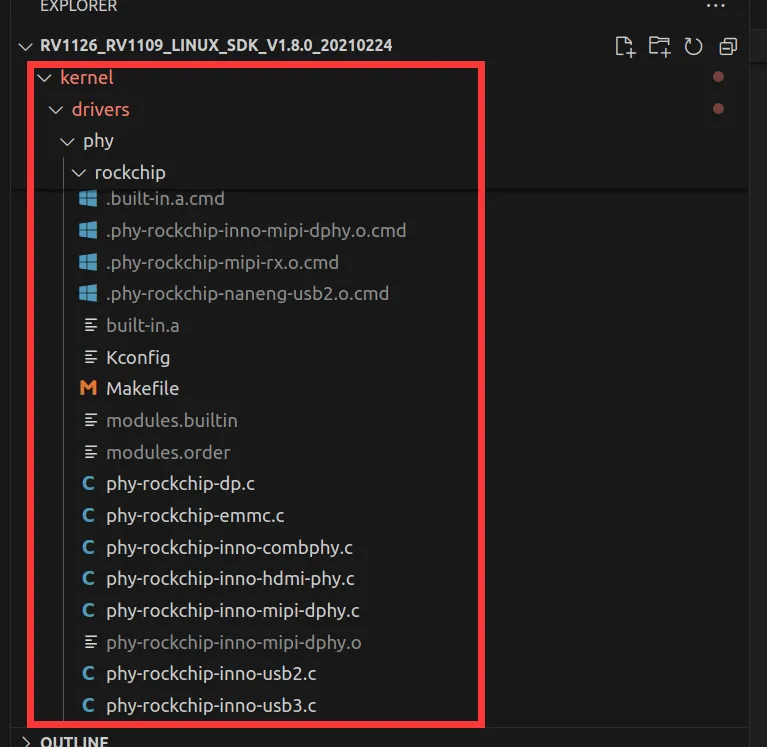
第10期内容讲解：

本期内容开始，我们开始新的内容学习，也就是开始讲解mipi\_d-phy驱动源码学习；

有问题可以加v进行交流（需要系统学习嵌入式音视频项目的付费也可加）：tu18879499804

首先mipi-dpy的驱动源码，在课程sdk里面的Linux kernel路径下，如下图所示



首先在讲解mipi的物理层驱动代码讲解之前，非常有必要详细的来看一下mipi物理层的学习，这块我是参考原文档来学习汇总的(版本不一样，内容稍微也会不同，但是大体是一样的)：

[📎MIPI\_D-PHY\_specification\_v1-2.pdf](https://www.yuque.com/attachments/yuque/0/2024/pdf/2620685/1711769740939-014602c7-4825-481e-8fff-a1d0e694b93f.pdf)

sensor的视频流整体路线：

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

\

对应的设备树：

&csi\_dphy0 {

status = "okay";

//mipi有两端一段连接Sensor，另一端连接 mipi\_csi2\_input

ports {

#address-cells = <1>;

#size-cells = <0>;

//这里为啥有两个端口呢，第一个端口是作为sensor的输入

port@0 {

reg = <0>;

#address-cells = <1>;

#size-cells = <0>;

mipi\_in\_ucam0: endpoint@1 {

reg = <1>;

remote-endpoint = <&ucam\_out0>;

// data-lanes = <1 2 3 4>;

data-lanes = <1 2>; // only for GC2053

};

};

//这个端口是输出给mipi\_csi2

port@1 {

reg = <1>;

#address-cells = <1>;

#size-cells = <0>;

csidphy0\_out: endpoint@0 {

reg = <0>;

//mipi\_csi2的port名称

remote-endpoint = <&mipi\_csi2\_input>;

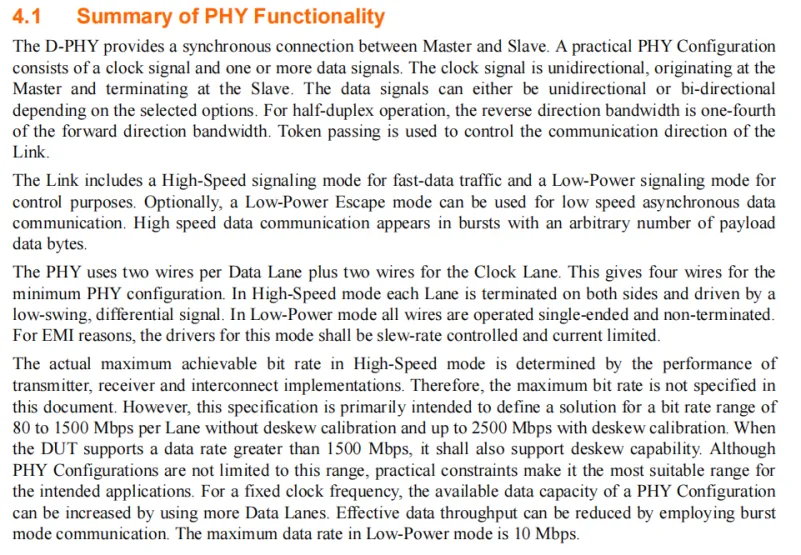
};

};

};

};

来看一下mipi\_d-phy的大纲介绍：



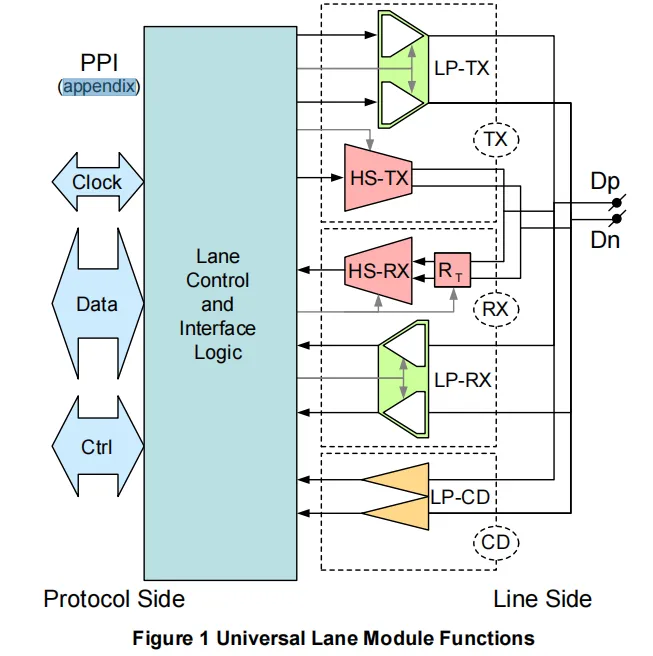
D-PHY在主设备和从设备之间提供了一个同步连接。一个实际的PHY配置包括一个时钟信号和一个或多个数据信号。时钟信号是单向的，起源于主设备并终止于从设备。数据信号可以是单向的或双向的，这取决于所选的选项。对于半双工操作，反向带宽是正向带宽的四分之一。使用令牌传递来控制链路的通信方向。(白话来说：D-PHY是一种高速串行接口，它通过同步连接实现主从设备之间的通信。在这种配置中，时钟信号负责同步，它从主设备发出并传输到从设备。数据信号则负责传输数据，其传输方向可以是单向或双向，具体取决于系统的设计需求。 在半双工模式下，D-PHY的通信是单向的，即在任何给定时刻，数据只能在一个方向上传输。这意味着反向带宽（从从设备到主设备的带宽）是正向带宽（从主设备到从设备的带宽）的四分之一。为了在主设备和从设备之间切换通信方向，D-PHY采用了令牌传递机制，这是一种确保在特定时间内只有一个设备在发送数据的协议，从而避免了数据冲突并提高了通信效率)。

链路包括一个高速信号模式，用于快速数据传输，以及一个低功耗信号模式，用于控制目的。可选地，可以使用低功耗逸出模式进行低速异步数据通信。高速数据通信以突发方式出现，具有任意数量的有效数据字节。

PHY为每个数据通道使用两根导线，另外为时钟通道使用两根导线。这构成了最小PHY配置所需的四根导线。在高速模式下，每个通道在两端都被终结，并且由一个低摆幅的差分信号驱动。在低功耗模式下，所有导线都是单端操作且未终结的。出于电磁干扰(EMI)的考虑，该模式下的驱动器应该是速率控制和电流限制的

（1）mipi\_d-phy的整体架构：

Lane Module: 数据传输通道的基本单元



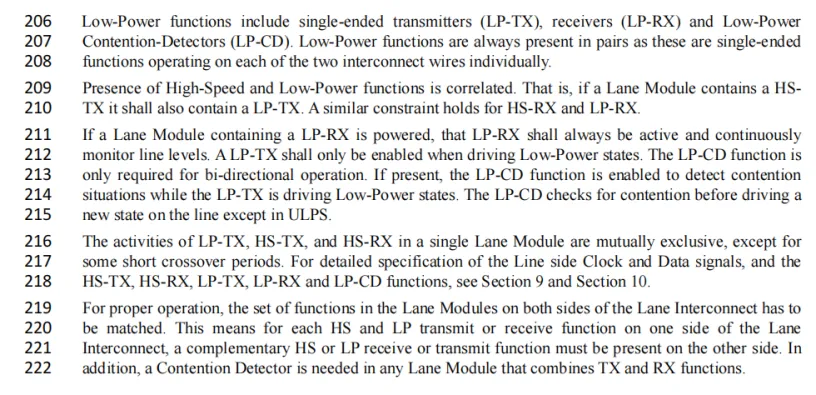
* HS-TX ：High-Speed Transmitter (Low-Swing Differential)
* HS-RS: High-Speed Transmitter(Low-Swing Differential)
* LP: Low-Power :identifier for opteration mode
* LP-CD: Low-Power Contention Dectector
* LP-RX : Low-Power Receiver (Large-Swing Single-Ended)
* LP-TX: Low-Power Transmitter (Large-Swing Single-Ended)
* 高速收发器（HS-TX/RX）：在高速模式下，负责发送和接收差分信号。
* 低功耗收发器（LP-TX/RX）：在低功耗模式下，负责发送和接收单端信号。
* 低功耗竞争检测器（LP-CD）：用于检测低功耗模式下的信号冲突。
* 控制逻辑：管理Lane Module的状态转换和操作模式。

一个通道模块可能包含一个高速发送器（HS-TX）、一个高速接收器（HS-RX）或两者都有。在单个通道模块内，高速发送器和高速接收器在正常操作期间永远不会同时启用。启用的高速功能应当在其通道互联一侧终止通道，如第9.1.1节和第9.2.1节所定义。如果通道模块中的高速功能未启用，则该功能应置于高阻态。

在MIPI接口的物理层设计中，每个通道模块（Lane Module）可以包含发送和接收功能，即可能只有高速发送器（HS-TX）、只有高速接收器（HS-RX）或者两者都具备。但是，在正常工作状态下，同一个通道模块内的发送器和接收器不会同时被激活。这样做的目的是为了避免信号冲突和资源浪费，确保数据传输的正确性和可靠性。

当通道模块的高速功能被启用时，发送器（HS-TX）会在其一侧的通道互联上产生信号，同时接收器（HS-RX）会在同一侧接收信号。根据MIPI规范的第9.1.1节和第9.2.1节的要求，发送器需要在通道互联的发送端正确地终止信号，以确保信号的完整性和稳定性。

如果通道模块的高速功能没有被启用，那么该模块应当进入高阻态。高阻态意味着该通道模块不会对信号线路产生影响，从而避免了可能的信号干扰和不必要的功耗。这种设计允许系统根据实际需要灵活地启用或关闭各个通道模块的高速功能，以适应不同的工作条件和性能要求。



争用检测器（LP-CD）。低功耗功能总是成对出现的，因为这些是单端功能，分别在两条互联线上单独操作。高速和低功耗功能的存在是相关的。也就是说，如果一个通道模块包含一个高速发送器（HS-TX），它也应当包含一个低功耗发送器（LP-TX）。类似的限制也适用于高速接收器（HS-RX）和低功耗接收器（LP-RX）。如果包含一个低功耗接收器（LP-RX）的通道模块被供电，那么该LP-RX应当始终处于活跃状态，并持续监控线路电平。低功耗发送器（LP-TX）仅在驱动低功耗状态时才被启用。争用检测器功能仅在双向操作中需要。如果存在，当LP-TX驱动低功耗状态时，争用检测器功能被启用以检测争用情况。在ULPS（超低功耗状态）之外，LP-CD在驱动线路上新状态之前检查争用。单个通道模块中LP-TX、HS-TX和HS-RX的活动是相互排斥的，除了一些短暂的交叉时期。有关线路侧时钟和数据信号以及HS-TX、HS-RX、LP-TX、LP-RX和LP-CD功能的详细规范，请参见第9节和第10节。为了正常操作，通道互联两侧的通道模块中的一组功能必须匹配。这意味着对于通道互联一侧的每个HS和LP发送或接收功能，另一侧必须存在一个补充的HS或LP接收或发送功能。此外，在任何结合了发送和接收功能的通道模块中，都需要一个争用检测器。

总线的作用？

在 Linux 内核 的设备驱动模型中，关心总线、设备和驱动这 3 个实体，总线将设备和驱动绑定。 在系统每注册一个设备的时候， 会寻找与之匹配的驱动； 相反的， 在系统每注册一个驱动的时候，会寻找与之匹配的设备，而匹配由总线完成。一个现实的Linux 设备和驱动通常都需要挂接在一种总线上，对于不属于某一实际总线的设备，比如 led， lcd,rtc 等设备， 系统为了对它们进行统一管理， 虚拟出了一条总线出来，称为 platform（平台） 总线， 相应的设备称为 platform\_device， 而驱动成为把驱动和设备资源进行分离：增加驱动的可移植性。所以说，驱动代码应该只包含与具体设备无任何关系的代码，只针对当前驱动所要操作的设备，提供出一套通用的操作方法。设备的信息应该都是由平台来描述，而不是在驱动中定义，驱动中应该从平台设备中获得具体信息