## 基本配置

### 开发环境配置

编译环境系统

ubuntu-20.04.2.0-desktop

硬件：

主控：Hi3536C;

FLASH: 16MB SPI Nor FLASH;

DDR: 512MB

BSP版本：

Hi3536C V100R001C02SPC040

### 日记管理

工程目录中的log目录。

先把日记打印功能添加上，方便调试。之前比较多的项目中使用了glog这个库，感觉也挺好用的，用的功能比较简单，这里就用这个库吧。

嵌入式系统中一般存储都很有限，我们先简单的记录到文件文件中先。

添加一个单独的目录管操作日记，使用一个单例的实例类来管理这个日记功能。

Glog里有一个比较好用的功能，即设置InstallFailureWriter,即在出错时，或程序崩掉时，会把一些简单的堆栈打印出来，不过很多时间也打印不出来，有时程序core dump时还是会有时给到一些信息，查找程序问题的。

### 代码中的断言

以前的一些开发习惯，我会比较多的在程序中逻辑、业务中不太可能出现、或出现即是错误的地方加断言，同时想着把断言打到日记里，所认加了一个ASSERT的宏定义，只是加了一个在出现问题时，把断言打到日记里。

见assertdef.h

### 三方库工具的编译和管理

三方库工具都放在工程third\_party目录里，本来想用buildroot来构建文件系统和三方工具这些库的，还是先用以前的习惯，简单的放在一个third\_party里，在里面建了一个tarball的目录，工程所有用到的三方库的源代码都放在这里，同时针对不同的芯片方案，建立一个目录，编写对应的编译脚本，以前这样做主要是想做到一键把所有的三方库给编译了，有对应的三方库的修改补丁也放在里面，以免后续的人来维护时，不了解是如何编译的和自己打过哪些补丁。

### **对像单例的使用**

对于很多类的对像，模块，在应用中都基本是单例的存在，比如日记管理、存储管理等，如果用Ｃ++，基本是抽像成一个类，创建一个对应的实例这样子用，我会比较习惯把这些对像建立成一个单例，这样用任务地方用，只要引用这个类的头文件，同时把类的构造、析构函数声明为private，调用类对应的单例获取函数即可！

如TLog里的getInstance方法。

## 软硬件板端开发环境配置

直接在硬件上跑编译程序时报C,C++库的版本不对的警告信息，看来是编译器的版本和当前板端系统上的不一致导致的，需要更新一下rootfs,使用编译器对应的BSP包：Hi3536C V100R001C02SPC040，编译一下rootfs，为了方便开发，使用jffs2的文件系统.

1. 分区配置:

mem=300M console=ttyAMA0,115200 root=/dev/mtdblock2 rootfstype=jffs2 rw mtdparts=hi\_sfc:1M(boot),4M(kernel),11M(rootfs);

1. BSP编译，解压BSP包后，在Hi3536C V100R001C02SPC040\01.software\board 找到Hi3536CV100\_SDK\_V2.0.4.0.tgz，解压出来，在目录下再执行：sdk.unpack。

参照osdrv\opensource\kernel的说明，把kernel下载下来解压，修改补丁打上，返回到osdrv目录下，安照readme\_cn.txt说明编译。

在我的ubuntu20提示mkimage找不到，这个用apt-get装上相应的工具就好了。

在编译到文件系统的mtd-utils1.5.0时会编译不过，记录几年前的处理方式我是更新到1.5.2的版本就好了。

错识主要是说找不到major,minor这几个函数，这次针对错误修改了一下，在对应的文件里加了 #include <sys/sysmacros.h>声明，还有提示zlib的头文件找不到，修改了一下对应的Makefile，因为只需要用到mkfs.jffs2，保证把这个工具编译出来后就没有往改了，编出mkfs.jffs2后，拷到osdrv/pub/bin/pc/mkfs.jffs2。

1. 参照SDK里的方式，打包一下rootfs:

osdrv/pub/bin/pc/mkfs.jffs2 -d osdrv/pub/rootfs\_glibc -l -e 0x10000 -o osdrv/pub/rootfs\_glibc\_64k.jffs2

1. 在ubuntu中装好tftpd-hpa的服务；
2. 进入板端的uboot命令中，配置好相关的ＩＰ地址；
3. 通过tftp,sf工具下发烧写好rootfs，如我的环境:

tftp 0x8200000 rootfs\_glibc\_64k.jffs2;

sf probe 0;sf erase 0x500000 0xb00000; sf write 0x82000000 0x500000 0xb00000;

1. 在进入系统设置好IP, mount到uboot的NFS目录，运行程序没有报错误了。
2. MPP环境配置；

把Hi3536CV100\_SDK\_V2.0.4.0\mpp\ko目录放到挂载目录，在板端执行,但报Unknown symbol devm\_pinctrl\_put这些模块错误，看来kernel最好也是要更新一下了。

先把板端的kernel备份出来，再更新kernel.

直接用BSP的里编译的Kernel能正常跑起来，网络也正常，看来比较幸运，厂家与官方的DEMO变化不大，再执行下面的脚本。

./load3536c -i -total 512 -osmem 300

MPP的相应ko都加载上了。

有了这些环境后，基本就可以进行在这个板上的全业务功能开发了，把SDK里的mpp的驱动、库拷到ubuntu的挂载目录下，通过命令加载上即可！

## Record主要业务对像

产品中，根据NVR录相机的配置中的习惯，有通道的概念，在配置录相时，把接入的相机定义为一个接入的通道，相机绑定到一个通道上，所以我们抽像一个通道的对像(TVideoChannel)，同时对于这个接入的设备有可能是一个摄像机，也有可能只是一路视频流，或一个回放视频的通道，那这时源就是一个录相文件。所以这里再抽像出一个TVideoSource的对像,使用多态实现视频源有:摄像机源、MP4源等。对于通道在系统里，如果配置好后，一般就会处于打开操作的状态，生命周期一般一直保持着，如录相操作，对于视频的回放预览在硬件还是软件层面都有一个解码器的概念，我们定义出一个播放器的对像概念，播放器则不见得是同通道绑定的或长时间开启，一般硬件的编解码的性能有限，这样播放器这个对像就要考虑多通道复用或动态的绑定，实时的创建，同时他存在的生命周期一般不像通道只要配置好一直运行着。播放器这个对像也是对软硬件解码器的隔离抽像。

根据上面的想法，需要下面的这些对像出现在系统中：

**视频通道-TVideoChannel**

主要是用来操作视频即叫视频通道吧，其实在产品中图片应该也是用这个解码显示了。

**视频源-TVideoSource**

源可以是一路视频，或一个摄像头设备、一个MP4文件等；

**播放器-TPlayView**

实现对通道视频对像的播放，实现解码、显示、控制功能，播放器这个对像也是对软硬件解码器的隔离抽像；

基于上面的这些对像，对于NVR来说，我们再实现一个统一的管理类:TRecordManage,由他来管理通道的生命周期，每个通道再聚合绑定自己的视频源，及绑定自己的播放器。

当然上面还很多小的对像对类被这几个大的对像聚合使用，如缓存管理、视频文件的解、封包、网络流的对像、预览时不同模式的展示，我们需要实现一个视频显示分割器管理TPlayView的显示等，后续再慢慢的分解抽像实像吧！

## 一些工具类对像

VThreadBase

这是一个线程抽像类，方便简单的线程使用。

TMutex

一个互斥对像；

TMutexLocker

互斥对像的一个方使使用的工具对像，使用过QT的都知道哈哈！

TServiceBase

一个服务类对像，即是一个线程从VThreadBase继承，为什么要做这个呢？主要是把系统中的各个功能线程看成是一个服务，在基类里统一注册到一个管理类里，这个服务类实现一些通用的约定的服务接口，这样在服务管理类里统一获取各个服务的状态及对其监控！

TSysServiceManage

服务管理类，所有继承TServiceBase的类都会注册到这个类对像，这样在使用WEB工具，PC工具，云服务时，我们通过这个类就能完成各个服务的状态获取和监控。

TAppConfig

系统配置管理类对像。

TLOG

日记管理类对像；