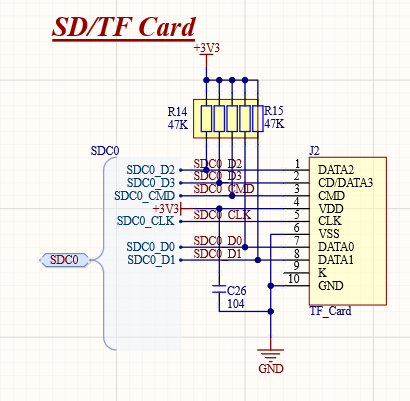


**模块介绍**

NS2009是使用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TWI0接口，A0端口用来表示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+，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。

**坐标值的转换**

## 4.7 SD\_eMMC



**模块介绍**

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。

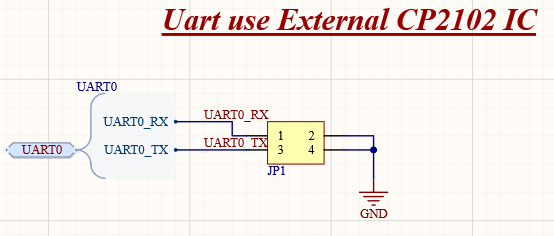
参考网上资料得知，当接口工作于SD模式时，为了避免引脚浮空，除CLK外所有引脚上拉到3.3V，CLK，CMD和DAT在插卡的状态都可能高阻。不能下拉的原因是CMD，DAT数据的起始标志是0。

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

### 4.8.1模块原理图

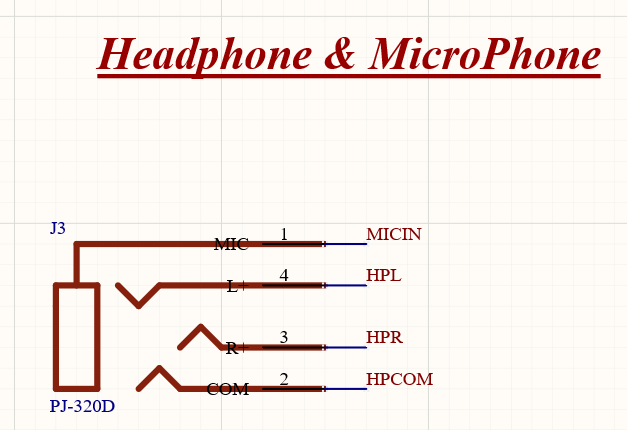


**模块介绍**

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

### 4.9.1模块原理图



**模块介绍**

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。

## 软件部分

## 4.10 U-Boot适配

根据Sunxi官网上的[U-Boot/Legacy U-Boot](http://linux-sunxi.org/U-Boot/Legacy_U-Boot)章节我们可以完成U-Boot到板子的移植，以下是官方原文：

This page provides installation instructions for the legacy unmaintained u-boot-sunxi fork. It is only useful for the devices, which are still not supported by the Mainline U-Boot.

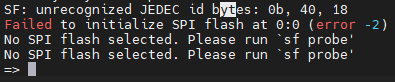
大体意思是说，我们可以使用sunxi fork的U-Boot非主线分支，经过一系列的修改，将其安装到我们自己的设备上，仅适用于未被主线U-Boot支持的设备。

## 4.11 U-Boot修改

在硬件部分我们说到，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[[12]](#footnote-12)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。确认板子可以从SPI Flash启动之后，进行下一步。

设备首次上电，从串口接收到如下信息

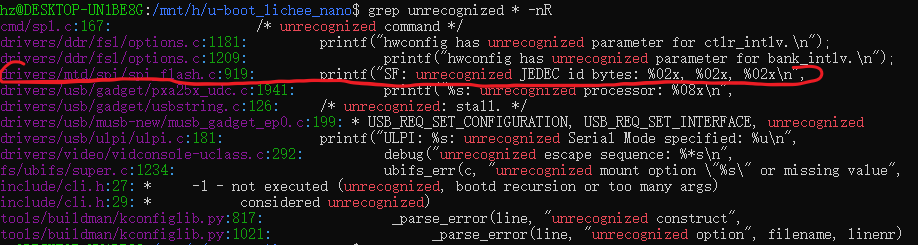


从uboot提供的启动信息可以得知

uboot以SPI方式启动，卡在SF: unrecognized JEDEC id bytes: 0b, 40, 18。表示未识别的JEDEC id

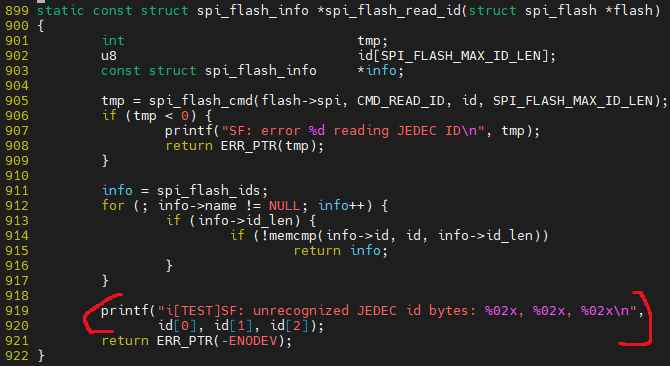
初步分析问题原因是我这块板子上的spi flash的信息并没有添加到uboot中，导致uboot无法识别。

使用grep工具查找打印出这段信息的函数。



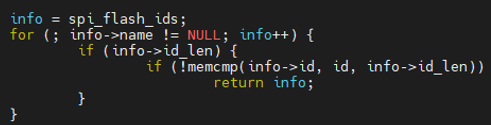
可以看到 drivers/mtd/spi/spi\_flash.c文件中第919行打印出了该信息，使用如下指令打开文件查看

发现这三个id数值是从数组id中获取的



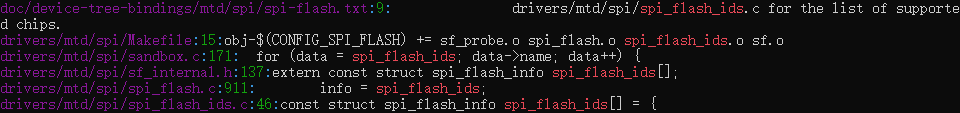
这里将info中的id注意与通过函数spi\_flash\_cmd获取到的id进行比较

本例中结果是比对失败，将id中的信息打印



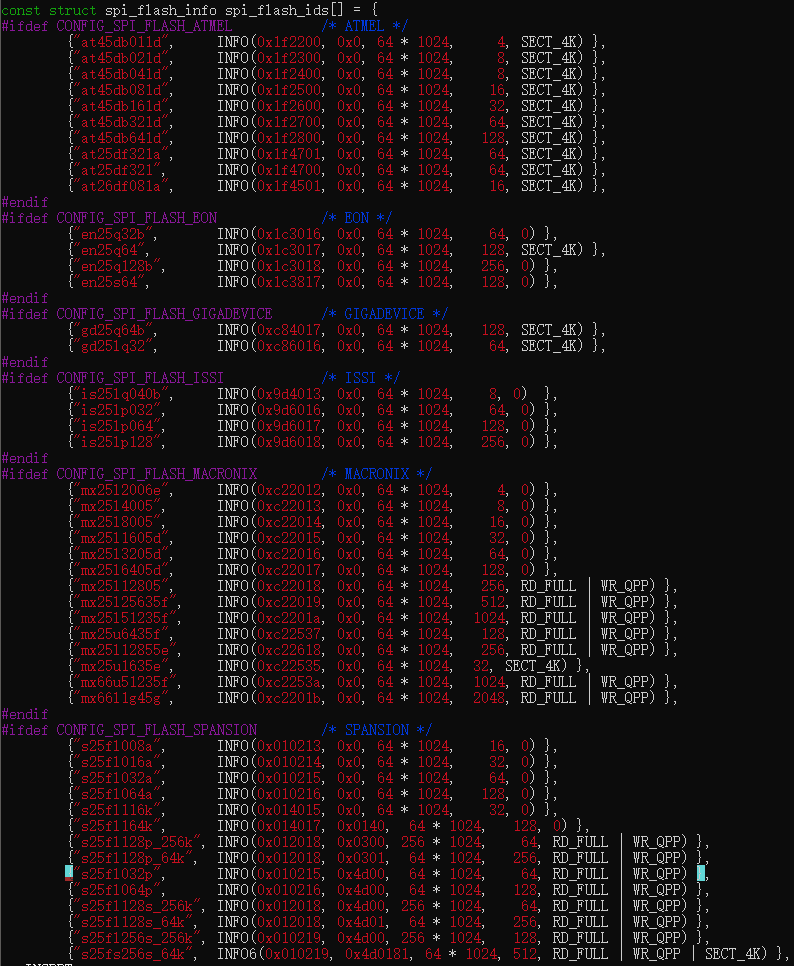
查看info定义

使用grep工具查找spi\_flash\_ids的定义



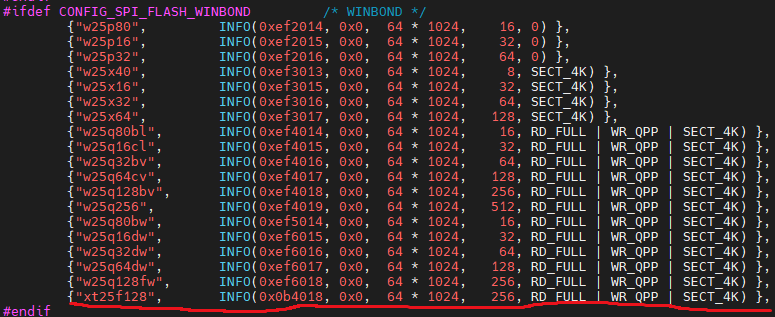
推测中存放spi flash JEDEC信息

打开文件查看

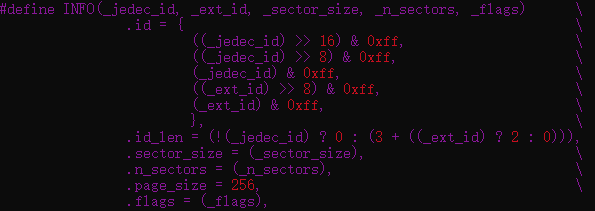


可以看到这里有大量的预先定义好的各种品牌flash信息

我们来到文件底部，仿照上方代码，定义一个spi flash信息



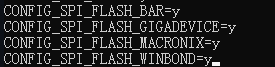
在文件的头部，有宏INFO的定义如下：



第一个参数即为JEDEC id

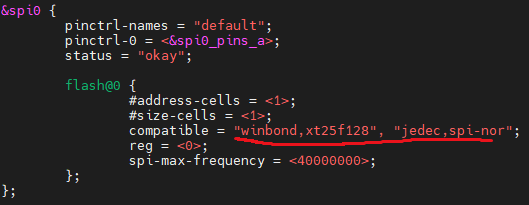
修改完成之后我们要查找这些宏（CONFIG\_SPI\_FLASH\_WINBOND）在哪里被控制开启关闭

结果显示在configs目录下，查看配置文件

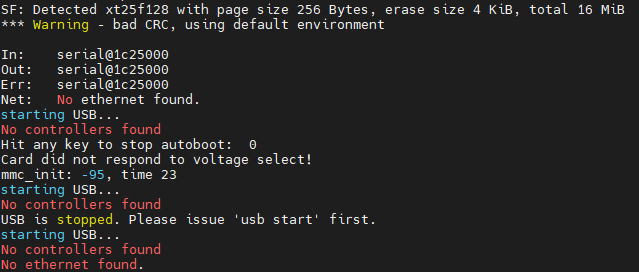


显示宏已经打开，无需其他操作。

最后修改设备树文件arch/arm/dts/suniv-f1c100s-licheepi-nano.dts中的spi0节点中的compatible属性



修改完成后，编译测试，观察到U-Boot打印如下信息，至此SPI Flash适配完成。



## 4.12 Linux Kernel适配

## 4.13 Linux Kernel修改

## 4.14 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[[13]](#footnote-14)，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[[14]](#footnote-15)，所以自动进入FEL模式。

编译好的u-boot文件、设备树文件、zImage内核文件和boot.cmd文件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。

## 4.15 系统的启动流程

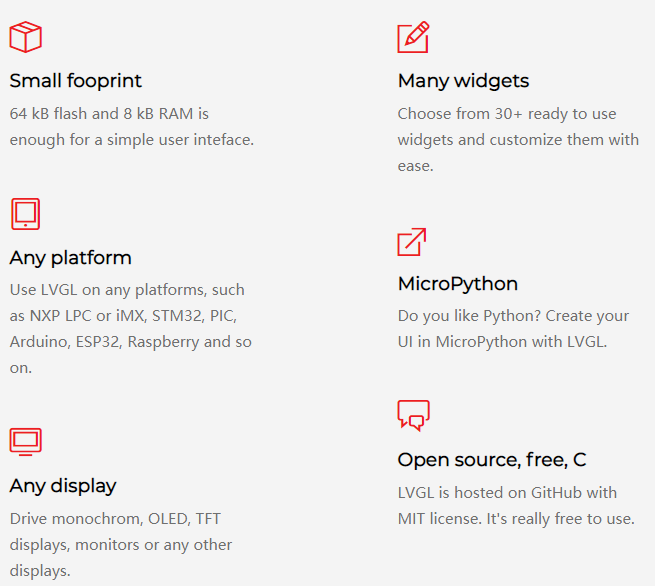
CPU上电后会执行BROM中的程序，这也就是说

## 4.16 Linux驱动开发

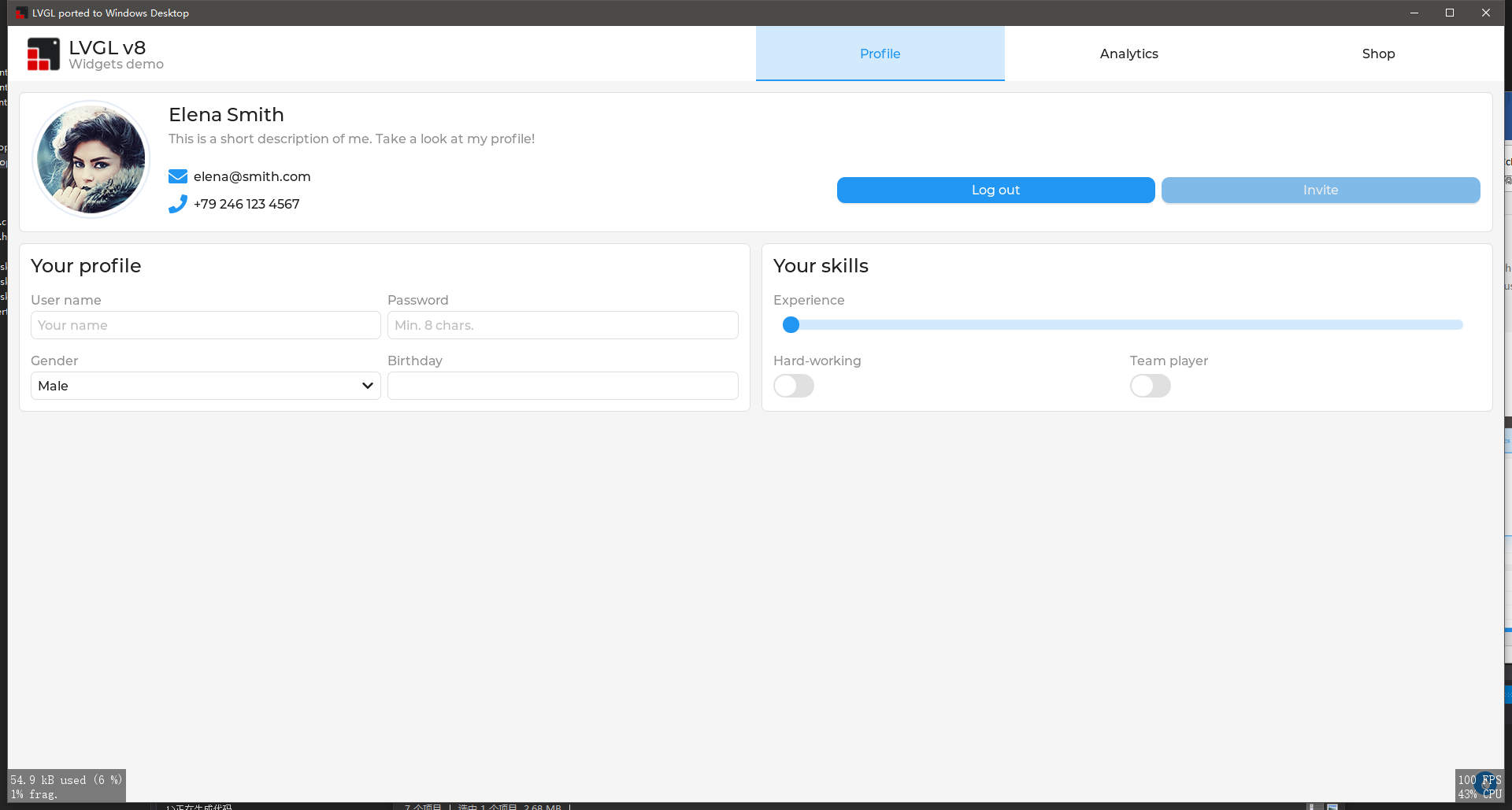
## 4.17 LVGL的移植

**关于LVGL**

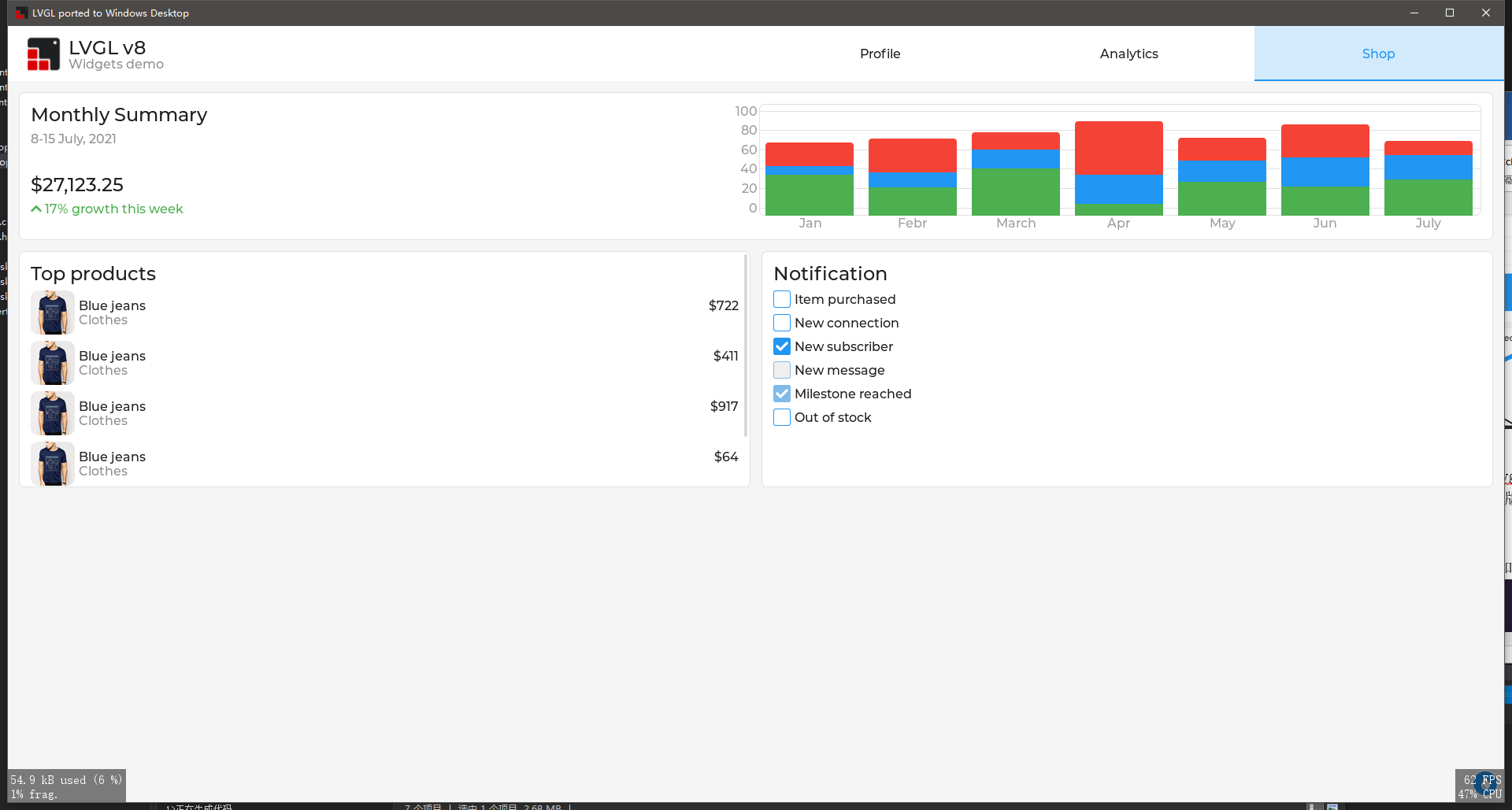
LVGL(Light and Versatile Graphics Library) 是一个用于显示和触摸屏的轻量级嵌入式库，提供构建功能齐全的嵌入式GUI所需的一切。它拥有如下特点：



**效果预览图**



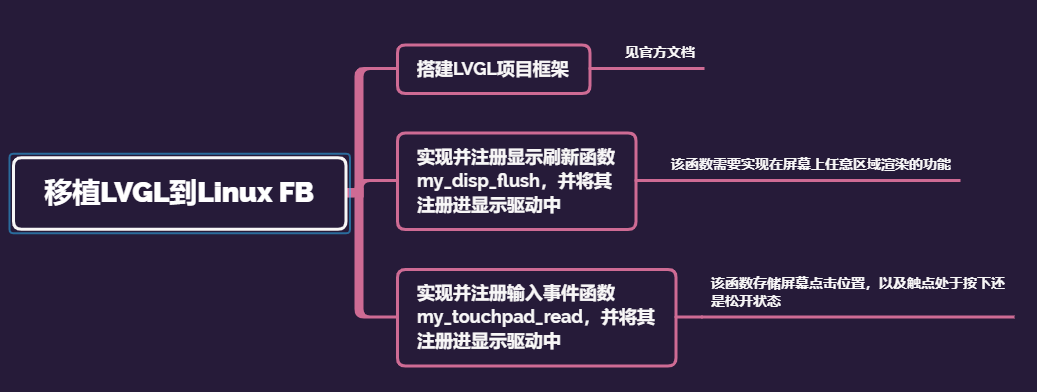




我在开始这个项目的时候使用的是lvgl V7版本，在我写这篇论文的时候，lvgl最新版已经是V8了，V8版本优化了很多问题，提供了一些新的特性，我正积极的将应用迁移到V8版本中。

下面的移植操作仅适用于V7版本，略微修改后可以适用于V8版本。

在本案例中，我将LVGL移植到Linux framebuffer，经过初步分析，需要做如下工作：



**实现过程**

**显示部分**

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？

打开设备节点/dev/fb0

通过ioctl获取fb\_var\_screeninfo

计算行宽、单像素宽度、屏幕像素数量

映射framebuffer到内存中

清除整个屏幕

做完这些操作之后你就可以在my\_disp\_flush[[15]](#footnote-16)函数中使用memcpy函数向framebuffer所在的内存中存储数据了[[16]](#footnote-17)。实现代码见附录。

**输入设备部分**

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：

同步事件 EV\_SYN

用来间隔事件

按键事件 EV\_KEY

压力值 BTN\_TOUCH

绝对位移事件 EV\_ABS

触点ID ABS\_MT\_TRACKING\_ID

X坐标 ABS\_MT\_POSITION\_X ABS\_X

Y坐标 ABS\_MT\_POSITION\_Y ABS\_Y

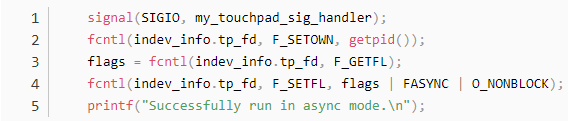
这些事件已经足够足够我们完成输入系统的移植了。

**输入事件的处理**

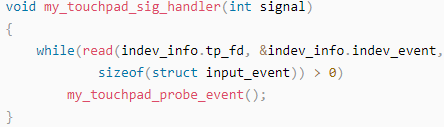
1. 异步通知

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下设置



我使用的信号处理函数



我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。

1. POLL机制（影响动画刷新速率）

这里需要特别注意的是poll的定时时间会直接影响屏幕的刷新速度，所以我特别建议你将该时间设置为<=5ms，小于等于官方建议的系统响应时间。该数值越小，动画刷新越流畅。

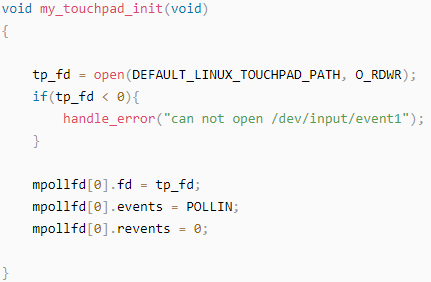
我采用的poll定时（INPUT\_SAMEPLING\_TIME）为1ms。

这部分的移植比我想象中的要复杂一些，我实现了两个函数，分别是my\_touchpad\_init和my\_touchpad\_thread。

先看第一个函数my\_touchpad\_init

这个函数仅仅是打开了/dev/input/event1，并设置了pollfd结构体数组的fd、events 、revents

函数代码



再来看第二个函数my\_touchpad\_thread

这个函数用来处理输入事件并存储事件值，先调用poll实现poll机制

int poll(struct pollfd \*fds, nfds\_t nfds, int timeout);

调用read读取输入设备中的数据

通过结构体input\_event中的type来区分事件从而存储code值

需要**注意**的是，我们应该通过一个独立的任务来处理这些数据，LVGL中提供了创建任务的函数

lv\_task\_t \* lv\_task\_create(lv\_task\_cb\_t task\_xcb, uint32\_t period, lv\_task\_prio\_t prio, void \* user\_data)

我们使用该函数创建一个线程来接收输入事件



my\_touchpad\_thread函数相关代码见附录。

## 4.18 多种开源项目的移植

4.18.1 tslib

4.18.2 freetype

4.18.3 libpng

4.18.4 mplayer

## 4.19 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，LVGL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，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，但最后能够拿出来见人的应用demo也仅仅只有2个，它们分别是自动售货机应用和应用桌面。

### 4.19.1 自动售货机应用

### 4.19.2 应用桌面

## 4.20 使用CMake进行Makefile的构建

## 4.21 多线程的应用

## 4.22 TCP/IP通信的使用

## 4.23 基于GTK、FLTK的高级应用编写

# 参考文献

[1]李树, 陈永编著. 材料工艺学[M]. 北京: 化学工业出版社, 2000: 98-107.

[2]Dobbs J M, Wong J M. Modification of supercritical fluid phase behavior using polor coselvent[J]. Ind.Eng.

Chem. Res, 1987, 10(5): 26-56.

[3]刘武, 姜础. 元谋古猿牙齿测量数据的统计分析及其在分类研究上的意义[J]. 科学通报, 1999, 44(23): 2481-2488.

# 附录

重要的测试结果、图表、程序等可列在附录中。

# 致谢

致谢正文致谢正文……

×××

2018年×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 | | |  | | |
| 学院(校区) | |  | | 综合成绩(Y) |  |
| 专 业 | |  | | 班 级 |  |
| 学生学号 | |  | | 学生姓名 |  |
| 指导教师评语 |  | | | | |
| (满分40分)成绩(X1)： 指导教师签名： 年 月 日 | | | | |
| 答辩组评  语 |  | | | | |
| (满分60分)成绩(X2)： 答辩组组长签名： 年 月 日 | | | | |

1. sunxi-tools即全志科技为其旗下Soc芯片推出的固件烧写工具 [↑](#footnote-ref-1)
2. 项目中**绝大多数**部分已经有解决方案，但仍有**少许**仍处于探索实验阶段。限于时间紧迫，最初版本仅能实现部分功能 [↑](#footnote-ref-2)
3. F1C200s为100s的增强版本，根据官方介绍，其相比100s略有差别，笔者在撰写这篇文章时还未使用200s进行替换。 [↑](#footnote-ref-3)
4. Sunxi代表着中国无晶圆厂半导体公司Allwiner Technology的ARM SoC家族。 [↑](#footnote-ref-4)
5. 这里的预处理指的是使用mkimage工具给boot.cmd文件添加uboot头部信息，指令格式参考附录。 [↑](#footnote-ref-5)
6. Boot.cmd文件中存放着启动阶段必要的配置信息，常见格式见附录。 [↑](#footnote-ref-6)
7. TWI（Two—wire Serial Interface）接口是对总线接口的继承和发展，完全兼容总线，具有硬件实现简单、软件设计方便、运行可靠和成本低廉的优点。 [↑](#footnote-ref-7)
8. 即Winbond品牌。 [↑](#footnote-ref-8)
9. FEL是一个包含在所有Allwiner 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。 [↑](#footnote-ref-9)
10. 页大小即256KB [↑](#footnote-ref-10)
11. 这里容易混淆概念，S8即状态寄存器2的最低位，芯片手册中将3个寄存器的24个位依次排开，例如状态寄存器2中的8位为S8~S15 [↑](#footnote-ref-11)
12. 这里的U-Boot是sunxi分支修改过的非主线U-Boot，其中已经添加了大量的本地化功能。 [↑](#footnote-ref-12)
13. 官方建议，完全关闭电源可能需要移除串口上的设备，因为这可能使得CPU处于微工作状态。 [↑](#footnote-ref-14)
14. SPI方式启动具有最低级别的优先级 [↑](#footnote-ref-15)
15. 函数原型见附录 [↑](#footnote-ref-16)
16. 像素点在framebuffer中的位置：framebuffer起始地址 + y \* 横向屏幕像素宽度 + x \* 像素宽度 [↑](#footnote-ref-17)