



Hi3510 客户端 H.264 解码库应用

Application Notes

文档版本	03
发布日期	2007-04-20
BOM编码	N/A

深圳市海思半导体有限公司为客户提供全方位的技术支持，用户可与就近的海思办事处联系，也可直接与公司总部联系。

深圳市海思半导体有限公司

地址： 深圳市龙岗区坂田华为基地华为电气生产中心 邮编： 518129

网址： <http://www.hisilicon.com>

客户服务电话： 0755-28788858

客户服务传真： 0755-28357515

客户服务邮箱： support@hisilicon.com

版权所有 © 深圳市海思半导体有限公司 2007。 保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



HISILICON、海思，均为深圳市海思半导体有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。



目 录

前 言.....	1
1 概 述.....	1-1
2 解码应用.....	2-1
2.1 解码库使用流程.....	2-2
2.2 Nalu/AU解码的码流分割.....	2-5
2.2.1 H264_BITSTREAM_S结构.....	2-5
2.2.2 码流分割流程.....	2-5
2.3 创建解码器句柄.....	2-7
2.4 销毁解码器句柄.....	2-8
2.5 流媒体网传的方式.....	2-8
2.6 判断I帧的方法.....	2-10
2.7 解码前的丢包策略.....	2-11
2.8 图像序列开始的 4 个小包.....	2-11
3 解码示例.....	3-1
3.1 流式接口解码示例.....	3-2
3.2 DecodeAU方式解码示例.....	3-5
4 其他.....	4-1
4.1 Deinterlace.....	4-2
4.1.1 概述.....	4-2
4.1.2 Deinterlace库调用流程.....	4-2
4.1.3 初始化Deinterlace.....	4-3
4.1.4 释放Deinterlace.....	4-4
4.1.5 Deinterlace主函数.....	4-4
4.2 跨度.....	4-5
4.2.1 概述.....	4-5
4.2.2 解码算法跨度.....	4-6
4.2.3 DirectDraw显示跨度.....	4-7
4.2.4 提高效率的方法.....	4-7



A 缩略语 A-1



插图目录

图 2-1 Nalu/AU方式解码库API函数使用流程图	2-3
图 2-2 流式解码库API函数使用流程图	2-4
图 2-3 码流分割流程图.....	2-6
图 4-1 Deinterlace库API函数调用流程图	4-3
图 4-2 跨度示意图.....	4-6



前 言

概述

本节介绍本文档的内容、对应的产品版本、适用的读者对象、行文表达约定、历史修订记录等。

产品版本

与本文档相对应的产品版本如下所示。

产品名称	产品版本
Hi3510 通信媒体处理器芯片（简称 Hi3510）	Hi3510 V100
	Hi3510 V101
	Hi3510 V110
Hi3510 DVS 解决方案	Hi3510 DMS V100R001
Hi3510 Video Phone 解决方案	Hi3510 DMS V200R001

读者对象

本文档适用于程序员初学者阅读，描述了基于 Hi3510 通信媒体处理器客户端 H.264 解码库应用的参考信息。使用本文档的程序员应该：

- 熟练使用 C++ 语言
- 掌握基本的 Windows32 调用

内容简介

本文档内容组织如下。



章节	内容
1 概 述	简单介绍 H.264 视频压缩编码标准的特性。
2 解码应用	介绍解码库调用流程、码流分割、创建和销毁解码器句柄等信息。
3 解码示例	提供流式接口解码和 DecodeAU 方式解码参考示例。
4 其他	介绍 Deinterlace 和跨度的相关信息。

约定

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
 危险	以本标志开始的文本表示有高度潜在危险，如果不能避免，会导致人员死亡或严重伤害。
 警告	以本标志开始的文本表示有中度或低度潜在危险，如果不能避免，可能导致人员轻微或中等伤害。
 注意	以本标志开始的文本表示有潜在风险，如果忽视这些文本，可能导致设备损坏、数据丢失、设备性能降低或不可预知的结果。
 窍门	以本标志开始的文本能帮助您解决某个问题或节省您的时间。
 说明	以本标志开始的文本是正文的附加信息，是对正文的强调和补充。

通用格式约定

格式	说明
宋体	正文采用宋体表示。
黑体	一级、二级、三级标题采用黑体。
楷体	警告、提示等内容一律用楷体，并且在内容前后增加线条与正文隔离。



格式	说明
“Terminal Display” 格式	“Terminal Display” 格式表示屏幕输出信息。此外，屏幕输出信息中夹杂的用户从终端输入的信息采用加粗字体表示。

修改记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

修改日期	版本	修改说明
2007-04-20	03	<ul style="list-style-type: none">• 文档名称由《Hi3510 客户端 H.264 解码应用》改为《Hi3510 客户端 H.264 解码库应用》。• 第 1 章概述：进行了简单的修改。• 第 2 章解码应用：本章作了很大调整，增加了许多内容。• 第 3 章解码示例：修改了解码示例。• 第 4 章其他：进行了简单的修改。• 增加了缩略语附录。
2006-12-20	02	增加了第 4 章其他。
2006-07-12	01	第一次发布



1 概述

2003年3月，ITU-T/ISO 正式公布了 H.264 视频压缩编码标准（简称 H.264）。与以往的视频压缩编码标准相比，H.264 具有更出色的性能，因此 H.264 被称为新一代视频压缩编码标准。

H.264 与 H.263 或 MPEG-4 相比，主要特性如下：

- 在相同图像质量下，H.264 能将数据码率降低一半。
- 在相同数据码率下，H.264 信噪比明显提高。

H.264 Decoder Library 是在 Windows 下的动态链接库，支持基于 Intel Pentium4 2.8G 以上 CPU 进行多路实时 H.264/MPEG-4 AVC 解码。本文阐述基于 H.264 Decoder Library 的解码应用。



2 解码应用

关于本章

本章描述内容如下表所示。

标题	内容
2.1 解码库使用流程	简单介绍解码库的使用流程。
2.2 Nalu/AU 解码的码流分割	简单介绍码流分割的结构和流程。
2.3 创建解码器句柄	介绍创建解码器句柄的方法。
2.4 销毁解码器句柄	介绍销毁解码器句柄的方法。
2.5 流媒体网传的方式	阐明流媒体网传要以 Nalu 为最基元的原因。
2.6 判断 I 帧的方法	给出判断 I 帧的方法。
2.7 解码前的丢包策略	简单介绍解码前的丢包策略。
2.8 图像序列开始的 4 个小包	简单介绍图像序列开始的 4 个小包的相关信息。



2.1 解码库使用流程

H.264 解码器 V1.0.0.23 新增流式易用性解码函数 (HI_H264DEC_DecodeFrame)，原有解码函数及功能特性保持不变。

图 2-1 和图 2-2 给出了使用 Nalu/AU 方式和流式解码库 API 函数使用示例流程。详细用法请参见“3.1 流式接口解码示例”。



图2-1 Nalu/AU 方式解码库 API 函数使用流程图

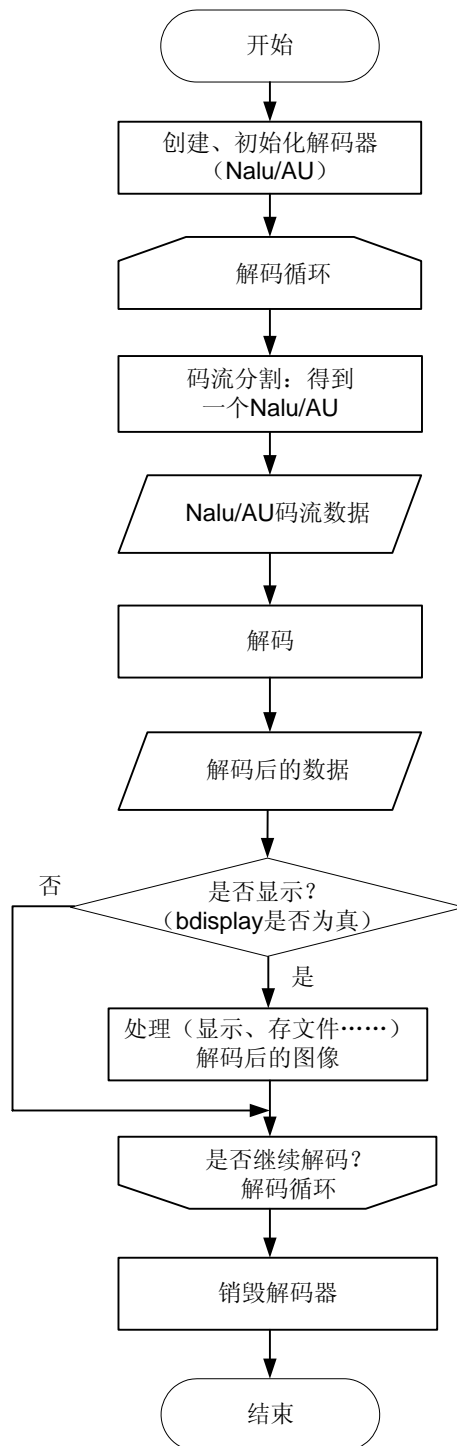
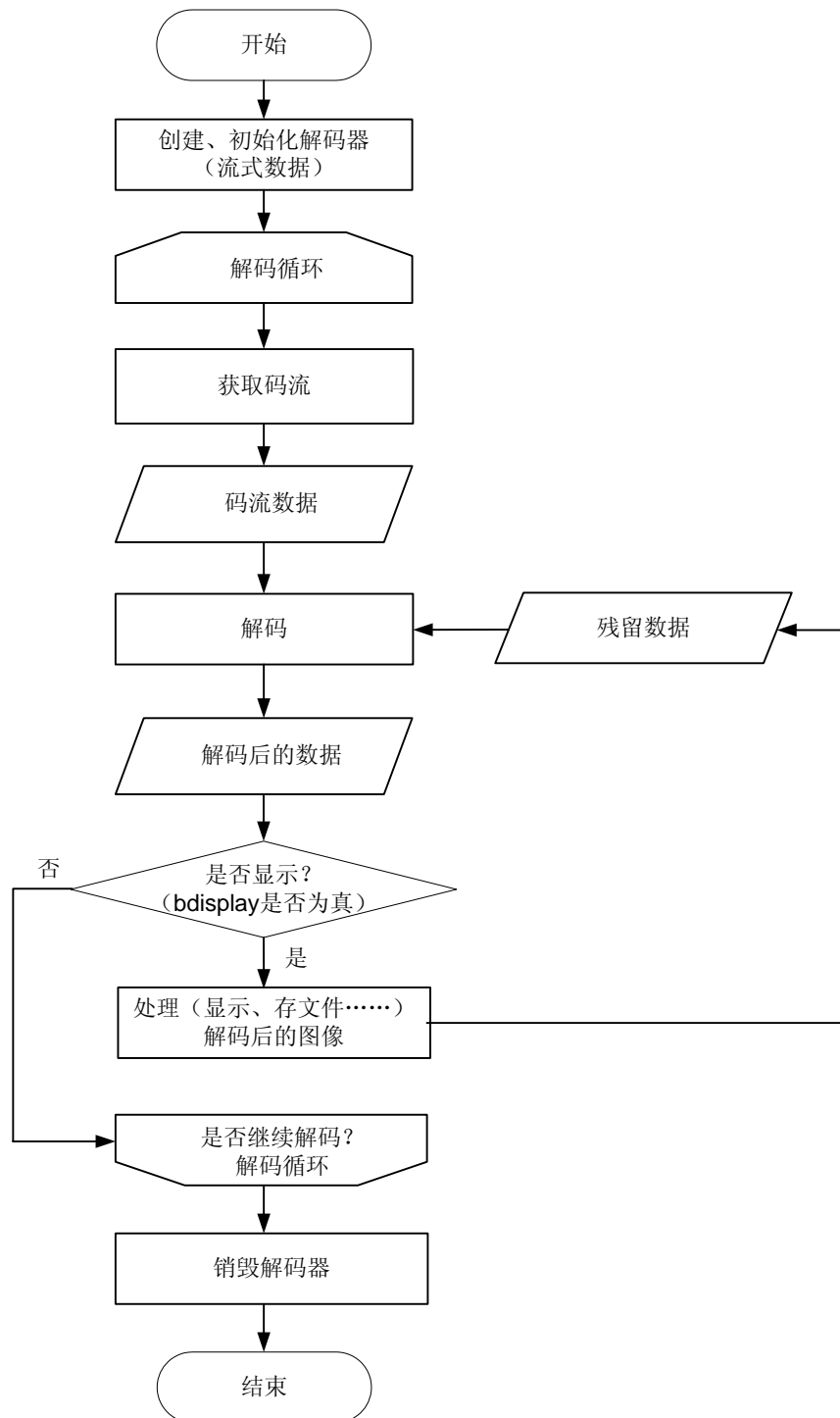




图2-2 流式解码库 API 函数使用流程图



码流数据可以是任意长度的线性码流，不局限于 Nalu 方式、AU 方式和流式。

解码后的残留数据的调用方式：将 DecodeFrame 的参数 pStream 置为空，pStreamLen 置成 0。



若将已知一帧边界的码流送给解码器，使用解码函数进行解码，还需要按照上面的流程解码两次。第一次成功解出一帧，第二次试图解码残留数据，解码器返回错误码，提示需要更多的码流。若不进行第二次解码，直接去解码下一帧数据，会出现花屏。

2.2 Nalu/AU 解码的码流分割

解码器 V1.0.0.23 版本新增了易用性流式解码函数(HI_H264DEC_DecodeFrame)，使用该函数解码无需考虑每次送码流的边界和 Nalu、AU、Nalu Type 等 H.264 语法概念，只需要将标准的 H.264 码流送给解码器即可，每次送码流尺寸最小为 1Byte，最大没有限制，但要考虑 PC 硬件的问题，最好不要超过 100MByte。

当使用解码器提供的 Decode_NALU 或 Decode_AU 函数进行解码时，因为原始码流是线性的，无法直接判断帧或 Nalu 边界，需要码流分割函数 Load_Nalu 或者 Load_AU 进行码流分割。分割后的数据单元是已知边界的 Nalu 或者 AU，再把这个数据送至解码器进行解码。

2.2.1 H264_BITSTREAM_S 结构

为了方便分割，在解码库中定义一个数据结构 H264_BITSTREAM_S，具体请参见《Hi3510 客户端 H.264 解码库 API 参考》，可在程序中直接引用。

掌握 H264_BITSTREAM_S 的每个变量具体含义，对正确分割码流、建立播放索引十分有帮助。

H264_BITSTREAM_S 定义如下：

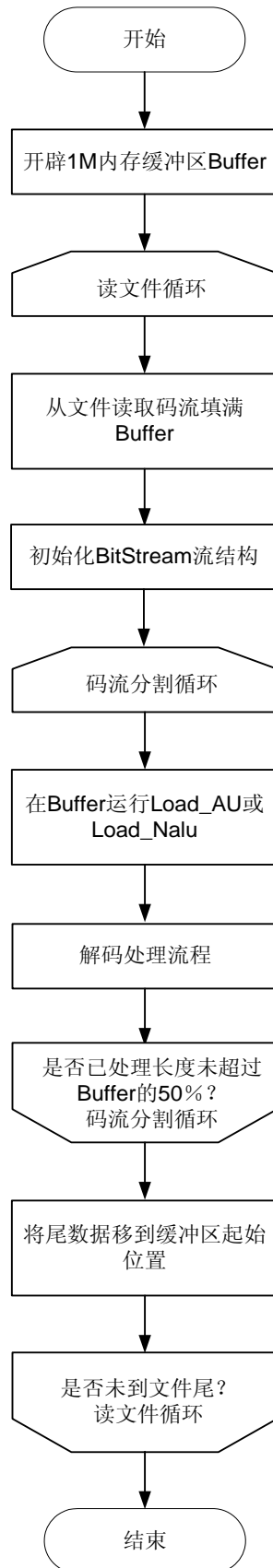
```
/* This data structure describes the bitstream buffers.*/  
typedef struct hiH264_BITSTREAM_S  
{  
    /*原始码流缓冲区*/  
    unsigned char *pStart;    /* 原始码流缓冲区起始指针 */  
    unsigned char *pEnd;    /* 原始码流缓冲区尾指针 */  
    unsigned char *pBuffer;    /* 每次码流分割后，pBuffer指针将指向下一单元 */  
    /* 分割码流缓冲区(存放分割后的码流数据单元) */  
    unsigned char *pStream;    /* 分割码流缓冲区起始指针 */  
    int iStreamLen;    /* 单元码流长度 */  
    int iUsed;    /* 每次分割原始码流的长度 */  
}H264_BITSTREAM_S;
```

2.2.2 码流分割流程

图 2-3 给出了针对 Nalu 或者 AU 方式解码的码流分割的流程。



图2-3 码流分割流程图





说明

图 2-3 点播的应用场景与图 2-4 回放文件的流程类似。

在图 2-3 的流程中，需要强调的是分割处理 Buffer 的数据时，不要处理到 Buffer 的尾部再填充新码流，处理到超过 Buffer 的一半时就填充新的码流，

填充新码流前：

步骤 1 将 Buffer 中剩余未处理的码流搬运到 Buffer 起始位置；

步骤 2 在 Buffer 剩余空间填充新的码流；

步骤 3 进入码流分割循环。

----结束



注意

这里的“一半”和图 2-3 中的“50%”都只是为说明而使用，具体数值用户可以自己设定。

该方案的特点是给码流分割处理营造线性 Buffer 环境，并且永远处理不到 Buffer 的尾部。在整个处理过程中，Load_Nalu 或者 Load_AU 函数没有处理到 Buffer 尾部的临界位置，Buffer 尾部码流是一段不确定的数据，一般不再是一个完整的 Nalu 或 AU，在进行码流切割时容易产生异常，如果将异常切割出的码流送给解码器，会导致解码器崩溃或者花屏等现象。本方案可以规避以上问题。

如果使用流式解码接口（HI_H264DEC_DecodeFrame）则不需要注意以上上层应用方案细节。

2.3 创建解码器句柄

创建解码器句柄函数 HI_H264DEC_Create（H264_DEC_ATTR_S *pDecAttr）在解码初始化时调用，通知解码器一些重要参数、创建内部参考空间、返回解码器句柄。

关于函数的详细描述请参见《Hi3510 客户端 H.264 解码库 API 参考》。

H264_DEC_ATTR_S 结构定义如下：

```
typedef struct hiH264_DEC_ATTR_S
{
    unsigned int    uPictureFormat;
    unsigned int    uStreamInType;
    unsigned int    uReserved;
}H264_DEC_ATTR_S;
```




uPictureFormat 参数

uPictureFormat 表示选择图像显示模式，uPictureFormat 可选项的选择如下：

```
/* Output picture format */  
#define HI_PICTURETYPE_I420    0  
#define HI_PICTURETYPE_I422    1  
#define HI_PICTURETYPE_RGB16   2
```

在调用解码器创建句柄函数前，建议先使用 DirectX 技术检测显卡支持的显示模式。大多数显卡支持 YUV 模式，如检测当前显卡不支持 YUV 模式，则使用 RGB16 模式。

RGB16 显示模式比 YUV 模式要多消耗 10%~20% 的 CPU 资源，在只支持 RGB16 模式的显卡上使用 YUV 模式显示，将显示黑屏。

uStreamInType 参数

uStreamInType 参数表示选择解码方式，包括 Nalu 方式、AU 方式和 RAW 方式。uStreamInType 可选项的选择如下：

```
//Input stream type  
#define HI_STREAMATYPE_NALU    0  
#define HI_STREAMATYPE_AU      1  
#define HI_STREAMATYPE_RAW     2
```

2.4 销毁解码器句柄

销毁解码器句柄的函数是 HI_H264DEC_Release (HI_HDL hDec)，在停止一个图像序列解码播放后调用该函数，销毁解码器的空间。

hDec 参数表示解码器句柄。



注意

每次调用 HI_H264DEC_Release 函数销毁解码器后，如果要重新使用解码器句柄，必须先将解码器句柄置为空再创建。

2.5 流媒体网传的方式

流媒体网传最好以 Nalu 为基本单元。

基于 MPEG-4 的方案，在网络传输时，通常用小于网络层的最大分组长度 (MTU) 的定长数据段来切分线性的原始码流，或者直接传输增加了私有格式头的一整帧数据，让网络层再次切分；上层应用则不理睬网络层由于网络壅塞而丢弃的包。



H.264 提出了比 MPEG-4 更优秀的抗误码解决方案。H.264 包含了 VCL（视频编码层）和 NAL（网络抽象层）。VCL 包括核心压缩引擎和块 / 宏块 / 片的语法级别定义，它被设计成尽可能地独立于网络。

NAL 将 VCL 产生的比特字符串适配到各种各样的网络和多元环境中，它覆盖了所有片级以上的语法级别，并包含了以下机制：

- 提供每个片解码时所需要的参数数据。
- 起始码冲突预防。
- 对附加增强信息（SEI）的支持。
- 将编码片的比特字符串在基于比特流的网络上进行传送的实现框架。

将 NAL 与 VCL 分离的主要目的有两个：

- 定义了一个 VCL 信号处理与 NAL 传输的接口，这样就允许 VCL 和 NAL 工作在完全不同的处理器平台上。
- VCL 和 NAL 都被设计成适应于异质传输环境，网关不需要因为网络环境不同而对 VCL 比特流进行重构和重编码。

IP 网络可分为三种类型：

- 不可控 IP 网络（如 Internet）。
- 可控 IP 网络（如广域网）。
- 无线 IP 网络（如 3G 网络）。

这三种 IP 网络有不同的最大传输单元（MTU）尺寸、比特出错概率和 TCP 使用标记。

最大传输单元尺寸是网络层最大的分组长度，H.264 编码时要使片的长度小于 MTU 尺寸，以避免在网络层再进行一次数据的分割。两个 IP 节点之间的 MTU 尺寸是动态变化的，通常假定有线 IP 网络的 MTU 尺寸是 1.5KByte，无线 IP 网络的 MTU 尺寸范围为从 100Byte 到 500Byte。

可见要适用于无线网络，H.264 必须采用数据分割技术使得片的长度小于 MTU 尺寸。

Hi3510 编码编出 Nalu 尺寸大于 1.4KByte 的概率非常小（最大尺寸为 2KByte），非常适合在有线网络中网传，用户在以 UDP 发包时，建议以 Nalu 为单元，可以做应用层封装，推荐使用 RTP 进行组包网传。一个 RTP 分组里放入一个 Nalu，将 Nalu（包括同步头）放入 RTP 的载荷中，设置 RTP 头信息。

由于包传送的路径不同，解码端要重新对片分组排序，RTP 包含的次序信息可以用来解决这一问题。

对于尺寸大于 MTU 的 UDP 分组，也不要随意丢弃，可直接发包，由网络驱动层进行拆包组包，对上层应用没有本质的影响。即使网络出现 UDP 丢包，也丢弃的是整个 Nalu。H.264 协议中，Nalu 是基于 VCL slice 的网络层封装，而每个 slice 对应于多个宏块，不同于 MPEG-4。对于 H.264 编码，宏块排列是非相邻的，因此，丢掉少量的 slice 只会图像个别区域出现短暂的块效应，对人眼是不敏感的。



注意

尽量避免送至解码器不完整的 Nalu，否则可能会导致崩溃等严重异常现象。

随着 H.264 的应用越来越广泛，在一段时期内，在项目实施中必将遇到 MPEG-4 与 H.264 并存的情形。



注意

有 MPEG-4 应用经验的用户在处理 H.264 码流时，一定要避免这样的思维定式：在发包侧，将一帧线性的码流直接等分成小于 MTU 的块加上私有头进行发包。

为了解码侧的系统稳定，避免软件崩溃，建议以 Nalu 为基元放入分组包里传送。这样可以做到 H.264 与 MPEG-4 互不干扰，稳定并存于一套系统中。

2.6 判断 I 帧的方法

在播放器设计中，定位播放、快进、快退都需要索引 I 帧。这里介绍如何识别一帧数据就是 I 帧，将线性码流分割出一帧的方案请参见“[2.2 Nalu/AU 解码的码流分割](#)”。

具体方案

H.264 码流是由 Nalu 序列组成，每个 Nalu 都有自己的类型。比如：IDR 包的类型值是 5。H.264 定义：IDR 帧内的所有 Nalu 类型都是 IDR，只有部分 IDR 包的帧不是 IDR 帧。Hi3510 的码流特性：IDR 帧内全是 IDR 包，P 帧内全无 IDR 包。

这样，对于海思 H.264 码流来说，I 帧的判断非常简单，只要一帧的第一个 Nalu 的 Nalu Type 值是 5 则该帧就是 I 帧。

计算 Nalu Type

Hi3510 编码出来的每个 Nalu 净荷数据前会有个同步头 0x00 00 00 01 四个字节，解码库 Load_AU 或者 Load_Nalu 码流分割出来的 Nalu 同步头是 0x00 00 01 三个字节。这两种序列都是符合 ISO MPEG 系列标准的。同步头后的第一个字节的低 5 位的数值就是 Nalu Type。Nalu Type 的详细资料请参见《新一代视频压缩编码标准 —H.264/AVC》。举例如下（假设存储一帧码流的缓冲区是 char *pBuffer）：

Hi3510 编码出的码流：

```
if (5 == (pBuffer[4] & 0x1F))
{
    /*当前帧是I帧*/
}
else
{
```



```
        /*当前帧是非I帧*/  
    }  
  
    解码库码流分割出来码流:  
  
    if (5 == (pBuffer[3] & 0x1F))  
    {  
        /*当前帧是I帧*/  
    }  
    else  
    {  
        /*当前帧是非I帧*/  
    }  
}
```



说明

pBuffer 取的位置不同，是因为两种码流同步头的差异。若做通用的判断 I 帧功能应用函数，需要将该帧所有 Nalu Type 列表，若全部都是 IDR 包，这帧才是 I 帧，否则就不是。

2.7 解码前的丢包策略

播放器在下列情况需要考虑丢包策略：

- 计算机 CPU 负荷过重或者网络抖动等原因引起的播放器内部解码前 Buffer 即将溢出。
- 上层应用检测到当前帧网络丢包严重。
- 播放器调度需要。

目前通用的做法：

- 尽可能将所有收到的信息送至解码器，不要轻易丢弃任何已经收到的码流信息。
- 如果不得已需要解码前丢弃帧，上层应用就从当前帧开始丢弃，一直丢弃到下一个 I 帧，这里需要注意的是丢弃整帧，不是丢弃 Nalu 包，需要上层应用确保送至解码器的是整齐的帧数据，避免参差不齐。

2.8 图像序列开始的 4 个小包

在 Hi3510 的 H.264 码流中，每个 I 帧前都会有 4 个小包(大小都为十几个 Byte)，分别为 SPS、PPS、PPS 和 SEI 包，存放着尺寸大小等图像序列关键信息。解码库只有遇到该组小包才开始进行解码，如果开始送至解码器 DecodeAU 的不是这些小包，那么解码时将返回错误。

针对 AU 方式的解码，每次解码器需要用码流分割函数 Load_AU 对码流进行分割。第一次遇到该组小包时，将陆续作为 4 帧数据分割出来，分 4 次送至解码器 DecodeAU；接下来就是 I 帧，再送至解码器，就有图像输出了。



注意

这里调用了 5 次解码，输出一帧图像，为避免播放器整体延迟。前 4 帧应该尽快解码，避免按照播放节拍调度。



3 解码示例

关于本章

本章描述内容如下表所示。

标题	内容
3.1 流式接口解码示例	提供流式接口解码参考示例。
3.2 DecodeAU 方式解码示例	提供 DecodeAU 方式解码参考示例。



3.1 流式接口解码示例

流式接口解码示例如下所示:

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <Winbase.h>
#include "hi_h264api.h"

int main(int argc, char *argv[])
{
    /* 解码库句柄 */
    HI_HDL hH264DecHandle = NULL;

    /* 初始化解码库的属性 */
    H264_DEC_ATTR_S struAttr;

    /* 解码输出图像结构变量 */
    H264_DEC_FRAME_S struOutframe;

    /* 码流源文件 */
    FILE* pstruH264File = NULL;

    /*成功解码帧数计数器*/
    int iNum = 0;

    /*默认每次送给解码器数据块的大小, 不限此值*/
    int iblockLength = 0x200;

    /*送给解码器数据块的大小*/
    unsigned int ulInstreamLen = 0;

    /*装载从文件中读取码流的内存*/
    unsigned char* pszInStream = NULL;

    /*每次送给解码器数据块的起始地址*/
    unsigned char* pszBuffer = NULL;

    /*函数执行返回值*/
```



```
int iRet = HI_H264DEC_OK;

/*设置解码后图像为YUV420格式*/
struAttr.uPictureFormat = HI_PICTYPE_I420;

/*设置解码库以流方式解码*/
struAttr.uStreamInType = HI_STREAMTYPE_RAW;

/*检查输入参数*/
if (argc < 2)
{
    fprintf(stderr, "No H.264 stream found!\n");
    fprintf(stderr, "Usage: h264cmd.exe stream.264 \n");
    goto exitmain;
}

/*打开文件*/
pstruH264File = fopen(argv[1], "rb");
if (pstruH264File == NULL)
{
    fprintf(stderr, "Unable to open pstruH264File file %s.\n", argv[1]);
    goto exitmain;
}

/*分配内存以存放成码流*/
pszInStream = (unsigned char*)malloc(1024*1024);
if (!pszInStream)
{
    fprintf(stderr, "No enough memory!\n");
    goto exitmain;
}

/*创建解码库句柄*/
hH264DecHandle = HI_H264DEC_Create(&struAttr);
if (!hH264DecHandle)
{
    goto exitmain;
}

/*循环读取码流，送给解码器解码*/
while (!feof(pstruH264File))
{
    /*以实际读取的字节数作为解码输入长度*/
```




```
    ulInstreamLen = fread(pszInStream, 1, iblockLength, pstruH264File) ;

    /*解码输入码流内存地址*/
    pszBuffer = pszInStream ;

    while (1)
    {
        /*以流方式解码*/
        iRet = HI_H264DEC_DecodeFrame(hH264DecHandle, pszBuffer,
ulInstreamLen, &struOutframe);
        if (HI_H264DEC_ERR_MBITS==iRet)
        {
            /*不足以解出一帧数据，需要更多的码流*/
            break;
        }

        if (struOutframe.bDisplay)
        {
            iNum ++;

            /* 解码成功，可以在此显示，或存YUV文件，Deinterlace等 */
        }
        /*为了对解码器中的残留数据进行解码，我们将输入码流地址设置为NULL，长度设置
为0*/
        pszBuffer = NULL;
        ulInstreamLen = 0;
    }
}

fprintf(stdout, "%d frame(s) decoded\n", iNum);

exitmain:

    if (pszInStream)
    {
        free(pszInStream);
    }
    if (pstruH264File)
        fclose(pstruH264File);
    if (hH264DecHandle)
        HI_H264DEC_Release(hH264DecHandle);

    return 0;
```



```
}
```

3.2 DecodeAU 方式解码示例

DecodeAU 方式解码示例如下所示:

```
/* 一帧码流解码示例 (片断) */  
/* 输入码流为一帧数据 */  
/* 注意: 板端对于D1来说是两场数据作为一帧发过来, 一起作为一帧数据输入到下面的函数进行解  
码 */  
/* 如果返回成功, 可以将输出图像用以显示 */  
bool ::DecoderOneFrame()  
{  
    if (NAL_TYPE_SPS == H264_Get_NalType(bs.pBuffer[4]))  
    {  
        /* 带SPS包的I帧, 一次吞吐 */  
        /* SPS */  
        HI_BITSTREAM_load_AU(&bs);  
        HI_H264DEC_DecodeAu(h, bs.pStream, bs.iStreamLen, &outframe);  
  
        /* PPS */  
        HI_BITSTREAM_load_AU(&bs);  
        HI_H264DEC_DecodeAu(h, bs.pStream, bs.iStreamLen, &outframe);  
  
        /* PPS */  
        HI_BITSTREAM_load_AU(&bs);  
        HI_H264DEC_DecodeAu(h, bs.pStream, bs.iStreamLen, &outframe);  
  
        /* SEI */  
        HI_BITSTREAM_load_AU(&bs);  
        HI_H264DEC_DecodeAu(h, bs.pStream, bs.iStreamLen, &outframe);  
    }  
  
    /* 如果码流分割成功 */  
    if (HI_H264DEC_OK == HI_BITSTREAM_load_AU(&bs))  
    {  
        /* 解码 (可能是CIF的一帧, 也可能是D1的前半场) */  
        HI_H264DEC_DecodeAu(h, bs.pStream, bs.iStreamLen, &outframe);  
  
        if (PROGRESSIVE != outframe.tPicFlag)  
        {
```



```
/* 如果是D1(已经解码了前半场) */
if (outframe.bDisplay)
{
    /* 输出图像到Deinterlace模块 */
    DeInterlace( );
}

/* 解码D1的后半场 */
HI_H264DEC_DecodeAu(h, bs.pBuffer, bs.pEnd - bs.pBuffer ,
&outframe);

if (outframe.bDisplay)
{
    /* 输出图像到Deinterlace模块 */
    DeInterlace( );
}
}
return outframe.bDisplay;
}
```



4 其他

关于本章

本章描述内容如下表所示。

标题	内容
4.1 Deinterlace	首先介绍 Deinterlace 的功能和特性,接着介绍 Deinterlace 库的调用流程,最后给出 Deinterlace 库的三个 API 函数的参考信息。
4.2 跨度	首先介绍跨度的分类,接着详细介绍两种跨度,最后给出提高图像处理效率的改进方法。



4.1 Deinterlace

4.1.1 概述

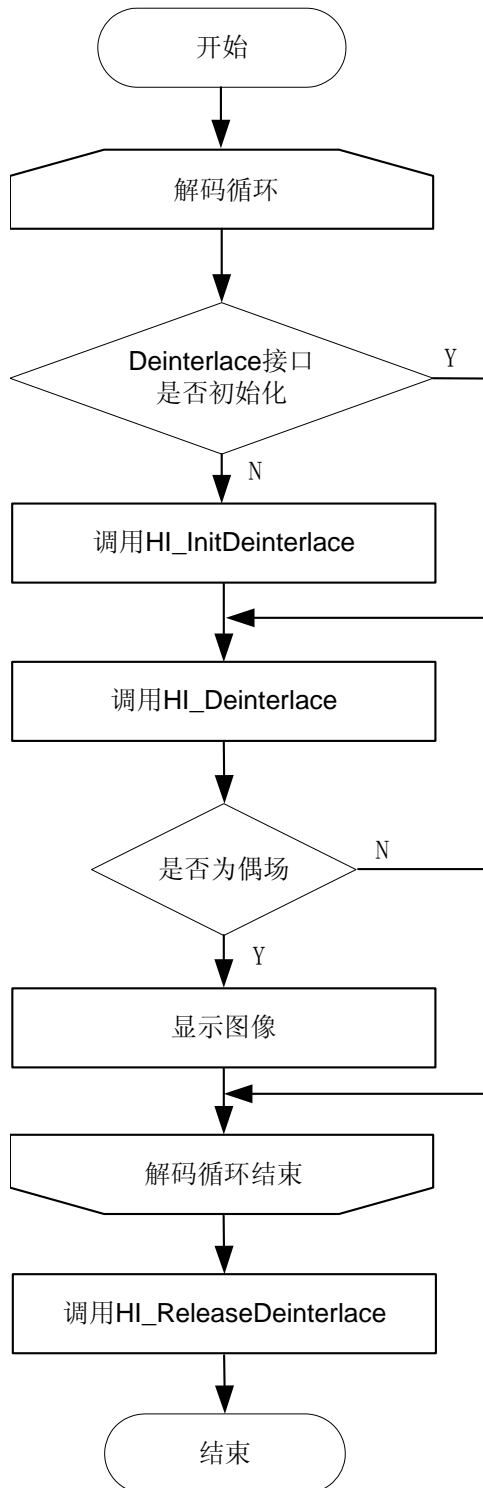
目前，绝大多数电视是隔行电视，导致电视节目仍是隔行扫描，视频采集设备也绝大多数都是按照 Interlace 视频（场交错成帧）隔行采集数据，但在 PC 端需按照逐行显示。由于组成一帧的奇场、偶场存在时间差，这些视频在逐行显示器播放过程中会出现锯齿、拉丝等图像质量问题，这就需要在播放前对视频进行去隔行交错处理（也称为去交错、Deinterlace）。

Hi3510 采集的 D1 图像按照场进行编解码，在 PC 端解码后，需要对图像进行 Deinterlace 处理。海思提供了进行 Deinterlace 处理的库，即 DllDeinterlace.lib。Deinterlace 库基于 YUV420，输入 YUV420 的场图像，每两场输出一帧去隔行了的 YUV420 的图像。该库较好的保持了图像的纹理细节，图像边缘清晰锐利，消除了锯齿现象，解决了剧烈运动最容易出现的拉丝现象，效率也达到了令人满意的程度。

4.1.2 Deinterlace 库调用流程

图 4-1 给出 Deinterlace 库 API 函数调用示例流程，详细用法可参见 Deinterlace 开发包。

图4-1 Deinterlace 库 API 函数调用流程图



4.1.3 初始化 Deinterlace

第一次解码之后，需先初始化 Deinterlace。在调用 Deinterlace 初始化函数前，需要把输入的场图像高宽、跨度信息以及输出帧图像的跨度信息配置进输入结构体参数中。初始



化成功则返回 Deinterlace 句柄，该句柄可供 Deinterlace 主函数和 Deinterlace 释放函数使用。

Deinterlace 初始化输入参数的结构体：

```
typedef struct hiDEINTERLACE_PARA_S
{
    int iFieldWidth;           //场宽
    int iFieldHeight;         //场高
    int iSrcYPitch;           //输入场图像Y分量的跨度
    int iSrcUVPitch;          //输入场图像UV分量的跨度
    int iDstYPitch;           //输出帧图像Y分量的跨度
    int iDstUVPitch;          //输出帧图像U、V分量的跨度
}DEINTERLACE_PARA_S;
```

对输入场图像的高宽、跨度信息的配置要求：

- 图像的宽度至少为 128。
- 图像的高度至少为 4。
- 图像的 Y 分量跨度至少为图像宽度。
- 图像的 U、V 分量跨度至少为图像宽度的一半。

对输出帧图像的跨度信息的配置要求：

- 图像的 Y 分量跨度至少为图像宽度。
- 图像的 U、V 分量跨度至少为图像宽度的一半。

4.1.4 释放 Deinterlace

在结束解码循环之后，需调用 Deinterlace 释放函数，释放所开辟的空间。



注意

调用 Deinterlace 释放函数后，还需把 Deinterlace 句柄置为空。

4.1.5 Deinterlace 主函数

解码循环中，每次解码之后可直接调用 Deinterlace 主函数。当把解码输出的 Y、U、V 分量地址直接赋给相应的输入参数，把解码输出的奇、偶场标志也赋给相应输入参数。Deinterlace 主函数会自动完成每偶场输出一帧图像的功能。

奇偶场标志枚举类型：

```
typedef enum hiPIC_TYPE_E
{
    PIC_PROGRESSIVE = 0,           //帧
```



```
PIC_INTERLACED_ODD,          //奇场, 顶场  
PIC_INTERLACED_EVEN         //偶场, 底场  
}PIC_TYPE_E;
```

输入参数判断信息:

- 当输入参数不为奇、偶场标志时, 不做任何处理直接返回错误码。
- 当输入奇场标志时, Deinterlace 库内部会保存数据, 但没有图像数据输出。
- 当输入偶场标志时, 有帧图像输出。

输出帧 YUV420 图像的结构体:

```
typedef struct hiDEINTERLACE_FRAME_S  
{  
    unsigned char *pszY;        //Y分量  
    unsigned char *pszU;        //U分量  
    unsigned char *pszV;        //V分量  
}DEINTERLACE_FRAME_S;
```

说明

Deinterlace 处理之后的图像是去隔行了的 YUV420 的帧图像, 若要送至显示, 还需要将它转换成显卡支持的图像格式。当然为了提高效率, 图像格式转换的目的地址可以直接为显存的地址。

4.2 跨度

4.2.1 概述

Hi3510 应用解码器进行解码, 将输出的 YUV 图像传送给 DirectDraw 显示模块。

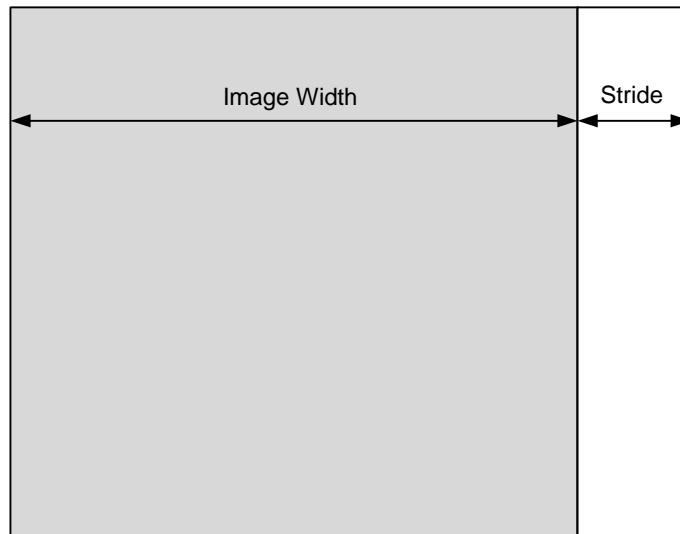
跨度在解码模块和 DirectDraw 显示模块的作用不同:

- 在解码模块是算法跨度。
- 在 DirectDraw 显示模块是显示跨度。

跨度示意图如图 4-2 所示。



图4-2 跨度示意图



4.2.2 解码算法跨度

H.264 都采用 YUV420 方式进行解码，并且 Y 分量、U 分量、V 分量分别存在于各自独立的线性空间。解码库中定义一般 Y 分量的跨度是 64Byte，U 或 V 分量的跨度是 32Byte。

在解码时，为了便于算法上处理图像中的一行像素，需要进行扩边处理，按照 YUV420 方式输出的图像，是带了扩边的毛胚图像。将图像传送给 DirectDraw 显示模块时，还需要进行去边处理。

解码库可以选择按照 YUV422 方式输出图像，实际上是在解码库内部出口处调用 YUV420TOUYVY422 函数。UYVY422 图像与 YUV420 图像不同，做了去边处理。发布的解码 Demo 例子程序中提供了 YUV420TOYUYV422 的转换程序源代码，可以清晰看到去掉算法跨度的过程。

解码库输出图像的结构：

```
typedef struct hiH264_DEC_FRAME_S
{
    unsigned char    *pY;
    unsigned char    *pU;
    unsigned char    *pV;

    int              iWidth;
    int              iHeight;
    int              iYStride;        //Y分量的算法跨度
    int              iUVStride;      //U或V分量的算法跨度
    int              bDisplay;
    H264_PIC_TYPE    tPicFlag;
    int              bIntra;
}
```



```
int iUserData;  
H264_USERDATA_S *pUserData;  
}H264_DEC_FRAME_S;
```

4.2.3 DirectDraw 显示跨度

DirectDraw 显示模块根据图像的宽与高在显存中开辟一块缓冲区，该缓冲区的逻辑宽度大于图像宽度，并且按照显卡指定的字节数对齐（不同显卡指定的字节数不同）。每次向显存缓冲区填充视频图像前，必须先锁定该缓冲区，锁定后就可以取出该缓冲区的实际宽度 `ddsd.lPitch`，这个数值减去图像宽度就是显示跨度。



注意

向显存缓冲区填充视频图像时，必须考虑显示跨度，否则，显示出来的图像会扭曲。

4.2.4 提高效率的方法

在播放器中统一考虑跨度问题，可以避免多次循环处理大幅图像。

理论和统计结果都表明，解码按照 YUV420 方式输出带算法跨度的毛胚图像，效率最高。

- 现有做法：应用 `YUV420TOYUYV422` 函数对整幅图像进行去除算法跨度后，送入 DirectDraw 显示模块，再对整幅图像进行循环增加显示跨度，然后送给 VO（Video Out）的显示页面进行显示。
- 改进方法：如果将 `YUV420TOUYVY422` 函数略加改动，解码后将去除算法跨度和增加显示跨度在一个循环内考虑，可以提高整体效率。输出的图像可以直接送入 VO 的显示页面进行显示。



A 缩略语

A

API Application Program Interface 应用程序界面

AU Access Unit 接入单元

I

IDR Instantaneous Decoding Refresh 解码及时刷新

ISO International Organization for Standardization 国际标准化组织

ITU-T International Telecommunication Union -
Telecommunication Standardization Sector 国际电信联盟-电信标准部

N

Nalu Network abstract layer unit NAL 单元