本期视频讲解内容：

1、驱动的一些基础知识

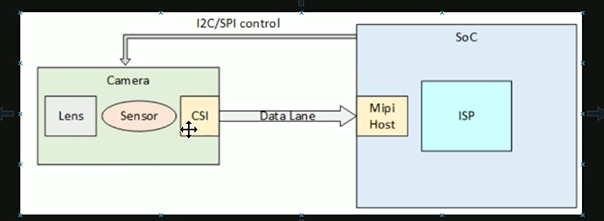
2、v4l2整体架构的介绍，结合开发板rv1126,捋清楚/dev/media /dev/video0 /dev/v4l-sebdev0

3、gc2053sensor的数据手册的解读，以及sensor源码的初步窥探，sensor驱动模块的初始化

4、mipi协议里面的cci接口(camera control interface),梳理一下cci(i2c)和cci(i3c)数据操作特性

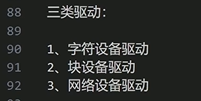
回顾上节课：

Camera数据的转发流程：



sensor采集的一个raw数据，其底层会通过MIPI的物理层去做传输，在传输到MIPI的CSI一个解析的接口，最终通过MIPI再传到这个ISP做一个处理。

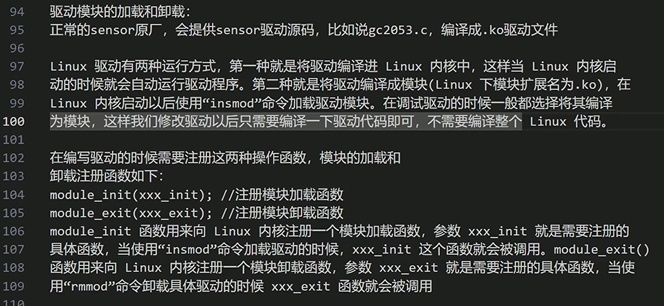




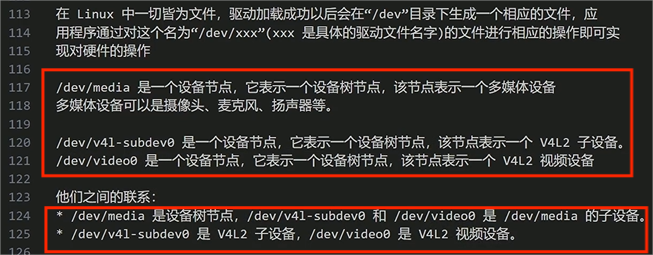
字符设备：对设备硬件操作是按字节去操作。如LED灯、i2c、spi等。

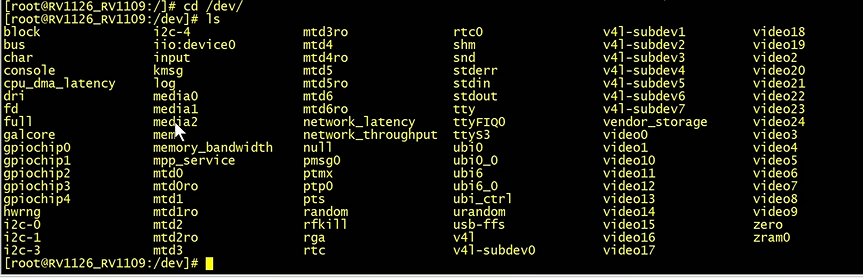
块设备：SD卡、Flash等

网络设备：可以上网的设备：如对网卡的操作。



这里驱动加载指的主要是：加载的.ko文件。

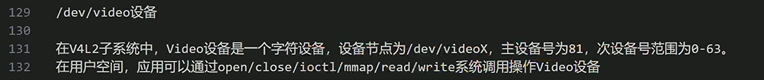




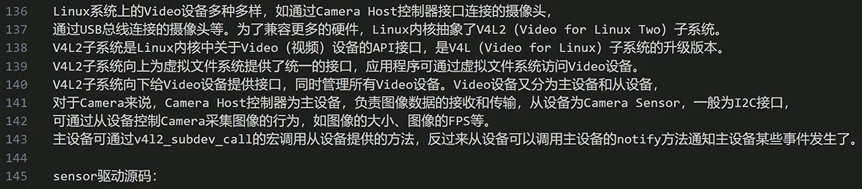
Media:多媒体的驱动，是总开关，是设备树节点。

Video:是视频设备，比如我们的摄像头（USB、MIPI、DVP）的设备节点。

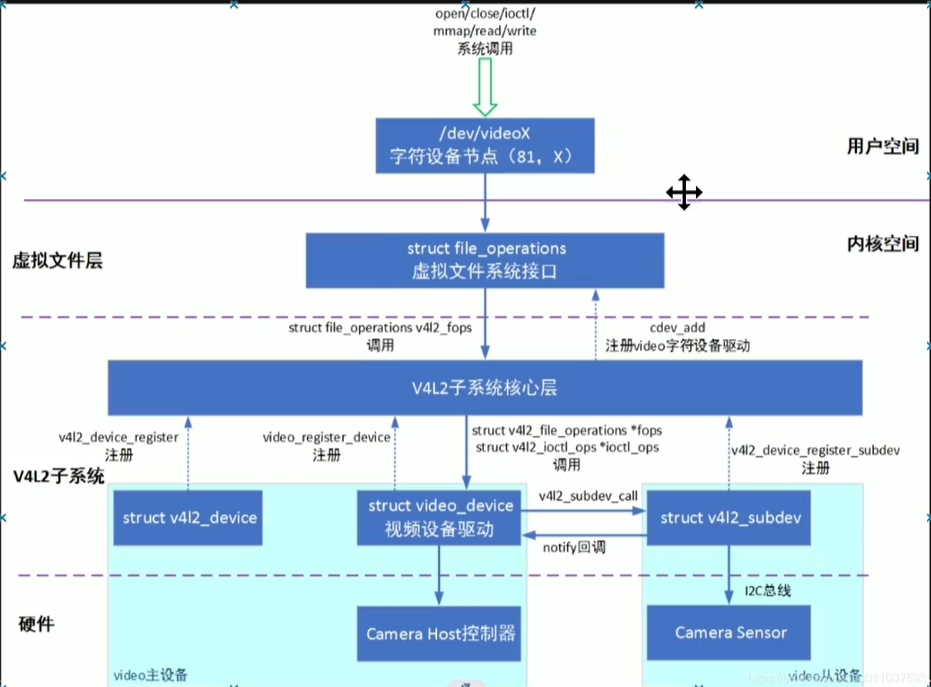
V4l2-subdev:是v4l2子设备，可以看成：video节点包含在v4l2子节点下面。

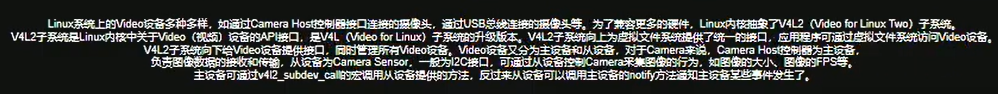


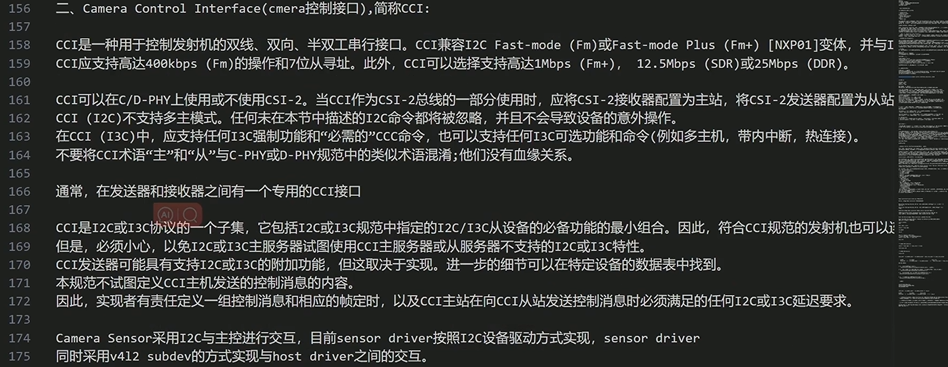
Alsa是主音频控制的V4l的整体架构：

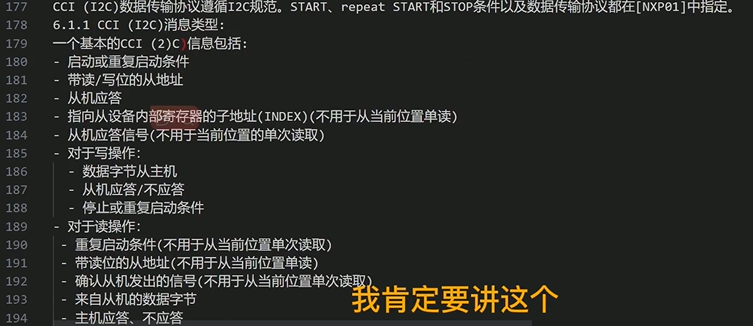
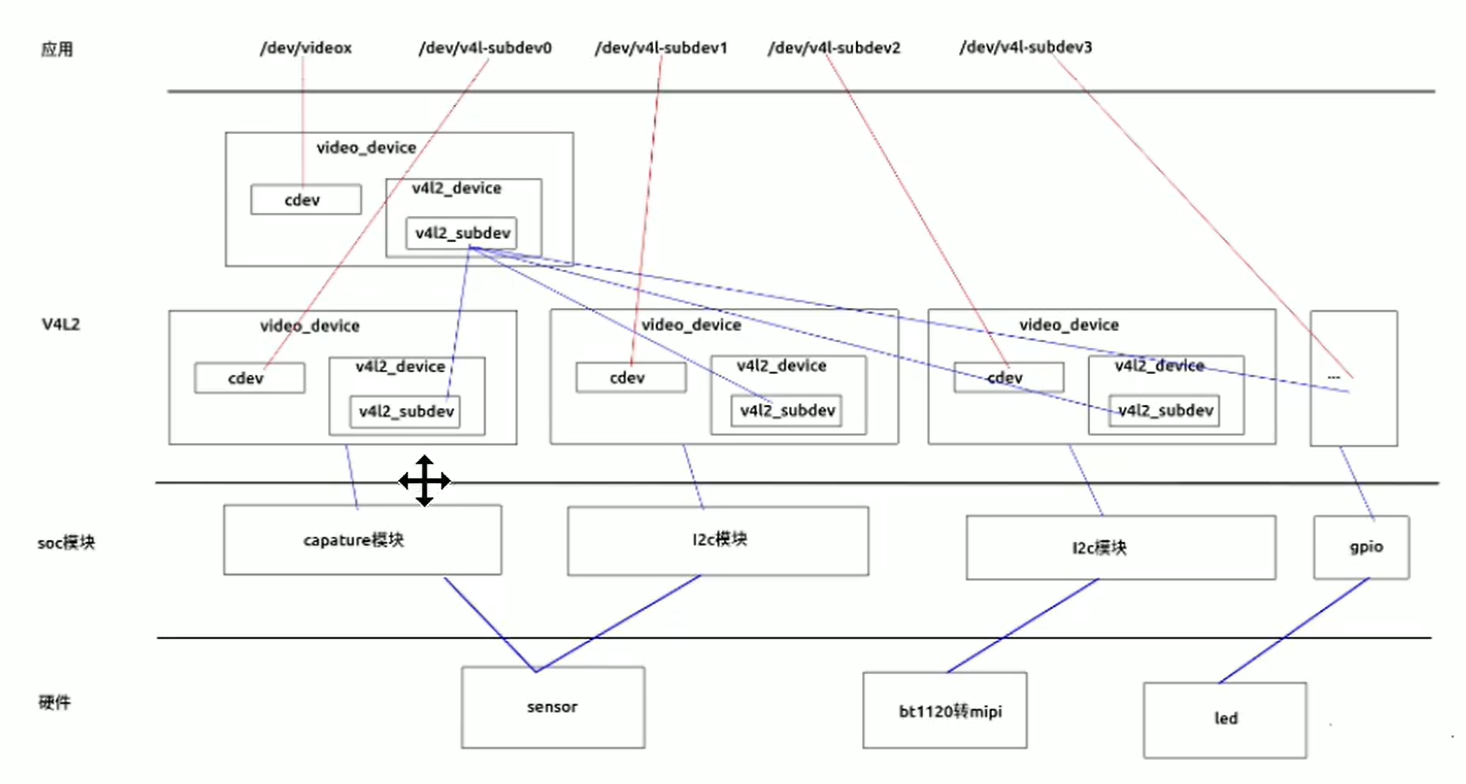


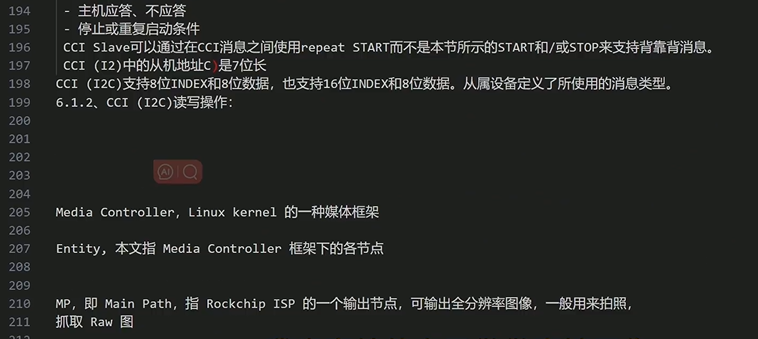
gc2053\_probe











该控制接口是两线双向传输，半双工串行接口来控制。

CCI兼容IIC fast mode模式。同时支持I3C接口的一个单数据和双数据率的一个协议标准。

I2C单数据传输没有I3C多，并且其不支持多主机模式。

第四期内容：

本期视频讲解内容：

1、驱动的一些基础知识

2、v4l2整体架构的介绍，结合开发板rv1126,捋清楚/dev/media /dev/video0 /dev/v4l-sebdev0

3、gc2053sensor的数据手册的解读，以及sensor源码的初步窥探，sensor驱动模块的初始化

4、mipi协议里面的cci接口(camera control interface),梳理一下cci(i2c)和cci(i3c)数据操作特性

1、驱动调试准备工作：

-1、sensor数据手册，也就是sensor datasheet(模组厂或者sensor FAE提供)

- 2、模组规格书(镜头fov，光圈，畸变等信息)--------模组厂提供

-3 、sensor初始化寄存器序列(sensor FAE提供)

-4、sensor曝光(gate/time)配置公式(sensor datasheet or sensor FAE)

1、首先datasheet ，原厂给你提供，一般在公司里面开发的话

三类驱动：

1、字符设备驱动

2、块设备驱动

3、网络设备驱动

驱动模块的加载和卸载：

正常的sensor原厂，会提供sensor驱动源码，比如说gc2053.c，编译成.ko驱动文件

Linux 驱动有两种运行方式，第一种就是将驱动编译进 Linux 内核中，这样当 Linux 内核启

动的时候就会自动运行驱动程序。第二种就是将驱动编译成模块(Linux 下模块扩展名为.ko)，在

Linux 内核启动以后使用“insmod”命令加载驱动模块。在调试驱动的时候一般都选择将其编译

为模块，这样我们修改驱动以后只需要编译一下驱动代码即可，不需要编译整个 Linux 代码。

在编写驱动的时候需要注册这两种操作函数，模块的加载和

卸载注册函数如下：

module\_init(xxx\_init); //注册模块加载函数

module\_exit(xxx\_exit); //注册模块卸载函数

module\_init 函数用来向 Linux 内核注册一个模块加载函数，参数 xxx\_init 就是需要注册的

具体函数，当使用“insmod”命令加载驱动的时候，xxx\_init 这个函数就会被调用。module\_exit()

函数用来向 Linux 内核注册一个模块卸载函数，参数 xxx\_exit 就是需要注册的具体函数，当使

用“rmmod”命令卸载具体驱动的时候 xxx\_exit 函数就会被调用

在 Linux 中一切皆为文件，驱动加载成功以后会在“/dev”目录下生成一个相应的文件，应

用程序通过对这个名为“/dev/xxx”(xxx 是具体的驱动文件名字)的文件进行相应的操作即可实

现对硬件的操作

/dev/media 是一个设备节点，它表示一个设备树节点，该节点表示一个多媒体设备

多媒体设备可以是摄像头、麦克风、扬声器等。

/dev/v4l-subdev0 是一个设备节点，它表示一个设备树节点，该节点表示一个 V4L2 子设备。

/dev/video0 是一个设备节点，它表示一个设备树节点，该节点表示一个 V4L2 视频设备

他们之间的联系：

\* /dev/media 是设备树节点，/dev/v4l-subdev0 和 /dev/video0 是 /dev/media 的子设备。

\* /dev/v4l-subdev0 是 V4L2 子设备，/dev/video0 是 V4L2 视频设备。

/dev/video设备

在V4L2子系统中，Video设备是一个字符设备，设备节点为/dev/videoX，主设备号为81，次设备号范围为0-63。

在用户空间，应用可以通过open/close/ioctl/mmap/read/write系统调用操作Video设备

alsa

Linux系统上的Video设备多种多样，如通过Camera Host控制器接口连接的摄像头，

通过USB总线连接的摄像头等。为了兼容更多的硬件，Linux内核抽象了V4L2（Video for Linux Two）子系统。

V4L2子系统是Linux内核中关于Video（视频）设备的API接口，是V4L（Video for Linux）子系统的升级版本。

V4L2子系统向上为虚拟文件系统提供了统一的接口，应用程序可通过虚拟文件系统访问Video设备。

V4L2子系统向下给Video设备提供接口，同时管理所有Video设备。Video设备又分为主设备和从设备，

对于Camera来说，Camera Host控制器为主设备，负责图像数据的接收和传输，从设备为Camera Sensor，一般为I2C接口，

可通过从设备控制Camera采集图像的行为，如图像的大小、图像的FPS等。

主设备可通过v4l2\_subdev\_call的宏调用从设备提供的方法，反过来从设备可以调用主设备的notify方法通知主设备某些事件发生了。

二、Camera Control Interface(cmera控制接口),简称CCI:

CCI是一种用于控制发射机的双线、双向、半双工串行接口。CCI兼容I2C Fast-mode (Fm)或Fast-mode Plus (Fm+) [NXP01]变体，并与I3C [MIPI03]接口的单数据速率(SDR)或双数据速率(DDR)协议兼容。

CCI应支持高达400kbps (Fm)的操作和7位从寻址。此外，CCI可以选择支持高达1Mbps (Fm+)， 12.5Mbps (SDR)或25Mbps (DDR)。

CCI可以在C/D-PHY上使用或不使用CSI-2。当CCI作为CSI-2总线的一部分使用时，应将CSI-2接收器配置为主站，将CSI-2发送器配置为从站。当使用CCI而不使用CSI-2 over C/D-PHY时，应使用主机作为主机。CCI能够处理总线上的多个从站

CCI (I2C)不支持多主模式。任何未在本节中描述的I2C命令都将被忽略，并且不会导致设备的意外操作。

在CCI (I3C)中，应支持任何I3C强制功能和“必需的”CCC命令，也可以支持任何I3C可选功能和命令(例如多主机，带内中断，热连接)。

不要将CCI术语“主”和“从”与C-PHY或D-PHY规范中的类似术语混淆;他们没有血缘关系。

通常，在发送器和接收器之间有一个专用的CCI接口

CCI是I2C或I3C协议的一个子集，它包括I2C或I3C规范中指定的I2C/I3C从设备的必备功能的最小组合。因此，符合CCI规范的发射机也可以连接到系统I2C或I3C总线上。

但是，必须小心，以免I2C或I3C主服务器试图使用CCI主服务器或从服务器不支持的I2C或I3C特性。

CCI发送器可能具有支持I2C或I3C的附加功能，但这取决于实现。进一步的细节可以在特定设备的数据表中找到。

本规范不试图定义CCI主机发送的控制消息的内容。

因此，实现者有责任定义一组控制消息和相应的帧定时，以及CCI主站在向CCI从站发送控制消息时必须满足的任何I2C或I3C延迟要求。

Camera Sensor采用I2C与主控进行交互，目前sensor driver按照I2C设备驱动方式实现，sensor driver

同时采用v4l2 subdev的方式实现与host driver之间的交互。

CCI (I2C)数据传输协议遵循I2C规范。START、repeat START和STOP条件以及数据传输协议都在[NXP01]中指定。

6.1.1 CCI (I2C)消息类型:

一个基本的CCI (2)C)信息包括:

- 启动或重复启动条件

- 带读/写位的从地址

- 从机应答

- 指向从设备内部寄存器的子地址(INDEX)(不用于从当前位置单读)

- 从机应答信号(不用于当前位置的单次读取)

- 对于写操作：

- 数据字节从主机

- 从机应答/不应答

- 停止或重复启动条件

- 对于读操作：

- 重复启动条件(不用于从当前位置单次读取)

- 带读位的从地址(不用于从当前位置单读)

- 确认从机发出的信号(不用于从当前位置单次读取)

- 来自从机的数据字节

- 主机应答、不应答

- 停止或重复启动条件

CCI Slave可以通过在CCI消息之间使用repeat START而不是本节所示的START和/或STOP来支持背靠背消息。

CCI (I2)中的从机地址C)是7位长

CCI (I2C)支持8位INDEX和8位数据，也支持16位INDEX和8位数据。从属设备定义了所使用的消息类型。

6.1.2、CCI (I2C)读写操作：

Media Controller，Linux kernel 的一种媒体框架

Entity, 本文指 Media Controller 框架下的各节点

MP，即 Main Path，指 Rockchip ISP 的一个输出节点，可输出全分辨率图像，一般用来拍照，

抓取 Raw 图

SP，即 Self Path，指 Rockchip ISP 的一个输出节点，最高只能输出 1080p 分辨率，一般用

作预览

Pipeline，本文指 media controller 下各个 entity 相互连接形成的链路

IOMMU，Input-Output Memory Management Unit，本文指 Rockchip 系列芯片中的iommu 模块，用于将物理上分散的内存页映射成 cif、isp 可见的连续内存

Async Sub Device，特指在 Media Controller 结构下的 V4L2 子设备

Bayer Raw，或者 Raw Bayer，表示设备（Sensor 或 ISP）输出的 RGGB，BGGR，GBRG，GRBG 等格式，或该格式的数据帧

sensor 相关dts:

i2c1

mipi 物理总线相关:

sensor----mipi dphy

csi\_dphy0

isp\_in

mipi 协议解析相关 (MIPI Host)

每个mipi phy都需要一个csi2模块来解析mipi协议

sensor----mipi dphy ---- mipi\_csi2

mipi\_csi2

图像处理芯片(vicap)相关

sensor----mipi dphy ---- mipi\_csi2-----vicap

rkcif\_mipi\_lvds

图像处理芯片(isp)相关

rkisp\_vir0

ucam\_out0 - csi\_dphy0\_input - csi\_dphy0\_output - mipi\_csi2\_input <-> mipi\_csi2\_output

gc2053 csi\_dphy0 mipi\_csi2 rkcif\_mipi\_lvds

ucam\_out0 ---> csi\_dphy0\_input -- csi\_dphy0\_output --> mipi\_csi2\_input -- mipi\_csi2\_output <---> cif\_mipi\_in

驱动汇总文件：

1 关于 sensor模块分析： sensor.c

kernel\drivers\media\i2c\techpoint\techpoint\_v4l2.c gc2053.c

2 关于 mipi 物理层 (csi dphy)模块 （csi\_dphy0 / csi\_dphy1）分析 : mipi\_csi\_dphy.c

kernel\drivers\phy\rockchip\phy-rockchip-mipi-rx.c

3 关于 mipi解析模块( mipi csi)(MIPI Host)模块分析：mipi\_csi.c

kernel\drivers\media\platform\rockchip\cif\mipi-csi2.c

4 关于 图像处理vicap 模块分析：rkcif\_mipi.c

kernel\drivers\media\platform\rockchip\cif\dev.c

isp驱动

1、sensor驱动代码，i2c

2、mipi物理层、解析层驱动代码

3、isp处理驱动代码

ucam\_out1 - csi\_dphy1\_input - csi\_dphy1\_output <-> isp\_in

gc2053 csi\_dphy1 rkisp\_vir0

ucam\_out1 --> csi\_dphy1\_input -- csi\_dphy1\_output <---> isp\_in

第一部分Sensor相关即不同板子摄像头选型肯定是不一样的，这部分的驱动就是要实现Sensor的初始化，寄存器的配置等，比如启动图像捕获的时候，需要配置寄存器，从应用层会使用一个ioctl一步

一步的调用下来到驱动中，所以这部分是Sensor相关的，需要Sensor提供相应的驱动。

第二部分MIPI相关，比如我们常用的SOC和Sensor连接就是使用MIPI接口，所以就需要实现MIPI相关部分的驱动，这部分一般来说是由SOC厂商进行提供。

第三部分图像处理芯片(如ISP)部分，如果我们的SOC平台如果有ISP模块那么就有ISP模块部分的驱动代码，这部分也是SOC厂商提供的。

从dts的配置中我们可以看到，驱动的绑定路径为，从Sensor连接到MIPI然后MIPI连接到ISP，所以说Sensor捕获到的数据通过MIPI传入ISP，然后通过ISP处理后传动应用层进行处理，整个Camera的驱动框

架大概就是这个流程