ANYKA
芯 随 你 动

版本: 1.0.1 2015 年 03 月

Sword 37C 用户开发手册

声　　明

本手册不对包括但不限于下列事项担保：适销性、特殊用途的适用性；实施该用途不会侵害第三方的知识产权等权利。

安凯技术公司不对由使用本手册或执行本手册内容而带来的任何损害负责。

本手册是按当前的状态提供参考，随附产品或本书内容如有更改，恕不另行通知。

联 系 方 式

安凯（广州）微电子技术有限公司

地址：广州科学城科学大道 182 号创新大厦 C1 区 3 楼
电话: (86)-20-3221 9000

传真: (86)-20-3221 9258

邮编: 510663

销售热线:

(86)-20-3221 9499

电子邮箱:

sales@anyka.com

主页:

http: //www. anyka.com

版 本 变 更 说 明

以下表格对于本文档的版本变更做一个简要的说明。版本变更仅限于技术内容的变更，不包括版式、格式、句法等的变更。

版本	说明	完成日期
V1.0.0	正式发布	2014 年 08 月
V1.0.1	1、手册结构调整及优化 （1）将状态机、AKOS 介绍合并到平台和系统整体架构，并增加状态机、设备驱动、文件系统的说明 （2）状态机机制合并到《Sword37 状态机应用接口说明》 （3）中间层接口说明独立介绍，参见《Sword37C 中间层接口说明》 2、增加系统配置说明	2015 年 03 月

目录

1 Sword37C平台简介

Sword37C 系统开发平台是基于安凯 AK37C 系列芯片的针对指纹考勤机、录像海螺机等产品的开发平台。该平台是一个基于 RTOS 的多任务操作系统，全面支持多线程及线程间通信和同步机制。

2 建立开发环境

Sword37 平台的开发环境必须包括：Cygwin 及 ADS 编译环境、Perl，以及其他一些辅助程序如串口调试工具、PC 模拟器、烧录工具等。

各个工具的关系以及相应文件如下图所示。

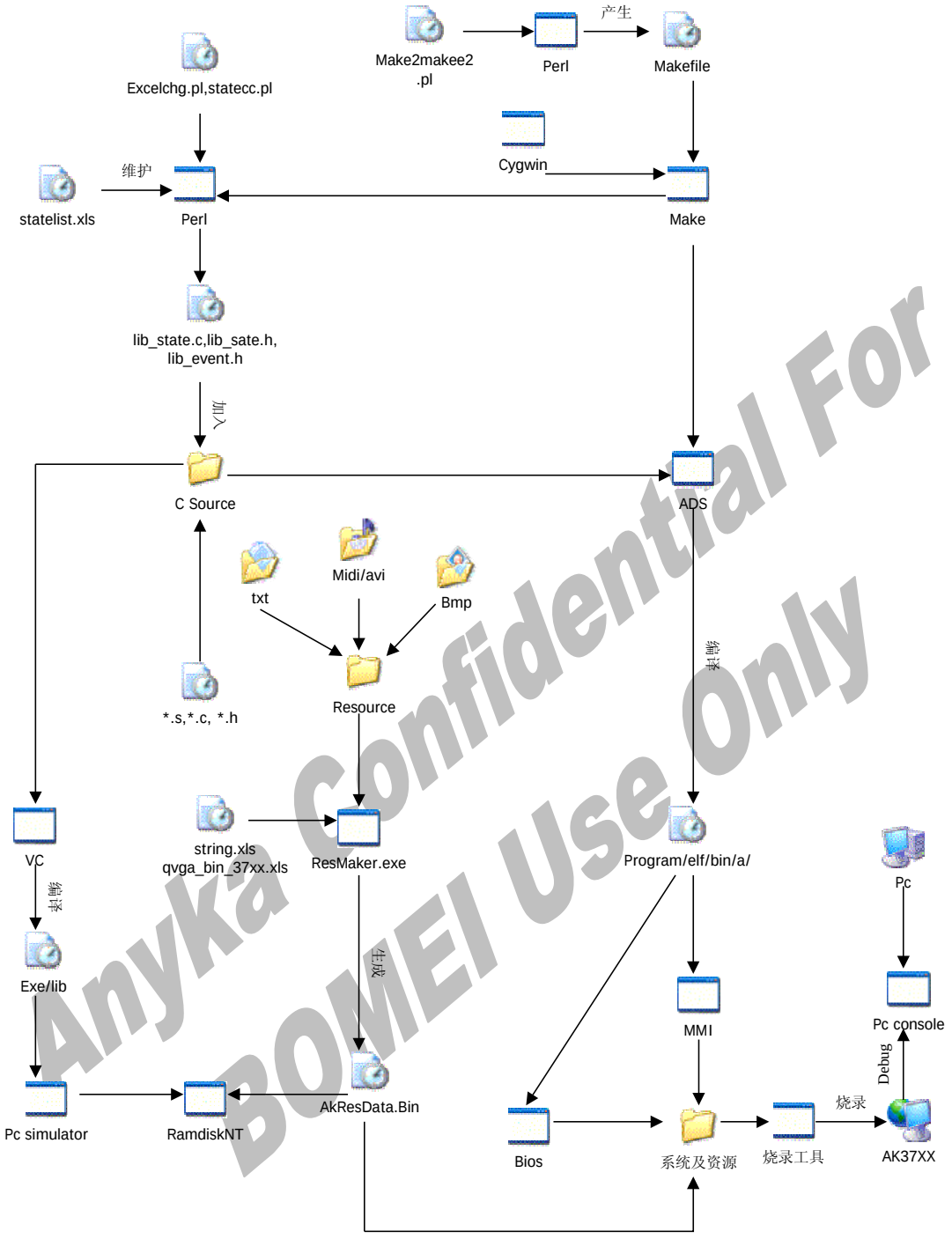

图 2-1 工具关系说明

2.1 开发板使用说明

有关平台开发板的详细使用说明和按键定义，详见《Sword37C 平台操作手册》文档。

2.2 开发环境软件安装和使用

Sword37C 平台开发所需要用到的软件包括 Cygwin（交叉编译环境）、ADS（ARM 处理器专用开发软件）以及 Perl（Perl 语言解析器）等。

关于 Cygwin、ADS、Perl 的获取，请用户购买正版软件或者自行解决。

注意：安凯微电子不对前述推荐使用的软件作出任何保证和承诺，不保证其适用性、可用性、合法性及其它任何权力，用户基于该软件选择和使用而产生的任何责任由用户承担，与安凯微电子无关。

有关相关软件的安装和使用说明，请参见《开发环境软件安装参考说明》文档。

3 编译及烧录说明

3.1 开发板版本编译说明

3.1.1 BIOS源码编译

当前版本只支持 SPI BOOT ，不需要编译 BIOS。

3.1.2 编译资源

系统中用到的图片、字符、文字等资源需要编译成相应二进制文件，同时修改相应的源文件。编译采用…\firmware\Resource 路径下的批处理命令 build_37xx_xx.cmd 完成。

各个批处理文件的具体用途，请参考…\firmware\Resource\Readme.txt。下面以 build_37xx_qvga.cmd 为例进行说明。

1、运行 build_37xx_qvga.cmd，调用如下图所示的资源生成程序。

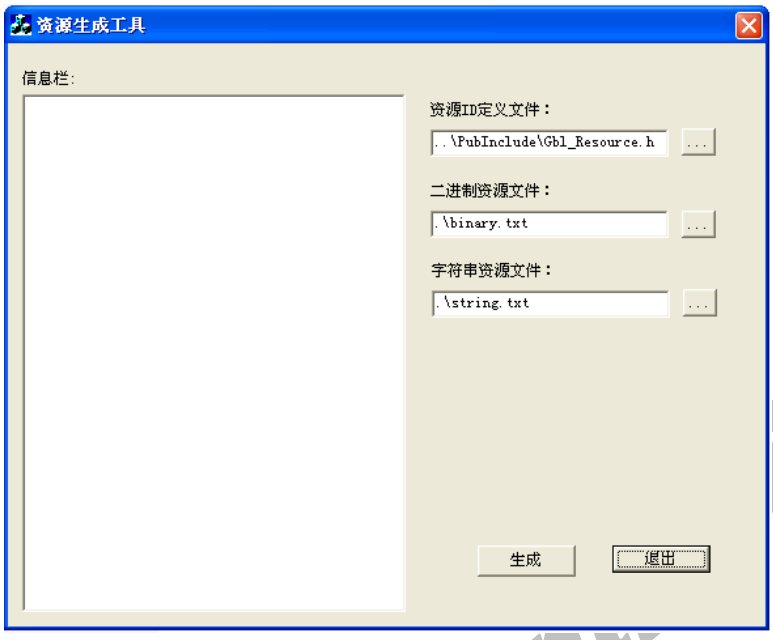

图 3-1 资源生产工具界面

2、 单击生成按钮，当出现如下图所示的提示时，表示资源生成成功。

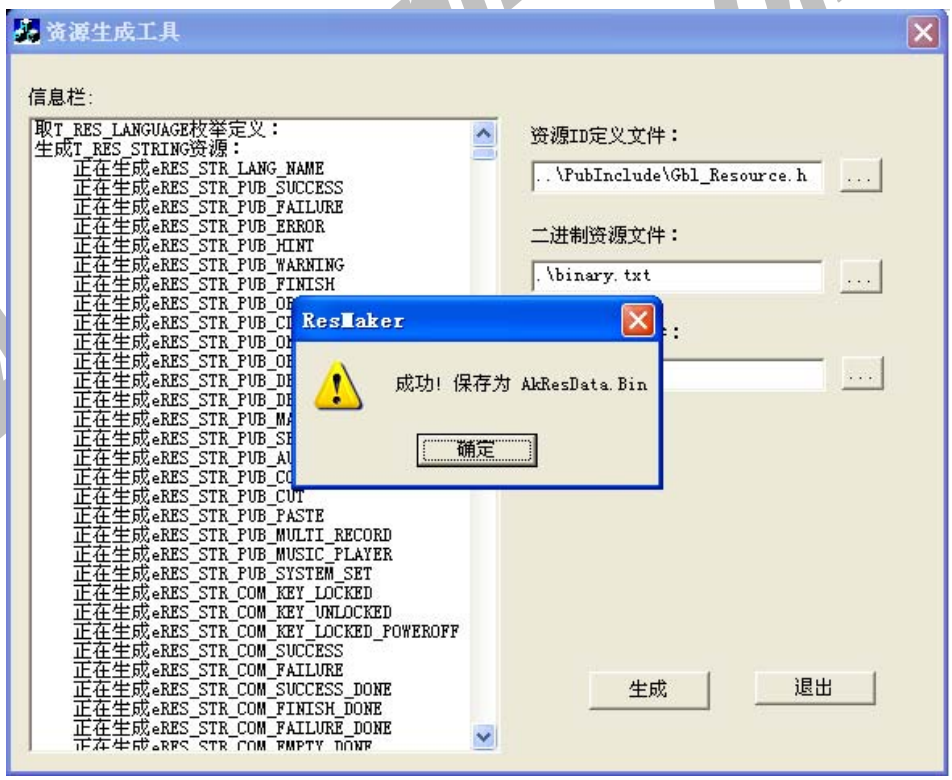

图 3-2 资源生产成功界面

3.1.3 编译源程序

3.1.3.1 编译文件用法

参见 目录 firmware\Build\BoardTarget 下的 readme.txt。

3.1.3.2 源程序编译方法

源程序编译按照以下步骤进行：

1、进入“firmware/Build/BoardTarget”目录，运行 perl 脚本 make2makee2.pl 产生 makefile 文件。

注意：当有编译源文件增加/删除时，需要重新运行 make_ads.pl

2、 选择合适的 command_xx.txt 文件，重命名为 command.txt，按照需要修改相应的配置项。详细信息请参考”编译文件用法”一节。运行批处理文件 Build.bat，编译通过后会在当前目录下生成 Sword37.bin。

注意：

1、 如果源码有所更新，包括.s、.h、.c，或者输入的参数有变，并且 makefile 文件并没有进行更新检查时，可能需要 make clean 后再次编译。

2、如果更新了资源 ID， firmware\PubInclude\Gbl_Resource.h 将发生变化，请重新编译；如果状态机发生变化，状态机文件 Lib_state.c、 Lib_state.h、Lib_event.h 发生变化，也需要重新编译。

3.1.4 程序烧录及运行

BOOT 启动方式的流程图如下图所示。

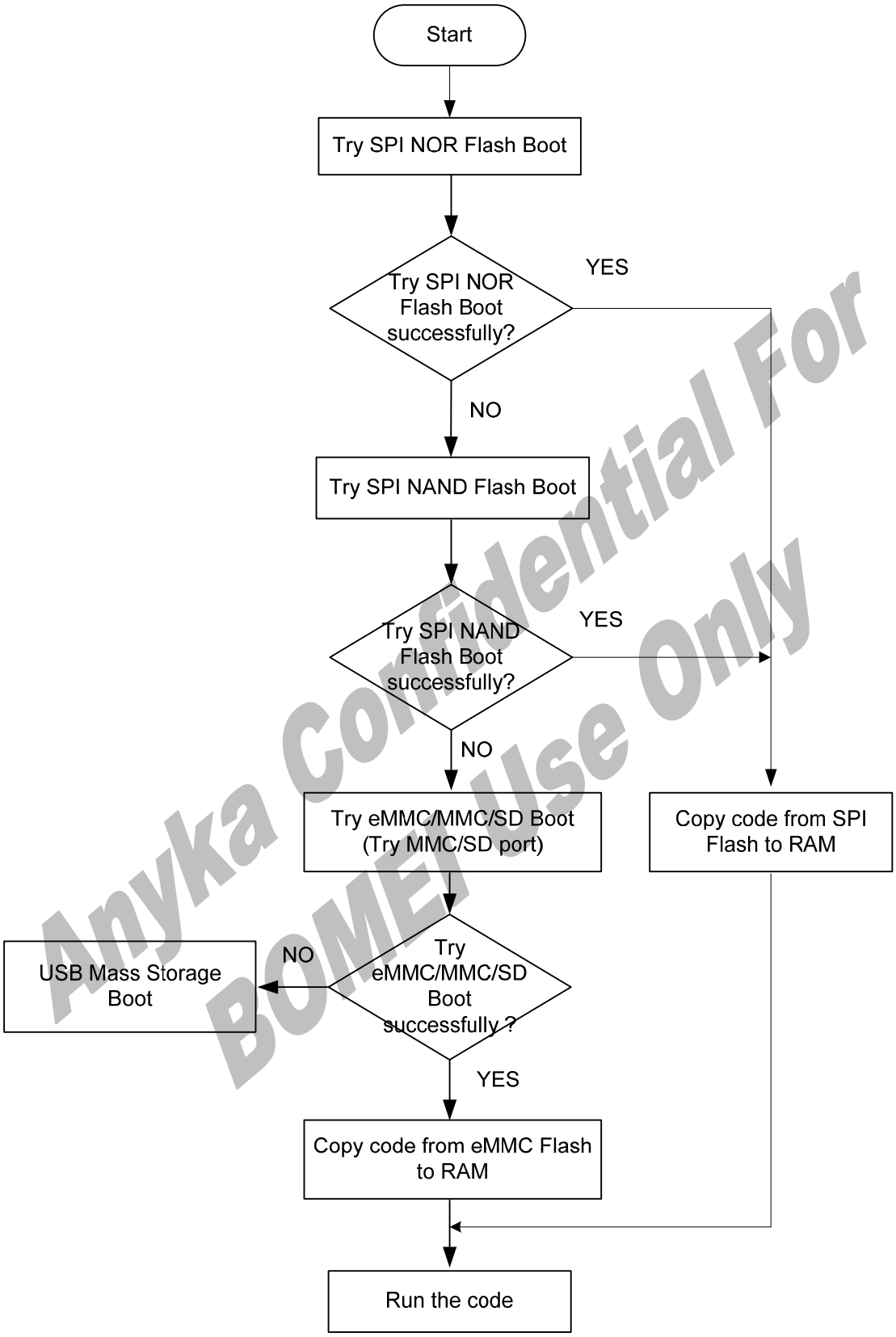

图 3-3 BOOT 流程

在裸板上烧录系统，需要准备以下文件：

-	工具/文件	工具名称
PC 端	烧录工具	BurnTool.exe
	烧录配置文件	config.txt
开发板端	烧录程序	producer_37xx.bin
	引导程序	xxbootxx.bin
	Bios	XXXX_bios_XXXX.bin (spiboot 不需要)
	MMI	Sword37.bin
	资源	AkResData.Bin
	字库	DynamicFont4_XX.bin
	语言	LangCodepageXX.bin

为了运行程序，需要使用路径“platform\burn_tool”下的烧录工具将程序下载到开发板上。有关烧录工具的详细使用方法，请参见《烧录工具使用说明》文档。

3.2 模拟器版本编译说明

本平台暂不支持模拟器。

4 系统调试

4.1.1 调试信息

在程序运行过程中的打印信息会自动保存到文件。为方便调试程序，文件保存在调试工具的当前路径下。

4.1.2 调试说明

DEBUG 调试将会打开看门狗功能和任务列表打印功能。

1、通过长按【UP】键会打印出当前系统所有已被创建的任务信息。比如 name:MMI, stat:4, prio:100, pree:10, sched:101620, slice:2, stack:30ef0780, size:196608, used: 27180。

其打印中各关键词所代表的意义如下下表所示。

关键词	意义
name	任务的名称
stat	任务当前的运行状态
prio	任务的优先级
pree	任务是否是抢占的
sched	任务被调度的次数
slice	任务执行的时间片
stack	任务堆栈基址
size	任务栈大小（bytes）
used	已用栈大小（bytes）

2、软件的看门狗（Watchdog）功能为在线程运行期间，如果两次喂狗操作间隔超过设置时间，则会打印出喂狗操作的所在线程的信息。(**注意**：超时后不代表任务崩溃，仅代表任务执行超时)

使用方式： AK_Feed_Watchdog(second)；// 其中参数 second 代表两次喂狗的最长时间间隔，单位为秒。

如果两次调用超过此时间，则会打印喂狗所在线程的信息；如果为 0，则关闭喂狗。

4.1.3 内存问题调试参考说明

本节主要介绍平台内存调试的一些参考说明，为客户提供解决问题提供参考方法和思路。

4.1.3.1 检查error.txt文件中各个段是否过大

```
================================================================================

      Code    RO Data    RW Data    ZI Data      Debug

   3273384     485037     479269     711509    8043908   Grand Totals

================================================================================

    Total RO  Size(Code + RO Data)                 3758421 (3670.33kB)
    Total RW  Size(RW Data + ZI Data)             1190778 (1162.87kB)
    Total ROM Size(Code + RO Data + RW Data)      4237690 (4138.37kB)

================================================================================
```

图 3-6 检查 error.txt 段大小

查看 Total ROM Size 是否是在你所编译版本（8M/16M/32M）SDRAM 大小之内。如果超过，表明 BIN 文件肯定有问题，就不要把 BIN 文件下载到硬件上。

一般解决思路：

1、当前编译的版本功能过多，不该加的都加在了编译参数中，特别是 8M 版本的编译。

2、查看平台代码某处是否定义了较大（几十 KB 甚至上 MB）的静态数组。如果是，建议修改为动态分配。

3、查看是否某个库的.a 文件使用了 DEBUG 版本而不是 RELEASE 版本，两者的大小有时候相差很大。

4、查看烧录工具的设置是否正确。

5、检查 raminit.c 文件中下面代码设置是否合理，即内存库实际管理的内存大小，其余的空间用于代码 BIN 和堆栈。

```
#ifdef OS_ANYKA
extern T_U32 Image$$ER_ZI$$ZI$$Limit;
#define MALLOC_MEM_ENTRY ((T_U32)&Image$$ER_ZI$$ZI$$Limit + 4)
#define MALLOC_MEM_SIZE (SVC_MODE_STACK - 8*1024 -
(T_U32)MALLOC_MEM_ENTRY)
#else
#define MAX_RAMBUFFER_SIZE   ((SDRAM_MODE << 20))
static T_U8 gb_RAMBuffer[MAX_RAMBUFFER_SIZE];
#define MALLOC_MEM_ENTRY  gb_RAMBuffer
#define MALLOC_MEM_SIZE   MAX_RAMBUFFER_SIZE
#endif
```

4.1.3.2 调试中请打开内存调试宏

为加快编译速度，内存调试宏定义在 Fwl_osMalloc.c 文件中。目前平台上内存调试宏都处于打开状态（代码如下所示）。这些宏打开后会打印出一些内存信息。

```
#ifndef ENABLE_MEMORY_DEBUG
  #include "eng_debug.h"
  #define ENABLE_MEMORY_DEBUG    //内存越界自动监测器
```

```
#ifndef OS_WIN32
    #define DEBUG_TRACE_MEMORY_LEAK
#endif
    #define USE_MEMORY_MULTITHREAD
#endif
```

4.1.3.3 单独检查一行或多行代码是否有内存泄漏

1、内存泄露检查

需要检查的代码位于以下两个函数之间：

```
Fwl_RamLeakMonitorPointBeg(T_VOID);
    ……
    要检查的代码行
    ……
Fwl_RamLeakMonitorPointEnd(T_VOID);
```

这两个函数一定要成对出现。如果代码存在问题，函数内部会打印相应的信息。注意，这对函数不能检测编译阶段的变量或数组的越界问题，只能找运行阶段的。

2、实时查看内存信息

用户可以在需要的地方加上 Fwl_RamBeyondMonitorGetbyTimer(T_U32 LLD);（参数 LLD 可以赋任意 T_U32 数值）实时查看内存信息。如果代码有问题，函数内部会打印相应的信息。

4.1.3.4 Trace中打印信息含义说明

1、样例 1：

********** Memor Exception **********

LID=0 ptr=0x305d4730 size=144 filename=– , fileline=0, type=272

这种情况，确定存在内存问题（红色数字）。

type=

BEYOND_OK = 0x000, //ptr 无越界

BEYOND_PREV = 0x101, //ptr 本身发生前向越界 257 = 0x101

```
BEYOND_BACK      = 0x110,      //ptr 本身发生后向越界 272 = 0x110
BEYOND_BOTH      = 0x111,      //ptr 本身发生全越界   273 = 0x111
BEYOND_INVADE    = 0x202,      //ptr 越界到后驱节点
```

2、样例 2：

```
exitinit_at_before_pin
[[[****** Auto Memory Leak Active In Stack = 19 Begin ******
******************** Memor Leak ********************
FileName:Null Filename, maybe not input!,        Line:0,  Addr:30712e90, Size:112
****** Auto Memory Leak Active In Stack = 19 End ******]]]
enter @@@initinit_inquire_pin_state
```

从一个状态机进入另一个状态机（没有文件名打印出来，一定是某个库里面的打印出来），存在申请的内存没有释放，但不一定是泄漏。因为某些资源可能是不释放的，所以这时就需要用户根据实际情况来判断（根据打印信息和运行的代码），是否真的存在某个地方泄漏内存。

3、样例 3：

```
[[[****** Auto Memory Leak Active In Stack = 19 Begin ******
******************** Memor Leak ********************
FileName:fs/string.c,        Line:118,        Addr:3073ba20,        Size:32
******************** Memor Leak ********************
FileName:fs/string.c,        Line:127,        Addr:3073ba70,        Size:32
****** Auto Memory Leak Active In Stack = 19 End ******]]]
```

同样例 2，可以看出这个是文件系统里相应的地方有内存分配。文件系统是不释放内存的，因此这个情况是正常的。

4.1.3.5 内存泄漏一般调试方法

一般由于某个变量没有初始化或者没有 malloc()空间就在使用或者 malloc()空间少了 1 个或多个字节而造成死机。

下面介绍调试方法，仅供参考。

1、根据现象粗略定位。如果能在 WIN32 上单步调试，那就设置断点进行调试，一般能很快解决。

2、如果不能在 WIN32 上调试，则需增加打印信息判断分析。有时候打印信息可能还没有来得及打印出来，就已经死机或重新开机，此时可以用 while(1);语句让打印全部输出，一步一步移动此语句，查看代码的实际流程。例如：

```
……
Fwl_Printf("\naaaaaaaaaaaa1************************\n");
Add1 ();
Fwl_Printf("\naaaaaaaaaaaa2************************\n");
Add2 ();
Fwl_Printf("\naaaaaaaaaaaa3************************\n");
Add3 ();
Fwl_Printf("\naaaaaaaaaaaa4************************\n");
Add 4();
Fwl_Printf("\naaaaaaaaaaaa5************************\n");
```

可能 Trace 里面只打出

```
aaaaaaaaaaa1************************
aaaaaaaaaaa2************************
```

然后重新开机了，此时如果在 Add2 ()中看不出来任何问题，建议用户加上 while(1);语句再看 Trace 信息。

```
……
Fwl_Printf("\naaaaaaaaaaaa1************************\n");
Add1 ();
Fwl_Printf("\naaaaaaaaaaaa2************************\n");
Add2 ();
Fwl_Printf("\naaaaaaaaaaaa3************************\n");
Add3 ();
while(1);
Fwl_Printf("\naaaaaaaaaaaa4************************\n");
Add 4();
Fwl_Printf("\naaaaaaaaaaaa5************************\n");
```

可能 Trace 里面打出

aaaaaaaaaaa1**********

aaaaaaaaaaa2**********

aaaaaaaaaaa3**********

然后重新开机了，此时说明 Add2 ()中没有问题，Add3 ()应该成为重点侦察对象。

3、粗略判断一个状态机是否存在内存泄漏可以使用以下方法：

Fwl_GetTotalRamSize()函数可以得到当前所有内存的大小

Fwl_GetUsedRamSize()函数可以得到当前已用内存的大小

在该状态机 init 函数第一句之前加上

Fwl_Printf("\n***name*** %d\n", (Fwl_GetTotalRamSize()-Fwl_GetUsedRamSize()));

在该状态机 exit 函数最后一句之后加上

Fwl_Printf("\n***name*** %d\n", (Fwl_GetTotalRamSize()-Fwl_GetUsedRamSize()));

重新编译下载运行中如果这两句话打印出来的数字不一样，并且每进出一次这个状态机，其数字都在减少，此时可以肯定，该状态机某处有内存泄漏。

4、有时候出现 PC=0x30XXXXXX!!这种死机信息，接着后面打印很多类似信息，这表示代码在某处死掉了。此时用户可取第一个这种地址进行反编译，到 Sword37.txt 文件里搜索，便可找到是在执行哪个函数。例如串口打印如下：

```
#################################################################
     Error type:  abort error
     abort address: 0x70e77a1e
     Stack pointer : 0x3050930c
     PC    value   is : 0x30094c64
     SPSR  value   is : 0x80000013
     R[0]  value   is : 0xab1e
     R[1]  value   is : 0x390a
     R[2]  value   is : 0x390a
     R[3]  value   is : 0xca
     R[4]  value   is : 0x305096f0
     R[5]  value   is : 0x140
     R[6]  value   is : 0xf0
     R[7]  value   is : 0x30508090
     R[8]  value   is : 0x30508030
     R[9]  value   is : 0x23
     R[10]  value   is : 0xb0
     R[11]  value   is : 0x2d
     R[12]  value   is : 0xab1e
     the values in the stack is:
     0x0     0x0     0x0     0x0     0x0     0x0     0x0     0x0     0x0
#################################################################
```

其中 PC value is : 0x30094c64，到 Sword37.txt 文件里搜索 0x30094c64，如下图所示。

```
    0x30094a44:    332e382e    .8.3    DCD     858667054
    0x30094a48:    00000000    ....    DCD     0
GE_BrowseEffect
$a
.text
    0x30094a4c:    e92d4fff    .O-.    STMFD   r13!, {r0-r11, r14}
    0x30094a50:    e24ddf67    g.M.    SUB     r13, r13, #0x19c
    0x30094a54:    e1a08000    ....    MOV     r8, r0
    0x30094a58:    e1a07001    .p..    MOV     r7, r1
    0x30094a5c:    e1a04002    .@..    MOV     r4, r2
    0x30094a60:    e3a02000    . ..    MOV     r2, #0
    0x30094a64:    e3a01000    ....    MOV     r1, #0
    0x30094a68:    e59f0e74    t...    LDR     r0, 0x300958e4
    0x30094a6c:    e58d109c    ....    STR     r1, [r13, #0x9c]
    0x30094a70:    e58d2098    . ..    STR     r2, [r13, #0x98]
    0x30094a74:    e59db1d0    ....    LDR     r11, [r13, #0x1d0]
```

.....中间代码略.....

```
    0x30094c48:    e59d2098    . ..    LDR     r2, [r13, #0x98]
    0x30094c4c:    ea000025    %...    B       0x30094ce8
    0x30094c50:    e5970014    ....    LDR     r0, [r7, #0x14]
    0x30094c54:    e081c081    ....    ADD     r12, r1, r1, LSL #1
    0x30094c58:    e7d0a00c    ....    LDRB    r10, [r0, r12]
    0x30094c5c:    e598e014    ....    LDR     r14, [r8, #0x14]
    0x30094c60:    e0820082    ....    ADD     r0, r2, r2, LSL #1
    0x30094c64:    e7dee000    ....    LDRB    r14, [r14, r0]
    0x30094c68:    e2822001    . ..    ADD     r2, r2, #1
    0x30094c6c:    e2811001    ....    ADD     r1, r1, #1
    0x30094c70:    e04aa00e    ..J.    SUB     r10, r10, r14
    0x30094c74:    e00a0a99    ....    MUL     r10, r9, r10
```

由图可知是在 GE_BrowseEffect 函数中执行，则查 GE_BrowseEffect 函数的代码或者给该函数传入的参数是否有问题。

5 平台和系统整体架构

5.1 平台整体架构

Sword37 整体架构如下图所示，其主要分为 4 层：应用层（Applications Layer）、中间层（Middle Layer）、核心层（AKOS Layer）和驱动层（Drivers Layer）。

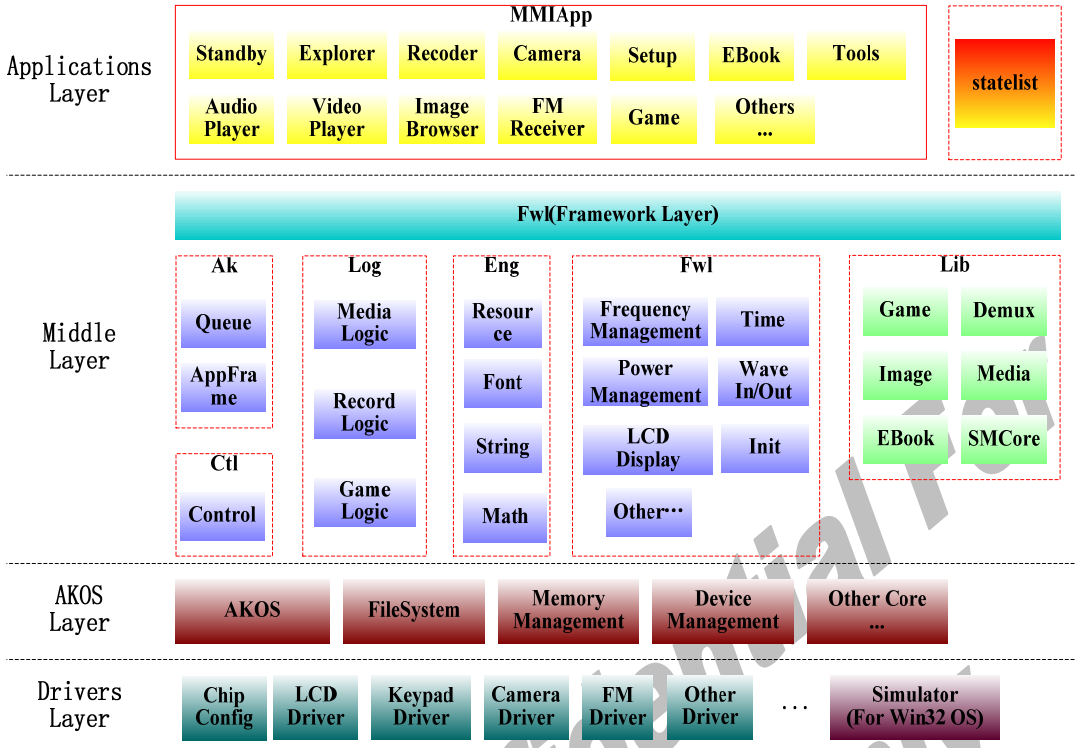

图 5-1 平台整体架构图

各层说明如下表所示。

名称	说明
应用层 （Applications Layer）	上层应用相关模块主要包括应用程序模块和相关的 GUI 程序，如主菜单和菜单控件（主要处理菜单显示：初始化、创建新菜单、删除菜单等，按键事件响应等），音频播放，视频播放，图片浏览，设置，摄像头，图像，工具包等都属于应用层。状态机模块（状态机内核在中间层）也独立的放在应用层。只有应用层才涉及修改或添加状态机，中间层只实现状态机逻辑。
中间层 (Middle Layer)	封装上层应用需要用到的通用模块、中间逻辑、函数库，比如，应用程序框架、资源管理、字库管理、电源管理、数学函数封装、字符串函数封装、多媒体播放逻辑等。其中 Fwl（Framework Layer）层作为该层的简单的接口适配层，负责中间层向上的接口差异化封装。中间层的淡绿色模块表示以库文件的方式提供，淡蓝色模块表示以源文件方式提供。
核心层 (AKOS Layer)	内核层。核心层包含了基于 nucleus 的操作系统核，文件系统，内存管理库和驱动核心逻辑的驱动库。

名称	说明
驱动层（Drivers Layer）	该层包含各个具体设备的驱动代码以及 Win32 操作系统下各种驱动的仿真程序。

5.1.1 中间层

中间层封装的接口模块主要包括以下部分：文件系统，资源管理，字库，频率管理，内存管理，控件模块，媒体库，图片解码库，拍照录像，App Frame 模块等。

5.1.1.1 中间适配层

中间适配层的实现依赖驱动库、文件系统库、多媒体库等底层库。通常应用应该调用中间层的函数而不直接调用底层库，这样可保证良好的移植性。有关中间适配层各模块接口函数请参见《Sword37C 中间层接口说明》。

5.1.1.2 中间层功能模块

有关资源管理库、字库的详细接口说明，详见《Sword37 资源管理库接口说明》、《Sword37 字库接口说明》。

多媒体库的接口说明，详见《媒体播放库接口说明》、《媒体解析库接口说明》、《媒体录制库接口说明》、《图像库接口说明》、《音视频库总体使用说明》 、《视频驱动库接口说明》 、《音频库接口说明》和《运动检测库接口说明》。

5.2 软件代码结构

5.2.1 整体结构

为了使代码结构清晰，方便以后的维护，代码结构设计如下：

目录	说明
lib_source	存放各种库源码，目前只有驱动库
platform	平台主目录，应用源码、库执行码、烧录工具程序和相关烧录执行码都存放与此
product_source	其他可执行程序源码目录，烧录工具源码、烧录程序源码 、boot 程序源码存放于此

5.2.2 库源码目录

目录	说明
DrvLib	驱动库源码

5.2.3 平台主目录

在根目录 platform 下，划分为三个文件夹，分别为 akbios、burn_tool 和 firmware。下面分别就三个文件夹的作用和结构进行说明。

目录		说明
Akbios		bios 源码(spiboot 不需要 bios)
Burn_tool		用于存放烧录工具
Firmware	AKOS	AKOS 用于存放 mmi 内核层相关代码， 驱动库、文件系统库都存放与此
	Applications	Applications 用于存放 mmi 应用层相关代码和状态机列表等
	Build	Build 文件夹用于存放 makefile 文件，编译代码使用的批处理文件，编译代码后产生的 Sword37.bin 文件以及用于编译模拟器的 winvme.dsp 文件等
	Drivers	Drivers 文件夹用于存放设备的驱动程序
	Middle	Middle 用于存放 mmi 中间层相关代码
	PubInclude	PubInclude 用于存放 mmi 一些共用的头文件
	Resource	Resource 文件夹用于存放 DynamicFont4_16，DynamicFont4_12，LangCodepage 等 bin 文件，资源生成工具 ResMaker 以及运行该工具生成的 AkResData 资源文件

5.2.4 其他可执行程序源码目录

目录	说明
BurnTool	烧录工具源码
Producer	烧录程序源码 , 需要驱动库。
spiboot	Spiboot 程序源码

5.3 AKOS架构及接口说明

5.3.1 AKOS多任务软件架构

AKOS 多任务整体软件架构如下图所示。AKOS 整体软件架构主要包括 Thread 线程模块、App 应用程序模块、AppMgr 任务管理模块以及 AKMsgDispatch 消息管理模块和子线程（SubThread）管理模块等。

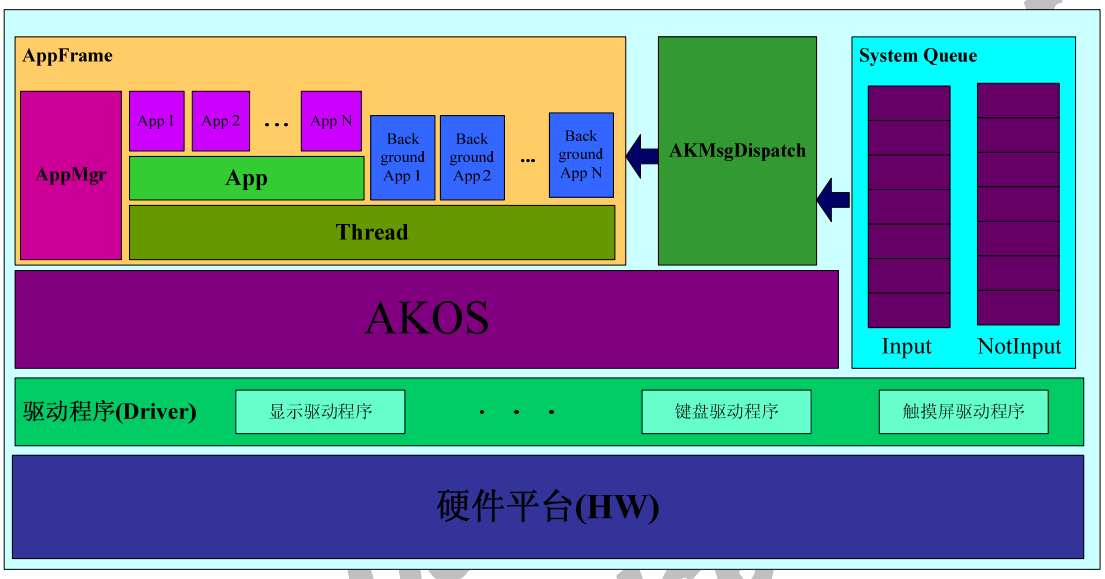

图 5-2 AKOS 多任务软件架构

5.3.2 AKOS中间层子模块说明

有关 AKOS 的接口说明以及各个中间层子模块的接口详细说明，请参见《Sword37AKOS 接口说明》文档和《Sword37AKOS 子模块接口说明》文档。

5.4 状态机

状态机是用于定义系统所有可能的状态及负责状态维护、状态间跳转的调度等核心程序，通过状态切换和事件跳转来显示不同的界面。

状态机设计的思想是将应用功能分割成小状态（状态机）。每一个小状态尽可能简单和独立。由于状态机是以状态栈的形式进行管理的，按照先入后出的原则进行状态机跳转。在任何情况下，状态栈里面至少有一个状态机（standby 状态机），栈顶状态机（当前

状态）只有一个，只有栈顶状态机能收到事件队列的事件。所谓状态跳转是指状态栈最顶上的状态机变化了。

关于状态机的机制和接口请参见《Sword37 状态机应用接口说明》。

5.5 设备驱动

驱动中间层提供了显示、按键、声音播放、录音等接口， 供应用使用，而接口的实现依赖驱动库.。对于 camera, lcd 等可选设备的驱动，需要按照驱动库框架，实现必须的函数。这些设备的代码文件，可以配置在平台上。

驱动库的接口函数说明，详见《Sword37C 驱动库接口说明》。

5.6 文件系统

文件系统 中间层提供了与文件系统无关的文件操作接口。目前文件接口的实现采用 FAT 文件系统。

FAT 文件系统库的接口请参见《EXFAT 库接口说明》,

存储设备非文件区域可以通过 FHA 和 FSA 库访问，FHA 和 FSA 库的接口请参见《FHA 和 FSA 库接口说明》。

6 音视频播放

6.1 功能概述

音频功能：支持 PCM、ADPCM、MP3、WMA 等格式解码，支持最高采样率可达 48KHZ，搭载 ID3管理器，快速加载，同时支持音乐播放特效。

视频功能：支持 MPEG4/ MJPEG/ H263 视频解码格式。

6.2 整体模块概述

6.2.1 模块层次结构

如下图所示，整个媒体播放逻辑分为文件读取、音视频数据分离(DMX)、音频解码和视频解码。 在视频解码时视频线程会释放对 CPU 的占用(视频硬解码)，DMX、音频解码

与视频解码各自独占一个线程(在视频播放时，对不支持 YUV 刷新 LCD 的 37XX 系列芯片，视频线线程还会创建一个 MPU 的子线程，异步执行将解码出来的 YUV 转换成 RGB 并刷新 MPU 屏上)。

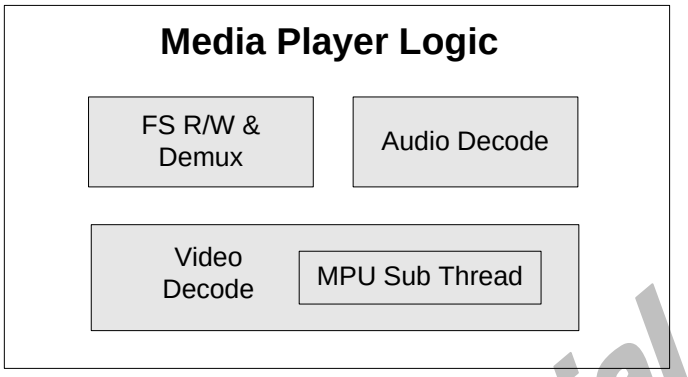

图 6-1 媒体播放逻辑

6.2.2 数据流说明

图 6-2 所示为整个播放逻辑的数据流程图。具体实现过程如下：

1、当 UI 线程打开文件，分析媒体头后确认解码器能支持，则跳入 2，否则返回错误；

2、UI 线程发送“开始播放”消息开始播放给 MEDIA 线程；

3、MEDIA 线程接收到“开始播放”后，调用 DEMUX 分别对 AUDIO BUFFER, VIDEO BUFFER 填写相应的数据，并开始音/视频解码定时器驱动 AUDIO/VIDEO THREAD 解码；

4、AUDIO THREAD 开始音频解码，将解码数据送至 DA BUFFER，开始音频播放；

5、VIDEO THREAD 开始视频解码，将解码数据送至 MPU Sub THREAD 队列，并驱动 MPU Sub THREAD 工作；

6、MPU Sub THREAD 取队列的 YUV 数据，转换成 RGB 并刷到 LCD 上。

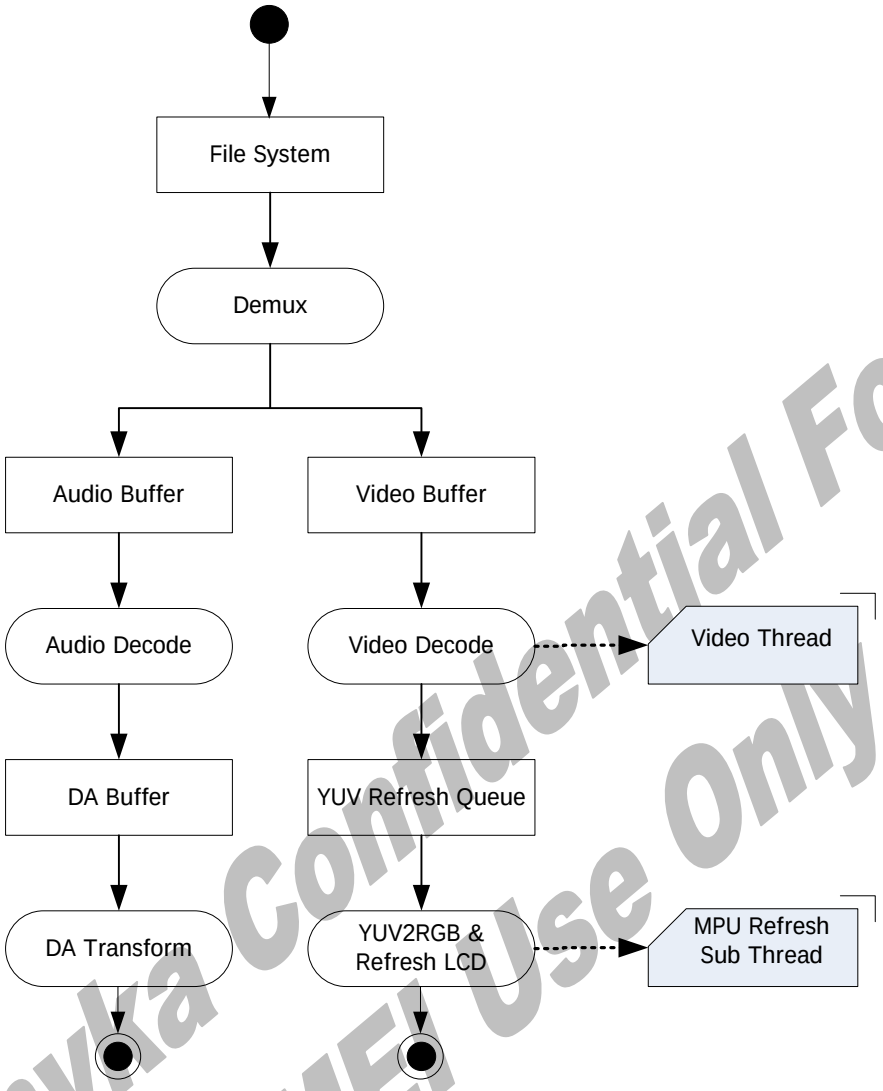

图 6-2 播放逻辑数据流程图

6.2.3 线程工作说明

音频解码线程、视频解码线程和 MPU 子线程由媒体播放状态机（音频/视频）管理。由状态机创建线程，申请资源，释放资源，响应用户事件等。主要的流程如下图所示。

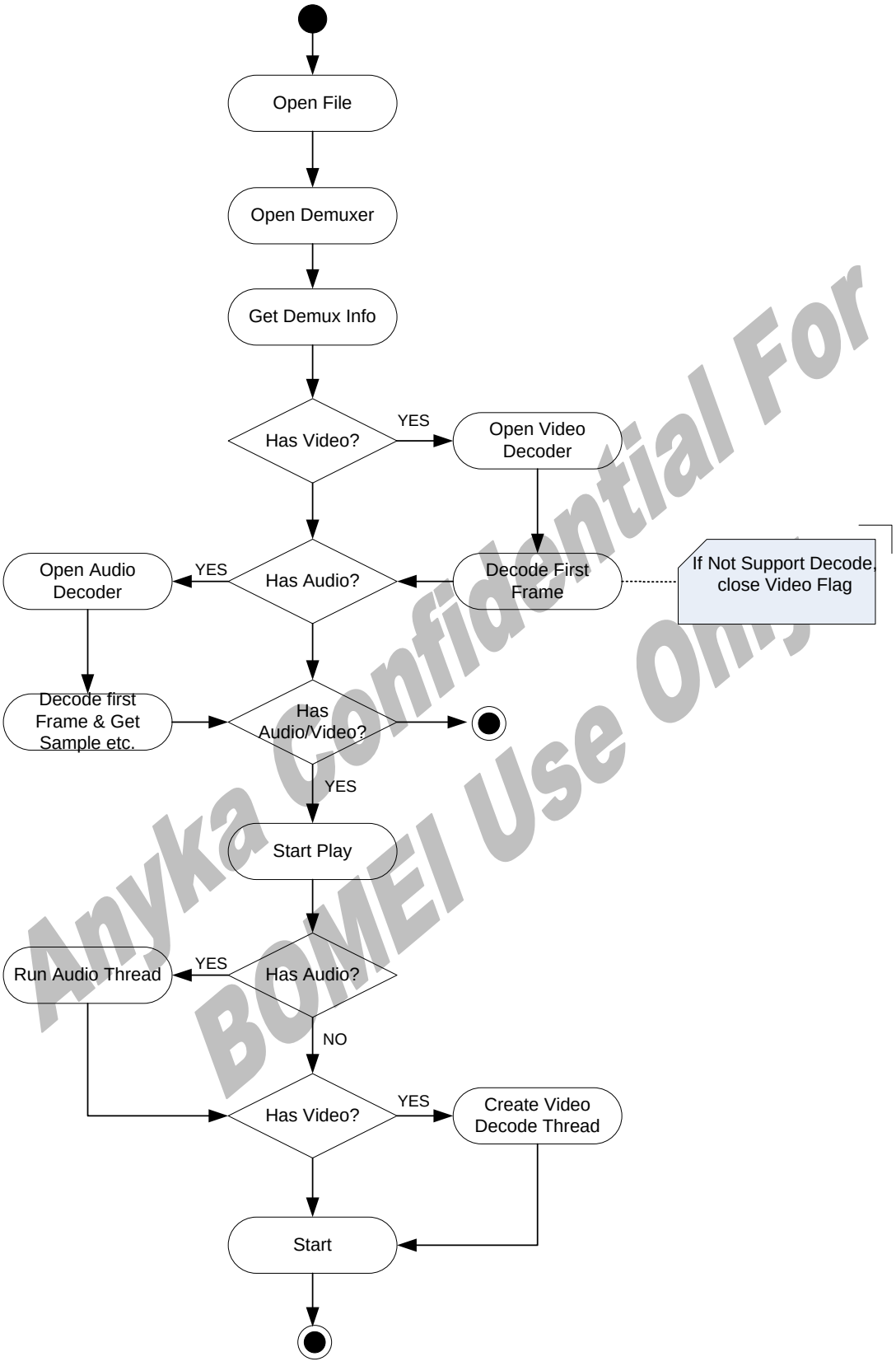

图 6-3 线程管理流程

6.2.3.1 MEDIA Thread (FS Read & Demux, Audio Decode)

Media Thread 在系统初始化时创建，长驻后台， 此线程在媒体播放逻辑中的任务是将指定的数据从磁盘上读到系统缓冲区里，分离音视频数据并写到指定的地址上，如下图所示。

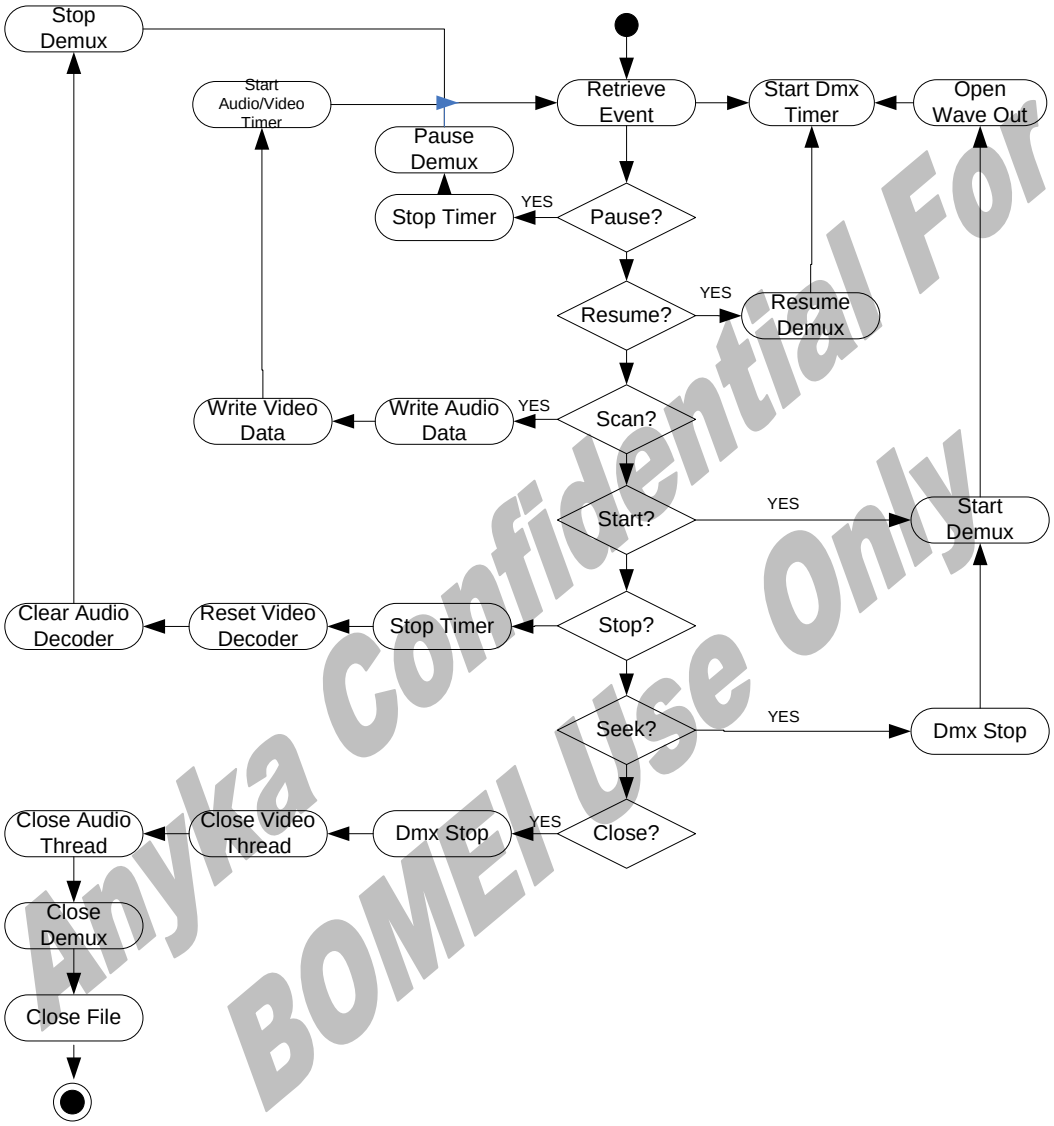

图 6-4 Media Thread

6.2.3.2 Audio Decode

Audio Decode 模块是从音频数据缓冲区内取到数据并将其解码，得到波形数据，查询 L2 Buffer 是否为满；若否，则将数据写入 L2 Buffer，主要过程如下图所示。

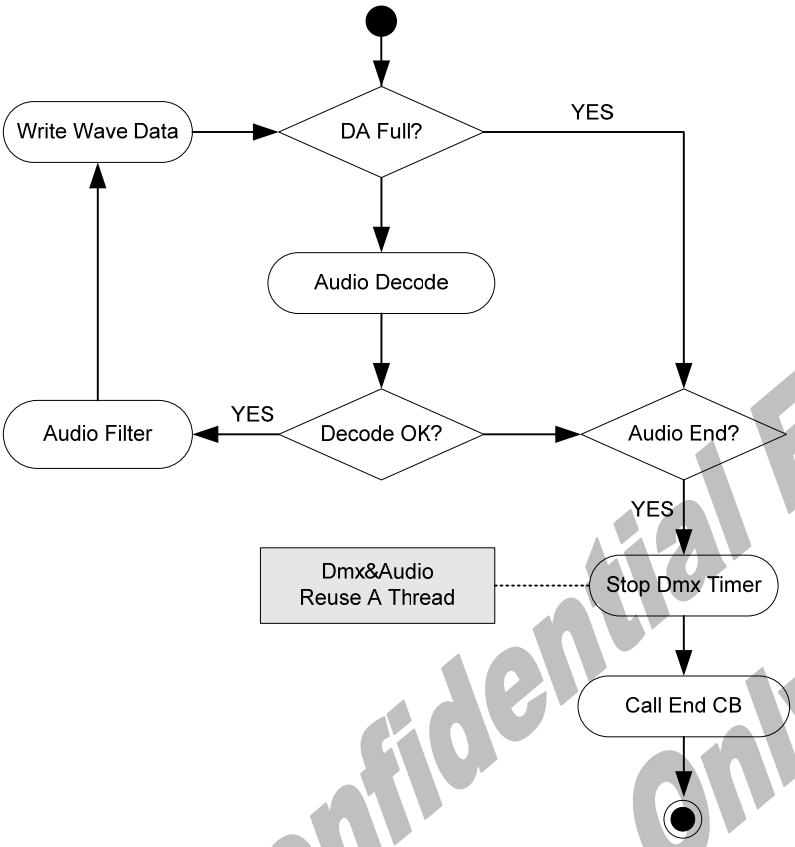

图 6-5 Audio Decode 流程

6.2.3.3 Video Decode

此线程式的任务是从视频缓冲取得视频数据，将数据送到硬件解码器解码，在硬件解码时释放对 CPU 的占用，这时 CPU 可以为其它的线程服务。当视频数据解码完毕，送给显示模块刷新屏幕。主要过程如下图所示。

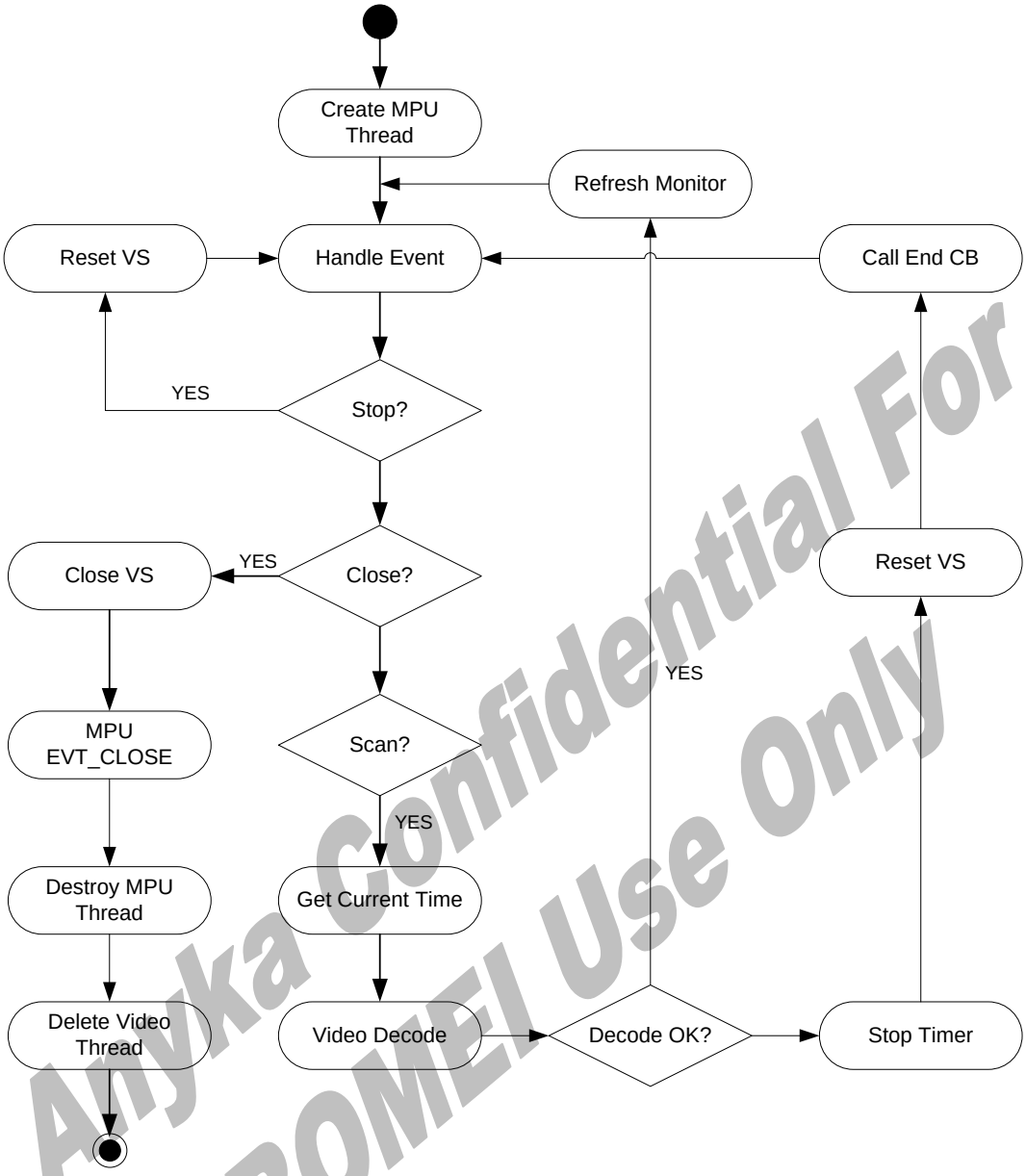

图 6-6 Video Decode 流程

6.3 工作原理

6.3.1 运行模块组合

音频和视频解码模块对 DEMUX 分离出来的数据访问采用轮询的方式。在运行时的媒体播放逻辑处于 “打开”、“播放”和 “暂停”三种状态之一。当调用 MPlayer_Open()后处于播放状态，调用 MPlayer_Play()则处于播放状态。具体的状态转换条件如下图所示。

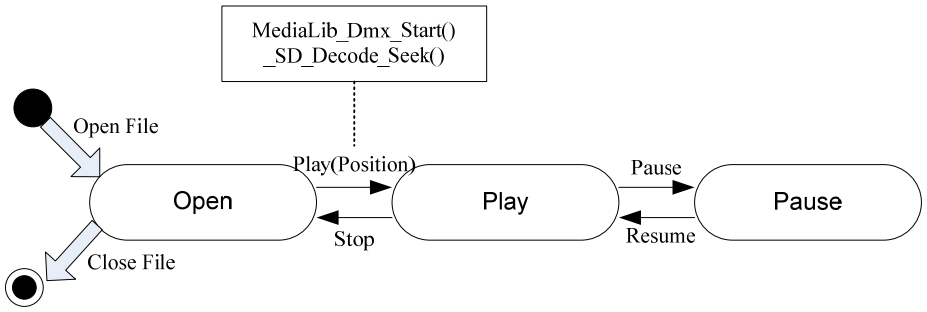

图 6-7 音频和视频解码模块状态转换

6.3.2 运行控制

媒体播放过程中四个线程的相互协作时序如下图所示。

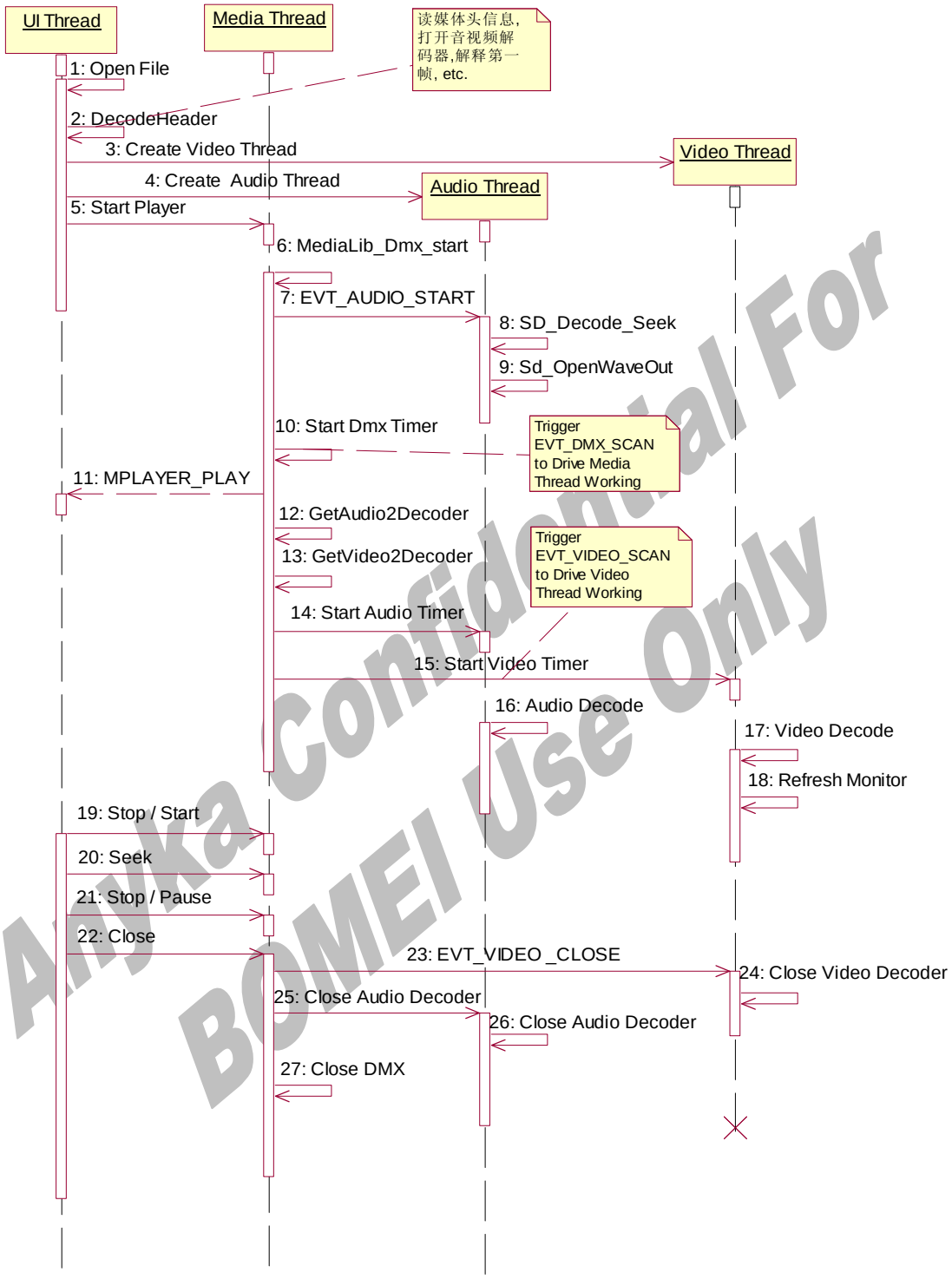

图 6-8 媒体播放过程线程协作图

7 录像

7.1 功能概述

- 支持 QVGA/VGA 录像
- 支持无缝循环录像
- 支持移动侦测
- 单文件最大录像长度为 4GB，大小支持【320×240】、【640×480】

7.2 整体模块概述

整个录像逻辑主要分为四个部分： Zoom(Stream 缩放)、MediaEncode(媒体编码)、VideoDisplay(视频显示)和 ExecptionManage(异常管理)。录像逻辑依赖的外部模块有：CameraStreamManage、PCM Manage、LcdRefreshManage、ImageEncode 和 FileSystem。各模块之间的关系以及其与外部模块的关系如下图所示。

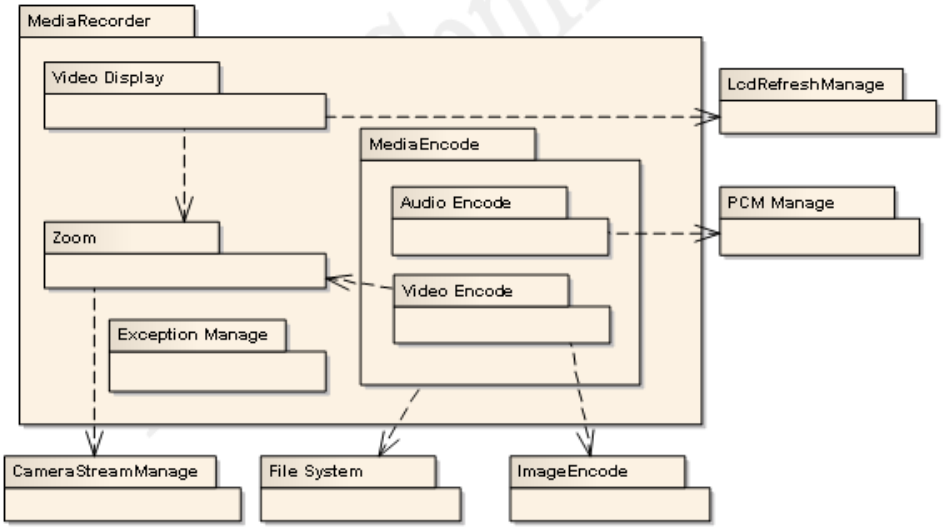

图 7-1 录像逻辑关系图

7.3 运行模块组合

录像运行模块的组合关系如下图所示，其中“Stream 管理”负责给“编码管理”和“视频显示管理”提供输入数据服务，从而驱动编码模块和显示模块的运行。“音频编

码”、“文件系统”、“异常检测”模块提供服务给“编码管理”，用来实现整个录像的编码、写文件、异常控制。

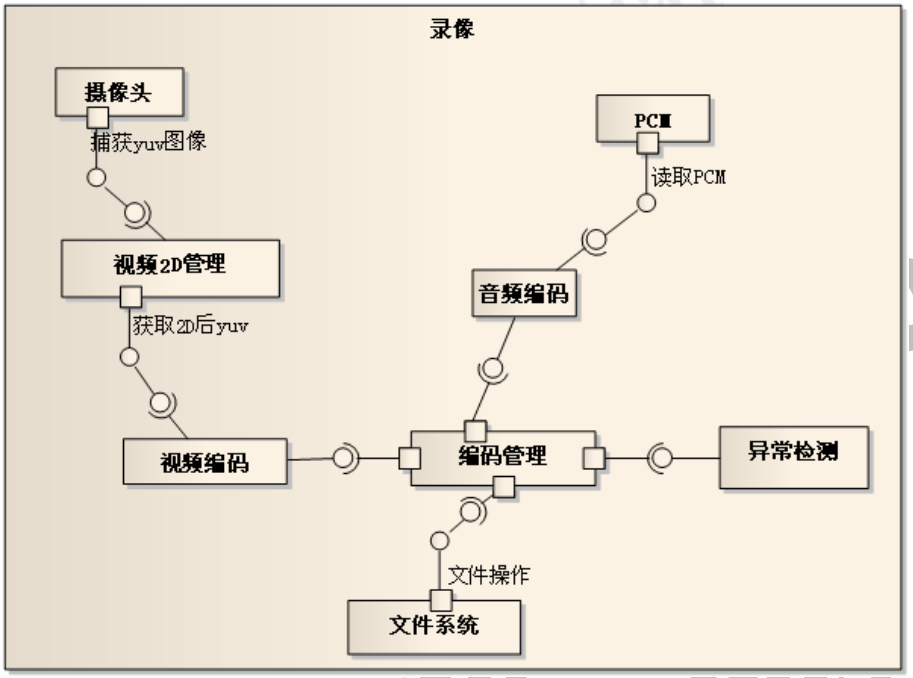

图 7-2 录像运行模块的组合关系

7.4 录像参数配置说明

录像参数的配置分可为三个部分，通常在 Ctl_VideoRecord.c 的 VideoRecord_StartRecord 中进行配置。

7.4.1 录像运行控制参数

用于配置录像基本参数，其参数保存在 一个类型为 T_REC_CTRL_INIT_PARAM 的结构体中。

录像运行控制参数说明(此处以一个类型为 T_REC_CTRL_INIT_PARAM 的变量 recCtlParam 来说明)：

```
//录像文件存储路径
Utl_UStrCpy(recCtlParam.recFilePath, Fwl_GetDefPath(eVIDEOREC_PATH));
//录像临时索引文件存储路径
Utl_UStrCpy(recCtlParam.indexFilePath, Fwl_GetDefPath(eRECIDX_PATH));
//循环录像时间间隔(毫秒), 此处配置为 15 分钟（如果为 0，则为普通录像）
```

recCtlParam.cycRecTimeMs = 15*60*1000;

//普通录像时间限制 (秒), 此处配置为 2 小时（如果为 0，则无时间限制）

recCtlParam.recTimeSecLimit = 2*60*60;

// 单个文件大小限制, 此处配置为 2 G 字节

recCtlParam.recSizeLimit = 2<<30;

//存储空间保留大小, 此处配置为 32M 字节，这些空间将不被使用

recCtlParam.recResSize = 32<<20;

//编码临时索引是否用 RamFile, 此处配置使用 DiskFile，不适用内存模拟

recCtlParam.useMemIndex = AK_FALSE;

//异步写文件缓冲大小, 此处配置为 2M 字节（如果为 0，则不适用异步写文件）

recCtlParam.asynWriteSize = (2<<20);

//当配置为循环录像时，此配置有效。用于在循环录像时存储空间不够用时，如何

//控制删除文件。此处配置指删除文件时，只要删到够用，则停止删除。

recCtlParam.autoDelAllFile = AK_FALSE;

//是否打开动态变焦功能

recCtlParam.isEnableFocus = AK_TRUE;

//录像时候静止画面的侦测间隔时间（毫秒）, 此处配置为 1 分钟

recCtlParam.detectNoMovingTimeMs = (1000 * 60 * 1);

//录像时候静止画面超时后，丢帧间隔时间，即多少时间编码一帧，此处配置为 1 秒

recCtlParam.encMsLimitPerFrame = 1000;

7.4.2 音频编码参数

用于指定录像音频编码，其参数保存在 一个类型为 T_REC_AUDIO_INIT_PARAM 的结构体中。

音频编码参数说明(此处以一个类型为 T_REC_AUDIO_INIT_PARAM 的变量 recAudioParam 来说明)：

//音频编码的格式, 此处配置为 wave 格式

recAudioParam.audioEncType = eRECORD_MODE_WAV;

//音频编码的采样率大小，此处配置为 8K 采样率，目前只能配置为 8000.

recAudioParam.audioEncSamp = 8000;

7.4.3 视频编码参数

用于指定录像视频编码，其参数保存在 一个类型为 T_REC_VIDEO_INIT_PARAM 的结构体中。

视频编码参数说明：(此处以一个类型为 T_REC_VIDEO_INIT_PARAM 的变量 recVideoParam 来说明)

//视频编码的格式，此处配置为 AVI 格式

recVideoParam.videoEncType = MEDIALIB_REC_AVI_NORMAL;

//视频编码的大小，此处配置为 VGA

recVideoParam.videoWidth = 640;

recVideoParam.videoHeight = 480;

//视频编码使用的源大小，此处配置为 VGA

recVideoParam.viedoSrcWinWidth = 640;

recVideoParam.viedoSrcWinHeight = 480;

//视频编码的帧率，此处配置为 30 帧

recVideoParam.FPS = 30;

//视频编码的码流比特率，此处配置为 20M 比特每秒

recVideoParam.vbps = 20<<20;

8 网络

8.1 整体架构

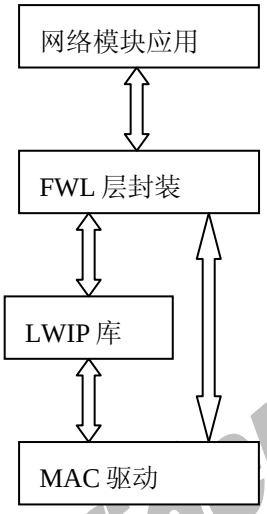

图 8-1 网络整体架构

上图描述了平台网络相关的调用层次。底层 MAC 驱动，注册函数给 lwip 库用； FWL 层对 LWIP 库的 API 接口和 MAC 驱动接口进行封装，供上层应用调用；上层应用经过 FWL 层，注册某些回调函数给 LWIP 库。

8.2 主要流程

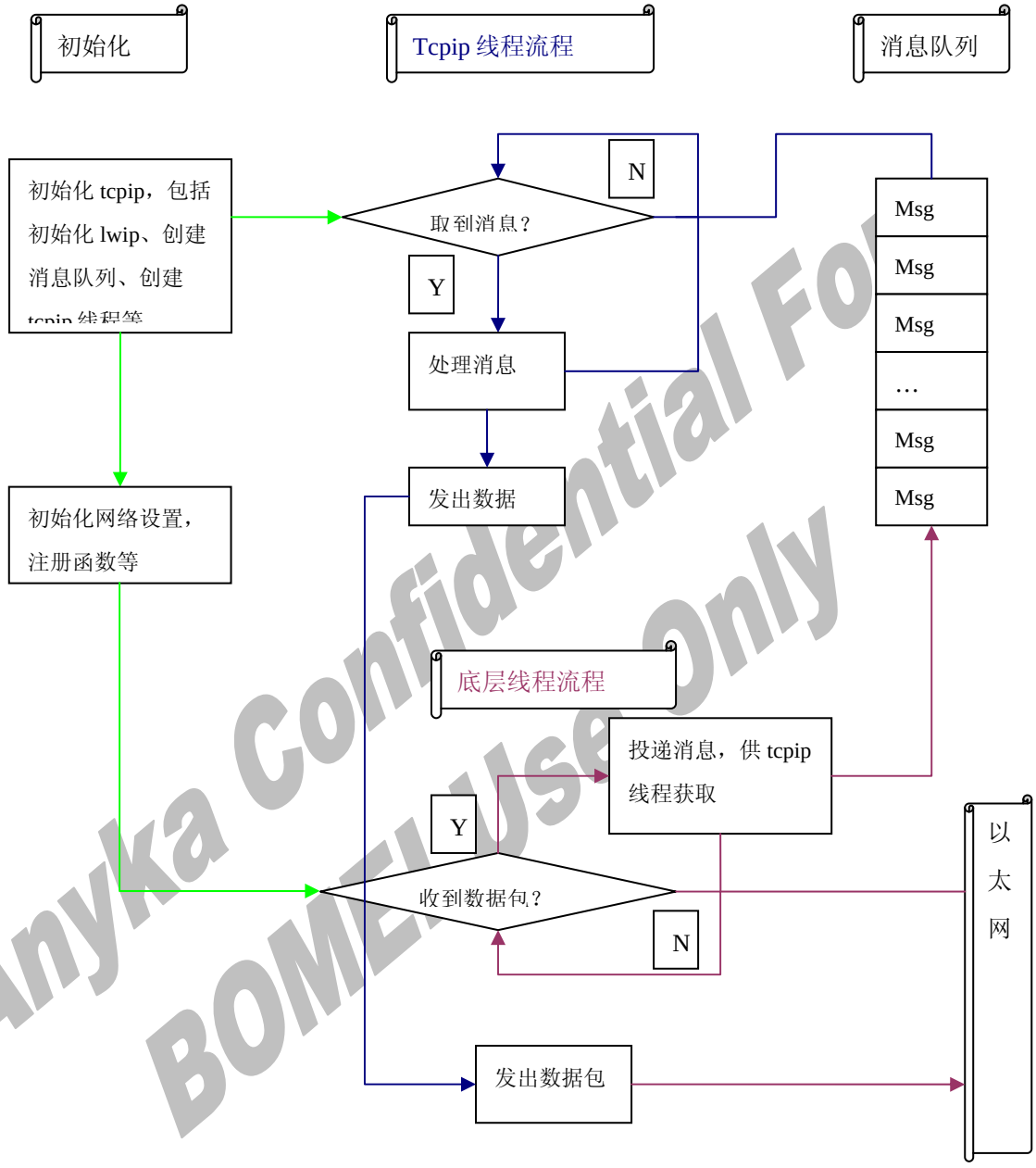

图 8-2 LWIP 主要流程

lwip 运行的主要流程：初始化时，会创建消息队列、创建 tcpip 线程以及初始化网络相关设置、（winpcap 模拟则需要创建 pcap 线程）注册 input、output、linkoutput 等函数。底层线程循环收数据包，投递消息到消息队列；Tcpip 线程循环从消息队列中取出消息，进行相应的处理；若处理后需要发出数据，则通过底层发出数据包。

8.3 运行内存占用

1、tcpip 线程的栈分配了 10K 内存，目前运行未发现溢出异常，可根据需要修改 TCPIP_THREAD_STACKSIZE 宏的值。

2、 lwip 堆内存 10K，可根据需要修改 MEM_SIZE 宏的值。

3、 lwip 内存池大约 30K，可根据需要修改 memp_std.h 里规定的各种类型的结构体分配个数。

4、 每个网络连接句柄，配备一个接收缓存 buffer，大小 8K，见 NETWORK_RECV_BUF_SIZE 宏的值，注意不要小于设定给 lwip 库的接收 buffer 大小 LWIP_RECV_BUF_SIZE 的值。

5、 其余是应用根据需要申请 buffer、创建新线程时申请栈等。

9 系统配置

9.1 SPI Flash 参数配置

SPI Flash 的配置参数，由烧录工具烧写到 SPI 介质上。配置参数由 T_SFLASH_PARAM 结构体定义，具体的参数说明，请参考驱动库。 Spiboot 、平台 MMI 通常都会用烧录的 SPI 配置参数。

9.1.1 SPI Flash 线模式

SPI 读写速度 由 SPI Clock 和 SPI 的线模式决定 。目前平台支持 1 线、2 线、4 线。线模式，由初始化函数 spi_flash_init 和参数配置函数 spi_flash_set_param 决定。只有在 spi_flash_init 设置了采用 4 线模式并且 spi_flash_set_param 设置的参数设置 4 线读写标识，4 线读写才启作用，否则按照 1 线模式读写。2 线模式设置原理相同。

目前平台烧录使用的是 1 线模式； 芯片读取 spiboot 采用的是 1 线模式(由给芯片的参数确定)；spiboot 启动采用的线模式由烧录的参数决定，可以是 1 线或者 4 线；平台采用的线模式也由烧录的参数决定，可以是 4 线或者 1 线。

9.2 MAC 地址

MAC 地址由烧录工具烧录在 SPI Flash BIN 区。关于 MAC 地址的烧录，请参考《烧录工具使用说明》。

10 配套工具使用说明

10.1 资源生成工具

有关资源生成工具的详细使用配置说明，请参见《Sword37 资源生成工具使用说明》文档。

10.2 烧录工具

有关烧录工具的详细使用配置说明，请参见《烧录工具使用说明》文档。

10.3 字模生成工具

有关字模生成工具的详细使用配置说明，请参见《Sword37 字模生成工具使用说明》文档。

10.4 多国语言资源生成工具

有关多国语言资源生成工具的详细使用说明，请参见《Sword37 多国语言资源生成工具使用说明》文档。