密级状态：绝密( ) 秘密( ) 内部( ) 公开( √ )

**Rockit项目概要设计**

（技术部，系统产品一部）

|  |  |  |
| --- | --- | --- |
| 文件状态：  [√] 正在修改  [ ] 正式发布 | 当前版本： | V0.2 |
| 作 者： | Martin.Cheng |
| 完成日期： | 2018/6/25 |
| 审 核： |  |
| 完成日期： |  |

福州瑞芯微电子有限公司

Fuzhou Rockchips Semiconductor Co . , Ltd

(版本所有,翻版必究)

版 本 历 史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本号 | 作者 | 修改日期 | 修改说明 | 备注 |
| V0.2 | Martin | 2018/6/25 | 初始版本 |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

相 关 文 档

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本 | 作者 | 最后修改 | 文档名称 | 完成度 |
| V0.2 | Martin | 2018/6/25 | Rockit项目实施计划.docx | ☆ |
|  |  |  | Rockit需求分析文档.docx | ☆ |
|  |  |  | Rockit概要设计文档.docx | 无 |
|  |  |  | Rockit详细设计文档.docx | 无 |
|  |  |  | Rockit软件测试文档.docx | 无 |

Rockit概要设计文档

[1. Rockit概要设计文档 5](#_Toc517705588)

[1. 引言 5](#_Toc517705589)

[1.1 编写目的 5](#_Toc517705590)

[1.2 项目风险 5](#_Toc517705591)

[1.3 预期读者和阅读建议 5](#_Toc517705592)

[1.4 参考资料 5](#_Toc517705593)

[2. 设计概述 6](#_Toc517705594)

[2.1 限制和约束 6](#_Toc517705595)

[2.2 设计目标 6](#_Toc517705596)

[2.3 设计原则和设计要求 7](#_Toc517705597)

[2.3.1 代码风格 7](#_Toc517705598)

[2.3.2 版本管理 7](#_Toc517705599)

[2.3.3 代码自动测试工具 7](#_Toc517705600)

[2.3.4 内存泄漏检测工具 8](#_Toc517705601)

[2.3.5 面向对象的基本原理 8](#_Toc517705602)

[3. 软件架构设计 9](#_Toc517705603)

[3.1 Rockit应用层设计 10](#_Toc517705604)

[3.2 Rockit组件层设计 10](#_Toc517705605)

[3.3 Rockit原语层设计 11](#_Toc517705606)

[3.3.1 媒体原语层设计 11](#_Toc517705607)

[3.3.2 操作系统原语层设计 11](#_Toc517705608)

[4. Rockit工作计划 12](#_Toc517705609)

[4.1 预研部分 12](#_Toc517705610)

[4.2 系统详细设计 12](#_Toc517705611)

[5. 模块预研评估与结论 12](#_Toc517705612)

[5.1 代码管理预研(评估和结论) 12](#_Toc517705613)

[5.1.1 代码版本管理工具 13](#_Toc517705614)

[5.1.2 代码格式管理 13](#_Toc517705615)

[5.1.3 项目模块和目录规划 13](#_Toc517705616)

[5.1.4 cmake跨平台 13](#_Toc517705617)

[5.1.5 第三方库的优雅管理(ffmpeg) 13](#_Toc517705618)

[5.2 持续集成预研(评估和结论) 13](#_Toc517705619)

[5.3 组件管理预研(评估和结论) 13](#_Toc517705620)

[5.4 原语层预研(评估和结论) 14](#_Toc517705621)

# Rockit概要设计文档

# 引言

引言是对这份软件系统概要设计报告的概览，是为了帮助阅读者了解这份文档是如何编写的，并且应该如何阅读、理解和解释这份文档。

## 编写目的

说明这份软件系统概要设计报告是基于哪份软件产品需求规格说明书编写的，开发这个软件产品意义、作用、以及最终要达到的意图。通过这份软件系统概要设计报告详尽说明了该软件产品的软件结构，从而对该软件产品的结构的描述。

如果这份软件系统概要设计报告只与整个系统的某一部分有关系，那么只定义软件系统概要设计报告中说明的那个部分或子系统。

## 项目风险

具体说明本软件开发项目的全部风险承担者，以及各自在本阶段所需要承担的主要风险，首要风险承担者包括：

* 任务提出者；
* 软件开发者；
* 产品使用者。

## 预期读者和阅读建议

列举本软件系统概要设计报告所针对的各种不同的预期读者，例如，可能的读者包括：

* 开发人员；
* 项目经理；
* 测试人员；
* 文档编写人员；

描述文档中，其余部分的内容及其组织结构，并且针对每一类读者提出最适合的文档阅读建议。

## 参考资料

列举编写软件产品概要设计报告时所用到的参考文献及资料，可能包括：

* 本项目已经批准的计划任务书；
* 开发本项目时所要用到的标准；
* 系统规格需求说明；
* 使用实例文档；
* 属于本项目的其它已发表文件；
* 本软件系统概要设计报告中所引用的文件、资料：
* 相关软件系统概要设计报告：
* 等等。

为了方便读者查阅，所有参考资料应该按一定顺排列。如果可能，每份资料都应该给出：

* 标题名称；
* 文件编号或者版本号；
* 发表日期或者签约日期；
* 出版单位或者资料来源。

# 设计概述

本节描述现有开发条件和需要实现的目标，说明进行概要设计时应该遵循的设计原则和必须采用的设计方法。

## 限制和约束

简要描述起到限制和约束作用的各种可能存在的条件

|  |  |  |
| --- | --- | --- |
| 限制和约束 | |  |
| 技术条件 | 对音视频有3年左右的沉淀，团队有实力开发出一套普适的音视频框架。 | |
| 开发环境 | adroid/linux/windows gcc跨平台开发 | |
| 时间限制 | 7月-10月 | |
| 人员限制 | 3-4名音视频工程师 | |

图表-xx限制和约束

## 设计目标

并且说明在上述条件下，应该实现的系统目标

|  |  |  |
| --- | --- | --- |
| 模块 | 功能需求 | 优先级 |
| 命令行 | 测试命令行程序 | ★ |
| 复用器 | FFMPEG DEMUXER/MUXER | ★ |
| 编解码器 | MediaCodec/FFMPEG(Software)/VPUAPI/MPP | ★ |
| 后处理 | 后处理(AVFilter); Resampler; | ★ |
| 音视频输出 | 视频显示(overlay&opengl) / 音频输出(ALSA/) | ★ |
| 接口要求 | IPTV接口支持  Android MediaPlayer接口; Android版本无关。  蓝光支持  3D支持  音视频倍速播放  缩略图接口 | ★ |
| 高级特性 | 动态调试(日志等)  异常处理  单元测试，接口一致性测试，压力测试代码显示Buffer的管理独立化考虑  KPI(QOS)  配置管理模块 | ★ |
| 测试数据源 | 测试数据源 (生成假的视频流和音频流)  测试码流库 | ☆ |
| 其他接口 | DRM支持(TEE)  编码转码  RTP/RTSP服务器  跨平台(ANDROID/LINUX/WINDOWS) | ☆ |

图表-xx 通用媒体架构功能需求

## 设计原则和设计要求

描述对本软件系统进行概要设计的原则，通常可以考虑以下几方面的内容：

### 代码风格

|  |  |  |
| --- | --- | --- |
| 版本细分 | 备选技术说明 | 重要性 |
| 代码风格 | [Google C++ Style Guide](https://google.github.io/styleguide/cppguide.html)  [C Coding Standard[ece.cmu.edu]](https://users.ece.cmu.edu/~eno/coding/CCodingStandard.html)  [C++ Coding Standard[ece.cmu.edu]](https://users.ece.cmu.edu/~eno/coding/CppCodingStandard.html) | ★ |
| 代码自动格式化 | [Artistic Style 3.1: A Fast Small Automatic Formatter for C, C++](http://astyle.sourceforge.net/astyle.html) | ★ |

图表-xx 代码风格管理

代码风格工具提供代码风格模板和代码风格规约，即C/C++代码风格标准。在版本集成方面，提供代码自动格式化脚本，提交代码时自动格式化。

### 版本管理

|  |  |  |
| --- | --- | --- |
| 版本细分 | 备选技术说明 | 重要性 |
| 代码管理 | github | ★ |
| 文档 | github/doc (markdown) | ★ |
| 开源库管理 | git脚本+patch | ★ |
| 跨平台编译配置 | cmake | ★ |

图表-xx 版本管理技术备选

版本管理工具基本要求是代码管理。代码管理也可以包含: 代码格式和代码风格的自动化管理。另外版本管理流程与文档工具集成，自动化生成文档。

### 代码自动测试工具

|  |  |  |
| --- | --- | --- |
| 目标 | 备选技术 | 重要性 |
| 覆盖率测试 | <https://codecov.io/> (与github集成) | ★ |
| 单元测试工具 | https://github.com/google/googletest | ★ |
| 集成测试 | https://jenkins.io/index.html | ★ |

图表-xx代码自动测试技术备选

当前流程的开源软件开发，流程将自动测试单元与持续集成工具集成，当前在线的持续集成工具有: GitLab Continuous Integration和 Travis CI, 自建搭建持续集成系统Jenkins。使用GitLab Continuous Integration和 Travis CI等商用集成工具，成本略高。

单元测试用例需要考虑各种边界条件。

### 内存泄漏检测工具

|  |  |  |
| --- | --- | --- |
| 目标 | 备选技术 | 重要性 |
| 内存泄露工具 | valgrind | ★ |

图表-xx 代码自动测试技术备选

### 面向对象的基本原理

* 单一职责原则SRP(Single Responsibility Principle)

对于单一职责原则，其核心思想为：一个类，最好只做一件事，只有一个引起它的变化。单一职责原则可以看作是低耦合、高内聚在面向对象原则上的引申，将职责定义为引起变化的原因，以提高内聚性来减少引起变化的原因。职责过多，可能引起它变化的原因就越多，这将导致职责依赖，相互之间就产生影响，从而大大损伤其内聚性和耦合度。

* 开放封闭原则OCP(Open－Close Principle)

对于开放封闭原则，它是面向对象所有原则的核心，软件设计说到底追求的目标就是封装变化、降低耦合。假如基类设计足够优秀，能够覆盖基础和抽象的所有功能。那么为了接口一致性，修改对基类是关闭的，修改对扩展类是开放的。最佳实践: 系统有一套稳定并且抽象的类库定义。

* 替换原则(the Liskov Substitution Principle LSP)

核心思想是：子类必须能够替换其基类，Liskov替换原则是关于继承机制的设计原则。Liskov替换原则能够保证系统具有良好的拓展性，同时实现基于多态的抽象机制，能够减少代码冗余，避免运行期的类型判别。

* 依赖倒置原则(the Dependency Inversion Principle DIP)

依赖倒置原则，其核心思想是：依赖于抽象。具体而言就是高层模块不依赖于底层模块，二者都同依赖于抽象；抽象不依赖于具体，具体依赖于抽象。抽象的稳定性决定了系统的稳定性，因为抽象是不变的，依赖于抽象是面向对象设计的精髓，也是依赖倒置原则的核心。

* 接口隔离原则(the Interface Segregation Principle ISP)

对于接口隔离原则，其核心思想是：使用多个小的专门的接口，而不要使用一个大的总接口。具体而言，接口隔离原则体现在：接口应该是内聚的，应该避免“胖”接口。一个类对另外一个类的依赖应该建立在最小的接口上，不要强迫依赖不用的方法，这是一种接口污染。接口有效地将细节和抽象隔离，体现了对抽象编程的一切好处，接口隔离强调接口的单一性。

# 软件架构设计

充分调研开源多媒体架构之后，开源多媒体架构难于满足我们的需求。比如GStreamer组件化架构过于庞大臃肿，大量组件不是我们必须的，它的第三方库依赖非常多，并且android支持不太完善，即支持Android的特性很少。其他多媒体架构比如: vlc, kodi等，也有类似的问题。

我们充分参考开源多媒体架构的设计思想。Rockit的总体设计目标设定为: Rockit分为应用层，组件层和原语层；应用层负责组件的组装(Pipeline/工作流/流水线)，以确保组件的灵活组装特性；组件层负责封装基本媒体组件(Demuxer, Codec, Filter, Sink), 组件层要求稳定的接口定义，能支持多种开源技术的无缝接入，比如Filter组件派生出与一个OpenCVFilter以支持OpenCV技术等；原语层分层媒体原语和操作系统原语， 原语层也要求稳定的接口定义。



图表-xx 基于组件化的Rockit架构

几个核心问题的思考

* 音视频同步怎么做？

应用层做音视频同步。

* 组件之间如何传递数据，以及如何高效传递数据(零拷贝)

N/A 待确定并实验验证

* 组件层的接口和架构设计？

N/A 待确定并实验验证

* 原语层的内涵及其边界？

N/A 待确定并实验验证

## Rockit应用层设计

Rockit应用层使用Rockit组件层和Rockit抽象层的API，构建多媒体应用。以MediaPlayer的数据流为例，MediaPlayer使用Rockit组件层和Rockit抽象层的API构建播放器应用。



图表-xx MediaPlayer构建的数据流

## Rockit组件层设计



图表-xx Rockit组件层设计

## Rockit原语层设计

### 媒体原语层设计

媒体原语层设计包含媒体操作需要使用的基本数据结构。在架构范围内只能使用原语层的数据结构和函数，以保证最大兼容性。具体设计参考预研文档，主要的数据结构如下：

RTPacket: 基本流数据包的结构体。

RTFrame: 解码帧数据包的结构体。

RTBuffer: 平台无关，架构无关的视频buffer定义

RTQueue: queue定义

### 操作系统原语层设计

RTLock: 互斥锁定义。

RTThread: 线程定义。

RTTask: 任务定义

RTMemory: 平台内存申请和释放封装。

# Rockit工作计划

本节描述Rockit在概要阶段的工作计划，概要设计阶段完成架构设计，并验证架构的可行性。可行性验证将架构化整为零，分模块验证可行性。然后给出架构概要设计评估结论。

## 预研部分

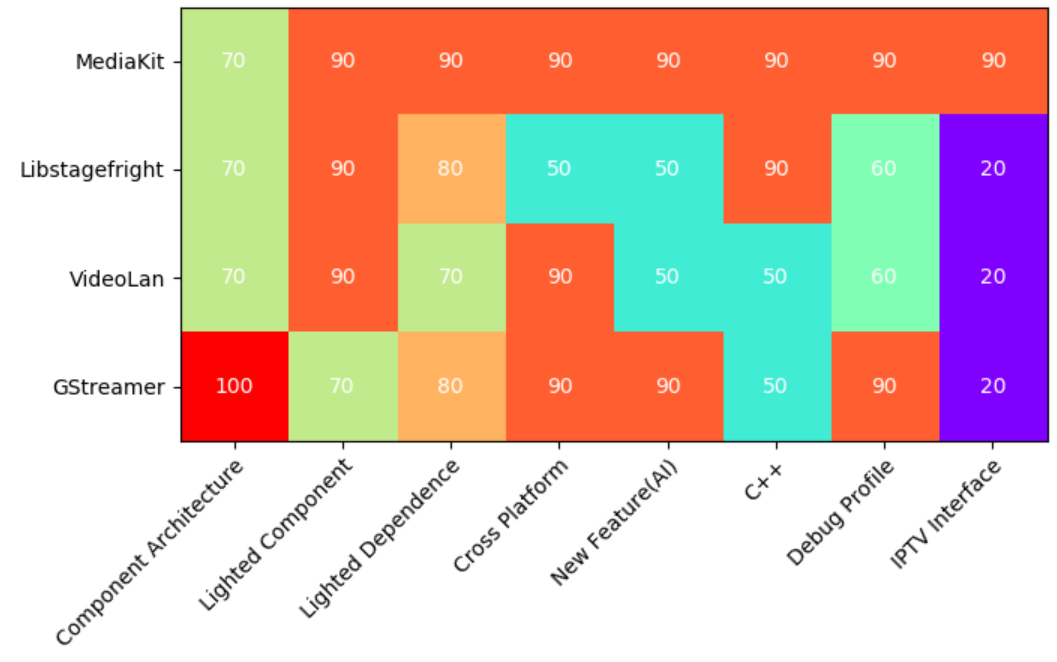
|  |  |  |
| --- | --- | --- |
| 预研项目 | 预研要求 | 重要性 |
| 代码管理预研 | 代码版本管理工具  项目模块和目录规划 | @明传 |
| 代码管理预研 | cmake跨平台  第三方库的优雅管理(ffmpeg)  doc自动生成  代码格式管理 | @丽明 |
| 持续集成预研 | 单元测试选型与评估  内存泄漏工具选型与评估  集成测试选型与评估 | @文杰 |
| 组件管理预研 | 轻量级组件架构选型与评估  组件间数据拷贝选型与评估 | @明传  @小方 |
| 原语层预研 | 媒体原语层选型与评估  操作系统原语层选型与评估 | @明传 |
| 原语层预研 | Buffer管理 | @小方 |
| 应用层预研 | 音视频同步 | @小方 |

图表-xx 预研模块明细

预研部分请记录相关文档，给后续加入的同事，熟悉环境工具所需的背景和原理知识。

## 设计目标细分

Rockit的设计希望能够对标主流的开源多媒体框架。多媒体特性对比热力图。希望我们的架构能在开源社区流行起来。成为一个经典的架构。



图表-xx 多媒体特性对比热力图(满分100)

# 模块预研评估与结论

## 代码管理预研(评估和结论)

预研人员：XXX

|  |  |  |
| --- | --- | --- |
| 预研项目 | 预研要求 | 评估结论 |
| 代码管理预研 | 代码版本管理工具  代码格式管理  项目模块和目录规划  cmake跨平台  第三方库的优雅管理(ffmpeg) | N/A  N/A  N/A  N/A  N/A |

图表-xx 代码管理模块评估和结论

### 代码版本管理工具

工程地址: [https://github.com/Rockit/Rockit](https://github.com/mediakit/mediakit) user: xxx pwd: xxx

### 代码格式管理

代码格式工具: AStyle

代码风格推荐: Google C/C++代码风格

手动代码格式化方法：xxx

### 项目模块和目录规划

|  |  |  |
| --- | --- | --- |
| 模块 | 模块 | 备注 |
| src | 核心源代码 |  |
| third-party | 第三方库 |  |
| third-party/ffmpeg | 第三方库-ffmpeg |  |
| build | 跨平台编译脚本 |  |
| doc | markdown文档 |  |

图表-xx项目模块和目录规划

### cmake跨平台

cmake 版本

cmake的android编译脚本参考 build/android/arm

cmake的linux编译脚本参考 build/linux/arm

cmake的windows编译脚本参考 build/windows/win32

### 第三方库的优雅管理(ffmpeg)

基本方法： git + patch

## 持续集成预研(评估和结论)

预研人员：XXX

|  |  |  |
| --- | --- | --- |
| 预研项目 | 预研要求 | 评估结论 |
| 持续集成预研 | 单元测试选型与评估  内存泄漏工具选型与评估  集成测试选型与评估 | N/A  N/A  N/A |

图表-xx 持续集成预研评估和结论

## 组件管理预研(评估和结论)

预研人员：XXX

|  |  |  |
| --- | --- | --- |
| 预研项目 | 预研要求 | 评估结论 |
| 组件管理预研 | 轻量级组件架构选型与评估  组件间数据拷贝选型与评估 | N/A  N/A |

图表-xx 组件管理预研评估和结论

## 原语层预研(评估和结论)

预研人员：XXX

|  |  |  |
| --- | --- | --- |
| 预研项目 | 预研要求 | 评估结论 |
| 原语层预研 | 媒体原语层选型与评估  操作系统原语层选型与评估 | N/A  N/A |

图表-xx 原语层预研评估和结论