ESWIN

ENNP 开发者手册

EIC7x 系列 AI 数字 Soc

Engineering Draft / Rev0.94

2025-01-24

声明

产品安装中可能包含开源软件和/或自由软件的许可声明。

修订记录：

文档版本	日期	说明
Rev 0.6	2024/02/29	初始版本
Rev 0.92	2024/10/23	npu runtime 增加复合模型相关接口
Rev 0.93	2025/1/21	npu runtime 增加从内存加载模型接口
Rev 0.94	2025/1/24	第 3 章 AcceleratorKit 修改了 ES_AK_DSP_WarpAffine 接口参数和文档说明

目 录

1. DSP RUNTIME 2
1.1 概述 2
1.2 功能描述 2
1.3 API 参考 2
1.4 数据类型和数据结构 17
1.5 错误码 20
1.6 调试信息 22
1.7 使用注意事项 25
2. NPU RUNTIME 25
2.1 概述 25
2.2 功能描述 26
2.3 API 参考 28
2.4 数据类型和数据结构 47
2.5 错误码 52
2.6 调试信息 54
2.7 使用注意事项 54
3. ACCELERATORKIT 54
3.1 概述 54
3.2 功能描述 55
3.3 API 参考 55
3.4 数据类型和数据结构 69
3.5 错误码 74
3.6 调试信息 75
3.7 使用注意事项 75

图目录

图 2-1 NPU Runtime 软件栈 25
图 2-2 Device、Context、Stream、Task 关系 27

1. DSP Runtime

1.1 概述

ENNP（ESWIN Neural NetWork Processing）平台是奕斯伟媒体处理芯片智能计算异构加速平台，包含了对 NPU、DSP、HAE 以及 GPU 的开发工具及接口。DSP（Digital Signal Process）是 ENNP 平台下的可编程硬件加速模块。用户基于 DSP 开发智能分析方案可以加速智能分析，降低 CPU 占用率。

1.2 功能描述

1.2.1 重要概念

- 算子句柄（Operator Handle）：用户在加载 DSP 算子时，系统会为每个算子分配一个 Handle，用于标识不同的算子。
- 算子任务描述信息（Task Description）：用户在提交算子任务时需要填充的必要信息，参考 ES_DSP_TASK_S 定义。
- 算子任务句柄（Task Handle）：用户在调用 DSP 处理算子任务时，系统会为每个任务分配一个 Handle，用于标识不同的算子任务。
- 算子任务查询（Query）：用户使用类型一（详见 1.3.1）接口提交异步任务后，根据系统返回的算子任务句柄（handle），调用 ES_DSP_QueryTask 可以查询对应算子任务是否完成。
- 算子任务查询（Process Report）：用户使用类型二（详见 1.3.1）接口提交异步任务后，需调用 ES_DSP_LL_ProcessReport 处理异步任务，该接口会调用已完成任务的回调函数。

1.2.2 模块参数

- fw_timeout：DSP 任务超时时间，单位为：秒。
  用户可以依据实际使用场景配置合理的最大超时时间。
  使用方法，DSP 驱动加载后通过以下命令配置：

```
echo 10 > /sys/module/eic7700_dsp/parameters/fw_timeout
```

1.3 API 参考

1.3.1 API 类型

DSP 提供两种不同类型的 API，其中类型一接口对底层接口进行了部分隐藏，接口使用

较为简单；类型二接口暴露了更多底层功能，适用于专业用户。用户不能混合使用两种类型的接口，以防止计算过程出错导致该进程内部逻辑出现问题。具体如下：

1.3.1.1 类型一

- ES_DSP_GetVersion：获取 DSP Runtime 接口数字信息版本
- ES_DSP_GetVersionString：获取 DSP Runtime 接口字符串版本
- ES_DSP_GetCapability：获取 DSP 能力信息
- ES_DSP_SetLogLevel：设置 DSP 打印日志等级
- ES_DSP_Open：打开 DSP 设备
- ES_DSP_Close：关闭 DSP 设备
- ES_DSP_LoadOperator：加载 DSP 算子
- ES_DSP_UnloadOperator：卸载 DSP 算子
- ES_DSP_SubmitTask：同步执行 DSP 算子任务
- ES_DSP_SubmitTaskAsync：异步执行 DSP 算子任务
- ES_DSP_QueryTask：查询 DSP 算子任务是否完成

1.3.1.2 类型二

- ES_DSP_LL_GetVersion：获取 DSP Runtime 接口数字信息版本
- ES_DSP_LL_GetVersionString：获取 DSP Runtime 接口字符串版本
- ES_DSP_LL_GetCapability：获取 DSP 能力信息
- ES_DSP_LL_SetLogLevel：设置 DSP 打印日志等级
- ES_DSP_LL_Open：打开 DSP 设备，获取设备 fd
- ES_DSP_LL_Close：根据设备 fd 关闭 DSP 设备
- ES_DSP_LL_LoadOperator：加载算子，获取算子句柄
- ES_DSP_LL_UnloadOperator：根据算子句柄卸载算子
- ES_DSP_LL_PrepareDMABuffer：根据算子数据申请 DMA Buffer
- ES_DSP_LL_UnprepareDMABuffer：释放申请的 DMA Buffer
- ES_DSP_LL_SubmitTask：同步方式执行算子任务
- ES_DSP_LL_SubmitTaskAsync：异步方式执行算子任务
- ES_DSP_LL_ProcessReport：执行异步算子任务提交时注册的回调函数

1.3.2 Runtime API

1.3.2.1 ES_DSP_GetVersion

【函数体】

ES_S32 ES_DSP_GetVersion(ES_U64 *version)

【描述】

获取 DSP 数字版本信息。

【参数】

参数	描述	输入/输出
version	DSP 版本号	输出，参见 ES_SDK_VERSION_U

【返回值】

返回值	描述
0	成功
非 0	失败，参见 1.5 错误码

1.3.2.2 ES_DSP_GetVersionString

【函数体】

ES_S32 ES_DSP_GetVersionString(

ES_CHAR *version,
ES_U32 maxSize)

【描述】

获取 DSP 字符串版本信息。

【参数】

参数	描述	输入/输出
version	DSP 版本号	输出
maxSize	DSP 版本字符串最大长度	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见 1.5 错误码

1.3.2.3 ES_DSP_GetCapability

【函数体】

ES_S32 ES_DSP_GetCapability(ES_DSP_Capability_S *capability)

【描述】

获取 DSP 能力信息。

【参数】

参数	描述	输入/输出
dspId	DSP 能力信息	输出，参见 ES_DSP_Capability_S

【返回值】

返回值	描述
0	成功
非 0	失败，参见 1.5 错误码

1.3.2.4 ES_DSP_SetLogLevel

【函数体】

ES_S32 ES_DSP_SetLogLevel(ES_U32 level)

【描述】

设置 DSP 运行时的打印日志级别。

【参数】

参数	描述	输入/输出
level	DSP 打印日志等级	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见 1.5 错误码

1.3.2.5 ES_DSP_Open

【函数体】

ES_S32 ES_DSP_Open(

ES_DSP_ID_E dspId,
ES_DSP_LOAD_POLICY_E opLoadPolicy)

【描述】

根据 DSP ID 号打开 DSP 设备。

【参数】

参数	描述	输入/输出
dspId	DSP ID 序号	输入，参见 ES_DSP_ID_E
opLoadPolicy	DSP 算子加载策略	输入，参见 ES_DSP_LOAD_POLICY_E

【返回值】

返回值	描述
0	成功
非 0	失败，参见 1.5 错误码

1.3.2.6 ES_DSP_Close

【函数体】

ES_S32 ES_DSP_Close(ES_DSP_ID_E dspId)

【描述】

根据 DSP ID 号关闭 DSP 设备。

【参数】

参数	描述	输入/输出
dspId	DSP ID 序号	输入，参见 ES_DSP_ID_E

【返回值】

返回值	描述
0	成功
非 0	失败，参见 1.5 错误码

1.3.2.7 ES_DSP_LoadOperator

【函数体】

ES_S32 ES_DSP_LoadOperator(

ES_DSP_ID_E dspId,
const ES_CHAR* operatorName,
const ES_CHAR* operatorLibDir,
ES_DSP_HANDLE* operatorHandle)

【描述】

加载指定名称和库路径的算子到指定的 DSP 设备，并返回算子句柄。

【参数】

参数	描述	输入/输出
dspId	DSP ID 序号	输入，参见 ES_DSP_ID_E
operatorName	DSP 算子名	输入
operatorLibDir	DSP 算子库目录	输入
operatorHandle	DSP 算子句柄	输出

【返回值】

返回值	描述
0	成功
非 0	失败，参见 1.5 错误码

1.3.2.8 ES_DSP_UnloadOperator

【函数体】

ES_S32 ES_DSP_UnloadOperator(

ES_DSP_ID_E dspId,
const ES_DSP_HANDLE operatorHandle)

【描述】

在指定的 DSP 设备通过已加载的算子句柄卸载该算子。

【参数】

参数	描述	输入/输出
dspId	DSP ID 序号	输入
operatorHandle	DSP 算子句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见 1.5 错误码

1.3.2.9 ES_DSP_SubmitTask

【函数体】

```
ES_S32 ES_DSP_SubmitTask(
                ES_DSP_ID_E dspId,
                ES_DSP_TASK_S* operatorTask)
```

【描述】

同步提交算子任务到指定的 DSP 设备，接口等待 DSP 完成本次计算，并返回即代表任务完成。

【参数】

参数	描述	输入/输出
dspId	DSP ID 序号	输入
operatorTask	DSP 算子任务消息体	输入，参见 ES_DSP_TASK_S

【返回值】

返回值	描述
0	成功
非 0	失败，参见 1.5 错误码

1.3.2.10 ES_DSP_SubmitTaskAsync

【函数体】

```
ES_S32 ES_DSP_SubmitTaskAsync(
                ES_DSP_ID_E dspId,
                ES_DSP_TASK_S* operatorTask,
                ES_DSP_HANDLE* taskHandle)
```

【描述】

异步提交算子任务到指定的 DSP 设备，并返回算子任务句柄。需要使用 ES_DSP_QueryTask 查询任务是否完成。

【参数】

参数	描述	输入/输出
dspId	DSP ID 序号	输入
operatorTask	DSP 算子任务消息体	输入，参见 ES_DSP_TASK_S
taskHandle	DSP 算子任务句柄	输出

【返回值】

返回值	描述
0	成功
非 0	失败，参见 1.5 错误码

1.3.2.11 ES_DSP_QueryTask

【函数体】

```
ES_S32 ES_DSP_QueryTask(
                ES_DSP_ID_E dspId,
                ES_DSP_HANDLE taskHandle,
                ES_BOOL block,
                ES_BOOL* taskFinish)
```

【描述】

根据算子任务句柄查询该算子在指定的 DSP 设备是否完成运算。

【参数】

参数	描述	输入/输出
dspId	DSP ID 序号	输入
taskHandle	DSP 算子任务句柄	输入
block	是否阻塞查询	输入
taskFinish	DSP 算子任务完成状态指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败，参见 1.5 错误码

1.3.3 Runtime Low-level API

1.3.3.1 ES_DSP_LL_GetVersion

【函数体】

ES_S32 ES_DSP_LL_GetVersion(ES_U64 *version)

【描述】

获取 DSP 数字版本信息。

【参数】

参数	描述	输入/输出
version	DSP 版本号	输出，参见 ES_SDK_VERSION_U

【返回值】

返回值	描述
0	成功
非 0	失败，参见 1.5 错误码

1.3.3.2 ES_DSP_LL_GetVersionString

【函数体】

ES_S32 ES_DSP_LL_GetVersionString(

ES_CHAR* version,
ES_U32 maxSize)

【描述】

获取 DSP 字符串版本信息。

【参数】

参数	描述	输入/输出
version	DSP 版本号	输出
maxSize	DSP 版本字符串最大长度	输入

【返回值】

返回值	描述
0	成功

非 0	失败，参见 1.5 错误码

1.3.3.3 ES_DSP_LL_GetCapability

【函数体】

ES_S32 ES_DSP_LL_GetCapability(ES_DSP_Capability_S *capability)

【描述】

获取 DSP 能力信息。

【参数】

参数	描述	输入/输出
dspId	DSP 能力信息	输出，参见 ES_DSP_Capability_S

【返回值】

返回值	描述
0	成功
非 0	失败，参见 1.5 错误码

1.3.3.4 ES_DSP_LL_SetLogLevel

【函数体】

ES_S32 ES_DSP_LL_SetLogLevel(ES_U32 level)

【描述】

设置 DSP 运行时的打印日志级别。

【参数】

参数	描述	输入/输出
level	DSP 打印日志等级	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见 1.5 错误码

1.3.3.5 ES_DSP_LL_Open

【函数体】

```
ES_S32 ES_DSP_LL_Open(
                ES_DSP_ID_E dspId,
                ES_S32 *dspFd)
```

【描述】

根据 DSP ID 号打开 DSP 设备。

【参数】

参数	描述	输入/输出
dspId	DSP ID 序号	输入，参见 ES_DSP_ID_E
dspFd	DSP 设备描述符	输出

【返回值】

返回值	描述
0	成功
非 0	失败，参见 1.5 错误码

1.3.3.6 ES_DSP_LL_Close

【函数体】

```
ES_S32 ES_DSP_LL_Close(ES_S32 dspFd)
```

【描述】

根据 DSP ID 号关闭 DSP 设备。

【参数】

参数	描述	输入/输出
dspFd	DSP 设备描述符	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见 1.5 错误码

1.3.3.7 ES_DSP_LL_LoadOperator

【函数体】

```
ES_S32 ES_DSP_LL_LoadOperator(
        ES_S32 dspFd,
        const ES_CHAR *operatorName,
        const ES_CHAR *operatorLibDir,
        ES_DSP_HANDLE *operatorHandle)
```

【描述】

加载指定名称和库路径的算子到指定的 DSP 设备，并返回算子句柄。

【参数】

参数	描述	输入/输出
dspFd	DSP 设备描述符	输入
operatorName	DSP 算子名	输入
operatorLibDir	DSP 算子库目录	输入
operatorHandle	DSP 算子句柄	输出

【返回值】

返回值	描述
0	成功
非 0	失败，参见 1.5 错误码

1.3.3.8 ES_DSP_LL_UnloadOperator

【函数体】

```
ES_S32 ES_DSP_LL_UnloadOperator(
        ES_S32 dspFd,
        const ES_DSP_HANDLE operatorHandle)
```

【描述】

在指定的 DSP 设备通过已加载的算子句柄卸载该算子。

【参数】

参数	描述	输入/输出
dspFd	DSP 设备描述符	输入

参数	描述	输入/输出
operatorHandle	DSP 算子句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见 1.5 错误码

1.3.3.9 ES_DSP_LL_PrepareDMABuffer

【函数体】

```
ES_S32 ES_DSP_LL_PrepareDMABuffer(
        ES_S32 dspFd,
        ES_DEV_BUF_S buffer)
```

【描述】

为算子任务所需的 DMA Buffer 提前建立在该 DSP 设备上的 SMMU 映射，这样可以加速后续的数据执行过程。

【参数】

参数	描述	输入/输出
dspFd	DSP 设备描述符	输入
buffer	DMA Buffer	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见 1.5 错误码

1.3.3.10 ES_DSP_LL_UnprepareDMABuffer

【函数体】

```
ES_S32 ES_DSP_LL_UnprepareDMABuffer(
        ES_S32 dspFd,
        ES_U64 fd)
```

【描述】

根据申请的 DMA Buffer 对应的 fd 释放在该 DSP 上的设备 SMMU 映射。需要与 ES_DSP_LL_PrepareDMABuffer 成对使用，并且在不需要对该 Buffer 再次执行该 DSP 设备上任务时调用。

【参数】

参数	描述	输入/输出
dspFd	DSP 设备描述符	输入
fd	DMA Buffer 描述符	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见 1.5 错误码

1.3.3.11 ES_DSP_LL_SubmitTask

【函数体】

```
ES_S32 ES_DSP_LL_SubmitTask(
        ES_S32 dspFd,
        ES_DSP_TASK_S *operatorTask)
```

【描述】

提交同步算子任务到指定的 DSP 设备。

【参数】

参数	描述	输入/输出
dspFd	DSP 设备描述符	输入
operatorTask	DSP 算子任务消息体	输入，参见 ES_DSP_TASK_S

【返回值】

返回值	描述
0	成功
非 0	失败，参见 1.5 错误码

1.3.3.12 ES_DSP_LL_SubmitTaskAsync

【函数体】

```
ES_S32 ES_DSP_LL_SubmitTaskAsync(
        ES_S32 dspFd,
        ES_DSP_TASK_S *operatorTask)
```

【描述】

提交异步算子任务到指定的 DSP 设备。异步任务需要使用 ES_DSP_LL_ProcessReport 查询并等待执行任务完成，并在完成时调用回调函数。

【参数】

参数	描述	输入/输出
dspFd	DSP 设备描述符	输入
operatorTask	DSP 算子任务消息体	输入，参见 ES_DSP_TASK_S

【返回值】

返回值	描述
0	成功
非 0	失败，参见 1.5 错误码

1.3.3.13 ES_DSP_LL_ProcessReport

【函数体】

```
ES_S32 ES_DSP_LL_ProcessReport(
        ES_S32 dspFd,
        ES_S32 timeout)
```

【描述】

执行异步算子任务提交时注册的回调函数。

【参数】

参数	描述	输入/输出
dspFd	DSP 设备描述符	输入
timeout	单位：ms	输入

参数	描述	输入/输出
	没有任务完成时，等待任务完成的最大时间，-1 则阻塞等待直到有任务完成	

【返回值】

返回值	描述
0	成功
非 0	失败，参见 1.5 错误码

1.4 数据类型和数据结构

DSP 相关数据类型、数据结构定义如下：

- ES_DSP_Capability_S：定义 DSP 能力信息数据结构
- ES_DSP_ID_E：定义 DSP ID 枚举值
- ES_DSP_PRI_E：定义 DSP 算子任务优先级枚举值
- ES_DSP_HANDLE：定义 DSP 算子任务句柄
- ES_DSP_TASK_S ：定义 DSP 算子任务消息格式
- ES_DSP_TASK_CALLBACK：定义 DSP 算子任务回调函数

1.4.1 ES_DSP_Capability_S

【说明】

定义 DSP 能力信息。

【定义】

```
typedef struct DSP_Capability_S {
    ES_U64 reserved;
} ES_DSP_Capability_S
```

【成员】

参数	描述
reserved	预留字段

1.4.2 ES_DSP_ID_E

【说明】

定义 DSP ID。

【定义】

```
typedef enum DSP_ID_E {
    ES_DSP_ID_0 = 0x0,
    ES_DSP_ID_1 = 0x1,
    ES_DSP_ID_2 = 0x2,
    ES_DSP_ID_3 = 0x3,
    ES_DSP_ID_4 = 0x4,
    ES_DSP_ID_5 = 0x5,
    ES_DSP_ID_6 = 0x6
    ES_DSP_ID_7 = 0x7,
    ES_DSP_ID_BUTT
} ES_DSP_ID_E
```

【成员】

参数	描述
ES_DSP_ID_0	DSP ID 0
ES_DSP_ID_1	DSP ID 1
ES_DSP_ID_2	DSP ID 2
ES_DSP_ID_3	DSP ID 3
ES_DSP_ID_4	DSP ID 4
ES_DSP_ID_5	DSP ID 5
ES_DSP_ID_6	DSP ID 6
ES_DSP_ID_7	DSP ID 7
ES_DSP_ID_BUTT	DSP ID 枚举个数

1.4.3 ES_DSP_PRI_E

【说明】

定义 DSP 任务优先级。同一个 DSP 设备会根据优先级调度排队的任务。

【定义】

```
typedef enum DSP_PRI_E {
    ES_DSP_PRI_0 = 0x0,
```

```
    ES_DSP_PRI_1 = 0x1,
    ES_DSP_PRI_2 = 0x2,
    ES_DSP_PRI_3 = 0x3,
    ES_DSP_PRI_BUTT
} ES_DSP_PRI_E
```

【成员】

参数	描述
ES_DSP_PRI_0	优先级 0 最高
ES_DSP_PRI_1	优先级 1
ES_DSP_PRI_2	优先级 2
ES_DSP_PRI_3	优先级 3
ES_DSP_PRI_BUTT	优先级枚举个数

1.4.4 ES_DSP_HANDLE

【说明】

定义 DSP 任务句柄。

【定义】

```
typedef ES_U64 ES_DSP_HANDLE
```

1.4.5 ES_DSP_TASK_CALLBACK

【说明】

定义 DSP 算子任务回调函数。

【定义】

```
typedef void (*ES_DSP_TASK_CALLBACK)(void *arg， ES_S32 state)
```

1.4.6 ES_DSP_TASK_S

【说明】

定义 DSP 算子任务描述信息。

【定义】

```
typedef struct DSP_TASK_S {
```

```
    ES_DSP_HANDLE operatorHandle;
    ES_DEV_BUF_S dspBuffers[BUFFER_CNT_MAXSIZE];
    ES_U32 bufferCntCfg;
    ES_U32 bufferCntInput;
    ES_U32 bufferCntOutput;
    ES_DSP_PRI_E priority;
    ES_BOOL syncCache;
    ES_BOOL pollMode;
    ES_DSP_HANDLE taskHandle;
    ES_DSP_TASK_CALLBACK callback;
    ES_VOID *cbArg;
} ES_DSP_TASK_S
```

【成员】

参数	描述
operatorHandle	DSP 已加载的算子句柄
dspBuffers	DSP 算子将使用的设备缓冲区
bufferCntCfg	算子配置信息的缓冲区总数
bufferCntInput	算子输入信息的缓冲区总数
bufferCntOutput	算子输出信息的缓冲区总数
priority	算子任务优先级
syncCache	驱动程序负责主机端缓存同步
pollMode	DSP 使用 polling 模式
taskHandle	算子任务提交后返回的句柄
callback	任务完成后需要调用的回调函数
cbArg	回调函数的参数

1.5 错误码

【说明】

定义 DSP API 错误码。

【定义】

```
typedef enum RET_DSP_E {
    ES_DSP_SUCCESS = 0x0,
    ES_DSP_ERROR_INVALID_DEVID = 0x2030001,
    ...
    ES_DSP_ERROR_READ_FILE
} ES_S32
```

【成员】

参数	值	描述
ES_DSP_SUCCESS	0x0	操作执行成功
ES_DSP_ERROR_INVALID_DEVID	0x2030001	设备 ID 超出合法范围
ES_DSP_ERROR_INVALID_CHNID	0x2030002	通道组号错误或无效区域句柄
ES_DSP_ERROR_ILLEGAL_PARAM	0x2030003	参数超出合法范围
ES_DSP_ERROR_EXIST	0x2030004	重复创建已存在的设备、通道或资源
ES_DSP_ERROR_UNEXIST	0x2030005	试图使用或者销毁不存在的设备、通道或者资源
ES_DSP_ERROR_NULL_PTR	0x2030006	函数参数中有空指针
ES_DSP_ERROR_NOT_CONFIG	0x2030007	模块没有配置
ES_DSP_ERROR_NOT_SUPPORT	0x2030008	不支持的参数或者功能
ES_DSP_ERROR_NOT_PERM	0x2030009	该操作不允许，如试图修改静态配置参数
ES_DSP_ERROR_NOMEM	0x203000C	分配内存失败，如系统内存不足
ES_DSP_ERROR_NOBUF	0x203000D	分配缓存失败，如申请的图像缓冲区太大
ES_DSP_ERROR_BUF_EMPTY	0x203000E	缓冲区中无图像
ES_DSP_ERROR_BUF_FULL	0x203000F	缓冲区中图像满
ES_DSP_ERROR_NOTREADY	0x2030010	系统没有初始化或没有加载相应模块
ES_DSP_ERROR_BADADDR	0x2030011	地址非法

参数	值	描述
ES_DSP_ERROR_BUSY	0x2030012	系统忙
ES_DSP_ERROR_SYS_TIMEOUT	0x2030040	系统超时
ES_DSP_ERROR_QUERY_TIMEOUT	0x2030041	Query 查询超时
ES_DSP_ERROR_OPEN_FILE	0x2030042	打开文件失败
ES_DSP_ERROR_READ_FILE	0x2030043	读文件失败

1.6 调试信息

1.6.1 Proc 调试信息

调试信息采用了 Linux 下的 proc 文件系统，可实时反映当前系统的运行状态，所记录的信息可供问题定位及分析时使用。

- 文件目录，/proc/esdsp/info
- 信息查看方法：在控制台上可以使用 cat 命令查看信息，例如：cat /proc/esdsp/info

也可以使用其他常用的文件操作命令，例如：cp /proc/esdsp/info ./

将文件拷贝到当前目录，在应用程序中可以将上述文件当作普通只读文件进行读操作，例如 fopen、fread 等。

1.6.2 Proc 信息说明

- 调试信息

```
# cat /proc/esdsp/info
----------------------------------DSP PARAM INFO-------------------------------
 DieId   CoreId   Enable   CmdTOut(s)
 0         0       Yes      1000000
 0         1       NO          0
 0         2       NO       1000000
 1         0       NO          0
 1         1       NO          0
 1         2       NO          0

--------------------------------DSP RUNTIME INFO-------------------------------
 DieId   CoreId   TotalIntCnt   FinishedTaskCnt   FailedTaskCnt   PendingTaskCnt   LTaskRunTm
 0         0         32              31               0               0        67780000
 0         1         0               0                0               0        0
 0         2         0               0                0               0        0
 1         0         0               0                0               0        0
 1         1         0               0                0               0        0

-----------------------DSP HW PERF INFO----------------------------
DieId CoreId TaskName StartTm   PrepSTm   PrepETm   EvalSTm   EvalETm   IPCSTm   EndTm
 0       0   argmax_f16f32 9081856 9081876 9081956 9081976 9091856 9092856 9181856
 0       1      NULL          0         0         0         0         0        0        0
 0       2      NULL          0         0         0         0         0        0        0
 0       3      NULL          0         0         0         0         0        0        0
 1       0      NULL          0         0         0         0         0        0        0
 1       1      NULL          0         0         0         0         0        0        0
 1       2      NULL          0         0         0         0         0        0        0
 1       3      NULL          0         0         0         0         0        0        0

-------------------------------DSP INVOKE INFO-------------------------------
 DieId   CoreId   Pri   TaskName   TaskHnd   TaskStat   TaskRunTm
 0         0       0     softmax    7fe069c0    eval       78900000
 0         1       0     NULL         0         NULL       0
 0         2       0     NULL         0         NULL       0
 1         0       0     NULL         0         NULL       0
 1         1       0     NULL         0         NULL       0
 1         2       0     NULL         0         NULL       0
```

- 调试信息分析

记录当前 DSP 工作状态资源信息，主要包括 DSP 队列状态信息，任务状态信息，运行时状态信息和调用信息。

- 参数说明

模块	参数	描述
DSP PARAM INFO	DieId	Die ID
	CoreId	DSP 设备 ID
	Enable	DSP 使能状态：0 未使能，1 使能
	TaskTmOut	任务执行超时时间。单位：s

模块	参数	描述
DSP RUNTIME INFO	DieId	Die ID
	CoreId	DSP 设备 ID
	TotalIntCnt	DSP 产生中断的总次数
	LTaskRunTm	最后一个任务执行时间，单位：us
	FinishedTaskCnt	成功任务总数
	FailedTaskCnt	失败任务总数
	PendingTaskCnt	待执行任务列表
DSP HW PERF INFO	DieId	Die ID
	CoreId	DSP 设备 ID
	Pri	任务优先级
	TaskName	正在运行的任务名称
	StartTm	算子开始时间，单位：ns
	PrepSTm	任务 prepare 开始时间，单位：ns
	PrepETm	任务 prepare 结束时间，单位：ns
	EvalSTm	任务 eval 开始时间，单位：ns
	EvalETm	任务 eval 开始时间，单位：ns
	IPCSTm	DSP 通知驱动时间，单位：ns
	EndTm	算子结束时间，单位：ns
DSP INVOKE INFO	DieId	Die ID
	CoreId	DSP 设备 ID
	Pri	任务优先级
	TaskName	正在运行的任务名称
	TaskHnd	正在运行的任务 handle
	TaskStat	正在运行任务的状态，prepare 或者 eval
	TaskRunTm	任务已运行时间，单位：us

1.7 使用注意事项

对于如下操作需要成对出现：

ES_DSP_Open/ ES_DSP_Close

ES_DSP_LoadOperator/ ES_DSP_UnloadOperator

ES_DSP_LL_PrepareDMABuffer/ ES_DSP_LL_UnprepareDMABuffer

否则会出现不可以预期的错误。

2. NPU Runtime

2.1 概述

ENNP（ESWIN Neural NetWork Processing）平台是奕斯伟媒体处理芯片智能计算异构加速平台，包含了对 NPU、DSP、HAE 以及 GPU 的开发工具及接口。NPU Runtime 是 ENNP 提供的一套加载 NN 模型并执行推理任务的运行时系统和接口，可以直接加载基于 ENNP 工具量化编译的离线模型，用于实现目标识别、图像分类等功能。用户基于 NPU Runtime 开发智能分析方案，由编译器自动分析并优化执行过程，并自动生成优化好的离线模型，最大化复用 NPU、DSP、HAE 以及 GPU 等硬件，提升硬件利用率并优化系统功耗。

NPU Runtime 软件栈如下图所示：

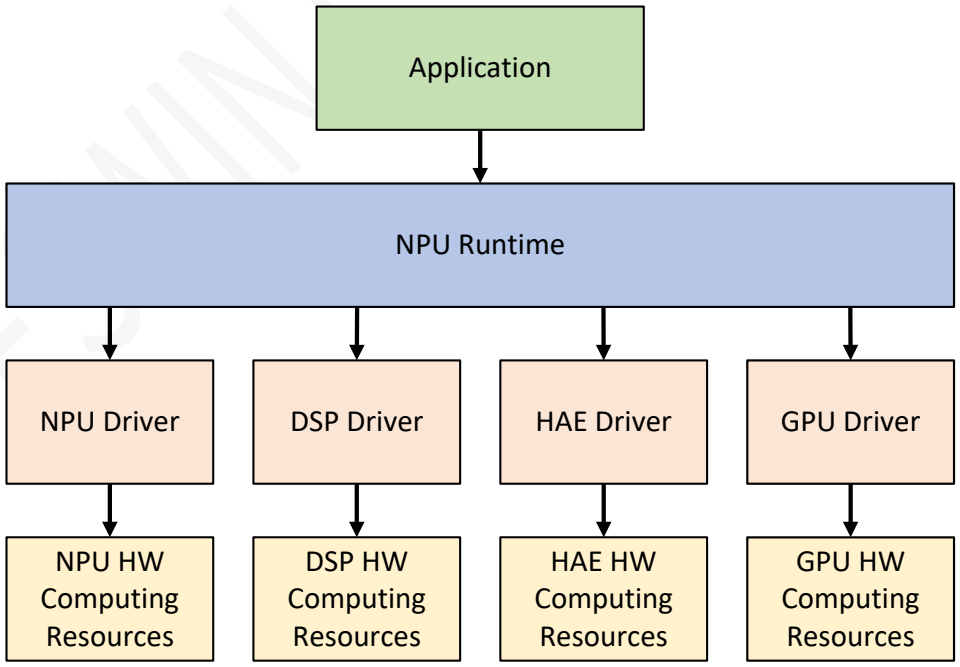

图 2-1 NPU Runtime 软件栈

2.2 功能描述

2.2.1 功能描述

EIC7700 包含了 NPU、DSP、HAE 以及 GPU 等硬件。根据用户的配置及模型的实现，复合模型在运行时会分别按需调用相应的硬件模块进行对应的计算。不同算子之间的调度是由 Runtime 根据模型所描述的依赖关系确定的，支持实现多任务、多进程提交。

2.2.2 概念介绍

NPU Runtime 提供 Device 管理、Context 管理、Stream 管理、内存管理、模型加载与执行等 C 语言 API 库。

参数	描述
同步/异步	同步、异步是站在调用者和执行者的角度，在当前场景下，若在板端环境调用接口后不等待 Device 执行完成再返回，则表示板端环境的调度是异步的； 若在板端环境调用接口后需等待 Device 执行完成再返回，则表示板端环境的调度是同步的
进程/线程	本文中提及的进程、线程，若无特别注明，则表示板端环境上的进程、线程
Device	Device 表示板端环境上的一个 NPU（Neural-Network ProcessingUnit）硬件单元
Context	Context 描述了 NPU 运行的一个配置上下文，同时它作为一个容器，管理了在对应 Context 上创建对象的生命周期，用于管理异步提交的任务。一个 Context 可以关联到一个进程的多个线程，且提交的任务都会按照先后顺序进行调度，不同 Context 的 Stream 是完全隔离的。在进程或线程中调用 ES_NPU_CreateContext 接口创建一个 Context
Stream	Stream 用于维护异步操作，任务完成后等待用户取回的队列。在进程或线程中调用 ES_NPU_CreateStream 接口创建一个 Stream

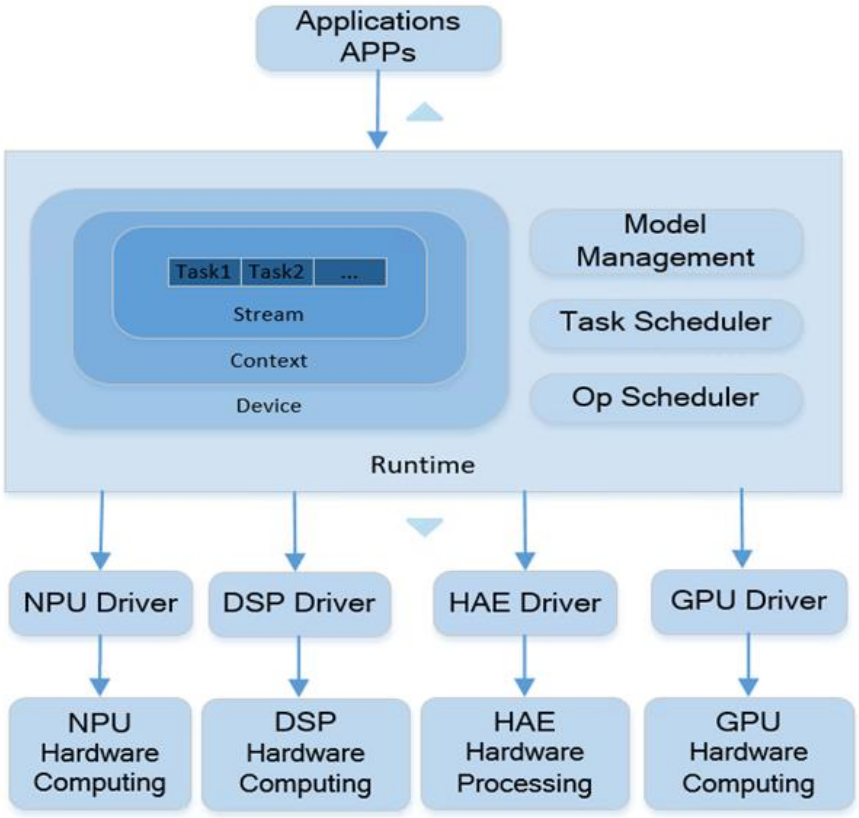

图 2-2 Device、Context、Stream、Task 关系

- 用户使用 ES_NPU_SetDevice 接口指定用于运算的 Device，当调用 ES_NPU_ReleaseDevice 后，对应的 Device 被释放。Device 被释放后，其对应的资源在本进程不可用。
- 在使用 NPU 异步提交任务接口时，Context 用于指定任务使用的等待队列。一个 NPU 设备可以包含多个 Context，但是一个 Context 只能属于一个特定的设备。Context 生命周期始于 ES_NPU_CreateContext，结束于 ES_NPU_DestroyContext。Context 与用户线程绑定，一个用户线程对应于一个 Context，可以调用 ES_NPU_SetCurrentContext 接口来切换或者绑定 Context。一个 Context 可以与多个线程进行绑定。
- 在使用 NPU 异步提交任务接口时，Stream 用于指定任务完成后的完成等待队列。Stream 生命周期始于 ES_NPU_CreateStream，终结于 ES_NPU_DestroyStream。一个 Context 可以对应多个 Stream。
- 如果使用 NPU 同步提交任务接口，则不用申请创建 Context 和 Stream。
- Task 是 Device 上真正的执行体。在提交任务时，关联到特定的 Stream。

2.3 API 参考

2.3.1 ES_NPU_GetVersion

【函数体】

ES_S32 ES_NPU_GetVersion(ES_U64 *version)

【描述】

获取 NPU runtime 软件栈版本信息。

【参数】

参数	描述	输入/输出
version	Runtime 版本号	输出，参见 ES_SDK_VERSION_U

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

2.3.2 ES_NPU_GetVersionString

【函数体】

ES_S32 ES_NPU_GetVersionString(

ES_CHAR *version,
ES_U32 maxSize)

【描述】

获取字符格式的 NPU runtime 软件栈版本信息。

【参数】

参数	描述	输入/输出
version	Runtime 版本号字符串	输出
maxSize	Runtime 版本号字符串最大大小	输入

【返回值】

返回值	描述

0	成功
非 0	失败，参见 2.5 错误码

2.3.3 ES_NPU_GetCapability

【函数体】

ES_S32 ES_NPU_GetCapability(ES_NPU_Capability_S *capability)

【描述】

获取 NPU 能力信息。

【参数】

参数	描述	输入/输出
capability	存储 NPU 能力信息的结构体	输出

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

2.3.4 ES_NPU_SetLogLevel

【函数体】

ES_S32 ES_NPU_SetLogLevel(ES_U32 level)

【描述】

设置 NPU 运行时的打印日志级别。

【参数】

参数	描述	输入/输出
level	NPU 打印日志等级	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

2.3.5 ES_NPU_GetDeviceProperties

【函数体】

```
ES_S32 ES_NPU_GetDeviceProperties(
        ES_S32 deviceId,
        NPU_DEVICE_PROP_S*  prop)
```

【描述】

查询指定 deviceId 的设备属性信息。

【参数】

参数	描述	输入/输出
deviceId	要查询设备的 deviceId	输入
prop	设备的属性信息	输出

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

2.3.6 ES_NPU_GetNumDevices

【函数体】

```
ES_S32 ES_NPU_GetNumDevices(ES_U16 *deviceNum)
```

【描述】

获取当前环境中可用的 device 数量。

【参数】

参数	描述	输入/输出
deviceNum	获得的 device 数量	输出

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

2.3.7 ES_NPU_SetDevice

【函数体】

ES_S32 ES_NPU_SetDevice(ES_U16 deviceId)

【描述】

设置用于运算的 device。

【参数】

参数	描述	输入/输出
deviceId	当前 deviceId	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

【注意】

deviceId 应小于通过 ES_NPU_GetNumDevices 获取得到的 deviceNum。

2.3.8 ES_NPU_ReleaseDevice

【函数体】

ES_S32 ES_NPU_ReleaseDevice(ES_U16 deviceId)

【描述】

释放当前用于运算的 device。

【参数】

参数	描述	输入/输出
deviceId	当前 deviceId	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

【注意】

传入参数应为 ES_NPU_SetDevice 接口设置的 deviceId。

2.3.9 ES_NPU_GetDevice

【函数体】

ES_S32 ES_NPU_GetDevice(ES_U16 *deviceId)

【描述】

获取当前正在使用的 deviceId。

【参数】

参数	描述	输入/输出
deviceId	当前 deviceId	输出

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

2.3.10 ES_NPU_CreateContext

【函数体】

ES_S32 ES_NPU_CreateContext(

npu_context* context,
ES_U16 deviceId)

【描述】

创建运行时上下文环境。

【参数】

参数	描述	输入/输出
context	返回的 context 指针	输出
deviceId	输入当前 context 要使用的 deviceId	输入

【返回值】

返回值	描述

0	成功
非 0	失败，参见 2.5 错误码

2.3.11 ES_NPU_SetCurrentContext

【函数体】

ES_S32 ES_NPU_SetCurrentContext(npu_context context)

【描述】

把 context 设置给当前线程。

【参数】

参数	描述	输入/输出
context	要设置的 context	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

2.3.12 ES_NPU_GetCurrentContext

【函数体】

ES_S32 ES_NPU_GetCurrentContext(npu_context *context)

【描述】

获取当前线程的 context。

【参数】

参数	描述	输入/输出
context	当前的 context	输出

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

2.3.13 ES_NPU_DestroyContext

【函数体】

ES_S32 ES_NPU_DestroyContext (npu_context context)

【描述】

销毁 context。

【参数】

参数	描述	输入/输出
context	要销毁的 context	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

2.3.14 ES_NPU_CreateStream

【函数体】

ES_S32 ES_NPU_CreateStream(npu_stream *stream)

【描述】

在当前线程对应的 Context 中创建一个 stream。

【参数】

参数	描述	输入/输出
stream	新创建 stream 的指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

2.3.15 ES_NPU_DestroyStream

【函数体】

ES_S32 ES_NPU_DestroyStream(npu_stream stream)

【描述】

在当前线程中销毁一个 stream。

【参数】

参数	描述	输入/输出
stream	要销毁的 stream	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

2.3.16 ES_NPU_SynchronizeStream

【函数体】

ES_S32 ES_NPU_SynchronizeStream(npu_stream stream)

【描述】

阻塞线程直到指定 stream 中的所有任务都完成。

【参数】

参数	描述	输入/输出
stream	要同步的 stream	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

2.3.17 ES_NPU_AbortStream

【函数体】

ES_S32 ES_NPU_AbortStream(npu_stream stream)

【描述】

取消该 stream 中的所有任务。

【参数】

参数	描述	输入/输出
stream	要取消的 stream	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

2.3.18 ES_NPU_LoadModelFromFile

【函数体】

```
ES_S32 ES_NPU_LoadModelFromFile(
        ES_U32 * modelId,
        const ES_CHAR * modelPath)
```

【描述】

从文件中加载离线模型。

【参数】

参数	描述	输入/输出
modelId	返回的离线模型 Id	输出
modelPath	离线模型文件路径	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

2.3.19 ES_NPU_LoadModelFromMemory

【函数体】

```
ES_S32 ES_NPU_LoadModelFromMemory(
        ES_U32 * modelId,
        ES_CHAR * pBuffer,
        ES_U32 nBufLen)
```

【描述】

从内存中加载离线模型。

【参数】

参数	描述	输入/输出
modelId	返回的离线模型 Id	输出
pBuffer	存储离线模型的内存起始地址	输入
nBufLen	存储离线模型的内存长度	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

2.3.20 ES_NPU_UnloadModel

【函数体】

ES_S32 ES_NPU_UnloadModel(ES_U32 modelId)

【描述】

卸载离线模型。

【参数】

参数	描述	输入/输出
modelId	要卸载的离线模型 Id	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

2.3.21 ES_NPU_LoadCompositeModel

【函数体】

```
ES_S32 ES_NPU_LoadCompositeModel(
                ES_U32 * modelId,
                const ES_CHAR *modelPath)
```

【描述】

加载复合模型，复合模型是为了更大化利用 npu/dsp 硬件而制作的模型集。

【参数】

参数	描述	输入/输出
modelId	返回的离线模型 Id	输出
modelPath	模型目录路径	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

【注意】

复合模型加载接口仅用于复合模型加载，无法与普通离线模型通用。

2.3.22 ES_NPU_GetCompositeModelInfo

【函数体】

```
ES_S32 ES_NPU_GetCompositeModelInfo(
                ES_U32 modelId,
                NPU_COMPOSITE_MODEL_INFO_S *compositeModelInfo)
```

【描述】

获取复合模型信息，包含模型集中所有模型的详细信息。

【参数】

参数	描述	输入/输出
modelId	复合模型 Id	输入
compositeModelInfo	复合模型的模型信息	输出

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

【注意】

此接口仅针对复合模型的 modelId 有效。

2.3.23 ES_NPU_UnloadCompositeModel

【函数体】

ES_S32 ES_NPU_UnloadCompositeModel(ES_U32 modelId)

【描述】

获取复合模型信息，包含模型集中所有模型的详细信息。

【参数】

参数	描述	输入/输出
modelId	复合模型 Id	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

【注意】

此接口仅针对复合模型的 modelId 有效。

2.3.24 ES_NPU_GetNumInputTensors

【函数体】

ES_S32 ES_NPU_GetNumInputTensors(

ES_U32 modelId,
ES_S32 * inputTensors)

【描述】

获取指定模型的 input tensor 的数量。

【参数】

参数	描述	输入/输出
modelId	离线模型 Id	输入
inputTensors	返回的 input tensor 的数量	输出

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

2.3.25 ES_NPU_GetNumOutputTensors

【函数体】

```
ES_S32 ES_NPU_GetNumOutputTensors(
        ES_U32 modelId,
        ES_S32 *outputTensors)
```

【描述】

获取指定模型的 output tensor 的数量。

【参数】

参数	描述	输入/输出
modelId	离线模型 Id	输入
outputTensors	返回的 output tensor 的数量	输出

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

2.3.26 ES_NPU_GetInputTensorDesc

【函数体】

```
ES_S32 ES_NPU_GetInputTensorDesc(
        ES_U32 modelId,
        ES_S32 tensorId,
        NPU_TENSOR_S *tensor)
```

【描述】

获取指定模型指定的输入 tensor 描述。

【参数】

参数	描述	输入/输出
modelId	离线模型 Id	输入
tensorId	tensorId，从 0 开始，最大值为 ES_NPU_GetNumInputTensors 接口获取的 inputTensors - 1	输入
tensor	返回 tensor 描述信息	输出

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

2.3.27 ES_NPU_GetOutputTensorDesc

【函数体】

```
ES_S32 ES_NPU_GetOutputTensorDesc(
        ES_U32 modelId,
        ES_S32 tensorId,
        NPU_TENSOR_S *tensor)
```

【描述】

获取指定模型指定的输出 tensor 描述。

【参数】

参数	描述	输入/输出
modelId	离线模型 Id	输入
tensorId	tensorId，从 0 开始，最大值为 ES_NPU_GetNumOutputTensors 接口获取的 outputTensors - 1	输入
tensor	返回 tensor 描述信息	输出

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

2.3.28 ES_NPU_Prepare

【函数体】

ES_S32 ES_NPU_Prepare (

ES_DEV_BUF_S *devBuf)

【描述】

为任务执行中所需的 DMA Buffer 建立对 NPU 设备的 SMMU 映射。调用此接口前必须先调用 ES_NPU_SetDevice。

【参数】

参数	描述	输入/输出
devBuf	需要映射的 DMA Buffer 指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

2.3.29 ES_NPU_Unprepare

【函数体】

ES_S32 ES_NPU_Unprepare (

ES_DEV_BUF_S *devBuf)

【描述】

释放此 DMA Buffer 对 NPU 设备的 SMMU 映射，与 ES_NPU_Prepare 成对使用。

【参数】

参数	描述	输入/输出
devBuf	需要反映射的 DMA Buffer 指针	输入

【返回值】

返回值	描述
0	成功

非 0	失败，参见 2.5 错误码

2.3.30 ES_NPU_Submit

【函数体】

```
ES_S32 ES_NPU_Submit(
        NPU_TASK_S *tasks,
        ES_U32 numTasks)
```

【描述】

提交任务，同步接口。该接口提交 NPU 任务并等待所有任务执行完成。

【参数】

参数	描述	输入/输出
tasks	提交的 tasks 数组指针	输入
numTasks	提交的 task 的数量	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

2.3.31 ES_NPU_SubmitAsync

【函数体】

```
ES_S32 ES_NPU_SubmitAsync(
        NPU_TASK_S *tasks,
        ES_U32 numTasks,
        npu_stream stream)
```

【描述】

提交任务，异步接口。该接口提交 NPU 任务后直接返回，用户需要通过 ES_NPU_ProcessReport 查询对应 stream 上任务的完成状态。

【参数】

参数	描述	输入/输出
tasks	提交的 tasks 数组指针	输入

参数	描述	输入/输出
numTasks	提交的 task 的数量	输入
stream	要提交任务到指定 stream 的指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

2.3.32 ES_NPU_AllocTaskMemory

【函数体】

```
ES_S32 ES_NPU_AllocTaskMemory(
        ES_U32 modelId,
        ES_U32 nums,
        NPU_TASK_MEM_S *taskMem);
```

【描述】

分配任务所需内存。

【参数】

参数	描述	输入/输出
modelId	所需分配内存的任务对应的模型 id	输入
nums	所需分配内存的任务数量	输入
taskMem	保存分配好的内存信息	输出

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

2.3.33 ES_NPU_ReleaseTaskMemory

【函数体】

```
ES_S32 ES_NPU_ReleaseTaskMemory(
        ES_U32 modelId,
        ES_U32 nums,
        NPU_TASK_MEM_S *taskMem);
```

【描述】

释放任务内存。

【参数】

参数	描述	输入/输出
modelId	所需释放的内存对应的模型 id	输入
nums	所需释放的内存数量	输入
taskMem	需要释放的内存信息	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

2.3.34 ES_NPU_SetFlexibleTaskAttr

【函数体】

```
ES_S32 ES_NPU_SetFlexibleTaskAttr(
        ES_U32 modelId,
        NPU_FLEXIBLE_TASK_ATTR_S *attr)
```

【描述】

设置 flexible task 所需属性。

【参数】

参数	描述	输入/输出
modelId	复合模型 Id	输入
attr	Flexible task 属性	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

【注意】

此接口仅针对复合模型的 modelId 有效。

2.3.35 ES_NPU_SubmitFlexibleTask

【函数体】

```
ES_S32 ES_NPU_SubmitFlexibleTask(
        NPU_TASK_S *tasks,
        ES_U32 numTasks,
        npu_stream stream);
```

【描述】

异步提交复合模型的推理任务。相对于普通离线模型任务，用户不必按照批次大小，组装对应的模型任务，可直接根据单帧要求，构造相应任务，降低任务构造复杂度。对于多帧任务，可以通过任务数量指定。同时，npu runtime 对于复合模型任务的调度，会自动调节批大小，以达到更优的性能。

【参数】

参数	描述	输入/输出
tasks	需要提交的任务指针	输入
numTask	提交的任务数量	输入
stream	提交的任务对应的 stream	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

【注意】

此接口提交的任务需要使用 ES_NPU_AllocTaskMemory 接口分配内存，并且只能用于复合模型的任务提交。

2.3.36 ES_NPU_ProcessReport

【函数体】

```
ES_S32 ES_NPU_ProcessReport(
        npu_stream stream,
        ES_S32 timeoutMs)
```

【描述】

设置等待超时时间，同时查询哪些 task 运行完成，然后调用 task 的回调函数。如果在超时时间到达时没有任务完成，则该函数也会返回。

【参数】

参数	描述	输入/输出
stream	需要处理的 stream 指针	输入
timeoutMs	单位为 ms。如果 timeoutMs 为-1，此接口会阻塞等待直到有 task 运行完成。如果 timeoutMs >=0，等待指定时间去查询 task 完成	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见 2.5 错误码

2.4 数据类型和数据结构

NPU Runtime 相关数据类型、数据结构定义如下：

- NPU_DIMS4_S：定义 tensor 维度参数
- NPU_TENSOR_S：定义 tensor 结构参数
- NPU_TASK_S：定义 npu task 结构体描述
- NPU_TaskCallback：定义 task 执行完毕后需调用的回调函数
- npu_context：定义 npu 的 context 指针
- npu_stream ：定义 npu 的 stream 指针

2.4.1 NPU_DIMS4_S

【说明】

定义 tensor 维度参数。

【定义】

```
typedef struct {
    ES_S32 n;
    ES_S32 c;
    ES_S32 h;
    ES_S32 w;
} NPU_DIMS4_S
```

【成员】

参数	描述
n	批处理大小，表示 batch 数量
c	表示一张图像中的通道数
h	表示图像垂直维度的像素数
w	表示图像水平维度的像素数

2.4.2 NPU_TENSOR_S

【说明】

定义 tensor 结构参数。

【定义】

```
typedef struct {
    ES_CHAR name[ES_NPU_TENSOR_DESC_NAME_MAX_LEN + 1];
    ES_U64 bufferSize;
    NPU_DIMS4_S dims;
    ES_U8 dataFormat;
    ES_U8 dataType;
    ES_U8 dataCategory;
    ES_U8 pixelFormat;
    ES_U8 pixelMapping;
    ES_U32 stride[ES_NPU_TENSOR_DESC_NUM_STRIDES];
```

} NPU_TENSOR_S

【成员】

参数	描述
name	Tensor 的名字
bufferSize	Tensor 的 buffer 大小
dims	Tensor 的维度参数
dataFormat	Tensor 数据排布格式(NCHW/NHWC)
dataType	Tensor 的数据类型(int8、int16、fp16、fp32 等)
dataCategory	Tensor 表达的数据类型(image、weight、bias 等)
pixelFormat	像素格式
pixelMapping	像素映射方式
stride	stride 信息

2.4.3 NPU_TASK_S

【说明】

定义 task 结构参数。

【定义】

```
typedef struct {
    ES_U32 taskId;
    ES_U32 modelId;
    ES_U8 inputFdNum;
    ES_U8 outputFdNum;
    ES_DEV_BUF_S inputFd[ESTASK_MAX_FD_CNT];
    ES_DEV_BUF_S outputFd[ESTASK_MAX_FD_CNT];
    NPU_TaskCallback callback;
    ES_VOID *callbackArg;
    NPU_TASK_STATE_E state;
    ES_U8 sdkPrivate[ES_TASK_SDK_PRIVATE_LEN];
} NPU_TASK_S
```

【成员】

参数	描述
TaskId	task 的 ID，task 提交后由系统分配
modelId	该 task 所对应的模型 ID
inputFdNum	作为输入的内存块数量
outputFdNum	作为输出的内存块数量
inputFd	存放输入内存块的信息
outputFd	存放输出内存块的信息
callback	task 执行完毕后需调用的回调函数
callbackArg	回调函数的参数
state	task 状态

2.4.4 NPU_TASK_MEM_S

【定义】

```
typedef struct {
    ES_U8 inputFdNum;
    ES_U8 outputFdNum;
    ES_DEV_BUF_S inputFd[ES_TASK_MAX_FD_CNT];
    ES_DEV_BUF_S outputFd[ES_TASK_MAX_FD_CNT];
} NPU_TASK_MEM_S;
```

【成员】

参数	描述
inputFdNum	作为输入的内存块数量
outputFdNum	作为输出的内存块数量
inputFd	存放输入内存块的信息
outputFd	存放输出内存块的信息

2.4.5 NPU_MODEL_INFO_S

【定义】

```
typedef struct {
    ES_CHAR modelName[MAX_MODEL_FILE_NAME];
```

```
    NPU_MODEL_TYPE_E modelType;
    ES_U32 modelBatch;
    ES_FLOAT modelCost;
    ES_U32 modelId;
    ES_U32 modelDevices;
} NPU_MODEL_INFO_S;
```

【成员】

参数	描述
modelName	模型文件路径
modelType	模型类型
modelBatch	模型批大小
modelCost	模型代价，消耗的理论时间（ms）
modelId	模型 Id
modelDevices	模型需要的硬件数量

2.4.6 NPU_COMPOSITE_MODEL_INFO_S

【定义】

```
typedef struct {
    ES_U32 modelNums;
    NPU_MODEL_INFO_S modelsInfo[MAX_MODELS_OF_SET];
} NPU_COMPOSITE_MODEL_INFO_S;
```

【成员】

参数	描述
modelNums	模型数量
modelsInfo	模型信息数组

2.4.7 NPU_FLEXIBLE_TASK_ATTR_S

【定义】

```
typedef struct {
    ES_S32 timeOut;
} NPU_FLEXIBLE_TASK_ATTR_S;
```

【成员】

参数	描述
timeOut	任务超时时间（超时后，尽快调度任务 ms）

2.4.8 NPU_TaskCallback

【说明】

task 执行完毕后需调用的回调函数。

【定义】

typedef ES_S32 (*NPU_TaskCallback)(void *arg)

2.4.9 npu_context

【说明】

定义 npu 的 context 指针。

【定义】

typedef void * npu_context

2.4.10 npu_stream

【说明】

定义 npu 的 stream 指针。

【定义】

typedef void * npu_stream

2.5 错误码

参数	值	描述
ES_NPU_ERROR_BAD_PARAM	0xA00F6003	非法参数
ES_NPU_ERROR_NULL_PTR	0xA00F6006	空指针
ES_NPU_ERROR_NO_MEMORY	0xA00F600C	内存不足
ES_NPU_ERROR_INVALID_ADDR	0xA00F6011	无效地址
ES_NPU_ERROR_BUSY	0xA00F6012	设备繁忙

参数	值	描述
ES_NPU_ERROR_NOT_INIT	0xA00F6040	设备未初始化
ES_NPU_ERROR_TIME_OUT	0xA00F6041	操作超时
ES_NPU_ERROR_INVALID_STATE	0xA00F6042	无效状态
ES_NPU_ERROR_INVALID_SIZE	0xA00F6043	无效尺寸
ES_NPU_ERROR_BAD_VALUE	0xA00F6044	错误的数值
ES_NPU_ERROR_DEV_NOT_FOUND	0xA00F6045	设备未找到
ES_NPU_ERROR_DEV_MULTI_SET	0xA00F6046	多次设备设置
ES_NPU_ERROR_DEV_OPEN_FAILED	0xA00F6047	设备打开失败
ES_NPU_ERROR_MODEL_NOT_PRESENT	0xA00F6048	模型未加载
ES_NPU_ERROR_MODEL_NOT_FOUND	0xA00F6049	未找到模型
ES_NPU_ERROR_CONTEXT_NOT_FOUND	0xA00F604A	上下文未找到
ES_NPU_ERROR_CONTEXT_INVALID	0xA00F604B	无效上下文
ES_NPU_ERROR_STREAM_NOT_FOUND	0xA00F604C	流未找到
ES_NPU_ERROR_STREAM_INVALID	0xA00F604D	无效流
ES_NPU_ERROR_TASK_NOT_FOUND	0xA00F604E	任务未找到
ES_NPU_ERROR_FILE_WRITE_FAIL	0xA00F604F	文件写入失败
ES_NPU_ERROR_FILE_READ_FAIL	0xA00F6050	文件读取失败
ES_NPU_ERROR_FILE_OPREATION_FAIL	0xA00F6051	文件操作失败
ES_NPU_ERROR_END_OF_FILE	0xA00F6052	文件结束
ES_NPU_ERROR_DIR_OPREATION_FAIL	0xA00F6053	目录操作失败
ES_NPU_ERROR_END_OF_DIR_LIST	0xA00F6054	目录列表结束
ES_NPU_ERROR_IOCTL_FAIL	0xA00F6055	IOCTL 操作失败
ES_NPU_ERROR_CREAT_EVENTFD_FAIL	0xA00F6056	创建 EventFD 失败
ES_NPU_ERROR_LOAD_OP_FAILED	0xA00F6058	加载操作失败
ES_NPU_ERROR_SUBMIT_TASK_FAILED	0xA00F6059	提交任务失败
ES_NPU_ERROR_LOAD_SRAM_FAIL	0xA00F605A	加载 SRAM 失败

参数	值	描述
ES_NPU_ERROR_BAD_SRAM_SIZE	0xA00F605B	错误的 SRAM 大小
ES_NPU_ERROR_DSP_QUERY_FAILED	0xA00F605C	DSP 查询失败

2.6 调试信息

日志等级	值	输出信息
ES_LOG_ERR	3	仅打印错误日志
ES_LOG_WARN	4	打印错误和警告的日志
ES_LOG_NOTICE	5	打印错误和警告及提示等级日志
ES_LOG_INFO	6	信息级别，用于调试和开发阶段的详细信息
ES_LOG_DBG	7	调试级别，提供更为详尽的信息，通常用于深度调试

2.7 使用注意事项

在对象的构造和析构过程中，遵循 Device->Context->Stream 的构造顺序和 Stream->Context->Device 的析构顺序是一种良好的实践，以确保对象的正确初始化和资源的正确释放。

具体而言，在构造对象时，首先构造最外层的 Device 对象，然后构造其依赖的 Context 对象，接着构造依赖于 Context 的 Stream 对象。这确保了对象的依赖关系得到正确满足，每个对象在构造时都能够依赖于已经正确构造的外层对象。

在析构对象时，首先析构 Stream 对象，接着析构依赖于 Stream 的 Context 对象，最后析构依赖于 Context 的 Device 对象。这样的顺序确保了对象的依赖关系得到正确释放，每个对象在析构时都能够依赖于已经正确析构的内层对象。

3. AcceleratorKit

3.1 概述

AcceleratorKit 封装了对于 NPU、DSP、HAE、GPU 等硬件的调用并将其导出成对应的计算加速功能接口。它包括一些较为复杂的前处理、后处理算子，图像处理，机器视觉等功能。AcceleratorKit 库提供了可在不同类型设备上执行的算子和功能，以便最大化发挥不同类型设备的计算能力。

3.2 功能描述

AcceleratorKit 提供了如下表所示的常用计算加速算子供用户使用：

算子类型	加速接口功能	运行设备
cosDistance	计算两组数据之间的余弦距离	DSP
argmax	进行 Argmax 计算，对输入的数据的指定维度进行最大排序	DSP
detectionOut	对检测网络的输出 feature map 做后处理	DSP
softmax	进行 Softmax 计算	DSP
perspectiveAffine	对图像做透视变换	DSP
warpAffine	对图像做仿射变换	DSP
similarityTransform	计算相似变换系数矩阵	CPU

3.3 API 参考

3.3.1 ES_AK_GetVersion

【函数体】

ES_S32 ES_AK_GetVersion(ES_U64 *version)

【描述】

获取 Accelerator 数字版本信息。

【参数】

参数	描述	输入/输出
Version	Accelerator 版本号	输出，参见 ES_SDK_VERSION_U

【返回值】

返回值	描述
0	成功
非 0	失败，参见 3.5 错误码

3.3.2 ES_AK_GetVersionString

【函数体】

ES_S32 ES_ AK _GetVersionString(

ES_CHAR *version,
ES_U32 maxSize)

【描述】

获取 Accelerator 字符串版本信息。

【参数】

参数	描述	输入/输出
Version	Accelerator 版本号	输出
maxSize	Accelerator 版本字符串最大长度	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见 3.5 错误码

3.3.3 ES_AK_GetCapability

【函数体】

ES_S32 ES_AK_GetCapability(ES_AK_Capability_S *capability)

【描述】

获取 Accelerator 能力信息。

【参数】

参数	描述	输入/输出
capability	Accelerator 能力信息	输出，参见 ES_AK_Capability_S

【返回值】

返回值	描述
0	成功
非 0	失败，参见 3.5 错误码

3.3.4 ES_AK_SetLogLevel

【函数体】

ES_S32 ES_AK_SetLogLevel(ES_U32 level)

【描述】

设置 Accelerator 运行时的打印日志级别。

【参数】

参数	描述	输入/输出
level	Accelerator 打印日志等级	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见 3.5 错误码

3.3.5 ES_AK_Init

【函数体】

ES_S32 ES_AK_Init()

【描述】

初始化设备资源，包括打开硬件设备，如 DSP。

【参数】

无。

【返回值】

返回值	描述
0	成功
非 0	失败，参见 3.5 错误码

3.3.6 ES_AK_Deinit

【函数体】

ES_S32 ES_AK_Deinit()

【描述】

释放设备资源，并关闭硬件设备。

【参数】

无。

【返回值】

返回值	描述
0	成功
非 0	失败，参见 3.5 错误码

3.3.7 ES_AK_SetDevice

【函数体】

```
ES_S32 ES_AK_SetDevice (
        const ES_AK_DEVICE_E*   devices,
        ES_U32                  devNum)
```

【描述】

配置用于运算的设备，比如配置指定 DSP 和 GPU 用于运算。

【参数】

参数	描述	输入/输出
devices	运行计算需要的硬件设备	输入
devNum	硬件设备配置数目	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见 3.5 错误码

3.3.8 ES_AK_GetDevice

【函数体】

```
ES_S32 ES_AK_GetDevice (
        ES_AK_DEVICE_E*   devices,
        ES_U32            devNum)
```

【描述】

获取当前的运行设备。

【参数】

参数	描述	输入/输出
devices	当前运算的硬件设备	输出
devNum	硬件设备数目	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见 3.5 错误码

3.3.9 ES_AK_DSP_CosDistance

【函数体】

```
ES_S32 ES_AK_DSP_CosDistance(
                ES_TENSOR_S     inputBase,
                ES_TENSOR_S     inputQuery,
                ES_TENSOR_S     output)
```

【描述】

计算两组数据之间的余弦距离。

【支持情况】

输入 float16，输出 float16。

输入 float16，输出 float32。

输入 float32，输出 float32。

【参数】

参数	描述	约束规格
inputBase	计算余弦距离的输入数据信息	形式为 inputBase[featureNum1, dim]，只支持二维数据运算，featureNum1 表示 inputBase 特征向量个数，dim 表示特征向量长度； featureNum1 值域：[1,512] dim 值域：[1,8192]
inputQuery	计算余弦距离的输入数据信息	形式为 inputQuery[featureNum2, dim]，只支持二维数据运算，featureNum2 表示 inputQuery 特征

参数	描述	约束规格
		向量个数，dim 表示特征向量长度，与 inputBase 相同， featureNum2 值域：[1,8192]
output	输出 InputBase 和 inputQuery 的余弦距离信息	形式为 output[featureNum1, featureNum2]

【返回值】

返回值	描述
0	成功
非 0	失败，参见 3.5 错误码

3.3.10 ES_AK_DSP_PerspectiveAffine

【函数体】

```
ES_S32 ES_AK_DSP_PerspectiveAffine(
                ES_TENSOR_S          input,
                ES_TENSOR_S          output,
                ES_FLOAT             M[AFFINE_MATRIX_SIZE],
                ES_INTER_FLAG_E      flag
                ES_BORDER_TYPES_E    borderMode,
                ES_U8                borderValue)
```

【描述】

对图像做透视变换。

【支持情况】

输入 uint8，输出 uint8。

【参数】

参数	描述	约束规格
input	输入图像数据	只支持四维数据运算，形式为 input[batch, height, width, channel]。 batch 值域：[1,512]； channel 只能为 3，即只支持 3 通道图像数据； height, width 值域：[1,8192]

参数	描述	约束规格
output	用户指定输出图像维度，经计算输出透视变换后图像数据	只支持四维数据运算，形式为 output[batch, height, width, channel]。 batch 值域：[1,512]； channel 只能为 3； height, width 值域：[1,8192]
M	透视变换矩阵	形式为 M[m11, m12, m13, m21, m22, m23, m31, m32, m33]
flag	插值计算方法	当前版本只支持 nearest 插值
borderMode	描述边界填充规则	当前只支持 BORDER_CONSTANT 模式
borderValue	BORDER_CONSTANT 填充时，填充用的值	

【返回值】

返回值	描述
0	成功
非 0	失败，参见 3.5 错误码

3.3.11 ES_AK_DSP_WarpAffine

【函数体】

```
ES_S32 ES_AK_DSP_WarpAffine(
                ES_TENSOR_S          input,
                ES_TENSOR_S          output,
                ES_S32               channel_first,
                ES_FLOAT             M[AFFINE_MATRIX_SIZE],
                ES_INTER_FLAG_E      flag
                ES_BORDER_TYPES_E    borderMode,
                ES_U8                borderValue)
```

【描述】

对图像做仿射变换。

【支持情况】

输入 uint8，输出 uint8；

输入 uint16，输出 uint16；

输入输出图像长宽比 [1/5, 5]。

【参数】

参数	描述	约束规格
input	输入图像数据	只支持四维数据： batch 值域：[1] Channel={1,3}； height, width 值域：详见附注
output	用户指定输出图像维度，经计算输出仿射变换后图像数据	只支持四维数据： batch 值域：[1]； Channel={1,3}； height, width 值域：详见附注
channel_first	数据维度含义	可选配置选项为：0、1。含义如下： 0：维度含义为[batch, height, width, channel]； 1：维度含义为[batch,channel, height, width]
M	仿射变换矩阵	形式为 M[m11, m12, m13, m21, m22, m23] 缩放因子部分>1/10
flag	插值计算方法	支持双线性插值：ES_INTER_LINEAR； 支持最近邻插值：ES_INTER_NEAREST
borderMode	边界填充规则	当前只支持 BORDER_CONSTANT 模式
borderValue	BORDER_CONSTANT 填充时，填充用的值	当前只支持填充 0

【返回值】

返回值	描述
0	成功
非 0	失败，参见 3.5 错误码

【附注】

受硬件空间的约束，输入图像和输出图像的尺寸受到一定的限制，由于约束条件较为复杂，因此给出严格可以执行的尺寸范围较为困难。图像尺寸主要受到以下几个方面的限制：

- 图像位数（8bit/16bit）；
- 图像通道数（单通道/三通道）；
- 图像通道格式（NCHW/NHWC），不同的格式有一些额外的空间使用消耗；
- 输出图像尺寸；
- 输入输出图像长宽比；输出图像尺寸；
- 仿射变换矩阵，主要是放缩和旋转分量；

另外，输入图像的原始尺寸并无约束，受到约束的是输出图像在仿射变换矩阵 M_inv（M 的逆矩阵）的作用下在输入图像中的有效区域的大小。以下给出一些经典条件下的示例，其余案例下的算子可执行性请关注算子的返回值。

经典实例如下：

1、uint8、单通道图像示例：

单通道图像不区分前后通道格式，尺寸约束一致。
input_shape: "[1,2200,2200,1]"
output_shape: "[1,2200,2200,1]"
M：旋转角度 45° 放缩系数 1.0

2、uint8、channel_first=1、三通道图像示例：

input_shape: "[1,3,2200,2200]"
output_shape: "[1,3,2200,2200]"
M：旋转角度 45° 放缩系数 1.0

3、uint8、channel_first=0、三通道图像示例：

input_shape: "[1,2200,2200,3]"
output_shape: "[1,1100,1100,3]"
M：旋转角度 45° 放缩系数 1.0

4、uint16 图像示例：

在以上 uint8 图像上缩小尺寸到一半以内

5、其他情况：

旋转 45°、135°，输出图在输入图中的对应的仿射变换的区域面积最大，其余角度对应的面积要小于此类情况，因此可以接受更大的输出图尺寸；

输出图相比输入图缩小（M 矩阵缩放分量<1）时，输出图在输入图中的对应的仿射变换的区域面积会相较不放缩时更大，因此接受的输出图尺寸会更小。

3.3.12 ES_AK_DSP_DetectionOut

【函数体】

```
ES_S32 ES_AK_DSP_DetectionOut(
                ES_TENSOR_S *           inFeaturesPtr,
                ES_S32                  inFeaturesNum,
                ES_TENSOR_S             outBoxInfo,
                ES_TENSOR_S             outBoxNum,
                ES_DET_NETWORK_E detectNet,
                ES_S32                  anchorsNum,
                ES_FLOAT    anchorScale[MAX_TOTAL_ANCHORS_NUM * 2],
                ES_S32                  imgH,
                ES_S32                  imgW,
                ES_S32                  clsNum,
                ES_FLOAT    inputScale[MAX_IN_TENSOR_NUM],
                ES_NMS_METHOD_E nmsMethod,
                ES_IOU_METHOD_E iouMethod,
                ES_BOX_TYPE_E   outBoxType,
                ES_BOOL                 coordNorm,
                ES_S32                  maxBoxesPerClass,
                ES_S32                  maxBoxesPerBatch,
                ES_FLOAT                scoreThreshold,
                ES_FLOAT                iouThreshold,
                ES_FLOAT                softNmsSigma,
                ES_FLOAT                effecImgOffsetX,
                ES_FLOAT                effecImgOffsetY)
```

【描述】

对检测网络的输出 feature map 做后处理，输出满足条件的目标 class_id 、score 以及检测框的坐标。本算子为 box_decode 和 nms 的融合算子。

【支持情况】

输入 float16，outBoxInfo 输出 float16， outBoxNum 输出 int32。

输入 float16，outBoxInfo 输出 float32，outBoxNum 输出 int32。

输入 int16，outBoxInfo 输出 float16，outBoxNum 输出 int32。

输入 int16，outBoxInfo 输出 float32，outBoxNum 输出 int32。

【参数】

参数	描述	约束规格
inFeaturesPtr	输入 feature 的 tensor 描述起始地址，该地址存放了 inFeaturesNum 个 feature	

参数	描述	约束规格
inFeaturesNum	输入的 feature 个数	值域：[1,9]
outBoxInfo	输出的 box 信息	只支持二维数据，形式为 outBoxInfo [dim0, dim1]。其中 dim0 恒为 7，表示每个 box 存放的数据为 [batchId, classId, score, coord1, coord2, coord3, coord4]。用户根据这 7 个值可以找到对应的图片、物体类别、score 以及对应的 box 坐标。其中后 4 个值为 box 坐标，支持 xyxy、xywh 等多种坐标格式的输出，由用户通过 params 进行配置。 dim1 取值无限制，为“输入 batch * maxBoxesPerBatch”
outBoxNum	输出每张图的 box 个数	为一维数据，宽度值域：[1,512]
detectNet	检测模型的名称	详细信息参见 3.4.4 枚举
clsNum	要预测的类数	值域：[1,256]
imgH	输入图片高度	需为 128 ~ 1024 间的 32 倍数
imgW	输入图片宽度	需为 128 ~ 1024 间的 32 倍数
scoreThreshold	置信度阈值	值域：[0.1,1)
iouThreshold	iou 阈值	值域：[0.1,1)
inputScale	输入数据为 int16 时生效，表示输入数据为 int16 时需要配置的量化系数	
anchorsNum	每个特征点的 anchor 个数	值域：[1,9]
anchorScale	默认为 yolo v5 官方 anchor，含义为：anchor 的 H、W 尺寸	
nmsMethod	nms 方式	详细信息参见 3.4.5 枚举
iouMethod	iou 方式	详细信息参见 3.4.6 枚举
outBoxType	输出数据格式	详细信息参见 3.4.7 枚举
coordNorm	输出数据是否做归一化	

参数	描述	约束规格
maxBoxesPerClass	每张图每个类别的最多保留 boxes 数量	值域：[1,128]
maxBoxesPerBatch	每张图最多保留 boxes 数量	值域：[1,512]
softNmsSigma	当 nmsMethod 选择为 SOFT_NMS_GAUSSIAN 时生效	值域：[0,1]
effecImgOffsetX	真实图像相对于 resize 后图像左侧的相对偏移量	值域：[0,1]
effecImgOffsetY	真实图像相对于 resize 后图像上侧的相对偏移量	值域：[0,1]

【返回值】

返回值	描述
0	成功
非 0	失败，参见 3.5 错误码

3.3.13 ES_AK_DSP_Argmax

【函数体】

```
ES_S32 ES_AK_DSP_Argmax(
                ES_TENSOR_S    input,
                ES_TENSOR_S    output,
                ES_TENSOR_S    outputIdx,
                ES_S32          k,
                ES_S32          axis)
```

【描述】

调用 DSP 设备进行 argmax 计算。

【支持情况】

输入 float16，output 输出 float16，outputIdx 输出 uint16。

输入 float16，output 输出 float32，outputIdx 输出 uint16。

输入 float32，output 输出 float32，outputIdx 输出 uint16。

【参数】

参数	描述	约束规格
input	输入数据的描述，包括 shape、类型和地址，地址内为待排序数值	只支持四维数据运算，形式为 input[batch, channel, height, width]，值域均为[1,8192]
output	输出数据的描述，包括 shape、类型和地址，地址内为（最大）排序后的数值	只支持四维数据运算，形式为 output[batch, channel, height, width]，值域均为[1,8192]
outputIdx	输出数据的描述，包括 shape、类型和地址，地址内为（最大）排序后的数值的原索引	
k	输出结果中排序维度的数据个数	值域：[1,32]
axis	指定排序的维度	当前只支持 channel 维度排序

【返回值】

返回值	描述
0	成功
非 0	失败，参见 3.5 错误码

3.3.14 ES_AK_DSP_Softmax

【函数体】

```
ES_S32 ES_AK_DSP_Softmax(
        ES_TENSOR_S     input,
        ES_TENSOR_S     output,
        ES_FLOAT        scaleIn)
```

【描述】

调用 DSP 设备进行 Softmax 计算。

【支持情况】

输入 int8，输出 float16。

输入 int8，输出 float32。

输入 float16，输出 float16。

输入 int16，输出 float16。

【参数】

参数	描述	约束规格
input	输入数据的描述，包括 shape、类型和地址，地址内为待处理数据	只支持四维数据运算，形式为 input[batch, channel, height, width]，值域均为[1,8192]
output	输出数据的描述，包括 shape、类型和地址，地址内为 softmax 处理后的数据	只支持四维数据运算，形式为 output[batch, channel, height, width]，值域均为[1,8192]
scaleIn	输入数据量化系数	值域：[0.00001，1.0] 输入为 float 时需置 1.0

【返回值】

返回值	描述
0	成功
非 0	失败，参见 3.5 错误码

3.3.15 ES_AK_CPU_SimilarityTransform

【函数体】

```
ES_S32 ES_AK_CPU_SimilarityTransform(
                ES_TENSOR_S    srcPoint,
                ES_TENSOR_S    dstPoint,
                ES_TENSOR_S    transformMat)
```

【描述】

计算相似变换矩阵。

【支持情况】

输入 float32，输出 float32。

【参数】

参数	描述	约束规格
srcPoints	计算相似变换矩阵的源点	只支持二维数据运算，形式为 srcPoints[dim0,dim1]，dim0 表示源点的个数，当前只支持 5 个点，即 dim0 只能设为 5.dim1 表示源点坐标的维度，只能为 2，即每个点是

参数	描述	约束规格
		形如（x , y）的二维坐标
dstPoints	计算相似变换矩阵的目标点	同 srcPoints
transformMat	相似变换矩阵计算结果	形状为[3,2]的矩阵

【返回值】

返回值	描述
0	成功
非 0	失败，参见 3.5 错误码

3.4 数据类型和数据结构

AcceleratorKit 相关数据类型、数据结构定义如下：

- ES_AK_Capability_S：定义 Accelerator 能力信息数据结构
- ES_AK_DEVICE_E：定义 Accelerator 支持的硬件设备
- ES_DATA_BUFFER：定义存储数据的 buffer 地址信息
- ES_DEV_BUF_S：定义 device 侧的 buffer 地址信息
- ES_TENSOR_S：描述 tensor 的 shape、buffer 地址等信息
- ES_DET_NETWORK_E：定义 ES_AK_DSP_DetectionOut 算子支持的检测网络类型
- ES_IOU_METHOD_E：定义 ES_AK_DSP_DetectionOut 算子支持的 iou 计算方法
- ES_BOX_TYPE_E：定义 ES_AK_DSP_DetectionOut 算子支持的输出 box 格式
- ES_INTER_FLAG_E：定义插值类算子的插值计算方法信息
- ES_BORDER_TYPES_E：定义边界填充方式的枚举类型

3.4.1 ES_AK_Capability_S

【说明】

定义 Accelerator 能力信息。

【定义】

```
typedef struct AK_Capability_S {
    ES_U64 reserved;
} ES_AK_Capability_S
```

【成员】

参数	描述
reserved	预留字段

3.4.2 ES_AK_DEVICE_E

【说明】

定义 Accelerator 支持的硬件设备。

【定义】

```
typedef enum AK_DEVICE_E {
        ES_AK_DEV_DSP_0 = 0x0,
        ES_AK_DEV_DSP_1 = 0x1,
        ES_AK_DEV_DSP_2 = 0x2,
        ES_AK_DEV_DSP_3 = 0x3,
        ES_AK_DEV_DSP_4 = 0x4,
        ES_AK_DEV_DSP_5 = 0x5,
        ES_AK_DEV_DSP_6 = 0x6,
        ES_AK_DEV_DSP_7 = 0x7,
        ES_AK_DEV_HAE_0 = 0x8,
        ES_AK_DEV_HAE_1 = 0x9,
        ES_AK_DEV_GPU_0 = 0xa,
        ES_AK_DEV_GPU_1 = 0xb,
        ES_AK_DEV_BUTT
} ES_AK_DEVICE_E
```

【成员】

参数	描述
ES_AK_DEV_DSP_0	DSP0 设备
ES_AK_DEV_DSP_1	DSP1 设备
ES_AK_DEV_DSP_2	DSP2 设备
ES_AK_DEV_DSP_3	DSP3 设备
ES_AK_DEV_DSP_4	DSP4 设备
ES_AK_DEV_DSP_5	DSP5 设备

参数	描述
ES_AK_DEV_DSP_6	DSP6 设备
ES_AK_DEV_DSP_7	DSP7 设备
ES_AK_DEV_HAE_0	HAE0 设备
ES_AK_DEV_HAE_1	HAE1 设备
ES_AK_DEV_GPU_0	GPU0 设备
ES_AK_DEV_GPU_1	GPU1 设备
ES_AK_DEV_BUTT	最大设备数

3.4.3 ES_TENSOR_S

【说明】

Tensor 结构参数。

【定义】

```
typedef struct {
    union {
            ES_DEV_BUF_S pData;
            void *hostBuf;
        };
    ES_DATA_PRECISION_E dataType;
    ES_U32 shapeDim;
    ES_U32 shape[MAX_DIM_CNT];
} ES_TENSOR_S
```

【成员】

参数	描述
dataType	Tensor 的数据类型，ES_DATA_PRECISION_E 枚举有： ES_PRECISION_INT8 = 1, ES_PRECISION_UINT8 = 2, ES_PRECISION_INT16 = 3, ES_PRECISION_UINT16 = 4, ES_PRECISION_INT32 = 5, ES_PRECISION_UINT32 = 6, ES_PRECISION_INT64 = 7,

	ES_PRECISION_UINT64 = 8, ES_PRECISION_FP16 = 9, ES_PRECISION_FP32 = 10
pData	device 侧的 buffer 起始地址
hostBuf	cpu 侧的 buffer 起始地址
shapeDim	描述 tensor 的真实维度个数
shape	长度为 MAX_DIM_CNT 的数组，按顺序存放 tensor 每个维度的宽度信息。MAX_DIM_CNT 值为 6

3.4.4 ES_DET_NETWORK_E

【说明】

detectionOut 算子支持的检测网络类型。

【定义】

```
typedef enum {
    yolov3_u = 0,
    yolov3_d = 1,
    yolov4 = 2,
    yolov5 = 3,
    yolov7 = 4,
    yolov8 = 5
} ES_DET_NETWORK_E
```

【注】

yolov3_d 指的是 darknet 版，原始的 yolo v3 版本；yolov3_u 指的是 ultralytics 版，是目前使用最为广泛的版本。用户根据自己模型的来源，自行选择对应的版本，如果不确定模型出处，建议首先尝试配置 yolov3_u 进行测试。

3.4.5 ES_NMS_METHOD_E

【说明】

detectionOut 算子支持的 NMS 计算方法。

【定义】

```
typedef enum {
    HARD_NMS,
```

```
    SOFT_NMS_GAUSSIAN,
    SOFT_NMS_LINEAR
} ES_NMS_METHOD_E
```

3.4.6 ES_IOU_METHOD_E

【说明】

detectionOut 算子支持的 iou 方法。

【定义】

```
typedef enum {
    IOU,
    GIOU,
    DIOU
} ES_IOU_METHOD_E
```

3.4.7 ES_BOX_TYPE_E

【说明】

detectionOut 算子支持的输出 box 格式。

【定义】

```
typedef enum {
    XminYminXmaxYmax,
    XminYminWH,
    YminXminYmaxXmax,
    XmidYmidWH
} ES_BOX_TYPE_E
```

3.4.8 ES_INTER_FLAG_E

【说明】

描述插值方式的枚举类型。

【定义】

```
typedef enum {
    ES_INTER_NEAREST = 0,
    ES_INTER_LINEAR = 1,
    ES_INTER_AREA = 2,
```

```
    ES_INTER_CUBIC = 3,
    ES_INTER_LANCZOS4 = 4,
    ES_INTER_NEAREST_EXACT = 6,
    ES_INTER_STRETCH = 20,
    ES_INTER_FILTER = 21
} ES_INTER_FLAG_E
```

3.4.9 ES_BORDER_TYPES_E

【说明】

描述边界填充方式的枚举类型。

【定义】

```
typedef enum {
    BORDER_CONSTANT = 0,
    BORDER_REPLICATE = 1,
    BORDER_REFLECT = 2,
    BORDER_WRAP = 3,
    BORDER_REFLECT_101 = 4,
    BORDER_TRANSPARENT = 5,
    BORDER_ISOLATED = 16
} ES_BORDER_TYPES_E
```

3.5 错误码

参数	值	描述
ES_AK_ERROR_INVALID_DEVID	0xA0146029	设备 ID 超出合法范围
ES_AK_ERROR_ILLEGAL_PARAM	0xA014602A	非法参数
ES_AK_ERROR_LOAD_OP_FAILED	0xA014602B	加载算子失败
ES_AK_ERROR_SUBMIT_TASK_FAILED	0xA014602C	提交任务失败
ES_AK_ERROR_QUERY_FAILED	0xA014602D	Query 查询超时
ES_AK_ERROR_NOT_SUPPORT	0xA014602E	不支持的参数或者功能
ES_AK_ERROR_OUTBUF	0xA014602F	申请的 buffer 过多
ES_AK_ERROR_GET_OP_FUNC_FAILED	0xA0146030	获取算子执行函数失败

参数	值	描述
ES_AK_ERROR_OP_EVAL_FAILED	0xA0146031	算子执行失败
ES_AK_ERROR_OPEN_FILE_FAILED	0xA0146032	文件打开失败
ES_AK_ERROR_READ_FILE_FAILED	0xA0146033	文件读取失败
ES_AK_ERROR_WRITE_FILE_FAILED	0xA0146034	文件写入失败
ES_AK_ERROR_INVALID_VER	0xA0146035	非法版本信息

3.6 调试信息

用户可以通过以下日志信息了解软件当前执行状态，方便在出错时准确定位问题。

- invalid vec size：shape 字段为空或者无法正常解析。
- The operator registration list is empty, check that compiler.a or manager.so is linked correctly：算子列表为空，没有正确链接到 compiler.a 和 manager.so，需要在 cmake 中或者其他形式链接到这两个库文件，库文件包含了已经注册的算子。
- No candidate operators for xxx：调用算子时配置的参数不符合要求，没找到匹配的算子。
- Failed to emit operator description：发射算子失败，可能是参数校验失败；如果是自定义算子则可能是算子描述指针为空（未正确设置）或者 param 的 buffer 超出上限（32 个）。
- Failed to initialize buffers：Input、output 或者 param 的 fd 配置异常。请检查 input、output 以及 param 的 buffer 的数量。
- Failed to load operator：加载指定算子失败，请检查 dsp 算子库是否正确编译；或者检查 emit 发射的算子名是否符合要求。
- Failed to submit task for operator：计算执行失败，检查 driver，framework 侧的打印信息。

3.7 使用注意事项

3.7.1 参数配置

在调用 Acclerator-kit 接口前，首先需要配置算子对应的 config 结构体作为入参，比如调用 argmax 算子前需要提前配置好 ES_DSP_Argmax 结构体。

此外 inputs、outputs 的数据 shape、type 也需要提前做好配置，配置数据范围需满足接口要求，否则无法通过参数校验。

