# BM1682 Video API Document

Revision History

|  |  |  |
| --- | --- | --- |
| Version | Comments | Author |
| 0.1 | Initial Draft | Li Xun 2018-3-1 |
|  |  |  |
|  |  |  |

## API说明

* BMVidDecCreate
* BMVidDecReset
* BMVidDecGetCaps
* BMVidDecDecode
* BMVidDecGetOutput
* BMVidDecClearOutput
* BMVidDecFlush
* BMVidDecDelete

1. BMVidDecCreate

[purpose]

* 创建解码器句柄（句柄中包含coreIdx, currInstance，coreCaps，初始化参数等）
* 根据当前core0/1的当前计算能力，以默认1080p的计算资源分配当前instance的位置(coreIdx, instant) （在后期得到SeqInit后，可以对当前的计算资源进行调整，内存资源可以暂时浪费些），当codecType为HEVC时，使用coreIdx=2
* 调用VPU\_init, VPU\_DecOpen来初始化当前的decoder instance
* 如果当前coreIdx是第一次启动，启动相应的线程，并使该线程处于等待数据状态。
* 该线程流程见最后

[syntax]

void \*BMVidDecCreate(BMCodecType type, DecOpenParam \*param);

1. BMVidDecReset

[Purpose]

* 允许用户快速重置当前的解码器配置(codecType有变化时，用户必须重新delete/Create)
* 清除当前的解码缓存，等待重新进行解码

[syntax]

Int BMVidDecReset(void \*decoder, DecOpenParam \*param);

1. BMVidDecGetCaps

[Purpose]

* 允许用户查询当前handle的解码能力，解码信息（主要是VPU\_SeqInit后的参数）
* 允许用户查询当前的解码状态（如seqInit\_done，decoding等）
* 后期客户需求的信息

[syntax]

int XAVSDecoderGetCaps(void \*decoder， BMVidDecCaps \*Caps);

1. BMVidDecDecode

[Purpose]

* 将客户的bitstream数据加入到VPU input bitsteam buffer中，如果full则返回失败，由客户再次尝试写入。（VPU\_DecGetBitStreamBuffer, VPU\_DecUpdateBistreamBuffer）
* 该函数在将数据写入后，不应等待解码结果，立即返回

[syntax]

Int BMVidDecDecode(void\* decoder, const BMVideoPacket \*packet);

1. BMVidDecGetOutput

[Purpose]

* 获取指定handle的解码输出buffer，这个buffer用mmap将底层的硬件解码输出地址直接映射上来，避免拷贝（VPU\_DecGetOutputInfo），该接口必须配合BMVidDecClearOutput来避免输出buffer被占用。
* 获取解码Output的方式，除了这种polling的方式，还可以通过在BMVideDecCreate中传过去的param参数中提供CallBack Function，由BMVidDec主动来调用，这可根据客户需求，将来添加。这种方式不需要暴露BMVidDecClearOutput接口给客户。
* 如果当前handle没有就绪的输出buffer，当TimeOut==0的时候则一直等待，否则超过TimeOut时间后返回失败。客户处理错误，等待一段时间后再次Polling

[Syntax]

Int BMVidDecGetOutput(void \*decoder, BMVidSurface \*surface，int TimeOut);

// BMVidSurface除mmap后的地址外，还包含color\_space,resolution，timestamp, 等相关信息，还有后面释放需要用到的buf index.

1. BMVidDecClearOutput

[Purpose]

* 释放指定handle的解码输出buffer（VPU\_DecClrDispFlag）
* 注意，这里的VPU\_DecClrDispFlag是通过置标志位，由thread执行到该handle时进行释放的，不能在线程外执行，否则VPU Firmware有可能不响应。

[Syntax]

Int BMVidDecClearOutput(void \*decoder, BMVidSurface \*surface);

1. BMVidDecFlush

[Purpose]

* 将指定handle的stream buffer设置STREAM\_END\_FLAG (VPU\_DecUpdateBitstreamBuffer)，将残留的所有数据都解完
* 提供两种模式：a)仍然解出来的output Image，由BMVidDecGetOutput来拉空Image

b)不再进行解码，直接free所有的stream buffer

这两种模式由param初始化参数中给出

[Syntax]

Int BMVidDecFlush(void \*decoder);

1. BMVidDecDelete

[Purpose]

* 释放BMVidDecCreate占用的资源
* 当max\_instance==0的时候终止该coreIdx的线程

[Syntax]

Int BMVidDecDelete(void \*decoder);

## 流程框图

这里最核心的是每个coreIdx启动的thread，每个thread就是个小的任务伺服器，对外界的输入进行轮询处理。



Figure 1 decoder thread flow

整体的框图流程就比较简单



Figure 2 Decoder Framework