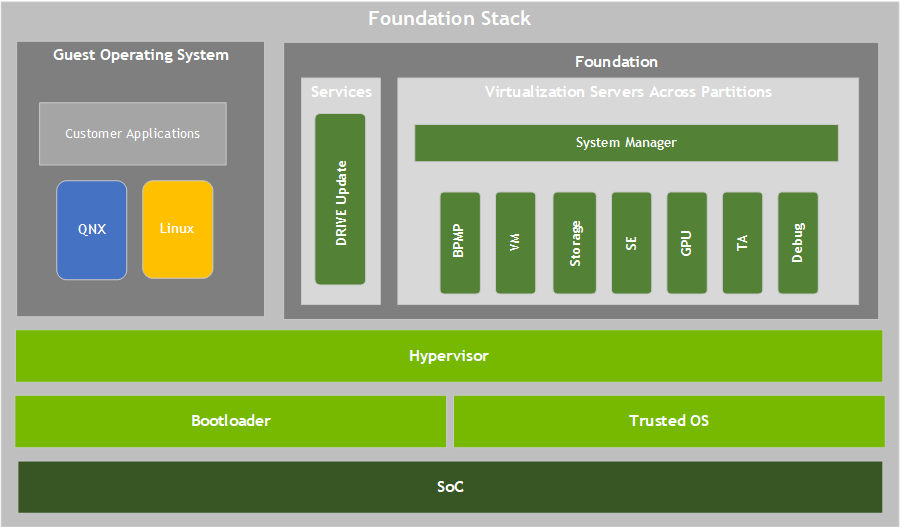
# 1. Overview

## 1.1 Platform Software Stacks



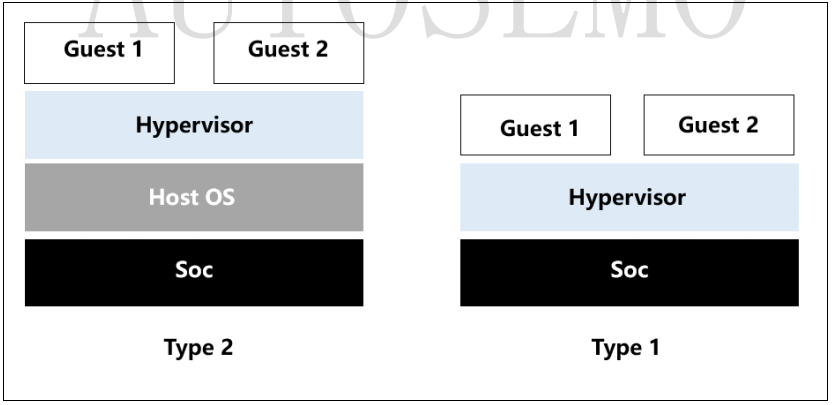
Hypervisor

将系统分成分区的受信任软件服务器。每个分区可以包含操作系统或裸机应用程序。

Hypervisor 可以划分为两大类，一类是 Type1 裸机型，Hypervisor 直接运行在硬件设备上的，也叫做 Bare-Metal Hardware Virtualization（裸机虚拟化环境）；一类是 Type2 主机托管型，也叫做 Hosted Virtualization （主机虚拟化环境）。

Type2 型 Hypervisor 需要借助宿主操作系统来管理 CPU、内存、网络等资源，由于 Hypervisor 和硬件之间存在一个宿主操作系统，Hypervisor 及 VM 的所有操作都要经过宿主操作系统，所以就不可避免地会存在延迟、性能损耗，同时宿主操作系统的安全缺陷及稳定性问题都会影响到运行在之上的 VM（虚拟机），所以 , Type-2 型 Hypervisor 主要用于对性能和安全要求不高的场合，比如 : 个人 PC 系统。

这里看，DRIVE OS的Hypervisor是属于type1。



Drive OS最核心的目标之一就是在单一硬件基础上提供高效的多操作系统支持，

DRIVE OS处于底层硬件和Hypervisor之上。

guest OS

分配用户的操作系统需要控制的外围设备

Services

DRIVE Update 的服务

[Bootloader](https://developer.nvidia.com/docs/drive/drive-os/archives/6.0.4/linux/sdk/common/topics/bootloader_setup/BootloaderProgramming1.html)

在引导期间运行以加载固件组件的固件，如引导映像、分区映像和其他固件

[Trusted OS](https://developer.nvidia.com/docs/drive/drive-os/archives/6.0.4/linux/sdk/common/topics/security_concepts/TrustedOS84.html)

在PCT的配置信息描述了虚拟的Trusted OS设备（PCT是什么？），这里的trusted os是一个什么样的角色？

BPMP

启动和电源管理处理器

Guest Virtual Machines和BPMP之间的通讯

VM Server

host1X virtualizes NvHost

Security Engine (SE) Server

并行虚拟化技术允许多个访客虚拟机访问安全引擎加密硬件加速器

Trusted Applications (TA)

通过Trusted Applications，Trusted OS公开了一组核心服务，这些服务在加密操作中使用托管安全资产，而不向非安全客户软件公开

GPU

运行在虚拟化核心之上，并处理多个客户端用户虚拟机之间的GPU共享。GPU虚拟化还包括在每个用户虚拟机内部运行的准虚拟化GPU客户端驱动程序

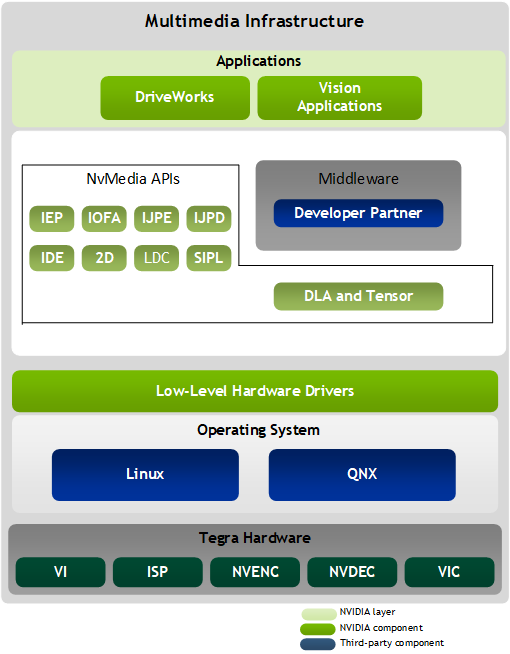
Storage Server

并行虚拟化存储访问，以便在多个用户虚拟机之间共享物理存储设备

Debug Server

为用户虚拟机（VM）的内核级调试提供支持

NvMedia Architecture



NvMedia提供强大的多媒体数据处理功能，在NVIDIA DRIVE®Orin上实现真正的硬件加速™ 设备。通过NvMedia和Orin固件组件，多媒体应用程序支持同时处理多个摄像机传送过来的图像数据。

提供了多媒体应用支持，包括多摄像头输入，ISP，sensor控制。支持多种图像格式，例如YUV、RGB还有RAW。以及各种为图像捕获，处理，加速，编码，显示等提供的库。另外，NvMedia还可以将图像和视频传递给OpenGL ES或者CUDA，满足开发者的其它需求

# 2. Setup and Configuration

讲述了DRIVE OS支持的板子，硬件的设置，MCU的配置，网络的配置等

# 3. Flashing

1、烧录预处理的二进制文件

2、如何定制烧录文件

3、如何在芯片上绑定NVIDIA Orin系统

| **PCT name** | **PCT** | **Profile Makefile for Standard build** |
| --- | --- | --- |
| Single Linux GOS VM | linux | profile.mk |
| Single QNX GOS VM | qnx | profile.mk |
| Dual QNX GOS VMs | dual-qnx | profile.mk |

2.2、使用Bootburn utility去烧录引导加载程序、内核和文件系统到板子上。

Bootburn utility作为Foundation包的一部分提供

2.3、更新用户数据， create\_bsp\_images.py, flash\_bsp\_images.py, or bootburn.py script

2.4、离线生成二进制文件

2.5、怎样烧录二进制文件到板子上

# 4. Embedded Software Components

## 4.1 Graphics Programming

* [OpenGL ES 2/3 Specs and Extensions](https://developer.nvidia.com/docs/drive/drive-os/archives/6.0.4/linux/sdk/api_reference/group__ogl2__group.html) in the [API Reference](https://developer.nvidia.com/docs/drive/drive-os/archives/6.0.4/linux/sdk/api_reference/index.html)
* [EGL and EGL Extensions](https://developer.nvidia.com/docs/drive/drive-os/archives/6.0.4/linux/sdk/api_reference/group__egl__group.html) in the [API Reference](https://developer.nvidia.com/docs/drive/drive-os/archives/6.0.4/linux/sdk/api_reference/index.html)
* [Vulkan Specs and Extensions](https://developer.nvidia.com/docs/drive/drive-os/archives/6.0.4/linux/sdk/api_reference/group__vulkan__group.html) in the [API Reference](https://developer.nvidia.com/docs/drive/drive-os/archives/6.0.4/linux/sdk/api_reference/index.html)
* [Vulkan SC Specs and Extensions](https://developer.nvidia.com/docs/drive/drive-os/archives/6.0.4/linux/sdk/api_reference/group__vulkansc__group.html) in the [API Reference](https://developer.nvidia.com/docs/drive/drive-os/archives/6.0.4/linux/sdk/api_reference/index.html)

图形化编程，包含了上述的部分，主要是针对一些类似于OpenGL的接口，方便直接做一些调用

Opengl es

是opengl三维图形API的子集，对手机、PDA和游戏主机等嵌入式设备而设计

EGL

是渲染平面和opengl之间的第一层，它负责连接渲染平面和管理opengl es上下文

Vulkan

是一个低开销、跨平台的二维、三维图形与计算的应用程序接口（API）。

与 OpenGL 类似的是，Vulkan 针对全平台即时3D图形程序（如电子游戏和交互媒体）而设计，并提供高性能与更均衡的CPU与GPU占用。

与 OpenGL 区别的是，Vulkan是一个底层API，而且能执行并行任务。除此之外，Vulkan还能更好地分配多个CPU核心的使用。

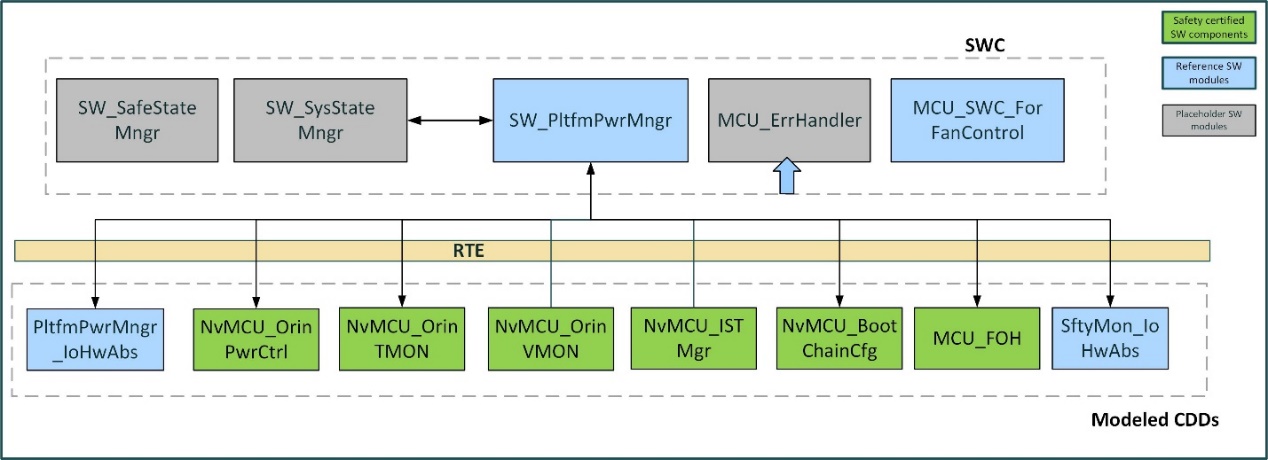
## 4.2 Window Systems

* X11
* Wayland
* EGLDevice/EGLOutput

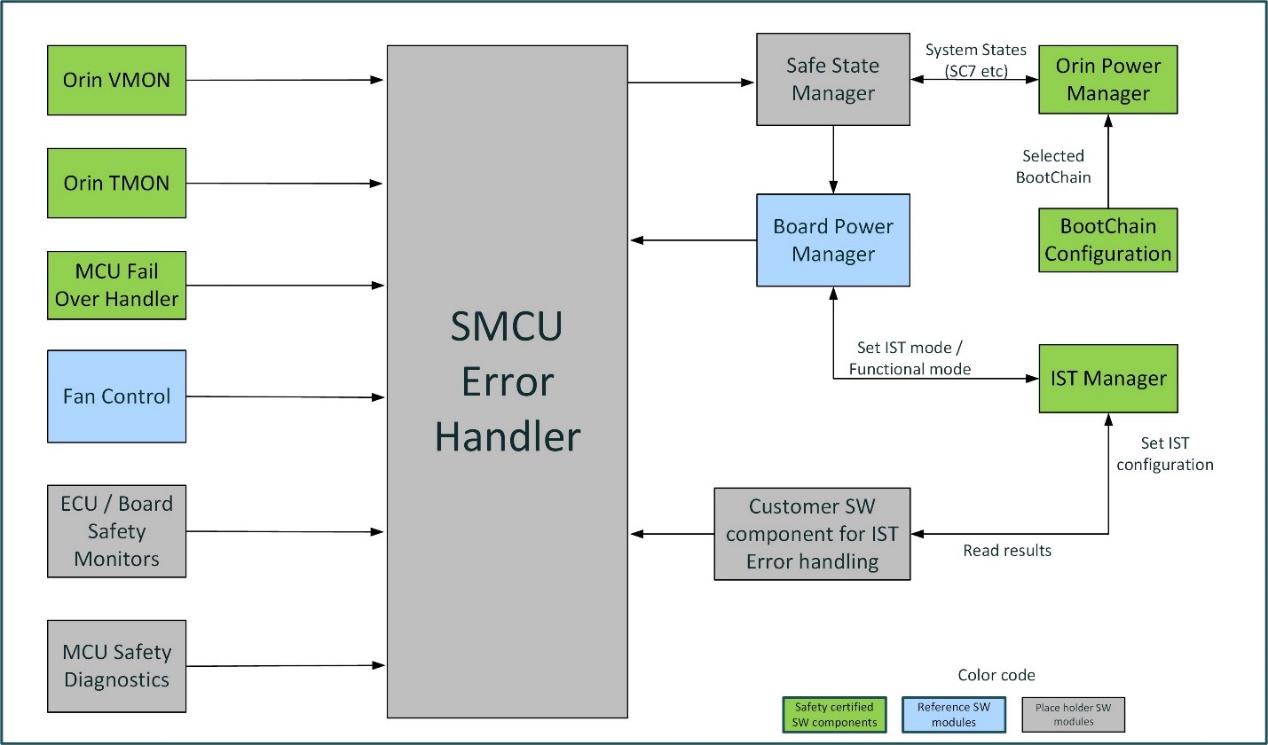
上述三种窗口图形化编程的说明，API说明

## 4.3 MCU Software Modules

High-Level Architecture



英伟达驱动操作系统 MCU 软件



NvMCU\_SwModules和错误处理程序软件之间的交互示例 元件

Platform(Board) Power State Management

1. 决定电路板的完整通电/断电顺序，并执行硬件设计建议的步骤
2. 使用NvMCU\_SwModules提供的接口执行安全相关检查、
3. 提供从客户应用程序打开和关闭电路板的接口。这些接口基于MCU串行控制台，执行shell命令
4. 提供Orin重置、MCU重置、强制关闭等接口
5. 在电源启动期间检测故障并向错误处理模块报告
6. 使用TLF关闭和唤醒MCU
7. SW\_PltfWrMgr（接口）根据硬件（唤醒、KL15…）和软件（IST、安全关机、用户命令…）输入触发的事件，方便NVIDIA Orin SoC的通电、断电和强制关机
8. PltfWrMgr\_IoHwAbs（CDD），它向SW\_PltfWrMgr提供主要与平台电源控制、以太网初始化、网络通信等相关的服务
9. 安全停机顺序作

NVIDIA Orin Power Management

1、Orin的电源管理

2、NVIDIA Orin SoC 电源的接口

3、为NVIDIA Orin SoC 的引导链设置 GPIO 引脚

4、检测硬件错误场景，实现 VRS10 和 VRS11 的安全相关潜在故障检测要求

5、Orin的安全关机

NVIDIA Orin Voltage Monitoring via VRS12 (VMON)

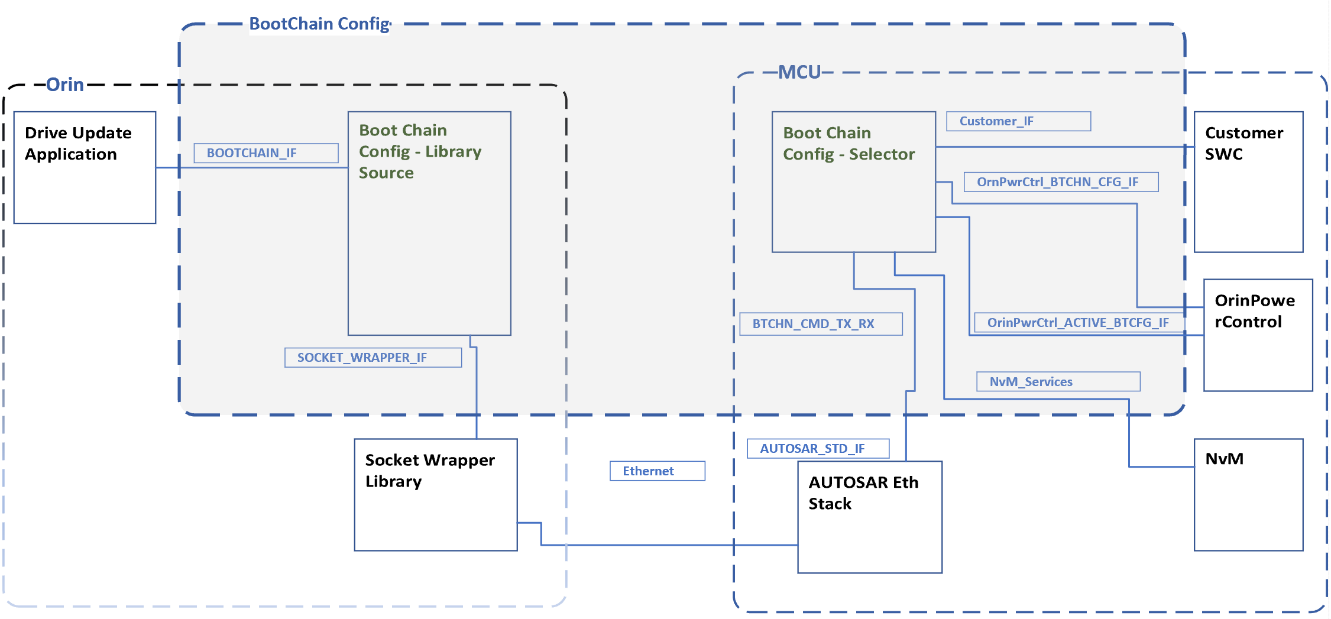
Orin的电压监测

NVIDIA Orin Temperature Monitoring

Orin的温度监测

NVIDIA Orin 溫度监测 SoC TMON 温度传感器 （TMP451）

NVIDIA Orin Boot Chain Configuration Support



在Orin上运行的NVIDIA DRIVE Update应用程序通过维护多个boot chain（boot chainA、B、C、D）可靠地管理NVIDIA Orin的软件更新。

在MCU和Orin上都运行着boot chain

NVIIDA Orin IST Manager

什么是IST

Failover Handler on MCU

对orin的处理进行监测， Error Pin ，引脚的方式

Fan Control and Monitoring

风扇控制

Common Interface

在DRIVE AGX Orin中，NVIDIA Orin通过以太网使用UDP协议与MCU通信，以更新MCU固件及其配置

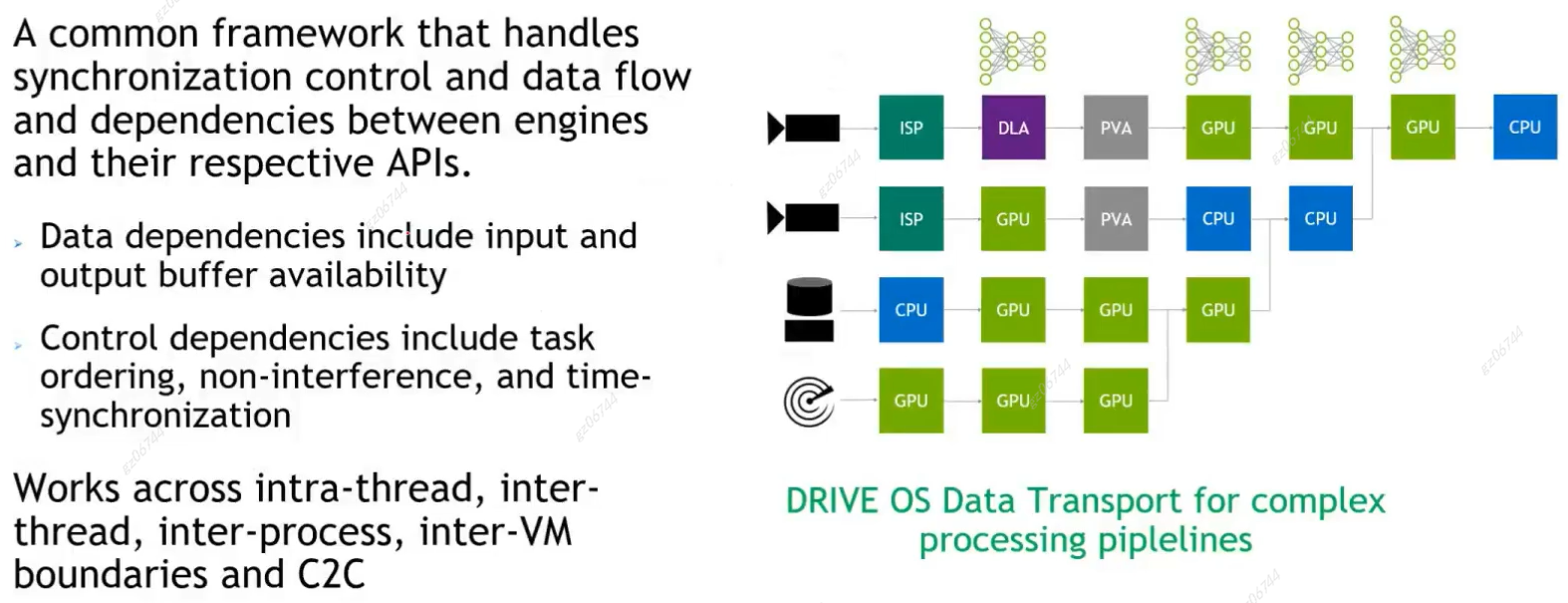
## 4.4 NvDisplay

Nvidia drive os的显示体系结构

主要是一下的两个库

* OpenWFD (libtegrawfd.so)
* Nvdisp (devg-modeset)

## 4.5 NvStreams



NVIDIA SDK提供了几种不同的库（NvMedia、CUDA、OpenGL），用于生成和处理各种类型的图像和更高维度的数据。这些库使用的接口是独立编写的，以满足不同的需求，每个库都有自己的内存和同步资源表示方法。它们之间没有直接的数据交换方式。

此外，这些库使用的资源的分配方式和分配时间的大部分细节对应用程序是隐藏的。

Comparison with EGL

EGL不适合安全认证系统的严格要求

NvStreams要求应用程序比EGL更直接地参与确定资源需求、分配资源和交换资源

NvStreams Libraries

* [NvSciBuf](https://developer.nvidia.com/docs/drive/drive-os/archives/6.0.4/linux/sdk/common/topics/nvsci/BufferAllocation1.html)
* [NvSciSync](https://developer.nvidia.com/docs/drive/drive-os/archives/6.0.4/linux/sdk/common/topics/nvsci/Synchronization1.html)
* [NvSciStream](https://developer.nvidia.com/docs/drive/drive-os/archives/6.0.4/linux/sdk/common/topics/nvsci/NvStreams1.html)

[NvSciBuf](https://developer.nvidia.com/docs/drive/drive-os/archives/6.0.4/linux/sdk/common/topics/nvsci/BufferAllocation1.html)

NvSciBuf是一个缓冲区分配模块，它可以使应用程序在由不同引擎API管理的各种硬件引擎之间分配可共享的缓冲区。如果两个或多个硬件引擎想要访问公共缓冲区（例如，一个引擎正在将数据写入缓冲区，另一个引擎在从缓冲区读取数据。）

NvSciIpc

如果应用程序涉及多个进程，则NvSciBuf结构的交换只能通过NvSciIpc通道

分配模型

1. 应用程序为每个访问器创建一个属性列表
2. 设置应用程序将在对应的属性列表中创建的缓冲区的属性

应用程序必须使用 NvSciBuf/UMD API，必须直接设置 NvSciBuf 常规属性

对于 CUDA，应用程序必须设置所有必需的属性

1. 协调这些多个属性列表。和解进程保证分配一个满足所有访问器约束的公共缓冲区
2. 使用协调的属性列表分配缓冲区。已协调的属性列表用于分配对象并在这些对象的生命周期内都与其关联起来
3. 与所有访问器共享缓冲区

支持的分配buffer的类型

* **RawBuffer**: Raw memory that is used by an application for storing data.
* **Image**: Memory used to store image data.
* **ImagePyramid**: Memory used to store ImagePyramid, a group of images arranged in multiple levels, with each level of image scaled to a specific scaling factor.
* **NvSciBufArray**: Memory used to store a group of units, where each unit represents data of various basic types such as int, float etc.
* **Tensor**: Memory used to store Tensor data.

应用程序可以选择从以下域分配内存

1. 系统内存
2. 特定dGPU的Vidmem（基于标准构建）

Buff的属性

1. 数据类型属性：特定于上一节中提到的缓冲区类型之一的属性。如果属性列表中的缓冲区类型是一种类型，则设置另一种类型的属性会返回错误
2. 常规属性：不特定于任何缓冲区类型并描述缓冲区的常规属性的属性

Attribute Lists

1. 未协调的属性列表

应用程序可以创建未协调的属性列表并执行 get/set 未协调属性列表中每个属性的操作，每个属性只允许一次set操作

NvSciBufAttrListCreate 和NvSciBufAttrListIpcImportUnreconciled 都返回一个 未协调的属性列表

应用程序不能使用未协调的属性列表来分配 NvSciBuf 对象

1. 协调的属性列表

已协调的属性列表是各种未协调列表协调的结果（即，合并和 验证），它定义了最终的缓冲区的布局/分配属性。

 成功调用

NvSciBufAttrListReconcile, NvSciBufAttrListIpcImportReconciled, or NvSciBufObjGetAttrList

返回协调的属性列表

应用程序可以使用此列表来分配 NvSciBuf 对象

1. 冲突属性列表

冲突的属性列表是由于调用NvSciBufAttrListReconcile 失败而产生

不允许应用程序对冲突执行获取/设置操作 属性列表

应用程序只能通过API NvSciBufAttrListDump使冲突属的性列表转储其内容。

举例子：

如果应用程序想要创建一个缓冲区，该缓冲区被一个引擎视为图像，并被另一个引擎视为Tensorrt，则应用程序可以创建一个属性列表，NvSciBufGeneralAttrKey\_Types属性同时设置为图像和Tensorrt，协调列表并分配对象。或者，应用程序可以创建两个不同的属性列表，一个具有图像属性，另一个具有Tensorrt属性，协调两者并分配对象。

局限性提醒

NvSciBuf 支持创建图像和Tensorrt属性的或者它们之间的协调列表。其余的 NvSciBuf 数据类型不支持与其他数据类型共存

图像和Tensorrt属性的协调只有在使用 NvMedia API 填充Tensorrt的情况下才能成功

Buffer管理

应用程序可以使用协调的属性列表来创建任意数量的 NvSciBufObj 对象。

每个 NvSciBufObj 表示一个缓冲区，协调的属性列表与每个对象相关联，直到释放对象。

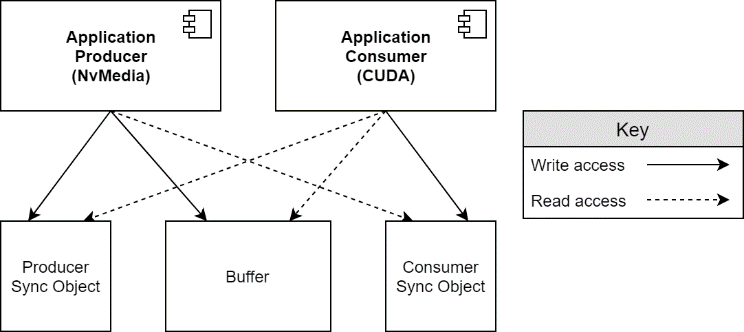
应用程序可以使用NvMedia/CUDA API从分配的缓冲区中创建NvMedia/CUDA数据类型，并且可以通过使用适当的API提交到NvMedia/CUDA硬件引擎来对缓冲区进行操作。

VidaMen缓冲区

NvSciBuf支持从特定dGPU的Vidmem分配缓冲区。应用程序可以将特定 dGPU 的 UUID 设置为NvSciBufGeneralAttrKey\_VidMem\_GpuId 的密钥，NvSciBuf 从该特定 的dGPU 分配缓冲区

一个简单的流，单个缓冲区

使用 NvSciBuf 和 NvSciSync 从 NvMedia 控制的相机拍摄图像并将图像发送到 CUDA 进行处理



Producer Sync Object生产的同步对象

Consumer Sync Object 消费的同步对象

Stream流循环举例

对于每一帧数据，应用程序指示 NvMedia 将图像写入缓冲区，然后发出指示写入完成时间的栅栏。CUDA 被指示在继续任何后续操作之前等待该围栏，然后将处理帧的命令发送到 CUDA。最后，CUDA 被告知生成自己的围栏，指示其所有操作何时完成。此围栏被反馈给 NvMedia，后者在开始将下一个图像写入缓冲区之前等待它。缓冲区的“所有权”在生产者和消费者之间来回循环

while (!done)

{

    NvMedia2DComposeresult result;

    NvMedia2DComposeParameters params;

    NvMedia2DGetComposeParameters(nvmedia2D, &params);

    /\* Generate a fence when rendering finishes \*/

    NvMedia2DSetNvSciSyncObjforEOF(nvmedia2D, params, nvmediaToCudaSync);

    /\* Instruct NvMedia pipeline to wait for the fence from CUDA \*/

    NvMedia2DInsertPreNvSciSyncFence(nvmedia2D, params, cudaToNvmediaFence)

     /\* Generate NvMedia image \*/

    NvMedia2DSomeRenderingOperation( ...);

    NvMedia2DCompose(nvmedia2D, params, &result);

    NvMedia2DGetEOFNvSciSyncFence(nvmedia2D, result, &nvmediaToCudaFence);

    /\* Instruct CUDA pipeline to wait for fence from NvMedia \*/

    CUDA\_EXTERNAL\_SEMAPHORE\_WAIT\_PARAMS cudaWaitParams;

    cudaWaitParams.params.nvSciSync.fence = (void\*)&nvmediaToCudaFence;

    cudaWaitParams.flags = 0;

    cudaWaitExternalSemaphoresAsync(&nvmediaToCudaSem, &cudaWaitParams, 1, cudaStream);

    /\* Process the frame in CUDA \*/

    cudaSomeProcessingOperation(..., cudaBuffer, ...);

    /\* Generate a fence when processing finishes \*/

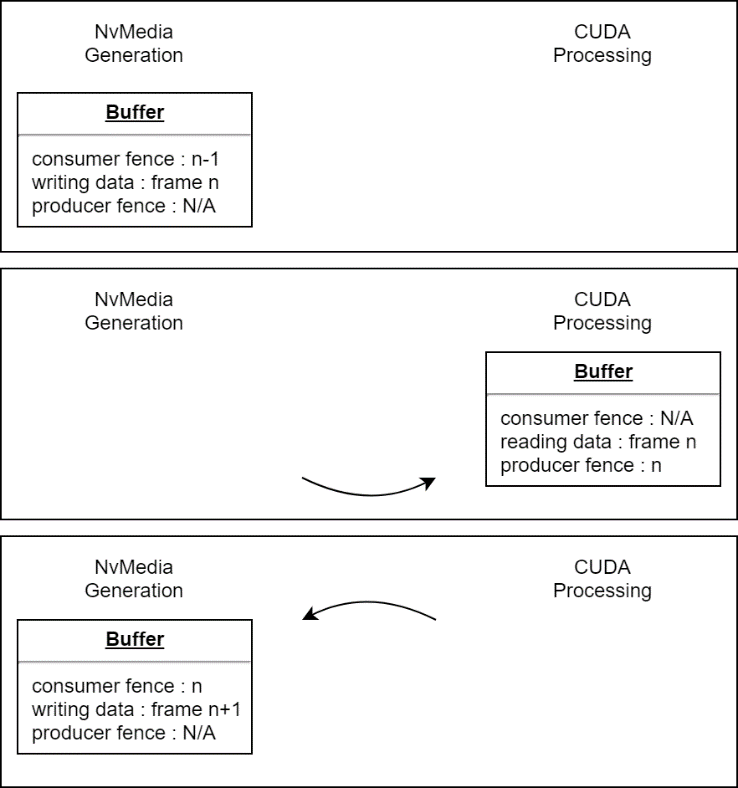
    CUDA\_EXTERNAL\_SEMAPHORE\_SIGNAL\_PARAMS cudaSignalParams;

    cudaSignalParams.params.nvSciSync.fence = (void\*)&cudaToNvmediaFence;

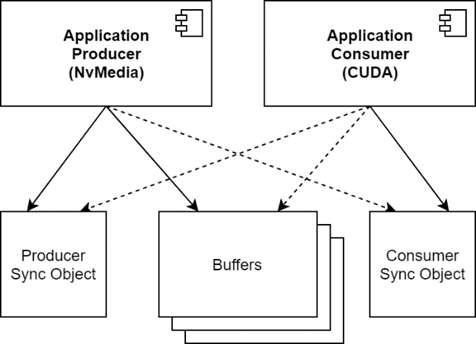
    cudaSignalParams.flags = 0;

    cudaSignalExternalSemaphoresAsync(&cudaToNvmediaSem, &cudaSignalParams, 1, cudaStream);

}



Multiple Buffers



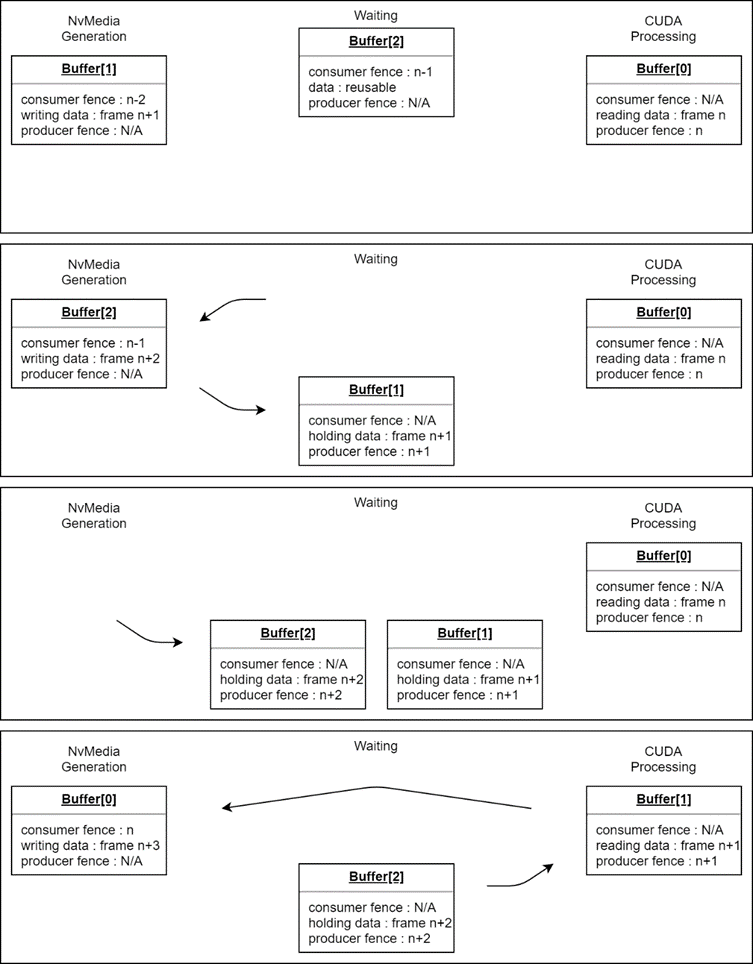
NvMedia 和 CUDA 使用 NVIDIA 硬件的不同部分。

举例中有：计算机硬件在2D硬件工作期间处于空闲状态，2D硬件在计算处工作期间处于空闲状态。通过分配一个或多个附加缓冲区并在它们之间循环，创建帧队列，可以使管道更高效。

这样，CUDA 可以处理一个图像，而 NvMedia 准备下一个图像。不需要其他同步对象，但必须为每个帧生成一个栅栏，因此必须跟踪哪个围栏与每个缓冲区的当前内容相关联。一旦流有多个可用的缓冲区，就可以考虑几种不同的可能操作模式。

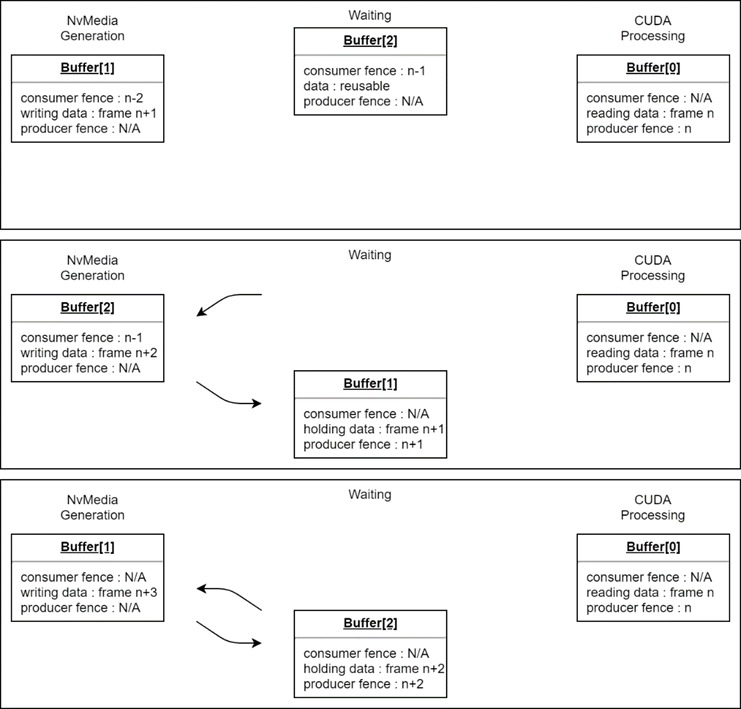
FIFO Mode

排队等候，先进先出



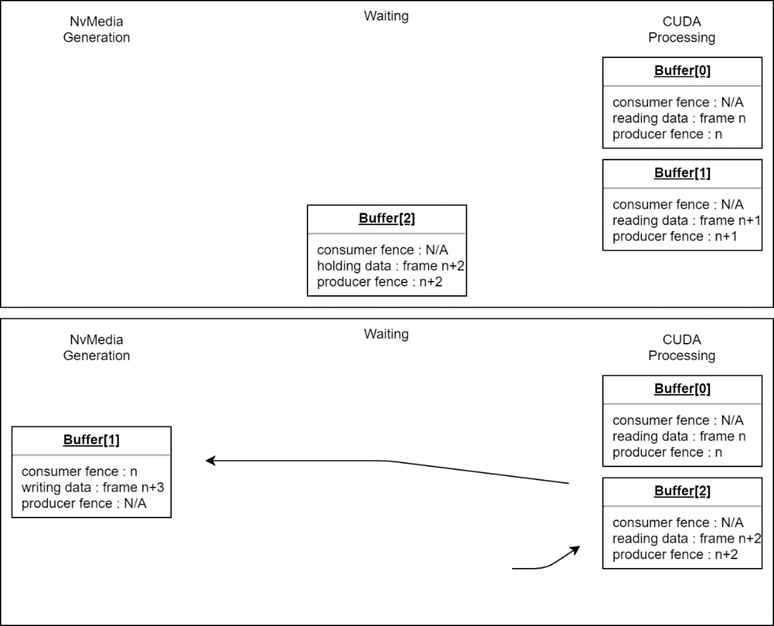
Mailbox Mode

生产和消费的顺序随时可能发生变化，用邮箱的形式通知。



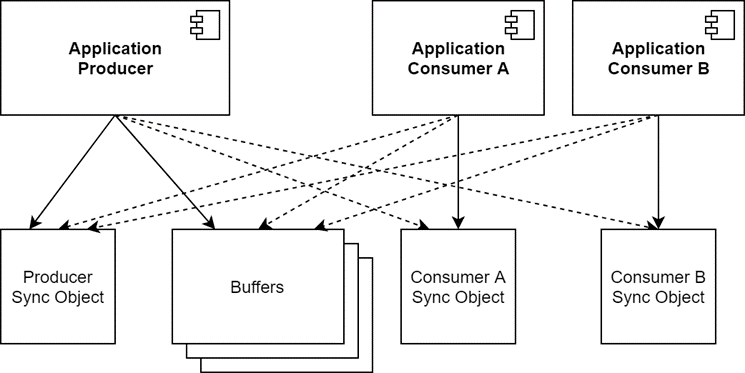
Multiple Acquired Frames

某些处理算法必须比较序列中多个帧的数据。在这种情况下，使用者可以同时持有多个缓冲区。这些缓冲区可以按到达以外的顺序释放以供重用

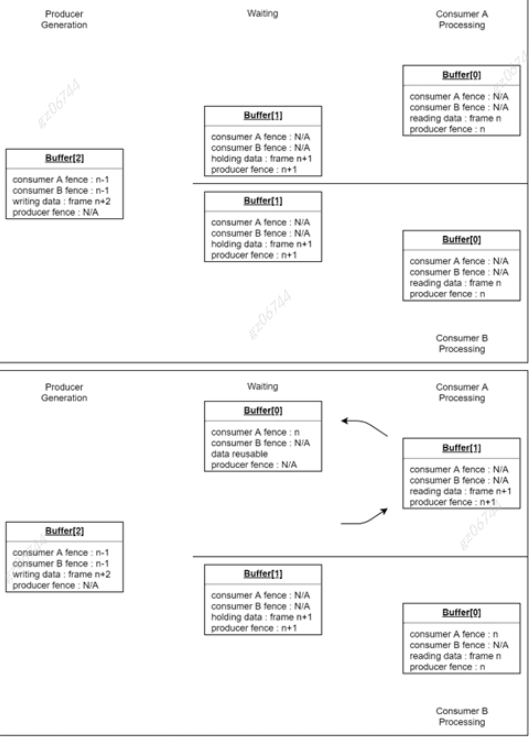
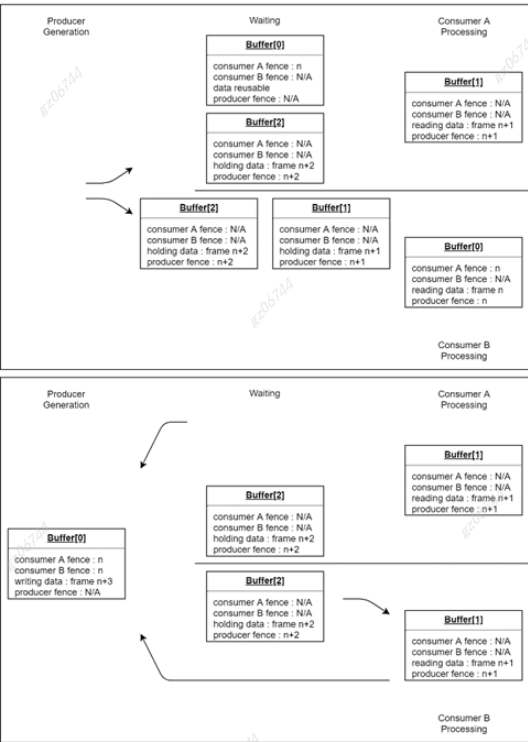


Multiple Consumers

在某些情况下，必须将单个生产者的输出发送给多个使用者。这意味着在初始化期间，必须收集来自所有使用者的需求，并且必须将所有使用者的同步对象映射到生产者中。



在流式处理期间，系统必须跟踪每个缓冲区相对于所有使用者的状态，并等待来自所有使用者的围栏完成，然后生产者才能重用它。如果一个使用者需要 FIFO 行为，而另一个使用者需要邮箱，情况可能会进一步复杂。它们可能会以不同的顺序返回缓冲区。

→ 

在多使用者的情况下，存在安全性和稳定问题，即如果其中一个使用者应用程序恶意或通过某些错误停止返回数据包以供重新使用，它将使生产者和所有其他使用者停止，因为它们将耗尽新数据有效负载的空间。缓解此风险的一种方法是对允许向不可靠应用程序下游发送的数据包数设置一个可选的上限。

Cross-Application

将生产者和消费者置于不同的应用程序中需要在流程的每一步进行进程间通信。在设置时，必须在端点之间传输所有要求，然后必须在进程之间共享分配的对象。

在流式传输过程中，每次生产者生成帧时，都必须向另一端发送一条消息，让使用者知道它已准备就绪，其中包括围栏。当使用者从帧读取完后，必须在另一个方向发送类似的消息。每个应用程序都必须准备好读取和处理这些消息，以便可以及时进行流式处理。

Chip-to-chip (C2C)

在不同 SOC 上的不同应用程序上运行的生产者和消费者需要 SOC 间通信。数据传输将跨越内存边界。这流式处理将不再是无拷贝操作。除了要求跨应用程序用例，每个帧的缓冲区数据将从 生产者系统到消费者系统。

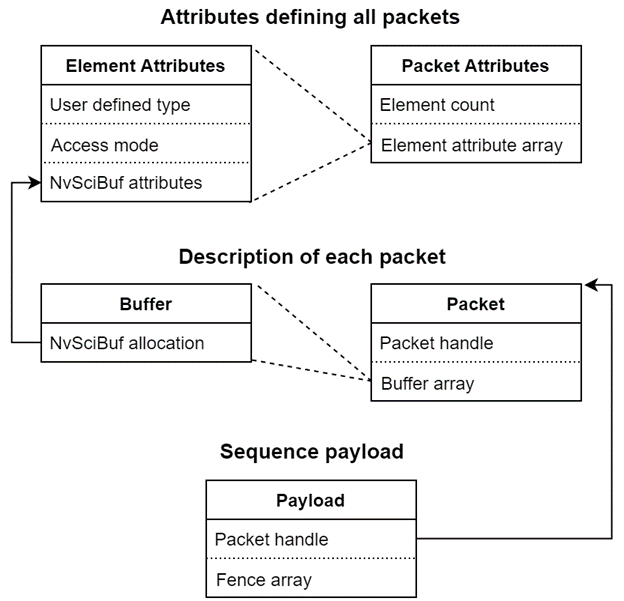
Data Packets

NvSciStream支持缓冲区包含更多的任意的数据

数据包是一组包含图像、Tensorrt、元数据或其他信息的索引缓冲区。

每个流可以使用一个或多个数据包

对于给定的流，所有数据包具有相同数量的元素，并且所有数据包的第 i 个元素分配具有相同的 NvSciBuf 属性。因此，所有数据包都具有统一的内存要求和签名。生产者和使用者一次发送和接收整个数据包，以及相关的围栏，以指示它们包含的有效负载何时准备就绪。



Element Type

在复杂的模块化应用程序套件中，给定的生产者可能知道如何生成许多不同类型的数据，而不同的消费者可能只需要其中的一部分。NvSciStream 提供了帮助应用程序协商给定流所需的数据的机制。顶级系统集成商必须定义一组整数值，以与应用程序套件支持的每种数据类型相关联。任何非零值都可用于这些类型，建议开发人员规划未来的类型和向后兼容性。

类似于这样的枚举

| **数据类型** | **分配的枚举** |
| --- | --- |
| 左前摄像头 | 0x1101 |
| 前置中置摄像头 | 0x1102 |
| 右前摄像头 | 0x1103 |

……

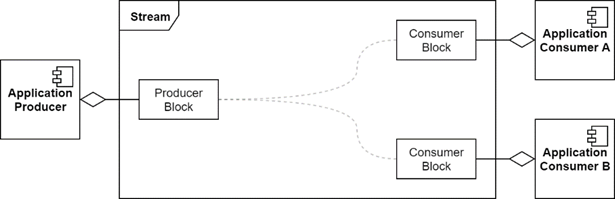
Element Mode

由 NVIDIA 硬件生成和处理的有效载荷数据通常是异步写入和读取的，需要等待围栏才能访问。

但在某些用例中，辅助数据可能由 CPU 同步生成，并且必须在发出处理其余数据的命令之前读取。例如，异步生成图像的相机生产者，但也包括同步生成的元数据字段，其中包含相机的焦距、曝光时间和其他设置。使用者首先同步读取元数据，并在发出命令时使用它包含的值异步处理图像。

Building Block Model

Endpoints



每个流都以一个生产者块开始，以一个或多个使用者块结束。这些被称为流的“端点”

Producer

创建者块提供与应用程序创建者组件的以下交互

具体交互列表这里没有列出来

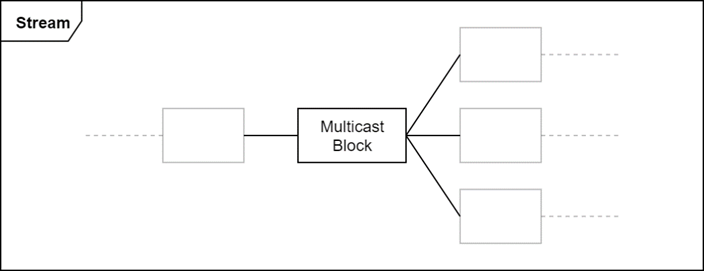
Consumer

使用者块与其相应的应用程序使用者组件提供以下交互：

具体交互列表这里没有列出来

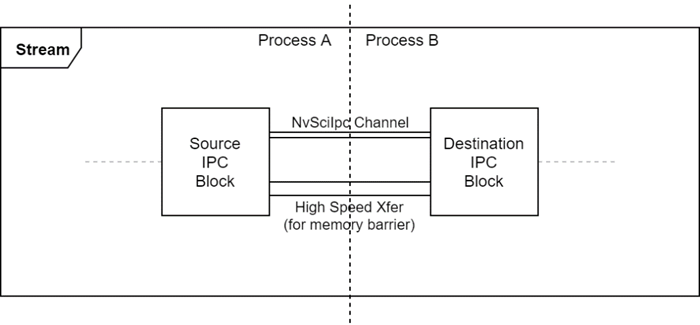
Multicast

当流具有多个使用者时，将使用多广播块来连接单独的管道。该块将生产者的资源和操作分配给消费者，而消费者不知道彼此的存在。它结合了消费者的资源和行为，使它们在生产者面前看起来像一个单一的虚拟消费者。该块不直接提供任何机制来保护其任何消费者免受其他消费者的错误或恶意行为。当存在安全或安保问题时，可以使用其他块将单个使用者管道与其他使用者管道隔离开来。



IPC

当流的端点驻留在单独的进程中时，IPC 块用于弥合差距。它们是成对创建的，上游进程中有一个源 IPC 块，下游进程中有一个目标 IPC 块。必须首先由应用程序使用 NvSciIpc 建立通信，然后将通道端点传递到两个 IPC 块。他们拥有通道的所有权，并使用它来协调需求和资源的交换，并发出数据包可用性的信号。



Memory Sharing IPC

当流的两半访问相同的物理内存时，可以使用内存共享 IPC 块。它们协调两端之间的资源共享，但不需要访问有效载荷数据。

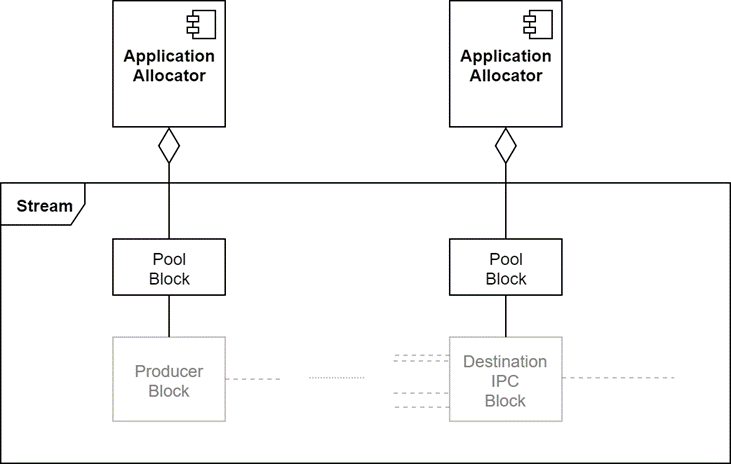
Memory Boundary IPC

当两个进程不共享内存时，必须使用内存边界 IPC 块。

流的每一部分都必须提供一组单独的数据包。当新的有效负载到达时，源块将所有数据传输到目标块，在那里将其复制到新数据包中。可以为此目的建立辅助通信通道。传输完成后，原始数据包将返回上游以供重用，而无需等待使用者完成读取数据，因为它访问一组不同的缓冲区。

POOL

pool用于将数据包引入流并跟踪可供重用的数据包。所有流必须至少有一个池连接到创建者块。如果流包含内存边界，则流的每个部分都需要额外的池，这些部分使用其自己的数据包集，附加到 IPC 块。对于每个池块，都有一个应用程序组件负责根据要求确定数据包布局并分配缓冲区。



分为静态pool和动态pool

静态

静态池提供一组数据包，这些数据包在流的生命周期内保持固定。必须在创建时指定缓冲区的数量，并在设置阶段添加缓冲区

动态

动态池仅在非安全版本中可用。它们允许随时添加和删除缓冲区。这支持生成者可能需要更改某些缓冲区属性（例如数据的大小或像素格式）的用例

[NvSciSync](https://developer.nvidia.com/docs/drive/drive-os/archives/6.0.4/linux/sdk/common/topics/nvsci/NvSciSyncModule12.html)

一个应用程序或一组应用程序所需的同步对象，以及如何使用它们来控制 NVIDIA 硬件执行操作的顺序。

* 指定向同步对象发出信号的硬件组件施加的限制。
* 指定在同步围栏上等待的硬件组件施加的限制。
* 将每组同步对象的信息收集到一个应用程序中，然后由应用程序分配它们。
* 与其他应用程序共享分配的同步对象。
* 将同步对象映射到 UMD (用户模式驱动程序)特定接口。

在 NVIDIA 硬件系统中，通常有多个执行代理同时运行。例如：CPU、GPU 和其他引擎。引擎的任务接口以异步方式运行。CPU 为其准备作业并将其在接口中排队。引擎上可能有多个作业挂起，这些作业在硬件中自动调度。CPU 和应用程序事先不知道作业的执行顺序。这就需要与它们之间的依赖关系进行作业同步，就像处理相同数据的多个引擎的管道一样，最好不应该涉及昂贵的 CPU 干预。

代理： 系统中执行指令的实体。代理可以是 CPU 线程，也可以是硬件引擎。代理是与 NvSciSync 抽象的实际硬件基元交互的实体。NvSciSync的目标是同步代理。

NvSciSync 提供了一个抽象层，用于隐藏在具体情况中使用的同步基元的详细信息。

NvSciSync 最基本的概念之一是同步对象。它抽象化特定同步基元的单个实例。同步对象具有当前状态，可以发出信号。向同步对象发出信号会将其移动到下一个状态。通常，应用程序开发人员将同步对象与必须按相同顺序发生的事件链相关联。

例如，视频输入引擎在生成相机帧后可能始终向同一同步对象发出信号。这样，可以随时检查同步对象以检查哪些帧已经写入。同步对象只能由单个代理发出信号。发出信号（应应用程序请求）的代理称为信号器。

NvSciSync的另一个基本概念是同步围栏。同步围栏与特定同步对象相关联，并包含该对象状态的快照。如果围栏的快照落后于或等于对象的当前状态，则认为栅栏已过期。物体尚未到达其状态的栅栏被称为待定。通常，多个栅栏与单个同步对象相关联，并且可能对应于该对象的不同状态。同步围栏由信号器应用程序生成并与其他人共享。应用程序可以使代理在栅栏上等待。在给定同步对象的上下文中，在同步栅栏上等待的代理称为等待者。

NvSciSyncModule

要使用 NvSciSync，您必须首先打开 NvSciSyncModule。此模块表示为该应用程序创建的库实例，并充当其他 NvSciSync 资源的容器。通常，应用程序中只有一个NvSciSyncModule，但是将所有资源包含在NvSciSyncModule中允许多个线程或其他库以隔离的方式使用NvSciSync。所有其他 NvSciSync 资源在创建时都与 NvSciSyncModule 相关联。

Inter-Application

如果涉及多个进程，则NvSciSync结构的所有通信都应通过NvSciIpc通道进行。每个应用程序都需要打开自己的 Ipc 端点。

Inter-Process Communication

如果应用程序涉及多个进程，则NvSciBuf结构的交换只能通过NvSciIpc通道。每个应用程序都必须打开自己的 NvSciIpc 终结点。

NvSciIpc

NvSciIpc 库为系统中的任何两个实体提供 API，无论它们放置在何处，都可以相互通信

* 在同一进程的不同线程中。
* 在同一进程中。
* 在同一 VM 的不同进程中。
* 在同一 SoC 上的不同 VM 中。
* 或在不同的 SoC 中。

虚拟机内、进程与线程

INTER\_THREAD、INTER\_PROCESS通过POSIX 实现共享

两个虚拟机之间

INTER\_VM通道依靠虚拟机监控程序在两个虚拟机之间设置共享区域详细信息

SOC之间

NVIDIA 软件通信接口，用于通过直接 PCIe 进行芯片到芯片的连接

Chip to Chip Communication

NVIDIA 软件通信接口，通过PCIE连接提供芯片之间的通讯（NvSciC2cPcie），为用户应用程序提供跨Orin的数据交互交换的能力 。

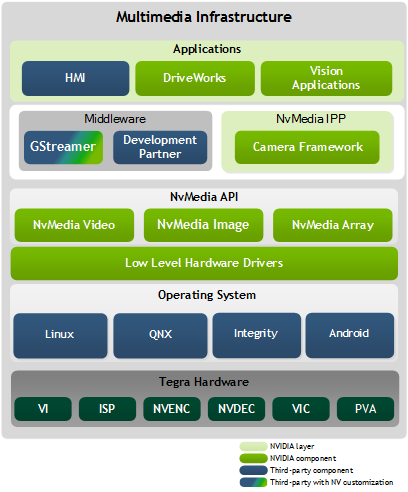
直接 PCIe 连接位于第一个/一个 NVIDIA 之间 DRIVE AGX Orin Devkit 作为 PCIe 根端口与第二个/另一个 NVIDIA DRIVE AGX 一起使用 Orin Devkit 作为 PCIe 端点。

|  |  |
| --- | --- |
| **Channel** | NvSciIpc 通道连接允许双向交换 正好两个 NvSciIpc 终结点之间的固定长度消息. |
| **Endpoint** | 使用 NvSciIpc 通道与之通信的软件实体 另一个软件实体. |
| **Frame** | 由沿 NvSciIpc 通道从 NvSciIpc 通道的两个 NvSciIpc 端点之一到 其他 NvSciIpc 端点的字节数据 |
| **Frame Size** | 通过 NvSciIpc 通道交换的每条消息的大小（以字节为单位），每个 NvSciIpc 通道可能具有不同的帧大小 |
| **Frame Count** | 沿 NvSciIpc 通道在每个方向上排队等待传输的，且可以同时使用的 NvSciIpc 帧的最大数量 |
| **Backend** | NvSciIpc 后端为特定实现 NvSciIpc 功能 NvSciIpc 频道类。例如，对于仅限于SoC的NvSciIpc通信， 有五种不同类别的 NvSciIpc 通道，具体取决于 NvSciIpc 通道支持的 NvSciIpc 端点之间的分离 |
| **Endpoint Handle** | 传递给所有 NvSciIpc 通道的抽象数据类型 通信 |
| **Channel Reset** | 定义 NvSciIpc 之一通信的突然结束 端点。在重置的情况下，不允许通过 NvSciIpc 通道进行通信，直到 两个终结点都重置其内部状态，并准备好进行通信。NvSciIpc 依靠其后端来实现通道重置机制。它可能需要 来自两个端点的合作，其中两个端点必须等待确认 另一个已重置其本地状态。 |
| **Notification** | 沿 NvSciIpc 通道的异步信号，通过该通道 NvSciIpc 通道的两个端点指示另一个 NvSciIpc 端点 可能存在后一个 NvSciIpc 端点要处理的事件。 |

## 4.6 NvMedia

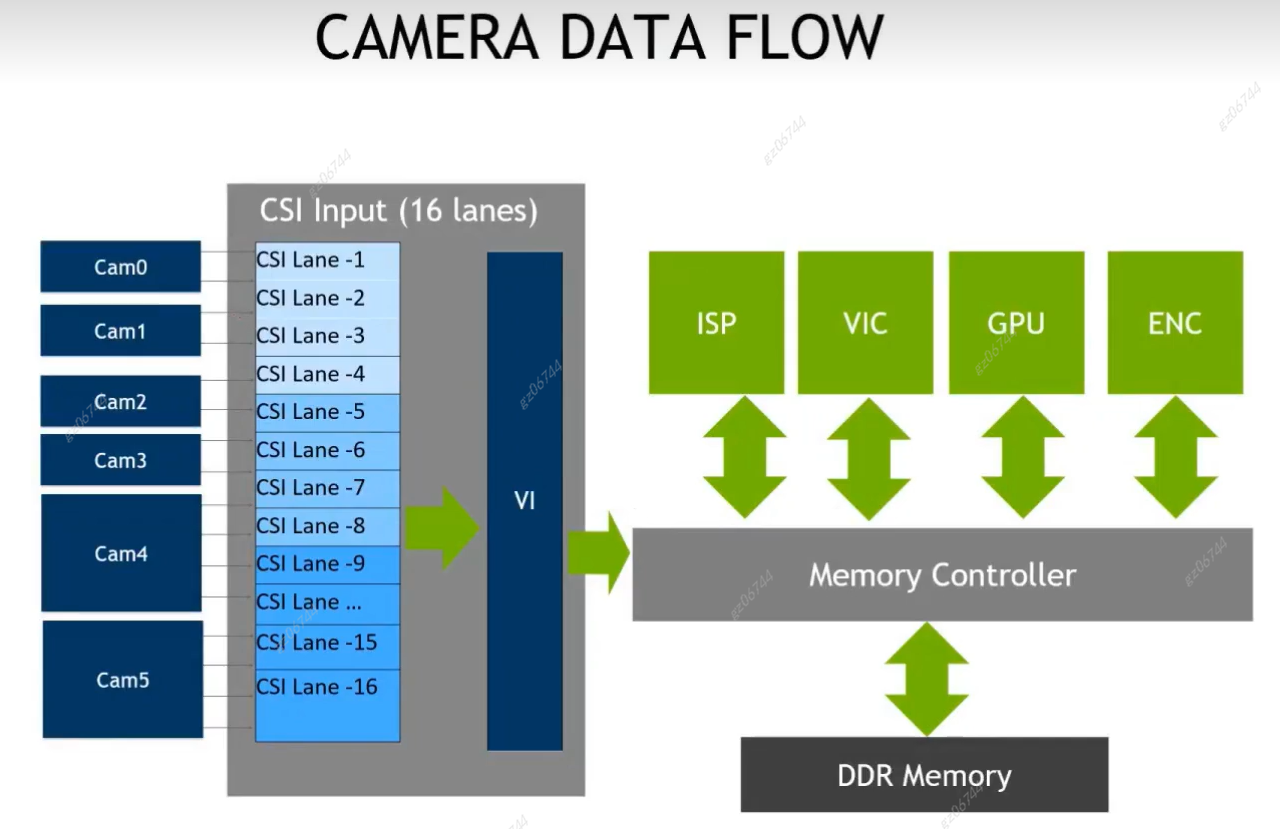
NvMedia提供了多媒体应用支持，包括多摄像头输入，ISP，sensor控制。支持多种图像格式，例如YUV、RGB还有RAW。以及各种为图像捕获，处理，加速，编码，显示等提供的库。另外，NvMedia还可以将图像和视频传递给OpenGL ES或者CUDA，满足开发者的其它需求。

* VI（Video Input）,控制接收MIPI CSI图像
* ISP，各种camera输入图像处理，包括3A、demosaic、坏点去除，甚至可以修改像素值。
* Codec，各种视频协议的压缩和解压缩
* VIC（Video Interlace Compositor）, 主要包括deinterlace等视频格式转换（个人认为已经不太需要deinterlace处理）
* PVA（Programmable Vision Accelerator）,通过API加速各种计算机视觉算法，目前很多自驾平台都提供了此类加速以满足低功耗的需求



|  |  |
| --- | --- |
| **Tegra Hardware** | **Description** |
| Video Input (VI) | 从摄像机接收 CSI 数据。 |
| Image Signal Processor  (ISP) | 根据从图像传感器捕获的图像数据生成已处理的图像。例如，它可以进行像素级更改，例如反转像素位、自动曝光和白平衡校正。 |
| NVIDIA Encoder (NVENC) | 将原始图像数据转换为受支持的图像格式之一。 |
| NVIDIA Decoder (NVDEC) | 将编码的图像数据转换为原始图像数据。 |
| Video Interlace Compositor (VIC) | 转换视频数据以进行反隔行、合成和格式转换。 |
| Programmable Vision Accelerator (PVA) | 通过 NvMediaVPI API 加速计算机视觉算法。 |

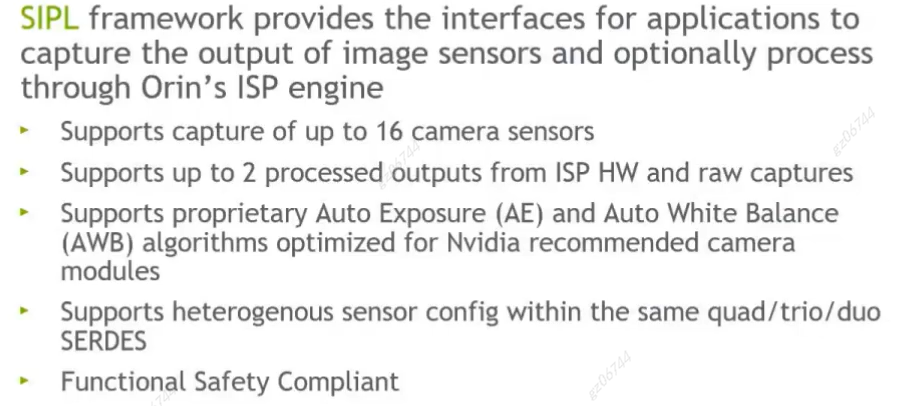
Video surface处理

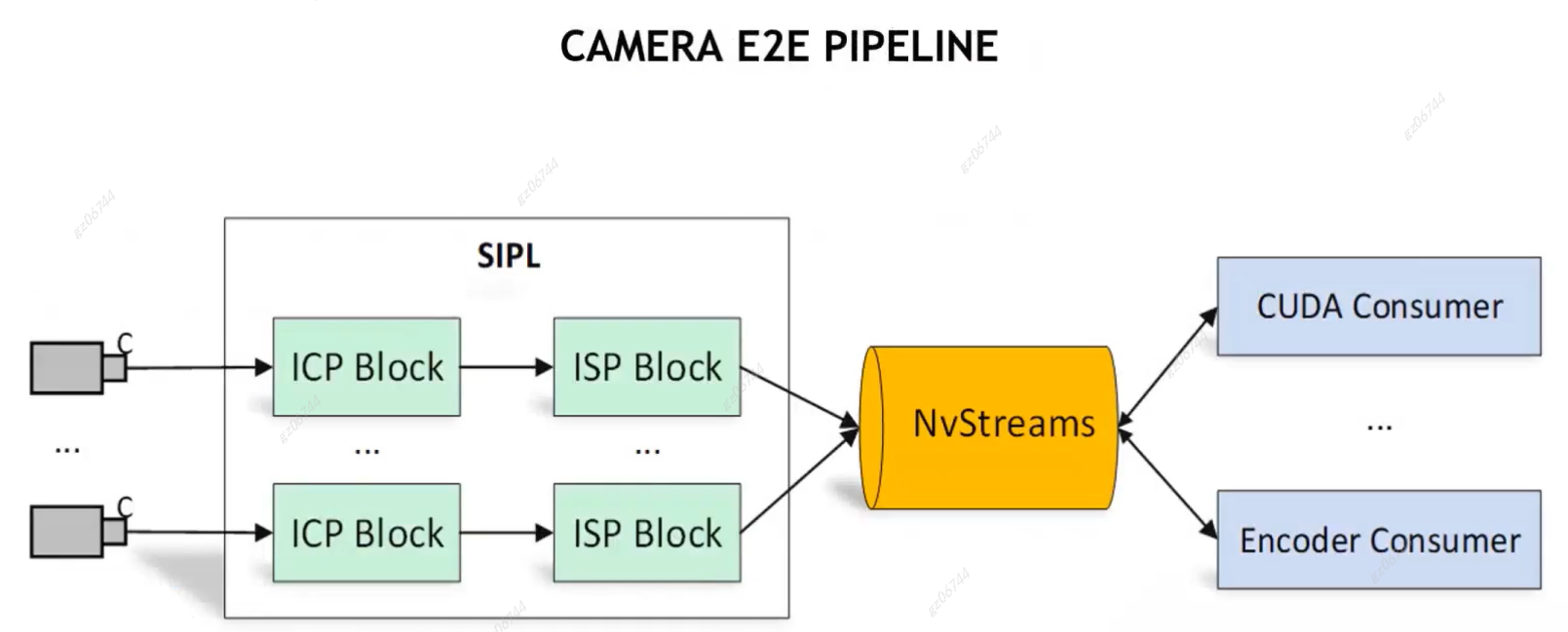


SIPL

NvMedia SIPL框架提供了一个简化的API来捕获图像传感器的输出连接到 Orin™ 平台

Safe Input Processng Library



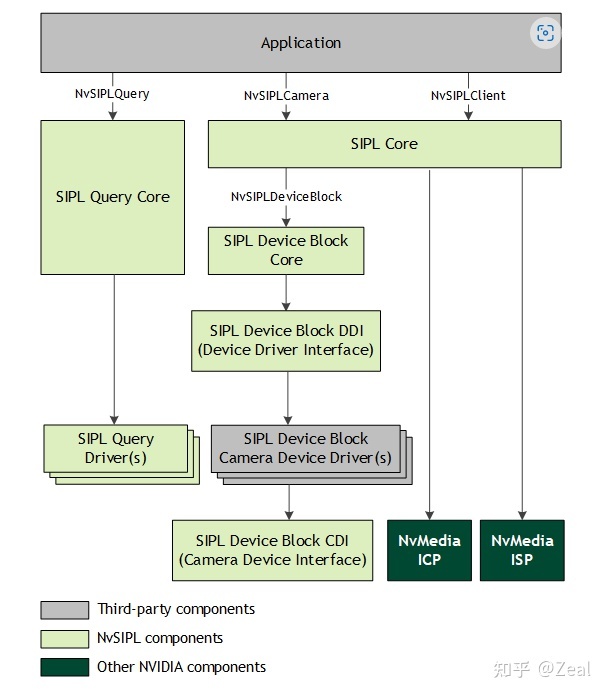


文档中提到的使用SIPL的步骤，这里就没有详细的列出来

SIPL框架是DriveOS中NvMedia中一个组成部分，抽象了

1）、与相机相关硬件设备比如相机、EEPROM、Serializer、Deserializer等的操作；

2）、提供对平台的编程接口来捕捉和处理图片（ISP：Image Signal Processing）分层  
 SIPL提供Query组件管理设备信息。Query组件保存该库支持的所有的硬件设备和平台设置的信息库。SIPL Core提供对相机操作的抽象接口



如何做到兼容多相机，SIPL要求相机模组提供者按照要求，集成Device Block的Device Driver Interface（DDI）接口，实现自己的相机驱动，以此来保证了SIPL软件栈可以适配多种相机。

## 4.7 Logging

DRIVE OS 有一个日志记录组件，允许来自多个 DRIVE OS 组件的日志 存储（或）从 DRIVE AGX 硬件流式传输出来日志。目前，支持仅适用于 DRIVE OS Linux Guest VM 中的应用程序

# 5. System Software Components and Interfaces

## 5.1 Calculating GPIO Index in Linux

介绍了怎么去计算GPIO的索引

## 5.2 Persistence Across Bootburn Flashing Using Persistent Partition

作为安全登录功能的一部分，将一些需要一直保存的数据通过Persistent Partition存储起来，即使是在bootburn flashing过程中也不会被刷掉

那其中有哪些数据是会一直保存不被刷掉的？

1、在 SSH 连接设置期间唯一标识系统的 SSH 主机密钥。

2、用户帐户元数据文件，如 、 等，其中包含用户的列表、用户的哈希密码等

为什么需要跨刷写的持久数据

如上一节所示，持久性数据用于存储基于标识的数据。例如，引导刻录闪烁后存在的 SSH 密钥表示系统的身份尚未受到损害。

同样，用户帐户数据显示系统中最终用户的身份。

永久分区的工作流

重置永久分区的方法

## 5.3 Linux Filesystems

基础层是Canonical Ubuntu Base Filesystem。 这是来自 Canonical 的文件系统，不会更改，但必须是初始的 基于文件系统构建在上面。

下一层名为driveos-core-rfs，旨在包含特定于汽车的 Debian 软件包以及驱动程序、库、工具、固件、脚本和任何其他 核心功能的必要文件。此文件系统通常提供给并用于 生产系统。

下一层名为driveos-oobe-rfs，是面向开发人员的开箱即用控制台。它 包含特定于汽车的示例、文档和开发人员友好的工具。这 文件系统旨在供开发人员为开发人员使用

最后一层名为driveos-oobe-desktop-rfs，是一个开箱即用的GUI桌面 开发人员的经验。它包含汽车GUI桌面界面，特定示例， 文档和开发人员友好的工具。此文件系统旨在由 软件开发开发人员。

| **文件系统** | **要下载哪个文件** | **目的** |
| --- | --- | --- |
| Driveos-oobe-ubuntu-20.04-RFS | nv-driveos-linux-driveos-oobe-ubuntu-20.04-rfs-<release>-<DRIVEOS\_GCID>\_<release>-<DRIVEOS\_GCID>\_amd64.deb | 旨在供开发人员用作开发或参考 文件系统。  包含 NVIDIA 汽车示例应用程序及其相应的 文件。 |
| driveos-oobe-desktop-ubuntu-20.04-rfs | nv-driveos-linux-driveos-oobe-desktop-ubuntu-20.04-rfs-<release>-<DRIVEOS\_GCID>\_<release>-<DRIVEOS\_GCID>\_amd64.deb | 旨在供开发人员用作开发或参考 文件系统。  包含 NVIDIA 汽车 GUI 桌面界面，示例 应用程序以及相应的文档。 |
| Driveos-core-ubuntu-20.04-RFS | nv-driveos-linux-driveos-core-ubuntu-20.04-rfs-<release>-<DRIVEOS\_GCID>\_<release>-<DRIVEOS\_GCID>\_amd64.deb | 旨在供生产系统使用。  包含 NVIDIA 汽车驱动程序、库、工具、固件、 脚本和核心功能所需的其他文件。 |

## 5.4 Ubuntu Setup (Target)

## 5.5 DRIVE OS Linux Filesystems Customization Quick Start Page

## 5.6 CopyTarget

**一个文件复制的工具**

复制目标工具可靠地将文件从源位置复制到目标位置 位置，同时维护文件元数据，包括所有权和权限。

复制目标支持以下操作：

* 复制文件
* 创建目录
* 更新文件元数据
* 更新目录元数据
* 删除文件
* 删除空目录
* 创建符号链接

## 5.7 Nvidia build-kit

## 5.8 CAN Driver

TegraCAN在Linux内核中提供了两个SocketCAN接口。SocketCAN接口类似于Linux内核中的网络接口

## 5.9 Kernel Modules and Limitations

NVIDIA 修改了 Linux defconfig 以构建许多驱动程序作为模块，从而减少了启动 时间。

## 5.10 Crypto Interface

SE 硬件模块功能在 SE 服务器中实现，SE 服务器的接口在虚拟 SE 驱动程序（tegra-hv-vse 驱动程序）中实现。

## 5.11 I2C Settings

Linux SDK内核支持标准 I2C 内核接口

## 5.12 PCIe Retimer

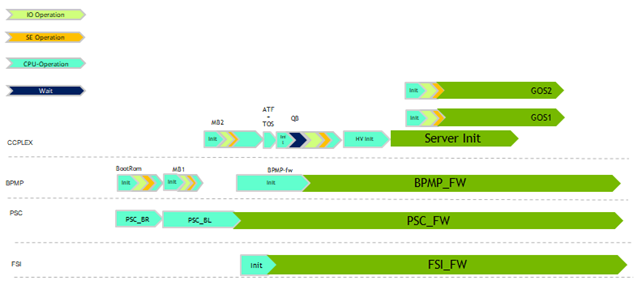
这个retimer是用来做啥的？

## 5.13 Suspend to RAM / SC7

SC7 是一种 SOC 电源状态，其中 Tegra 处于低功耗模式，DRAM 处于自刷新状态 模式

# 6.Bootloader Programming

## 6.1 [Understanding the Boot Flow Process](https://developer.nvidia.com/docs/drive/drive-os/archives/6.0.4/linux/sdk/common/topics/bootloader_setup/UnderstandingtheBootFlowProcess1.html)



Notice that the BPMP processor runs:

* BootROM
* MB1
* BPMP-FW

Notice that the CCPLEX processor runs:

* MB2

Notice that the PSC processor runs:

* PSC-BootROM
* PSC\_BL

## 6.2 [Customizing Quickboot](https://developer.nvidia.com/docs/drive/drive-os/archives/6.0.4/linux/sdk/common/topics/bootloader_setup/CustomizingQuickboot1.html)

CPU 的快速启动启动加载程序：

* 提供调试模式。
* 包括适用于所有平台的快速启动的单个版本。所有特定于平台的配置必须通过 DTB 文件进行。
* 针对嵌入式和汽车应用进行了优化。

引导阶段

快速启动从分区加载内核映像 闪光。根据图像是否被压缩，它会执行动态解压缩 内核映像和引导相同。KERNEL\_PRIMARY

压缩

快速启动支持解压缩算法：

* LZ4：以更快的解压时间执行良好的压缩。

默认解压方案为 LZ4。

## 6.3 [Using the Bootloader Recovery Mechanism](https://developer.nvidia.com/docs/drive/drive-os/archives/6.0.4/linux/sdk/common/topics/bootloader_setup/UsingtheBootloaderRecoveryMechanism1.html)

引导加载程序包括:

* BootROM
* Microboot 1 (MB1)
* Microboot 2 (MB2)
* Quickboot (QB)
* Hypervisor which includes:
  + Partition Loader (PL)

这些组件加载其他固件组件

* 启动映像
* 分区映像
* 其他固件

## 6.4 [Ratcheting](https://developer.nvidia.com/docs/drive/drive-os/archives/6.0.4/linux/sdk/common/topics/bootloader_setup/Ratcheting1.html)

由于系统缺陷，引导代码可能会更新。也可能在软件中发现安全漏洞，并在较新版本中修复。攻击者可以利用旧版本软件中的已知漏洞来利用系统安全性。因此，必须禁止过时的二进制文件在芯片上运行。

尽管如此，那些过时的二进制文件曾经拥有有效的凭据来通过安全启动身份验证。因此，需要一种版本控制机制来阻止旧二进制文件通过身份验证，以便攻击者无法利用这些二进制文件来利用系统缺陷。回滚预防在引导期间根据存储在保险丝中的已知硬件版本检查软件二进制文件的版本，并在发现软件版本不可接受的旧版本时阻止二进制文件继续。[**Ratcheting**](https://developer.nvidia.com/docs/drive/drive-os/archives/6.0.4/linux/sdk/common/topics/bootloader_setup/Ratcheting1.html)是 NVIDIA 的回滚预防机制。

## 6.5 [Ratchet Checks](https://developer.nvidia.com/docs/drive/drive-os/archives/6.0.4/linux/sdk/common/topics/bootloader_setup/ratchet_checks.html)

## 6.6 [Rachet Fuse Programming](https://developer.nvidia.com/docs/drive/drive-os/archives/6.0.4/linux/sdk/common/topics/bootloader_setup/rachet_fuse_programming.html)

## 6.7 [Passing Ratchet Status to Guest OSes](https://developer.nvidia.com/docs/drive/drive-os/archives/6.0.4/linux/sdk/common/topics/bootloader_setup/passing_ratchet_status_to_guest_oses.html)

## 6.8 [Lock Fuse Burning](https://developer.nvidia.com/docs/drive/drive-os/archives/6.0.4/linux/sdk/common/topics/bootloader_setup/lock_fuse_burning.html)

## 6.9 [Passing Customer Data to Guest OSes](https://developer.nvidia.com/docs/drive/drive-os/archives/6.0.4/linux/sdk/common/topics/bootloader_setup/passing_customer_data_to_guest_OS.html)

## 6.10 [Configuring MB1 Boot Configuration Table](https://developer.nvidia.com/docs/drive/drive-os/archives/6.0.4/linux/sdk/common/topics/bootloader_setup/ConfiguringMB1BootConfigurationTable1.html)