ID	UG-FIDEC_SDKUserManual		
SEC.	PUBLIC		
ISS.	2012/11/13	**REV.**	0.94

上海泛腾电子科技有限公司

FIDEC 通用音视频编解码开发包

SDK 使用手册

泛腾科技

2012/10/9

履历

版本	日期	修改人	说　　明
V0.94	2012/11/13	周康成	[A] 初始版本

目录

1 引言

1.1 编写目的

1.2 参考资料

1.3 缩写及术语

缩写/术语	全名及说明
DXE	Unified **D**ata **X**fer **E**ngine 泛腾科技研发的跨平台多数据源统一传输接口。
FIDEC	**FI**val multi-platform co**DEC** development package 泛腾科技研发的通用音视频编解码开发包，包含一块板载多核处理器的 PCIe 通用加速卡(代号 Polaris)以及软件 SDK。 用户基于 FIDEC 可快速开发高性能高灵活度的各种音视频应用。
FIDEC SDK	FIDEC 开发包中包含的 SDK。主要包括 FIDEC Server、Library、Monitor 及 Daemon 组件以及驱动等软件。其中 Server 提供编解码处理服务，Library 使用并向用户提供 API， Monitor 显示编解码处理结果及统计信息，由 Daemon 统一管理并监控。 下文中有时会简称为 SDK。
FIDEC Server	FIDEC 中封装多个高性能引擎 ,提供编解码服务的程序。利用 DXE 传输层与 FIDEC Library 交互数据，完成用户的各种编解码请求。可运行于多种平台。
FIDEC Library libfidec	FIDEC 中提供用户 API 接口的应用程序库 ,利用 DXE 传输层与 FIDEC Server 通讯完成编解码处理，可运行于多种平台。 下文中有时会称为 libfidec。
FIDEC Monitor	FIDEC 中用于显示编解码处理结果及统计信息的应用程序。 （缺省未包含在开发包中）
FIDEC Daemon	FIDEC 中用于管理及监控包含硬件在内的 Server 运行状态。 （开发中）
Polaris/Aries	泛腾科技研发的 PCIe 通用加速卡，板载多核处理器并运行 Linux 操作系统，已在音视频处理、加解密、网络处理等多领域成功应用。Polaris 和 Aries 尺寸均为全高半长，支持 PCIe2.0x8(Gen2)，Aries 板载 4 个 10G/1G 自适应以太网口，均可运行 FIDEC Server。 下文中有时会称为加速卡。

2 产品说明

2.1 产品简介

FIDEC 是泛腾科技研发的通用音视频编解码开发包，包含一块板载多核处理器的 PCIe 通用加速卡(代号 Polaris)以及软件 SDK，主要面向多路实时的音视频编解码应用开发。

2.2 产品特点

【全】SDK 中集成了多款不同特点的编解码引擎，面向各种不同的应用场景。图像格式及尺寸、质量、码流大小、性能等均可调整，对于特殊处理可以深度定制。

【快】易用的 API 接口使用户在短时期间就可以开发出各种音视频应用，缩短业务周期。开发及应用中常见的包括平台切换、编解码引擎切换、功能切换、参数修改在内的需求，不修改源码就可以实现，方便快捷。

【易】安装只需在普通 PC 或服务器中插入加速卡，解压 SDK 并执行启动脚本即可，无需额外工作。用户能通过 API 了解包括硬件在内的工作状态、负荷及各种统计数据，并可在线升级，大规模上线后的运维工作十分简单。

2.3 模块说明

SDK 中主要包括 Server、Library、Monitor 及 Daemon 组件， 其中 Server 为应用程序，利用内置的处理引擎提供编解码服务。Library 为应用程序库，向用户提供 API，使用 Server 的服务进行编解码处理。 Monitor 为应用程序，可以显示编解码处理结果和统计信息，Daemon 为守护进程，用于管理及监控包括硬件在内的 Server 运行状态。

2.3.1 Server

FIDEC Server 是集成 I/O、分发、处理引擎、管理等模块的服务程序。其中 I/O 采用泛腾科技研发的 DXE 跨平台多数据源统一传输接口，屏蔽了物理底层及各种协议的区别，便于移植和调试。分发采用线程池模型，保证在不同的负载下都能最大化处理器的性能。处理引擎用统一接口封装了几款经过优化的通用编解码器，并根据要求调用。

FIDEC Server 可以在主机上运行，也可以在 Polaris 加速卡上运行。在加速卡上运行时，libfidec 通过 PCIe 总线与加速卡通信，理论上 40Gbps 的带宽远远大于编解码的处理能力，不会成为系统瓶颈。编解码处理则由板载处理器完成，释放了主机的编解码负担，节约了宝贵的处理器资源。

Server 由于模块化程度非常高，可以深度定制并开发一些特殊功能，例如编解码时的 OSD，分割拼接、放大缩小等图像处理。如运行于 Aries 加速卡，还可以把编解码引擎输出的数据直接由网口发送。

FIDEC Server 已有标准 Linux 及 Tile-Linux 版本，可根据用户需求提供动态库、32 位或特定操作系统版本。

2.3.2 Library

FIDEC Library(libfidec)提供用户 API 接口，利用 DXE 传输层与 Server 交互数据完成编解码处理。复杂的编解码处理只需使用 open/proc/close 几个简单的函数即可完成。libfidec 支持最多 64 个各自独立的逻辑通道，各通道间完全独立，可以选用不同的 Server 及编解码引擎完成不同的功能，互不影响。支持多线程和多进程、同步和异步各种调用模式，符合各种不同复杂度的应用程序模型。

libfidec 已有 X86_64/i386 Linux 及 Windows 版本，可根据用户需求提供动态库或特定操作系统版本。

2.3.3 Monitor

FIDEC Monitor 是用于显示编解码处理结果及统计信息的应用程序。

可显示经编解码引擎输出、libfidec 转发的 H.264 ES 和 YUV420P 格式数据。与 libfidec 间的数据传输为 socket，支持多通道同时显示。在显示图像数据的同时还可显示通道统计数据，例如以 Server 为参照，当前输入输出数据的总块数、总字节数、当前传输速度、延迟，处理器的占用率等。

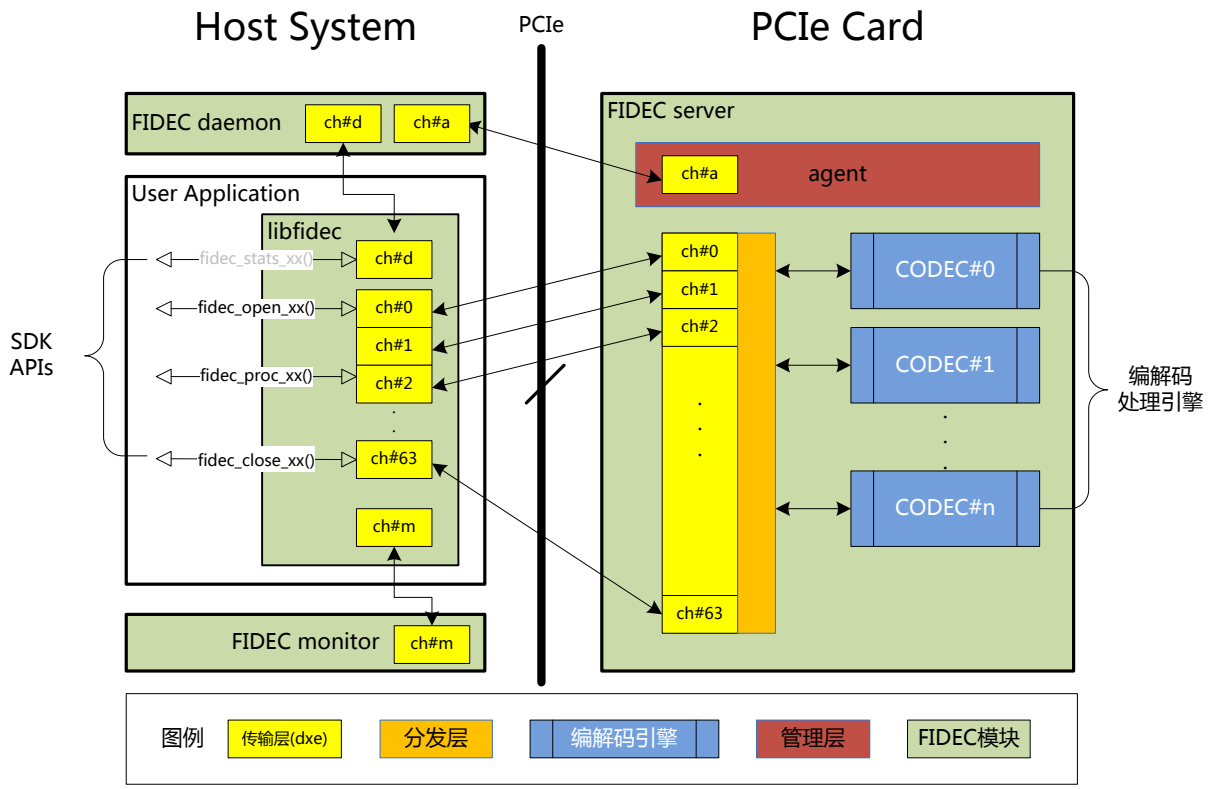
Monitor 不包含在 FIDEC 中。

2.3.4 Daemon

待定。

2.4 系统结构

以下为使用 Polaris 加速卡作为 FIDEC Server 时的系统结构图。



- 编解码数据流流向

User Application(SDK APIs) → libfidec → 传输层 ch#0~63 → 分发层 → 编解码引擎 → 分发层 → 传输层 ch#0-63 → libfidec → User Application (→ 传输层 ch#m → FIDEC Monitor)

- 管理数据流流向

User Application(SDK APIs) → libfidec → 传输层 ch#d → FIDEC daemon → 传输层 ch#a → Agent → 传输层 ch#a → FIDEC daemon → 传输层 ch#d → libfidec → User Application

2.5 参数

FIDEC 的主要功能及特点如下：

- 多通道音视频数据编解码，最大为 64 通道，各通道间完全独立，支持多进程同时使用。
- 编解码调用支持同步及异步方式。
- FIDEC Library 支持 X86 处理器平台(操作系统为 Windows/Linux)。
- FIDEC Server 支持 X86 及 Polaris 板载处理器，在 Polaris 加速卡上运行能达到最佳性能。
- Server 端具有多个编解码引擎，目前主力使用的编解码器功能如下：
  - 支持 H.263/H.263+，MPEG-1/2，MJPEG，AMR-NB。
  - 支持 H.264 baseline，支持 ES/RTP 封装，支持 slice 级并行，每通道可根据负载设置不同的并行线程数。
  - 支持 QCIF@15 至 1080P@60 的常见图像尺寸。
  - 支持固定码率，固定质量编码方式。质量、码率、帧率、I 帧间隔等、slice 并行数等参数可调。
  - H.264 main/high profile 的功能正在开发中
- ffmpeg 编解码引擎的功能正在优化，功能待定。

2.6 推荐开发平台

Linux

RHEL/CentOS 6.0(x86_64) 或以上

Ubuntu 12.04 Desktop i386

Windows

Microsoft® Windows7™(X64)

2.7已知问题及限制

- 各通道在打开后均为独占，无法与其他线程或进程共享
- 编码前或解码后图像帧数据只支持 YUV420P(planar)格式
- 编码图像宽度必须为 4 的倍数
- 缺省配置下编解码图像最大尺寸为 1920x1080

2.8 TODO

- 完成 FIDEC Daemon，增加编解卡自动启动、状态检测，统计数据及日志回传功能
- 支持 H.264 MainProfile/HighProfile 编码及解码，支持 TS 封装
- 支持各格式间转码
- 支持更多音视频格式及各格式间转码

3 安装及使用

3.1 安装

1） 安装 Polaris 加速卡

在 PC 或服务器机箱中安装开发包中所带的 Polaris 加速卡，加速卡需要 PCIe x8(或 x16)的插槽，安装时请确认加速卡紧密插入及完全固定。

2） 安装 SDK

Linux

SDK 安装包以 fidec_release_*${yymmdd}*.tar.gz 文件形式存在，安装时解压至任意目录即可。

Windows

待定。

安装包文件名中的*${yymmdd}*为年月日，代表版本号。

本文以后都以*$(FIDEC_ROOT)*表示安装目录。

3） 启动加速卡

Linux

取得 root 权限后在$(FIDEC_ROOT)目录下运行：

sbin/polaris.init start

并等待成功信息出现，可能耗时 20~25 秒。

如脚本运行失败(无 Start Successfullly 信息)，请检查加速卡硬件的安装或联系我们。

非 RHEL/CentOS 用户可能会需要 chkconfig 安装包，请自行下载并安装。否则有可能引起驱动不能在系统启动时加载，重启后必须安装驱动。

Windows

待定。

4） 编译测试程序

Linux

取得 root 权限后在$(FIDEC_ROOT)/src/simple_encoder 目录下运行：

make

会编译并生成 simple_encoder 可执行文件。

Windows

待定。

5） 运行测试程序

Linux

取得 root 权限后在$(FIDEC_ROOT)/src/simple_encoder 目录下运行：

#./simple_encoder --input=../../test_data/foreman_100_352x288_30.yuv --width=352 --height=288 --output-file

正常应该会在当前目录下生成 output_ch0.h264 码流文件。

Windows

待定。

3.2 目录及文件

3.2.1 doc

内含使用手册及文档，供参考。

文件名	说明
UG-FIDEC_SDKUserManual.pdf	FIDEC SDK 使用手册（本文档）
UG-POLARIS_user_manual.pdf	FIVAL Polaris 众核 PCIe 加速卡使用手册（制作中）

3.2.2 src

开发示例代码及测试程序。

文件名	说明
simple_encoder/	多通道 H.264 编码演示（使用 Polaris 加速卡）

3.2.3 include

开发用头文件。

文件名	说明
ascmn/errno.h	部分错误代码定义
ascmn/codec.h	部分错误代码定义，视频及封装格式定义以及 helper 函数定义
fica/fica.h	常用宏定义
fica/fica_list.h	链表数据结构及使用函数定义
fica/fica_opt.h	命令行输入参数解析函数定义
fica/fica_linwin.h	linux/windows 开发兼容性定义
fica/fica_mutex.h	互斥量结构及使用函数定义

3.2.4 lib

开发用库文件，全部以静态库形式提供。

Linux

文件名	说明
dxe/libdxe-x86_64.a	DXE 库，链接时需使用，x86_64 平台用
dxe/libdxe-i386.a	DXE 库，链接时需使用，32 位平台用
libfidec/libfidec-x86_64.a	FIDEC 用户 API 库(FIDEC Library) ，x86_64 平台用
libfidec/libfidec-i386.a	FIDEC 用户 API 库(FIDEC Library) ，32 位平台用

Windows

文件名	说明
dxe/libdxe.lib	DXE 库，链接时需使用
libfidec/libfidec.lib	FIDEC 用户 API 库(FIDEC Library)

3.2.5 test_data

测试数据。

文件名	说明
foreman_100_352x288_30.yuv	测试用 YUV 图像帧数据

3.2.6 sbin

管理工具、加速卡驱动及固件镜像。

文件名	说明
bootrom/fidec_svr-gx.bootrom	Polaris 加速卡用固件镜像，内含 FIDEC Server

Linux

文件名	说明
driver/linux	Polaris 加速卡驱动
polaris.init	Polaris 加速卡管理脚本

Windows

文件名	说明
driver/win	Windows 驱动

3.3 使用说明

Linux

【开发】

- 源码中加入*#include "libfidec/libfidec.h"*
- Makefile 编译参数中加入*-I$(FIDEC_ROOT)/include*
- x86_64 平台用户在 Makefile 编译参数中加入-DARCH_64BIT=1，链接参数中加入*-L$(FIDEC_ROOT)/lib/libfidec -lfidec-x86_64 -L$(FIDEC_ROOT)/lib/dxe -ldxe-x86_64 -lpthread*
- 32 位平台用户在 Makefile 链接参数中加入*-L$(FIDEC_ROOT)/lib/libfidec -lfidec-i386 -L$(FIDEC_ROOT)/lib/dxe -ldxe-i386 -lpthread*

【运行】

- 只需要 sbin/整个目录，SDK 中的其他内容可不使用。
- 使用 Polaris 加速卡时，每次服务器重启后需要运行一次启动加速卡脚本：
  $(FIDEC_ROOT)/sbin/polaris.init start
  并等待脚本结束后才能启动编解码应用程序。
- 加速卡运行异常时可再次运行启动脚本，但需保证所有使用 libfidec 的程序完全退出。

Windows

待定。

4 结构体/常量/宏定义

4.1 编解码通道

4.1.1 FIDEC_MAX_CHANNELS

【类型】 常量

【功能】 最大通道数定义

值	说 明
64	最大通道数，通道号为 0~63

4.2 视频数据

4.2.1 FIDEC_FORMAT

【类型】 枚举

【功能】 音视频格式定义

成员名称	值	说 明
FORMAT_RAW	0	二进制数据
FORMAT_YUV_STD	1	标准 YUV420P
FORMAT_YUV_FV	2	FIDEC YUV420P，编解码引擎内部使用
FORMAT_H264	3	H.264
FORMAT_MPEG2	4	MPEG2
FORMAT_MJPEG	5	MJPEG

4.2.2 FIDEC_ENCAP

【类型】 枚举

【功能】 音视频封装格式定义

成员名称	值	说　明
ENCAP_NONE	0	无封装
ENCAP_RTP	1	RTP
ENCAP_RTP_TAGGED	2	Tagged RTP（目前暂不支持）
ENCAP_TS	3	TS（目前暂不支持）

4.2.3 FIDEC_STREAM_DESC

【类型】 结构

【功能】 音视频码流格式及封装定义

成员变量类型	变量名称	说　明
FIDEC_FORMAT	fmt	指定音视频码流格式
FIDEC_ENCAP	encap	指定音视频码流封装格式

4.3 传输相关

4.3.1 FIDEC_CHUNK

【类型】 结构

【功能】 编解码数据块(data chunk)结构定义

成员变量类型	变量名称	说　明
(uint8_t*) [4]	data	数据块各段数据指针
int [4]	linesize	数据块各段单位大小
uint8_t*	base	数据块首指针

【使用说明】

编解码数据块用来容纳多段不连续数据（例如 YUV420P 中的 Y/U/V 平面数据）。

容纳 YUV420P 数据时与 ffmpeg 的 AVFrame 存储方式相同， data[0]/[1]/[2]分别为 YUV 数据段首指针，linesize[0]/[1]/[2]分别为 YUV 的行字节数。

容纳单数据段时 data[0]为数据段首指针，linesize[0]为数据段字节数。base 及其余空余字节不使用。

4.4 编解码功能

4.4.1 FIDEC_FN

【类型】 枚举

【功能】 编解码功能定义

成员名称	值	说　明
FN_DECODE	0	解码
FN_ENCODE	1	编码
FN_TRANSCODE	2	转码（暂不支持）

4.4.2 FIDEC_PARAM

【类型】 结构

【功能】 编解码参数定义

成员变量类型	变量名称	说　　明
int	asvc_id	保留
FIDEC_FN	fn	编解码功能（编/解/转）
FIDEC_STREAM_DESC	in	输入码流描述
FIDEC_STREAM_DESC	out	输出码流描述
int	width	图像宽度
int	height	图像高度
int	scale_width	保留
int	scale_height	保留
int	framerate	帧率
int	quality	质量（0~100）
int	is_fix_bitrate	固定码率模式标志位
int	bitrate	码率
int	qp	qp
int	min_qp	最小 qp
int	max_qp	最大 qp
int	iframe_interval	强制插入 I 帧间隔
int	pipelined_frames	最大帧间并行数
int	threads	最大使用线程数
int	allow_skip	允许跳帧标志位
int	mpi	保留
int	gdr_interval	保留
int	is_lowlatency	低延迟模式标志位
int	monitor_enabled	保留
uint64_t	monitor_channels	保留
char [32]	monitor_ip	保留
FIDEC_SERVER_TYPE	server	保留
int	debug	保留

4.4.3 FIDEC_IO_STATS

【类型】 结构

【功能】 编解码通道数据传输性能统计信息

成员变量类型	变量名称	说　　明
uint64_t	chunks	输入/输出总块数
uint64_t	bytes	输入/输出总字节数
float	bps_current	当前传输速率(bit/秒)
float	bps_max	最高传输速率(bit/秒)
float	bps_min	最低传输速率(bit/秒)
float	bps_avg	平均传输速率(bit/秒)
float	cps_current	当前传输速率(数据块/秒)
float	cps_max	最高传输速率(数据块/秒)
float	cps_min	最低传输速率(数据块/秒)
float	cps_avg	平均传输速率(数据块/秒)

4.4.4 FIDEC_CHECKPOINT

【类型】 结构

【功能】 编解码通道处理延迟信息

成员变量类型	变量名称	说　　明
uint64_t	frame_id	当前统计数据块 ID
uint64_t	start	开始处理时间，保留
uint64_t	submit	该数据块开始提交至 Server 时的时间戳
uint64_t	submit_done	该数据块提交完成时的时间戳
uint64_t	done	该数据块在 Server 处理完毕时（包含数据回传）的时间戳
uint64_t	end	保留

4.4.5 FIDEC_STATS

【类型】 结构

【功能】 编解码通道统计信息

成员变量类型	变量名称	说　　明
uint64_t	tm_init	通道打开时间
uint64_t	tm_term	通道关闭时间
FIDEC_STREAM_DESC	in	输入数据统计数据
FIDEC_STREAM_DESC	out	输出数据统计数据
FIDEC_CHECKPOINT	chk_pnts	每帧延时统计数据
FIDEC_CH_STATUS	status	通道状态
int32_t	cpu_load	FIDEC Server 负载，范围 0~100
pthread_mutex_t	lock	保留
uint64_t[2]	dummy	32/64 位系统对齐用，保留

5 函数一览

5.1 初始化/终止

函 数 名	说　明
fidec_init	全局初始化
fidec_term	全局终止

5.2 编解码处理

函 数 名	说　明
fidec_open_codec	用指定编解码参数打开一个编解码通道，通道在打开后可以进行编解码处理。
fidec_proc_codec	向已打开的编解码通道输入待处理数据，并等待处理完毕，通道在打开后可反复调用直至全部数据完毕。
fidec_close_codec	关闭一个编解码通道。

5.3 参数设置/数据转换

函 数 名	说　明
fidec_setparam_encode_h264	初始化编解码参数并设置为 H264 编码模式
fidec_setparam_decode_h264	初始化编解码参数并设置为 H264 解码模式
fidec_setchunk_yuv420	转换 YUV420P planar 数据至 FIDEC_CHUNK 格式
fidec_setchunk_raw	转换通用二进制数据至 FIDEC_CHUNK 格式

5.4 其他

函 数 名	说　明
fidec_get_stats	取得编解码通道的统计数据
fidec_get_version	取得 FIDEC Library 的软件版本号

6 函数详细说明

6.1 初始化/终止

6.1.1 fidec_init

【原型】 int fidec_init(void)

【功能】

全局初始化

【参数】

无

【返回值】

返回值		说　明
FIDEC_RET_SUCCESS(0)		成功
< 0	ASE_BAD_SEQ	已初始化

【使用说明】

在使用 FIDEC 编解码函数前必须调用。多进程模型每进程只需调用一次，多线程模型在创建线程前调用一次。

与 fidec_term()配对使用。

【调用示例】

```
int main()
{
    ...
    int ret = fidec_init();
    if (ret < 0) {
        printf("fidec_init() returned %d, quit!", ret);
        exit(-1);
    }
    ...
}
```

6.1.2 fidec_term

【原型】 int fidec_term(void)

【功能】

全局终止

【参数】

无

【返回值】

返回值		说　明
FIDEC_RET_SUCCESS(0)		成功
< 0	ASE_BAD_SEQ	未初始化或有编解码通道未关闭

【使用说明】

在程序退出前必须关闭所有编解码通道并调用 fidec_term()。多进程模型每进程只需调用一次，多线程模型在线程全部退出后调用一次。

与 fidec_init()配对使用。

【调用示例】

```
int main()
{
    ...
    pthread_join(&worker_thread, NULL);

    int ret = fidec_term();
    if (ret < 0) {
        printf("fidec_term() returned %d.", ret);
    }
    ...
}
```

6.2 编解码处理

6.2.1 fidec_open_codec

【原型】 int fidec_open_codec(int ch,
FIDEC_PARAM* param,
FIDEC_CBFN cb_fn,
void* cb_context)

【功能】

用指定编解码参数打开一个编解码通道。

【参数】

参数类型	参数名	I/O	说　　明
int	ch	I	通道号
FIDEC_PARAM *	param	I	编解码参数
FIDEC_CBFN	cb_fn	I	回调函数
void*	cb_context	I	回调用户上下文

【返回值】

返回值		说　　明
FIDEC_RET_SUCCESS(0)		成功
< 0	ASE_INVALID_CH	非法通道号
	ASE_INVALID_PARA	非法参数
	ASE_BAS_SEQ	该通道已被使用
	ASE_IO_ERROR	硬件错误

【使用说明】

每个编解码通道必须使用 fidec_open_codec 打开后才能使用 ,通道一旦成功打开后，在关闭前无法再次打开。

在多进程模型下，由于 FIDEC Library 无法识别通道是否已被占用，请用户自行保证通道号的唯一性，不要重复使用。

编解码功能、使用的输入输出数据格式以及其他参数（如图像尺寸、码率等）通过 param 编解码参数指定，通道一旦成功打开则无法改变。如需改变必须先行关闭通道，修改编解码参数后再次打开。param 编解码参数由于项目较多，可以使用 fidec_setparam_xxx 系列函数初始化，用户在不明确用途的情况下请勿随意修改。

cb_fn 为用户回调函数，每段编解码数据处理完毕后被调用。回调函数不区分用户调用 fidec_proc_xxx 函数时使用同步或异步模式，只要设置就会被调用。cb_context 为回调函数的用户上下文，由用户自行设置使用。如用户不需要回调函数请设置成 NULL。

与 fidec_close_codec()配对使用。

【调用示例】

```
int main()
{
    ...
    CODEC_PARAM param;
    int ch = 7; /* channel id */
    fidec_setparam_encode_h264(&param);
    ret = fidec_open_codec(ch, &param, NULL, NULL);
    if (ret < 0) {
        printf("fidec_open_codec() returned %d, quit!", ret);
        exit(-1);
    }
    ...
}
```

6.2.2 fidec_proc_codec

【原型】 int fidec_open_codec (int ch,
FIDEC_CHUNK* picture,
uint8_t* obuf,
size_t* osize,
int timeout)

【功能】

向已打开的编解码通道输入待编解码数据，并等待处理完毕。

【参数】

参数类型	参数名	I/O	说　　明
int	ch	I	通道号
FIDEC_CHUNK*	picture	I	输入数据
uint8_t*	obuf	I	输出缓冲区指针
size_t*	osize	I/O	输出缓冲区大小（单位：字节）
int	timeout	I	超时（单位：毫秒）

【返回值】

返回值		说　　明
FIDEC_RET_SUCCESS(0)		成功
FIDEC_RET_TIMEOUT		超时
< 0	ASE_INVALID_CH	非法通道号
	ASE_INVALID_PARA	非法参数
	ASE_BAS_SEQ	该通道未打开
	ASE_NOT_ENOUGH_BUF	输出缓冲区过小，不能容纳输出数据
	ASE_IO_ERROR	硬件错误

【使用说明】

每个编解码通道在打开后可以使用 fidec_proc_codec 向已打开的编解码通道输入待编解码数据，并等待处理完毕。在编码时输入 YUV420P 图像帧数据，输出 H264 码流，解码时输入 H264 码流，输出 YUV420P 图像帧数据。通道在打开后可反复调用直至全部数据处理完毕。

输入数据为 FIDEC_CHUNK 数据块格式，必要时可以通过 fidec_setchunk_yuv420 或 fidec_setchunk_raw 进行用户数据至 FIDEC_CHUNK 数据块的格式转换。在编码时如用户使用 RGB 格式，或者 YUV420 数据为 packed 格式请转换至 planar 再进行编码。FIDEC_CHUNK 存储 YUV420P 数据时不要求 YUV 平面数据连续，如用户使用类似 ffmpeg 的 AVFrame 格式可以直接复制 AVFrame 中 YUV 平面的 data[]指针及 linesize[]至 FIDEC_CHUNK 结构后进行编码。

输入数据块的格式及大小必须符合通道打开时所使用的编解码参数。

输出缓冲区由用户分配及释放，调用时设置缓冲区指针(obuf)及相应大小(osize)，libfidec 填入输出数据后会修改 osize 大小至输出数据块实际大小，在输出数据块过大时不会填写并返回 ASE_NOT_ENOUGH_BUF。处理失败或无数据时 libfidec 会重置 osize 的大小为 0。

输入数据送至相应引擎处理后，fidec_proc_xx 会等待该处理完毕并有数据输出，等待的时间可以通过 timeout 设置。timeout<0 为同步模式，即一直阻塞至处理完毕。timeout>=0 为异步模式，函数不等待处理结果，所有输出数据会投入一个数据池并按照 FIFO 顺序依次由函数返回。出于效率考虑，在同步模式下第一块输入数据不会等待处理结果，直接返回 FIDEC_RET_TIMEOUT，请用户判断函数返回值及 osize 正常后再处理缓冲区中数据。

如打开通道时注册了用户回调函数 ,在每次处理结束有输出数据时 FIDEC Library 会回调该函数 ,回调在 libfdec 的接收线程中完成，用户的工作线程使用时需注意线程间同步及互斥。libfidec 会使用回调函数的返回值，如<0 则会丢弃该数据。回调函数中的输出数据缓冲区为 libfidec 管理，用户无需释放。

推荐用户在异步模式下使用回调函数中的输出数据并返回<0 ,数据被丢弃后 fidec_proc_codec 函数会一直返回 FIDEC_TIMEOUT。在同步模式如有回调函数请勿返回<0，否则会导致数据丢弃后 fidec_proc_codec 的阻塞。

【调用示例】

```
int main()
{
    ...
    CODEC_PARAM param;
    int ch = 7; /* channel id */
    char* yuv_frame; /* YUV420P planar */
    char* h264_buffer = malloc(4*1024*1024);
    size_t h264_buf_size = 4*1024*1024;
    FIDEC_CHUNK chunk;

    fidec_setparam_encode_h264(&param);
    ret = fidec_open_codec(ch, &param, NULL, NULL);
    if (ret < 0) {
        printf("fidec_open_codec() returned %d, quit!", ret);
        exit(-1);
    }
    while(1) {
        fidec_setchunk_yuv420(&chunk, yuv_frame, frame_width, frame_height);
        h264_buf_size = 4*1024*1024;
        ret = fidec_proc_codec(ch, &chunk, h264_buffer, &h264_buffer_size, -1/*sync*/)
        if (ret < 0) {
            printf("fidec_ proc _codec() returned %d, abort!", ret);
            break;
        }
        if (osize > 0) {
            /* handle h264_buffer */
            ...
        }
        yuv_frame = /* get next frame */
    }
    fidec_close_codec(ch);
    ...
}
```

6.2.3 fidec_close_codec

【原型】 int fidec_close_codec(int ch)

【功能】

关闭相应编解码通道。

【参数】

参数类型	参数名	I/O	说　　明
int	ch	I	通道号

【返回值】

返回值	说　　明
FIDEC_RET_SUCCESS(0)	成功

【使用说明】

与 fidec_open_codec()配对使用。

【调用示例】

参见 fidec_proc_codec()示例代码。

6.3 参数设置/数据转换

6.3.1 fidec_setparam_encode_h264

【原型】void fidec_setparam_encode_h264(FIDEC_PARAM* param)

【功能】

设置编解码参数为 H264 编码模式的缺省值。

【参数】

参数类型	参数名	I/O	说　明
FIDEC_PARAM*	param	I/O	编解码参数

【返回值】

无

【使用说明】

使用内置的缺省值对编解码参数 param 进行初始化，并设置成 H264 的编码模式。

缺省值如下：

成员变量名称	说明	设置值
fn	编解码功能	FN_ENCODE
in	输入码流描述	ENCAP_NONE:FORMAT_YUV_STD
out	输出码流描述	ENCAP_NONE:FORMAT_H264
width	图像宽度	1024
height	图像高度	768
framerate	帧率	30
quality	质量	50
is_fix_bitrate	固定码率模式标志位	1
bitrate	码率	6*1000*1000
qp	qp	26
min_qp	最小 qp	0
max_qp	最大 qp	0
iframeinterval	强制插入 I 帧间隔	30
threads	最大使用线程数	8
allow_frame_skipping	允许跳帧标志位	1
lowlatency	低延迟模式标志位	0
pipelined_frames	最大帧间并行数	1

【调用示例】

```
int main()
{
    ...
    CODEC_PARAM param;
    int ch = 7; /* channel id */
    fidec_setparam_encode_h264(&param);
    /* our frame is 1920x1080@60 */
    param.width=1920;
    param.height=1080;
    param.framerate = param.iframeinterval = 60;
    ret = fidec_open_codec(ch, &param, NULL, NULL);
    if (ret < 0) {
        printf("fidec_open_codec() returned %d, quit!", ret);
        exit(-1);
    }
    ...
}
```

6.3.2 fidec_setparam_decode_h264

【原型】void fidec_setparam_decode_h264(FIDEC_PARAM* param)

【功能】

设置编解码参数为 H264 解码模式的缺省值。

【参数】

参数类型	参数名	I/O	说　　明
FIDEC_PARAM*	param	I/O	编解码参数

【返回值】

无

【使用说明】

使用内置的缺省值对编解码参数 param 进行初始化，并设置成 H264 的解码模式。

缺省值如下：

成员变量名称	说明	设置值
fn	编解码功能	FN_DECODE
in	输入码流描述	ENCAP_NONE:FORMAT_H264
out	输出码流描述	ENCAP_NONE:FORMAT_YUV_STD
threads	最大使用线程数	8
pipelined_frames	最大帧间并行数	1

【调用示例】

```
int main()
{
    ...
    CODEC_PARAM param;
    int ch = 7; /* channel id */
    fidec_setparam_decode_h264(&param);
    ret = fidec_open_codec(ch, &param, NULL, NULL);
    if (ret < 0) {
        printf("fidec_open_codec() returned %d, quit!", ret);
        exit(-1);
    }
    ...
}
```

6.3.3 fidec_setchunk_yuv420

【原型】void fidec_setchunk_yuv420(FIDEC_CHUNK* picture,
uint8_t* ibuf,
int width,
int height)

【功能】

主要用于编码时转换用户 YUV420P 数据至 fidec_proc_xx 函数所使用的 FIDEC_CHUNK 数据块格式。

【参数】

参数类型	参数名	I/O	说　　明
FIDEC_CHUNK*	picture	I/O	数据块结构体
uint8_t*	ibuf	I	YUV 平面间连续的 YUV420P 数据缓冲区指针
int	width	I	YUV 数据的图像宽度
int	height	I	YUV 数据的图像高度

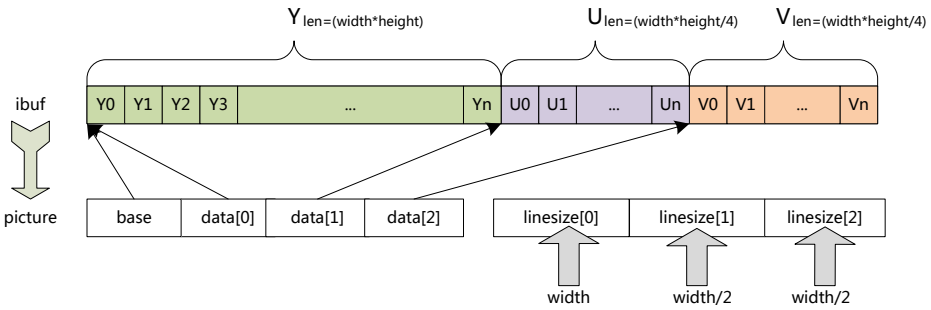
【返回值】

无

【使用说明】

如用户数据为 YUV 平面间连续的 YUV420P 数据（例如直接从文件读出的图像帧数据），则可以使用该函数快速转换成 FIDEC_CHUNK 格式进行编码处理。

用户数据要求及转换示意图如下：



【调用示例】

略

6.3.4 fidec_setchunk_raw

【原型】void fidec_setchunk_raw(FIDEC_CHUNK* picture,
uint8_t ibuf,
size_t size)

【功能】

主要用于解码时转换码流数据至 fidec_proc_xx 函数所使用的 FIDEC_CHUNK 数据块格式。

【参数】

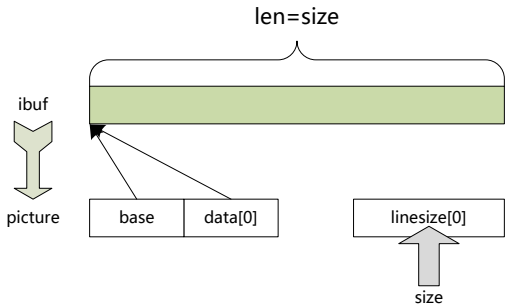
参数类型	参数名	I/O	说　　明
FIDEC_CHUNK*	picture	I/O	数据块结构体
uint8_t*	ibuf	I	用户数据指针
size_t	size	I	用户数据长度（字节）

【返回值】

无

【使用说明】

用户数据要求及转换示意图如下：



【调用示例】

略

6.4 其他

6.4.1 fidec_get_stats

【原型】void fidec_get_stats(int ch, FIDEC_STATS* stats)

【功能】

读取编解码通道状态及统计数据。

【参数】

参数类型	参数名	I/O	说　明
int	ch	I	通道号
FIDEC_STATS*	stats	I/O	状态及统计数据

【返回值】

返回值	说　明
FIDEC_RET_SUCCESS(0)	成功
< 0	失败

【使用说明】

主要统计数据分为两大类：数据传输性能及处理延迟。

数据传输性能统计包含对于传输次数及字节数的统计，输入、输出均含有最大、最小、平均及当前速率信息。

处理延迟数据中的时间戳均为相对时间，单位为 us，两个处理间的时间戳相减即可得到两个处理间的延迟。目前只统计用户数据开始提交至 Server、提交完毕以及 Server 处理完毕（包含数据回传延尽）的时间戳。从开始提交至处理完毕的延迟可以认为是整个编解码器的延迟时间。

【使用示例】

传输性能统计数据的使用示例请参见 simple_encoder.c 中的__print_stats_ch 函数。

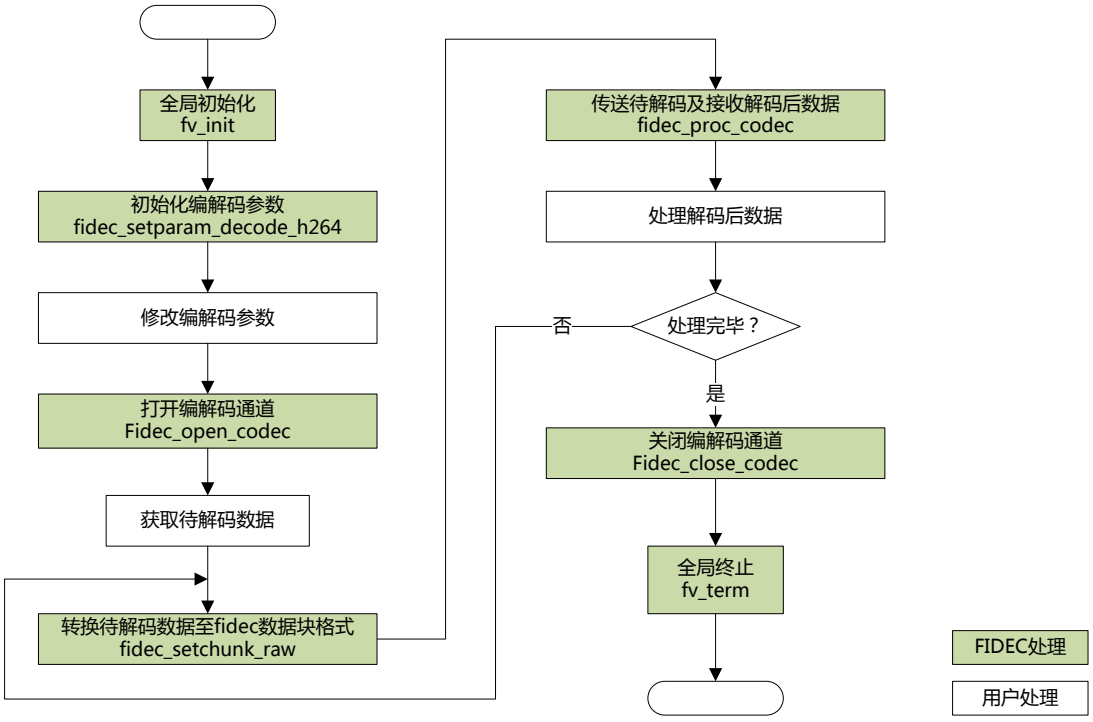
处理延迟统计数据的使用示例请参见以下代码。

```
int main()
{
    ...
    FIDEC_STATS stats;
    int ret = fidec_get_stats(ch, &stats);
    if (ret == 0) {
        printf("frame#%llu: cost %.3f ms\n",
                stats.frame_id,
                (float) (stats.chk_pnts.done - stats.chk_pnts.submit)/1000);
    }
    ...
}
```

7 应用编写方法举例

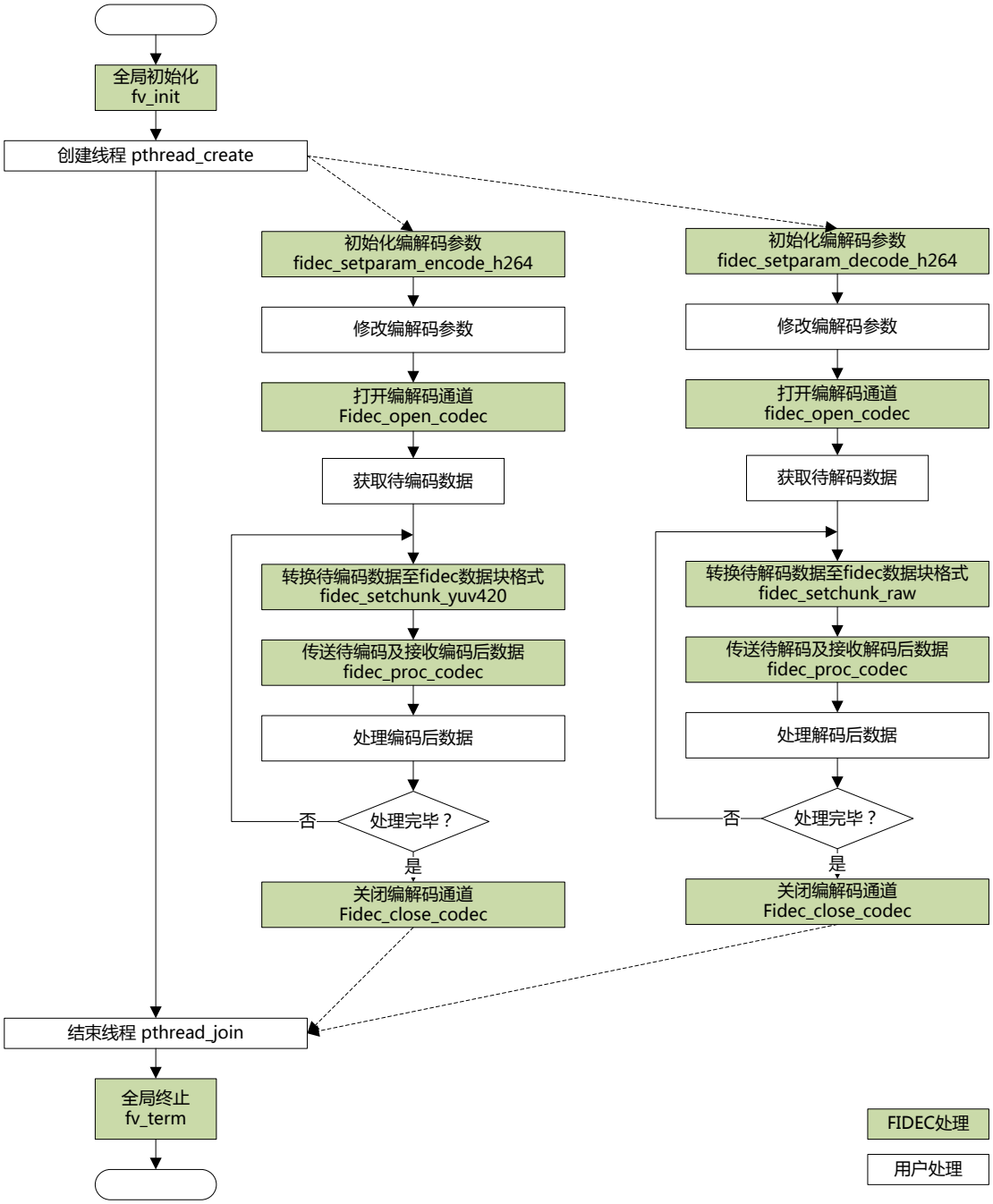
7.1 单通道编解码应用

下图为单通道 H264 解码处理的流程，可扩展为多进程多通道应用。



7.2 多通道编解码应用

下图为多线程模型下 H264 编解码处理的流程， 其中 2 个线程占用 2 个通道同时进行 H264 的编码和解码处理。



7.3 参考源码

请参考 SDK 中的 simple_encoder 示例程序。

8 错误代码一览

错误代码定义	值	说　　明
RET_SUCCESS	0	无错误
RET_TIMEOUT	99	超时
ASE_INVALID_CH	-1001	通道号异常
ASE_INVALID_PARA	-1002	输入参数异常
ASE_UNKNOWN_ASVC	-1003	未知的编解码引擎
ASE_BAD_SEQ	-1004	操作顺序异常
ASE_NOT_SUPPORT	-1005	未支持的功能
ASE_INVALID_PLTYPE	-1006	保留
ASE_IO_ERROR	-1007	IO 异常
ASE_NOT_ENOUGH_BUF	-1008	缓冲区长度过小
ASE_WRONG_VER	-1009	FIDEC Server 与 FIDEC Library 版本不匹配
ASE_ML_FRAMELEN	-1101	YUV 图像帧长度与尺寸不符