H.264/H.265 Video Codec Unit v1.2

LogiCORE IP 产品指南

Vivado Design Suite

PG252 2018 年 12 月 5 日

条款中英文版本如有歧义，概以英文文本为准。

XILINX

目录

引言

赛灵思 Zynq® UltraScale+™ MPSoC 器件的 LogiCORE™ IP H.264/H.265 Video Codec Unit (VCU) 核能够以每秒 60 帧 (4K UHD @ 60 Hz)、高达 3840×2160 像素的分辨率对视频流同步压缩和解压缩。 H.264/H.265 功能是以 Zynq UltraScale+ MPSoC EV 器件内的嵌入式硬 IP来实现的。 VCU 适用于包括但不限于视频监控和网络视频连接应用，这些应用包括视频会议、嵌入式视觉、生物医学仪器等。

特点

- 多标准编码/解码支持，包括：
  - ISO MPEG-4 第 10 部分：高级视频编码 (AVC)/ITU H.264
  - ISO MPEG-H 第 2 部分：高效视频编码 (HEVC)/ITU H.265
  - HEVC：Main、 Main Intra、 Main10、 Main10 Intra、 Main 4:2:2 10、 Main 4:2:2 10 Intra （直到 Level 5.1 (High Tier)）
  - AVC：Baseline、 Main、 High、 High10、 High 4:2:2、 High10 Intra、 High 4:2:2 Intra （直到 Level 5.2）
- 支持多达 32 个流的同步编码和解码 （最大聚合带宽为 3840x2160 @ 60fps）
- 低时延速率控制
- 灵活的速率控制：CBR、 VBR 和常量 QP
- 支持分辨率高达 4K UHD @ 60 Hz 的同步编码和解码
- 支持 8 K UHD (~15 Hz) 的降低帧速率

LogiCORE™ IP 相关信息表	
核相关信息	
支持的器件系列(1)	Zynq® UltraScale+™ MPSoC 系列 EV 器件
支持的用户接口	AXI4-Lite、 AXI4-Memory Mapped
资源	性能与资源利用网页
核提供的材料	
设计文件	加密 RTL
示例设计	Verilog
测试平台	Verilog
约束文件	赛灵思设计约束 (XDC)
仿真模型	Verilog 或 VHDL 源 HDL 模型
支持的 S/W 驱动	包含在 PetaLinux 中
经过测试的设计流程(2)	
设计入门	Vivado® Design Suite
仿真	有关支持的仿真器，请参阅《Vivado Design Suite 用户指南：版本说明、安装和许可》。
综合	Vivado 综合
支持	
由赛灵思支持网页提供	

注释：

1. 有关所支持器件的完整列表，请参阅 Vivado IP 目录。
2. 有关所支持的版本，请参阅《Vivado Design Suite 用户指南：版本说明、安装和许可》。

特点 （续）

- H.264 和 H.265 相关渐进式支持；与开发中的 H.265 相关的隔行扫描支持
- 视频输入：
  - YCbCr 4:2:2、 YCbCr 4:2:0 和 Y-only （单色）
  - 每个颜色通道 8 位和 10 位

XILINX

第 1 章

简介

引言

LogiCORE™ IP H.264/H.265 Video Codec Unit (VCU) 核支持多标准视频编码和解码，包括支持符合 H.264 标准的高效视频编码 (HEVC) 和高级视频编码 (AVC)。这个单元可提供编码 （压缩）和解码 （解压缩）功能，并且能够同时编码和解码。

VCU 是特定 Zynq® UltraScale+ MPSoC 的可编程逻辑 (PS) 里的集成块，无需直接连接到处理器系统 (PS)，并包含编码器和解码器接口。VCU 还能提供一些便于 VCU 和 PL 之间连接的附加功能。VCU 操作要求应用处理器单元 (APU) 来服务中断以协调数据传输。编码器由 APU 通过预先准备的任务列表控制，并且 APU 响应时间没有列入执行关键路径中。 VCU 没有音频支持。音频编码和解码可以通过 PS 或 PL 中的软 IP 在软件中完成。图 1-1 显示的是 VCU 核的顶层原理图。

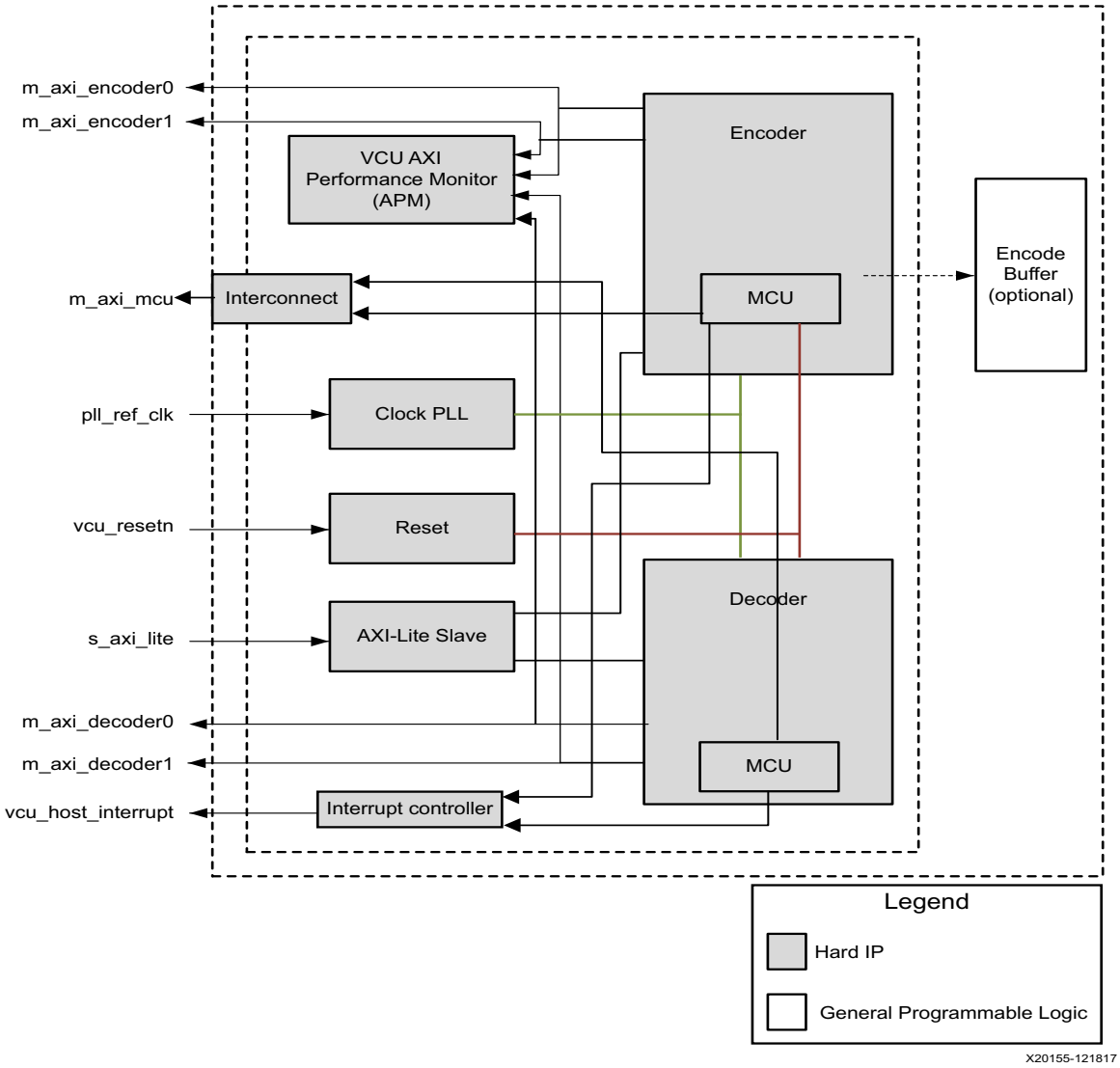

图 1-1：顶层原理图

编码器块简介

编码器引擎被设计来用 HEVC (ISO/IEC 23008-2 高效视频编码）和 AVC (ISO/IEC 14496-10 高级视频编码）标准对视频流进行处理。它全面支持这些标准，包括支持 8 位和 10 位颜色、Y-only（单色）、4:2:0 和 4:2:2 色度格式以及高达 4K UHD @ 60 Hz 的性能表现。图 3-1 显示的是编码器块的顶层接口和详细架构。编码器还包含多个全局寄存器、一个中断控制器和一个定时器。编码器由微控制器 (MCU) 子系统控制。在 APU 上运行的 VCU 应用通过赛灵思 VCU 控制软件库 API 与编码器微控制器进行交互。如需了解更多信息，请参阅第 12 章中的“VCU 控制软件”。微控制器固件 (MCU Firmware) 不是用户可修改的。

APU 通过 32 位的 AXI4-Lite 接口来控制 MCU （以配置编码参数）。两个 128 位的 AXI4 主接口用于将视频数据和元数据移入、移出系统存储器。 32 位的 AXI4 主接口用于获取 MCU 软件 （指令高速缓存接口）并加载或存储附加的 MCU 数据 （数据高速缓存接口）。

解码器块简介

解码器块能够通过 HEVC (ISO/IEC 23008-2 高效视频编码）和 AVC (ISO/IEC 14496-10 高级视频编码）标准对视频流进行处理。它全面支持这些标准，包括支持 8 位和 10 位颜色深度、Y-only（单色）、4:2:0 和 4:2:2 色度格式以及高达 4K UHD @ 60 Hz 的性能表现。它还包含多个全局寄存器、一个中断控制器和一个定时器。

VCU 解码器由微控制器 (MCU) 子系统控制。APU 使用 32 位的 AXI4-Lite 从接口来控制 MCU。两个 128 位的 AXI4 主接口用于将视频数据和元数据移入、移出系统存储器。32 位的 AXI4 主接口用于获取 MCU 软件（指令高速缓存接口）并加载或存储附加的 MCU 数据（数据高速缓存接口）。在 APU 上运行的 VCU 应用通过赛灵思 VCU 控制软件库 API 与解码器微控制器进行交互。如需了解更多信息，请参阅第 12 章中的“VCU 控制软件”。微控制器不是用户可修改的。

解码器包括多个控制寄存器、一个桥接单元和一组内部存储器。此桥接单元用于管理解码器所需的所有外部存储器访问的仲裁请求、突发地址和突发长度。

MCU 简介

编码器和解码器块均可实现 32 位 MCU，以处理与硬件块的交互。 MCU 从 APU 接收命令，将命令解析为多个 slice 或块级命令，并在编码器和解码器块上执行这些命令。执行命令后， MCU 会将状态传递给 APU，并重复此过程。

应用

VCU 核是位于 PL 中的专用电路，可为各种用例提供最大的灵活性，而内存带宽是一个关键的驱动因素。无论应用是需要同时对 4K UHD @ 60 Hz 的编码和解码进行处理，还是需要对单个 SD 流进行处理，都可以实现系统设计和存储器拓扑结构，从而达到特定用例在性能、最优化和集成方面的平衡。在图 1-2 显示的用例示例中， VCU 核与 PS 和 PL DDR 外部存储器一起工作。

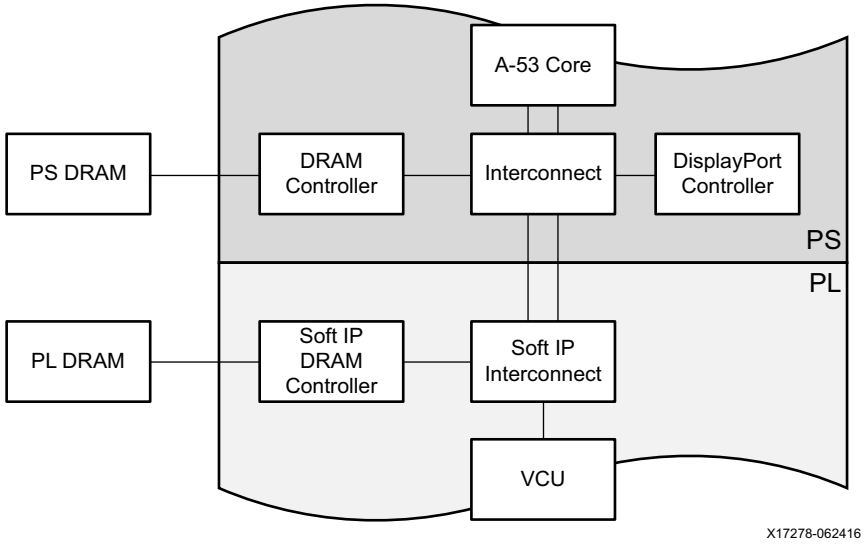

图 1-2：VCU 应用

不支持的功能

不支持以下功能：

- AVC/HEVC 中的可扩展视频编码
- H.264 (AVC)
  - 隔行扫描视频格式
  - 编码器：
    - 无损模式 （变换旁路、 I_PCM）
  - 解码器：
    - 灵活宏块排序 (FMO)
    - 任意 slice 排序 (ASO)
    - 冗余 slice (RS)
    - 单个流中的动态分辨率/动态色度格式/动态配置信息/动态级别更改
- H.265 (HEVC)
  - 编码器：
    - 自适应偏移 (SAO) 滤波器示例
    - 不对称运动分区 (AMP)
    - 无损模式 （跨越旁路、 PCM）
    - 变换跳过模式
    - 波前并行处理 (WPP)
  - 解码器：
    - 单个流中的动态分辨率/动态色度格式/动态配置信息/动态级别更改

如需了解更多信息，请参阅 《Zynq UltraScale+ MPSoC 量产勘误表》[参照 12]。

许可和订购信息

许可证类型

赛灵思 LogiCORE IP 模块仅适用于 Zynq® UltraScale+™ MPSoC 系列的 EV 器件，而且根据赛灵思最终用户许可证条款，该模块随附赛灵思 Vivado Design Suite 免费提供。有关此模块和其他 LogiCORE IP 模块的信息，请访问赛灵思知识产权页面。有关其他 LogiCORE IP 模块和工具的信息，请联系您所在当地的赛灵思销售代表。

如需了解更多信息，请访问 Zynq UltraScale+ MPSoC 产品页面。

可能需要提供第三方许可证并支付版税才能实现 H.264 或 H.265 视频压缩标准；更多信息可以从各个专利持有人和行业联盟 （例如 MPEG LA 和 HEVC Advance）获得。

XILINX

第 2 章

产品规格

标准

编码器和解码器块符合以下标准：

- ISO/IEC 14496-10:2014(en)
  信息技术 - 视听对象编码 - 第 10 部分：高级视频编码
- ITU 建议书 - T H.264 | 国际标准 ISO/IEC 14496-10：用于通用视听服务的高级视频编码
- ITU 建议书 - T H.265 | 国际标准 ISO/IEC 23008-2：高效视频编码
- ISO/IEC 23008-2:2017
  信息技术 - 异构环境中的高效编码和媒体传送 - 第 2 部分：高效视频编码

性能

以下部分将详细介绍 H.264/H.265 Video Codec Unit 的性能特征。

最大频率

以下是 《Zynq UltraScale+ MPSoC 数据手册：DC 和 AC 开关特性》中描述的目标器件的典型时钟频率。系统可达到的最大时钟频率可能会变化。最大可实现的时钟频率和所有资源数可能会受其他工具选项、其他器件逻辑、不同版本的赛灵思工具使用以及其他因素的影响。

吞吐量

VCU 支持分辨率高达 4K UHD @ 60 Hz 的同步编码和解码技术。该吞吐量可以是 4K UHD 的单个流，也可以分成多达 32 个较小的流 （最高 480p @ 30 Hz）。如果累积吞吐量不超过 4K UHD @ 60 Hz，则可以支持几个不同分辨率组合的 1 - 32 个流的几种组合。

资源利用

分辨率为 4K UHD @ 60 Hz 的流会消耗大量的外部存储器接口带宽,也会消耗处理器系统与可编程逻辑之间大量的 ArmAMBA®AXI4 总线带宽。如需了解更多信息，请参阅第 3 章中的“DDR 存储器占用要求”。

对于同步编码器和解码器操作 （包括转码用例），请考虑使用赛灵思 PS 存储器控制器和专用的赛灵思 VCU 存储器控制器。如需了解更多信息，请参阅第 6 章"Zynq UltraScale+ EV 架构 Video Codec Unit DDR4 LogiCORE IP"。

如需了解有关资源利用的详情，请访问性能与资源利用。

核接口与寄存器空间

该核提供下列接口：

- 四个 128 位的 AXI 主接口，用于与外部存储器进行通信
- 一个 32 位的 AXI 主接口，用于控制通信
- AXI4-Lite 接口，用于与应用处理器单元 (APU) 进行通信

四个 128 位的 AXI 主接口用来通过 PS 存储器接口和 PL 存储器接口将视频数据移入和移出外部存储器。两个 AXI 接口被分配来进行编码，两个被分配来进行解码。

端口描述

VCU 核的顶层信号接口如图 2-1 所示。

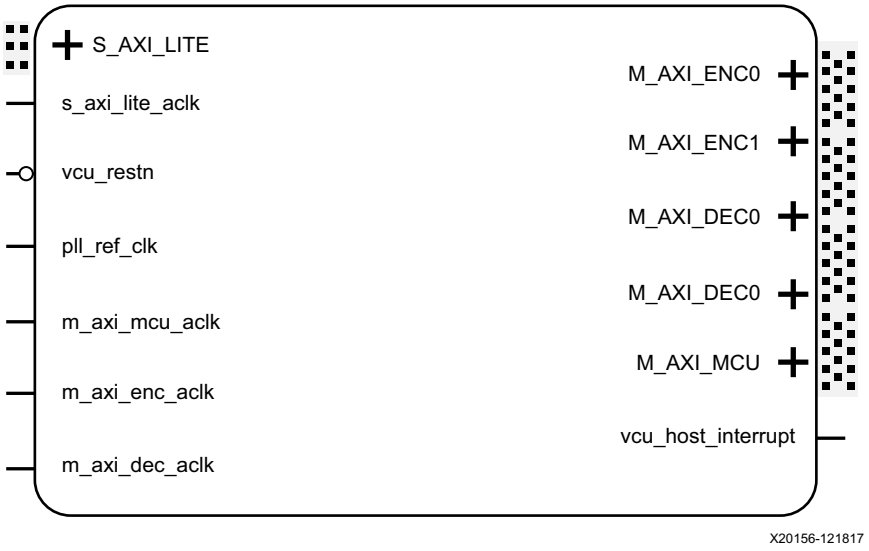

图 2-1：VCU 核的顶层信号接口

表 2-1 对核接口进行了总结。

表 2-1：VCU 接口

接口名称	接口类型	描述
M_AXI_ENC0	AXI-4 存储器映射主接口	用于编码器块的 128 位存储器映射接口。如需了解更多详情，请参阅第 1 章中的“编码器块简介”。
M_AXI_ENC1	AXI-4 存储器映射主接口	用于编码器块的 128 位存储器映射接口。如需了解更多详情，请参阅第 1 章中的“编码器块简介”。
M_AXI_DEC0	AXI-4 存储器映射主接口	用于解码器块的 128 位存储器映射接口。如需了解更多详情，请参阅第 1 章中的“解码器块简介”。
M_AXI_DEC1	AXI-4 存储器映射主接口	用于解码器块的 128 位存储器映射接口。如需了解更多详情，请参阅第 1 章中的“解码器块简介”。

表 2-1：VCU 接口 （续）

接口名称	接口类型	描述
M_AXI_MCU	AXI-4 存储器映射主接口	用于 MCU.的 32 位存储器映射接口。如需了解更多详情，请参阅第 1 章中的“MCU 简介”。
S_AXI__LITE	AXI4-Lite 储器映射从接口	用于访问主控制器的 AXI4-Lite 存储器映射接口。如需了解更多详情，请参阅第 1 章中的“MCU 简介”。

共享接口信号

表 2-2 主要介绍专用 AXI4 接口共享的信号，或非 AXI4 接口信号。

表 2-2：VCU 端口

端口名称	方向	描述
m_axi_enc_aclk	输入	M_AXI_VCU_ENCODER0 和 M_AXI_VCU_ENCODER1 的 AXI 输入时钟
s_axi_lite_aclk	输入	S_AXI_PL_VCU_LITE 的 AXI 输入时钟
pll_ref_clk	输入	PLL 参考时钟输入
vcu_resetn	输入	来自 PL 的低电平有效复位输入
vcu_host_interrupt	输出	来自 VCU 的高电平有效中断输出。可以映射到 PL-PS 中断引脚。
m_axi_dec_aclk	输入	M_AXI_VCU_DECODER0 和 M_AXI_VCU_DECODER1 的 AXI 输入时钟
m_axi_mcu_aclk	输入	M_AXI_MCU 接口的输入时钟

寄存器空间

Zynq UltraScale+ VCU IP （软 IP）在可编程逻辑中实现寄存器。表 2-3 主要介绍软 IP 寄存器。这些寄存器可通过 AXI4-Lite 总线从 PS 访问。

表 2-3：软 IP 寄存器

寄存器	地址偏移	宽度	类型	定义
视频设置				
VCU_ENCODER_ENABLE	0x41000	32	Ro	1 = 启用 0 = 禁用
VCU_DECODER_ENABLE	0x41004	32	Ro	1 = 启用 0 = 禁用
VCU_MEMORY_DEPTH	0x41008	32	Ro	编码器中的条目数 缓存
VCU_ENC_COLOR_DEPTH	0x4100C	32	Ro	0 = 每像素 8 位 1 = 每像素 8 位和 10 位
VCU_ENC_VERTICAL_RANGE	0x41010	32	Ro	0 = 低 1 = 中 2 = 高
VCU_ENC_FRAME_SIZE_X	0x41014	32	Ro	编码器的水平像素大小
VCU_ENC_FRAME_SIZE_Y	0x41018	32	Ro	编码器的垂直像素大小

表 2-3：软 IP 寄存器 （续）

寄存器	地址偏移	宽度	类型	定义
VCU_ENC_COLOR_FORMAT	0x4101C	32	Ro	0 = 4:2:0 1 = 4:2:2 2 = 4:0:0
VCU_ENC_FPS	0x41020	32	Ro	表示每秒帧数
VCU_ENC_VIDEO_STANDARD	0x41038	32	Ro	0 = H.265 (HEVC) 1 = H.264 (AVC)
VCU_STATUS	0x4103C	32	Ro	0 = INTERRUPT 1 = POWER_STATUS_VCUINT 2 = POWER_STATUS_VCUAUX 3 = PLL_LOCK_STATUS
VCU_DEC_VIDEO_STANDARD	0x4104C	32	Ro	0 = H.265 (HEVC) 1 = H.264 (AVC)
VCU_DEC_FRAME_SIZE_X	0x41050	32	Ro	解码器的水平像素大小
VCU_DEC_FRAME_SIZE_Y	0x41054	32	Ro	解码器的垂直像素大小
VCU_DEC_FPS	0x41058	32	Ro	解码器的每秒帧数
VCU_BUFFER_B_FRAME	0x4105C	32	Ro	B 帧的使用 0 =禁用 B 帧 1 =启用 B 帧
ENC_NUM_CORE	0x4106C	32	Ro	用于所提供配置的编码器核的个数
VCU_PLL_CLK_HI	0x41034	32	Ro	报告 Vivado 块设计中设置的 PLL 时钟频率的整数值。每个单位为 10 kHz。默认值：33.33 MHz
VCU_PLL_CLK_LO	0x41064	32	Ro	报告 Vivado 块设计中设置的 PLL 时钟频率的小数值。每个单位为 10 kHz。默认值：33.33 MHz
VCU_GASKET_INIT	0x41074	32	Rw	在访问寄存器之前，确保 VCU AXI 总线/AXI4-Lite 总线中没有待处理的 AXI 数据传输。 在访问寄存器之前置位和取消置位 vcu_resetn 信号。 位 0 = 设置为 1 表示 在 VCUINT_VC 和 VCCAUX 的电压 完全达到预定的幅度、且 PL 被编程后移除垫片隔离 位 1 = 设置为 0 表示将复位置位为 VCU。软件需要将其取消置位为 1，不进行复位。

注释：对于多流用例，寄存器 表 2-3 显示的是在 GUI 中输入的混合值。

VCU 编码器和解码器块分别如图 2-2 和图 2-3 所示。

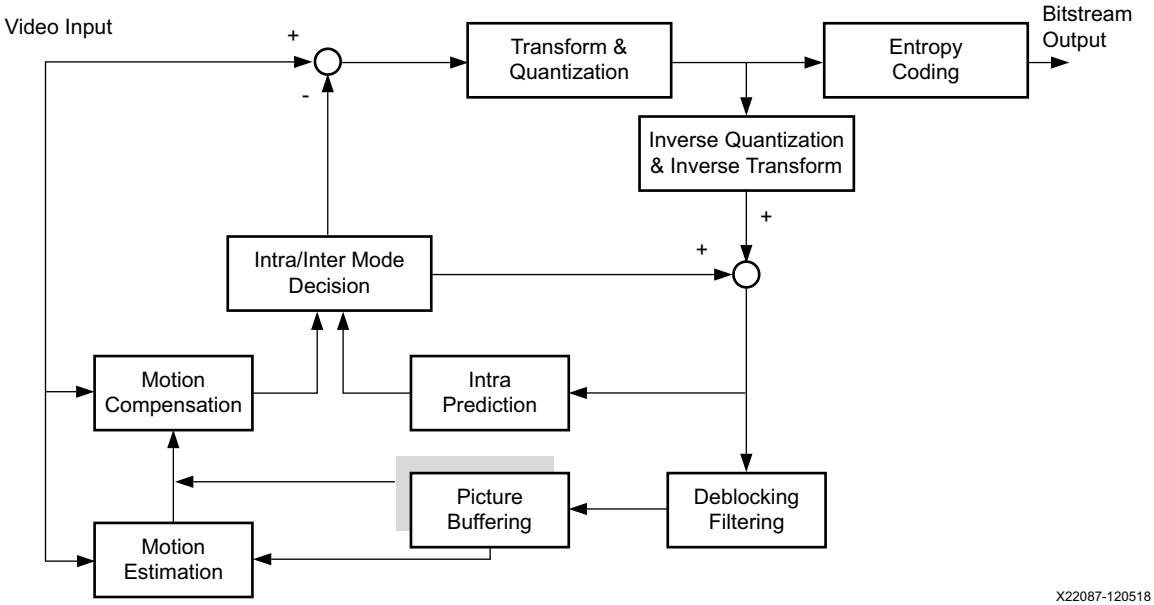

图 2-2：VCU 编码器块

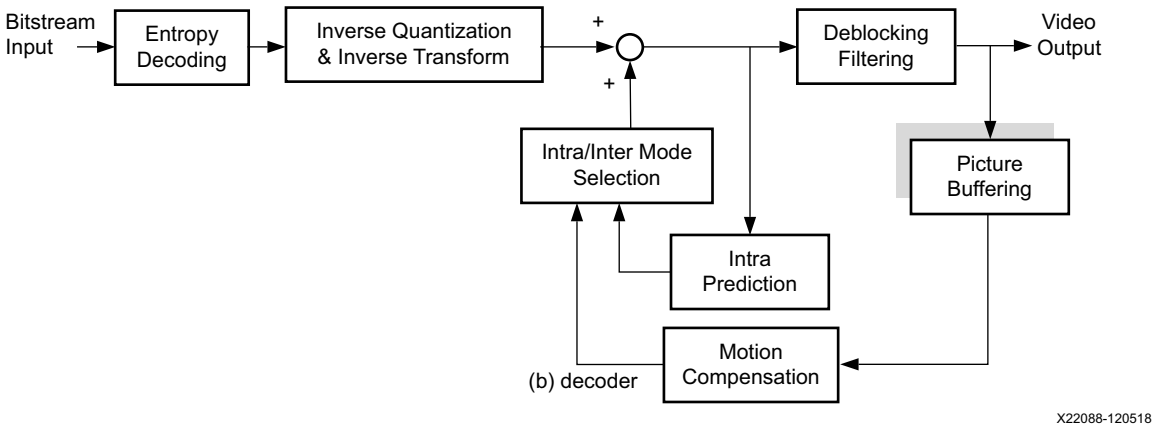

图 2-3：VCU 解码器块

XILINX

第 3 章

编码器块

引言

Video Codec Unit (VCU) 核编码器块是采用 H.265 （ISO/IEC 23008-2 高效视频编码）和 H.264 （ISO/IEC 14496-10 高级视频编码）标准对视频流进行处理的视频编码器引擎。编码器块全面支持上述标准（这些标准包括支持 8 位和 10 颜色深度、 4:2:0、 4:2:2 和 4:0:0 色度格式、以及高达 4K UHD @ 60 Hz 分辨率的性能）。

特点

表 3-1 主要描述的是 VCU 编码器块的特点。

表 3-1：编码器块的特点

视频编码的特点	H.264	H.265
性能		
配置信息	Baseline Main High High 10 High 4:2:2 High10 Intra High 4:2:2 Intra	Main Main Intra Main 10 Main 10 Intra Main 4:2:2 10 Main 4:2:2 10 Intra
级别	高达 5.2(1)	高达 5.1 High Tier(1)
速率为 @ 667 MHz(2) 时的性能 32 个流 @ 720×480p @ 30 Hz 8 个流 @ 1920×1080p @ 30 Hz 4 个流 @ 1920×1080p @ 60 Hz 2 个流 @ 3840×2160p @ 30 Hz 1 个流 @ 3840×2160p @ 60 Hz 1 个流 @ 7680×4320p @ 15 Hz	支持	支持
可配置分辨率 图像的宽度和图像高度为 8 的整数倍 最小尺寸：128×64 最大宽度或高度：16384 最大尺寸：3350 万像素	支持	支持
可配置帧速率	支持	支持
可配置比特率	支持	支持

表 3-1：编码器块的特点 （续）

视频编码的特点	H.264	H.265
编码工具		
采样位深度：8 bpc、 10 bpc	支持	支持
色度格式：YCbCr 4:2:0、YCbCr 4:2:2、Y-only（单色）	支持	支持
slice 类型：I、 P、 B	支持	支持
仅限 Progressive format	支持	支持
编码块大小	16×16 的宏块	LCU 大小：32×32 CU 大小：小至 8×8
预测大小	低至 4×4 （适用于帧内预测）、低至 8×8 （适用于帧间预测）	低至 4×4（适用于帧内预测）、低至 8×8 （适用于帧间预测）
变换大小	4×4、 8×8	4×4、 8×8、 16×16、 32×32
帧内预测模式	所有帧内 4×4、帧内 8×8、帧内 16×16 模式	33 个方向的所有模式、平面、DC
受限制的 Intra Pred 支持	支持	支持
运动估计：用于 P slice 的 1 个参考图像或用于 B slice 的 2 个参考图像	支持	支持
运动估计和补偿：1/4 样本插值	支持	支持
运动矢量预测模式	除 spatial direct 模式和 direct_8×8_inference_flag=0 以外的所有运动矢量预测模式	所有运动矢量预测/合并/跳过模式
加权预测	支持	支持
QP 控制	每帧常数、每 MB 可配置或每 MB 自适应	每帧常数、每 LCU 可配置或每 CU 自适应
色度 QP 偏移	支持	支持
缩放列表	支持	支持
In-loop 去块效应滤波器	支持	支持
熵编码	CABAC、 CAVLC	CABAC
可配置的 CABAC 初始化表	支持	支持
slice 支持	支持 （性能高于 1080p60 所需的 slice）	支持
非独立 slice 支持	不适用	支持
块支持	不适用	支持 （性能高于 1080p60 所需的块）

注释：

1. 支持 8K15 使用 6 级的子集。
2. AVC 最小图像分辨率：80×96；HEVC 最小图像分辨率：128×128。

表 3-2 对不同配置信息/级别组合可实现的最大比特率进行了总结。

表 3-2：最大比特率

标准	级别	配置信息	最大比特率 (Mbits/s)
H.264 (AVC)	4.2 (1080p60)	Baseline、 Main	50
		High	62.5
		High 10	150
		High 4:2:2	200
	5.2 (2160p60)	Baseline、 Main	240
		High	300 （内容自适应可变长度编码 (CAVLC) 或内容自适应二进制算术编码 (CABAC)，仅限内部） 267 （CABAC non-Intra-only）
		High 10	720 (CAVLC or CABAC Intra-only) 267 (CABAC non-Intra-only)
		High 4:2:2	960 (CAVLC or CABAC Intra-only) 267 (CABAC non-Intra-only)
H.265 (HEVC)	4.1 (1080p60) High Tier	Main、 Main 10	50
		Main 4:2:2 10	84
		Main 4:2:2 10 Intra	167
	5.1 (2160p60) High Tier	Main、 Main 10	160
		Main 4:2:2 10	267
		Main 4:2:2 10 Intra	533

功能说明

图 3-1 显示的是编码器块顶层接口和详细架构。

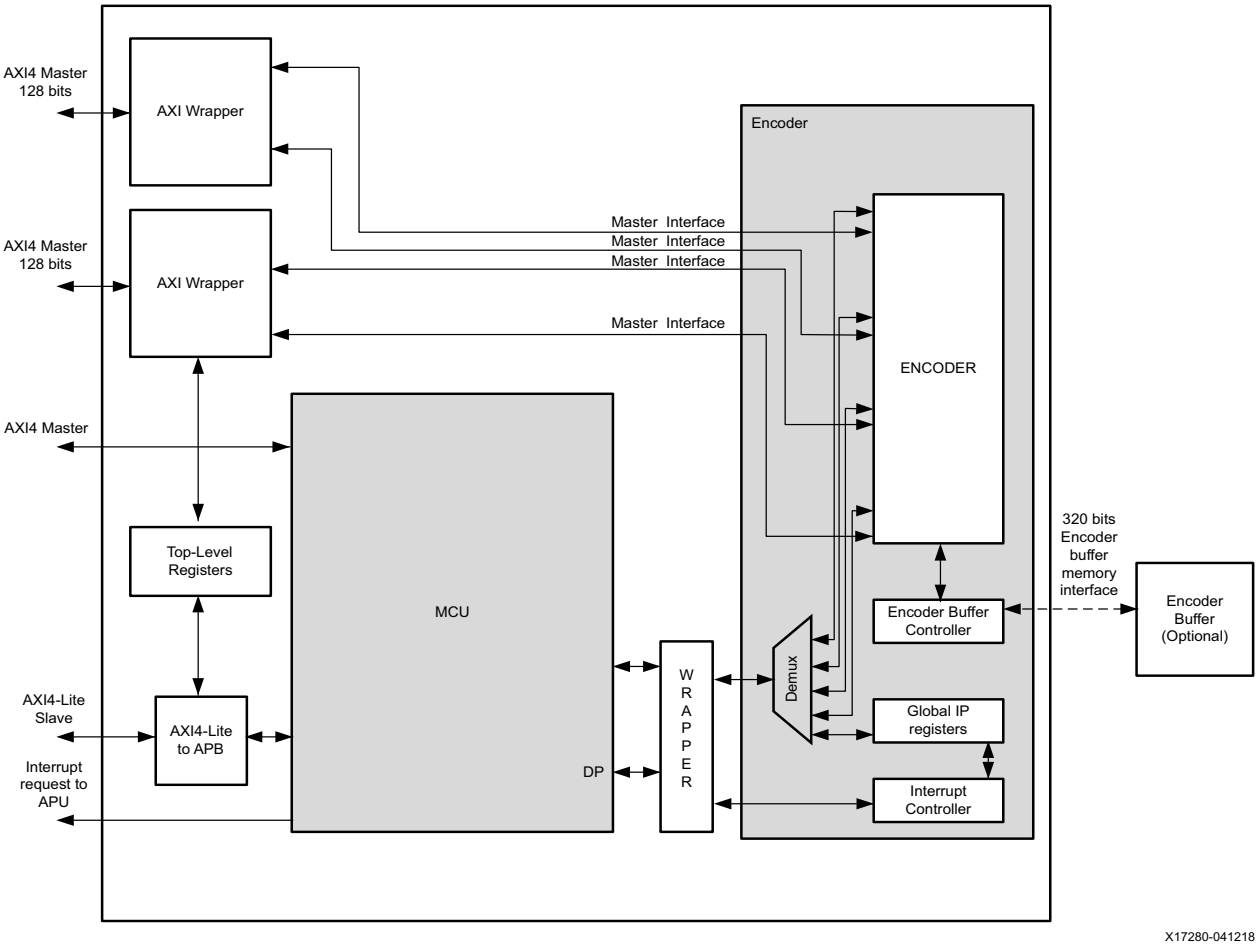

图 3-1：编码器块的详细架构

注释：MCU 的 AXI-4 主接口与解码器的相应 AXI-4 主接口进行多路复用。多路复用器输出可在嵌入式 VCU 中使用。

- 编码器块包括压缩引擎、控制寄存器、中断控制器和带存储器控制器的可选编码器缓存。编码器缓存连接到可编程逻辑中的 UltraRAM 或 RAM 块，并通过寄存器启用。
- 编码器块由微控制器单元 (MCU) 子系统控制，包括带有 32 KB 指令高速缓存的 32 位 MCU、1 KB 的数据高速缓存和 32 KB 的本地 SRAM。
- APU 通过 32 位 AXI4-Lite 从接口来控制 MCU，以配置编码器参数、启动或停止处理或获取状态从而获得结果。
- 两个 128 位的 AXI-4 主接口用于获取视频输入数据、加载和存储中间数据，并将压缩数据存储回存储器中。
- 32 位 AXI-4 主接口用于获取 MCU 软件并加载或存储其他 MCU 数据。

VCU 控制软件可以更改编码参数，甚至可以动态地在 H.264 和 H.265 编码之间进行切换，但是，必须选择可用的存储器和带宽才能针对最坏情况提供应用所需的支持。通过 VCU GUI 来探索带宽要求。

接口与端口

使用编码器块应用必须连接所有的编码器端口 （名称以 m_axi_dec 开头的端口）。

表 3-3 显示的是顶层编码器块的端口或接口列表。

表 3-3：编码器端口

名称	大小（比特）	方向	描述
时钟与复位			
pll_ref_clk	1	输入	来自 PL 的 VCU PLL 参考时钟
m_axi_enc_aclk	1	输入	存储器接口的编码器时钟
m_axi_dec_aclk	1	输入	存储器接口的解码器时钟
s_axi_lite_aclk	1	输入	AXI4-Lite 时钟
m_axi_mcu_aclk	1	输入	来自 PL 的 VCU MCU AXI 接口时钟
vcu_resetn	1	输入	低有效。来自 PL 的 VCU 复位
VCU-Interrupt (s_axi_lite_aclk)			
vcu_host_interrupt	1	输出	高有效。中断从 VCU 到 PS
VCU 编码器块、 128 位 AXI 主接口 0 (m_axi_enc_aclk domain)			
vcu_pl_enc_araddr0	44	输出	接口 0 的 AXI 主控制器读取地址总线
vcu_pl_enc_arburst0	2	输出	AXI 主控制器读取突发类型信号
vcu_pl_enc_arid0	4	输出	接口 0 的 AXI 主控制器读取突发 ID
vcu_pl_enc_arlen0	8	输出	接口 0 AXI 主控制器读取突发长度
pl_vcu_enc_arready0	1	输入	接口 0 的 AXI 主控制器读取地址准备好了
vcu_pl_enc_arsize0	3	输出	接口 0 的 AXI 主控制器读取接口大小
vcu_pl_enc_arvalid0	1	输出	接口 0 的 AXI 主控制器读取地址有效
vcu_pl_enc_awaddr0	44	输出	接口 0 的 AXI 主控制器写入地址
vcu_pl_enc_awburst0	2	输出	接口 0 的 AXI 主控制器写入突发类型
vcu_pl_enc_awid0	4	输出	接口 0 的 AXI 主控制器写入突发 ID
vcu_pl_enc_awlen0	8	输出	接口 0 的 AXI 主控制器写入突发长度
pl_vcu_enc_awready0	1	输入	接口 0 的 AXI 主控制器写入地址准备好了
vcu_pl_enc_awsize0	3	输出	接口 0 的 AXI 主控制器写入突发大小
vcu_pl_enc_awvalid0	1	输出	接口 0 的 AXI 主控制器写入地址有效
pl_vcu_enc_bresp0	2	输入	接口 0 的 AXI 主控制器写入响应
vcu_pl_enc_bready0	1	输出	接口 0 的 AXI 主控制器写入响应准备好了
pl_vcu_enc_bvalid0	1	输入	接口 0 的 AXI 主控制器写入响应有效
pl_vcu_enc_bid0	4	输入	接口 0 的 AXI 主控制器写入响应 ID
pl_vcu_enc_rdata0	128	输入	接口 0 的 AXI 主控制器读取数据
pl_vcu_enc_rid0	4	输入	接口 0 的 AXI 主控制器读取 ID 信号
pl_vcu_enc_rlast0	1	输入	接口 0 的 AXI 主控制器读取持续信号

表 3-3：编码器端口 （续）

名称	大小（比特）	方向	描述
vcu_pl_enc_rready0	1	输出	接口 0 的 AXI 主控制器读取准备好信号
Pl_vcu_enc_rresp0	2	输入	接口 0 的 AXI 主控制器读取响应信号
pl_vcu_enc_rvalid0	1	输入	接口 0 的 AXI 主控制器读取有效信号
vcu_pl_enc_wdata0	128	输出	接口 0 的 AXI 主控制器写入数据
vcu_pl_enc_wlast0	1	输出	接口 0 的 AXI 主控制器写入持续信号
pl_vcu_enc_wready0	1	输入	接口 0 的 AXI 主控制器写入准备好信号
vcu_pl_enc_wvalid0	1	输出	接口 0 的 AXI 主控制器写入有效信号
vcu_pl_enc_awprot0	1	输出	接口 0 的 AXI 主控制器写入保护信号，由 SLCR 控制
vcu_pl_enc_arprot0	1	输出	接口 0 的 AXI 主控制器读取保护信号，由 SLCR 控制
vcu_pl_enc_awqos0	4	输出	接口 0 的 AXI 主控制器写入 QOS 信号，由 SLCR 控制
vcu_pl_enc_arqos0	4	输出	接口 0 的 AXI 主控制器读取 QOS 信号，由 SLCR 控制
vcu_pl_enc_awcache0	4	输出	接口 0 的 AXI 主控制器写入高速缓存信号，由 SLCR 控制
vcu_pl_enc_arcache0	4	输出	接口 0 的 AXI 主控制器读取高速缓存信号，由 SLCR 控制
VCU 编码器块、 128 位 AXI 主接口 1 (m_axi_enc_aclk domain)			
vcu_pl_enc_araddr1	44	输出	接口 1 的 AXI 主控制器读取地址总线
vcu_pl_enc_arburst1	2	输出	AXI 主控制器读取突发类型信号
vcu_pl_enc_arid1	4	输出	接口 1 的 AXI 主控制器读取突发 ID
vcu_pl_enc_arlen1	8	输出	接口 1 AXI 主控制器读取突发长度
pl_vcu_enc_arready1	1	输入	接口 1 的 AXI 主控制器读取地址准备好了
vcu_pl_enc_arsize1	3	输出	接口 1 的 AXI 主控制器读取接口大小
vcu_pl_enc_arvalid1	1	输出	接口 1 的 AXI 主控制器读取地址有效
vcu_pl_enc_awaddr1	44	输出	接口 1 的 AXI 主控制器写入地址
vcu_pl_enc_awburst1	2	输出	接口 1 的 AXI 主控制器写入突发类型
vcu_pl_enc_awid1	4	输出	接口 1 的 AXI 主控制器写入突发 ID
vcu_pl_enc_awlen1	8	输出	接口 1 的 AXI 主控制器写入突发长度
pl_vcu_enc_awready1	1	输入	接口 1 的 AXI 主控制器写入地址准备好了
vcu_pl_enc_awsize1	3	输出	接口 1 的 AXI 主控制器写入突发大小
vcu_pl_enc_awvalid1	1	输出	接口 1 的 AXI 主控制器写入地址有效
Pl_vcu_enc_bresp1	2	输入	接口 1 的 AXI 主控制器写入响应
vcu_pl_enc_bready1	1	输出	接口 1 的 AXI 主控制器写入响应准备好了
pl_vcu_enc_bvalid1	1	输入	接口 1 的 AXI 主控制器写入响应有效
pl_vcu_enc_bid1	4	输入	接口 1 的 AXI 主控制器写入响应 ID
pl_vcu_enc_rdata1	128	输入	接口 1 的 AXI 主控制器读取数据
pl_vcu_enc_rid1	4	输入	接口 1 的 AXI 主控制器读取 ID 信号

表 3-3：编码器端口 （续）

名称	大小（比特）	方向	描述
pl_vcu_enc_rlast1	1	输入	接口 1 的 AXI 主控制器读取持续信号
vcu_pl_enc_rready1	1	输出	接口 1 的 AXI 主控制器读取准备好信号
Pl_vcu_enc_rresp1	2	输入	接口 1 的 AXI 主控制器读取响应信号
pl_vcu_enc_rvalid1	1	输入	接口 1 的 AXI 主控制器读取有效信号
vcu_pl_enc_wdata1	128	输出	接口 1 的 AXI 主控制器写入数据
vcu_pl_enc_wlast1	1	输出	接口 1 的 AXI 主控制器写入持续信号
pl_vcu_enc_wready1	1	输入	接口 1 的 AXI 主控制器写入准备好信号
vcu_pl_enc_wvalid1	1	输出	接口 1 的 AXI 主控制器写入有效信号
vcu_pl_enc_awprot1	1	输出	接口 1 的 AXI 主控制器写入保护信号，由 SLCR 控制
vcu_pl_enc_arprot1	1	输出	接口 1 的 AXI 主控制器读取保护信号，由 SLCR 控制
vcu_pl_enc_awqos1	4	输出	接口 1 的 AXI 主控制器写入 QOS 信号，由 SLCR 控制
vcu_pl_enc_arqos1	4	输出	接口 1 的 AXI 主控制器读取 QOS 信号，由 SLCR 控制
vcu_pl_enc_awcache1	4	输出	接口 1 的 AXI 主控制器写入高速缓存信号，由 SLCR 控制
vcu_pl_enc_arcache1	4	输出	接口 1 的 AXI 主控制器读取高速缓存信号，由 SLCR 控制
VCU 编码器 - 32 位 AXI 主控制器 MCU 指令和数据高速缓存接口			
vcu_pl_mcu_m_axi_ic_dc_araddr	44	输出	MCU 的 AXI 主控制器读取地址总线
vcu_pl_mcu_m_axi_ic_dc_arburst	2	输出	AXI 主控制器读取突发类型信号
vcu_pl_mcu_m_axi_ic_dc_arcache	4	输出	MCU 的 AXI 主控制器读取高速缓存
vcu_pl_mcu_m_axi_ic_dc_arid	3	输出	MCU 的 AXI 主控制器读取突发 ID
vcu_pl_mcu_m_axi_ic_dc_arlen	8	输出	MCU 的 AXI 主控制器读取突发长度
vcu_pl_mcu_m_axi_ic_dc_arlock	1	输出	MCU 的 AXI 主控制器读取锁定
vcu_pl_mcu_m_axi_ic_dc_arprot	3	输出	MCU 的 AXI 主控制器读取保护信号
vcu_pl_mcu_m_axi_ic_dc_arqos	4	输出	MCU 的 AXI 主控制器读取 QoS
pl_vcu_mcu_m_axi_ic_dc_arready	1	输入	MCU 的 AXI 主控制器读取地址准备好了
vcu_pl_mcu_m_axi_ic_dc_arsize	3	输出	MCU 的 AXI 主控制器读取地址大小
vcu_pl_mcu_m_axi_ic_dc_arvalid	1	输出	MCU 的 AXI 主控制器读取地址有效
vcu_pl_mcu_m_axi_ic_dc_awaddr	44	输出	MCU 的 AXI 主控制器写入地址
vcu_pl_mcu_m_axi_ic_dc_awburst	2	输出	MCU 的 AXI 主控制器写入突发类型
vcu_pl_mcu_m_axi_ic_dc_awcache	4	输出	MCU 的 AXI 主控制器写入高速缓存
vcu_pl_mcu_m_axi_ic_dc_awid	3	输出	MCU 的 AXI 主控制器写入地址 ID
vcu_pl_mcu_m_axi_ic_dc_awlen	8	输出	MCU 的 AXI 主控制器写入突发长度

表 3-3：编码器端口 （续）

名称	大小（比特）	方向	描述
vcu_pl_mcu_m_axi_ic_dc_awlock	1	输出	MCU 的 AXI 主控制器写入锁定
vcu_pl_mcu_m_axi_ic_dc_awprot	3	输出	MCU 的 AXI 主控制器写入保护
vcu_pl_mcu_m_axi_ic_dc_awqos	4	输出	MCU 的 AXI 主控制器写入 QoS
pl_vcu_mcu_m_axi_ic_dc_awready	1	输入	MCU 的 AXI 主控制器写入地址准备好信号
vcu_pl_mcu_m_axi_ic_dc_awsize	3	输出	MCU 的 AXI 主控制器写入突发大小信号
vcu_pl_mcu_m_axi_ic_dc_awvalid	1	输出	MCU 的 AXI 主控制器写入地址有效信号
pl_vcu_mcu_m_axi_ic_dc_bid	3	输入	MCU 的 AXI 主控制器写入响应 ID
vcu_pl_mcu_m_axi_ic_dc_bready	1	输出	MCU 的 AXI 主控制器写入响应准备好信号
pl_vcu_mcu_m_axi_ic_dc_bresp	2	输入	MCU 的 AXI 主控制器写入响应
pl_vcu_mcu_m_axi_ic_dc_bvalid	1	输入	MCU 的 AXI 主控制器写入响应有效信号
pl_vcu_mcu_m_axi_ic_dc_rdata	32	输入	MCU 的 AXI 主控制器读取数据信号
pl_vcu_mcu_m_axi_ic_dc_rid	3	输入	MCU 的 AXI 主控制器读取 ID 信号
pl_vcu_mcu_m_axi_ic_dc_rlast	1	输入	MCU 的 AXI 主控制器读取持续信号
vcu_pl_mcu_m_axi_ic_dc_rready	1	输出	MCU 的 AXI 主控制器读取准备好了信号
pl_vcu_mcu_m_axi_ic_dc_rresp	2	输入	MCU 的 AXI 主控制器读取响应信号
pl_vcu_mcu_m_axi_ic_dc_rvalid	1	输入	MCU 的 AXI 主控制器读取有效信号
vcu_pl_mcu_m_axi_ic_dc_wdata	32	输出	MCU 的 AXI 主控制器写入数据信号
vcu_pl_mcu_m_axi_ic_dc_wlast	1	输出	MCU 的 AXI 主控制器写入持续信号
pl_vcu_mcu_m_axi_ic_dc_wready	1	输入	接口 1 的 AXI 主控制器写入准备好信号
vcu_pl_mcu_m_axi_ic_dc_wstrb	4	输出	AXI 主控制器写入选通信号
vcu_pl_mcu_m_axi_ic_dc_wvalid	1	输出	MCU 的 AXI 主控制器写入有效信号
AXI4-Lite 从设备器接口 (s_axi_lite_aclk domain)			
pl_vcu_awaddr_axi_lite	20	输入	AXI4-Lite 写入地址总线
pl_vcu_awprot_axi_lite	3	输入	AXI4-Lite 写入保护信号
pl_vcu_awvalid_axi_lite	1	输入	AXI4-Lite 写入地址有效信号
vcu_pl_awready_axi_lite	1	输出	AXI4-Lite 写入地址准备好信号
pl_vcu_wdata_axi_lite	32	输入	AXI4-Lite 写入数据通道
pl_vcu_wstrb_axi_lite	4	输入	AXI4-Lite 写入选通信号
pl_vcu_wvalid_axi_lite	1	输入	AXI4-Lite 写入数据有效信号
vcu_pl_wready_axi_lite	1	输出	AXI4-Lite 写入准备好信号
vcu_pl_bresp_axi_lite	2	输出	AXI4-Lite 写入响应通道
vcu_pl_bvalid_axi_lite	1	输出	AXI4-Lite 写入响应有效信号
pl_vcu_bready_axi_lite	1	输入	AXI4-Lite 写入响应准备好信号
pl_vcu_araddr_axi_lite	20	输入	AXI4-Lite 读取地址通道
pl_vcu_arprot_axi_lite	3	输入	AXI4-Lite 读取通道保护信号

表 3-3：编码器端口 （续）

名称	大小（比特）	方向	描述
pl_vcu_arvalid_axi_lite	1	输入	AXI4-Lite 读取地址有效信号
vcu_pl_arready_axi_lite	1	输出	AXI4-Lite 读取地址准备好信号
vcu_pl_rdata_axi_lite	32	输出	AXI4-Lite 读取数据总线
vcu_pl_rresp_axi_lite	2	输出	AXI4-Lite 读取响应信号
vcu_pl_rvalid_axi_lite	1	输出	AXI4-Lite 读取数据有效信号
pl_vcu_rready_axi_lite	1	输入	AXI4-Lite 读取地址准备好信号
VCU 编码器的缓存接口			
Vcu_pl_enc_al_l2c_rvalid	1	输出	读取数据有效
Pl_vcu_enc_al_l2c_rready	1	输入	读取数据准备好了
Vcu_pl_enc_al_l2c_addr	17	输出	地址
Pl_vcu_enc_al_l2c_rdata	320	输入	读取数据
Vcu_pl_enc_al_l2c_wvalid	1	输出	写入数据有效
Vcu_pl_enc_al_l2c_wdata	320	输出	写入数据

时钟设置

如需了解更多有关时钟设置的信息，请参阅第 7 章"时钟设置和复位"。

复位

如需了解更多有关复位的信息，请参阅第 7 章"时钟设置和复位"。

MCU 子系统

编码器块配有嵌入式 MCU，可用来运行 MCU 固件和控制硬件编码器核。如需了解有关 MCU 的信息，请参阅第 5 章"微控制器单元简介"。

数据路径

编码器块配有两个 128 位的 AXI4 主接口，从外部 DDR 存储器获取视频数据发送给处理器系统 (PS) 或可编程逻辑 (PL)。

从存储器中获取的数据包括：

- 源帧像素
- 参考帧像素
- 参考帧运动矢量
- slice/帧参数：lambda 表、缩放列表、 QP 控件、 QP 表
- 剩余数据 （仅限 H.264）

写入存储器的数据包括：

- 剩余数据 （仅限 H.264）

- 重建的帧像素
- 重建的帧运动矢量
- 编码比特流

控制路径

APU 每帧访问 MCU 从接口一次 （APU 会向 IP 核发送帧级命令）。该接口不需要快速数据路径。在帧级触发中断即可在每个帧处理结束时唤醒 APU。这些命令由嵌入式 MCU 进行处理 （嵌入式 MCU 为解码器块硬件生成 slice 和块命令）。如需了解更多信息，请参阅第 5 章"微控制器单元简介"。

编码器缓存

编码器块的缓存控制器用于管理对编码器缓存的读写访问，而编码器缓存可存储来自参考帧的像素数据。它从系统存储器中的参考帧预取数据块，并将它们存储在编码器缓存中。编码器缓存存储来自参考帧的亮度和色度像素，这样当编码器需要它们时，它们就会显示在缓存中。编码器缓存必须是一个连续的内存访问区域 (CMA)，并且应该与 32 字节的边界对齐。要查看每个 EV 器件可用的器件内存，请参阅 《Zynq UltraScale+ MPSoC 数据表：概述》[参照 15]。

根据所需访问模式的内存控制器效率来计算降低的系统总带宽。如果计算的带宽不足，则启用编码器缓存。

可选的编码器缓存可用来降低存储器的带宽。此选项可能会略微降低视频质量。有关更多信息，请参阅表 12-16。除了大小之外，没有用户控件可来用来调整编码器缓存的使用。

注释：要启用编码器缓存，请将 `prefetch-buffer` 参数传递到使用硬件的 GStreamer 流水线中。例如：

```
gst-launch-1.0 videotestsrc ! omxh265enc prefetch-buffer=true ! fakesink
```

表 3-4：器件的可用内存

	ZU4EV	ZU5EV	ZU7EV
Block RAM (MB)	0.56	0.63	1.37
UltraRam (MB)	1.68	2.25	3.37

重要提示：使用编码器块缓存可降低外部 DDR 存储器的带宽要求。有关存储器带宽要求方面的更多信息，请参阅第 11 章中的“Vivado Design Suite 集成设计环境”。

4:2:0 和 4:2:2 采样格式所需的编码器缓存存储大小分别显示在表 3-5 和表 3-6 中。

表 3-5：4:2:0 采样的编码器缓存存储器要求

编码器设置	1920×1080	3840×2160	
	8 bpc	8 bpc	10 bpc
B-frame=NONE 运动矢量范围 = LOW	105 KB	291 KB	364 KB
B-frame=STANDARD 运动矢量范围=MEDIUM	395 KB	1,128 KB	1,410 KB
B-frame=HIERARCHICAL 运动矢量范围 = HIGH	761 KB	2,250 KB	2,813 KB

表 3-6：4:2:2 采样的编码器缓存存储器要求

编码器设置	1920×1080	3840×2160	
	8 bpc	8 bpc	10 bpc
B-frame=NONE 运动矢量范围 = LOW	139 KB	388 KB	485 KB
B-frame=STANDARD 运动矢量范围=MEDIUM	526 KB	1,504 KB	1,880 KB
B-frame=HIERARCHICAL 运动矢量范围 = HIGH	1,014 KB	3,000 KB	3,750 KB

注释：要查找最精确的帧缓存大小，请使用 VCU GUI 中的“Advance Configuration > Resource Summary”菜单选项。

编码器缓存要求

编码器输入和输出要求如表 3-7 所示：

表 3-7：编码器缓存要求

要求	
输入缓存	
缓存数	对于 AVC：2 + B 帧数 (num-B)。 对于 HEVC：1 + B 帧数 (num-B)。 注意：在 look-ahead 模式下，按 look-ahead 帧数添加此值。
邻近的缓存	是
对齐	32
大小	stride * slice-height * chroma-mode。 宽度应该是 32 位对齐。 对于 AVC， slice 的高度应为 16 位对齐；对于 HEVC， slice 的高度应为 32 位对齐。 注意：适用于 slice 高度为 8 位对齐的，但硬件的无害请求不在缓存范围之内。
输出缓存	
缓存数	对于 AVC：2 + B 帧数 (num-B)。 对于 HEVC：1 + B 帧数 (num-B)。 注意：在子帧模式下，将此值乘以 num-slices。
邻近的缓存	是
对齐	32
大小	通过以下 API 来获取大小。 AL_GetMitigatedMaxNalSize (AL_TDimension tDim、AL_EChromaMode eMode、int iBitDepth)

注释：由于 VCU 使用多个内部编码器引擎，因此无法降低对输出缓存的要求。

DDR 存储器占用要求

DDR 存储缓存大小取决于以下各项：

- 视频分辨率
- 色度子采样

- 颜色深度
- 编码标准

缓存的数量取决于以下内容：

- 编码标准 – H.264 或 H.265
- 图像组 (GOP) 模式

表 3-8 显示的是不同编码方案在最坏情况下的存储器占情况。使用 IP Integrator 计算特定用例所需的存储器占用量。

表 3-8：编码器块的存储器占用

两个 B 帧示例	720p			1080p			2160p		
	缓存	每个缓存	合计	缓存	每个缓存	合计	缓存	每个缓存	合计
源帧	5	2.3 MB	12 MB	5	5.3 MB	27 MB	5	21.1 MB	106 MB
基准帧	3	2.2 MB	7 MB	3	5.0 MB	15 MB	3	19.9 MB	60 MB
重建的框架	1	2.2 MB	3 MB	1	5.0 MB	5 MB	1	19.9 MB	20 MB
中级缓存	2	0.0 MB	0 MB	2	0.0 MB	0 MB	2	41.0 MB	83 MB
运动矢量缓存	4	0.1 MB	1 MB	4	0.3 MB	1 MB	4	1.0 MB	4 MB
比特流缓存	2	1.1 MB	3 MB	2	2.5 MB	5 MB	2	9.9 MB	20 MB
其他缓存	1	0.0 MB	1 MB	1	0.0 MB	1 MB	1	0.1 MB	1 MB
合计			27 MB			54			294

CMA 大小要求

表 3-8 包含对基于分辨率和格式的 VCU 编码器/解码器理论上连续的存储器访问 （CMA） 的缓存要求。下面的尺寸与编码器或解码器的一个实例相对应。将这些数据乘以多流用例的流的个数。其他元素 （如 kmssink/v4l2src） 通常会将 CMA 的缓存要求再提高 10% - 15%。

表 3-9：VCU 编码器 CMA 要求

分辨率	4:2:2 10-bit AVC	4:2:2 10-bit HEVC	4:2:2: 8-bit s AVC	4:2:2: 8-bit s HEVC	4:2:0 10-bit AVC	4:2:0 10-bit HEVC	4:2:0 8-bit AVC	4:2:0 8-bit HEVC
3840×2160 (MB)	294	199	248	155	243	151	208	117
1920×1080 (MB)	54	52	42	40	42	40	32	31
1280×720 (MB)	27	26	21	20	20	19	17	16

存储器带宽

赛灵思建议尽可能使用最快的 DDR4 存储器接口。具体来说， 8x8 位的存储器接口比 4x16 位的存储器接口更有效，因为 x8 模式有 4 个 bank 组，而 x16 模式只有 2 个 bank；而且 DDR4 允许同时访问 bank 组。如需了解更多信息，请参阅 AR# 71209。

源帧格式

源帧缓存包含输入帧的像素。它包含两部分：亮度像素 (Luma) 及其后面跟着的色度像素。如图 3-2 所示，亮度像素以像素的光栅扫描顺序存储。色度像素以 U/V 交织像素光栅扫描的顺序进行存储，因此当使用 4:2:0 格式时，色度部分是 Luma 部分的一半，而当使用 4:2:2 时， Chroma 部分的大小与 Luma 部分的大小相同。编码图像缓存必须是一个连续的存储区域。

注释：VCU 接收半平面数据。

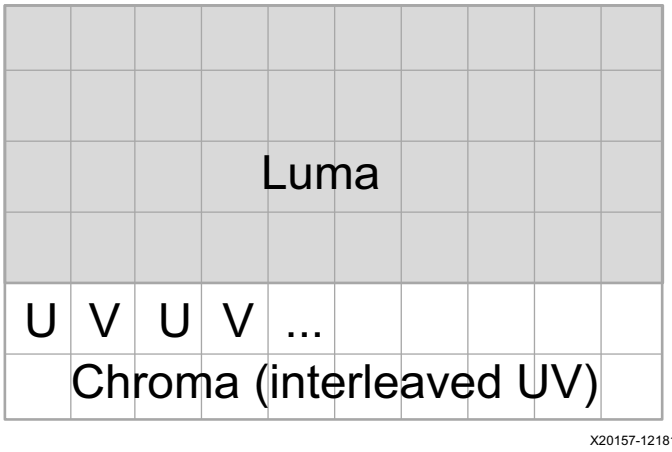

图 3-2：帧布局

外部存储器支持两种封装格式：每个组件 8 位或每个组件 10 位，如表 3-10 和表 3-11 分别所示。 8 位格式只能用于 8 位组件的深度，而 10 位格式只能用于 10 位 组件的深度。

表 3-10：8 位组件格式的源帧缓存格式

255 248	...				31 24	23 16	15 8	7 0	
$Y_{x+31,y}$	...				$Y_{x+3,y}$	$Y_{x+2,y}$	$Y_{x+1,y}$	$Y_{x,y}$	
... （以像素光栅扫描顺序排列的所有亮度） ...									
255 248	247 240	...			31 24	23 16	15 8	7 0	
$V_{x+15,y}$	$U_{x+15,y}$	...			$V_{x+1,y}$	$U_{x+1,y}$	$V_{x,y}$	$U_{x,y}$	
... （以像素光栅扫描顺序排列的所有交织色度） ...									

表 3-11：10 位组件格式的源帧缓存格式

255	254	253 244							31	30	29 20	19 10	9 0
0	0	$V_{x+23,y}$							0	0	$Y_{x+2,y}$	$Y_{x+1,y}$	$Y_{x,y}$
... （以像素光栅扫描顺序排列的所有亮度） ...													
255	254	253 244	...	63	62	61 52	51 42	41 32	31	30	29 20	19 10	9 0

表 3-11：10 位组件格式的源帧缓存格式

0	0	$V_{x+11,y}$	...	0	0	$V_{x+2,y}$	$U_{x+2,y}$	$V_{x+1,y}$	0	0	$U_{x+1,y}$	$V_{x,y}$	$U_{x,y}$
...（以像素光栅扫描顺序排列的所有交织色度） ...													

编码器缓存必须是一个连续的内存区域，并且应该与 32 字节的边界对齐。

帧缓存的宽度 (pitch) 可以大于帧的宽度。当间距大于帧宽时，忽略超出图像宽度的每行中的像素，如图 3-3 所示。

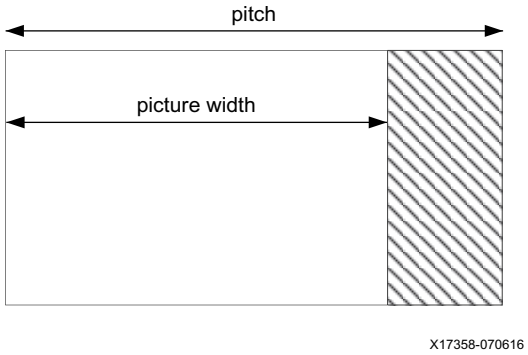

图 3-3：帧缓存间距

注释：

1. 编码器输入缓存的宽度和高度应为 32 的整数倍。
2. 解码器输出缓存的宽度是 256 字节的整数倍。高度是 64 字节的整数倍。
   例如，如果分辨率为 1920x1080，则解码器输出为 2048x1088。

编码器的块寄存器简介

表 3-12列出的是编码器的块寄存器。如需了解更多信息，请参阅 《Zynq UltraScale+ MPSoC 寄存器参考》(UG1087) [参照 14]。

表 3-12：编码器寄存器

寄存器	地址	宽度	类型	复位值	描述
MCU_RESET	0×A0009000	32	mixed	0x00000000	MCU 子系统复位
MCU_RESET_MODE	0xA0009004	32	mixed	0x00000001	MCU 复位模式
MCU_STA	0xA0009008	32	mixed	0x00000000	MCU 状态
MCU_WAKEUP	0xA000900C	32	mixed	0x00000000	MCU 唤醒
MCU_ADDR_OFFSET_IC0	0xA0009010	32	rw	0x00000000	MCU 命令高速缓存地址偏移 0
MCU_ADDR_OFFSET_IC1	0xA0009014	32	rw	0x00000000	MCU 命令高速缓存地址偏移 1
MCU_ADDR_OFFSET_DC0	0xA0009018	32	rw	0x00000000	MCU 数据高速缓存地址偏移 0
MCU_ADDR_OFFSET_DC1	0xA000901C	32	rw	0x00000000	MCU 数据高速缓存地址偏移 1
ITC_MCU_IRQ	0xA0009100	32	mixed	0x00000000	MCU 中断触发
ITC_CPU_IRQ_MSK	0xA0009104	32	rw	0x00000000	CPU 中断掩码
ITC_CPU_IRQ_CLR	0xA0009108	32	mixed	0x00000000	CPU 中断清除
ITC_CPU_IRQ_STA	0xA000910C	32	mixed	0x00000000	CPU 中断状态
AXI_BW	0xA0009204	32	rw	0x00000000	AXI 带宽测量窗口
AXI_ADDR_OFFSET_IP	0xA0009208	32	rw	0x00000000	视频数据地址偏移
AXI_RBW0	0xA0009210	32	ro	0x00000000	AXI 读取带宽状态 0
AXI_RBW1	0xA0009214	32	ro	0x00000000	AXI 读取带宽状态 1
AXI_WBW0	0xA0009218	32	ro	0x00000000	AXI 写入带宽状态 0
AXI_WBW1	0xA000921C	32	ro	0x00000000	AXI 写入带宽状态 1
AXI_RBL0	0xA0009220	32	rw	0x00000000	AXI 读取带宽限制器 0
AXI_RBL1	0xA0009224	32	rw	0x00000000	AXI 读取带宽限制器 1

提高 VCU 编码器的质量

编码视频的质量取决于目标比特率和视频内容的类型。如下所列，可以使用多个编码器参数来调整编码器的质量：

- 可以根据运动量调整 B 帧数 （Gop.NumB），例如，针对静态场景或视频会议增加到 2，或针对运动量大或帧速率高的序列减少到 0。
- 当某些序列部分的复杂度或运动量较低时， VBR 速率控制模式可以提升平均质量。
- 在用于视频会议或不需要随机访问时，您可能需要用 `LOW_DELAY_P` 更换 IPP...GOP，并可选择启用视频渐近刷新 (GDR) 的帧内刷新。
- 如果频繁更改场景，则可以通过启用 `ScnChgResilience` 设置来减少场景更改转换后的伪像。

- 如果系统可以检测到场景变化，则应该调用编码器的场景变化信号 API （例如，禁用 (`ScnChgResilience`)，以便编码器能够动态地调整编码参数和 GOP 模式。当使用控制软件测试应用时，可以在单独的输入文件 (`CmdFile`) 中提供场景更改信息。
- 如果针对的是最高 PSNR 值而不是主观质量，则建议设置 QPCtrlMode = UNIFORM_QP 和 ScalingList = FLAT。

XILINX

第 4 章

解码器块

引言

设计解码器块旨在按照 H.265 （HEVC）和 H.264 （AVC）标准来处理视频流。该设计对这些标准提供了全面支持，包括支持 8 位和 10 位颜色深度、 4:0:0、 4:2:0 和 4:2:2 的色度格式、以及高达 4K UHD @ 60 Hz 分辨率的性能表现。

解码器块可以有效地对视频进行解压缩。

IP 硬件可通过高带宽主接口直接访问系统数据总线，以传输出入外部存储器的视频数据。

IP 控制软件被分为两层。 VCU 控制软件在 APU 上运行，而 MCU 固件在 MCU 上运行 （MCU 是嵌入在硬件 IP 中的）。APU 通过从接口 （也是连接到系统总线的）与嵌入式 MCU 进行通信。 IP 硬件由嵌入式 MCU 通过寄存器映射来控制，通过内部的外设总线来设置解码参数。

特点

表 4-1 主要介绍解码器块的特点。

表 4-1：VCU 的特点

视频编码的特点	H.264	H.265
性能		
配置信息	Baseline (except FMO/ASO/RS) Constrained Baseline Main High High 10 High 4:2:2	Main Main Intra Main 10 Main 10 Intra Main 4:2:2 10 Main 4:2:2 10 Intra
级别	高达 5.2(2)	高达 5.1 High Tier(1)
速率为 @ 667 MHz 时的性能 • 32 个流 @ 720x480p @ 30 Hz • 8 个流 @ 1920x1080p @ 30 Hz • 4 个流 @ 1920x1080p @ @ 60 Hz • 2 个流 @ 3840x2160p @ 30 Hz • 1 个流 @ 3840x2160p @ 60 Hz • 1 个流 @ 7680x4320p @ 15 Hz	支持	支持

表 4-1：VCU 的特点 （续）

视频编码的特点	H.264	H.265
可配置分辨率 • 图像的宽度和高度：8 的整数倍。 • 最大宽度或高度：8192 • 最大图像尺寸：33.5 MP	支持 • 最小尺寸：80×96 • 最大宽度：8,192 • 最大高度：8,192	支持 • 最小尺寸：128×128 • 最大宽度：8184 （在 4/4.1 级或启用 WPP 时限制为 4,096） • 最大高度：8,192
可配置帧速率	支持	支持
编码工具		
采样位深度：8 bpc、 10 bpc	支持	支持
色度格式：YCbCr 4:2:0、YCbCr 4:2:2、Y-only（单色）	支持	支持
仅限 Progressive format	支持	支持

注释：

1. Support 8K15 使用 6 级的子级：最大亮度的图像大小高达 2^{25} 个样本， 5.1 级的其他约束适用 （例如最多 200 个 slice 和 11×10 个块），不支持宽度高于 4,096 的 WPP。
2. Support 8K15 使用 6 级的子级：最大亮度的图像大小高达 2^{25} 个样本，5.2 级的其他约束适用，最大 slice 大小为 65,535 个宏块，因此大小在 4K 以上则必须使用至少两个平衡的 slice。

表 4-2 介绍 VCU 解码器块支持的最大比特率。

表 4-2：VCU 解码器支持的最大比特率

标准	级别	配置信息	最大比特率 (Mbits/s)
H.264	4.2 (1080p60)	Baseline、 Main	50
		High	62.5
		High 10	150
		High 4:2:2	200
	5.2 (2160p60)	Baseline、 Main	240
		High	300
		High 10	720
		High 4:2:2	960 (CAVLC) 720 (CABAC)
H.264	4.1 (1080p60) High Tier	Main、 Main 10	50
		Main 4:2:2 10	84
		Main 4:2:2 10 Intra	167
	5.1 (2160p60) High Tier	Main、 Main 10	160
		Main 4:2:2 10	267
		Main 4:2:2 10 Intra	533

功能说明

图 4-1 显示的是解码器块的原理图。

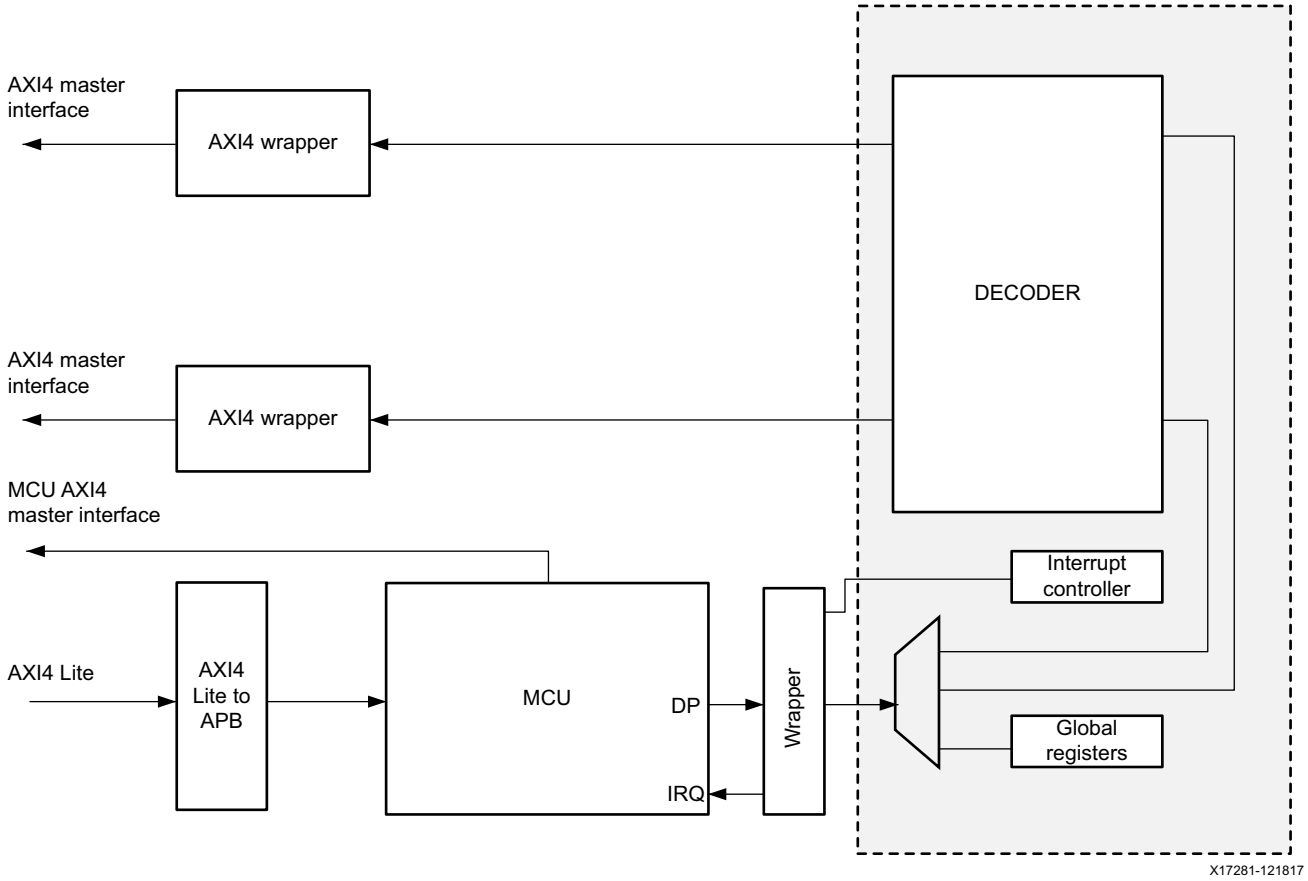

图 4-1：解码器块的详细架构

解码器块包括 H.265/H.264 解压缩引擎、控制寄存器和中断控制器块。

解码器块由 MCU 子系统控制。

系统 CPU 使用 32 位 AXI - Lite 从接口对 MCU 进行控制以配置解码器参数、开始处理视频帧并获取状态和结果。

两个 128 位 AXI-4 主接口用来获取出入系统存储器的视频输入数据，并存储出入系统存储器的视频输出数据。

AXI-4 主接口用于获取 MCU 软件并对其他 MCU 数据执行加载或存储操作。

接口与端口

使用解码器的应用必须连接解码器的所有端口 （即以 `m_axi_dec` 开头的端口）。表 4-3 显示解码器块的 AXI-4 主接口端口。

表 4-3：解码器端口

名称	宽度	方向	描述
vcu_pl_dec_araddr0/1	44	输出	AXI-4 ARADDR 信号
vcu_pl_dec_arburst0/1	2	输出	AXI-4 ARBURST 信号
vcu_pl_dec_arid0/1	4	输出	AXI-4 ARID 信号
vcu_pl_dec_arlen0/1	8	输出	AXI-4 ARLEN 信号
pl_vcu_dec_arready0/1	1	输入	AXI-4 ARREADY 信号
vcu_pl_dec_arsize0/1	3	输出	AXI-4 ARSIZE 信号
vcu_pl_dec_arvalid0/1	1	输出	AXI-4 ARVALID 信号
vcu_pl_dec_awaddr0/1	44	输出	AXI-4 AWADDR 信号
vcu_pl_dec_awburst0/1	2	输出	AXI-4 AWBURST 信号
vcu_pl_dec_awid0/1	4	输出	AXI-4 AWID 信号
vcu_pl_dec_awlen0/1	8	输出	AXI-4 AWLEN 信号
pl_vcu_dec_awready0/1	1	输入	AXI-4 AWREADY 信号
vcu_pl_dec_awsize0/1	3	输出	AXI-4 AWSIZE 信号
vcu_pl_dec_awvalid0/1	1	输出	AXI-4 AWVALID 信号
pl_vcu_dec_bresp0/1	2	输入	AXI-4 BRESP 信号
vcu_pl_dec_bready0/1	1	输出	AXI-4 BREADY 信号
pl_vcu_dec_bvalid0/1	1	输入	AXI-4 BVALID 信号
pl_vcu_dec_bid0/1	4	输入	AXI-4 BID 信号
pl_vcu_dec_rdata0/1	128	输入	AXI-4 RDATA 信号
pl_vcu_dec_rid0/1	4	输入	AXI-4 RID 信号
pl_vcu_dec_rlast0/1	1	输入	AXI-4 RLAST 信号
vcu_pl_dec_rready0/1	1	输出	AXI-4 RREADY 信号
pl_vcu_dec_rresp0/1	2	输入	AXI-4 RRESP 信号
pl_vcu_dec_rvalid0/1	1	输入	AXI-4 RVALID 信号
vcu_pl_dec_wdata0/1	128	输出	AXI-4 WDATA 信号
vcu_pl_dec_wlast0/1	1	输出	AXI-4 WLAST 信号
pl_vcu_dec_wready0/1	1	输入	AXI-4 WREADY 信号
vcu_pl_dec_wvalid0/1	1	输出	AXI-4 WVALID 信号
vcu_pl_dec_awprot0/1	1	输出	AXI-4 AWPROT 信号（由系统级控制寄存器 (SLCR) 控制）
vcu_pl_dec_arprot0/1	1	输出	AXI-4 ARPROT 信号 （由 SLCR 控制）
vcu_pl_dec_awqos0/1	4	输出	AXI-4 AWQOS 信号 （由 SLCR 控制）
vcu_pl_dec_arqos0/1	4	输出	AXI-4 ARQOS 信号 （由 SLCR 控制）
vcu_pl_dec_awcache0/1	4	输出	AXI-4 AWCACHE 信号 （由 SLCR 控制）
vcu_pl_dec_arcache0/1	4	输出	AXI-4 ARCACHE 信号 （由 SLCR 控制）

时钟设置

如需了解更多有关时钟的信息，请参阅第 7 章"时钟设置和复位"。

复位

如需了解更多有关复位的信息，请参阅第 7 章"时钟设置和复位"。

数据路径

主接口从外部存储器输入下列几种类型的视频数据：

- 比特流
- 参考帧像素
- 共位图像的运动矢量
- 报头和残留数据

主接口输出：

- 已解码的帧像素
- 报头和残留数据
- 已解码的帧运动矢量 （当图像稍后用作共位图像时）

控制路径

APU 每帧访问 VCU 从接口一次 （APU 会向 IP 发送帧级命令）。因此，该接口不需要快速数据路径。在每帧结束时会生成中断。这些命令由嵌入式 MCU 来处理 （嵌入式 MCU 向解码器块硬件生成块级和 slice 级命令）。

解码器缓存要求

解码器输入和输出要求如表 4-4 所示。

表 4-4：解码器缓存要求

要求	
输入缓存	
缓存数	>=1
邻近的缓存	No
对齐	0
大小	Any > 0
输出缓存	
缓存数	对于 AVC：AL_AVC_GetMinOutputBuffersNeeded (AL_TStreamSettings tStreamSettings, int iStack)。 对于 HEVC：AL_HEVC_GetMinOutputBuffersNeeded (AL_TStreamSettings tStreamSettings, int iStack)。 注意：尺寸、色度模式和位深度都设定在流设置中。
邻近的缓存	Yes

表 4-4：解码器缓存要求 （续）

要求	
对齐	32
大小	stride * slice-height * chroma-mode。 注意：步幅和 slice 高度应为 64 位对齐。

注释：由于 VCU 可使用多个内部解码器引擎，因此无法降低输出缓存的要求。

存储器占用要求

解码器块的存储器情况占用取决于解码参数。

缓存大小取决于以下各项：

- 视频分辨率
- 色度子采样
- 颜色深度
- 编码标准 – H.264 或 H.265

缓存数量取决于编码标准。

表 4-5 显示了不同缓存大小在最坏情况下所需的存储器占用情况。

表 4-5：解码器块的存储器占用

两个 B 帧示例	720p			1080p			2160p		
	缓存数	每缓存	合计	缓存数	每缓存	合计	缓存数	每缓存	合计
输入比特流缓存	2	1.9 MB	3.8 MB	2	4.0 MB	8.0 MB	2	16.0 MB	32.0 MB
循环比特流缓存	1	9.5 MB	9.5 MB	1	20.1 MB	20.1 MB	1	80.0 MB	80.0 MB
基准帧	23	2.5 MB	56.6 MB	23	5.3 MB	122.2 MB	12	21.1 MB	253.1 MB
重建的框架	1	2.5 MB	2.5 MB	1	5.3 MB	5.3 MB	1	21.1 MB	21.1 MB
中级缓存	5	4.9 MB	24.4 MB	5	11.1 MB	55.4 MB	5	44.0 MB	220.0 MB
运动矢量缓存	23	0.2 MB	5.1 MB	23	0.5 MB	11.5 MB	12	2.0 MB	23.7 MB
slice 参数	5	6.1 MB	30.7 MB	5	6.1 MB	30.7 MB	5	6.1 MB	30.7 MB
其他缓存	1	3.9 MB	3.9 MB	1	3.9 MB	3.9 MB	1	3.9 MB	3.9 MB
合计			137 MB			258 MB			665 MB

CMA 大小要求

表 4-6 包含对 VCU 解码器连续访问存储器 （CMA）的缓存要求 （这些要求基于分辨率和格式，是理论上而言的要求）。下面列出的尺寸与一个编码器或解码器实例相对应。将这些数据乘以多流用例的流的个数。其他元素 （如 kmssink/v4l2src）通常会将 CMA 的缓存要求再提高 10% - 15%。

表 4-6：VCU 解码器的 CMA 要求

分辨率	4:2:2 10-bit AVC	4:2:2 10-bit HEVC	4:2:2: 8-bit s AVC	4:2:2: 8-bit s HEVC	4:2:0 10-bit AVC	4:2:0 10-bit HEVC	4:2:0 8-bit AVC	4:2:0 8-bit HEVC
3840×2160 (MB)	665	582	597	513	524	466	473	414
1920×1080 (MB)	258	214	232	190	208	167	188	148
1280×720 (MB)	137	104	120	87	144	82	101	69

存储器带宽

解码器存储器的带宽取决于帧速率、分辨率、颜色深度、色度格式和解码器配置信息。 LogiCORE™ IP 根据 GUI 中选择的视频参数来提供解码器的带宽估算值。

赛灵思建议尽可能使用最快的 DDR4 存储器接口。具体来说， 8x8 位的存储器接口比 4x16 位的存储器接口更有效，因为 x8 模式有 4 个 bank 组，而 x16 模式只有 2 个 bank 组；而且 DDR4 允许同时访问多个 bank 组。如需了解更多信息，请参阅 AR# 71209。

存储器格式

解码图像缓存包含解码像素。它包含两个部分：亮度像素 (Luma) 及其后面跟着的色度像素。亮度像素以像素光栅扫描顺序进行存储。色度像素以 U/V 交织像素光栅扫描的顺序进行存储，因此当使用 4:2:0 格式时，色度部分是 Luma 部分的一半，而当使用 4:2:2 时，Chroma 部分的大小与 Luma 部分的大小相同。解码图像缓存必须是一个连续的存储区域。

注释：解码器输出缓存的宽度是 256 个字节的整数倍。高度是 64 个字节的整数倍。例如：如果是 1920x1080 的分辨率，解码器输出为 2048x1088。

外部存储器支持两种封装格式：每个组件 8 位或每个组件 10 位，如表 4-7 和表 4-8 分别所示。 8 位格式只能用于 8 位组件的深度，而 10 位格式只能用于 10 位组件的深度。表 4-7 和表 4-8 显示的是解码器块支持的光栅扫描格式，用于 8 位和 10 位颜色深度。

表 4-7：8 位组件格式的源帧缓存格式

255 248	...	31 24	23 16	15 8	7 0
$Y_{x+31,y}$	...	$Y_{x+3,y}$	$Y_{x+2,y}$	$Y_{x+1,y}$	$Y_{x,y}$
... （像素光栅扫描顺序中的所有亮度） ...					
255 248	247 240 ...	31 24	23 16	15 8	7 0
$V_{x+15,y}$	$U_{x+15,y}$ ...	$V_{x+1,y}$	$U_{x+1,y}$	$V_{x,y}$	$U_{x,y}$
... （以像素光栅扫描顺序排列的所有交织色度） ...					

表 4-8：10 位组件格式的源帧缓存格式

255	254	253 244							31	30	29 20	19 10	9 0
0	0	$V_{x+23,y}$							0	0	$Y_{x+2,y}$	$Y_{x+1,y}$	$Y_{x,y}$
... （像素光栅扫描顺序中的所有亮度） ...													
255	254	253 244	...	63	62	61 52	51 42	41 32	31	30	29 20	19 10	9 0

表 4-8：10 位组件格式的源帧缓存格式

0	0	$V_{x+11,y}$	...	0	0	$V_{x+2,y}$	$U_{x+2,y}$	$V_{x+1,y}$	0	0	$U_{x+1,y}$	$V_{x,y}$	$U_{x,y}$
...（以像素光栅扫描顺序排列的所有交织色度） ...													

帧缓存的宽度间距 (pitch) 可以大于帧宽度，结果在连续像素线之间会存在 （间距 - 宽度）忽略值。

解码器块的寄存器简介

表 4-9 列出的是解码器块寄存器。如需了解更多信息，请参阅 《Zynq UltraScale+ MPSoC 寄存器参考》(UG1087) [参照 14]。

表 4-9：解码器的寄存器

寄存器名称	地址	宽度	类型	复位值	描述
MCU_RESET	0xA0029000	32	mixed	0x00000000	MCU 子系统复位
MCU_RESET_MODE	0xA0029004	32	mixed	0x00000001	MCU 复位模式
MCU_STA	0xA0029008	32	mixed	0x00000000	MCU 状态
MCU_WAKEUP	0xA002900C	32	mixed	0x00000000	MCU 唤醒
MCU_ADDR_OFFSET_IC0	0xA0029010	32	rw	0x00000000	MCU 命令高速缓存地址偏移 0
MCU_ADDR_OFFSET_IC1	0xA0029014	32	rw	0x00000000	MCU 命令高速缓存地址偏移 1
MCU_ADDR_OFFSET_DC0	0xA0029018	32	rw	0x00000000	MCU 数据高速缓存地址偏移 0
MCU_ADDR_OFFSET_DC1	0xA002901C	32	rw	0x00000000	MCU 数据高速缓存地址偏移 1
ITC_MCU_IRQ	0xA0029100	32	mixed	0x00000000	MCU 中断触发
ITC_CPU_IRQ_MSK	0xA0029104	32	rw	0x00000000	CPU 中断掩码
ITC_CPU_IRQ_CLR	0xA0029108	32	mixed	0x00000000	CPU 中断清除
ITC_CPU_IRQ_STA	0xA002910C	32	mixed	0x00000000	CPU 中断状态
AXI_BW	0xA0029204	32	rw	0x00000000	AXI 带宽测量窗口
AXI_ADDR_OFFSET_IP	0xA0029208	32	rw	0x00000000	视频数据地址偏移
AXI_RBW0	0xA0029210	32	ro	0x00000000	AXI 读取带宽状态 0
AXI_RBW1	0xA0029214	32	ro	0x00000000	AXI 读取带宽状态 1
AXI_WBW0	0xA0029218	32	ro	0x00000000	AXI 写入带宽状态 0
AXI_WBW1	0xA002921C	32	ro	0x00000000	AXI 写入带宽状态 1
AXI_RBL0	0xA0029220	32	rw	0x00000000	AXI 读取带宽限制器 0
AXI_RBL1	0xA0029224	32	rw	0x00000000	AXI 读取带宽限制器 1

注释：VCU 编码器和解码器输出流存储在 DDR 存储器 （PS 或 PL）中，不能直接从编码器路由到解码器，反之亦然，因此需要将 DDR （PS 或 PL）与 VCU 编码器和解码器一起使用。

第 5 章

微控制器单元简介

引言

Video Codec Unit (VCU) 核包括两个微控制器单元 (MCU) 子系统，用于运行 MCU 固件和控制编码器和解码器块。编码器和解码器块各自都有自己的、用来执行固件的 MCU。 MCU 有 32 位的 RISC 架构，能够执行流水线数据传输。 MCU 有内部指令、数据高速缓存和 AXI 主接口，可与外部存储器连接。

功能描述

图 5-1 显示的是 MCU 的顶层接口和详细架构。

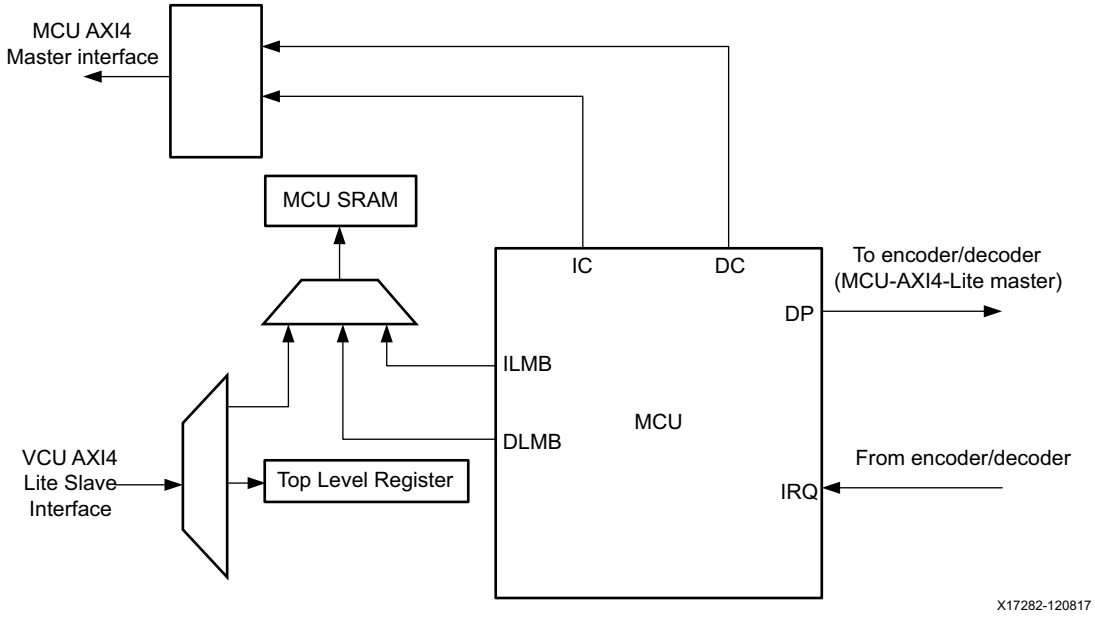

图 5-1：MCU (顶层)

MCU 通过 32 位的 AXI4-Lite 主接口来连接外设。它带有本地存储器总线和 AXI-4 32 位的指令和数据高速缓存接口。

MCU 块带有与 CPU 共享的 32 KB 的本地内存，用于启动和邮箱通信等内部操作。 MCU 带有 32 KB 的指令高速缓存，而且还配置有 32 字节的高速缓存行宽。它还带有 4 KB 的指令高速缓存，而且还配置有 16 字节的高速缓存行宽。数据高速缓存带有直写高速缓存实现。

接口与端口

表 5-1 显示的是 MCU 的 AXI4 指令和数据高速缓存接口端口。

表 5-1：MCU 端口

端口	大小（比特）	方向	描述
vcu_pl_mcu_m_axi_ic_dc_araddr	44	输出	AXI-4 读取地址
vcu_pl_mcu_m_axi_ic_dc_arburst	2	输出	AXI-4 读取突发类型
vcu_pl_mcu_m_axi_ic_dc_arcache	4	输出	AXI-4 ARCACHE 值
vcu_pl_mcu_m_axi_ic_dc_arid	3	输出	AXI-4 读取主 ID
vcu_pl_mcu_m_axi_ic_dc_arlen	8	输出	AXI-4 读取突发大小
vcu_pl_mcu_m_axi_ic_dc_arlock	1	输出	AXI-4 ARLOCK 信号
vcu_pl_mcu_m_axi_ic_dc_arprot	3	输出	AXI-4 ARPROT 信号
vcu_pl_mcu_m_axi_ic_dc_arqos	4	输出	AXI-4 ARQOS 信号
pl_vcu_mcu_m_axi_ic_dc_arready	1	输入	AXI-4 ARREADY 信号
vcu_pl_mcu_m_axi_ic_dc_arsize	3	输出	AXI-4 ARSIZE 信号
vcu_pl_mcu_m_axi_ic_dc_arvalid	1	输出	AXI-4 ARVALID 信号
vcu_pl_mcu_m_axi_ic_dc_awaddr	44	输出	AXI-4 AWADDR 信号
vcu_pl_mcu_m_axi_ic_dc_awburst	2	输出	AXI-4 AWBURST 信号
vcu_pl_mcu_m_axi_ic_dc_awcache	4	输出	AXI-4 AWCACHE 信号
vcu_pl_mcu_m_axi_ic_dc_awid	3	输出	AXI-4 AWID 信号
vcu_pl_mcu_m_axi_ic_dc_awlen	8	输出	AXI-4 AWLEN 信号
vcu_pl_mcu_m_axi_ic_dc_awlock	1	输出	AXI-4 AWLOCK 信号
vcu_pl_mcu_m_axi_ic_dc_awprot	3	输出	AXI-4 AWPROT 信号
vcu_pl_mcu_m_axi_ic_dc_awqos	4	输出	AXI-4 AWQOS 信号
pl_vcu_mcu_m_axi_ic_dc_awready	1	输入	AXI-4 AWREADY 信号
vcu_pl_mcu_m_axi_ic_dc_awsize	3	输出	AXI-4 AWSIZE 信号
vcu_pl_mcu_m_axi_ic_dc_awvalid	1	输出	AXI-4 AWVALID 信号
pl_vcu_mcu_m_axi_ic_dc_bid	3	输入	AXI-4 BID 信号
vcu_pl_mcu_m_axi_ic_dc_bready	1	输出	AXI-4 BREADY 信号
pl_vcu_mcu_m_axi_ic_dc_bresp	2	输入	AXI-4 BRESP 信号
pl_vcu_mcu_m_axi_ic_dc_bvalid	1	输入	AXI-4 BVALID 信号
pl_vcu_mcu_m_axi_ic_dc_rdata	32	输入	AXI-4 RDATA 信号
pl_vcu_mcu_m_axi_ic_dc_rid	3	输入	AXI-4 RID 信号

表 5-1：MCU 端口 （续）

端口	大小（比特）	方向	描述
pl_vcu_mcu_m_axi_ic_dc_rlast	1	输入	AXI-4 RLAST 信号
vcu_pl_mcu_m_axi_ic_dc_rready	1	输出	AXI-4 RREADY 信号
pl_vcu_mcu_m_axi_ic_dc_rresp	2	输入	AXI-4 RRESP 信号
pl_vcu_mcu_m_axi_ic_dc_rvalid	1	输入	AXI-4 RVALID 信号
vcu_pl_mcu_m_axi_ic_dc_wdata	32	输出	AXI-4 WDATA 信号
vcu_pl_mcu_m_axi_ic_dc_wlast	1	输出	AXI-4 WLAST 信号
pl_vcu_mcu_m_axi_ic_dc_wready	1	输入	AXI-4 WREADY 信号
vcu_pl_mcu_m_axi_ic_dc_wstrb	4	输出	AXI-4 WSTRB 信号
vcu_pl_mcu_m_axi_ic_dc_wvalid	1	输出	AXI-4 WVALID 信号

表 5-2 总结了 MCU 子系统的 AXI4-Lite 从接口端口。

表 5-2：AXI4-Lite 从端口

端口	宽度	方向	描述
pl_vcu_awaddr_axi_lite_apb	20	输入	AXI-4 AWADDR 信号
pl_vcu_awprot_axi_lite_apb	3	输入	AXI-4 AWPROT 信号
pl_vcu_awvalid_axi_lite_apb	1	输入	AXI-4 AWVALID 信号
vcu_pl_awready_axi_lite_apb	1	输出	AXI-4 AWREADY 信号
pl_vcu_wdata_axi_lite_apb	32	输入	AXI-4 WDATA 信号
pl_vcu_wstrb_axi_lite_apb	4	输入	AXI-4 WSTRB 信号
pl_vcu_wvalid_axi_lite_apb	1	输入	AXI-4 WVALID 信号
vcu_pl_wready_axi_lite_apb	1	输出	AXI-4 WREADY 信号
vcu_pl_bresp_axi_lite_apb	2	输出	AXI-4 BRESP 信号
vcu_pl_bvalid_axi_lite_apb	1	输出	AXI-4 BVALID 信号
pl_vcu_bready_axi_lite_apb	1	输入	AXI-4 BREADY 信号
pl_vcu_araddr_axi_lite_apb	20	输入	AXI-4 ARADDR 信号
pl_vcu_arprot_axi_lite_apb	3	输入	AXI-4 ARPROT 信号
pl_vcu_arvalid_axi_lite_apb	1	输入	AXI-4 ARVALID 信号
vcu_pl_arready_axi_lite_apb	1	输出	AXI-4 ARREADY 信号
vcu_pl_rdata_axi_lite_apb	32	输出	AXI-4 RDATA 信号
vcu_pl_rresp_axi_lite_apb	2	输出	AXI-4 RRESP 信号
vcu_pl_rvalid_axi_lite_apb	1	输出	AXI-4 RVALID 信号
pl_vcu_rready_axi_lite_apb	1	输入	AXI-4 RREADY 信号

控制流程

应用复位后， MCU 保持睡眠模式，直到内核器件驱动将固件启动代码下载到 MCU 的内部存储器中。在下载启动代码并完成 MCU 初始化序列之后，控制软件使用在 MCU 的内部 SRAM 存储器中实现的邮箱机制与 MCU 通信。MCU 向控制软件发送确认并执行编码/解码操作。当请求的操作完成后， MCU 会将状态传送给控制软件。

如需了解有关软件和 MCU 固件的详情，请参阅第 12 章"应用软件开发"。

MCU 寄存器简介

表 5-3 列出的是 MCU 寄存器。如需了解更多信息，请参阅 《Zynq UltraScale+ MPSoC 寄存器参考》(UG1087) [参照 14]。

表 5-3：编码器的 MCU 寄存器

寄存器	地址	宽度	类型	复位值	描述
MCU_RESET	0xA0009000	32	mixed	0x00000000	MCU 子系统复位
MCU_RESET_MODE	0xA0009004	32	mixed	0x00000001	MCU 复位模式
MCU_STA	0xA0009008	32	mixed	0x00000000	MCU 状态
MCU_WAKEUP	0xA000900C	32	mixed	0x00000000	MCU 唤醒
MCU_ADDR_OFFSET_IC0	0xA0009010	32	rw	0x00000000	MCU 命令高速缓存地址偏移 0
MCU_ADDR_OFFSET_IC1	0xA0009014	32	rw	0x00000000	MCU 命令高速缓存地址偏移 1
MCU_ADDR_OFFSET_DC0	0xA0009018	32	rw	0x00000000	MCU 数据高速缓存地址偏移 0
MCU_ADDR_OFFSET_DC1	0xA000901C	32	rw	0x00000000	MCU 数据高速缓存地址偏移 1

XILINX

第 6 章

Zynq UltraScale+ EV 架构 Video Codec Unit DDR4 LogiCORE IP

引言

基于赛灵思 Zynq UltraScale+™ 架构的 FPGA VCU DDR4 IP 核是一个组合式预制控制器和物理层 (PHY)，用于将 Zynq UltraScale+ MPSoC 可编程逻辑 (PL) 用户设计与 DDR4 SDRAM 连接。此 DDR4 控制器仅适用于 Zynq UltraScale+ MPSoC EV 产品，而不适用于任何其他赛灵思器件。

本章主要提供有关如何使用、自定义和仿真基于 Zynq UltraScale+ 架构的 MPSoC 的 LogiCORE™ IP DDR4 SDRAM 方面的信息。另外，还对核架构进行介绍，并提供了有关自定义和核连接方面的详细信息。

LogiCORE™ IP 相关信息表	
核相关信息	
支持的器件系列	Zynq® UltraScale+™ EV 器件
支持的用户接口	AXI4 存储器映射
资源	性能与资源利用网页
核提供的材料	
设计文件	RTL
示例设计	不提供
测试平台	不提供
约束文件	赛灵思设计约束 (XDC)
仿真模型	不提供
支持的 S/W 驱动	不适用
经过测试的设计流程	
设计入门	Vivado® Design Suite
综合	Vivado 综合
支持	
由赛灵思支持网页提供	

简介

赛灵思 UltraScale 架构包括 DDR4 SDRAM 核。这些核可提供与 SDRAM 存储器类型连接的解决方案，支持完整的存储器控制器和仅物理层 (PHY) 解决方案。我们还针对 VCU 流量模式 （特别是针对解码器对存储器的访问功能）对控制器进行了最优化。 DDR4 核的 UltraScale 架构由以下高级块组成：

控制器 – 控制器接受来自用户接口的突发数据传输，并生成出入 SDRAM 的数据传输。控制器负责 SDRAM 时序参数并会进行刷新。它可以合并写入和读取数据传输，从而减少总线上的死循环次数。控制器还会重新对命令进行排序以提高数据总线对 SDRAM 的利用率。

物理层 – 物理层为 SDRAM 提供高速接口。该层包括 FPGA 内部所需的硬块和软块校准逻辑，以确保与 SDRAM 连接的硬块的最佳时序。应用逻辑负责所有 SDRAM 数据传输、定时和刷新。

这些硬块包括：

- 数据序列化与传输
- 数据捕获与反序列化
- 高速时钟的生成和同步
- 每个引脚的粗略和精细延迟元件，及其电压和温度跟踪功能

软块包括：

- 存储器初始化 - 校准模块为特定存储器类型提供符合 JEDEC® 标准的初始化例程。如果需要，可以绕过初始化过程中的延迟以加速仿真时间。
- 校准 – 校准模块提供了一种完整的方法，用于设置硬块和软 IP 中的所有延迟以便使用存储器接口。每个位都经过单独培训，然后组合在一起以确保最佳的接口性能。另外，还可以通过赛灵思调试工具获得校准过程的结果。完成校准后， PHY 层还会为 SDRAM 提供原始接口。

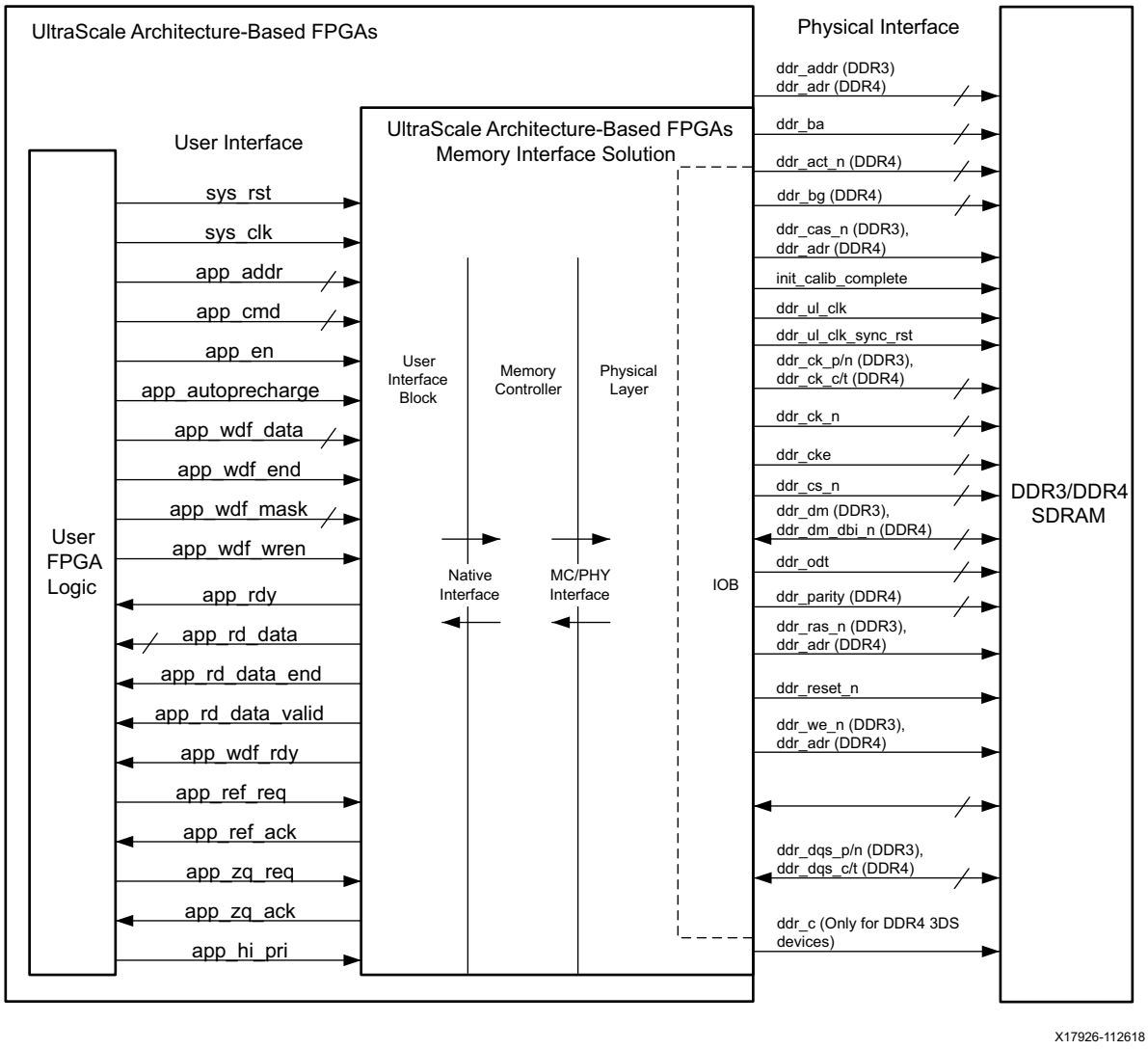

图 6-1：基于 UltraScale 架构的 FPGA DDR4 存储器接口解决方案

DDR4 DSDRAM 特点总结

重要提示: Zynq UltraScale+ EV 架构的 Video Codec Unit DDR4 LogiCORE IP 只能与 Zynq® UltraScale+™ MPSoC 器件的 H.264/H.265 Video Codec Unit (VCU) 核一起使用。

- 支持五个高性能 AXI 端口，用于连接解码器 0、解码器 1、 MCU、 PS 和显示控制器接口
- 组件/SODIMM 支持 64 位接口宽度
  - 支持 2133、 2400 和 2667 的速度箱
  - 支持 x8 (KVR21SE15S8/4)、 x16 (MT40A256M16GE-07) 和 SODIMM X8 的组件存储器
  - 支持 Zynq-UltraScale+ EV 系列 -1e、 -2、 -3e 器件和 2133 MT/s、 2400 MT/s、 2667 MT/s 的支持频率。
  - 不支持 -1i *器件
  - 支持 AXI
  - 对 FIFO 进行重新排序以提高效率
- x8 和 x16 器件支持

- 支持 8 字突发
- 支持 ODT
- 为 DDR4 写入均衡支持 （需要 fly-by 路由拓扑的组件设计）
- 符合 JEDEC 标准的 DDR4 初始化支持
- Verilog/VHDL 中的加密源代码交付
- 基于开放的、封闭的和数据传输的预充电控制器策略
- 通过 Vivado 硬件管理器提供的接口校准和培训信息
- 目标技术 （物理接口）：
  - 使用 MIG 生成的 phy-only 设计
    - UltraScale+ (Zynq)
    - 支持 DDR4 功能
- 控制器：
  - 通过下列各项实现多端口随机接入应用的高效率：
    - 重新排序
    - 突发最大化
    - 高度的前瞻性
    - Ping pong 模式
  - 高频率可以通过可配置的流水线来实现
  - 低资源数

许可和订购

根据赛灵思最终用户许可证条款，赛灵思 LogiCORE IP 模块随附赛灵思 Vivado 设计套件的 XCZU * EV 器件免费提供。有关其他赛灵思 LogiCORE IP 模块的信息，请访问赛灵思知识产权页面。有关其他赛灵思 LogiCORE IP 模块和工具的信息，请联系您所在当地的赛灵思销售代表。

许可证检查器

如果 IP 需要许可证密钥，则必须验证密钥。 Vivado® 设计工具设置有多个许可证检查点，旨在通过此流程对获得许可证的 IP 进行门控。如果许可证检查成功，则可以继续生成 IP。否则，系统会因错误而停止生成。许可证检查点由以下工具强制执行：

- Vivado 综合
- Vivado 实现
- write_bitstream (Tcl 命令)

产品规格

标准

该核支持符合 JEDEC® 固态技术协会 JESD79-4、 DDR4 SDRAM 标准的 DRAM。

如需了解更多有关 UltraScale™ 架构文档的信息，请参阅附录 B 中的“参考资料”。

性能

系统吞吐量

表 6-1 显示的是运行 4kp60、 4:2:2、 10 位解码显示流水线的存储器控制器性能。这些吞吐量数字基于 2133 DRAM 速度下控制器的 x8 配置。

表 6-1：存储器控制器的性能

带宽	解码器端口 0	解码器端口 1	显示	视频流量生成器 (PL)
读取带宽	1879.78 MB/s	1913.62 MB/s	1328.09 MB/s	950.72 MB/s
写入带宽	895.32 MB/s	792.01 MB/s	不适用	不适用

表 6-2 总结了解码图像/秒的性能 （针对使用 4 个 B 帧来解码 4kp60 视频流、 4:2:2、 10 位操作）。

表 6-2：解码性能

速度	仅帧率解码	帧率解码+显示
X8	93 帧/秒	74 帧/秒
X16	89 帧/秒	70 帧/秒

资源利用

有关性能和资源利用率的完整详细信息，请访问性能和资源利用率 (DDR4)。

表 6-3：IP 性能

名称	CLU LUTS (23040)	CLB 寄存器 (460800)	CARRY8 (28800)	F7 MUXES (11520)	F8 MUXES (57600)	BLOCK RAM TILE (312)	DSPS (1728)	HPIOBDIFFINBUF (192)	BITSLICE_RX_TX (416)	GLOBAL CLOCK BUFFERS (544)	PLL (16)	MMCM (8)
vcu_ddr4_controller	38586	34771	1834	1408	502	123	3	9	106	9	3	1

端口描述

图 6-2 示出了 VCU DDR4 控制器的框图（有五个 axi 端口、`s_axi_clk`、`s_axi_rst`、`c0_sys_clk`、`sys_rst`）。

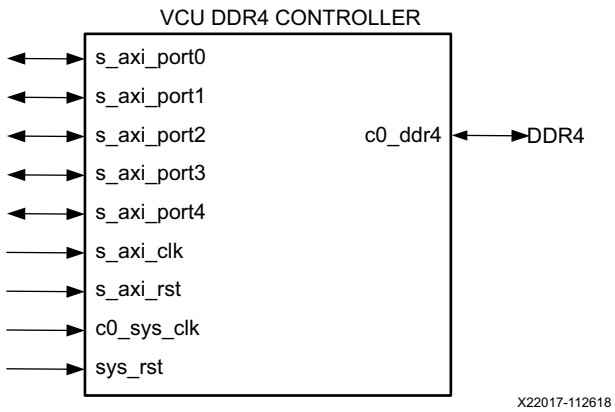

图 6-2：AXI 时钟端口/复位端口连接

S_AXI_CLK/S_AXI_RST：AXI 端口适用于这种时钟和复位，使用的是低电平有效复位。

C0_SYS_CLK/SYS_RST：这是用于 VCIU DDR4 控制器的实际时钟， x16 配置的频率为 125 MHZ， x8 配置的频率为 300 MHZ。使用的是低电平有效复位。

AXI 端口

AXI 规范在
http://www.arm.com/products/system-ip/amba/amba-open-specifications.php 上可用 （注册后）。

BA317 的 AXI 端口与 AXI4 规范 [1] 相匹配，但它们不进行重新排序。它们支持所有突发类型 （窄传输、增量突发、包装突发、固定突发）。

表 6-4 和表 6-5 代表 AXI 端口号，即 0、 1、 2、 3、 4。

表 6-4：AXI 写入端口

信号	方向	描述
写入地址通道		
pl_barco_slot_awidX [15:0]	In	
pl_barco_slot_awaddrX [31:0]	In	字节地址
pl_barco_slot_awlenX [7:0]	In	突发长度 （传输次数 - 1）
pl_barco_slot_awsizeX [2:0]	In	传输宽度： 000：8 比特 001：16 比特 010：32 比特 011：64 比特 ...
pl_barco_slot_awburstX [1:0]	In	突发类型： 00：固定 01：增量 10：包装 11：保留
pl_barco_slot_awvalidX	In	
barco_pl_slot_awreadyX	Out	
写入数据通道		

表 6-4：AXI 写入端口 （续）

信号	方向	描述
pl_barco_slot_widX[15:0]	In	
pl_barco_slot_wdataX [127:0]	In	写入数据
pl_barco_slot_wstrbX [15:0]	In	字节使能
pl_barco_slot_wlastX	In	端口未使用
pl_barco_slot_wvalidX	In	
barco_pl_slot_wreadyX	Out	
写入响应通道		
barco_pl_slot_bidX[15:0]	Out	
barco_pl_slot_brespX[1:0]	Out	绑定为零 (OKAY)。
barco_pl_slot_bvalidX	Out	
pl_barco_slot_breadyX	In	

表 6-5：AXI 读取端口

信号	方向	描述
读取地址通道		
pl_barco_slot_aridX[15:0]	In	
pl_barco_slot_araddrX [31:0]	In	字节地址
pl_barco_slot_arlenX[7:0]	In	突发长度 （传输次数 - 1）
pl_barco_slot_arsizeX[2:0]	In	传输宽度： 000：8 比特 001：16 比特 010：32 比特 011：64 比特 ...
pl_barco_slot_arburstX [1:0]	In	突发类型： 00：固定 01：增量 10：包装 11：保留
pl_barco_slot_arvalidX	In	
barco_pl_slot_arreadyX	Out	
读取数据通道		
barco_pl_slot_ridX[15:0]	Out	
barco_pl_slot_rdataX[127:0]	Out	读取数据
barco_pl_slot_rrespX[1:0]	Out	绑定为零 (OKAY)
barco_pl_slot_rlastX	Out	
barco_pl_slot_rvalidX	Out	
pl_barco_slot_rreadyX	In	

字节序

BA317 端口通常使用 little-endian 约定。表 6-6显示的是从不同类型的端口访问内存中的相同数据。

表 6-6：BA317 端口

端口	字节序	地址	数据
64 位缓存端口	小端	0x0	0x0807060504030201
32 位缓存端口	小端	0x0	0x04030201
		0x1	0x08070605
16 位缓存端口	小端	0x0	0x0201
		0x1	0x0403
		0x2	0x0605
		0x3	0x0807
8 位缓存端口	小端	0x0	0x01
		0x1	0x02
		0x2	0x03
		0x3	0x04
		0x4	0x05
		0x5	0x06
		0x6	0x07
		0x7	0x08
128 位 AXI 端口	小端	0x0	0x100F0E0D0C0B0A090807060504030201
32 位 AXI 端口	小端	0x0	0x04030201
		0x1	0x08070605

物理接口

表 6-7表列出的是系统支持的 SDRAM 和 FPGA 器件的所有可用物理接口。系统可能还会支持一些其他的 FPGA，因为它们与列出的 FPGA 兼容。

表 6-7：物理接口

PHY	SDRAM	FPGA	Rate
phyXilinxUltrascale	DDR4	Zynq-US+ EV	Quad

供应商指南适用。

UltraScale/UltraScale+ Xilinx Phy (phyXilinxUltrascale)

欲知详情，请参阅 《LogiCORE IP UltraScale 架构的 FPGAs Memory IP 产品指南》(PG150) [参照 10]。

核架构

本节介绍基于 UltraScale™ 架构的 FPGA 存储器接口解决方案核，还对模块和接口进行了简要概述。

简介

基于 UltraScale 架构的 FPGA 存储器接口解决方案如图 6-3 所示。

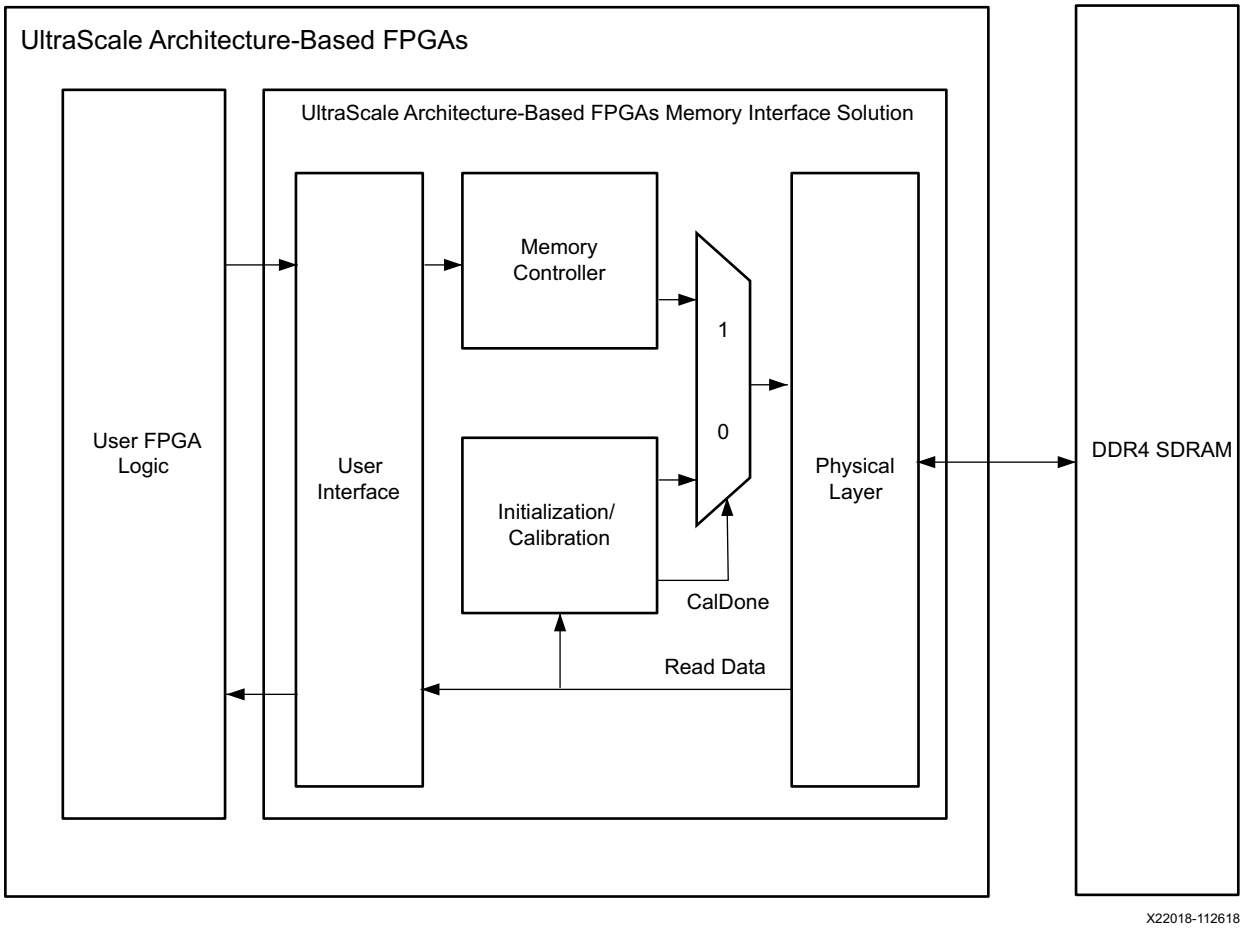

图 6-3：基于 UltraScale 架构的 FPGA 存储器接口解决方案的核架构

存储器控制器

存储器控制器 (MC) 被设计来用于从用户接口 (UI) 块获取读取、写入和读取、修改并写入的数据传输，并在尽可能少地使用 FPGA 资源的同时，以低时延有效地将这些数据传输发布到存储器，从而满足所有 DRAM 协议和时序要求。控制器以 DRAM 与系统时钟比 4:1 的比例运行，并且可以在每个系统时钟周期发出一个 Activate 命令、一个 CAS 命令和一个 Precharge 命令。

控制器支持开放的页面策略，并且可以通过具有高空间局部性的工作负载实现非常高的效率。控制器还支持关闭页面策略以及对数据传输进行重新排序的能力，以便使用更随机的地址模式有效地调度工作负载。控制器还允许对基于每个数据传输的 AutoPrecharge 的 UI 控制信号进行一定程度的低级功能控制，并可用来确定何时发出 DRAM 刷新命令的信号。

控制器命令路径的关键块包括：

- FSM 组 （这些组将数据传输排队，并检查 DRAM 的时序，然后确定什么时候要请求 Precharge、 Activate 和 CAS DRAM 命令）。
- “安全”逻辑和仲裁单元 （这些单元基于额外的 DRAM 时序检查，对各个 FSM 组之间的数据传输进行重新排序，同时还确保所有 DRAM 命令请求的进展）。
- 最终仲裁者 （最终决定要将哪些命令发布到 PHY 并将结果反馈给前一阶段）。

控制器命令路径的维护块包括：

- 生成 Refresh 和 ZQCS 命令的块
- 需要用来进行 VT 跟踪的命令

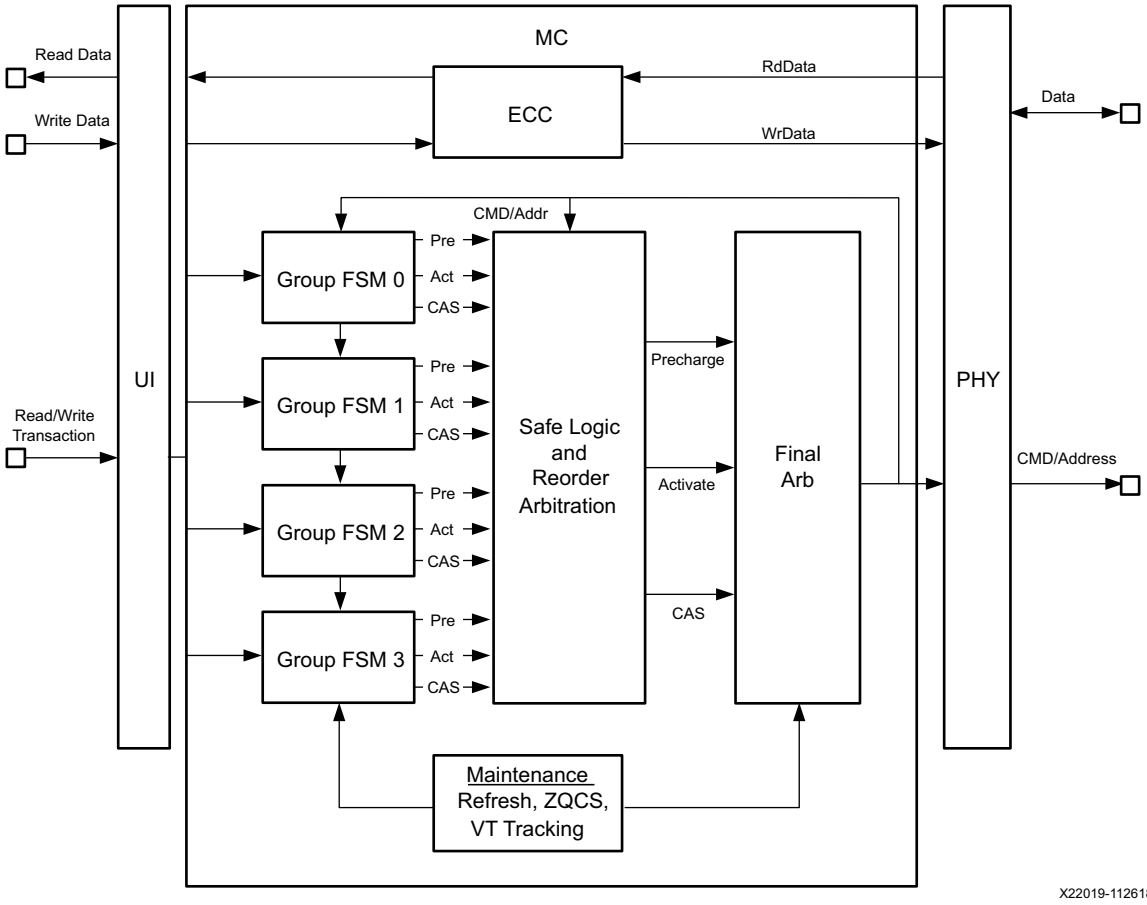

图 6-4：存储器控制器原理图

读写合并

在启用重新排序时，控制器会先读取、后写入。如果在 SDRAM 命令总线上发送读写 CAS 命令都是安全的，则控制器仅选择读取 CAS 命令进行仲裁。当发出读取 CAS 时，写入 CAS 命令会被参数 tRTW 指定的几个 SDRAM 时钟阻塞。在发出读取 CAS 之后，要让写入 CAS 变得安全所需的额外时间允许在命令总线上发出读取组，而不会被挂起的写入中断。

重新排序

映射到同一 mcGroup 的请求永远不会被重新排序。 mcGroup 实例之间的重新排序是由 ORDERING 参数来控制的。设置为“NORM”时，系统会启用重新排序，仲裁器将实现循环优先级计划，同时使用可安全发送到 SDRAM 的命令在 mcGroup 中按优先级顺序进行选择。

何时将命令发送到 SDRAM 比较安全可能会因目标 bank 或 bank 组及其页面状态而异。这通常有助于重新排序。

当 ORDERING 参数被设置为“STRICT”时，所有请求都会按照本机接口接受请求的顺序来发出 CAS 命令。STRICT 排序会覆盖所有其他控制器机制，例如有可能会合并读取请求，因此会降低某些工作负载中的数据带宽利用率。

当从 UI 接受新的数据传输时，它会被推入阶段 1 的数据传输 FIFO 中。检查阶段 1 头部 FIFO 数据传输的页面状态，并将其提供给阶段 1 的数据传输 FSM。FSM 会决定是否需要发出 Precharge 或 Activate 命令，以及什么时候根据 DRAM 定时器发出命令安比较全。

当页面打开且因阶段 2 FIFO 中的未决 RDA 或 WRA 尚未安排而关闭时，数据传输将从阶段 1 FIFO 传递到阶段 2 FIFO。此时，系统会弹出阶段 1 FIFO，阶段 1 FSM 会开始处理下一个数据传输。并行地，阶段 2 FSM 会在阶段 2 FIFO 的头部处理数据传输的 CAS 命令阶段。阶段 2 FSM 会在基于 tRCD 定时器安全时发出 CAS 命令请求。阶段 2 FSM 还会发出 RMW 数据传输的读写 CAS 请求。

读取 - 修改 - 写入流程

当在用户界面接受 `wr_bytes` 命令时，它最终会被分配给一组状态机，就像其他写入或读取数据传输一样。这组机器会将部分写入分为读取阶段和写入阶段。读取阶段将执行以下操作：

- 首先从存储器中读取数据。
- 检查读取数据中的错误。
- 纠正单个位错误。
- 将结果存储在存储器控制器中。

读取数据存储在控制器中后，写入阶段开始如下：

- 基于写入数据掩码位将写入数据与存储的读取数据合并。
- 由于为合并数据生成了新的检查位，读取阶段中的任何多位错误都会导致在写入阶段无法检测到错误。

写入阶段完成后，组计算机可用于处理新的数据传输。与简单的读取或写入相比， RMW 流占用组机器的时间更长，因此可能会影响性能。

PHY

PHY 被认为是外部 DDR3 或 DDR4 SDRAM 器件的低级物理接口，以及用来确保物理接口本身可靠运行的所有校准逻辑。 PHY 生成与存储器器件连接所需的信号定时和排序。

PHY 包含以下功能：

- 时钟/地址/控制生成逻辑
- 写入和读取数据路径
- 上电后初始化 SDRAM 的逻辑

此外， PHY 包含校准逻辑以执行读写数据路径的定时训练，并将系统的静态和动态延迟纳入考虑之中。

PHY 包含在完整的存储器接口解决方案核中，但也可以作为独立的 PHY 专用块来实现。如果您计划实现定制的存储器控制器，则可以选择仅 PHY 解决方案。预知与仅连接到 PHY 块相关的详情，请参阅 《基于 LogiCORE IP UltraScale 架构的 FPGAs Memory IP 产品指南》(PG150) [参照 10]。

核设计

时钟设置

在 2018.3 中，对参考时钟的硬性要求如下：

- 125 MHz (x8)
- 300 MHz (x16)

此外，与 2018.3 相关的速度等级支持：

- -1e speedgrade，支持的频率为 2400 MT/s
- -2e speedgrade，支持的频率可高达 2667 MT/s

存储器接口需要一个 MMCM，存储器接口使用的每个 I/O bank （I/O 管脚区）一个 TXPLL，以及两个 BUFG。这些时钟组件用于产生操作恰当的存储器接口所需的适当时钟频率和相移。

每个 bank 有两个 TXPLL。如果 bank 由两个存储器接口共享，则使用该 bank 中的两个 TXPLL。

注释：DDR4 SDRAM 生成适当的时钟结构，且不支持对 RTL 所做的任何修改。

DDR4 SDRAM 工具为所需接口生成适当的时钟结构。不得修改此结构。允许的时钟配置如下：

- 差分参考时钟源应连接到 GCIO
- GCIO 到 MMCM （在存储器接口的中心 bank 中）
- MMCM 到 BUFG （在存储器接口的中心 bank 中）驱动 FPGA 逻辑和所有 TXPLL
- MMCM 到 BUFG （在存储器接口的中心 bank 中）除以两个模式以驱动 1/2 速率的 FPGA 逻辑
- 接口时钟对必须与 SSI 技术器件的存储器接口的 SLR 相同

要求

GCIO

- 必须使用差分 I/O 标准
- 必须与存储器接口位于同一 I/O 列中
- 必须与 SSI 技术器件的存储器接口的 SLR 相同
- I/O 标准和终止方案取决于系统。如需了解更多信息，请参阅 《UltraScale 架构 SelectIO 资源用户指南》(UG571) [参照 11]。

MMCM

- MMCM 用于生成 FPGA 逻辑系统时钟 （存储器时钟的 1/4）
- 必须位于存储器接口靠近 bank 中心的位置
- 必须使用内部反馈
- 输入时钟频率除以输入分频器必须 ≥ 70 MHz (CLKINx/D ≥ 70 MHz)
- 必须使用整数乘法并输出除数值。适用于用例的等式如下：
  - 对于 x8 - 2400：300000000 (input clk) *5/6 = 250 MHz

- 对于 x8 – 2133：300000000 (input clk) *16/3*6 = 266 MHz
- 对于 x16 – 2400、 2667：125000000 (input clk) *8/4 = 250 MHz

输入时钟周期抖动

- 时钟输入周期抖动必须 ≤?50 ps 峰峰值

BUFG 和时钟根

- 一个 BUFG 用于生成 FPGA 逻辑的系统时钟，另一个 BUFG 用于将系统时钟除以 2。
- BUFG 和时钟根必须位于存储器接口最靠近 bank 中心的位置。
  - 如果有两个 bank 系统，所选字节数较多的存储系统会被选为中心 bank。如果在两个 bank 中选择的字节数相同，则选择位于顶部的 bank 作为中心 bank。
  - 如果有四个 bank 系统，可以选择任一中心 bank。DDR4 SDRAM 是指从最顶端的 bank 选择的第二个 bank 将作为中心 bank。
  - 两个 BUFG 必须在同一个 bank 中。

复位

可提供异步复位 (`sys_rst`) 输入。这是一个高电平有效复位，且 `sys_rst` 必须置位脉冲宽度为 5 ns 的最小脉冲。`sys_rst` 可以是内部或外部引脚。

如需了解更多有关复位的信息，请参阅第 7 章 中的“复位序列”。

注释：当 FPGA 活动从此复位输入的释放最小化到存储器接口被完全校准 （如 `init_calib_complete` 端口所示）时，即可实现最佳校准结果 （请参阅本文档中用户界面部分）

与 VCU DDR4 Controller IP 的连接

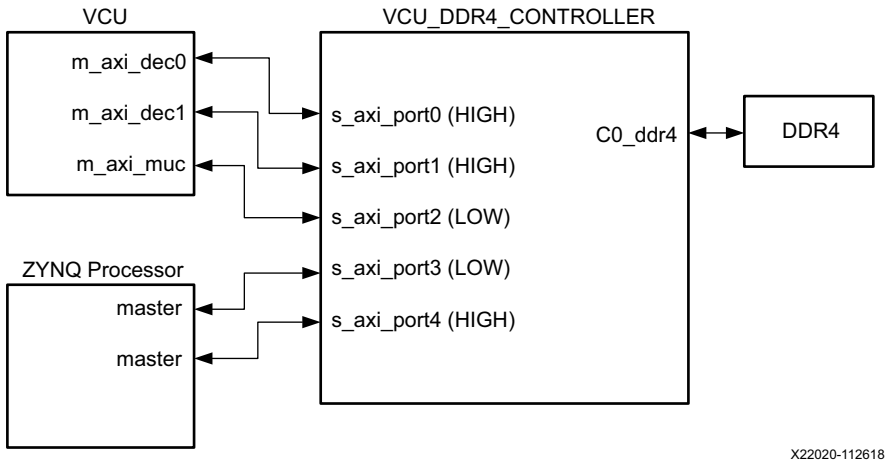

图 6-5：连接到 VCU IP

表 6-8：AXI 端口图与优先级

AXI 端口	映射	端口优先级 （在 v1.0 版已修复）
S_AXI_PORT_0	M_AXI_DEC_0 (VCU)	HIGH
S_AXI_PORT_1	M_AXI_DEC_1 (VCU)	HIGH

表 6-8：AXI 端口图与优先级 （续）

AXI 端口	映射	端口优先级 （在 v1.0 版已修复）
S_AXI_PORT_2	M_AXI_MCU (VCU)	LOW
S_AXI_PORT_3	Zynq processor	LOW
S_AXI_PORT_4	Zynq processor	HIGH

端口优先级概念

为了实现最佳的端口性能和内存带宽利用率，我们提供了以下优先级，并在 drop 中进行了硬编码。

通过解码器端口和 Zynq MP 帧获取的带宽消耗最高的端口具有最高优先级。在仲裁中，带宽消耗较低的另外两个端口会被指定为最低优先级。测试此配置是为了提供最佳性能。

有关 DDR4 的 PCB 指南、引脚和 bank 规则、协议说明、性能、 DIMM 配置、设置时序选项以及参考输入时钟速度的 M 和 D 支持的信息，请参阅《基于 LogiCORE IP UltraScale Architecture 架构的 FPGA 存储器 IP 产品指南》(PG150) [参照 10]。

设计流程步骤

本章介绍如何自定义和生成核、约束核以及应按照哪些特定于此 IP 核的仿真、综合和实现步骤进行操作。

自定义 VCU DDR4 控制器

您可以根据以下选项自定义 GUI：

- DRAM 总线宽度选择 - 可以是 x8 （zcu104 评估板）或 x16 （zcu106 评估板）
- DRAM 速度箱选项 - 2133 和 2400 用于 x8 选项。 2400 和 2667 用于 x16 选项

2018.3 向导中的其他参数是固定的。在 2019.1 版中，赛灵思将为 logiCORE 添加更多的自定义选项。

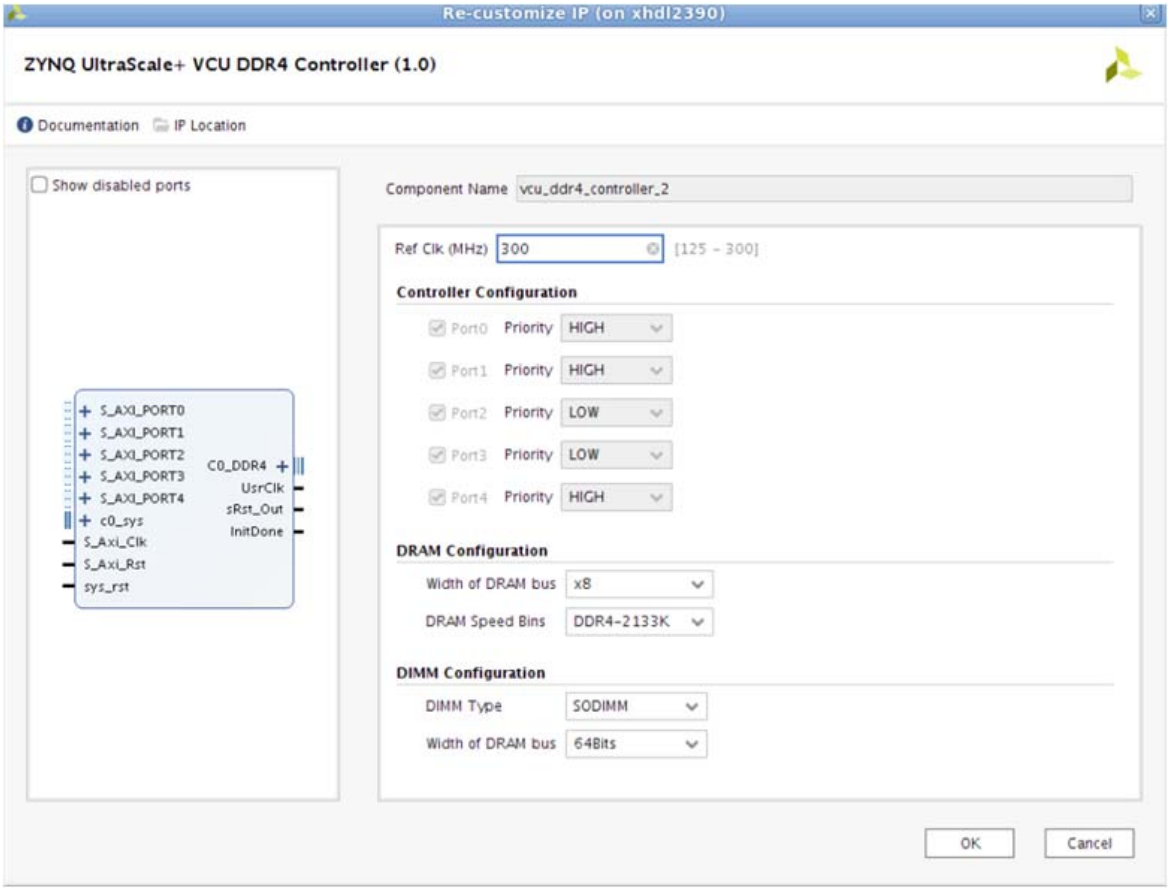
Re-customize IP (on xhdl2390)
ZYNQ UltraScale+ VCU DDR4 Controller (1.0)
Documentation
IP Location
Show disabled ports
S_AXI_PORT0
S_AXI_PORT1
S_AXI_PORT2
S_AXI_PORT3
S_AXI_PORT4
c0_sys
S_Axi_Clk
S_Axi_Rst
sys_rst
C0_DDR4
UsrClk
sRst_Out
InitDone
Component Name
vcu_ddr4_controller_2
Ref Clk (MHz)
300
[125 - 300]
Controller Configuration
Port0 Priority HIGH
Port1 Priority HIGH
Port2 Priority LOW
Port3 Priority LOW
Port4 Priority HIGH
DRAM Configuration
Width of DRAM bus
x8
DRAM Speed Bins
DDR4-2133K
DIMM Configuration
DIMM Type
SODIMM
Width of DRAM bus
64Bits
OK
Cancel

图 6-6：X8 GUI 设置

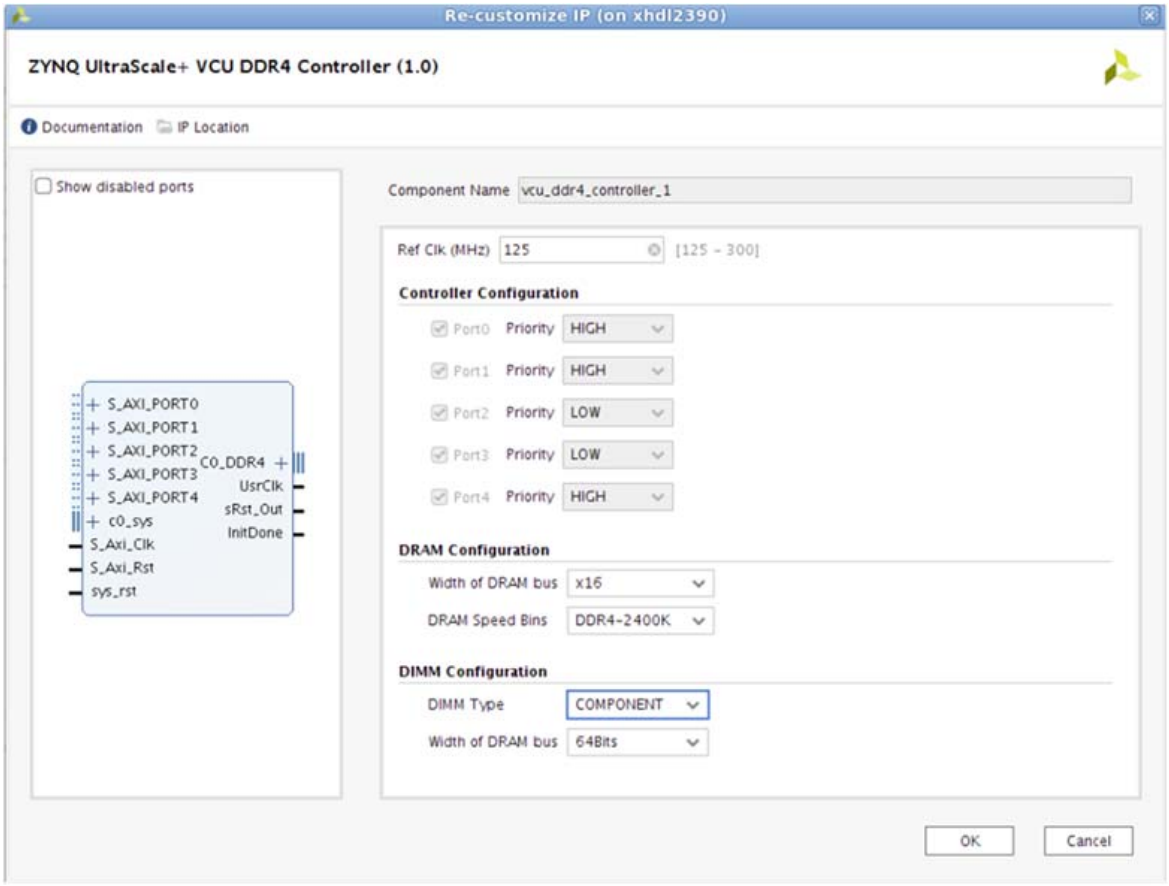

图 6-7：X16 GUI 设置

端口优先级在 2018.3 版本中修复。在此 IP 中启用了五个端口。高优先级端口需要连接到解码器/显示器接口。低优先级端口需要连接到 PS、微控制器单元 (MCU) 接口。

表 6-9：支持的设置

存储器器件	支持的数据率 MT/s	支持的 DIMM 类型	参考时钟频率	DRAM 宽度	支持的排名	时钟周期 CL – tRCD - tRP
X16 (MT40A256M16GE-07)	DDR4 - 2400，2667	COMPONENT	125 MHz	64 Bits	1	2400 MT/s –- 17-17-17 2667 MT/s – 19-19-19
X8 (KVR21SE15S8/4)	DDR4 - 2400，2133	COMPONENT SODIMM	300 MHz	64 Bits	1	2400 MT/s – 17-17-17 2133 MT/s – 15-15-15

连接 VCU LogiCORE IP

图 6-4 显示 IP 是如何连接到 VCU LogiCORE IP 的。

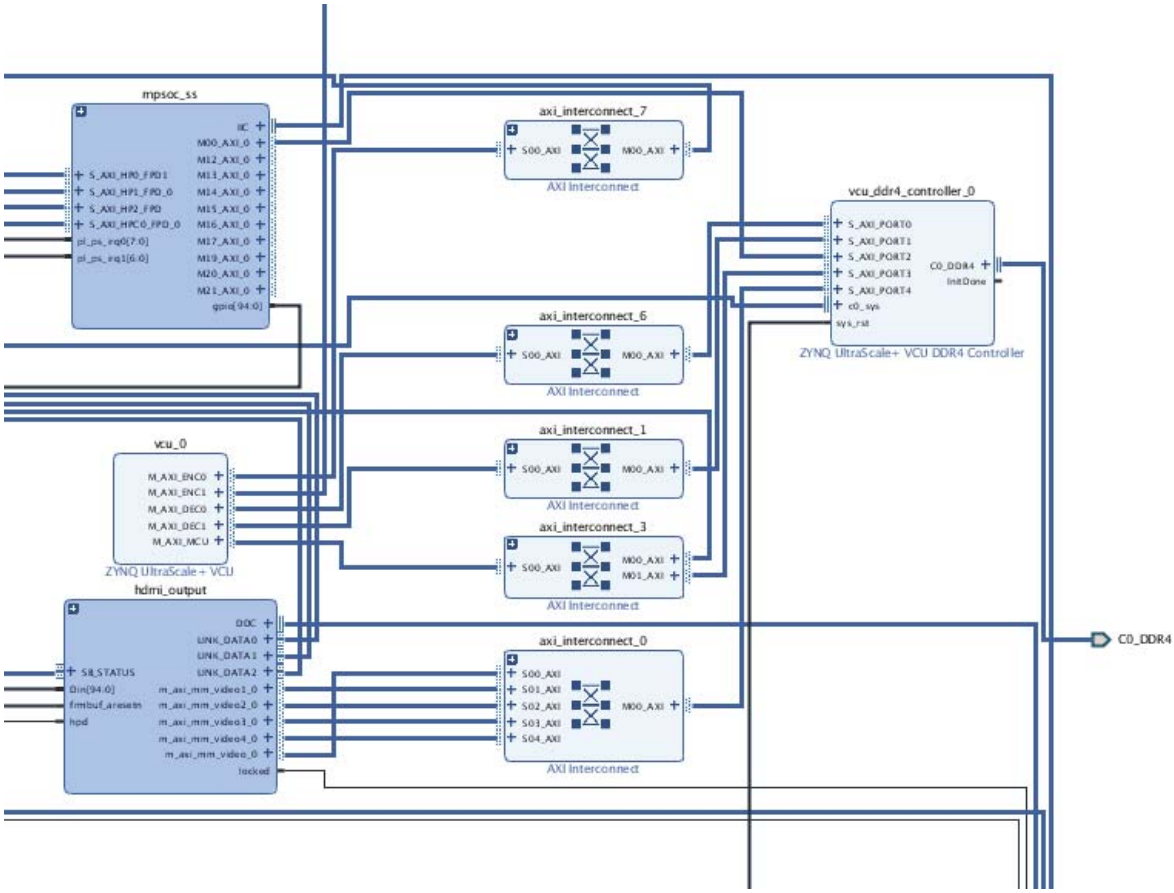

图 6-8：IP 连接

表 6-10 显示建议的端口连接。

表 6-10：端口连接

VCU DDR4 控制器端口	主端口	优先级
S_AXI_PORT0	M_AXI_DEC0 （通过 AXI 互连）	High
S_AXI_PORT1	M_AXI_DEC1 （通过 AXI 互连）	High
S_AXI_PORT2	M_AXI_MCU （通过 AXI 互连）	Low
S_AXI_PORT3	M_AXI_HPM0/1 （通过 AXI 互连）	Low
S_AXI_PORT4	M00_AXI （视频混合器或帧缓存读取主站）	High

I/O 管脚分配

DDR4 SDRAM I/O 管脚分配通过使用 Vivado I/O 管脚分配器的完整设计管脚分配来完成。可以通过几个 Vivado I/O 管脚分配器功能选择 DDR4 SDRAM I/O 引脚 （包括使用 I/O 端口视图、封装视图或 bank/字节规划器进行分配）。管脚分配还可以通过导入 XDC 或修改现有的 XDC 文件来进行。

这些选项适用于所有的 DDR4 SDRAM 设计，多个 DDR4 SDRAM IP 实例可在一个设置中完成。如需有关可用 Memory IP 管脚分配选项的更多信息，请参阅 《Vivado Design Suite 用户指南：I/O 管脚分配和时钟规划》(UG899) [参照 3]。

核的约束

本节包含有关如何在 Vivado Design Suite 中约束核的信息。

所需约束

本节包含有关如何在 Vivado Design Suite 中约束核的信息。如需了解更多有关 I/O 标准和其它约束的信息，请参阅《Vivado Design Suite 用户指南：I/O 管脚分配和时钟规划》(UG899) [参照 3]。具体位置由 Vivado IDE 根据为设计选择的 bank 和字节通道来选择。

I/O 标准按 Vivado IDE 中的存储器类型选择和选项以及引脚类型来选择。此处显示的是 dq[0] 示例。

```
set_property PACKAGE_PIN AF20 [get_ports "c0_ddr4_dq[0]"]
set_property IOSTANDARD POD12_DCI [get_ports "c0_ddr4_dq[0]"]
```

内部 VREF 通常用于 DDR4。就 DDR3 而言，内部 VREF 为可选项。此处显示的是 DDR4 示例。

```
set_property INTERNAL_VREF 0.600 [get_iobanks 45]
```

注释：内部 VREF 由工具自动生成 （无需指定）。此约束中列出的 VREF 值不能与 PODL12 I/O 一起使用。初始值设置为 0.84V。为了获得最大的接口性能，校准逻辑会根据需要调整此电压。

系统时钟必须正确设置时钟周期：

```
create_clock -name c0_sys_clk -period.938 [get_ports c0_sys_clk_p]
```

对于 HR bank，需更新分配给 HR bank 引脚的所有端口的 output_impedance （使用 reset_property 命令来完成）。如需了解更多信息，请参阅 AR:63852。

器件、封装和速度等级选择

本节不适用于此 IP 核。

时钟频率

本节不适用于此 IP 核。

时钟管理

如需了解更多有关时钟设置的信息，请参阅“时钟设置”。

时钟布局

本节不适用于此 IP 核。

bank 分配

本节不适用于此 IP 核。

收发器布局

本节不适用于此 IP 核。

I/O 标准与布局

DDR3/DDR4 SDRAM 工具根据 Vivado IDE 中为接口类型和选项所做的选择生成适当的 I/O 标准和布局。

仿真

有关 Vivado 仿真组件的全面信息以及与使用支持的第三方工具相关的信息，请参阅 《Vivado Design Suite 用户指南：逻辑仿真》(UG900) [参照 5]。

综合与实现

如需了解有关综合与实现的详情，请参阅 《Vivado Design Suite 用户指南：采用 IP 进行设计》(UG896) [参照 2]。

XILINX

第 7 章

时钟设置和复位

引言

Video Codec Unit (VCU) 核支持一种时钟拓扑：

- 内部 PLL：内部 VCU 锁相环 (PLL) 根据来自可编程逻辑 (PL) 的输入参考时钟驱动高频核 (667 MHz) 和 MCU (444 MHz) 时钟。内部 PLL 会为编码器和解码器块生成时钟。

注释：所有 AXI 时钟均由外部 PL 源提供时钟。这些时钟与核编码器和解码器块的时钟异步。编码器和解码器块会处理 AXI 端口中的异步时钟。

VCU 核在以下条件下会复位：

- 最初，当 PL 处于上电/配置模式时， VCU 核会保持复位状态。
- 完全配置 PL 后，可以使用基于 PL 的复位信号来复位 VCU 以进行初始化。处理器系统 (PS) 中的平台管理单元 (PMU) 可以驱动该复位信号以控制 VCU 的复位状态。
- 在部分重配置 (PR) 期间，如果 VCU 块是动态可重配置模块的一部分，则它将保持复位状态。

功能描述

时钟设置

解码器 (VDEC) 和编码器 (VENC) 块作为独立的单元独立工作，彼此之间没有任何依赖性。表 7-1 对 VCU 核中的时钟域进行描述。

表 7-1：VCU 时钟域

域	最大频率 (MHz)	描述
核时钟	667	处理核，大部分逻辑和存储器
MCU 时钟	444	内部微控制器
AXI 主端口时钟	333	m_axi_enc_aclk, m_axi_dec_aclk, enc_buffer_clk, pl_vcu_axi_mcu_clk 用于存储器访问的 AXI 主端口， 128 位，通常连接到 PS AFI-FM (HP) 端口或 PL 中的软存储器控制器。
AXI4-Lite 从端口时钟	167	s_axi_lite_aclk，用于寄存器编程的 AXI4-Lite 从端口 （32 位）

注释：所有 AXI 时钟均由外部 PL 源提供时钟。这些时钟与核编码器、解码器和 MCU 的时钟异步。VENC 和 VDEC 核是被设计来处理 AXI 端口中的异步时钟的。 `m_axi_mcu_aclk` 与 VCU 中使用的所有时钟异步。

如需了解更多信息，请参阅第 3 章中的“编码器缓存”。

图 7-1 显示 VCU 块内的时钟生成选项。请注意，以下块适用于单个时钟域：

- `pll_ref_clk` 通常由可编程时钟集成电路从外部提供给器件。
- 视频编码器和解码器块在 VCU PLL 生成的 `VENC_core_clk` 域下工作。
- 用于编码器和解码器的 MCU 在 VCU PLL 生成的 `VENC_MCU_clk` 域下工作。
- `m_axi_enc_aclk` 是来自 PL 的 AXI 时钟输入，用于编码器的 128 位 AXI 主接口。
- `m_axi_dec_aclk` 是来自 PL 的 AXI 时钟输入，用于解码器的 128 位 AXI 主接口。
- `s_axi_lite_aclk` 是来自 PL 的 AXI4-Lite 时钟。
- `m_axi_mcu_aclk` 是来自 PL 的 MCU AXI 主时钟。

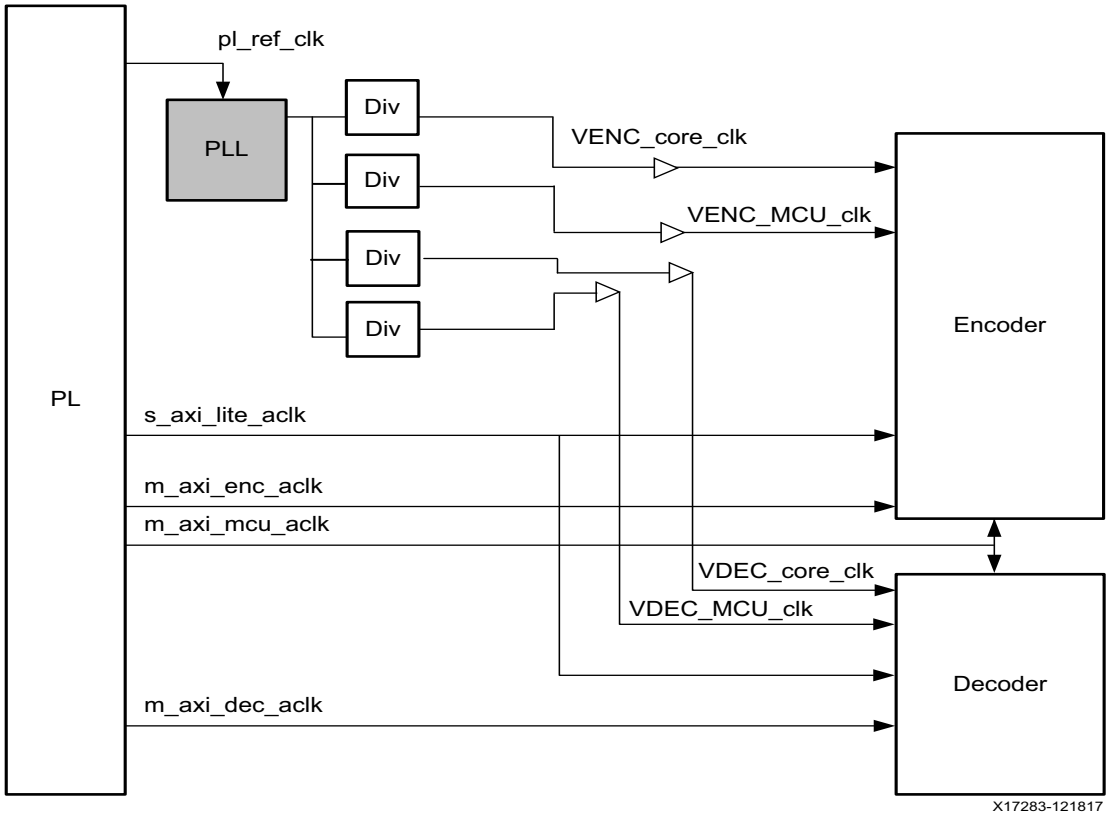

图 7-1：时钟生成选项

在提供来自 PL 时钟时，必须满足以下时钟频率要求：

- 编码器和解码器接口的 AXI 时钟被限制为 333 MHz。
- 需要满足以下比率要求：
  - $s\_axi\_lite\_aclk \leq 2 \times m\_axi\_enc\_aclk$
  - $s\_axi\_lite\_aclk \leq 2 \times m\_axi\_dec\_aclk$

如需了解有关 MCU 的信息，请参阅第 5 章"微控制器单元简介"。

`VENC_core_clk` 是基于 VCU PLL 生成的。

`VENC_MCU_clk` 是基于 VCU PLL 生成的。

`VDEC_core_clk` 是基于 VCU PLL 生成的。

PLL 简介

VCU 核带有用于生成编码器/解码器块时钟的 PLL。通常， PLL 带有外部源，例如 Si570 XO 可编程时钟发生器（该发生器通过 IBUFDS 直接连接到 PLL 参考时钟）。或者，IBUFDS 可以驱动 MMCM 以使其他模块能够共享外部时钟源，同时满足低于 100 ps 的抖动规范。由于抖动要求，建议不要将 PS PLL 用作时钟源。 PLL 参考时钟的范围是 27 到 60 MHz。PLL 生成高频时钟，可以将其分频以生成各种输出时钟频率。分频时钟可以提供给编码器块、解码器块和 MCU （用于视频编码器和解码器的独立 MCU）。

主时钟的生成

PLL 具有压控振荡器 (VCO) 块，该块根据输入参考时钟生成输出时钟。来自 VCO 的输出时钟基于倍频器的值生成。VCO 的输出时钟由输出分频器分频以生成最终时钟。

VCO 频率和 MF 值

可以通过使用以下关系来确定 VCO 的操作频率：

$$f_{vco} = f_{refclk} \times M$$

和

$$f_{clkout} = f_{vco}/O$$

在这里， M 对应于整数反馈分频值，而 O 对应于输出分频的值。请注意， PLL 不支持小数分频器值。

重要提示：根据支持的 VCO 频率范围 (f_{vco}) 选择 PLL 反馈乘数值。

如需了解更多有关 $_{vco}$ 运行范围的信息，请参考 《Zynq UltraScale+ MPSoC 数据手册：直流和交流开关特性》 [参照 16]。

重要提示：根据所需的核时钟或 MCU 时钟频率选择输出分频器 (O)。

重要提示：与其他 IP 共享 VCU 时钟输入可能会导致时钟抖动，从而降低 VCU 性能或图像质量。

复位序列

PL 上电期间 VCU 的状态和 VCU 的初始化顺序如下：

- PL 尚未配置。在这种情况下， VCU 保持复位状态。
- 当电源的电压达到预定的幅度时， VCU 核保持复位状态。在电压达到预定的幅度期间，存在于 VCU 核到 PL 接口的电压检测器使核保持在复位状态。
- PL 已完全配置。 PL 配置有 PS 或 PL 中的 CPU 与 VCU 核的 AXI 从端口之间的 AXI 连接。可以释放 VCU 核复位，以便核处于已知状态。

VCU 核复位取消置位后，使用该软件对 VCU PLL 进行编程，以生成 VCU 核和 MCU 块的时钟。在对 VCU PLL 进行编程时，按照 “PLL 整数分频器编程” 中描述的步骤对 PLL 配置参数进行编程。 PLL 锁定状态由 VCU_SLCR 指示。如需了解更多信息，请参阅 《Zynq UltraScale+ MPSoC 寄存器参考》(UG1087) [参照 14]。

注释：释放复位时， VCU 核时钟可用。应在释放原始复位之前配置 PL，原始复位可由 PMU 从 VCU 核外部来控制。

在对 PL 进行了配置且核处于复位释放状态之后，软件可通过对 VCU 核寄存器进行编程来完成额外的初始化。

PLL 整数分频器编程

要对 VCU PLL 执行操作，配置 VCU_SLCR VCU_PLL_CFG 寄存器（需使用表 7-2 中的值）。必须对以下字段进行编程：

- VCU_SLCR.VCU_PLL_CFG[LOCK_DLY]
- VCU_SLCR.VCU_PLL_CFG[LOCK_CNT]
- VCU_SLCR.VCU_PLL_CFG[LFHF]
- VCU_SLCR.VCU_PLL_CFG[CP]
- VCU_SLCR.VCU_PLL_CFG[RES]

FBDIV 值 （或 PLL 反馈乘数值 M）取决于输出 VCO 频率 (f_{VCO})。您必须根据表 7-2 中计算出的 FBDIV 值对 VCU_SRCR.VCU_PLL_CFG 进行编程。

表 7-2：用于整数反馈分频器值的 PLL 编程

FBDIV	CP	RES	LFHF	LOCK_DLY	LOCK_CNT
25	3	10	3	63	1000
26	3	10	3	63	1000
27	4	6	3	63	1000
28	4	6	3	63	1000
29	4	6	3	63	1000
30	4	6	3	63	1000
31	6	1	3	63	1000
32	6	1	3	63	1000
33	4	10	3	63	1000
34	5	6	3	63	1000
35	5	6	3	63	1000
36	5	6	3	63	1000
37	5	6	3	63	1000
38	5	6	3	63	975
39	3	12	3	63	950
40	3	12	3	63	925
41	3	12	3	63	900
42	3	12	3	63	875
43	3	12	3	63	850
44	3	12	3	63	850
45	3	12	3	63	825
46	3	12	3	63	800
47	3	12	3	63	775
48	3	12	3	63	775
49	3	12	3	63	750
50	3	12	3	63	750
51	3	2	3	63	725

表 7-2：用于整数反馈分频器值的 PLL 编程 （续）

FBDIV	CP	RES	LFHF	LOCK_DLY	LOCK_CNT
52	3	2	3	63	700
53	3	2	3	63	700
54	3	2	3	63	675
55	3	2	3	63	675
56	3	2	3	63	650
57	3	2	3	63	650
58	3	2	3	63	625
59	3	2	3	63	625
60	3	2	3	63	625
61	3	2	3	63	600
62	3	2	3	63	600
63	3	2	3	63	600
64	3	2	3	63	600
65	3	2	3	63	600
66	3	2	3	63	600
67	3	2	3	63	600
68	3	2	3	63	600
69	3	2	3	63	600
70	3	2	3	63	600
71	3	2	3	63	600
72	3	2	3	63	600
73	3	2	3	63	600
74	3	2	3	63	600
75	3	2	3	63	600
76	3	2	3	63	600
77	3	2	3	63	600
78	3	2	3	63	600
79	3	2	3	63	600
80	3	2	3	63	600
81	3	2	3	63	600
82	3	2	3	63	600
83	4	2	3	63	600
84	4	2	3	63	600
85	4	2	3	63	600
86	4	2	3	63	600
87	4	2	3	63	600

表 7-2：用于整数反馈分频器值的 PLL 编程 （续）

FBDIV	CP	RES	LFHF	LOCK_DLY	LOCK_CNT
88	4	2	3	63	600
89	4	2	3	63	600
90	4	2	3	63	600
91	4	2	3	63	600
92	4	2	3	63	600
93	4	2	3	63	600
94	4	2	3	63	600
95	4	2	3	63	600
96	4	2	3	63	600
97	4	2	3	63	600
98	4	2	3	63	600
99	4	2	3	63	600
100	4	2	3	63	600
101	4	2	3	63	600
102	4	2	3	63	600
103	5	2	3	63	600
104	5	2	3	63	600
105	5	2	3	63	600
106	5	2	3	63	600
107	3	4	3	63	600
108	3	4	3	63	600
109	3	4	3	63	600
110	3	4	3	63	600
111	3	4	3	63	600
112	3	4	3	63	600
113	3	4	3	63	600
114	3	4	3	63	600
115	3	4	3	63	600
116	3	4	3	63	600
117	3	4	3	63	600
118	3	4	3	63	600
119	3	4	3	63	600
120	3	4	3	63	600
121	3	4	3	63	600
122	3	4	3	63	600
123	3	4	3	63	600

表 7-2：用于整数反馈分频器值的 PLL 编程 （续）

FBDIV	CP	RES	LFHF	LOCK_DLY	LOCK_CNT
124	3	4	3	63	600
125	3	4	3	63	600

复位

在以下条件下， VCU 硬块可以保持在复位状态：

- 当外部复位输入 vcu_resetn 信号有效时。
- 在 PL 配置期间。
- 当 VCU 到 PL 隔离未被移除时。

VCU 复位信号必须至少在 VCU PLL 参考时钟的两个时钟周期 （VCU 的最慢时钟输入）中置位。在复位信号被取消置位后，可以访问 VCU 寄存器。

注释：如果软件在帧中间对 VCU 块进行复位，使用该软件清除为 VCU 分配的物理内存。

注释：在运行期间通过 VCU 控制软件对 VCU 配置进行的更改之间无需置位复位。

软件可以对 VCU_SLCR 中的偏移量 0x41074 处的 VCU_GASKET_INIT 寄存器进行编程，以向 VCU 块发出复位脉冲。使用以下流程对 VCU 进行复位：

1. 确保 VCU AXI 总线/AXI4-Lite 总线中没有待处理的 AXI 数据传输。
2. 通过 EMIO GPIO 引脚将 vcu_resetn 置位到 VCU LogiCORE IP。
3. 取消置位 vcu_resetn。
4. 将 0 写入 VCU 垫片隔离寄存器 VCU_GASKET_INIT[1] 以将复位置为 VCU。
5. 将 0 写入 VCU 垫片隔离寄存器 VCU_GASKET_INIT[0] 以启用 VCU 垫片隔离。
6. 关闭 VCU 电源。

如需了解更多信息，请参阅表 2-2。

VCU 核中的 PLL 可以通过 VCU_SLCR 寄存器复位，而该寄存器可通过 AXI4-Lite 接口进行访问。

每个编码器和解码器块都带有基于寄存器的软复位。

时钟设置和寄存器复位

表 7-3 列出了时钟和复位寄存器。

表 7-3：时钟设置与寄存器复位

寄存器	地址	宽度	类型	复位值	描述
CRL_WPROT	0xA0040020	1	rw	0x00000000	CRL SLCR 写入保护寄存器
VCU_PLL_CTRL	0xA0040024	32	mixed	0x0000510F	PLL 基本控制
VCU_PLL_CFG	0xA0040028	32	mixed	0x00000000	帮助数据
PLL_STATUS	0xA0040060	32	mixed	0x00000008	PLL 的状态

XILINX

第 8 章

VCU 流水线的时延

Video Codec Unit (VCU) 设计来用于支持分辨率高达 3,840×2,160 像素、每秒 60 帧 (4K UHD、 60 Hz)、带有一组 B 帧图像 (GOP) 的视频流 （最难的情况）。在帧边界之间定义时延，并允许所有 GOP 类型。某些 GOP 类型需要显示要重新排序的缓存。

玻璃时延

如图 8-1 所示，玻璃时延 (L) 是以下各项的总和：

- 相机时延
- 片上时延列 (L1)
  - 源帧缓存 DMA 时延
  - 编码器时延
  - 传输比特流缓存时延
- 网络或存储时延
- 片上时延列 (L2)
  - 编码图像缓存 (CPB)/抖动缓存时延
  - 解码器时延
  - 解码图像缓存 (DPB) 时延
  - 显示帧缓存 DMA 时延
- 显示监视器时延

当启用 B 帧时，由于使用了重排序缓存器，每个 B 帧会产生一帧的时延。为了最优化 CPB 时延，解码器和显示器 DMA 之间需要 PL 中的握手机制。假设捕获侧和显示侧都在公共 VSYNC 时序上工作。

VSYNC 时序可以是异步的，并且需要一个时钟恢复机制来使源时序与同步同步。

凭借独立的 VSYNC 时序和无时钟恢复机制，它需要一个额外的帧时延才能与显示器件同步。

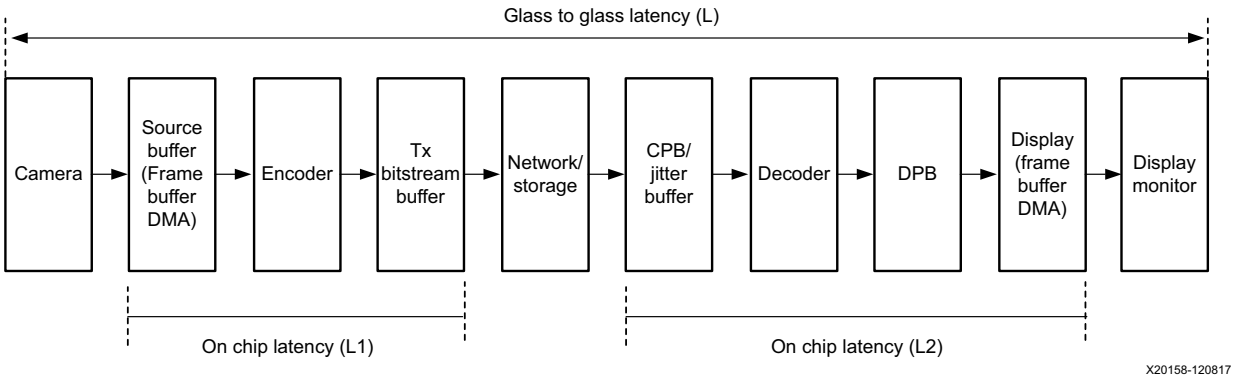

图 8-1：玻璃时延

VCU 时延模式

VCU 支持三种时延模式：正常时延、降低的时延和低时延。取决于帧结构、编码标准、级别和配置信息以及目标比特率，流水线上的时延会有所不同。

- 正常时延：VCU 编码器和解码器在帧级工作。支持所有可能的帧类型 （I、 P 和 B），对 GOP 结构没有限制。端到端的时延取决于配置信息/级别、 Gop-Structure 和用于处理的内部缓存的数量。
- 减低的时延：VCU 编码器仍可在帧级工作。硬件速率控制用于降低比特率的变化。仅支持 I 和 P 帧类型，无输出重新排序。 Gop 结构使用 low-delay-p，启用 GDR 进行解码器刷新。 VCU 器仍然在帧级工作。
- 低时延：帧被分成多个 slice， VCU 编码器输出和解码器输入以 slice 模式进行处理。 VCU 编码器输入和解码器输出仍然在帧模式下工作。VCU 编码器在 slice 的每一端生成“slice 已完成”中断，并为 slice 输出流缓存，而且该缓存随即即可用于下一个元素处理。因此，对于多个 slice，可以将 VCU 处理时延从多个 slice 一帧一帧地减少。

表 8-1 显示 VCU 流水线阶段的时延。

表 8-1：VCU 时延

用例	采集	编码	解码	显示
正常时延	16.6 ms	16.6 ms	83.33 ms	16.6 ms
时间延减少	16.6 ms	16.6 ms	50 ms	16.6 ms
低时延	16.6 ms	3 ms	22.6 ms	16.6 ms

VCU 编码器时延

图 8-2 显示的是的是编码器时延。

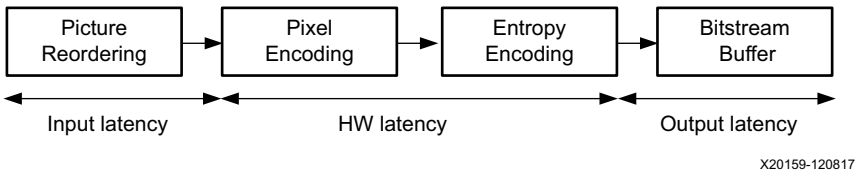

图 8-2：编码器时延

编码器的总时延为稳态时延，等于输入时延、硬件时延和输出时延之和。比特流缓存时延取决于应用。图像重新排序的时延时间等于每个 B 帧一帧的持续时间。

VCU 解码器时延

图 8-3 显示的是的是解码器时延。

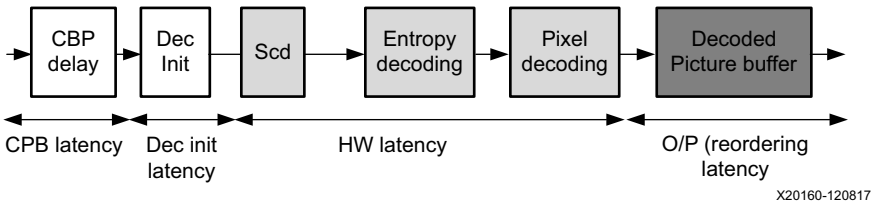

图 8-3：解码器时延

解码器器的总时延为稳态时延，等于输入时延、硬件时延和输出时延之和。硬件时延是连续消除解码 (SCD) 的时延、熵解码时延和像素解码时延之和。初始化时延是 CPB 时延和解码器初始化 (Dec Init) 时延之和。

第 9 章

AXI 性能监控器

简介

AXI 性能监控器 (APM) 在嵌入式 Video Codec Unit (VCU) 内部实现。VCU AXI 性能监视器 (VAPM) 允许以非侵入方式访问系统级行为，而无需为设计增加额外的软 IP。

APM 块能够在编码器/解码器块的 AXI 主总线上的测量窗口内测量读取或写入的字节数和基于地址的数据传输。 APM 还可以在测量窗口内测量基于主 ID 的读写时延。APM 支持累积时延值和考虑要进行时延测量的待处理传输数。当状态寄存器准备好被读取时， APM 能够中断主处理器。

功能描述

图 9-1 显示的是 VAPM。

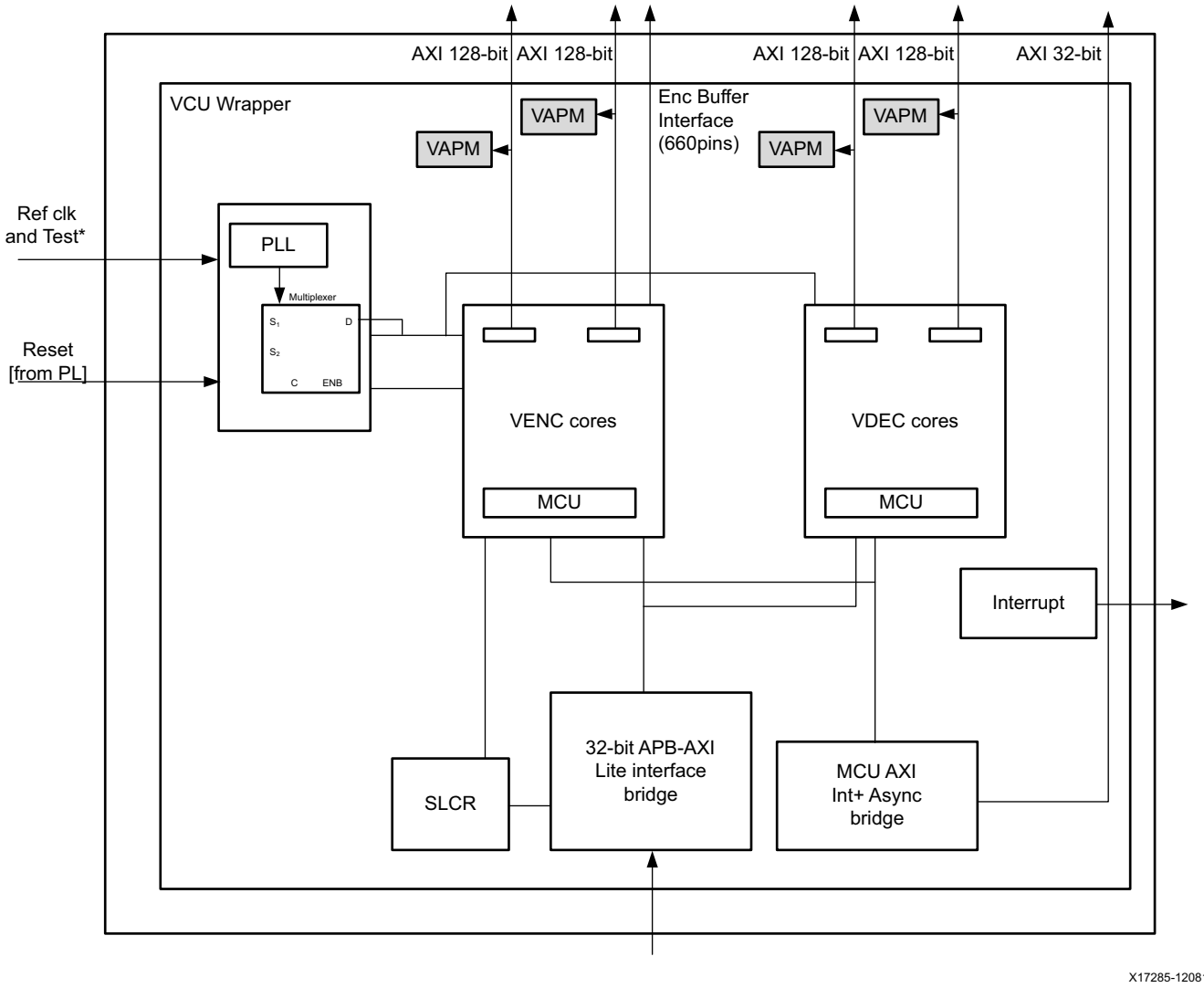

图 9-1：VCU AXI Performance Monitor （VCU AXI 性能监控、 VAPM）

以下部分描述了 VAPM 的不同操作模式。

操作时序窗口生成

VAPM 基于两个用户选择的操作模式生成测量参数。

启动/停止模式

在此模式下，测量窗口由 `VCU_SLCR.APMn_TRG[start_stop] (n=0,1,2,3)` 位中的开始/停止位决定。当该位的寄存器位从 0 设置为 1 时触发测量，当该位从 1 复位为 0 时，测量停止。

固定持续时间的时间窗口

在此模式下， 32 位计数器用于生成固定长度测量窗口。当计数器达到最大值时，它将复位为 VCU_SLCR.APMn_TIMER (n=0,1,2,3) 寄存器中指定的值。继续测量，直到 32 位计数器达到 APMn_TIMER 寄存器中设置的值，并生成捕获脉冲以将测量值存储在 VCU_SLCR 结果寄存器中。

VAPM 能够进行以下测量：

- AXI 读写数据传输测量
- AXI 读写字节计数测量
- AXI 数据传输时延测量

AXI 读写数据传输测量

两个 32 位寄存器计算在给定时序窗口中读写 128 位的 AXI 总线传输周期数。当基于启动/停止模式或固定持续时间时序窗口模式生成捕获脉冲时，测量值会被传输到 VCU_SLCR 结果寄存器。要计算传输的字节数， VCU_SLCR 必须乘以 16。

AXI 读写字节数测量

可通过实现两个 32 位寄存器来计算在给定时序窗口中传输的读写字节数。寄存器内容必须乘以 16 才能知道在 AXI 128 位主接口上传输的实际字节数。当基于启动/停止模式或固定持续时间时序窗口模式生成捕获脉冲时，测量值会被传输到 VCU_SLCR 结果寄存器。

AXI 数据传输时延测量

可以基于 AXI 主 ID 来测量读取/写入时延。读取时延会被定义为 AXI 读取地址，确认到上一个读取数据周期。写入时延会被定义为 AXI 写入地址，将用于写入主控制器和从设备之间的响应握手。可通过实现一个 13 位计数器来测量读取写入总线的时延。计时器用于为事件添加时间戳。两个事件之间的时间戳差异可用来计算时延。

时延可以基于数据传输 ID 来计算。可以选择单个 ID 或所有 ID 来进行时延计算。如需了解更多信息，请参阅 《Zynq UltraScale+ MPSoC 寄存器参考》(UG1087) [参照 14]。

表 9-1：寄存器

寄存器	地址	宽度	类型	复位值	描述
APM_InputT_GBL_CNTL	0×A0040090	32	mixed	0x00000001	该寄存器控制 APM 时序窗口完成中断。
APM0_CFG	0xA0040100	32	mixed	0x00000002	APM0_CFG
APM0_TIMER	0xA0040104	32	rw	0x00000000	APM0_TIMER
APM0_TRG	0xA0040108	32	mixed	0x00000000	APM0_TRG
APM0_RESULT0	0xA004010C	32	ro	0x00000000	APM0_RESULT0
APM0_RESULT1	0xA0040110	32	ro	0x00000000	APM0_RESULT1
APM0_RESULT2	0xA0040114	32	ro	0x00000000	APM0_RESULT2
APM0_RESULT3	0xA0040118	32	ro	0x00000000	APM0_RESULT3
APM0_RESULT4	0xA004011C	32	mixed	0x00000000	APM0_RESULT4
APM0_RESULT5	0xA0040120	32	mixed	0x00000000	APM0_RESULT5
APM0_RESULT6	0xA0040124	32	mixed	0x00000000	APM0_RESULT6
APM0_RESULT7	0xA0040128	32	mixed	0x00000000	APM0_RESULT7
APM0_RESULT8	0xA004012C	32	mixed	0x00000000	APM0_RESULT8

表 9-1：寄存器 （续）

寄存器	地址	宽度	类型	复位值	描述
APM0_RESULT9	0xA0040130	32	mixed	0x00000000	APM0_RESULT9
APM0_RESULT10	0xA0040134	32	mixed	0x00000000	APM0_RESULT10
APM0_RESULT11	0xA0040138	32	mixed	0x00000000	APM0_RESULT11
APM0_RESULT12	0xA004013C	32	mixed	0x00000000	APM0_RESULT12
APM0_RESULT13	0xA0040140	32	mixed	0x00000000	APM0_RESULT13
APM0_RESULT14	0xA0040144	32	mixed	0x00000000	APM0_RESULT14
APM0_RESULT15	0xA0040148	32	mixed	0x00000000	APM0_RESULT15
APM0_RESULT16	0xA004014C	32	mixed	0x00000000	APM0_RESULT16
APM0_RESULT17	0xA0040150	32	mixed	0x00000000	APM0_RESULT17
APM0_RESULT18	0xA0040154	32	mixed	0x00000000	APM0_RESULT18
APM0_RESULT19	0xA0040158	32	mixed	0x00000000	APM0_RESULT19
APM0_RESULT20	0xA004015C	32	mixed	0x1FFF0000	APM0_RESULT20
APM0_RESULT21	0xA0040160	32	mixed	0x1FFF0000	APM0_RESULT21
APM0_RESULT22	0xA0040164	32	mixed	0x1FFF0000	APM0_RESULT22
APM0_RESULT23	0xA0040168	32	mixed	0x1FFF0000	APM0_RESULT23
APM0_RESULT24	0xA004016C	32	mixed	0x00000000	APM0_RESULT24
APM1_CFG	0xA0040200	32	mixed	0x00000002	APM1_CFG
APM1_TIMER	0xA0040204	32	rw	0x00000000	APM1_TIMER
APM1_TRG	0xA0040208	32	mixed	0x00000000	APM1_TRG
APM1_RESULT0	0xA004020C	32	ro	0x00000000	APM1_RESULT0
APM1_RESULT1	0xA0040210	32	ro	0x00000000	APM1_RESULT1
APM1_RESULT2	0xA0040214	32	ro	0x00000000	APM1_RESULT2
APM1_RESULT3	0xA0040218	32	ro	0x00000000	APM1_RESULT3
APM1_RESULT4	0xA004021C	32	mixed	0x00000000	APM1_RESULT4
APM1_RESULT5	0xA0040220	32	mixed	0x00000000	APM1_RESULT5
APM1_RESULT6	0xA0040224	32	mixed	0x00000000	APM1_RESULT6
APM1_RESULT7	0xA0040228	32	mixed	0x00000000	APM1_RESULT7
APM1_RESULT8	0xA004022C	32	mixed	0x00000000	APM1_RESULT8
APM1_RESULT9	0xA0040230	32	mixed	0x00000000	APM1_RESULT9
APM1_RESULT10	0xA0040234	32	mixed	0x00000000	APM1_RESULT10
APM1_RESULT11	0xA0040238	32	mixed	0x00000000	APM1_RESULT11
APM1_RESULT12	0xA004023C	32	mixed	0x00000000	APM1_RESULT12
APM1_RESULT13	0xA0040240	32	mixed	0x00000000	APM1_RESULT13
APM1_RESULT14	0xA0040244	32	mixed	0x00000000	APM1_RESULT14
APM1_RESULT15	0xA0040248	32	mixed	0x00000000	APM1_RESULT15
APM1_RESULT16	0xA004024C	32	mixed	0x00000000	APM1_RESULT16

表 9-1：寄存器 （续）

寄存器	地址	宽度	类型	复位值	描述
APM1_RESULT17	0xA0040250	32	mixed	0x00000000	APM1_RESULT17
APM1_RESULT18	0xA0040254	32	mixed	0x00000000	APM1_RESULT18
APM1_RESULT19	0xA0040258	32	mixed	0x00000000	APM1_RESULT19
APM1_RESULT20	0xA004025C	32	mixed	0x1FFF0000	APM1_RESULT20
APM1_RESULT21	0xA0040260	32	mixed	0x1FFF0000	APM1_RESULT21
APM1_RESULT22	0xA0040264	32	mixed	0x1FFF0000	APM1_RESULT22
APM1_RESULT23	0xA0040268	32	mixed	0x1FFF0000	APM1_RESULT23
APM1_RESULT24	0xA004026C	32	mixed	0x00000000	APM1_RESULT24
APM2_CFG	0xA0040300	32	mixed	0x00000002	APM2_CFG
APM2_TIMER	0xA0040304	32	rw	0x00000000	APM2_TIMER
APM2_TRG	0xA0040308	32	mixed	0x00000000	APM2_TRG
APM2_RESULT0	0xA004030C	32	ro	0x00000000	APM2_RESULT0
APM2_RESULT1	0xA0040310	32	ro	0x00000000	APM2_RESULT1
APM2_RESULT2	0xA0040314	32	ro	0x00000000	APM2_RESULT2
APM2_RESULT3	0xA0040318	32	ro	0x00000000	APM2_RESULT3
APM2_RESULT4	0xA004031C	32	mixed	0x00000000	APM2_RESULT4
APM2_RESULT5	0xA0040320	32	mixed	0x00000000	APM2_RESULT5
APM2_RESULT6	0xA0040324	32	mixed	0x00000000	APM2_RESULT6
APM2_RESULT7	0xA0040328	32	mixed	0x00000000	APM2_RESULT7
APM2_RESULT8	0xA004032C	32	mixed	0x00000000	APM2_RESULT8
APM2_RESULT9	0xA0040330	32	mixed	0x00000000	APM2_RESULT9
APM2_RESULT10	0xA0040334	32	mixed	0x00000000	APM2_RESULT10
APM2_RESULT11	0xA0040338	32	mixed	0x00000000	APM2_RESULT11
APM2_RESULT12	0xA004033C	32	mixed	0x00000000	APM2_RESULT12
APM2_RESULT13	0xA0040340	32	mixed	0x00000000	APM2_RESULT13
APM2_RESULT14	0xA0040344	32	mixed	0x00000000	APM2_RESULT14
APM2_RESULT15	0xA0040348	32	mixed	0x00000000	APM2_RESULT15
APM2_RESULT16	0xA004034C	32	mixed	0x00000000	APM2_RESULT16
APM2_RESULT17	0xA0040350	32	mixed	0x00000000	APM2_RESULT17
APM2_RESULT18	0xA0040354	32	mixed	0x00000000	APM2_RESULT18
APM2_RESULT19	0xA0040358	32	mixed	0x00000000	APM2_RESULT19
APM2_RESULT20	0xA004035C	32	mixed	0x1FFF0000	APM2_RESULT20
APM2_RESULT21	0xA0040360	32	mixed	0x1FFF0000	APM2_RESULT21
APM2_RESULT22	0xA0040364	32	mixed	0x1FFF0000	APM2_RESULT22
APM2_RESULT23	0xA0040368	32	mixed	0x1FFF0000	APM2_RESULT23
APM2_RESULT24	0xA004036C	32	mixed	0x00000000	APM2_RESULT24

表 9-1：寄存器 （续）

寄存器	地址	宽度	类型	复位值	描述
APM3_CFG	0xA0040400	32	mixed	0x00000002	APM3_CFG
APM3_TIMER	0xA0040404	32	rw	0x00000000	APM3_TIMER
APM3_TRG	0xA0040408	32	mixed	0x00000000	APM3_TRG
APM3_RESULT0	0xA004040C	32	ro	0x00000000	APM3_RESULT0
APM3_RESULT1	0xA0040410	32	ro	0x00000000	APM3_RESULT1
APM3_RESULT2	0xA0040414	32	ro	0x00000000	APM3_RESULT2
APM3_RESULT3	0xA0040418	32	ro	0x00000000	APM3_RESULT3
APM3_RESULT4	0xA004041C	32	mixed	0x00000000	APM3_RESULT4
APM3_RESULT5	0xA0040420	32	mixed	0x00000000	APM3_RESULT5
APM3_RESULT6	0xA0040424	32	mixed	0x00000000	APM3_RESULT6
APM3_RESULT7	0xA0040428	32	mixed	0x00000000	APM3_RESULT7
APM3_RESULT8	0xA004042C	32	mixed	0x00000000	APM3_RESULT8
APM3_RESULT9	0xA0040430	32	mixed	0x00000000	APM3_RESULT9
APM3_RESULT10	0xA0040434	32	mixed	0x00000000	APM3_RESULT10
APM3_RESULT11	0xA0040438	32	mixed	0x00000000	APM3_RESULT11
APM3_RESULT12	0xA004043C	32	mixed	0x00000000	APM3_RESULT12
APM3_RESULT13	0xA0040440	32	mixed	0x00000000	APM3_RESULT13
APM3_RESULT14	0xA0040444	32	mixed	0x00000000	APM3_RESULT14
APM3_RESULT15	0xA0040448	32	mixed	0x00000000	APM3_RESULT15
APM3_RESULT16	0xA004044C	32	mixed	0x00000000	APM3_RESULT16
APM3_RESULT17	0xA0040450	32	mixed	0x00000000	APM3_RESULT17
APM3_RESULT18	0xA0040454	32	mixed	0x00000000	APM3_RESULT18
APM3_RESULT19	0xA0040458	32	mixed	0x00000000	APM3_RESULT19
APM3_RESULT20	0xA004045C	32	mixed	0x1FFF0000	APM3_RESULT20
APM3_RESULT21	0xA0040460	32	mixed	0x1FFF0000	APM3_RESULT21
APM3_RESULT22	0xA0040464	32	mixed	0x1FFF0000	APM3_RESULT22
APM3_RESULT23	0xA0040468	32	mixed	0x1FFF0000	APM3_RESULT23
APM3_RESULT24	0xA004046C	32	mixed	0x00000000	APM3_RESULT24

XILINX

第 10 章

用核设计

本章包括有助于用核进行设计的指南和其他信息。

一般设计指南

Video Codec Unit (VCU) 核是编程逻辑 (PL) 中的专用硬件块。所有接口都是通过 PL 中的 AXI 互连块连接的。VCU 核在其 AXI 主接口上符合 AXI-4 标准。它可以连接到与 PL 存储器控制器的 PS 或 AXI 接口兼容的 S_AXI_HP_FPD （或 S_AXI_LPD、 S_AXI_HPC_FPD）端口。从 VCU 到处理器系统 (PS) 之间没有直接的 （硬连线）连接。

对于需要同时进行编码器和解码器操作的高带宽 VCU 应用，编码器应连接到 PS DDR；解码器应连接到 PL DDR。如需了解更多信息，请参阅第 6 章"Zynq UltraScale+ EV 架构 Video Codec Unit DDR4 LogiCORE IP"。该方法最有效地利用有限的 AXI4 读/写发布能力来最小化解码器的时延。 DMA 缓存共享要求将决定应如何将捕获、显示和中间处理阶段映射到 PS 或 PL DDR。

VCU 核的寄存器编程接口是连接到 PS 通用 (GP) 端口的。时钟可以从 PL 使用，也可以通过 VCU 核内的内部 PLL 使用。

中断

从 VCU 核到 PS (vcu_host_interrupt) 有一条中断线。该中断必须连接到 PL-PS-IRQ0[7:0] 或 PL-PS-IRQ1[7:0]。如果设计中存在其他中断，则必须将中断与其他中断连接，然后连接到 PS。

XILINX

第 11 章

设计流程步骤

本章介绍如何自定义和生成核、约束核以及应按照哪些特定于此 IP 核的仿真、综合和实现步骤进行操作。如需了解标准的 Vivado® 设计流程和 IP 集成器方面的详情，请参阅以下 《Vivado Design Suite 用户指南》：

- 《Vivado Design Suite 用户指南：使用 IP 集成器设计 IP 子系统》(UG994) [参照 1]
- 《Vivado Design Suite 用户指南：使用 IP 进行设计》(UG896) [参照 2]
- 《Vivado Design Suite 用户指南：入门指南》(UG910) [参照 4]
- 《Vivado Design Suite 用户指南：逻辑仿真》(UG900) [参照 5]

Vivado Design Suite 集成设计环境

VCU 只能被添加到 Vivado Design Suite 中的 Vivado IP 集成块设计中。如需了解有关如何在 Vivado IP 集成器中自定义和生成核的详情，请参阅 《Vivado Design Suite 用户指南：使用 IP 集成器设计 IP 子系统》(UG994) [参照 1]。验证或生成设计时， IP 集成器可能会自动计算某些配置值。要查看配置值是否会更改，请参阅本章中的参数说明。要查看参数值，在 Tcl 控制台中运行 validate_bd_design 命令。

您可以使用以下步骤通过指定各种 IP 核相关参数值对设计中要使用的 IP 进行自定义：

1. 在 Flow Navigator 中，单击 IP 集成器标题下的“Create Block Diagram”或“Open Block Design”。
2. 右键单击图表并选择“Add IP”。

   系统随即会打开可搜索的 IP 目录。您还可以通过单击“IP Integrator Block Design”菜单左侧的“Add IP”按钮来添加 IP。

3. 单击 IP 名称，然后按“Enter”键或双击 IP 名称。
4. 双击所选的 IP 块，或从右键单击弹出的菜单中选择“Customize Block”命令。

欲知详情，请参阅 《Vivado Design Suite 用户指南：使用 IP 进行设计》(UG896) [参照 2] 和 《Vivado Design Suite 用户指南：入门指南》(UG910) [参照 4]。

注释：本章中的附图是 Vivado IDE 的插图。此处展示的布局可能与当前版本中的布局有所不同。

注释：LogiCORE™ IP 仅通过 Vivado IP Integrator （IP 集成器）流程提供，而不作为独立的 IP.提供。

如图 11-1 所示， “Basic Configuration”标签允许选择用来计算编码器缓存大小和编码器/解码器块使用的动态总功耗的视频参数。

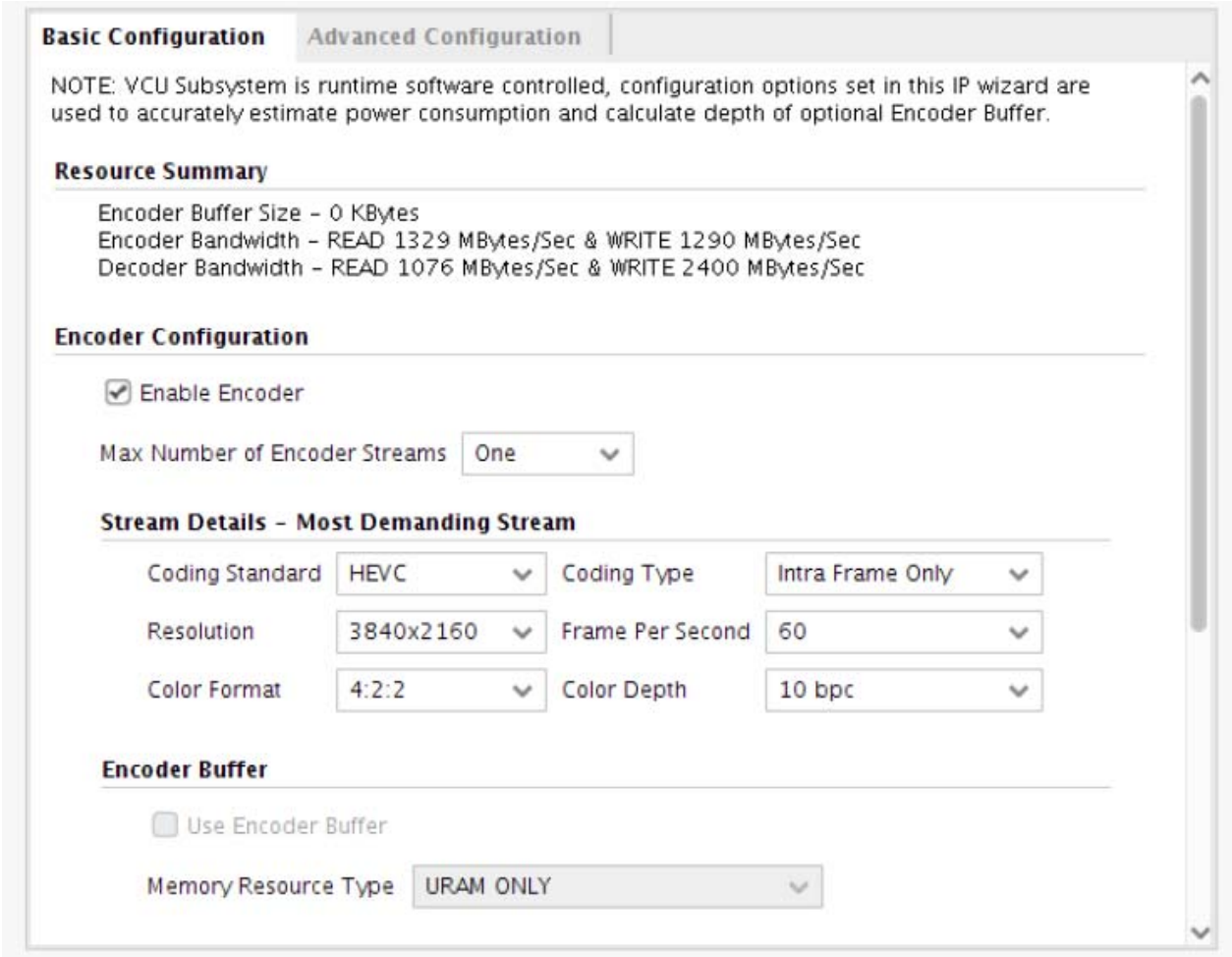

图 11-1：“Basic Configuration”标签

VCU 子系统在运行时由软件控制。 VCU GUI 中设置的配置选项用来估算功耗、估算带宽和计算编码器缓存的大小。有关在运行时控制配置参数的信息，请参阅第 124 页的“VCU 控制软件示例应用”和第 133 页的“VCU 控制软件 API”。

“Basic Configuration”标签上的参数如下：

- Component Name：组件名称由 IP 集成器自动设置。
- Resource Summary：报告编码器缓存的大小。报告编码器和解码器的带宽。
- Encoder Configuration：编码器设置有以下参数：
  - Enable Encoder：启用编码器及相关参数。
  - Max Number of Encoder Streams：选择 1 到 8 个流。确定存储器要求。

    注释：VCU 支持 32 个流，但赛灵思建议使用 GUI 中的可用选项来选择最接近的组合。
  - Coding Standard：选择 AVC 或 HEVC。
  - Coding Type：选择要用于编码的 GOP 结构：
    - Intra Frame Only - 仅限 I 帧
    - Intra & Inter Frame - I 帧、 B 帧和 P 帧

    只有在选择 Intra 和 Inter Frame 时才能启用编码器的缓存。
  - Resolution：选择以下分辨率之一：

- 1280×720
- 1920×1080
- 3840×2160
- 4096×2160
- 7680×4320

◦ Frames Per Second：选择 15、 30、 45 或 60 fps。在 7680×4320 分辨率下，只有 15fps 可用。

◦ Color Format：选择以下颜色格式之一：

- 4:0:0 - 单色
- 4:2:0
- 4:2:2

◦ Color Depth：选择每通道 8 位或 10 位。

◦ Use Encoder Buffer：选择是否要使用编码器缓存。只有在选择了 Intra 和 Inter Frame 之后才能启用编码器的缓存。编码器缓存通过缓存可编程逻辑中的数据来减少外部存储器的带宽，但它可能会略微降低视频的质量。

◦ Memory Resource Type：选择下列存储器类型选项：

- URAM ONLY
- BRAM ONLY
- COMBINATION - URAM & BRAM

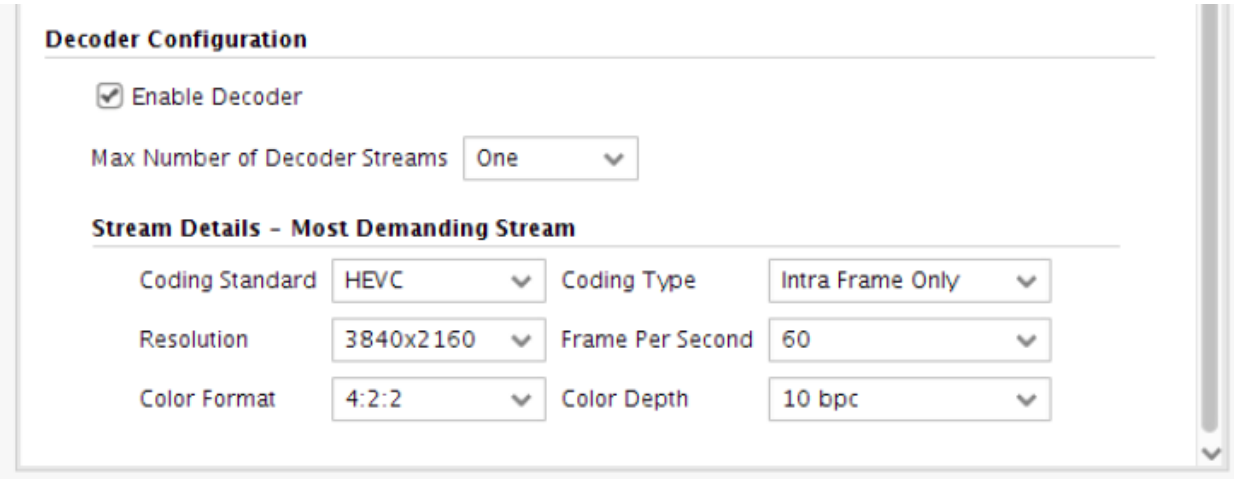

图 11-2：“Decoder Basic Configuration”标签

• Decoder Configuration：解码器设有以下参数：

◦ Enable Decoder：启用解码器及相关参数。

◦ Max Number of Decoder Streams：选择 1 到 8 个流。确定存储器要求。

注释：VCU 支持 32 个流，但赛灵思建议使用 GUI 中的可用选项来选择最接近的组合。

◦ Coding Standard：选择 AVC 或 HEVC。

◦ Resolution：选择以下分辨率之一：

- 1280×720
- 1920×1080
- 3840×2160
- 4096×2160
- 7680×4320

- Frames Per Second：选择 15、 30、 45 或 60 fps。在 7680×4320 分辨率下，只有 15fps 可用。
- Color Format：选择以下颜色格式之一：
  - 4:0:0 - 单色
  - 4:2:0
  - 4:2:2
- Color Depth：选择每通道 8 位或 10 位。

如图 11-3 所示， “Advanced Configuration”标签让您可以覆盖编码器缓存的存储器深度、压缩功能和编码器核时钟。

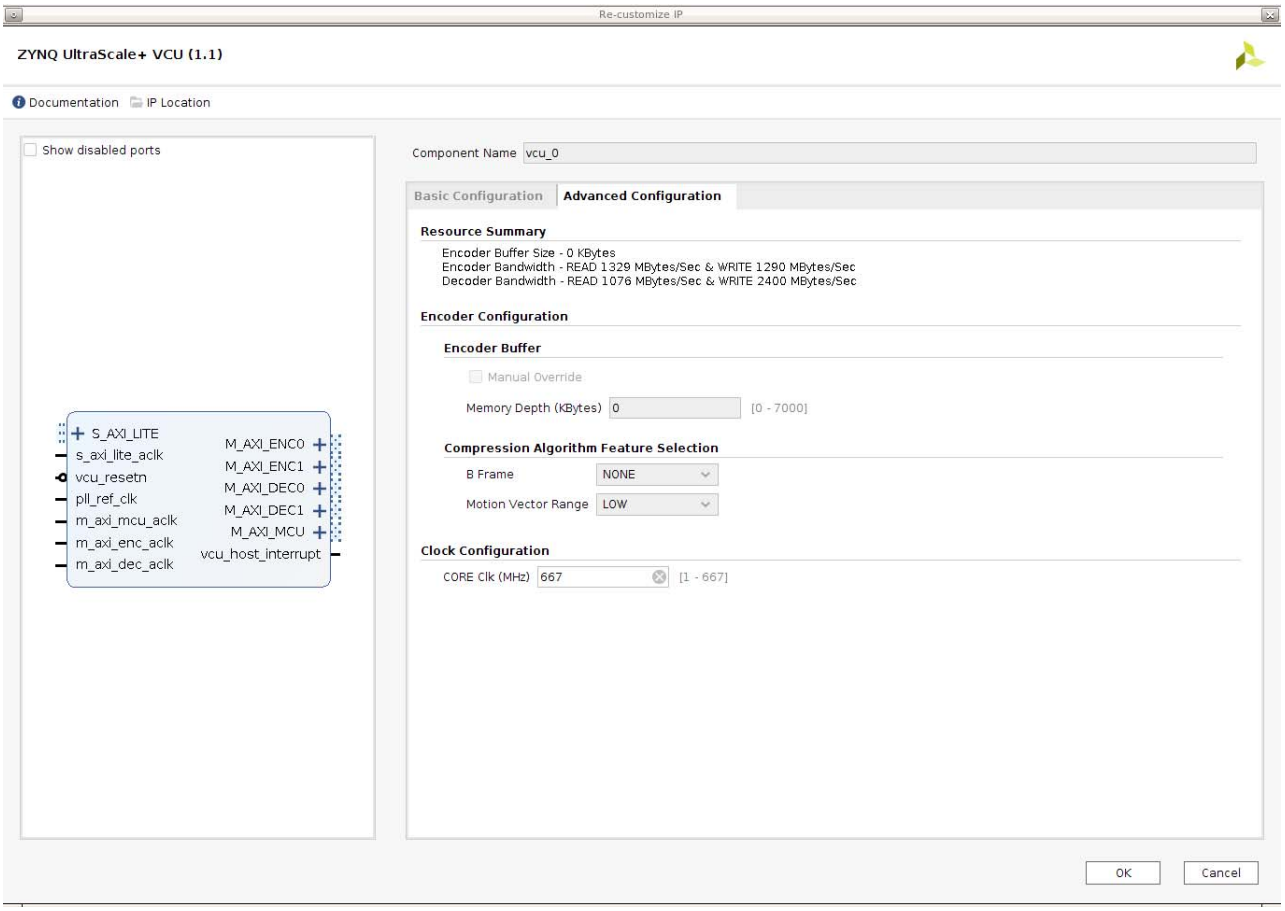

图 11-3：“Advanced Configuration”标签

仅当“Basic Configuration”标签设置如下时，系统才会启用编码器缓存的高级配置选项：

- “Coding Type”是“Intra & Inter Frame”
- “Use Encoder Buffer”被选中

“Advanced Configuration”标签上的参数设置如下：

- Manual Override：选择此选项可覆盖由 IP 集成器计算的编码器缓存存储器的大小。
- Memory Depth (Kbytes)：如果选中“Manual Override”复选框，则可以输入 0 到 7,000 Kbyte 的存储器大小。
- B Frame：选择以下选项之一：
  - NONE - 最低时延
  - STANDARD - GOP 配置 IPPP （内部周期为 30 ms）

  - HIERARCHICAL - 又称为金字塔形结构。可与 3、 5 或 7 个 B 帧配合使用。
- Motion Vector Range：决定编码器缓存的大小：
  - LOW
  - MEDIUM
  - HIGH

  资源摘要中报告了确切的编码器缓存大小。
- CORE Clk (MHz)：选择 1 - 667 MHz 的时钟频率。

多流解码器配置用例

例如，使用“Decoder Configuration”标签对三个分辨率为 1080p60 的流进行编码的多流用例。 VCU 的最大带宽为：

1 Stream * 3840 x 2160 fps 60

或者

8 Stream * 1920 x 1080 fps 30

1. 设置“Max Number of Encoder Streams”。
2. 设置“Resolution”和“Frames Per Second”，这样最大分辨率乘以流的最大个数会重新呈现您的系统上使用的最大带宽计划。

   针对上面列出的案例而言：
   - 将“Max Number of Encoded Streams”设置为 3。
   - 将“Resolution”设置为 1920x1080。
   - 将“Frames Per Second”设置为 60。

注释：

- 如果多个流使用不同的颜色格式和颜色深度，则必须输入视频参数 （表示颜色格式或颜色深度）范围的上限。
- 仅针对 Intra 编码和 Inter frame 编码启用编码器缓存选项。编码器缓存用于对运动进行估计。

将核与 Zynq UltraScale + MPSoC 器件连接

要将 VCU 核集成到 IP Integrator (IP 集成器) 块设计中，请执行以下步骤：

1. 启动 Vivado IDE 创建一个新工程。（图 11-4）

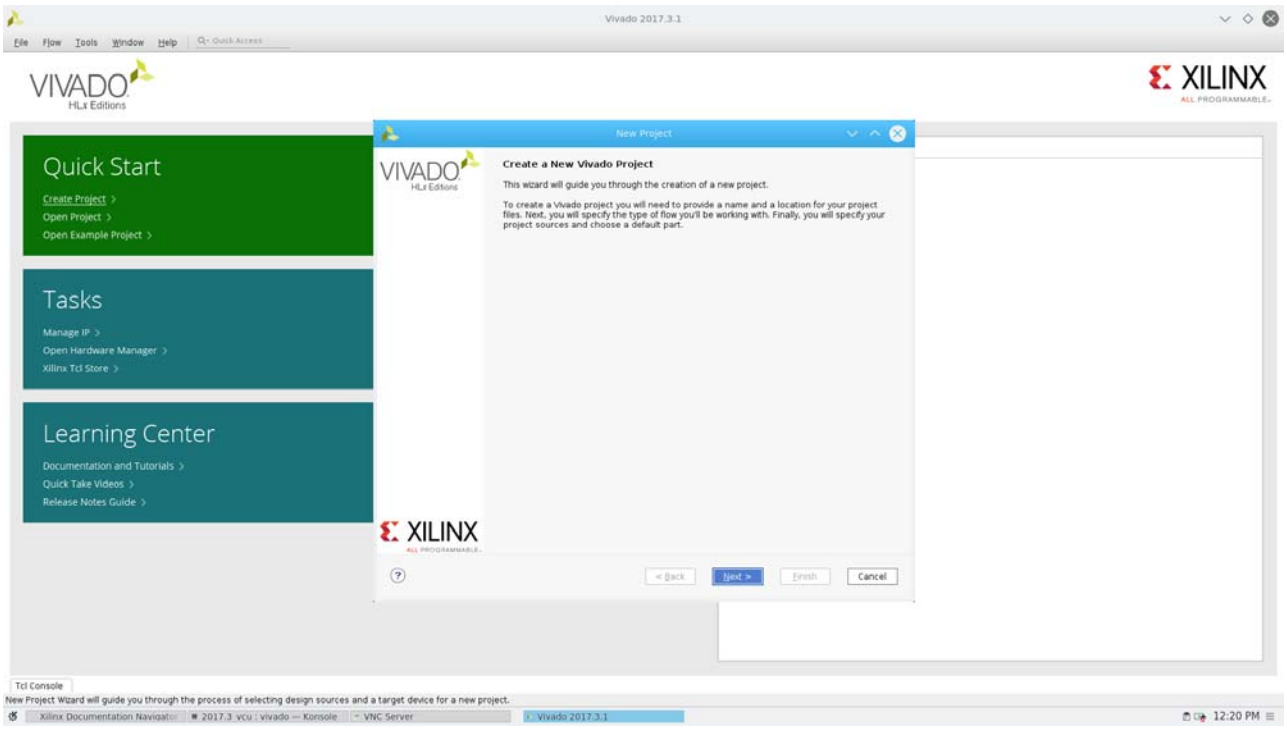

图 11-4：新的 Vivado 工程

2. 在 New Project wizard 上单击“Next”，直到转到“Family Selection”窗口。
3. 选择 VCU 核的目标器件。（图 11-5）

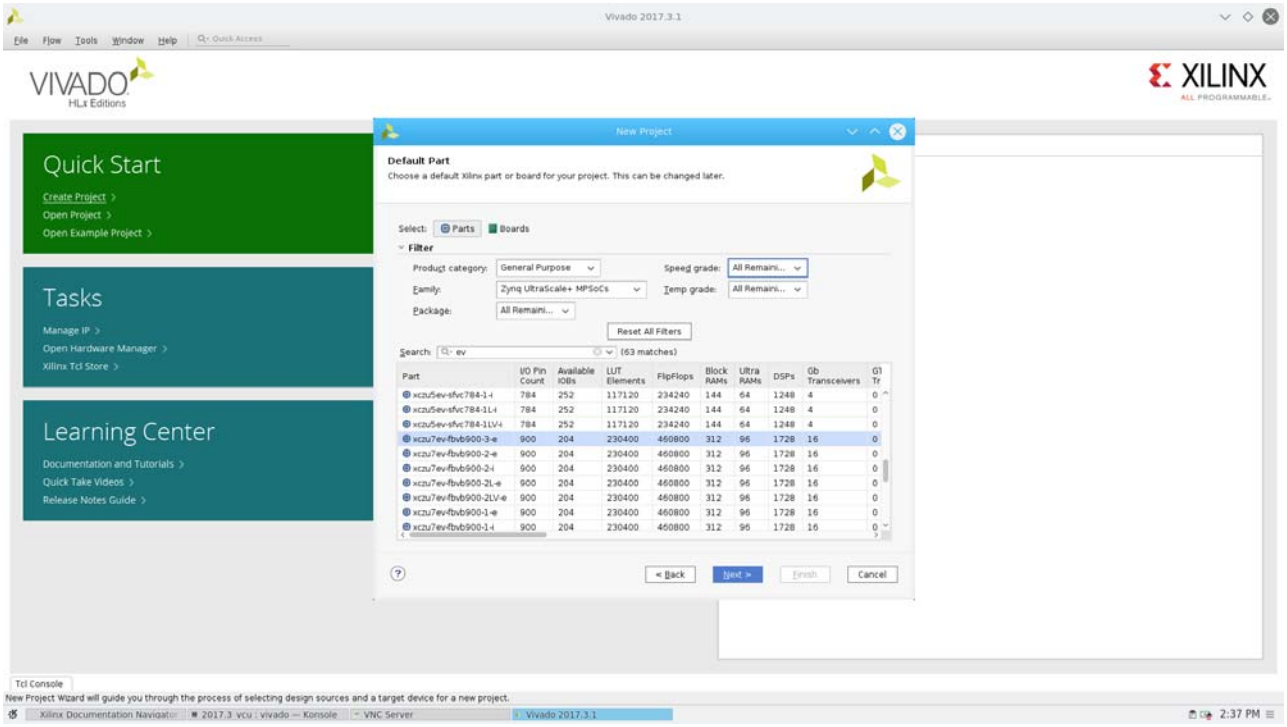

图 11-5：“Family Selection”标签

4. 单击“Project Settings”窗口。如图 11-6 所示，单击“Implementation”。

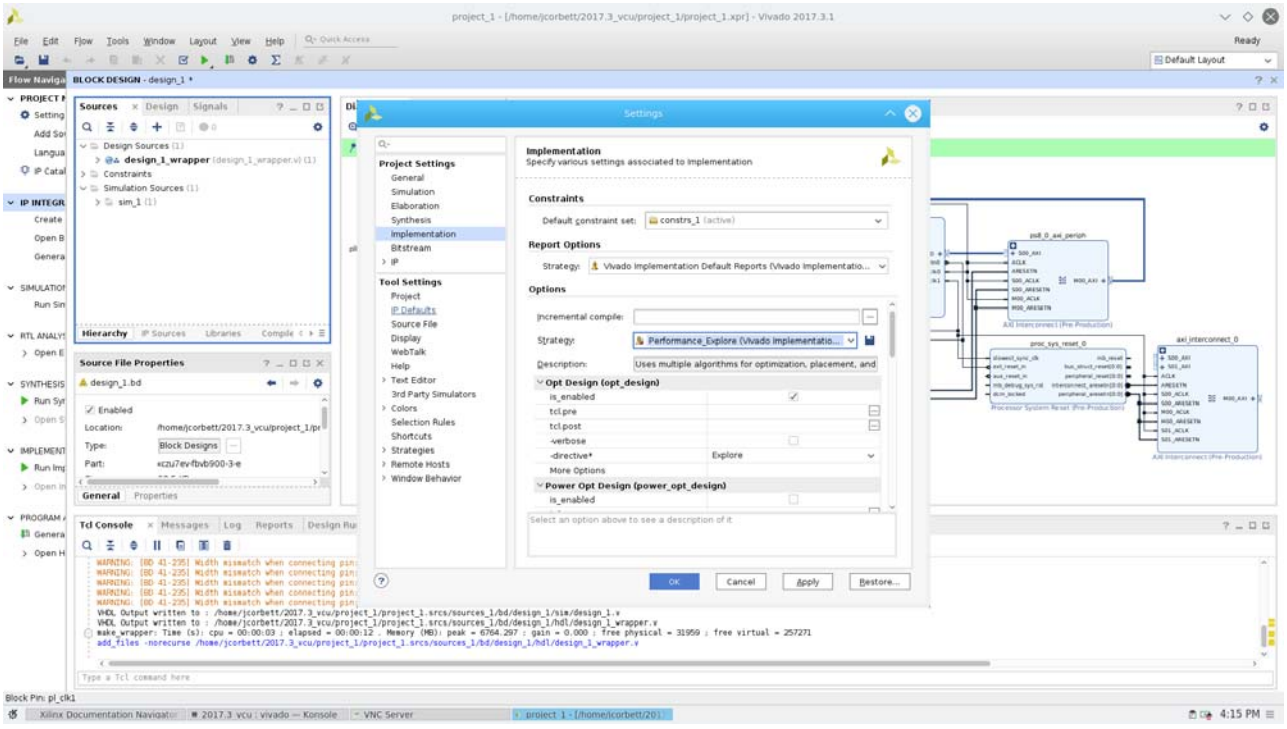

图 11-6：“Project Settings”标签

5. 在“Settings”窗口，启用“Performance_Explore”选项。
   “Settings > Implementation > Options > Strategy: Performance_Explore”
   如需了解更多信息，请参阅 《Vivado Design Suite 用户指南：设计分析和收敛技巧》[参照 8]。
6. 单击“Create Block Design”选项。
7. 单击“Add IP”选项，然后键入“VCU”。系统随即会显示以下 IP。

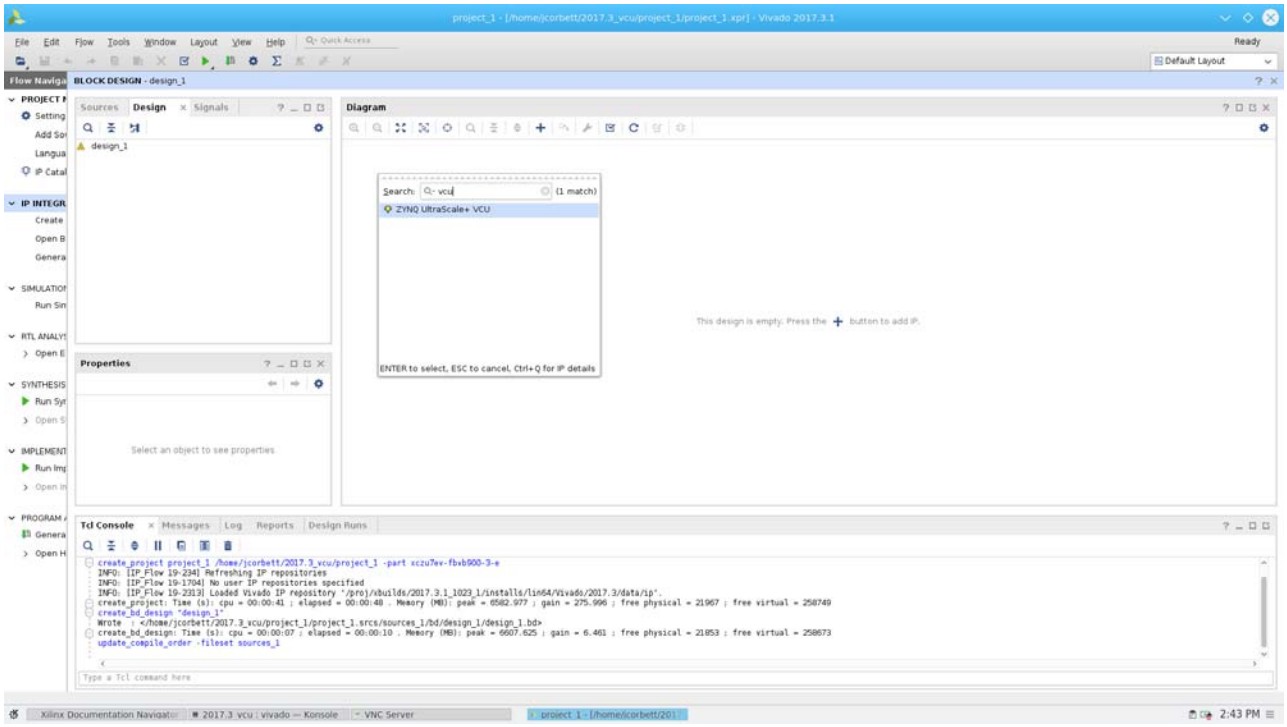

图 11-7：Zynq UltraScale+ VCU

8. 将 Zynq UltraScale+ VCU 添加到块设计中。
9. 将 Zynq UltraScale+ MPSoC IP 添加到块设计中，如图 11-8 所示。

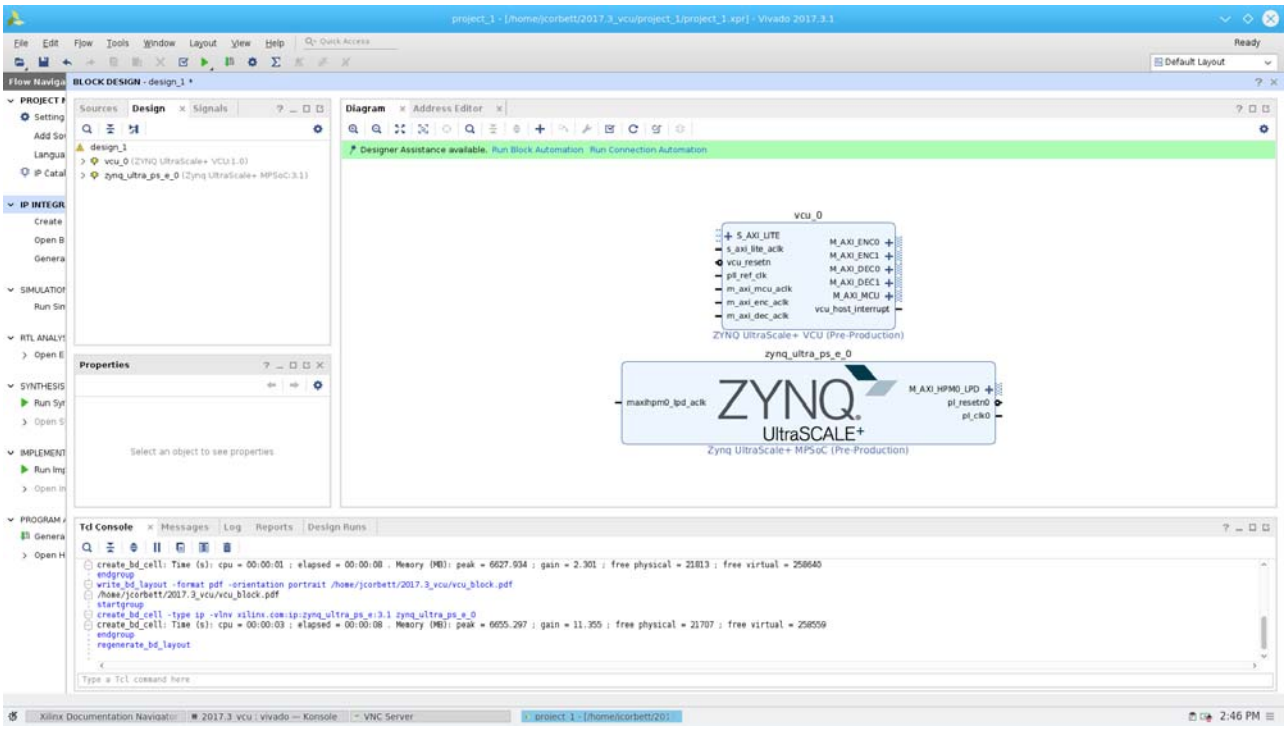

图 11-8：Zynq UltraScale+ MPSoC

10. 配置 Zynq UltraScale+ MPSoC，以根据您的设计要求启用 AXI 从接口、时钟和 PL-PS 中断信号。有关 Zynq UltraScale+ MPSoC IP 的配置选项，请参阅 《Zynq UltraScale+ MPSoC Processing System 产品指南》[参照 17]。

    图 11-9 显示的是配置 PS-PL 接口信号的示例。

11. 如图 11-9 所示，选择 PL1 的时钟频率为 333 MHz。

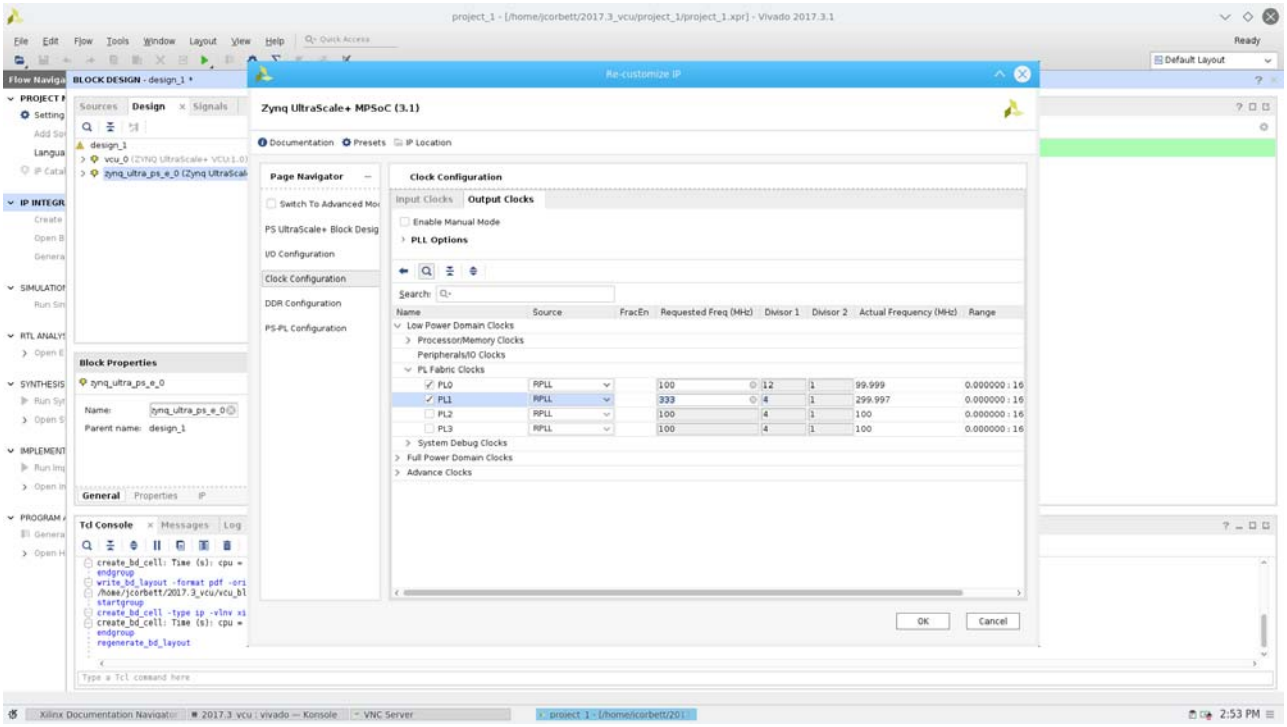

图 11-9：重新自定义 IP

12. 如图 11-10 所示，启用 IRQ0 [0-7] 和 HP0-3 端口。

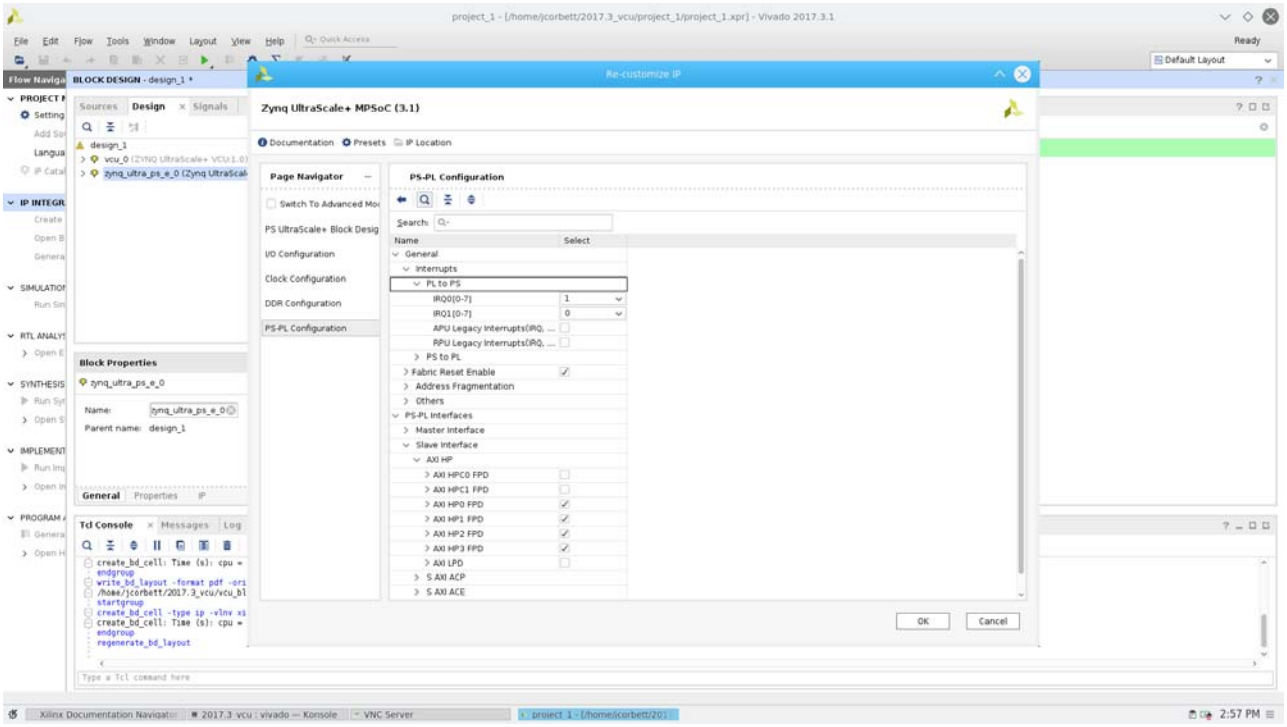

图 11-10：输出 PL-PS 配置

13. 使用自动连接将 VCU IP 的 S_AXI_LITE 接口连接到 M_AXI_HPM0_LPD 接口。

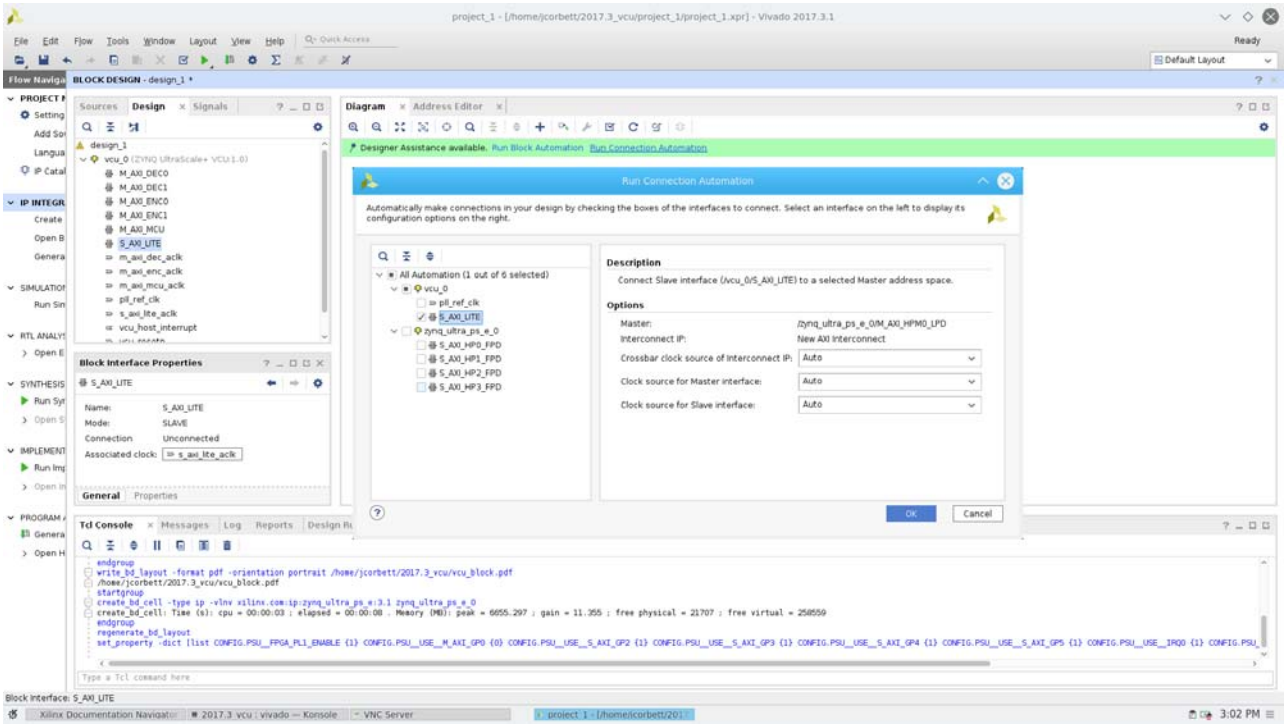

图 11-11：Connection Wizard

14. 手动连接以下接口：
    - Zynq UltraScale+ VCU.M_AXI_ENC1 到 Zynq UltraScale+ MPSoC.S_AXI_HP1_FPD
    - Zynq UltraScale+ VCU.M_AXI_DEC0 到 Zynq UltraScale+ MPSoC.S_AXI_HP0_FPD
    - Zynq UltraScale+ VCU.M_AXI_DEC1 到 Zynq UltraScale+ MPSoC.S_AXI_HP3_FPD

    请注意选择 MPSoC.S_AXI_HP1_FPD、 MPSoC.S_AXI_HP0_FPD 和 MPSoC.S_AXI_HP3_FPD。这些选项是用于大型数据集的非连贯、高性能的 DMA 端口。它们支持 AXI FIFO QoS-400 流量整形。每个端口都有一套相关的寄存器，因此需要寄存器地址并使用 devmem 命令对服务质量和发布功能的命令进行配置。

    从 《Zynq UltraScale+ MPSoC 寄存器参考资料》(UG1087) [参照 14] 中可以找到 S_AXI_HP1_FPD 的地址和描述。地址是 0xFD390000。寄存器被用来配置 QoS 和 FIFO，是 AFIFM 模块的一部分。 AFIFM Module 文档提供了定义流量优先级和最大读取或写入命令数字段的相对地址和值。

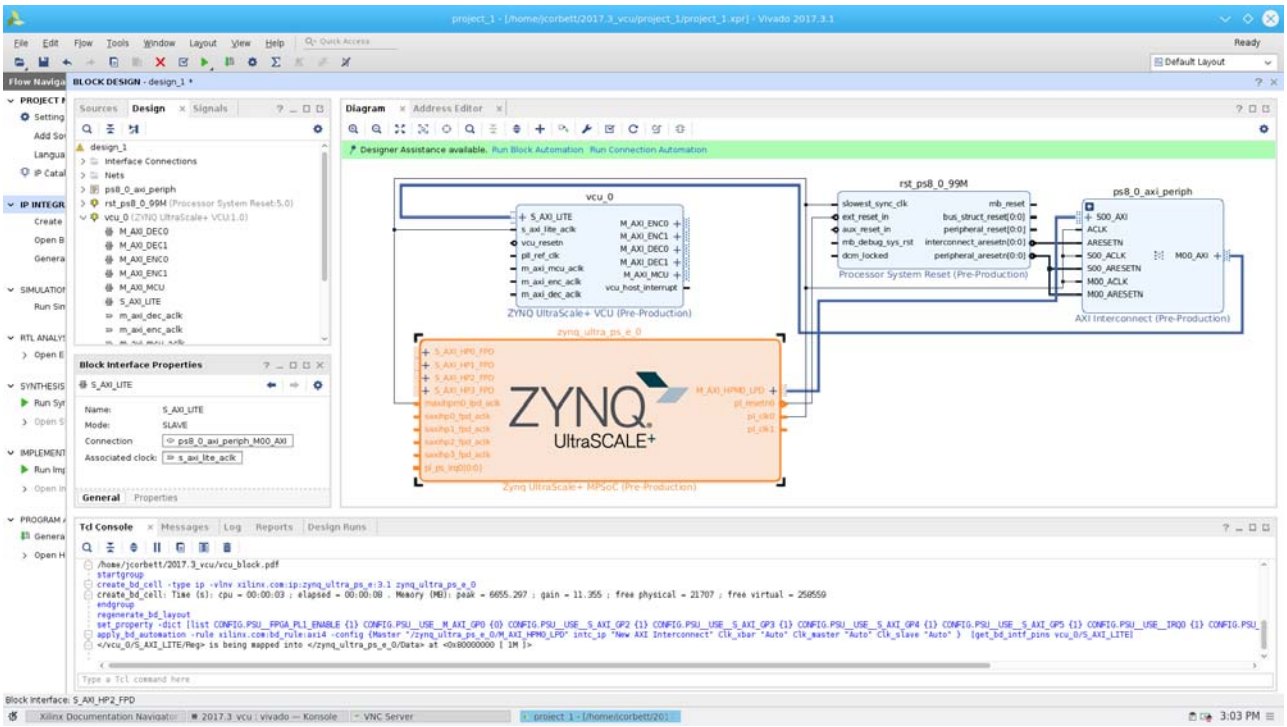

图 11-12：Connection Wizard

15. 如图 11-13 所示，添加 AXI Interconnect IP 并将从接口数设置为 2、将主接口设置为 1。

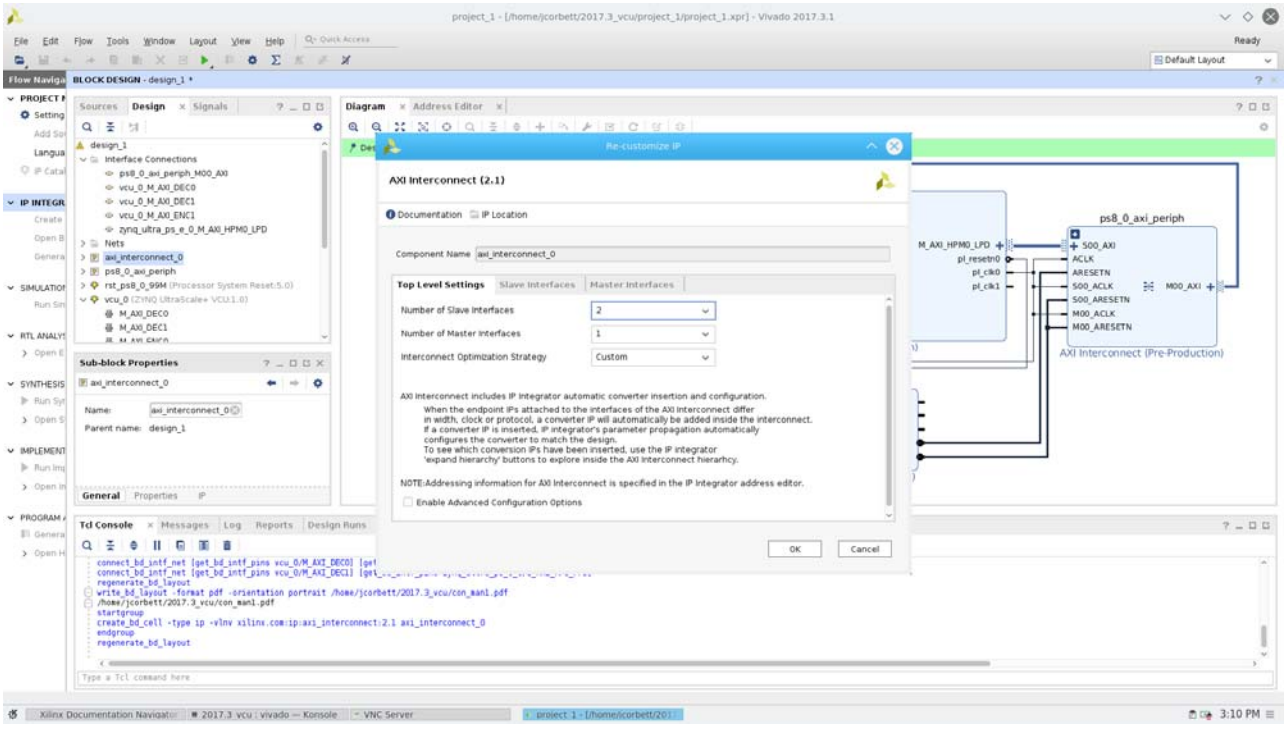

图 11-13：AXI 互连

16. 手动执行以下连接：
    - 将处理器系统实例化以复位 IP。 `pl_clk1` 时钟域需要第二个复位块。

- 将最慢的同步时钟连接到 `pl_clk1` 端口。
- 将 `interconnect_aresetn` 端口用作 AXI 互连 IP 核的 `ARESETN` 输入。
- 如图 11-14 所示，使用 `peripheral_aresetn` 端口作为 `S00_ARESETN`、 `S01_ARESETN` 和 `M00_ARESETN` 端口的复位输入。
- 将 `ext_reset_n` 信号连接到 Zynq UltraScale+ MPSoC 的 `pl_resetn0` 信号。
- 将 `vcu_host_interrupt` 连接到 Zynq UltraScale+ MPSoC IP 的 `pl_ps_irq` 端口。

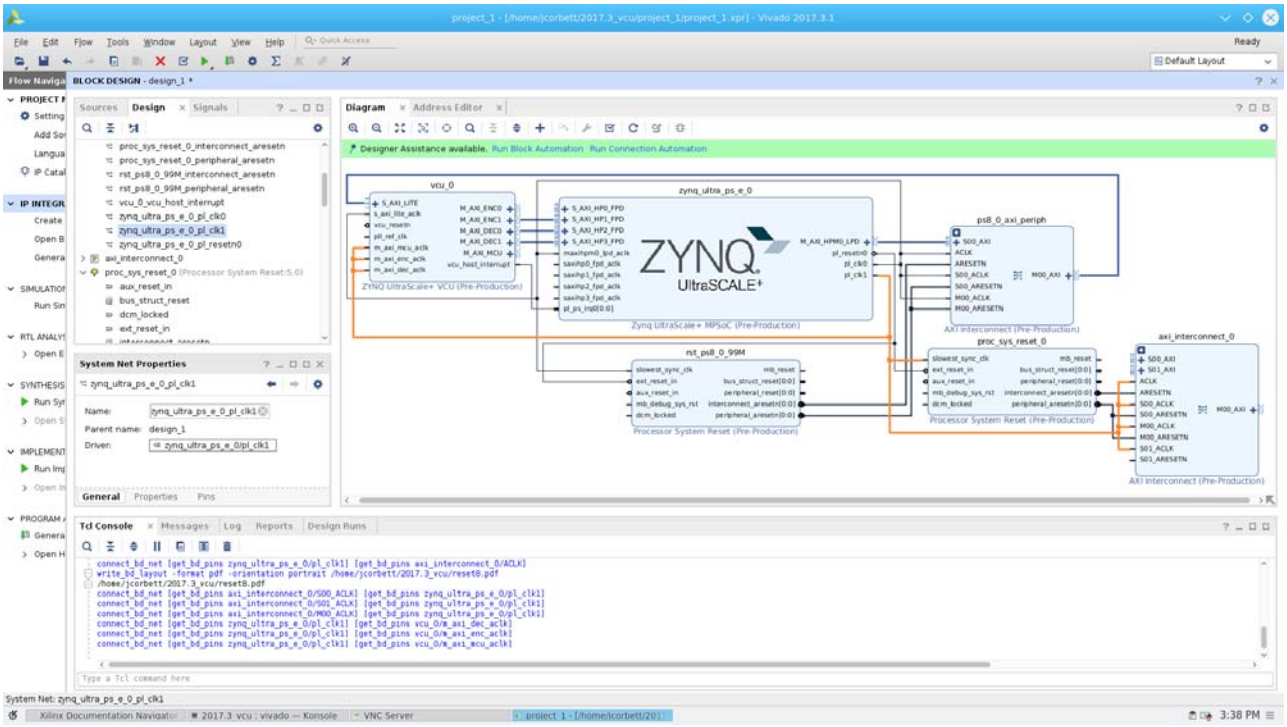

图 11-14：图

17. 将以下时钟连接到 Zynq UltraScale+ MPSoC 核的 `pl_clk1` 输出：
    - AXI Interconnect - aclk
    - AXI Interconnect - s00_aclk
    - AXI Interconnect - s01_aclk
    - AXI Interconnect - m01_aclk
    - VCU - m_axi_mcu_aclk
    - VCU - m_axi_enc_aclk
    - VCU - m_axi_dec_aclk
    - Zynq UltraScale+ MPSoC - saxihp0_fpd_aclk
    - Zynq UltraScale+ MPSoC - saxihp1_fpd_aclk
    - Zynq UltraScale+ MPSoC - saxihp2_fpd_aclk
    - Zynq UltraScale+ MPSoC - saxihp3_fpd_aclk
18. 将 `saxihp0_fpd_aclk`、 `saxihp1_fpd_aclk`、 `saxihp2_fpd_aclk` 和 `saxihp3_fpd_aclk` 连接到 Zynq UltraScale+ MPSoC 核的 `pl_clk1` 输出。
19. 使用 LogiCORE IP 锁定 Zynq UltraScale+ VCU 常数 1 的 `vcu_resetn` 信号。
20. 将 `pll_ref_clk` 信号设为外部信号。

21. 在“Address Editor”窗口中，展开 EncData 地址段并自动分配地址。图 11-15 示出的是一个地址映射示例。

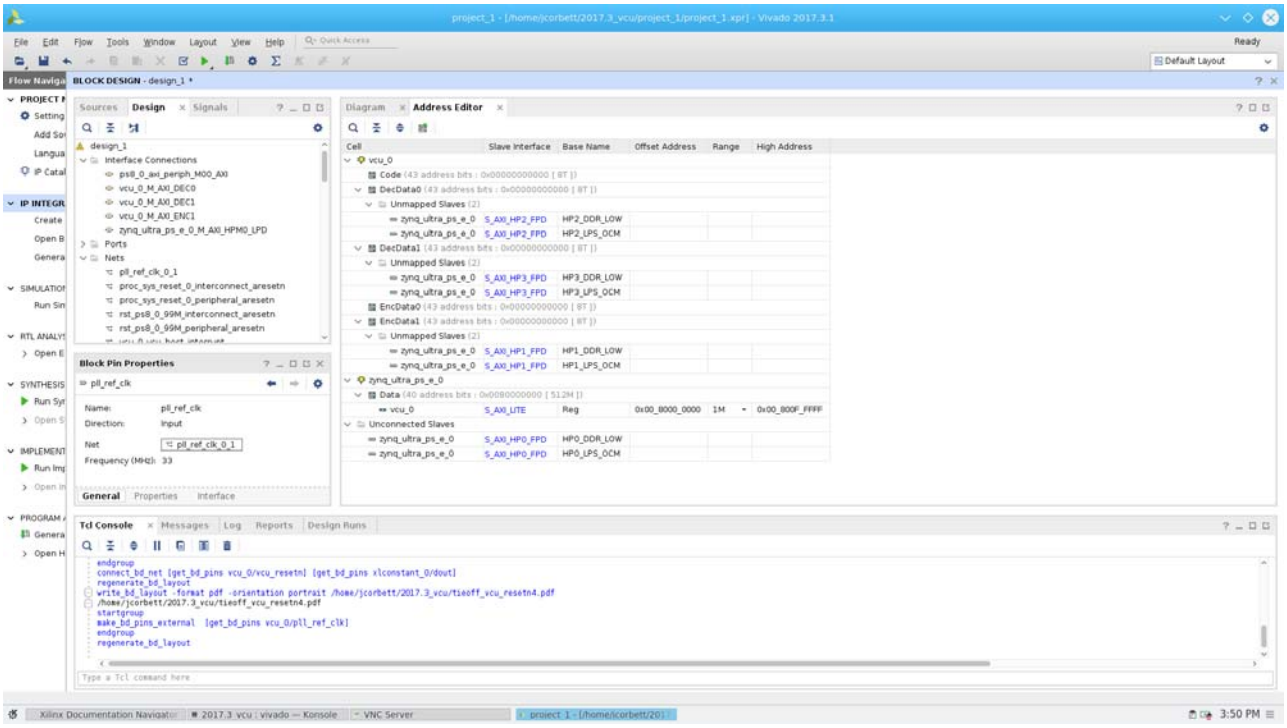

图 11-15：Address Editor

22. 单击“Validate Block Design”验证连接。

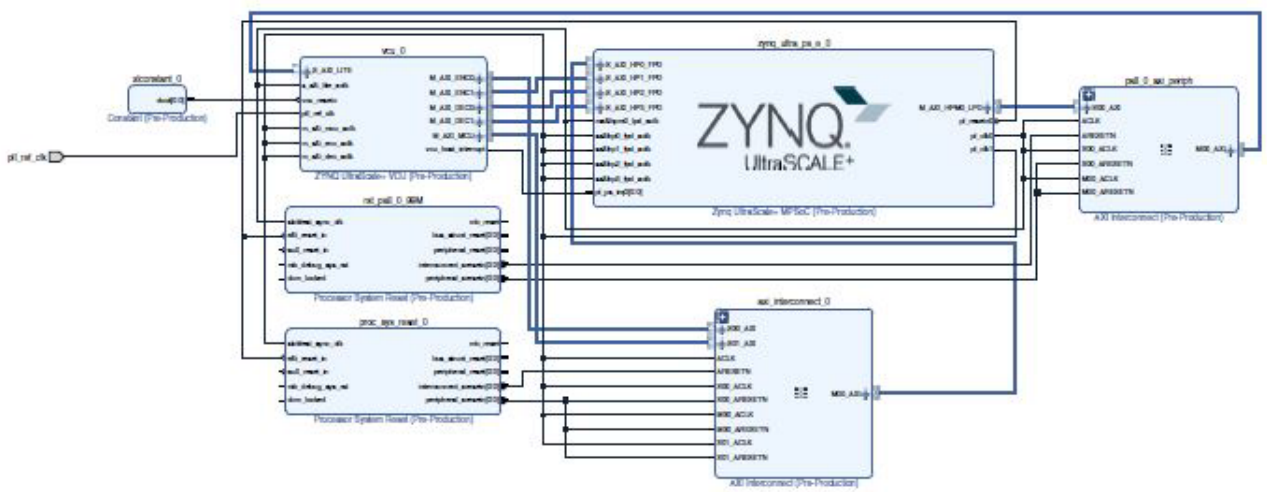

图 11-16：Validate Block Design

23. 如图 11-17 所示，通过右键单击“Block Design”并选择“Create HDL Wrapper”选项来创建顶层 Vivado 封装。

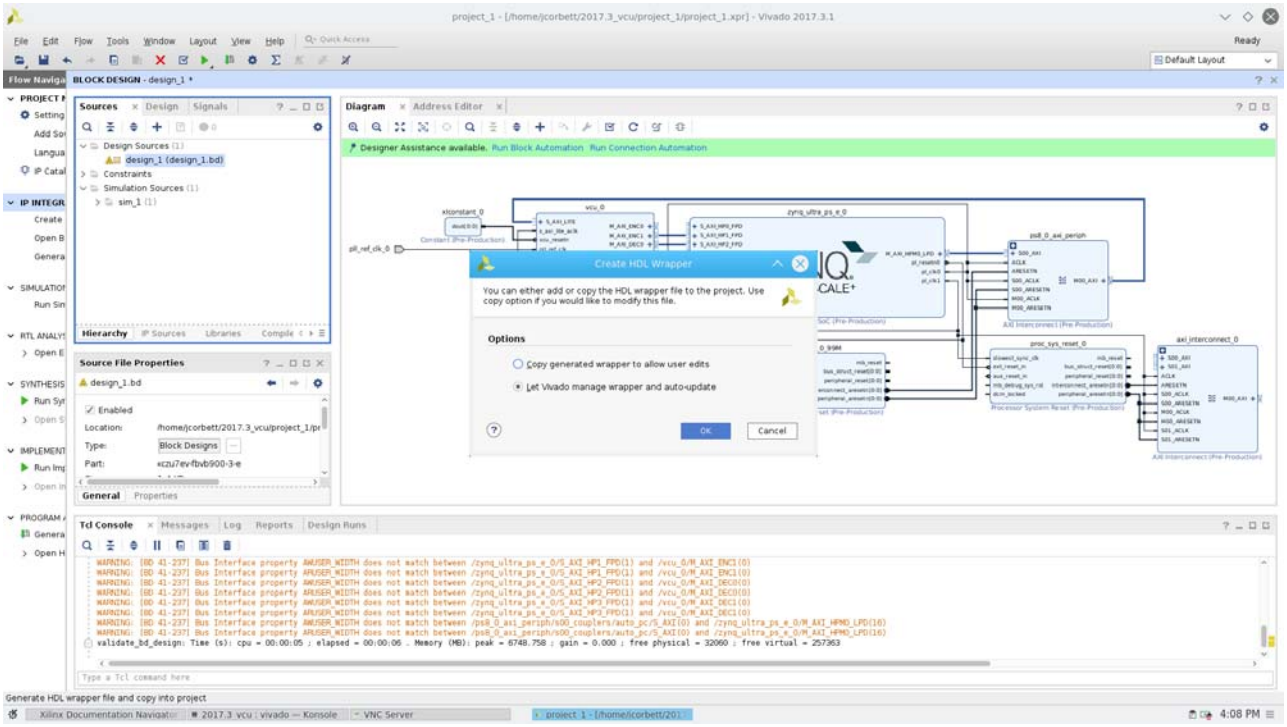

图 11-17：创建 HDL 封装

24. 将约束文件添加到工程中。
25. 如果可用，从板级支持包 (BSP) 添加约束文件 (.xdc)。如果没有可用的约束文件，则必须从其默认值更改多个设置以实现无差错的比特流生成。在 I/O 端口窗口中，对于 `pll_ref_clk_0`，必须将 I/O Std 从 LVCMOS18 （默认）更改为 LVCMOS18。

    在同一行，必须选中“Fixed”复选框。这与包含以下内容的 XDC 文件相对应：

    - `set_property IOSTANDARD LVCMOS18 [get_ports pll_ref_clk_0]`。
    - `set_property PACKAGE_PIN AA2 [get_ports pll_ref_clk_0]`
26. 单击“Run Synthesis”、 “Run Implementation”或“Generate Bitstream”选项。

为解码器启用 PL-DDR

用例 1 (UC1) 指的是多媒体流水线，在此流水线中解码器和编码器使用 PS_DDR 进行缓存分配和进行视频处理的存储器读/写操作。目前的系统能够以 4k @ 60 fps 的分辨率进行编码和解码、以带 PS_DDR 的可用带宽、 4k @ 30 fps 的分辨率进行转码。目标是实现以 4k @ 60fps 的分辨率转码，并且已经确定 PS_DDR 带宽是瓶颈。目前已经提出一种新的设计来克服 PS_DDR 带宽的限制。

用例 2 (UC2) 是一种新的设计方法，建议使用 PL_DDR 进行解码、使用 PS_DDR 进行编码。这样，DDR 带宽就足以实现 4k @ 60fps 的转码了。图 11-18 对转码流水线进行了介绍。解码器可以完成解码过程、将数据写入 PL_DDR，并将其复制到 PS_DDR （编码器在这里会消耗掉数据）。从 PS_DDR 到 PL_DDR 的缓存副本是通过 DMA 传输实现的。

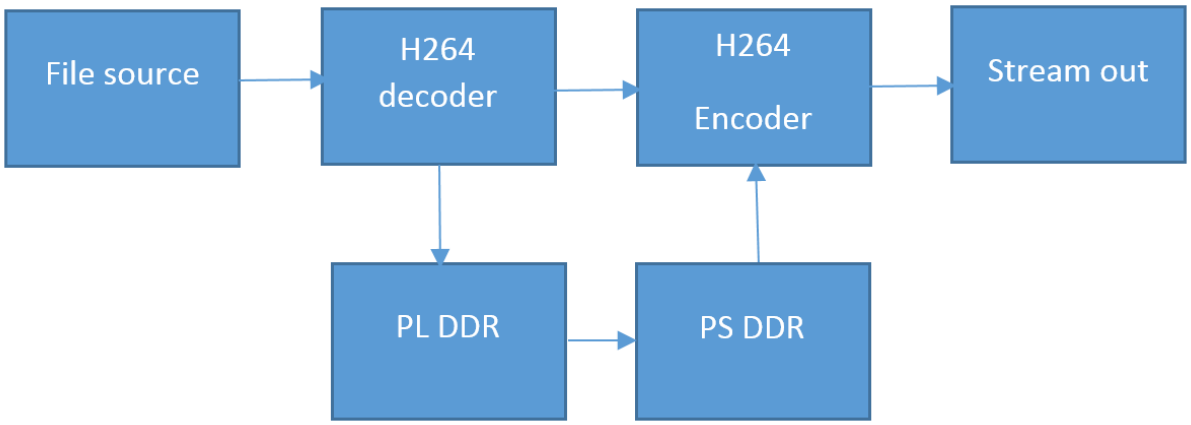

图 11-18：用例 2

启用 PL-DDR

要在 BSP 中启用 PL-DDR，需要进行以下 dts 更改。

UC2 设计需要两个帧缓存 IP 才能进行 DMA 传输，以下器件节点可构建视频的 mem2mem 流水线。

```
video_m2m {
            compatible = "xlnx, v-m2m";
            dmas = <&mem2mem_v_frmbuf_rd_0 0>, <&mem2mem_v_frmbuf_wr_0 0>;
            dma-names = "tx", "rx";
};
```

在 PL_DDR 设备树节点中更新专用于 VCU 解码器的专用存储器。

```
mig_0: ddr4@004800000000 {
           compatible = "xlnx, ddr4-2.2";
           reg = <0x00000048 0x00000000 0x0 0x80000000>;
           plmem_vcu_dec: pool@0 {
                     reg = <0x48 0x00000000 0x0 0x40000000>;
         };
};
```

如下所示，需要更改 VCU 解码器设备树的节点才能从 PL_DDR 分配存储器。

```
decoder: al5d@a0120000 {
            compatible = "al, al5d-1.0", "al, al5d";
            interrupt-parent = <&gic>;
            interrupts = <0 96 4>;
            reg = <0x0 0xa0120000 0x0 0x10000>;
           xlnx, dedicated-mem = <&plmem_vcu_dec>;
};
```

用于运行转码的 GStreamer 流水线 （解码到编码）

```
gst-launch-1.0 filesrc location=avc_4k_60fps.mp4! qtdemux ! h264parse ! omxh264dec !
queue max-size-bytes=0 ! v4l2videoxconvert output-io-mode=5 capture-io-mode=4
disable-passthrough=1 ! queue max-size-bytes=0 ! omxh264enc ! filesink
location=file.avc
```

核的约束

Vivado Design Suite 的核代提供了必要的 XDC 约束。

所需约束

本节不适用于此 IP 核。

器件、封装和速度等级选择

本节不适用于此 IP 核。

时钟频率

对速度等级没有限制。所有速度等级均支持最大操作频率。

时钟管理

本节不适用于此 IP 核。

时钟布局

本节不适用于此 IP 核。

bank 分配

本节不适用于此 IP 核。

收发器布局

本节不适用于此 IP 核。

I/O 标准与布局

本节不适用于此 IP 核。

综合与实现

如需了解有关综合与实现的详情，请参阅 《Vivado Design Suite 用户指南：采用 IP 进行设计》(UG896) [参照 2]。

XILINX

第 12 章

应用软件开发

简介

视频编解码器单元 (VCU) 软件堆栈具有软件开发者可在多个抽象层级进行编程的分层架构，如图 12-1 所示。从高层到低层的应用接口如下所列：

- GStreamer
- OpenMAX Integration Layer
- VCU 控制软件

GStreamer 是一个跨平台的开源多媒体框架，提供可集成多个多媒体组件并创建流水线的基础架构。GStreamer 框架可在 OpenMAX™ Integration Layer API 支持的 GStreamer 版本 1.12.2. [参照 20] 中实现。

OpenMAX Integration Layer API [参照 19] 对免版税的标准化媒体组件接口进行了定义，使开发者和平台提供商能够与在硬件或软件中执行的多媒体编解码器进行集成和通信。

VCU Control Software 是 VCU 应用开发者可见的最低级软件。所有 VCU 应用必须直接或间接地使用 Xilinx 提供的 VCU 控制软件。 VCU Control Software 包括定制内核模块、定制用户空间库以及 AL_Encode 和 AL_Decode 应用。OpenMAX IL (OMX) 层集成在 VCU Control Software 的顶层。

用户应用可以使用最适合其要求的 VCU 软件堆栈的一层或多层。

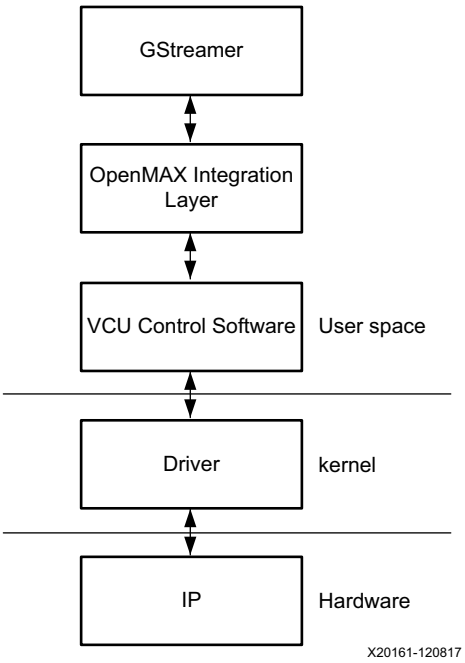

图 12-1：VCU 软件堆栈

软件要求

使用 VCU 的所有软件先决条件都包含在赛灵思软件开发套件 (SDK) 版本中的赛灵思 PetaLinux 中。请参阅嵌入式设计中心 - PetaLinux 工具页面底部的发行说明链接或下载页面的发行说明链接。

使用 VCU 的应用软件编写在以下库和模块的上端，如表 12-1 所示。

表 12-1：应用软件

软件	版本	源
GStreamer 插件		https://github.com/Xilinx/gst-plugins-good/commits/master-rel-1.12.2 https://github.com/Xilinx/gst-omx/tree/xilinx-master https://github.com/Xilinx/gst-plugins-good/tree/master-rel-1.12.2 https://github.com/Xilinx/gst-plugins-bad/tree/master-rel-1.12.2 https://github.com/xilinx/gst-plugins-base/tree/master-rel-1.12.2
软件 Gstreamer 库	1.12.2	https://gstreamer.freedesktop.org/
OpenMAX™ Integration Layer API	1.1.2	https://github.com/Xilinx/vcu-omx-il
VCU 控制软件	1.0.41	https://github.com/Xilinx/vcu-ctrl-sw
VCU 固件	1.0.0	https://github.com/Xilinx/vcu-firmware
VCU 内核模块		https://github.com/Xilinx/vcu-modules

表 12-1：应用软件 （续）

软件	版本	源
VCU 配方文件		https://github.com/Xilinx/meta-xilinx/tree/master/meta-xilinx-bsp/recipes-multimedia/vcu https://github.com/Xilinx/meta-petalinux/tree/master/recipes-multimedia/gstreamer
GStreamer 配方文件		https://github.com/Xilinx/meta-petalinux/tree/master/recipes-multimedia/gstreamer

编码器功能

VCU 支持多标准视频编码，如表 12-2 表所示。

表 12-2：编码器功能

视频编码参数	H.265 (HEVC)	H.264 (AVC)
Profiles	Main Main Intra Main10 Main10 Intra Main 4:2:2 10 Main 4:2:2 10 Intra	Baseline Main High High10 High 4:2:2 High10 Intra High 4:2:2 Intra
Levels	高达 5.1 High Tier	高达 5.2
Resolution and Frame Rate(1)(2)	3840×2160p60 3840×2160p30 1920×1080p60 1920×1080p30 1280×720p60 1280×720p30	3840×2160p60 3840×2160p30 1920×1080p60 1920×1080p30 1280×720p60 1280×720p30
Bit Depth		
GStreamer	8 位、 10 位	8 位、 10 位
OMX	8 位、 10 位	8 位、 10 位
VCU Control Software	8 位、 10 位	8 位、 10 位
Chroma Format		
GStreamer	4:2:0、 4:2:0	4:2:0、 4:2:0
OMX	4:2:0、 4:2:0	4:2:0、 4:2:0
VCU Control Software	4:2:0、 4:2:0	4:2:0、 4:2:0
Slices Types	I、 P、 B	I、 P、 B
Bit Rate	受级别和配置信息的限制	受级别和配置信息的限制

注释：

1. H.265 (HEVC) 最小图像分辨率为 128×128。
2. H.264 (AVC) 最小图像分辨率为 80×96。

层级和级别如表 12-3 所示。

表 12-3：层级与级要求

级别	Max Luma 图像尺寸	最大 CPB 尺寸		每张图像的最大 slice 区段数	最大块行数	最大块列数
		主层	高层			
1	36,864	350	-	16	1	1
2	122,880	1,500	-	16	1	1
2.1	245,760	3,000	-	20	1	1
3	552,960	6,000	-	30	2	2
3.1	983,040	10,000	-	40	3	3
4	2,228,224	12,000	30,000	75	5	5
4.1	2,228,224	20,000	50,000	75	5	5
5	8,912,896	25,000	100,000	200	11	10
5.1	8,912,896	40,000	160,000	200	11	10
5.2	8,912,896	60,000	240,000	200	11	10
6	35,651,584	60,000	240,000	600	22	20
6.1	35,651,584	120,000	480,000	600	22	20
6.2	35,651,584	240,000	800,000	600	22	20

针对 Subframe/Normal 时延模式下分辨率为 1080p/4K 时支持的最大 slice 数。

模式	编码	FullHD	4K
Normal （正常）	AVC	68	135
	HEVC	34	22
Low latency （低时延）	AVC	32	32
	HEVC	32	22

GStreamer 编码参数

GStreamer 编码参数如表 12-4。

表 12-4：GStreamer 编码参数

VCU 参数	GStreamer 属性	描述
Rate Control Mode	control-rate	可用比特率控制模式、常数比特率（CBR）和可变比特率 （VBR）编码 0 = 禁用 （CONST_QP、常数 QP） 1 = 变量 (VBR) 2 = 常数 (CBR) 3 = 变量跳帧 （不支持） 4 = 常量跳帧 （不支持） 2130706433 = 低时延 默认值：2 **注释：**变量跳帧和常量跳帧来自 gstreamer 默认插件（VCU 不支持）。(1)、(2)、(3)、(4)、(5)
Target Bit Rate	target-bitrate	目标比特率 (Kbps) 默认值：64
Maximum Bit Rate	max-bitrate	最大比特率 (Kbps)，用于 VBR 速率控制模式。默认值：目标比特率。最大比特率值应始终 ≥ 目标比特率 默认值：64000
Profile	Through caps	支持的相应编解码器的配置信息如表 12-2 中所示。 默认值：HEVC_MAIN
Level	Through caps	支持的相应编解码器的级如表 12-2 中所示。 默认值：51
Tier	Through caps	编解码器支持的层级如表 12-2 中所示。 默认值：MAIN_TIER
Slice QP/I-frame QP	quant-i-frames	CONST_QP 模式下 I 帧的量化参数，也用作其他速率控制模式下的初始 QP。范围 0–51。 默认值：30
P-frame QP	quant-p-frames	CONST_QP 模式下 P 帧的量化参数。范围 0–51。 默认值：30
B-frame QP	quant-b-frames	CONST_QP 模式中 B 帧的量化参数。范围 0–51。 默认值：30
GOP Length	gop-length	两个连续帧内帧之间的距离。指定 0 到 1,000 之间的整数值。值 0 和 1 对应于仅帧内编码 默认值：30
Number of B-frames	b-frames	两个连续 P 帧之间的 B 帧数。仅在 gop-mode 为 basic 或 pyramidal 时使用。 范围： 0-4 （适用于 gop-mode = basic 时） 3、 5 或 7 （适用于 gop-mode = pyramidal 时） 当 latency-mode = low-latency & reduced-latency 时，b 帧应设置为零，因为在设置 b 帧时不能对任何帧进行重新排序。 默认值：0

表 12-4：GStreamer 编码参数 （续）

VCU 参数	GStreamer 属性	描述
Number of Slices	num-slices	指定每帧使用的 slice 数。每个 slice 包含一个或多个完整的 LCU 行或行，并尽可能有规律地分布在帧上。最小值为 1。 支持的最大值如下： 在 latency-mode = low-latency 的情况下 • H.264(AVC)：32 • H.265 (HEVC)：22 由于开销，在低时延模式下 slice 数超过 16 个不会带来太多好处 在 latency-mode = normal 的情况下 • H.264(AVC)：picture_height/16 • H.265(HEVC)：最低 picture_height/32 在正常模式下可以超过支持的最大值，但当编码器使用多个核时，对 HEVC 会有一些限制 当 HEVC 编码器使用多个核 （即分辨率超过 1080p60 的任何核）时，最大 slice 数将受“最大块行数 (MaxTileRows)”的限制。 请参阅表 12-3。 默认值：1
Minimum QP	min-qp	编码会话中允许的最小 QP 值。 范围 0–51。 默认值：10
Maximum QP	max-qp	编码会话中允许的最大 QP 值。 范围 0–51。 默认值：51
GOP Configuration	gop-mode	指定图像组配置 0 = basic （内部周期为 30 时的 IPPP 模式） 1 = pyramidal (hierarchical) （带分层级 B 帧结构的高级 GOP 模式。适用于 B=3、 5 和 7） 2 = adaptive （自适应 B 帧） 3 = low_delay_p (IPPPPP...) 4 = low_delay_b (IBBBBB...) • 支持开放式 GOP 和封闭式 GOP 结构。 • 当 Gop.FreqIDR≠Gop.Length 且 B 帧≠ 0 时，可以打开 GOP 结构。 • 对于金字塔形 gop 模式，编码器仅使用封闭式 GOP。 • 没有 B 帧，则仅使用封闭式 GOP。 • 对于 B 帧，gop-mode=pyramidal，则使用封闭式 GOP。 • 对于 B 帧，gop-mode=default & periodicity-idr = 0，则使用开放式 GOP。 • 对于 B 帧，gop-mode=default & IDR_freq= GOP。Length，则使用封闭式 GOP。 （对于不同的 IDR_freq，您可以混合使用开放式 GOP 和封闭式 GOP） 默认值：0

表 12-4：GStreamer 编码参数 （续）

VCU 参数	GStreamer 属性	描述
Gradual Decoder Refresh	gdr-mode	指定当 gop-mode = low_delay_p 时应使用哪种渐近解码器 (Gradual Decoder Refresh) 刷新方案 0 = 禁用 1 =vertical （使用垂直条从左到右移动逐渐刷新） 2 = horizontal （使用水平条从上到下移动逐渐刷新） 默认值：0
QP Control Mode	qp-mode	VCU 编码器使用的 QP 控制模式 0 = uniform （对帧的所有编码单位使用一个 QP） 1 = roi （根据每个帧上定义的感兴趣区域调整 QP。必须提供 ROI 元数据） 2 = auto (让 VCU 编码器根据其内容更改每个编码单元的 QP) 默认值：0
Filler data	filler-data	在 CBR 速率控制模式下启用/禁用填充数据添加功能。布尔：true 或 false 默认值：True
Entropy Mode	entropy-mode	指定 H.264 (AVC) 编码过程的熵模式 0 = CAVLC 1 = CABAC 默认值：1
Deblocking Filter	loop-filter-mode	启用/禁用去块效应滤波器选项 0 = 启用 1 = 禁用 2 =禁用 slice 边界 （从过滤中排除 slice 边界） 默认值：0
IDR picture frequency	periodicity-idr	指定连续瞬时解码器刷新 (IDR) 图像之间的帧数。periodicity-idr 属性以前被称为 gop-freq-idr。 允许的值：<正值>或 -1 以禁用 IDR 插入 默认值：1
Initial Removal Delay	initial-delay	象在 HRD 模型中指定的那样指定初始删除延迟（以毫秒为单位）。当 control-rate = disable 时不使用 默认值：1500
Coded Picture buffer size	cpb-size	象在 HRD 模型中指定的那样指定编码图像缓存 (CPB) （以毫秒为单位）。当 control-rate = disable 时不使用 默认值：3000
Dependent slice	dependent-slice	指定附加 slice 是否依赖于多个 slice 编码会话中的其他 slice 片段或常规 slice。仅用于 H.265 (HEVC) 编码。布尔：true 或 false 默认值：False
Target slice size	slice-size	如果设置为 0，则 slice 由 num-slices 参数定义，否则它将指定目标 slice 的大小 （以字节为单位）。用于自动将比特流分割成大小大致相等的 slice。 范围 0-65,535。 默认值：0

表 12-4：GStreamer 编码参数 （续）

VCU 参数	GStreamer 属性	描述
Scaling Matrix	scaling-list	指定缩放列表模式 0 = flat 1 = 默认值 默认值：1
PrefetchLevel2	prefetch-buffer	在编码过程中启用/禁用 L2Cache 高速缓存 默认值：false
Vertical Search Range	low-bandwidth	指定低带宽模式。减少用于 P 帧运动估计的垂直搜索范围。true 或 false。 默认值： H.264(AVC)：16 H.265(HEVC)：32
Slice Height	slice-height	指定上游元素的输入缓存高度对齐 （如果有）。 默认值：1
Aspect-Ratio	aspect-ratio	选择要在 SPS/VUI 中写入的视频序列的显示宽高比。 0 =自动 - 4:3 用于标清视频，16:9 用于高清视频，未指定用于未知格式 1 = aspect_ratio_4_3 （宽比高 4:3） 2 = aspect_ratio_16_9 （宽比高 16:9） 3 = none （流中不存在宽高比信息） 默认值：0
Latency Mode	latency-mode	指定编码器的时延模式 0 = normal （为下一个元素提供帧级输出） 1 = low-latency （为下一个元素提供 slice 级输出）） 默认值：0 **注释：**当速率控制为 CBR/VBR 时，不建议使用低时延模式。
Constrained Intra Prediction	constrained-intra-prediction	如果启用，则预测仅使用来自使用帧内预测模式编码的相邻译码块的残余数据和解码样本。布尔：true 或 false。 默认值：False
Long term Ref picture	long-term-ref	如果启用，编码器接受动态插入和使用来自上游元素的长期参考图像事件 布尔：true 或 false
Long term picture frequency	long-term-freq	编码过程中长期参考图像标记的周期性帧中的单位、两个连续长期参考图像之间的距离
Dual pass encoding	look-ahead	在第二次传递编码之前处理的帧数。如果小于 2，则禁用双通道编码

注释：

1. 速率控制仅在 MCU FW 中处理，并且在速率控制过程中 （在软件 API 或 FPGA 信号中）没有信号。
2. **CBR**：CBR 的目标是 （在一个或几个 GOP 级上）达到平均目标比特率并遵守“HRD”模型，即避免解码器缓存溢出和下溢。在 CBR 模式中，单个比特率值定义目标流比特率和“输出”/“传输” （“漏桶”）比特率。（参考解码器缓存参数是 CPBSize 和 Initial Delay）。
3. **VBR**：使用 VBR 时，允许编码器缓存下溢 （为空），并且最大比特率 （用于缓存模型的传输比特率）可以高于目标比特率。因此，VBR 放宽了缓存约束，允许降低简单内容的比特率，并且可以通过在复杂帧上允许更多比特来提高质量。
4. CBR 和 VBR 都使用来自 HW 的帧级统计来更新下一帧的初始 QP （速率控制可以与 QP 控制相结合， QP 控制可以在块级调整 QP 以提升主观质量）。
5. LOW_LATENCY 速率控制 （也称为硬件速率控制）以块精度计算块级的 QP，以精确的方式达到每帧的目标比特流大小，因此特别有助于支持低时延流水线。

解码器功能

表 12-5：Decoder Features

视频编码参数	H.265 (HEVC)	H.264 (AVC)
Profiles	Main Main Intra Main10 Main10 Intra Main 4:2:2 10 Main 4:2:2 10 Intra	Baseline （ASO/RS 除外） Main High High10 High 4:2:2 High10 Intra High 4:2:2 Intra
Levels	高达 5.1 High Tier	高达 5.2
Resolutions	3840×2160p60 3840×2160p30 1920×1080p60 1920×1080p30 1280×720p60 1280×720p30	3840×2160p60 3840×2160p30 1920×1080p60 1920×1080p30 1280×720p60 1280×720p30
Chroma Format	4:2:0、 4:2:0	4:2:0、 4:2:0
Bit Depth	8 位、 10 位	8 位、 10 位

GStreamer 解码参数

表 12-6：GStreamer 解码参数

参数	GStreamer 属性	描述
Entropy Buffers	internal-entropy-buffers	指定解码器内部熵缓存，用于平滑熵解码性能。指定 2 和 16 之间的整数值。 默认值：5。增加 internal-entropy-buffers 会增加解码器的存储器占用量。
Latency	latency-mode	指定解码器的时延模式。 0 = 默认值 1 = reduced-latency （低参考 DPB 模式） 2 = 低时延

解码器最大比特率测试

用来测量解码器产生 30 fp 的最大比特率的流水线。

表 12-7：解码器最大比特率测试

Codec	IPPP	IPBB	Intra-only
H.264 (AVC)	350 Mbps	335 Mbps	400 Mbps
H.265 (HEVC)	275 Mbps	275 Mbps	270 Mbps

用于测量的示例流水线：

```
gst-launch-1.0 filesrc location="/run/test_file.mp4 " ! qtdemux ! h264parse !
omxh264dec internal-entropy-buffers=9 ! queue max-size-bytes=0 ! fpsdisplaysink
```

```
name=fpssink text-overlay=false video-sink=kmssink sync=true
fps-update-interval=3000 -v
```

准备 PetaLinux 以运行 VCU 应用

在 VCU 应用能够在 PetaLinux 中成功运行之前，可能需要更改服务质量 (QoS) 设置以更改可读取/写入数据的任何 AFI 端口的优先级，并且可能有助于解决与带宽相关的问题。必须修改将 AXI 端口的 QoS 设置和指令发布功能更改为 VCU 解码器 (M_AXI_DEC0 和 M_AXI_DEC1)，以避免在解码和显示用例中出现显示端口信号拥挤情况。如果未修改 QoS 设置， DisplayPort/HDMI 传输可能欠载，在视频播放期间间歇性地产生绿色或黑色帧。不增加命令发布能力可能会限制挑战性设置 （如 4kp60）的帧速率。

有关连接到解码器的端口，请参阅第 10 章"用核设计"。了解下面提到的地址的来源会让您能够通过使用备用的 AXI 连接对以下命令进行调整。有关 AXI 控制寄存器地址和设置，请参阅《Zynq UltraScale+ MPSoC 寄存器参考》(UG1087) [参照 14]。

1. 使用 PetaLinux 预先构建的映像启动电路板。
2. 使用用户名登录：root 和密码：root
3. 设置连接到 M_AXI_DEC0 的端口的读写 QoS
   - 将 S_AXI_HP0_FPD RDQoS (AFIFM) 寄存器设置为 LOW_PRIORITY

     ```
     devmem 0xFD380008 w 0x3
     ```

   - 将 S_AXI_HP0_FPD WRQoS (AFIFM) 寄存器设置为 LOW_PRIORITY

     ```
     devmem 0xFD38001C w 0x3
     ```

4. 设置连接到 M_AXI_DEC1 的端口的读写 QoS
   - 将 S_AXI_HP3_FPD RDQoS (AFIFM) 寄存器设置为 LOW_PRIORITY

     ```
     devmem 0xFD3B0008 w 0x3
     ```

   - 将 S_AXI_HP3_FPD WRQoS (AFIFM) 寄存器设置为 LOW_PRIORITY

     ```
     devmem 0xFD3B001C w 0x3
     ```

5. 提升连接到 M_AXI_DEC0 的端口的读写发布能力。默认情况下，一次最多可能需要四个请求，并且增加发布功能可能会使端口始终忙于处理队列中的某些请求。
   - 设置 S_AXI_HP0_FPD RDISSUE (AFIFM) 寄存器以允许 16 个命令

     ```
     devmem 0xFD380004 w 0xF
     ```

   - 设置 S_AXI_HP0_FPD WRISSUE (AFIFM) 寄存器以允许 16 个命令

     ```
     devmem 0xFD380018 w 0xF
     ```

6. 提高连接到 M_AXI_DEC1 的端口的读写发布能力
   - 设置 S_AXI_HP3_FPD RDISSUE (AFIFM) 寄存器以允许 16 个命令

     ```
     devmem 0xFD3B0004 w 0xF
     ```

   - 设置 S_AXI_HP3_FPD WRISSUE (AFIFM) 寄存器以允许 16 个命令

     ```
     devmem 0xFD3B0018 w 0xF
     ```

现在上面提到的 GStreamer、 OMX 和赛灵思控制软件流水线就可以在电路板上运行了。

集成 VCU 与 GStreamer 补丁

1. 解压缩 PetaLinux BSP。
2. 在 project-spec/meta-user 文件夹中创建“recipe-multimedia”文件夹。

```
cd project-spec/meta-user
mkdir recipe-multimedia
```

3. 对于 gstreamer 补丁，请按照以下步骤操作：
   a. 在 recipe-multimedia 文件夹中创建“gstreamer”目录。

```
cd project-spec/meta-user/recipe-multimedia
mkdir gstreamer
```

   b. gstreamer 有 5 个不同的、用来下载代码并进行编译的配方文件。
      - gstreamer1.0_%.bbappend
      - gstreamer1.0-omx_%.bbappend
      - gstreamer1.0-plugins-bad_%.bbappend
      - gstreamer1.0-plugins-base_%.bbappend
      - gstreamer1.0-plugins-good_%.bbappend

   c. 根据它所属的 gstreamer 包所附带的补丁，需要创建该包的 bbappend 文件，以便在最新的源代码上应用和编译这些补丁。

      例如，如果修补程序适用于 gst-omx，按照以下步骤操作：

      - 在 recipe-multimedia/gstreamer 文件夹中创建“gstreamer1.0-omx”目录。

```
mkdir gstreamer1.0-omx
```

      - 复制“gstreamer1.0-omx”目录中的 gst-omx 补丁。

```
cp test1.patch recipe-multimedia/gstreamer/gstreamer1.0-omx
cp test2.patch recipe-multimedia/gstreamer/gstreamer1.0-omx
```

      - 在 recipe-multimedia/gstreamer 文件夹中创建“gstreamer1.0-omx_%.bbappend”文件。
      - 在“gstreamer1.0-omx_%.bbappend”文件中添加以下行：

```
FILESEXTRAPATHS_prepend: = "${THISDIR}/gstreamer1.0-omx:"
SRC_URI_append = " \
file://test1.patch \
 file://test2.patch \
"
```

      注释： 为其他 gstreamer 包创建类似的 bbappend 文件和文件夹，以在 PetaLinux Build 中集成任一定制补丁。

4. 对于 VCU 补丁，请按照以下步骤操作：
   a. 在 recipe-multimedia 文件夹中创建“vcu”目录。

```
cd project-spec/meta-user/recipe-multimedia
mkdir vcu
```

   b. VCU 有 4 个不同的、用来下载代码并进行编译的配方文件。
      - kernel-module-vcu_%.bbappend
      - vcu-firmware_%.bbappend
      - libvcu-xlnx_%.bbappend
      - libomxil-xlnx_%.bbappend

   c. 根据它所属的 VCU 源代码附带的补丁，需要创建该代码库的 bbappend 文件，以便在最新的源代码上应用和编译这些补丁。

例如，如果修补程序适用于 vcu 驱动，按照以下步骤操作：

- 在 recipe-multimedia/vcu 文件夹中创建“kernel-module-vcu”目录。

```
mkdir kernel-module-vcu
```

- 在“kernel-module-vcu”目录中复制 vcu 驱动补丁。

```
cp test1.patch recipe-multimedia/vcu/kernel-module-vcu
cp test2.patch recipe-multimedia/vcu/kernel-module-vcu
```

- 在 recipe-multimedia/vcu 文件夹中创建“kernel-module-vcu_%.bbappend”文件。
- 在“kernel-module-vcu_%.bbappend”文件中添加以下行：

```
FILESEXTRAPATHS_prepend: = "${THISDIR}/ kernel-module-vcu:"
SRC_URI_append = " \
file://test1.patch \
 file://test2.patch \
"
```

注释：为其他 VCU 组件创建类似的 bbappend 文件和文件夹，以在 PetaLinux Build 中集成任一定制补丁。

5. 按照 PetaLinux 构建步骤生成更新的二进制文件。

注释：如果是未使用 PetaLinux 进行编译的用户，确保查看设置 GStreamer 所需的其他文件的配方。例如，您必须在根文件系统中包含以下文件：/etc/xdg/gstomx.conf。该文件指示 gst-omx 在哪里可以找到 OMX 集成层库 - libOMX.allegro.core.so.1。

开箱即用的 VCU 示例

下面列出的是支持的、开箱即用的 VCU 示例。默认的桌面应用显示的是与 VCU-解码 → 显示用例对应的两个 VCU 示例（4K AVC 解码和 4K HEVC 解码）的图标，更多细节在示例-1 中会提到。

- VCU-解码 → 显示：将在 DP 监视器上解码并显示 AVC/HEVC 编码容器流。该操作支持 aac/vorbis 视频音频解码。
- USB 摄像头 → 编码 → 解码→显示：使用 VCU 从摄像头和编码/解码中捕获原始视频帧，并通过 DP 监视器进行显示。
- USB 摄像头 → 解码 → 显示：从摄像头捕获编码的视频内容，使用 VCU 捕获解码，并通过 DP 监视器进行显示。
- 转码→到文件：将输入 AVC/HEVC 编码比特流转码为 HEVC/AVC 格式并存储在磁盘上。
- **转码 → 使用以太网流出...**流入 → 解码 → 显示：将输入 HEVC/AVC 编码比特流转码为HEVC/AVC 格式，并通过 RTP/UDP 流式传输到另一个电路板；第二个电路板通过 VCU 接收数据解码，通过 DP 监视器进行显示。
- USB 摄像头 → 音频/视频编码 → 文件：视频录制用例。
- USB 摄像头 → 编码→流出...流入 → 解码 → 显示：视频会议用例。

验证示例

以下部分描述了如何验证该示例。

使用 PetaLinux BSP 启动电路板 (ZCU106/ZCU104)

赛灵思 PetaLinux 板级支持包 (BSP) 是一个运行示例用户设计的 Linux 操作系统。

1. 确保电路板已连接到以太网，以从赛灵思 Web 服务器下载示例视频内容。
2. 如果电路板连接到专用网络，则在“/home/root/”中导出代理设置。 bashrc 文件如下：

```
create/open a bashrc file using "vi ~/. bashrc"
```

3. 将以下行插入 bashrc 文件：

```
export http_proxy="<private network proxy address>"
export https_proxy="<private network proxy address>"
```

如果电路板未连接到 Internet，则可以使用主机下载压缩视频文件，并将输入文件复制到“/home/root/”文件夹。使用以下命令在主机 Linux 机器上下载内容。

4. 下载 AVC 示例文件：

```
wget petalinux.xilinx.com/sswreleases/video-files/
bbb_sunflower_2160p_30fps_normal_avc.mp4
```

5. 下载 HEVC 示例文件：

```
wget petalinux.xilinx.com/sswreleases/video-files/
bbb_sunflower_2160p_30fps_normal_hevc.mkv
```

示例 1：（解码 → 显示）

Matchbox Desktop 具有以下两个应用：

- 4K AVC 解码

  对于 4K AVC 解码，单击应用以下载示例 AVC/AAC 编码比特流并运行 VCU 示例 1 （解码 → 显示）。

  如果“/home/root”中已有示例 AVC 内容，则会解码 → 显示内容。

- 4K HEVC 解码

  对于 4K AVC 解码，单击应用以下载示例 HEVC/Vorbis 编码比特流并运行 VCU 示例 1 （解码 → 显示）。

  - 使用 -h 选项运行脚本会显示所有可能的选项。

```
vcu-demo-decode-display.sh -i /home/root/bbb_sunflower_2160p_30fps_normal_avc.mp4
-c avc -a aac
vcu-demo-decode-display.sh -i /home/root/bbb_sunflower_2160p_30fps_normal_hevc.mkv
-c hevc -a vorbis
```

  - 运行以下命令播放 youtube 视频，确保以太网连接到电路板

```
vcu-demo-decode-display.sh -u "youtube-URL"
```

  - 运行以下命令以获取帮助

```
vcu-demo-decode-display.sh -h
```

示例 2：（USB 摄像头 → 编码 → 解码 → 显示）

1. 将 USB 摄像头连接到电路板 （已认证的摄像头：Logitech HD 摄像头， C920）。
2. 运行以下命令 （用于 USB 视频捕获串行流水线）

```
vcu-demo-camera-encode-decode-display.sh -s 640x480
```

3. 对于有适用设置的音频的 USB 视频捕获串行流水线，运行以下命令：

  - ALSA 库 API：

```
vcu-demo-camera-encode-decode-display.sh -s 640x480 -a aac --use-alsasrc -i "hw:1,0"
--use-alsasink --audio-output "hw:0,0"
```

  - 脉冲库 API

```
  vcu-demo-camera-encode-decode-display.sh -s 640x480 -a aac -i "alsa_input.usb-
046d_HD_Pro_Webcam_C920_C06272EF-02. analog-stereo" \
```

```
--use-pulsesrc --use-pulsesink --audio-output "alsa_output.
platform-fd4a0000.zynqmp-  display_zynqmp_dp_snd_card. analog-stereo"
```

注释：有关如何选择要传递给 -i 和 --- audio-output 选项的参数值，请查阅第 10.2 节。

- Gstreamer Autoaudiosrc 与 Autoaudiosink 插件

```
vcu-demo-camera-encode-decode-display.sh -s 640x480 -a aac
```

注释：在上面使用 gstreamer 自动插件的示例中， gstreamer 会选择用来自动捕获和回放的 API 和器件。通常，它会尝试使用由 pulseaudio 服务器启动时枚举的默认器件，或者如 alsa 配置文件中所述进行操作。因此，它有时可能不会选择要使用的器件，在这种情况下，可以使用上述脚本实参。

示例 3：(USB 摄像头 → 解码 → 显示）

1. 将 USB 摄像头连接到电路板 （已认证的摄像头：Logitech 高清摄像头， C920）。
   - 运行以下命令 （适用于 USB 视频解码显示流水线）

   ```
   vcu-demo-camera-decode-display.sh -s 1920x1080
   ```

   - 运行以下命令 （适用于 USB 视频解码视频流水线）
     - ALSA 库 API

     ```
     vcu-demo-camera-decode-display.sh -a aac --use-alsasrc -i "hw:1,0" --use-alsasink
     --audio-output "hw:0,0"
     ```

     - 脉冲库 API

     ```
     vcu-demo-camera-decode-display.sh -s 640x480 -a aac -i
     "alsa_input.usb-046d_HD_Pro_Webcam_C920_C06272EF-02. analog-stereo" \
      --use-pulsesrc --use-pulsesink --audio-output "alsa_output.
     platform-fd4a0000.zynqmp-display_zynqmp_dp_snd_card. analog-stereo"
     ```

     注释：有关如何选择要传递给 -i 和 ---audio-output 选项的参数值的“如何引导”，请参见第 10.2 节。

     - Gstreamer Autoaudiosrc 与 Autoaudiosink 插件

     ```
     vcu-demo-camera-decode-display.sh -a aac
     ```

重要提示：示例 2 和 3 的分辨率必须根据 USB 摄像头功能来设置。

- 可以通过“v4l2-ctl --list-formats-ext”找到各个功能。
- 如果 V4lutils 没有安装在预建的图像中，则需要通过 dnf 或者重建 petalinux 图像 （包括 v4lutils）来安装。

示例 4：（转码 → 到文件）

此示例需要输入示例文件。如果已经执行了示例-1，则示例视频存储在“/home/root”文件夹中。

1. 使用以下命令将示例 avc 文件转码为 hevc 文件。

   ```
   vcu-demo-transcode-to-file.sh -i /home/root/
   bbb_sunflower_2160p_30fps_normal_avc.mp4 -c avc -a aac -o /home/root/transcode.mkv
   ```

2. 使用示例 1 查看 Display 上的转码文件，示例命令如下所述：

   ```
   vcu-demo-decode-display.sh -i /home/root/transcode.mkv -c hevc -a vorbis
   --use-alsasink --audio-output "hw:0"
   ```

示例 5：(转码 → 使用以太网流出...流入 → 解码 → 显示）

此示例需要两块电路板。电路板 1 用于转码和流出 （作为服务器），电路板 2 用于流入和解码目的 （作为客户端）或者主机上的 VLC 播放器可用作客户端而不是用作电路板 2。

1. 确保将输入视频文件复制到电路板-1 进行流式传输。
2. 使用以太网电缆背对背连接两块电路板，或确保两块电路板都连接到同一个以太网集线器。
   a. 如果将 VLC 播放器用作客户端，请确保主机和服务器 （电路板 1）连接到同一个以太网集线器或使用以太网电缆使其背对背连接。
   - 客户端设置

     如果电路板 2 被用作客户端，并用以太网电缆将电路板背对背连接，则设置客户端 IP 地址并在电路板 2 上执行流入 → 解码示例。

```
ifconfig eth0 192.168.0.2
```

注释：如果两块电路板已经连接到同一个 LAN，则不需要设置 ip。

```
vcu-demo-streamin-decode-display.sh -c avc
```

注释：这意味着客户端正在从服务器接收 AVC 比特流。如果服务器正在发送 HEVC 比特流，使用 -c hevc。

3. 如果将 VLC 播放器用作客户端，则将主机 IP 地址设置为 192.168.0.2 （如果它是通过以太网电缆直接连接到电路板的）。

注释：如果两块电路板已经连接到同一个 LAN，则不需要设置 ip。

4. 在设有以下内容的主机上创建 test.sdp 文件 （在 test.sdp 中为下面的每个项添加单独的行）并
   - 在主机上播放 test.sdp。

```
v=0 c=IN IP4 <Client machine IP address>
m=video 50000 RTP/AVP 96
a=rtpmap:96 H264/90000
a=framerate=30
```

   - VLC 播放器设置故障排除：

     IP4 是客户端 IP 地址

     H264/H265 基于客户端上接收的编解码器类型进行使用

     如果未收到发往 VLC 的数据包，则应关闭主机中的防火墙。

5. 设置服务器 IP 并在电路板 1上执行 Transcode→stream-out 示例。

```
ifconfig eth0 192.168.0.1
```

注释：如果两块电路板已经连接到同一个 LAN，则不需要设置 IP。

```
vcu-demo-transcode-to-streamout.sh -i /home/root/
bbb_sunflower_2160p_30fps_normal_hevc.mkv -c hevc -b 5000 -a <Client Machine/Board
IP address>
```

示例 6：(摄像头音频/视频->编码 → 文件 A/V 记录）

1. 将 USB 摄像头连接到电路板 （已认证的摄像头：Logitech HD 摄像头 C920）。
2. 执行以下命令进行 USB 视频录制。

```
vcu-demo-camera-encode-file.sh -a aac -n 1000 --use-alsasrc -i "hw:1"
```

输出文件 `camera_output.ts` 随即会在当前目录上生成 （该目录将在 mpegts 容器中记录音频/视频文件）。

该文件可以在 VLC 等媒体播放器上播放。

示例-7：(摄像头音频/视频 → 使用以太网流出...流入→解码→带音频显示）

此示例需要两块电路板，电路板-1 用于编码来自摄像头的实时 A/V 馈送和流出 （作为服务器），而电路板 2 用于流入和解码目的以播放接收到的音频/视频流 （作为客户端）。

1. 使用以太网电缆背对背连接两块电路板，或确保两块电路板都连接到同一个以太网集线器。
2. 设置服务器 IP 地址并在电路板 2 上执行-→解码器示例。

   ```
   ifconfig eth0 192.168.0.2
   ```

   注释：如果两块电路板已经连接到同一个 LAN，则不需要设置 ip。

   ```
   vcu-demo-streamin-decode-display.sh -c avc --audio-type aac
   ```

   注释：这意味着客户端正在从服务器接收 AVC 比特流。如果服务器正在发送 HEVC 比特流，使用 -c hevc。)。
3. 设置服务器 IP 并在电路板-1 上运行摄像头->编码->流出。

   ```
   ifconfig eth0 192.168.0.1 (Note: setting of ip is not required if the boards already
   connected to same LAN)
   vcu-demo-camera-encode-streamout.sh --audio-type aac --use-alsasrc -i "hw:1"
   --address 192.168.0.2
   ```

确定脉冲音频和 ALSA 器件名称

可以分别使用 `arecord` 和 `aplay` 实用程序找到音频源和播放器件 （提供--audio-output）的音频器件名称 （以提供 -i 选项）。

- ALSA 声音捕获器名称
  - arecord -l

  ```
  **** List of CAPTURE Hardware Devices ****
  card 1: C920 [HD Pro Webcam C920], device 0: USB Audio [USB Audio]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
  In this example, card number of capture device is 1 and device id is 0 hence "hw:1,0"
  can be passed to alsasrc to refer to this capture device using -i option.
  ```

- ALSA 声音播放器名称
  - aplay -l

  ```
  **** List of PLAYBACK Hardware Devices ****
  card 0: monitor [DisplayPort monitor], device 0: (null) xilinx-dp-snd-codec-dai-0 []
  Subdevices: 1/1
  Subdevice #0: subdevice #0
  card 0: monitor [DisplayPort monitor], device 1: (null) xilinx-dp-snd-codec-dai-1 []
  Subdevices: 1/1
  Subdevice #0: subdevice #0
  ```

  在此示例中，卡号“0”用于显示端口通道 0 的回放器件，器件 ID 为 0，因此“hw:0,0”可用于选择显示端口通道 0，作为使用 --audio-output 的回放器件 - 输出选项。

- PulseAudio 声音捕获器名称
  - pactl 列表短消息来源

  ```
  alsa_input.usb-046d_HD_Pro_Webcam_C920_758B5BFF-02.analog-stereo  ...
  ```

  在此示例中，“alsa_input.usb-046d_HD_Pro_Webcam_C920_758B5BFF-02.analog-stereo”是可以通过 -i 选项来选择的音频捕获器的名称。

- PulseAudio 声音回放器名称
  - pactl 列表短消息宿端

  ```
  alsa_output.platform-fd4a0000.zynqmp-display_zynqmp_dp_snd_card. analog-stereo ...
  ```

  在此示例中，“alsa_output.platform-fd4a0000.zynqmp-display_zynqmp_dp_snd_card.analog-stereo”是可以通过 --audio-output 选项来选择的音频播放器的名称。

VCU - 编码器功能描述

VCU 编码器支持动态地更改少量参数，而且很少为每个会话的静态设置。

- Target-bitrate （目标比特率）
- GOP Length （GOP 长度）
- Number of B-Frames （B 帧数）
- Inserting Key Frame （插入关键帧）
- Region of Interest Encoding （感兴趣区域编码）
- Long Term Reference Picture Support （长期参考图像支持）
- Adaptive GOP （自适应 GOP）
- Dual pass encoding （双通道编码）
- SEI Insertion and Extraction （SEI 插入与提取）
- Scene change detection （场景变化检测）
- Interlaced video （隔行扫描视频）
- GDR Intra Refresh （GDR 帧内刷新）

赛灵思 VCU 控制软件应用和 GStreamer 均支持给定帧编号的预定比特率更改。预先构建的 PetaLinux 映像包含“ /usr/bin/”中的 zynqmp_vcu_encode 应用。源代码 GStreamer 应用是 gst-omx repo 的一部分 (https://github.com/Xilinx/gst-omx/tree/xilinx-master/examples/zynqultrascaleplus)。

有关选项，请参阅用法消息。

```
zynqmp_vcu_encode --help
```

初始编码会话应该从动态比特率和动态 B 帧参数在最坏情况下的最大值开始。例如，如果您计划在编码会话期间动态修改 b 帧，则编码会话应以 num-bframes = 4 开头。同理，对于目标比特率，编码会话应该以计划的最大比特率开始。

动态比特率

动态比特率是在编码器处于活动状态时改变编码比特率 （目标比特率）的能力。

要将帧编号 100 号的视频比特率更改为 1 Mbp：

```
zynqmp_vcu_encode -w 3840 -h 2160 -e avc -r 5000 -o /run/op.h264 -i /run/input.yuv
-d BR:100:1000
```

-d 参数的语法是：

<keyword>:<frame number>:<value>

动态 GOP

动态 GOP 是在编码器处于活动状态时改变 gop-length、 B 帧数和强制 IDR 图像的能力。

要将第 100 帧的 gop-length （视频长度）更改为 45：

```
zynqmp_vcu_encode -w 3840 -h 2160 -e avc -g 30 -o /run/op.h264 -i /run/input.yuv -d
GL:100:45
```

要将视频第 20 帧的 B 帧数更改为 2：

```
zynqmp_vcu_encode -w 3840 -h 2160 -e avc -b 4 -o /run/op.h264 -i /run/input.yuv -d
BFrm:20:2
```

要在第 35 帧插入关键帧：

```
zynqmp_vcu_encode -w 3840 -h 2160 -e avc -b 4 -o /run/op.h264 -i /run/input.yuv -d
KF:35
```

感兴趣区域编码

感兴趣区域编码对要编码的视频帧中的区域进行标记，相对于图像背景 （未标记区域）以用户提供的质量 （高、中、低和不关心）进行编码。帧中定义的示例 ROI 如图 12-2 所示。

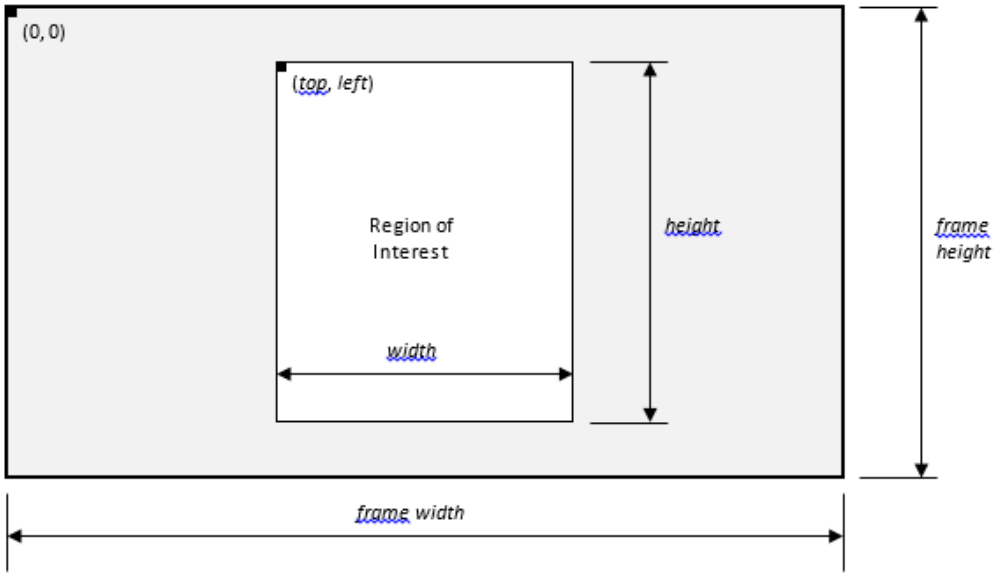

图 12-2：感兴趣区域编码

用户以像素为单位提供感兴趣区域 (ROI) 的位置 (顶部、左侧），以像素为单位提供宽度和高度以及质量指数。支持帧内的多个区域和重叠的 ROI 区域。示例 GStreamer 应用仅添加了一个 ROI 区域，但用户可以将多个 ROI 元数据属性附加到缓存。

输入格式如下：

ROI:<*frame number*>:<*top*>x<*left*>:<*width*>x<*height*>:<*ROI type*>

用来进行编码的视频示例 GStreamer 应用命令行选项，其中包含 ROI 区域附加到 100 处的特定帧编号：

```
zynqmp_vcu_encode -w 3840 -h 2160 -e avc -o /run/op.h264 -i /run/input.yuv -d
ROI:100:1280x360:1220x1500:high
```

ROI-QP

您可以使用 GStreamer 和 Control SW 动态设置 ROI 实时流媒体。

GStreamer：

如示例应用代码所示，使用以下两个 API 设置 ROI 区域和各帧的质量 https://github.com/Xilinx/gst-omx-xlnx/blob/xilinx-master/examples/zynqultrascaleplus/test-vcu-encode.c （行号 #301）。

- gst_buffer_add_video_region_of_interest_meta()

- gst_video_region_of_interest_meta_add_param()

CtrlSW：

Control-SW 导出一些 API，以便为每个帧的编码器提供 ROI 信息。使用下列 API 在实时流中添加 ROI。

- 在编码器创建期间调用 AL_RoiMngr_Create()。
- 为每个帧调用 AL_RoiMngr_Clear()、 AL_Roi_Mngr_AddROI() 和 AL_RoiMngr_FillBuff() 以向编码器提供 ROI 信息。

有关 API 的更多详细信息，请参阅 Doxygen 输出。

长期参考图像

在 GOP 中动态插入长期图像，并根据用户输入使用适用于特定图像的 LT 参考。

用于在第 10 帧插入长期图像示例 Gstreamer 应用的 cmd 行选项：

```
>> zynqmp_vcu_encode -w 3840 -h 2160 -e avc  -o /run/op.h264 -i /run/input.yuv -d
IL:10
```

用于第 15 帧的 LTR 的 cmd 行选项：

```
>> zynqmp_vcu_encode -w 3840 -h 2160 -e avc  -o /run/op.h264 -i /run/input.yuv -d
UL:15
```

自适应 GOP

- 用户指定可在 GOP 中使用的最大 B 帧数 （这些帧可通过控制软件中编码器配置文件中的“GopCtrlMode = ADAPTIVE_GOP”和 GStreamer 中的 gop-mode = adaptive 将 GOP 模式设置为自适应）
- 编码器基于视频内容的启发式方法来调整 gop 模式中所用 B 帧的数量
- 编码器不会高于用户指定的最大 B 帧数

在编码器中插入 SEI 数据

- AL_Encoder_AddSei()：将 SEI NAL 添加到流中。用户负责 sei 有效载荷，并必须按照 ITU-T 附件 D.3 的规定进行编写。编码器会完成包括有效载荷的反仿真在内的其余工作。
- 限制：SEI 不得超过 2 KB。对于所有的 SEI NAL 而言，这应该足够了。
- 流缓存需要有足够的空间来添加 SEI 部分。否则， AL_Encoder_AddSei() 即会报告失败。
- 支持前缀和后缀 SEI。编码器将 SEI 部分置于流缓存中的适当位置以创建符合条件的流。
- 控制软件应用会显示通过 AL_Encoder_AddSei() 添加 sei 消息的情况。
- 要将多个 SEI 消息插入流，可以多次调用 AL_Encoder_AddSei()，也可以将多个 sei 有效荷载添加到 AL_Encoder_AddSei() API 中。
- 每个流缓存设有 2 KB 的限制，即每个 AU 的所有 SEI 消息应该在 2 KB 限制内。
- SEI 消息已经与相应的视频帧同步，应该不需要任何同步时间戳机制。

如需了解更多详情，请参阅“VCU 控制软件 API”。

SEI 解码器 API

- 可以从解码器中检索 SEI 消息。
- 每次解码器解析 SEI 时，都会调用 AL_CB_ParsedSEI 回调。
- 提供 sei_payload 数据，如 ITU-T 附件 D.3 中所述。

- 反仿真已被解码器移除。
- 控制软件应用显示的是 API 使用示例 （可以使用命令行上的 -sei-file 选项来触发）。

如需了解更多详情，请参阅“VCU 控制软件 API”。

双通道编码

对视频进行两次编码，第一次编码收集序列相关信息，而 stats 用来改进第二次编码。

两个级别：

- 实时 GOP/帧级双通<超前模式>
- 离线序列级双通<双通模式>

2018.3 版本仅支持实时 GOP/帧级双通道

基于首次通过时的场景变化检测自动插入 IDR 帧。

基于内部流的大小/复杂度统计/场景变化来调整每个帧的 QP

通过在控制软件上使用“LookAhead”参数和 Gstramer 的“look-ahead”来启用 Gop/Fame-level 双通道编码。

LookAhead = <value>

- 默认值为 0：禁用双通道。
- 高于 2 (>=2)：场景变化检测和校正。
- 高于 10：恒定质量 （使用帧复杂度）。

场景变化检测

赛灵思视频场景变化检测 IP 可提供执行场景变化检测算法的视频处理块。此 IP 核对连续帧的垂直二次采样亮度帧上的直方图进行计算，然后通过绝对差之和 (SAD) 对这些帧的直方图进行比较。该 IP 核可通过全面的寄存器接口进行编程，以控制帧的大小、视频格式和子采样值。

场景变化检测 IP 是可配置的 IP 核，可以在内存模式下读取多达 8 个视频流。在内存模式下。所有输入都是从内存映射的 AXI4 接口读取的。该 IP 支持 64x64 到 8192x4320 的分辨率，采用的是各种 8 位和 10 位彩色格式。在为每个帧的所有输入流生成 SAD 值之后，IP 会发送中断。在内存模式下，系统会一个接一个地依次计算每个输入流的 SAD 值，并且中断发生在最终流的 SAD 计算之后。生成中断时，从 SAD 寄存器读取 SAD 值即可获得已配置流的个数；将这些值与用户确定的阈值进行比较，即可确定是否已发生场景变化。

场景变化检测最重要的应用是视频监控、机器学习和视频会议。在这些用例中， SCD IP 会在捕获流水线中，而且产生的中断会被附加到捕获的每个缓存中。同一个事件与缓存一起传递到编码器，再由编码器决定插入 I 帧而不插入 P 帧或 B 帧中。在场景改变的地方插入 I 帧将保持视频的质量。

Gstreamer 流水线：

1. File -> decode -> scd -> encode

```
gst-launch-1.0 filesrc location=file.mp4 ! qtdemux ! h264parse ! omxh264dec ! queue
! xilinxscd io-mode=5 ! queue ! omxh264enc target-bitrate=20000 control-rate=2
cpb-size=5000 ! filesink location=file.264
```

2. YUV -> scd -> encode

```
gst-launch-1.0 filesrc location=file.yuv ! videoparse width=1920 height=1080
format=nv12 framerate=30/1 ! xilinxscd ! omxh264enc target-bitrate=8000
control-rate=2 cpb-size=4000 ! filesink location=file.264
```

隔行扫描视频

隔行扫描视频是一种可将带宽相同的视频显示帧速率加倍的技术。隔行扫描帧包含在两个不同时间捕获的两个字段。这会增强观看者的运动感知，并会在继续运动时通过利用快速连续的观看来减少闪烁。

隔行扫描信号需要能够按顺序显示各个字段的显示器。 CRT 显示器用于显示隔行扫描信号。隔行扫描指的是通过扫描每行或每行的像素在显示屏上绘制视频图像。此技术使用两个字段来创建帧。一个字段包含图像中的所有奇数行，另一个字段包含所有偶数行。

Xilinx VCU 支持 H265 隔行扫描视频的编码和解码。 VCU 可以解码和编码各种视频分辨率，如 1920x1080i、 720x576i 和 720x480i。

Gstreamer 流水线：

1. filesrc -> omxh265dec -> omxh265enc -> filesink

   ```
   gst-launch-1.0 filesrc location=file_1080i.h265 ! omxh265dec ! omxh265enc
   target-bitrate=10000 control-rate=2 ! queue max-size-bytes=-1 ! filesink
   location=file.h265
   ```

2. v4l2src -> omxh265enc -> omxh265dec -> kmssink

   ```
   gst-launch-1.0 v4l2src io-mode=4 ! video/
   x-raw,interlace-mode=alternate,format=NV16_10LE32,width=1920,height=1080,framerate=
   30/1 ! omxh265enc target-bitrate=10000 control-rate=2 ! omxh265dec ! queue
   max-size-bytes=-1 ! kmssink bus-id=drm-pl-disp-drv
   connector-properties="props,sdi_mode=0,sdi_data_stream=2,is_frac=0"
   ```

GDR 帧内刷新

Gradual Decoder Refresh (GDR)，当 GOPCtrlMode 设置为 LOW_DELAY_P，GDRMode 参数用于指定是否应使用 GDR 方案。启用 GDR （水平/垂直）后，编码器会插入图像中的帧内 MB 行/列以刷新解码器。 Gop.FreqIDR 指定刷新模式应发生的频率。要允许完整的图像刷新， Gop.FreqIDR 参数应大于 CTB/MB 行数 (GDR_HORIZONTAL) 或列数 (GDR_VERTICAL)。

启用 GDR 后，编码器会在刷新模式开始时插入 SPS/PPS 和 SEI 恢复消息，并且解码器可以与位流中的 SEI 恢复点同步。实现非 IDR 解码器同步的好处是，在丢包情况下不需要发送 IDR 图像。在视频编码中插入周期性 IDR 图像 （不建议用于视频流）会生成比特消耗的峰值

示例 gstreamer 流水线如下：

在编码器元素中使用“gdr-mode=horizontal”和“periodicity-idr=<higher than Picture Height in Mbs>”。

实例：

```
gst-launch-1.0 filesrc location="/mnt/test_4Kp60_NV12.yuv"! videoparse width=3840
height=2160 format=nv12 framerate=30/1! omxh264enc control-rate=constant
target-bitrate=60000 gop-mode=low-delay-p gdr-mode=horizontal periodicity-idr=270!
video/x-h264, profile=high! filesink location="/run/test_output.avc"
```

在控制软件级添加以下部分和参数 （在 cfg 文件的 GOP 部分）

```
[GOP] #-----------------------------------------------------
GopCtrlMode                = LOW_DELAY_P
Gop.GdrMode                = GDR_HORIZONTAL
Gop.FreqIDR                = 270
```

GStreamer

下面提供的是从 PetaLinux 命令行运行 GStreamer 的示例。要查看 gstreamer 元素说明和每个元素中所用的属性，请使用 gst-inspect-1.0 命令。

例如，要获取"omxh264dec"元素的每个参数说明，请在命令提示符窗口输入以下内容：

```
gst-inspect-1.0 omxh264dec
```

H.264 解码

对基于 H.264 的输入文件进行解码，并显示在连接到 Display 端口的监视器上

```
gst-launch-1.0 filesrc location="input-file.mp4" ! qtdemux name=demux demux.video_0
! h264parse ! omxh264dec ! queue max-size-bytes=0 ! kmssink
bus-id=fd4a0000.zynqmp-display fullscreen-overlay=1
```

H.265 解码

对基于 H.265 的输入文件进行解码，并显示在连接到 Display 端口的监视器上

```
gst-launch-1.0 filesrc location="input-file.mp4" ! qtdemux name=demux demux.video_0
! h265parse ! omxh265dec ! queue max-size-bytes=0 ! kmssink
bus-id=fd4a0000.zynqmp-display fullscreen-overlay=1
```

注释：Input-file.mp4 可以是以下任一格式：

- 4:2:0 8 位
- 4:2:2 8 位
- 4:2:0 10 位
- 4:2:2 10 位

高比特率比特流解码

要在 4kP30 减少大于 100 Mbps 的比特流的帧解码时间，请使用以下选项：

- 将内部解码器缓存 （internal-entropy-buffers 参数）增加到 9 或 10。
- 在解码器输入端添加队列

以下命令使用增加数量的内部熵缓存来解码 H.264 MP4 文件，并通过 DisplayPort 显示。

```
gst-launch-1.0 filesrc location="input-file.mp4" ! qtdemux name=demux demux.video_0
! h264parse ! queue max-size-bytes=0 ! omxh264dec internal-entropy-buffers=10 !
queue max-size-bytes=0 ! kmssink bus-id=fd4a0000.zynqmp-display fullscreen-overlay=1
```

H.264 编码

将输入 3840×2160 4:2:0 8-bit (NV12) YUV 文件编码为 H.264。

```
gst-launch-1.0 filesrc location="input-file.yuv" ! videoparse format=nv12 width=3840
height=2160 framerate=30/1 ! omxh264enc ! filesink location="output.h264"
```

将 3840×2160 4:2:2 8-bit (NV16) YUV 文件编码为 H.264。

```
gst-launch-1.0 filesrc location="input-file.yuv" ! videoparse format=nv16 width=3840
height=2160 framerate=30/1 ! omxh264enc ! filesink location="output.h264"
```

将 3840×2160 4:2:0 10-bit (NV12_10LE32) YUV 文件编码为 H.264。

```
gst-launch-1.0 filesrc location="input-file.yuv" ! videoparse format=nv12-10le32
width=3840 height=2160 framerate=30/1 ! omxh264enc ! filesink location="output.h264"
```

将 3840×2160 4:2:2 10-bit (NV16_10LE32) YUV 文件编码为 H.264。

```
gst-launch-1.0 filesrc location="input-file.yuv" ! videoparse format=nv16-10le32
width=3840 height=2160 framerate=30/1 ! omxh264enc ! filesink location="output.h264"
```

H.265 编码

将 3840×2160 4:2:0 8-bit (NV12) YUV 文件编码为 H.265。

```
gst-launch-1.0 filesrc location="input-file.yuv" ! videoparse format=nv12 width=3840
height=2160 framerate=30/1 ! omxh265enc ! filesink location="output.h265"
```

将 3840×2160 4:2:2 8-bit (NV16) YUV 文件编码为 H.265。

```
gst-launch-1.0 filesrc location="input-file.yuv" ! videoparse format=nv16 width=3840
height=2160 framerate=30/1 ! omxh265enc ! filesink location="output.h265"
```

将 3840×2160 4:2:0 10-bit (NV12_10LE32) YUV 文件编码为 H.265。

```
gst-launch-1.0 filesrc location="input-file.yuv" ! videoparse format=nv12-10le32
width=3840 height=2160 framerate=30/1 ! omxh265enc ! filesink location="output.h265"
```

将 3840×2160 4:2:2 -10-bit (NV16_10LE32) YUV 文件编码为 H.265。

```
gst-launch-1.0 filesrc location="input-file.yuv" ! videoparse format=nv16-10le32
width=3840 height=2160 framerate=30/1 ! omxh265enc ! filesink location="output.h265"
```

注释：上面的命令行假定文件 input-file.yuv 采用的是指定的格式。

从 H.264 转码到 H.265

将基于 H.264 的输入容器格式文件转换为 H.265 格式

```
gst-launch-1.0 filesrc location="input-h264-file.mp4" ! qtdemux name=demux
demux.video_0 ! h264parse ! omxh264dec ! omxh265enc ! filesink
location="output.h265"
```

从 H.265 转码到 H.264

将基于 H.265 的输入容器格式文件转换为 H.264 格式

```
gst-launch-1.0 filesrc location="input-h265-file.mp4" ! qtdemux name=demux
demux.video_0 ! h265parse ! omxh265dec ! omxh264enc ! filesink
location="output.h264"
```

多码流解码

同时使用四个解码器元素对 H.265 输入文件进行解码，将文件保存到单独的文件

```
gst-launch-1.0 filesrc location=input_1920x1080.mp4 ! qtdemux ! h265parse ! tee
name=t
```

```
t. ! queue ! omxh265dec ! queue max-size-bytes=0 ! filesink
location="output_0_1920x1080.yuv"
t. ! queue ! omxh265dec ! queue max-size-bytes=0 ! filesink
location="output_1_1920x1080.yuv"
t. ! queue ! omxh265dec ! queue max-size-bytes=0 ! filesink
location="output_2_1920x1080.yuv"
t. ! queue ! omxh265dec ! queue max-size-bytes=0 ! filesink
location="output_3_1920x1080.yuv"
```

注释：tee 元素用于将相同的输入文件馈送到 4 个解码器实例中，用户可以使用单独的 gst-launch-1.0 应用来馈送不同的输入。

多码流编码

通过同时使用八个编码器元素将输入 YUV 文件编码为八个流。

```
gst-launch-1.0 filesrc location=input_nv12_1920x1080.yuv ! videoparse width=1920
height=1080 format=nv12 framerate=30/1 ! tee name=t
t. ! queue ! omxh264enc control-rate=2 target-bitrate=8000 ! video/x-h264,
profile=high ! filesink location="ouput_0.h264"
t. ! queue ! omxh264enc control-rate=2 target-bitrate=8000 ! video/x-h264,
profile=high ! filesink location="ouput_1.h264"
t. ! queue ! omxh264enc control-rate=2 target-bitrate=8000 ! video/x-h264,
profile=high ! filesink location="ouput_2.h264"
t. ! queue ! omxh264enc control-rate=2 target-bitrate=8000 ! video/x-h264,
profile=high ! filesink location="ouput_3.h264"
t. ! queue ! omxh264enc control-rate=2 target-bitrate=8000 ! video/x-h264,
profile=high ! filesink location="ouput_4.h264"
t. ! queue ! omxh264enc control-rate=2 target-bitrate=8000 ! video/x-h264,
profile=high ! filesink location="ouput_5.h264"
t. ! queue ! omxh264enc control-rate=2 target-bitrate=8000 ! video/x-h264,
profile=high ! filesink location="ouput_6.h264"
t. ! queue ! omxh264enc control-rate=2 target-bitrate=8000 ! video/x-h264,
profile=high ! filesink location="ouput_7.h264"
```

注释：tee 元素用于将一个输入文件馈送给八个编码器实例。

对于备用的输入 YUV 格式，需要在上述流水线中进行下列更改：

格式/配置信息	实参
4:2:2 8 位 (NV16)	format=nv16 profile=high-4:2:2
4:2:0 10-bit (NV12_10LE32)	format=nv12-10le32 profile=high-10
4:2:2 10 位 (NV16_10LE32)	format=nv16-10le32 profile=high-4:2:2

通过以太网进行转码和流媒体传输

```
gst-launch-1.0 filesrc location="test_0003.mp4" ! qtdemux ! h264parse ! omxh264dec !
omxh265enc control-rate=2 target-bitrate=20000 ! h265parse ! queue ! rtph265pay !
udpsink host=192.168.1.1 port=50000 buffer-size=20000000 async=false max-lateness=-1
qos-dscp=60
```

注释：192.168.1.1 是一个客户端的 IP 地址示例。您可能需要使用实际的客户端 IP 地址对其进行修改。

通过以太网流传输与解码到显示流水线

```
gst-launch-1.0 udpsrc port=50000 caps="application/x-rtp, media=video,
clock-rate=90000, payload=96, encoding-name=H265" buffer-size=20000000 !
```

```
rtpjitterbuffer latency=1000 ! rtph265depay ! h265parse ! omxh265dec ! queue !
kmssink bus-id=fd4a0000.zynqmp-display fullscreen-overlay=1 sync=false
```

适用于流传输的推荐设置

以下是适用于流式传输的推荐设置：

- 基于 low-delay-p gop-mode 的 Low-latency RC，并且 gdr-mode=horizontal、 periodicity-idr=Picture Height （以 MB 为单位）。
- low-delay-p gop-mode 的 Low-latency RC，并且 periodicity-idr=twice the framerate。
- low-delay-p gop-mode 的 CBR RC、 gdr-mode=horizontal、 periodicity-idr=Picture Height （以 MB 为单位）。
- low-delay-p gop-mode 的 CBR RC，并且 periodicity-idr=twice the framerate、 cpb-size="1000"

对于 AVC， 1 MB=HEVC 的 16x16 像素 ,因此用户需要以 Mbs= roundup(Height,64)/#Mb 行来计算图像的高度

注释：

- 如果您没有使用 `udpsrc` 的 buffer-size 属性，则必须根据其网络带宽利用率和要求使用 `sysctl` 命令手动设置它。

  ```
  -> sysctl -w net.core.rmem_default=60000000
  ```

- VBR 不是首选的流媒体模式。

已验证的 GStreamer 元素

表 12-8：已验证的 GStreamer 元素

元素	描述
filesink	将传入数据写入本地文件系统中的文件
filesrc	从本地文件系统中的文件读取数据
h264parse	解析 H.264 编码流
h265parse	解析 H.265 编码流
kmssink	直接在 DRM 器件的平面中呈现视频帧
omxh264dec	解码 OpenMAX H.264 视频
omxh264enc	编码 OpenMAX H.264 视频
omxh265dec	解码 OpenMAX H.265 视频
omxh265enc	编码 OpenMAX H.265 视频
qtdemux	将 .mov 文件解复用为原始或压缩音频和/或视频流。
queue	对数据进行排队，直到达到"max-size-buffers"、"max-size-bytes"或"max-size-time"属性指定的限制之一为止
rtph264depay	从 RTP 数据包流中提取 H.264 视频的有效载荷
rtph264pay	将 H.264 视频封装在 RTP 数据包流中
rtph265depay	从 RTP 数据包流中提取 H.265 视频的有效载荷
rtph265pay	将 H.265 视频封装在 RTP 数据包流中
rtpjitterbuffer	重新排序并删除从网络源接收的重复 RTP 数据包
tee	将数据拆分到多个管脚

表 12-8：已验证的 GStreamer 元素 （续）

元素	描述
udpsink	将 UDP 数据包接收到网络
udpsrc	从网络读取 UDP 数据包
v4l2src	从 v4l2 器件捕获视频，如网络摄像头和电视调谐卡
videoparse	将字节流转换为视频帧

使用 GStreamer 的已验证容器

表 12-9：使用 GStreamer 的已验证容器

格式	AVC （支持：有/无）	HEVC （支持：有/无）
MP4	有	有
MPEG2-TS	有	有
MKV	有	有
AVI	有	无
MPEG2-PS	有	无
FLV	有	无
3GP	有	无

使用 GStreamer 的已验证流媒体协议

表 12-10：使用 GStreamer 的已验证流媒体协议

协议	AVC （支持：有/无）	HEVC （支持：有/无）	格式
RTP	有	有	TS
UDP	有	有	TS
TCP	有	有	TS
HTTP	有	有	M3U8、 TS、 MP4

OpenMAX Integration Layer

OpenMax Integration Layer 示例应用

目前有两个使用 OpenMax 集成层构建的示例应用。

如需查看 OpenMax 示例应用 omx_encoder 和 omx_decoder 的源代码，请单击 https://github.com/Xilinx/vcu-omx-il/tree/master/exe_omx。

H.265 解码 （文件到文件）

```
omx_decoder input-file.h265 -hevc -o out.yuv
```

`omx_decoder –help` 命令显示所有选项。

H.265 编码 （文件到文件）

```
omx_encoder inputfile.yuv -w 352 -h 288 -r 30 -avc -o out.h264
```

`omx_encoder –help` 命令显示所有选项。

注释：如果是 8 位输入源， YUV 输入文件的格式应为 NV12 或 NV16。

VCU 控制软件

VCU 控制软件在帧或 slice 级上运行。其职责是：

- 生成编码器的 NAL （网络抽象层）单元。
- 解析解码器的 NAL 单元。
- 将每个帧的命令组合在一起并排队到 MCU 固件。
- 检索每个帧的状态。
- 连接由硬件和软件生成的视频比特流。

驱动

PetaLinux 中有三个与 VCU 相关的内核驱动。解码器驱动称为 al5d。编码器驱动称为 al5e。 VCU 内核驱动称为 allegro。 allegro 驱动具有以下职责：

- 加载 MCU 固件。
- 启动 MCU 启动顺序。
- 将邮箱消息写入 APU 和 MCU 之间共享的内存中。
- 提供新邮箱消息的通知。

MCU 固件

MCU 上运行的 MCU 固件具有以下职责：

- 将帧级命令从 VCU 控制软件转换为硬件 IP 核的 slice 级命令。
- 为每个命令配置硬件寄存器。
- 在每个帧之间执行速率控制。

编码堆栈

图 12-3 显示的是编码软件堆栈。

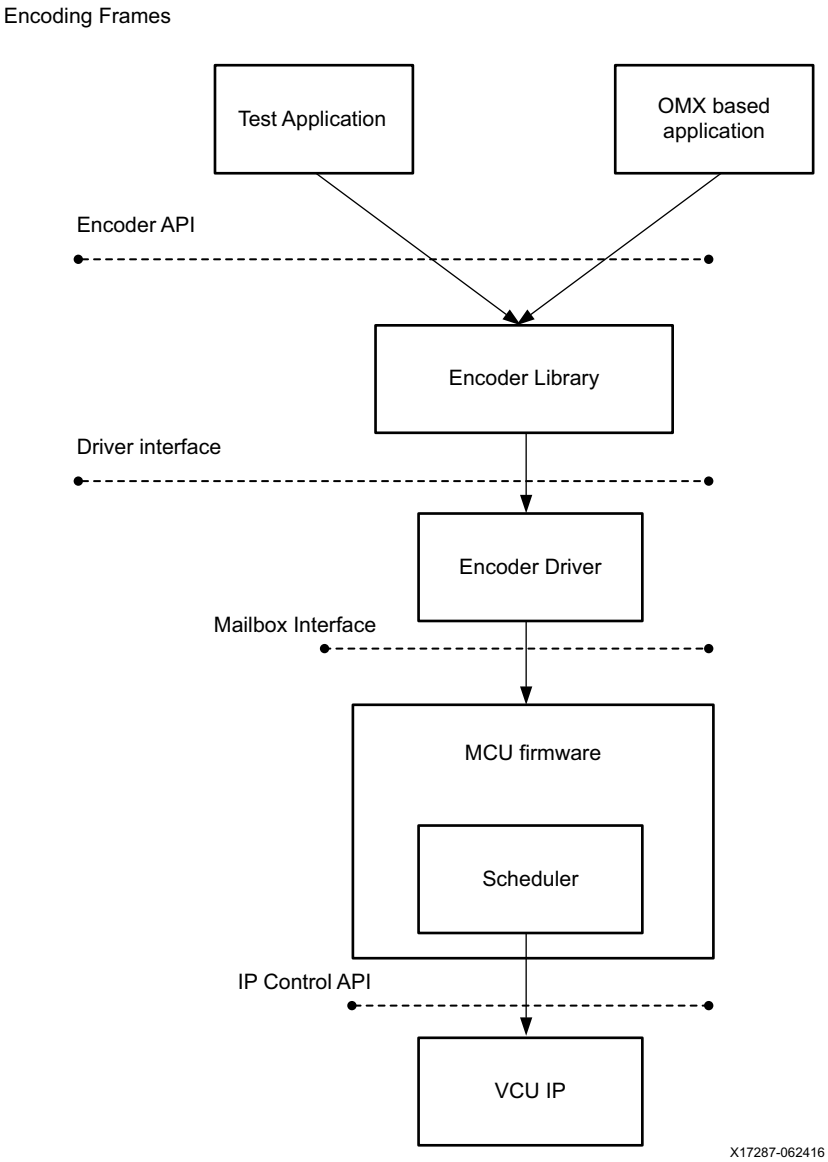

图 12-3：编码器简介

应用

应用是指使用 VCU 底层编码器功能的任何基于 OpenMAX 或独立的应用。

编码器库

编码器库可提供配置编码器并将帧发送到编码器的入口点。

编码器驱动

编码器驱动将 VCU 编码器必须操作的视频位流的控制信息和缓存指针传递给 MCU 固件。编码器驱动使用邮箱通信技术将此信息传递给 MCU 固件。

MCU 固件

固件通过邮箱接收控制和缓存信息。它会采取适当的措施并将状态传回给编码器驱动。

调度程序

调度程序用于指示硬件活动、处理中断，并管理编码的多通道和多 slice 操作等。

解码器堆栈

图 12-4 显示的是解码器软件堆栈。

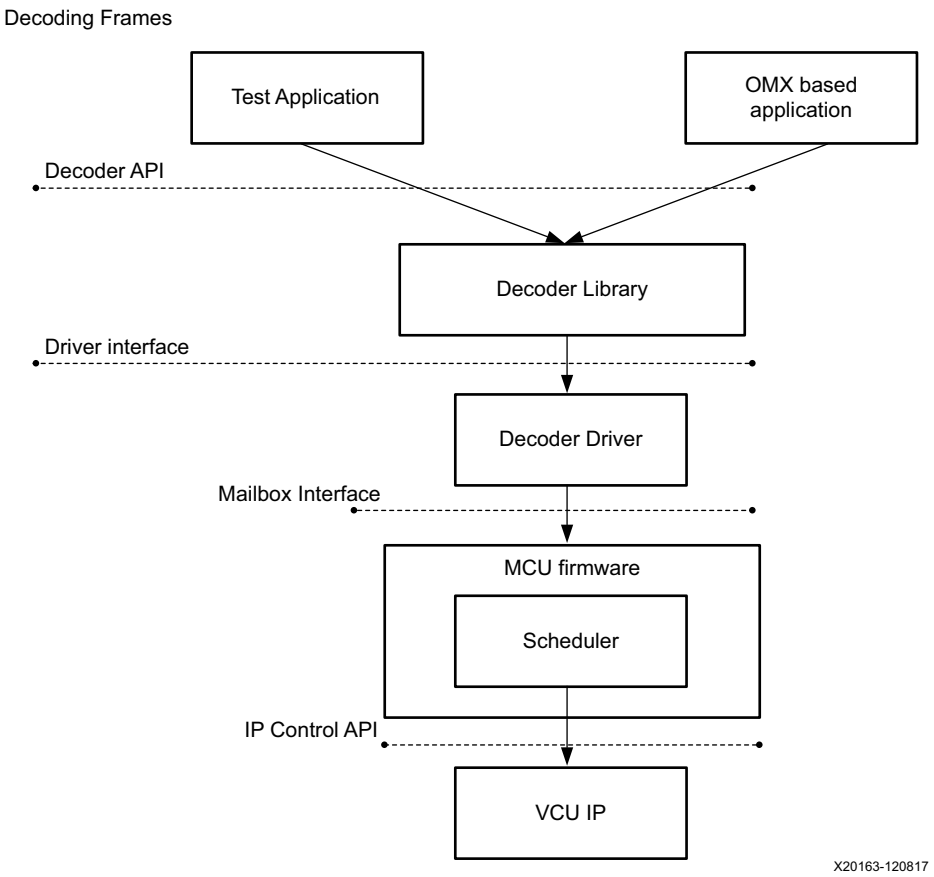

图 12-4：解码器块简介

应用

该应用可以是测试模式生成器，也可以是使用 VCU 解码器、基于 OpenMAX 的应用。

解码器库

解码器库使应用能够通过解码器驱动与 MCU 固件进行通信。

解码器驱动

解码器驱动将控制信息和视频的缓存指针传递给 MCU 固件。编码器驱动使用邮箱通信技术将此信息传递给 MCU 固件。

MCU 固件

固件通过邮箱接收控制和缓存信息。它会采取适当的措施并将状态传回给解码器驱动。

调度程序

调度程序是 MCU 固件的一部分，用于对 HW IP 进行编程、处理中断并对解编码的多通道和多 slice 操作进行管理。

VCU 控制软件示例应用

Ctrlsw_encoder 和 ctrlsw_decoder 是分别对视频进行编码和解码的完整示例应用。这些应用被设计来作为学习辅助工具用于 VCU 控制软件 API 和故障排除。如需查看 ctrlsw_encoder 和 ctrlsw_decoder 应用的源代码位，请单击 https://github.com/Xilinx/vcu-ctrl-sw。

以下示例中提到的示例配置文件和输入 yuv 文件可以在 VCU 控制软件源代码树的 test/cfg 文件夹中找到。我们还在下面的示例后提供了参数描述。

- H.264 解码 （文件到文件）

  ```
  ctrlsw_decoder -avc -in input-avc-file.h264 -out ouput.yuv
  ```

- H.265 解码 （文件到文件）

  ```
  ctrlsw_decoder -hevc -in input-hevc-file.h265 -out ouput.yuv
  ```

- 编码 （文件到文件）

  ```
  ctrlsw_encoder –cfg encode_simple.cfg
  ```

注释：有关完整的参数列表，请在命令行中键入以下内容：

```
ctrlsw_decoder --help
ctrlsw_encoder --help
```

VCU 控制软件编码器参数

输入参数

表 12-11：编码器的输入参数

参数	描述和可能的值
YUVFile	YUV 文件名
Width	帧宽 （以像素为单位）
Height	帧高 （以像素为单位）

表 12-11：编码器的输入参数 （续）

参数	描述和可能的值
Format	I420：YUV 文件包含以平面格式存储的 4:2:0 8 位视频样本，其中所有图像 Luma (Y) 样本后跟色度样本 （所有 U 样本后面是所有 V 样本） IYUV：与 I420.相同。 YV12：与 I420.相同，但反转了 U 和 V 的顺序。 NV12：YUV 文件包含以平面格式存储的 4:2:0 8 位视频样本，所有图像 Luma (Y) 样本后面跟着交织的 U 和 V 色度样本。 NV16：YUV 文件包含以平面格式存储的 4:2:0 8 位视频样本，所有 图像 Luma (Y) 样本后面跟着交织的 U 和 V 色度样本。 I0AL：YUV 文件包含 4:2:0 10 位视频样本，每个样本以平面格式存储在 16 位字中，所有图像 Luma (Y) 样本后面跟着色度样本 （所有 U 样本后面是所有 V 样本）。 P010：YUV 文件包含 4:2:0 10 位视频样本，每个样本以平面格式存储在 16 位字中，所有图像 Luma (Y) 样本后面跟着交织的 U 和 V 色度样本。 I422：YUV 文件包含以平面格式存储的 4:2:2 8 位视频样本，所有图像 Luma (Y) 样本后面跟着色度样本 （所有 U 样本后面是所有 V 样本）。 YV16：与 I422 相同。 I2AL：YUV 文件包含 4:2:2 10 位视频样本，每个样本以平面格式存储在 16 位字中，所有图像 Luma (Y) 样本后面跟着色度样本 （所有 U 样本后面是所有 V 样本）。 P210：YUV 文件包含 4:2:2 10 位视频样本，每个样本以平面格式存储在 16 位字中，所有图像 Luma (Y) 样本后面跟着交织的 U 和 V 色度样本。 XV20：YUV 文件包含 4:2:2 10 位视频样本，其中每 32 比特字使用半平面格式存储 3 个样本，所有图像 Luma (Y) 样本后面跟着交织的 U 和 V 色度样本。 Y800：输入文件包含单色 8 位视频样本。 XV15：YUV 文件包含 4:2:0 10 位比特视频样本，其中每 32 比特的 word 使用半平面格式存储 3 个样本，所有图像 Luma(Y) 样本之后是交织的 U 和 V 色度样本。 Y010：输入文件包含单色 10 位视频样本，每个样本存储在一个 16 位的 word 中。 **注释：**I420、IYUV、YV12、I0AL、P010、I422、YV16、I2AL 和 P210 格式是平面的，而且是通过软件转换为平面的。
Framerate	YUV 输入文件的每秒帧数。如果此参数不存在，则将其值设置为等于 “速率控制参数” 中指定的帧速率。当此参数大于速率控制部分中指定的帧速率时，编码器重复一些帧编码器丢弃一些帧；当此参数低于帧时。
CmdFile	文本文件，用于指定在给定帧编号下执行的命令。这些命令包括场景变化通知、关键帧插入等。
RoiFile	文本文件，用于指定给定帧编号处感兴趣区域的序列。

输出参数

表 12-12：编码器的输出参数

参数	描述和可能的值
BitstreamFile	基本流输出文件名
RecFile	指重构图像的可选输出文件名 （以 YUV 格式显示）。 如果此参数不存在，则不保存重建的 YUV 文件。
Format	有关重构 YUV 文件格式的 FOURCC 代码，请参阅输入部分了解可能的值。如果未指定，则输出使用输入的格式。

速率控制参数

表 12-13：编码器速率控制部分的参数

参数	描述和可能的值
RateCtrlMode	选择控制比特率的方式 CONST_QP：无速率控制，所有图像都使用 SliceQP 参数定义的同一个 QP。 CBR：使用常量比特率控制。 VBR：使用可变比特率控制。 LOW_LATENCY：使用可变比特率进行低时延应用。
BitRate	以 Kbits/s 为单位的目标比特率。当 RateCtrlMode = CONST_QP 时不使用 默认值：4000
MaxBitRate	以 Kbits/s 为单位的目标比特率。当 RateCtrlMode = CONST_QP 时不使用 默认值：4000 注意：当 RateCtrlMode 是 CBR 时， MaxBitRate 应设置为与 BitRate 相同的值
FrameRate	每秒帧数 默认值：30
SliceQP	量化参数。当 RateCtrlMode = CONST_QP 时，指定的 QP 应用于所有 slice。当 RateCtrlMode = CBR 时，指定的 QP 用作初始 QP。 允许的值：0 - 51 默认值：30
MinQP	允许的最小 QP 值。使用 VBR 速率控制时，此参数特别有用。在 VBR 速率控制中，值 AUTO 可用于让编码器根据 SliceQP 选择 MinQP。 允许的值：0 - SliceQP 默认值：10
MaxQP	允许的最大 QP 值。 允许的值：SliceQP - 51 默认值：51
InitialDelay	按照 HRD 模型中的规定，在几秒钟内完成初始移除延迟。当 RateCtrlMode = CONST_QP 时不使用。 默认值：1.5
CPBSize	按照 HRD 模型中的规定，在几秒钟内指定编码图像缓的大小。当 RateCtrlMode = CONST_QP 时不使用。 默认值：3.0
IPDelta	指定 I 帧和 P 帧之间的 QP 差异。 允许的值：自动或正值 (≥ 0)
PBDelta	指定 p 帧和 B 帧之间的 QP 差异。 允许的值：自动或正值 (≥ 0)
ScnChgResilience	用于指定编码过程中的场景更改弹性处理。在场景变化期间提升质量。 启用/禁用。

VCU 软件中的 `CPBSize` 默认值被设置为 3 秒（如果定义的编码级次和比特率参数允许的话），并可以根据应用要求更改此值。

编码器 CBR 速率控制试图在 GOP 的长度周期内达到目标比特率，但主要约束是它必须避免由标准假设参考解码器 (HDR) 模型定义的缓存下溢/溢出。

较大的 `CPBSize` 允许编码器速率控制在较大数量的帧上分配编码比特，使它可以处理连续帧之间的较大比特率变化并提升视频质量。

将 `CPBSize` 设置为较小的值可以降低比特率峰值，但也会影响视频质量。

重要提示：CPB 大小优化必须基于应用完成。

- 视频录制和存储应用，这里瞬时比特率波动不重要，而且可以支持更大的缓存，并且应该将 CPBSize 设置为更大的值 （~1 秒 - 3 秒）。
- 建议将 CPBSize 设置为大于 GOP 持续时间长度的值 （例如，对于 60 fps 的设置， GOP 长度可以设置为 60 帧，设置为大于 GOP 持续时间长度值 （例如，对于 60 fps 的设置， GOP 长度可以设置为 60 帧， CPBSize 可以设置为 1 秒以上）。
- 需要较小比特率变化的应用可以减少 CPBSize，但建议设置一个大于 ~6 帧周期的值。另外，建议在这种情况下启用低时延速率控制模式。
- 低时延应用应该使用“低延迟”GOP 类型 （或仅限内部 GOP 类型），然后可以将 CPBSize 减少到 ~ 1 - 2 个帧周期。

图像组参数

表 12-14：编码器 GOP 部分的参数

参数	描述和可能的值
GopCtrlMode	指定图像组配置模式。 允许的值： DEFAULT_GOP: IBBPBBP... （显示顺序） LOW_DELAY_P：GopPattern，开头有一张 I 图像，后面只有两个或多个 P 图像。每个 P 图像使用前一张图像作为参考。IPPPP.... LOW_DELAY_P：GopPattern，开头有一张 I 图像，后面只有 B 图像。每个 B 图像使用前面的图像作为第一个参考；第二个引用取决于 Gop.Length 参数。IBBB... PYRAMIDAL_GOP：具有分层级 B 帧的高级 GOP 模式。层级的大小取决于 Gop.NumB 参数
Gop.Length	帧中的 GOP 长度包括 I 图像。仅在 GopCtrlMode 设置为 DEFAULT_GOP 时使用。仅限于 Intra-only 设置为 0 时 范围：0-1,000。 默认值：30
Gop.FreqIDR	指定两个 IDR 图像 （AVC 和 HEVC）之间的最小帧数。 IDR 插入取决于 GOP 边界的位置。 允许的值：正整数或 -1，表示禁用 IDR 区域。 默认值：-1
Gop.FreqLT	以帧数为单位指定长期参考图像刷新频率。 允许的值：正整数或 0 表示禁用长期参考图像 默认值：0
Gop.NumB	GOP 中的最大连续 B 帧数。仅在 GopCtrlMode 设置为 DEFAULT_GOP 时使用 允许的值： 当 GopCtrlMode 设置为 DEFAULT_GOP 时， Gop.NumB 应在 0 - 4 的范围内。
Gop.GdrMode	当 GopCtrlMode 设置为 LOW_DELAY_P 或 LOW_DELAY_B 时，此参数指定是否应使用渐近解码器刷新 (GDR) 方案。启用 GDR 时，Gop.Length 指定刷新模式应发生的频率。要允许完整图像刷新，此参数应大于 CTB/MB 行数 (GDR_HORIZONTAL) 或列数 (GDR_VERTICAL)。 禁用：无 GDR GDR_VERTICAL：使用从左到右移动的垂直条逐渐刷新。 GDR_HORIZONTAL：使用从上到下移动的水平条逐渐刷新。

设置参数

表 12-15：编码器的设置参数

参数	描述和可能的值
Profile	指定比特流符合的配置信息 允 许 的 值：HEVC_MAIN、 HEVC_MAIN10、 HEVC_MAIN_STILL、 HEVC_MONO、 HEVC_MAIN_422_10、 HEVC_MAIN_INTRA、 HEVC_MAIN10_INTRA、 HEVC_MAIN_422_10_INTRA AVC_BASELINE、 AVC_MAIN、 AVC_HIGH、 AVC_HIGH10、 AVC_HIGH_422、 AVC_HIGH10_INTRA、 AVC_HIGH_422_INTRA
级别	指定比特流符合的级 允许的值：H.265 (HEVC) 1.0 - 5.1、 H.264 (AVC) 1.0 - 5.2
Tier	指定比特流符合的层级 （仅限 H.265 (HEVC)） 允许的值：MAIN_TIER、 HIGH_TIER
ChromaMode	选择用于对流进行编码的色度子采样模式 允许的值： CHROMA_MONO：流以单色 (4:0:0) 进行编码 CHROMA_4_2_0：流采用 4:2:0 色度子采样进行编码 CHROMA_4_2_2：流采用 4:2:2 色度子采样进行编码
BitDepth	指定编码流中的亮度和色度样本的位深度。 允许的值：8 或 10
NumSlices	指定每帧使用的 slice 数。每个 slice 包含一个或多个完整的 LCU 行，并尽可能有规律地分布在帧上。 允许的值：从 1 到帧中的编码单元行数。
SliceSize	目标 Slice Size 指定编码器用于自动将比特流拆分为大致相等大小的目标 slice 大小 （以字节为单位），其粒度为一个 LCU。这会影响性能，增加每个 slice 一个 LCU/命令处理时间的开销。使用多个核时，H.264 (AVC) 编码不支持此参数。当 SliceSize 为零时，slice 由 NumSlice 参数定义。此参数直接发送到编码器 IP，并仅指定 slice 数据的大小。它不包含 slice 标头的任何边距。因此，建议设置 SliceSize 参数时将目标值降低 5%。例如，如果您的目标值是每个 slice 1500 个字节，则应在配置文件中设置“SliceSize = 1425”。 允许的值：1000-65,535 或 0 表示禁用自动 slice 分割。
DependentSlice	当每帧有多个 slice （例如， NumSlices 大于 1 或 SliceSize 大于 0）时，该参数指定附加 slice 是从属 slice 片段还是常规 slice （仅 H.265 (HEVC)）。 允许的值：FALSE、 TRUE
EntropyMode	如果 Profile 被设置为 AVC_MAIN,AVC_HIGH、AVC_HIGH10 或 AVC_HIGH_422（仅限 AVC only），则选择熵编码模式 允许的值： MODE_CABAC：使用 CABAC 对流进行编码 MODE_CAVLC：使用 CAVLC 对流进行编码
CabacInit	指定 CABAC 初始化表索引 (H.264 (AVC))/初始化标志 (H.265 (HEVC))。 允许的值：H.264 (HEVC)) 0 - 2、 H.265 (AVC) 0 - 1
PicCbQpOffset	指定图像级别的第一个色度通道 (Cb) 的 QP 偏移。 (仅 H.265 (HEVC)) 允许的值：-12 到 +12 默认值：0
PicCrQpOffset	指定图像级别的第二个色度通道 (Cr) 的 QP 偏移 （仅 H.265(HEVC)） 允许的值：-12 到 +12 默认值：0

表 12-15：编码器的设置参数 （续）

参数	描述和可能的值
SliceCbQpOffset	指定 slice 级的第一个色度通道 (Cb) 的 QP 偏移。 允许的值：-12 到 +12 默认值：0
SliceCrQpOffset	指定 slice 级的第二个色度通道 (Cb) 的 QP 偏移 允许的值：-12 到 +12 默认值：0 注释 (PicCbQPOffset + SliceCbQPOffset) 应在 -12 到 +12 的范围内 (PicCrQpOffset + SliceCrQpOffset) 应在 -12 到 +12 的范围内
ScalingList	指定缩放列表模式 （仅限 H.264 (AVC) and H.265 (HEVC)）。 允许的值：FLAT、 DEFAULT
QpCtrlMode	指定如何生成每个编码单元 (CU) 中的 QP。 UNIFORM_QP：Slice 的所有 CU 使用相同的 QP。 AUTO_QP：使用预编程的查找表根据 CU 内容选择 QP。 LOAD_QP：每个 LCU 的 QP 来自从文件加载的外部缓存。该文件应命名为 QPs.hex 并保存在工作目录中。文件格式在 “量化参数 (QP) 文件格式” 中进行了描述。
CuQpDeltaDepth	指定每个 CU 粒度的 Qp （仅限 H.265 (HEVC)）。仅在 QpCtrl Mode 设置为 AUTO_QP 或 ADAPTIVE_AUTO_QP 时使用 0：低至 32×32， 1：低至 16×16 2：低至 8×8
ConstrainedIntraPred	指定 constrained_intra_pred_flag 语法元素的值。 允许的值：ENABLE、 DISABLE
VrtRange_P	指定用于 P 帧运动估计的垂直搜索范围： H.265 (HEVC) 的允许值：16 或 32：使用 16 允许减少内存带宽 （低带宽模式） H.264 (AVC)：的允许值：8 或 16：使用 8 允许减少内存带宽 （低带宽模式）
LoopFilter	启用/禁用解块滤波器。 允许的值：ENABLE、 DISABLE
LoopFilter.CrossSlice	启用/禁用跨帧的每个 slice 的左边界和上边界的去块环路滤波。仅用于 LoopFilter 设置为 ENABLE 时。 允许的值：ENABLE、 DISABLE
LoopFilter.CrossTile	启用/禁用跨帧的每个块的左边界和上边界的去块环路滤波。 (仅 H.265 (HEVC)） 仅在 LoopFilter 设置为 ENABLE 时使用。 允许的值：ENABLE、 DISABLE
LoopFilter.BetaOffset	指定去块滤波器的 beta 偏移。仅用于 LoopFilter 设置为 ENABLE 时。 允许的值：-6 到 +6 默认值：-1
LoopFilter.TcOffset	指定去块滤波器的 Alpha_c0 offset (H.264 (AVC)) 或 Tc 偏移 (H.265 (HEVC))。仅用于 LoopFilter 设置为 ENABLE 时。 允许的值：-6 到 +6 默认值：-1

表 12-15：编码器的设置参数 （续）

参数	描述和可能的值
CacheLevel2	可选的编码器缓存可用于减少存储器的带宽。此选项可能会略微降低视频质量。 此参数指定用于编码器缓存的内存的最大大小 （以 KB 为单位） 。当值为 0 或 DISABLE 时，不使用编码器缓存。当该值太低而无法处理最小运动估计范围时，编码器将失败并显示错误消息。 允许的值：DISABLE 或正整数 默认值：DISABLE
AspectRatio	选择要在 SPS/VUI 中写入的视频序列的显示宽高比 允许的值： ASPECT_RATIO_AUTO 4:3 用于普通标清视频， 16:9 用于普通高清视频，未指定用于未知格式。 ASPECT_RATIO_4_3 4:3 宽高比 ASPECT_RATIO_16_9 16:9 宽高比 ASPECT_RATIO_NONE 流中不存在宽高比信息
LookAhead	LookAhead 是 LookAhead 的大小 （两次传递之间的帧数）： 0：LookAhead 禁用 高于 2：SceneChange 检测和校正 高于 10：使用帧复杂度+场景变换检测的常数质量

注释：

1. 使用 GStreamer 时，不支持以下提到的 gop-length 和 b-frames 组合 （但这些组合在实时场景中多数未使用）。({2, 1}、 {3, 2}、 {4, 2}、 {4, 3}、 {5, 3}、 {5, 4}、 {6, 3}、 {6, 4}、 {7, 4}、 {8, 4}、 {9, 3}、 {11, 4}、 {12, 4}、 {16, 4})。
2. 使用 GStreamer 时，由于赛灵思的规格，您无法使用“Closed GOP”结构。

运行参数

表 12-16：编码器运行部分参数

参数	描述和可能的值
Loop	指定编码器在到达文件末尾时是否应环回 YUV 输入流的开头。 允许的值：TRUE、 FALSE
MaxPicture	要编码的帧数 允许的值：大于或等于 1 的整数值、或 ALL 以到达 YUV 输入流的末尾。（ALL 不应与 Loop = TRUE 一起使用，否则编码器永远不会结束）。
FirstPicture	指定要编码的第一帧。 允许的值：介于 0 和输入 YUV 文件中的图像数之间的整数值。

量化参数 (QP) 文件格式

使用 QpCtrlMode = LOAD_QP or QpCtrlMode = LOAD_QP | RELATIVE_QP 启用 QP 表的模式。

在这种情况下，参考模型使用工作目录中的文件“QPs.hex”来指定 LCU 级的 QP 值。每行 QPs.hex 包含一个 8 位十六进制值。

对于 H.264 (AVC)，每 16×16 MB （光栅扫描格式）有一个字节：[0;51] 中的绝对 QP 或 [-32;31] 中的相对 QP。

对于 H.265 (HEVC)，每 32×32 LCU （光栅扫描格式）有一个字节：[0;51] 中的绝对 QP 或 [-32;31] 中的相对 QP。

注释：每个字节只有 6 个 LSB 用于 QP 或段 ID，保留 2 个 MSB。

例如，要指定以下相对 QP 表，

-1	-2	0	1	1	4
-4	-1	1	2	2	1
-1	0	-1	1	1	4
0	0	-2	2	2	6
1	-2	0	1	4	2

H.265 (HEVC) 的 QPs.hex 文件是：

```
3F
3E
00
01
01
04
3C
3F
...
```

每个 LCU 的相对 QP

如果工作文件夹中存在名称为 QP_<frame number>.hex 的文件，则编码器将其用于帧号 <frame number> 而不用于 QPs.hex。

例如，如果工作文件夹中有以下文件：

- QP_0.hex
- QP_4.hex
- QPs.hex

编码器对帧 #0 使用 QP_0.hex，对帧 #4 使用 QP_4.hex，对所有其他帧使用 QPs.hex（即帧 #1、#2、#3、#5、#6 ...）

加载 QP

Gstreamer 不支持使用 LoadQP 选项更新/加载 QP，但可以在控制软件级的直播中使用 loadQP。使用 `AL_Encoder_Process` API 将 QP 表与输入帧缓存一起传递。您必须在实时流中更新每个帧的 pQPTable 值。

bool AL_Encoder_Process （AL_HEncoder hEnc、 AL_TBuffer * pFrame、 AL_TBuffer * pQpTable）

将帧缓存推送到编码器。根据 GOP 模式，可以或不可以对该帧缓存立即编码。

参数

[in]	hEnc	编码器对象句柄
[in]	pFrame	指向要编码的帧缓存的指针 pFrame 缓存需要有一个关联的 L_TSrcMetaData 来说明 yuv 是如何存储在内存中的。在编码器使用缓存的内存期间，不应更改该内存。源元数据存在一些限制。色度间距必须等于亮度间距，而且必须为 32 位对齐，并且应该优于分辨率的最小支持间距（参见 AL_EncGetMinPitch()）。色度偏移不应落在亮度块内。应该与通道中的相同（参见 AL_EncGetSrcFourCC()）
[in]	pQpTable	返回

返回

如果函数成功，则返回值为非零 (true)，如果函数失败，则返回值为零 (false)

命令文件格式

编码器支持仿真动态事件的输入命令，如动态比特率和关键帧插入。在配置文件中引用命令文件时，将启用此功能。请参阅中的 “输入参数” CmdFile 参数。命令文件格式如下所述。文件的每一行定义一个帧标识符，后跟一个或多个应用于该帧的命令。

句法

<frame number>: <command> [, <command>]

其中 <command> 是以下之一：

命令	描述
SC	场景变化
KF	KeyFrame （使用 IDR 重新启动新的 GOP）
LT	将下一个编码图像设置为长期参考图像
UseLT	长期使用参考图像
GopLen=<length>	改变 Gop 长度
NumB=<numB>	更改连续 B 帧的数量
BR=<Kbits/s>	更改目标比特率
Fps=<frames/s>	更改帧速率

注释： KF 等关键字区分大小写。

例如：

```
  0: LT
20: KF
  30: BR =100, GopLen = 10
  45: SC
  50: UseLT
  101: GopLen= 20, NumB =1
```

ROI 文件格式

对感兴趣区域的定义以指定帧标识符、背景质量和重叠区域的 ROI 顺序行开始，后面跟着一行或多行 （每行对该帧的 ROI 及后续帧进行定义，直到下一个 ROI 定义）。

句法

frame <*index*>: [BkgQuality=<*quality*>, Order=<*order*>]

<*posX*> :<*posY*>, <*width*>x<*height*>, <*quality*>

质量	描述	使用的 Delta QP
HIGH_QUALITY	质量高于速率控制给定值	-5
MEDIUM_QUALITY	质量与速率控制给定值相同	0
LOW_QUALITY	质量低于速率控制给定值	+5
NO_QUALITY	不感兴趣的地区的质量可能更糟糕	+31 （最大 delta QP 值）

其中 <posX>、 <posY>、 <width> 和 <height> 以像素为单位。然后它们会自动舍入到边界 LCU 单元 （AVC 为 16×16， HEVC 为 32×32）。 <quality> 是以下之一：

<order> 是以下值之一：

顺序	描述
QUALITY_ORDER	使用质量最好的地区的质量
INCOMING_ORDER	使用最早定义区域的质量

注释： KF 等关键字区分大小写。

这是一个感兴趣区域的示例文件。

```
frame 0: BkgQuality= MEDIUM _QUALITY, Order=INCOMING_ORDER
6:4, 8x4, LOW_QUALITY
11:9, 5x5, HIGH_QUALITY
20:15, 5x7, MEDIUM_QUALITY
frame 13:
12:8, 7x5, HIGH_QUALITY
14:8, 5x5, NO_QUALITY
```

VCU 控制软件 API

VCU 控制软件 API 由用户空间中的编码器库和解码器库组成，用户空间应用与用户空间链接以控制 VCU 硬件。

常见

检查错误和报告错误

VCU 控制软件 API 中有几种错误处理机制。最常见的机制是函数返回状态值，例如布尔值或失败情况下为 NULL 的指针。

编码器和解码器对象分别存储要用 `AL_Encoder_GetFrameError` 和 `AL_Decoder_GetFrameError` 访问的错误代码。

用户定义的回调有时会通过为一个通常不为 NULL 的指针传递 NULL 或不提供任何通知而通知异常情况，但假设回调本身使用其中一个访问器函数从编码器对象或解码器对象中检索错误状态。

在异常或意外情况下，某些功能可能会直接向控制台报告错误。这些是系统错误，而这些消息包含表 12-17 中的消息。

表 12-17：编码器和解码器的错误消息

错误类型	描述
AL_SUCCESS	操作成功 （没有遇到任何错误）
AL_ERROR	未知错误
AL_ERR_RESOLUTION_CHANGE	不支持分辨率更改
AL_ERR_NO_MEMORY	没有内存，所以无法分配资源。这可以是 dma 内存，mcu 特定内存 （如果可用）或只是虚拟内存不足
AL_ERR_STREAM_OVERFLOW	生成的流无法放入分配的流缓存

表 12-17：编码器和解码器的错误消息 （续）

错误类型	描述
AL_ERR_TOO_MANY_SLICES	如果支持 SliceSize 模式，则无法遵守约束，因为需要太多的 slice 才能遵守约束
AL_ERR_CHAN_CREATION_NO_CHANNEL_AVAILABLE	调度程序无法处理更多通道（AL_SCHEDULER_MAX_CHANNEL 的固定限制）
AL_ERR_CHAN_CREATION_RESOURCE_UNAVAILABLE	可用核的处理能力不足以处理此通道
AL_ERR_CHAN_CREATION_NOT_ENOUGH_CORES	无法将负载分散到足够的核（ERROR_RESOURCE_UNAVAILABLE 的特殊情况）或负载无法传播太多（每个核对能处理的最少资源数均有要求）
AL_ERR_REQUEST_MALFORMED	请求的某些参数值无效
AL_ERR_CMD_NOT_ALLOWED	某些配置中不允许使用动态命令
AL_ERR_INVALID_CMD_VALUE	与该命令关联的值无效（在当前配置中）

有各种方式可能会破坏编码比特流，而且检测压缩比特流中的错误很复杂，尤其是因为语法元素编码和解析依存而导致的错误。错误通常在损坏的位上检测不到，但更可能在下面的语法元素上检测到。

例如，编码的比特流带有缩放矩阵，并且“缩放矩阵的存在位”在流中被破坏。当解码器读取该比特流时，它首先假设流中没有缩放矩阵并继续解析实际缩放矩阵数据，将其作为可能导致错误的下一语法元素。理想情况下，错误是缩放矩阵位被损坏，但解码器无法检测到，而且这种情况在视频编解码器中很常见。

以下错误被硬件 IP 检测到，此错误隐藏了 slice 的剩余部分；没有错误代码，只有单个标志指示 slice 是否已被隐藏。

```
End of CtB/MB flag
End of slice flag
Invalid Prediction Mode (AVC only)
Invalid RefIdx (AVC only)
MB Skip error (AVC only)
Out of range residual
```

有关错误处理和错误报告的更多详细信息，请参阅 VPS/SPS/PPS 解析功能：

https://github.com/Xilinx/vcu-ctrl-sw/tree/master/lib_parsing

`lib_parsing/AvcParser.c and lib_parsing/HevcParser.c and check the calls to the macro COMPLY.`

另外，监控 `lib_parsing/SliceHdrParsing.c` 中的 `AL_AVC_ParseSliceHeader` 和 `AL_HEVC_ParseSliceHeader` 函数，并检查返回错误路径。

存储器管理

内存操作通过函数指针变为间接处理。 AL_Allocator 默认实现只包含 malloc 和 free 等。

两种更高级的技术可用于内存管理：引用计数缓存和缓存池。使用零引用计数创建引用计数缓存。 AL_Buffer_Ref 和 AL_Buffer_Unref 函数分别递增和递减引用计数。 AL_Buffer 接口会将对缓存元数据的管理跟对与缓存关联的数据存储器的管理分开。引用计数的使用为可选项。

AL_TBufPool 实现对带有环形缓存的缓存池进行管理。在编译时，某些环形缓存的大小是固定的。超过缓存池大小会导致未定义的行为。请查阅 AL_Decoder_PutDisplayPicture。

结构

AL_Buffer

Description

保存由 AL_TAllocator 对象管理的内存的句柄。当引用计数递减为零时提供要调用的引用计数，互斥锁和回调。引用计数使用为可选项。这种类型是不透明的。实现使用 al_t_BufferImpl。

查看

BufferAPI.h, al_t_BufferImpl

函数

void AL_Buffer_Ref(AL_TBuffer* hBuf)

描述

将 hBuf 的引用计数增加 1。

查看

AL_Buffer_Create_And_Allocate, AL_Buffer_WrapData

函数

void AL_Buffer_Unref(AL_TBuffer* hBuf)

描述

将 hBuf 的引用计数增加 1。如果引用计数为零，则调用与缓存关联的 pCallback 函数。

请参阅

AL_Buffer_Create_And_Allocate、 AL_Buffer_WrapData

结构

AL_TAllocator

描述

AL_TAllocator 类型使开发者能够包装或替换内存管理功能以进行内存跟踪、备用 DMA 缓存处理等。

类型	字段	描述
const AL_AllocatorVtable*	vtable	指向内存管理功能的函数指针的指针。

请参阅

AL_AllocatorVtable

结构

AL_AllocatorVtable

描述

AL_TAllocator 类型提供内存管理功能。

类型	字段	描述
函数指针(1)	pfnDestroy	释放与分配器关联的资源。注意：此函数与分配器有关，而不是分配的句柄。
函数指针(2)	pfnAlloc	分配给定大小的句柄。如果分配失败，则返回 NULL。
函数指针(3)	pfnFree	释放与 pfnAlloc 返回的句柄关联的资源。如果成功则返回 true，否则返回 false。
函数指针(4)	pfnGetVirtualAddr	将给定句柄转换为虚拟地址。
函数指针(5)	pfnGetPhysicalAddr	将给定句柄转换为物理地址。
函数指针(6)	pfnAllocNamed	分配给定名称的缓存。该名称适用于开发者代码，不在内部用于 VCU 编码器 API 或 VCU 解码器 API。

注释：

1. bool (*pfnDestroy)(AL_TAllocator*)
2. AL_HANDLE (*pfnAlloc)(AL_TAllocator*, size_t)
3. bool (*pfnFree)(AL_TAllocator*, AL_HANDLE)
4. AL_VADDR (*pfnGetVirtualAddr)(AL_TAllocator*, AL_HANDLE)
5. AL_PADDR (*pfnGetPhysicalAddr)(AL_TAllocator*, AL_HANDLE)
6. AL_HANDLE (*pfnAllocNamed)(AL_TAllocator*, size_t, char const* name)

函数

void AL_Buffer_SetData(const AL_TBuffer* hBuf, uint8_t* pData)

描述

将 pData 分配给缓存 hBuf 的数据指针。

函数

AL_HANDLE AL_Allocator_Alloc(AL_TAllocator* pAllocator, size_t zSize)

返回

返回指向新分配内存的指针，如果分配失败则返回 NULL。

函数

AL_VADDR AL_Allocator_GetVirtualAddr(AL_TAllocator* pAllocator, AL_HANDLE hBuf)

返回值

返回 hBuf。

请参阅

LinuxDma_GetVirtualAddr、 LinuxDma_Map、 AL_Allocator_GetPhysicalAddr

函数

AL_PADDR AL_Allocator_GetPhysicalAddr(AL_TAllocator* pAllocator, AL_HANDLE hBuf)

返回值

返回 NULL。

请参阅

AL_Allocator_GetVirtualAddr

枚举

AL_EMetaType

描述

枚举常数	值	描述
AL_META_TYPE_SOURCE	0	源缓存
AL_META_TYPE_STREAM	1	流缓存
AL_META_TYPE_CIRCULAR	2	流缓存
AL_META_TYPE_PICTURE	3	图像缓存
AL_META_TYPE_EXTENDED	0x7F000000	扩展缓存

请参阅

BufferMeta.h

函数

AL_TStreamMetaData* AL_StreamMetaData_Create(uint16_t uMaxNumSection)

描述

分配流元数据对象。元数据对象与缓存相关联，并提供有关如何处理缓存的相关内容。 uMaxNumSection 实参确定要分配的节结构数。节结构将标志与缓存区域相关联，正如 AL_StreamMetaData_AddSection 所设置的那样。元数据对象必须与一个 AL_TBuffer 对象相关联。解除分配缓存的函数 （例如 AL_Buffer_Destroy）会调用每个元数据对象的 destroy 函数。

返回

返回指向能够指定 uMaxNumSection 节的 AL_TStreamMetaData 结构的指针，除非内存分配失败 （在这种情况下则会返回 NULL）。

请参阅

AL_StreamMetaData_ClearAllSections、 AL_StreamMetaData_AddSection、 AL_Buffer_AddMetaData、 AL_Buffer_Destroy、 AL_MAX_SECTION、 BufferStreamMeta.h

函数

void AL_StreamMetaData_ClearAllSections(AL_TStreamMetaData* pMetaData)

描述

将节数设置为零。

请参阅

AL_StreamMetaData_AddSection

函数

uint16_t AL_StreamMetaData_AddSection(AL_TStreamMetaData* pMetaData, uint32_t uOffset, uint32_t uLength, uint32_t uFlags)

描述

分配定义新的一节的参数。 uOffset 和 uLength 以字节表示。如果该区域超出缓存的末尾，则可能导致内存损坏。uFlags 实参是一个位字段。请查阅 AL_Section_Flags 的枚举值。

返回值

返回添加章节的章节 ID （索引）。

请参阅

AL_ StreamMetaData_ClearAllSectionData、 AL_StreamMetaData_ChangeSection、AL_StreamMetaData_SetSectionFlags、 AL_StreamMetaData_Create、 AL_Section_Flags

枚举

AL_Section_Flags

描述

章节标志用于指定适用于缓存区域的属性。

常数	值	章节属性
SECTION_SYNC_FLAG	0x40000000	来自 IDR 的数据
SECTION_END_FRAME_FLAG	0x20000000	帧结束
SECTION_CONFIG_FLAG	0x10000000	SPS、 PS、 VPS、或 AUD

请参阅

AL_StreamMetaData_AddSection

函数

bool AL_Buffer_AddMetaData(AL_TBuffer* pBuf, AL_TMetaData* pMeta)

描述

将 pMeta 指针添加到 pBuf 的元数据中。元数据对象与缓存相关联，并提供有关如何处理缓存的相关内容。不建议添加给定类型的两个或多个元数据。

返回值

成功时返回 true。如果内存分配失败，则返回 false。 Thread-safe。

请参阅

AL_Buffer_GetMetaData、 AL_Buffer_RemoveMetaData

函数

AL_TMetaData* AL_Buffer_GetMetaData(AL_TBuffer* pBuf, AL_EMetaType eType)

返回值

获取 eType 类型的第一个 pBuf 元数据。如果未找到匹配的元数据，则返回 NULL。 Thread-safe。

请参阅

AL_Buffer_AddMetaData、 AL_Buffer_RemoveMetaData

函数

bool AL_Buffer_RemoveMetaData(AL_TBuffer* pBuf, AL_TMetaData* pMeta)

描述

将 pMeta 指针从 pBuf 元数据中移除。

返回值

始终返回 true。 Thread-safe。

请参阅

AL_Buffer_GetMetaData、 AL_Buffer_AddMetaData

函数

AL_TBuffer* AL_Buffer_Create_And_Allocate(AL_TAllocator* pAllocator, size_t zSize, PFN_RefCount_CallBack pCallBack)

描述

分配并初始化一个能够保存 zSize 字节的缓存。使用 AL_Buffer_Destroy 释放缓存。线程是安全的。在缓存引用计数达到零之后调用 pCallBack，而且可以安全地重用缓存。注意：使用引用计数为可选项。 AL_Buffer 不会获取与其 AL_HANDLE 关联的内存的所有权。使用 AL_Allocator_Free 删除 AL_HANDLE 内存，然后使用 AL_Buffer_Destroy 删除缓存本身。

返回值

返回指向缓存的指针。

请参阅

AL_Buffer_Ref、 AL_Buffer_Unref、 AL_Buffer_SetUserData、 AL_Buffer_Destroy、 AL_Buffer_Create、 AL_Buffer_Create_And_Allocate_Named

函数

void AL_Buffer_Destroy(AL_TBuffer* pBuf)

描述

释放与缓存 pBuf 相关的内存，包括元数据。在调用此函数之前， pBuf 引用计数必须为零。不取消分配用于数据存储器的 AL_HANDLE。该内存由调用程序单独管理。例如，可以使用引用计数回调来释放。

请参阅

AL_Allocator_Free

函数

AL_TBuffer* AL_Buffer_WrapData(uint8_t* pData, size_t zSize, PFN_RefCount_CallBack pCallBack)

描述

分配指向 pData、引用计数为零的缓存。调用程序应该调用 AL_Buffer_Ref 以增加引用计数。当引用计数递减到零时，将调用 pCallBack。

返回值

如果成功，则返回缓存。否则,返回 NULL。

请参阅

AL_Buffer_Ref、 AL_Buffer_Unref

函数

bool AL_Allocator_Free(AL_TAllocator* pAllocator, AL_HANDLE hBuf)

描述

释放与缓存 hBuf 相关的内存。

请参阅

AL_TAllocator

函数

AL_TAllocator* DmaAlloc_Create(const char* deviceFile)

描述

打开 deviceFile 并分配一个小结构来记录器件名称、文件描述符和操作分配器的函数指针。 deviceFile 必须 `/dev/allegroIP`。 DmaAlloc_Create 不获取器件文件字符串的所有权。当不再需要分配器时，应调用 AL_Allocator_Destroy 以释放相关的系统资源。

返回值

如果成功，则返回指向分配器的指针，否则，返回 NULL。

请参阅

AL_Allocator_Destroy

函数

bool AL_Allocator_Destroy(AL_TAllocator* pAllocator)

描述

关闭基础文件描述符并释放相关资源。当不再需要给定的内存分配器时调用此函数；在销毁编码器或解码器实例时调用 （该函数）。调用此函数后，分配器的 alloc 函数先前返回的所有句柄都应视为无效。

返回值

如果成功则返回 true，否则，返回 false。

请参阅

DmaAlloc_Create

函数

int AL_GetAllocSize_DecReference(AL_TDimension tDim, AL_EChromaMode eChromaMode, uint8_t uBitDepth, AL_EFbStorageMode eFrameBufferStorageMode)

[in] tDim	框架维度 （宽度、高度），以像素为单位
[in] eChromaMode	色度采样模式
[in] uBitDepth	图像样本的位大小
[in] eStorageMode	帧缓存的存储模式

描述

检索参考帧缓存的大小。

返回值

返回参考帧缓存所需的大小 （以字节为单位）。

函数

int AL_CalculatePitchValue(int iWidth, uint8_t uBitDepth, AL_EFbStorageMode eStorageMode)

[in] iWidth	帧宽 （以像素为单位）
[in] uBitDepth	YUV 位深度
[in] eStorageMode	源存储模式

返回值

根据源格式返回间距 (pitch) 值。假设 32 位突发对齐。

请参阅

AL_GetBitDepth、 GetStorageMode

数据格式转换

提供了许多图像数据转换功能。所有数据都将 AL_TBuffer 转换为 AL_TBuffer。因为行为从函数名称中显而易见并且参数类型是不变的，所以在一个示例之后，这些函数在下面以表格形式列出，每个 （行、列）条目与一个 （源、目的地）对相对应。

函数

void I0AL_To_I420(AL_TBuffer const* pSrc, AL_TBuffer* pDst)

从	I0AL	I2AL	I420	I422	IYUV	NV12	NV16	P010	P210	RX0A	RX2A	RXmA	T608	T60A	T628	T62A	T6m8	Y010	Y800	YV12
I0AL			●		●	●		●		●								●	●	●
I2AL							●		●		●									
I420	●				●	●		●		●								●	●	●
I422							●		●		●									
IYUV	●		●			●		●		●									●	●
NV12	●		●		●			●		●									●	●
NV16		●		●					●		●									
P010	●		●		●	●		●										●	●	●
P210		●		●																
RX0A	●		●		●	●		●										●	●	●
RX2A		●		●			●		●											
T608	●		●		●	●		●										●	●	●
T60A	●		●		●	●		●										●	●	●
T628		●		●			●		●									●	●	
T62A		●		●			●		●									●	●	
T6m8			●																	
Y010										●		●								
Y800	●		●		●	●		●		●		●						●	●	●
YV12	●		●		●	●		●		●									●	

常数

config.h

名称	值	描述
AVC_MAX_HORIZONTAL_RANGE_P	16	P slice 的 AVC 最大水平范围
AVC_MAX_VERTICAL_RANGE_P	16	P slice 的 AVC 最大垂直范围

名称	值	描述
AVC_MAX_HORIZONTAL_RANGE_B	8	B slice 的 AVC 最大水平范围
AVC_MAX_VERTICAL_RANGE_B	8	B slice 的 AVC 最大垂直范围
HEVC_MAX_HORIZONTAL_RANGE_P	32	P slice 的 HEVC 最大水平范围
HEVC_MAX_VERTICAL_RANGE_P	32	P slice 的 HEVC 最大垂直范围
HEVC_MAX_HORIZONTAL_RANGE_B	16	B slice 的 HEVC 最大水平范围
HEVC_MAX_VERTICAL_RANGE_B	16	B slice 的 HEVC 最大垂直范围
ENCODER_CORE_FREQUENCY	666,666,666	666.67 MHz
ENCODER_CORE_FREQUENCY_MARGIN	10	10 Hz
ENCODER_CYCLES_FOR_BLK_32X32	4,900	内部使用
AL_ENC_NUM_CORES	4	编码器核数
AL_DEC_NUM_CORES	2	解决码器核数
HW_IP_BIT_DEPTH	10	10 bpc
AL_CORE_MAX_WIDTH	4096	内部使用
AL_L2P_MAX_SIZE	4,259,840	物理内存可用于 2 级预取高速缓存（以字节为单位）
AL_MAX_ENC_SLICE	200	编码器使用的最大 slice 数
AL_BLK16X16_QP_TABLE	0	内部使用

函数

AL_EChromaMode AL_GetChromaMode(TFourCC tFourCC)

描述

执行以下映射。

FourCC	色度模式
I420、 IYUV、 YV12、 NV12、 I0AL、 P010、 T608、 T60A、 T508、 T50A、 RX0A	4:2:0
YV16、 NV16、 I422、 P210、 I2AL、 T628、 T62A、 T528、 T52A、 RX2A	4:2:2
Y800、 Y010、 T6m8、 T6mA、 T5m8、 T5mA、 RXmA	Monochrome

返回值

返回给定 tFourCC 实参的色度模式。如果未定义 FourCC 模式，则发生断言违例或返回 -1。

请参阅

AL_EChromaMode, FOURCC

宏

FOURCC(*A*)

描述

通过将 A 的文字字符转换为其 ASCII 码并将其打包成 32 位值 （仅为一例）将 A 转换为 FourCC 值。 FOURCC(A321) 即会变为 0x33 32 31 41。

枚举

AL_EPicFormat

描述

常数	值	亮度	色度	位深度	色度模式	描述
AL_400_8BITS	0x0088	8	8	8	0	4:0:0、 8-bpc
AL_420_8BITS	0x0188	8	8	8	1	4:2:0、 8-bpc
AL_422_8BITS	0x0288	8	8	8	2	4:2:2、 8-bpc
AL_444_8BITS	0x0388	8	8	8	3	4:4:4、 8-bpc
AL_400_10BITS	0x00AA	10	10	10	0	4:0:0、 10-bpc
AL_420_10BITS	0x01AA	10	10	10	1	4:2:0、 10-bpc
AL_422_10BITS	0x02AA	10	10	10	2	4:2:2、 10-bpc
AL_444_10BITS	0x03AA	10	10	10	3	4:4:4、 10-bpc

请参阅

AL_GET_BITDEPTH_LUMA、 AL_GET_BITDEPTH_CHROMA、 AL_GET_BITDEPTH、 AL_GET_CHROMA_MODE、 AL_SET_BITDEPTH_LUMA、 AL_SET_BITDEPTH_CHROMA、 AL_SET_BITDEPTH、 AL_SET_CHROMA_MODE

宏

AL_GET_BITDEPTH_LUMA(PicFmt)

返回值

从 PicFmt 返回亮度深度。 PicFmt 必须是 AL_EPicFormat 值。

请参阅

AL_EPicFormat

宏

AL_GET_BITDEPTH_CHROMA(PicFmt)

返回值

从 PicFmt 返回色度深度。 PicFmt 必须是 AL_EPicFormat 值。

请参阅

AL_EPicFormat

宏

AL_GET_BITDEPTH(PicFmt)

返回值

从 PicFmt 返回亮度深度和色度深度的最大值。 PicFmt 必须是 AL_EPicFormat 值。

请参阅

AL_EPicFormat

宏

AL_GET_CHROMA_MODE(PicFmt)

返回值

从 PicFmt 返回色度模式。 PicFmt 必须是 AL_EPicFormat 值。

请参阅

AL_EPicFormat

宏

AL_SET_BITDEPTH_LUMA(PicFmt, BitDepth)

返回值

将 BitDepth 分配给 PicFmt 的低位字节 （字节 0）。 PicFmt 必须是 AL_EPicFormat 值。

请参阅

AL_EPicFormat

宏

AL_SET_BITDEPTH_CHROMA(PicFmt, BitDepth)

返回值

将 BitDepth 分配给 PicFmt 的字节 1。 PicFmt 必须是 AL_EPicFormat 值。

请参阅

AL_EPicFormat

宏

AL_SET_BITDEPTH(PicFmt, BitDepth)

返回值

将 BitDepth 分配给 PicFmt 的字节 0 和字节 1。 PicFmt 必须是 AL_EPicFormat l-value。

请参阅

AL_EPicFormat

宏

AL_SET_CHROMA_MODE(PicFmt, BitDepth)

返回值

将 BitDepth 分配给 PicFmt 的字节 2。 PicFmt 必须是 AL_EPicFormat l-value。

请参阅

AL_EPicFormat

函数

uint8_t AL_GetBitDepth(TFourCC tFourCC)

描述

执行以下映射。

FourCC	比特深度
I420、IYUV、YV12、NV12、I422、YV16、NV16、Y800、T6m8、T608、T628、T5m8、T508、T528	8-bpc
I0AL、P010、I2AL、P210、Y010、T6mA、T60A、T62A、T5mA、T50A、T52A、RX0A、RX2A、RXmA	10-bpc

返回值

根据给定的 FourCC 值返回 8 或 10。如果未定义 FourCC 模式，则发生置位违例或返回 -1。

请参阅

FOURCC

函数

int GetPixelSize(uint8_t uBitDepth)

返回值

如果 uBitDepth 为 8 或更小，则返回 1。如果 uBitDepth 为 9 或更大，则返回 2。

函数

AL_EFbStorageMode GetStorageMode(TFourCC tFourCC)

返回值

根据 tFourCC 实参，返回 AL_FB_TILE_32x4 或 AL_FB_TILE_64x4。

请参阅

AL_Is32x4Tiled、 AL_Is64x4Tiled

函数

AL_EFbStorageMode AL_GetSrcStorageMode(AL_ESrcMode eSrcMode)

描述

执行以下映射：

源压缩模式	存储模式
AL_SRC_TILE_64x4、 AL_SRC_COMP_64x4	AL_FB_TILE_64x4
AL_SRC_TILE_32x4、 AL_SRC_COMP_32x4	AL_FB_TILE_32x4
其他	AL_FB_RASTER

请参阅

AL_EFbStorageMode, AL_ESrcMode

函数

int GetNumLinesInPitch(AL_EFbStorageMode eFrameBufferStorageMode)

描述

执行以下映射：

存储模式	Pitch 行数
AL_FB_TILE_64x4	4
AL_FB_TILE_32x4	4
AL_FB_RASTER	1

请参阅

AL_EFbStorageMode, AL_ESrcMode

宏

AL_IS_AVC(Prof)

描述

如果 Prof 是 AVC，则返回 true。 Prof 必须是 AL_EProfile 值。

请参阅

AL_EProfile

宏

AL_IS_HEVC(Prof)

描述

如果 Prof 是 HEVC，则返回 true。 Prof 必须是 AL_EProfile 值。

请参阅

AL_EProfile

宏

AL_IS_STILL_PROFILE(Prof)

描述

如果 Prof 是 HEVC Main Still Profile，则返回 true。 Prof 必须是 AL_EProfile 值。

请参阅

AL_EProfile

函数

bool AL_IsSemiPlanar(TFourCC tFourCC)

描述

如果 tFourCC 是块格式或以下之一：NV12、 P010、 NV16、 P210、 RX0A，或 RX2A，则返回 true。

请参阅

AL_Is64x4Tiled、 AL_Is32x4Tiled

函数

bool AL_IsTiled(TFourCC tFourCC)

描述

如果 tFourCC 是块格式，则返回 true。

请参阅

AL_Is64x4Tiled、 AL_Is32x4Tiled

函数

bool AL_Is32x4Tiled(TFourCC tFourCC)

描述

如果 tFourCC 是块格式或以下之一：T508、 T528、 T5m8、 T50A、 T52A 或 T5mA，则返回 true。

请参阅

AL_IsTiled、 AL_Is64x4Tiled

函数

bool AL_Is64x4Tiled(TFourCC tFourCC)

描述

如果 tFourCC 是块格式或以下之一：T608、 T628、 T6m8、 T60A、 T62A 或 T6mA，则返回 true。

请参阅

AL_IsTiled、 AL_Is32x4Tiled

函数

bool AL_Is10bitPacked(TFourCC tFourCC)

描述

如果 tFourCC 是块格式或以下之一：RX0A、 RX2A 或 RXmA。

函数

void AL_GetSubsampling(TFourCC fourcc, int* sx, int* sy)

描述

将 FourCC 模式的子采样写入 *sx* 和 *sy*，如下面的映射所示。

色度模式	sx	sy
4:2:2	2	2
4:2:0	2	1
其他	1	1

函数

TFourCC AL_EncGetSrcFourCC(AL_TPicFormat const picFmt)

描述

执行以下映射：如果下面未列出图像格式 picFmt，则会发生置位违规或返回 -1。

存储块模式	色度模式	BPC	FourCC
64×4	4:2:2	8	T628
		10	T62A
	4:2:0	8	T608
		10	T60A
	Mono	8	T6m8
		10	T6mA
32×4	4:2:2	8	T528
		10	T52A
	4:2:0	8	T508
		10	T50A
	Mono	8	T5m8
		10	T5mA

请参阅

AL_GetSrcFourCC、 Get64x64FourCC、 Get32x4FourCC

函数

Driver* AL_GetHardwareDriver()

描述

硬件驱动是包含函数指针的静态结构体。在创建编码器时打开器件。

返回值

返回指向驱动函数指针结构的指针。

请参阅

AL_SchedulerMcu_Create、 AL_Encoder_Create

函数

TScheduler* AL_SchedulerMcu_Create(Driver* driver, AL_TAllocator* pDmaAllocator)

描述

为调度程序分配和初始化内存。

返回值

如果成功则返回 true，否则，返回 false。

请参阅

AL_GetHardwareDriver、 DmaAlloc_Create、 AL_SchedulerMcu_Destroy

函数

TScheduler* AL_SchedulerMcu_Destroy(AL_TSchedulerMcu* schedulerMcu)

描述

释放与 schedulerMcu 相关的内存。

返回值

如果成功则返回 true，否则，返回 false。

请参阅

AL_Encoder_Create、 AL_SchedulerMcu_Create

函数

AL_ECodec AL_GetCodec(AL_EProfile eProf)

返回值

根据 eProf 返回 AL_CODEC_AVC 或 AL_CODEC_HEVC。

编码器流程

图 12-5 显示了使用 VCU 控制软件 API 的典型控制流程。

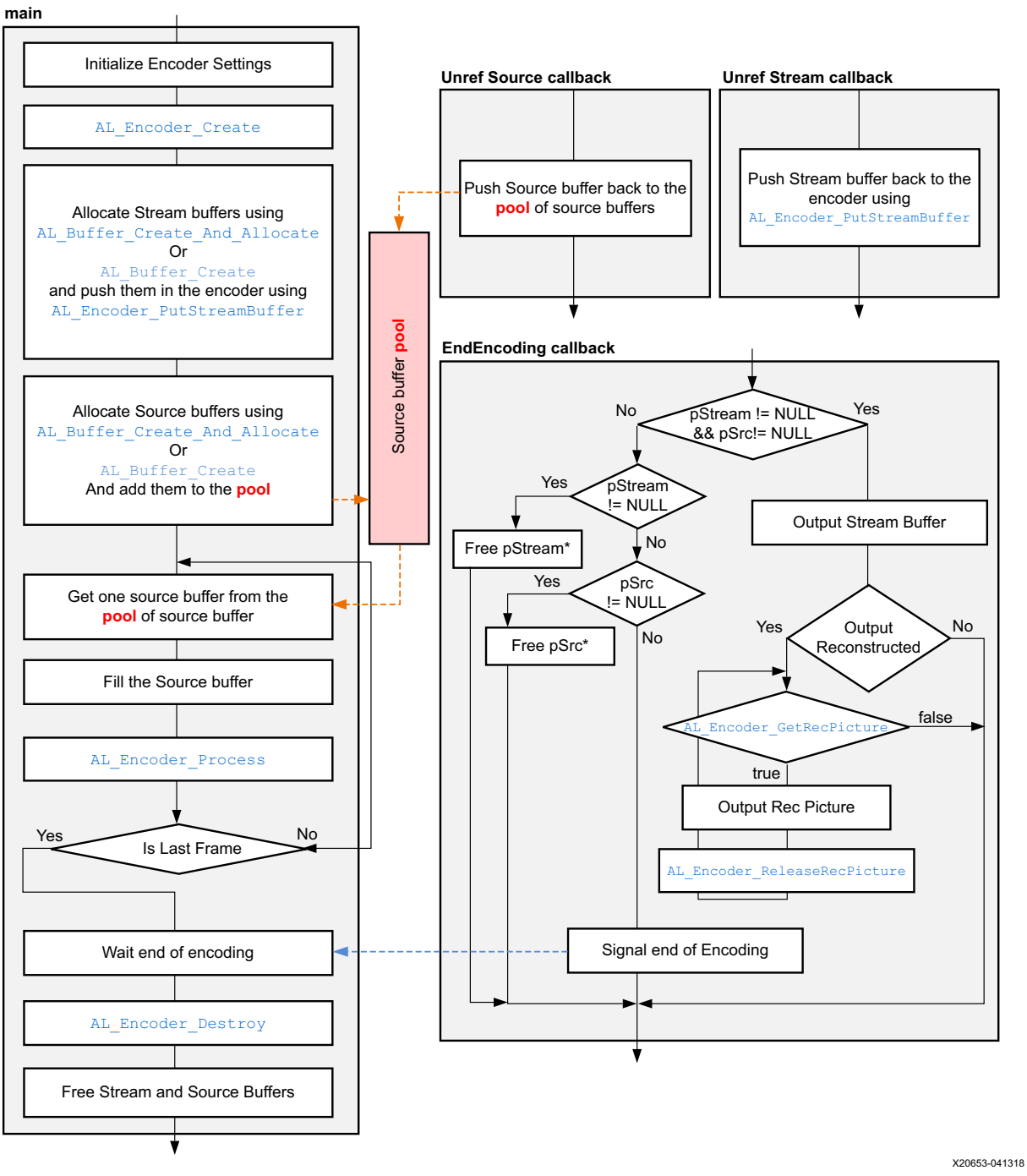

图 12-5：编码器 API 工作流程示例

编码器 API

在头文件中对编码器 API 进行了定义：

https://github.com/Xilinx/vcu-ctrl-sw/blob/master/include/lib_encode/lib_encoder.h

https://github.com/Xilinx/vcu-ctrl-sw/blob/master/include/lib_common_enc/Settings.h

https://github.com/Xilinx/vcu-ctrl-sw/blob/master/include/lib_common/StreamBuffer.h

对 API 的记录如下所述：

示例应用 `ctrlsw_encoder` 演示了如何使用 API。

函数

AL_ERR AL_Encoder_GetLastError(AL_HEncoder hEnc)

[in] hEnc 处理编码器

描述

从编码器中的内容结构返回错误代码。 Thread-safe。

错误代码	描述
AL_ERR_INIT_FAILED	由于参数不一致而无法初始化解码器
AL_ERR_CHAN_CREATION_NO_CHANNEL_AVAILABLE	通道未创建。无频道可用
AL_ERR_CHAN_CREATION_RESOURCE_UNAVAILABLE	通道未创建。可用核的处理能力不足
AL_ERR_CHAN_CREATION_NOT_ENOUGH_CORES	通道未创建。无法将负载分散到足够的核
AL_ERR_REQUEST_MALFORMED	通道未创建。请求格式不正确
AL_ERR_NO_FRAME_DECODED	没有帧解码
AL_ERR_RESOLUTION_CHANGE	不支持分辨率更改
AL_ERR_NO_MEMORY	检测到内存不足 （DMA、嵌入式内存或虚拟内存不足）
AL_SUCCESS	成功

请参阅

Error.h、 AL_Decoder_GetLastError

结构

AL_TEncSettings

描述

下面的章节将介绍编码器设置：

类型	字段	描述
AL_TEncChanParam	tChParam	
bool	bEnableAUD	启用单元分隔符存取功能。默认值：true
bool	bEnableFillerData	启用填充数据。默认值：true
uint32_t	uEnableSEI	启用补充增强信息。请参阅 AL_SeiFlag。默认值：SEI_NONE
AL_EAspectRatio	eAspectRatio	指定显示宽高比
AL_EColourDescription	eColourDescription	COLOUR_DESC_BT_709 = 1 COLOUR_DESC_BT_470_PAL = 5 (默认值)
AL_EScalingList	eScalingList	AL_SCL_FLAT = 0 AL_SCL_DEFAULT = 1 (默认值) AL_SCL_CUSTOM = 2 AL_SCL_RANDOM = 3
bool	bDependentSlice	
bool	bDisIntra	
bool	bForceLoad	默认值：true
int32_t	iPrefetchLevel2	CacheLevel2 以字节为单位。必须 ≥ 64 且 ≤ 图像宽度 （以像素为单位）
uint32_t	uL2PSize	
uint16_t	uClipHrzRange	2 级高速缓存水平范围。必须 ≥ 64 且 ≤ 图像宽度（以像素为单位）
uint16_t	uClipVrtRange	2 级高速缓存垂直范围。必须 ≥ 8 且 ≤ 图像高度（以像素为单位）
AL_EQpCtrlMode	eQpCtrlMode	质量参数控制模式。请查阅 AL_EQpCtrlMode。默认值：UNIFORM_QP
int	NumView	默认值：1
uint8_t	ScalingList[4][6][64]	
uint8_t	SclFlag[4][6]	
uint32_t	bScalingListPresentFlags	
uint8_t	DcCoeff[8]	
uint8_t	DcCoeffFlag[8]	
bool	bEnableWatchdog	未使用

请参阅

AL_Settings_SetDefaults、AL_Settings_SetDefaultParam、AL_Settings_CheckValidity、AL_Settings_CheckCoherency、AL_Encoder_Create、AL_Common_Encoder_CreateChannel、AL_ExtractNalsData、AL_AVC_SelectScalingList、AL_HEVC_SelectScalingList、AL_HEVC_GenerateVPS、AL_AVC_UpdateHrdParameters、AL_HEVC_UpdateHrdParameters、AL_AVC_GenerateSPS、AL_HEVC_GenerateSPS、AL_AVC_GeneratePPS、AL_HEVC_GeneratePPS、GetHevcMaxTileRow、ConfigureChannel、ParseRateControl、ParseGop、ParseSettings、ParseHardware、ParseMatrice (sic)、RandomMatrice (sic)、GenerateMatrice (sic)、ParseScalingListFile、PostParsingChecks、GetScalingListWrapped、GetScalingList

枚举

AL_EChEncOptions

描述

名称	值
AL_OPT_WPP	0x00000001
AL_OPT_TILE	0x00000002
AL_OPT_LF	0x00000004
AL_OPT_LF_X_SLICE	0x00000008
AL_OPT_LF_X_TILE	0x00000010
AL_OPT_SCL_LST	0x00000020
AL_OPT_CONST_INTRA_PRED	0x00000040
AL_OPT_QP_TAB_RELATIVE	0x00000080
AL_OPT_FIX_PREDICTOR	0x00000100
AL_OPT_CUSTOM_LDA	0x00000200
AL_OPT_ENABLE_AUTO_QP	0x00000400
AL_OPT_ADAPT_AUTO_QP	0x00000800
AL_OPT_TRANSFO_SKIP	0x00002000
AL_OPT_FORCE_REC	0x00008000
AL_OPT_FORCE_MV_OUT	0x00010000
AL_OPT_FORCE_MV_CLIP	0x00020000
AL_OPT_LOWLAT_SYNC	0x00040000
AL_OPT_LOWLAT_INT	0x00080000
AL_OPT_RDO_COST_MODE	0x00100000

函数

AL_ERR AL_Encoder_Create(AL_HEncoder* hEnc, TScheduler* pScheduler, AL_TAllocator* pAlloc,
AL_TEncSettings const* pSettings, AL_CB_EndEncoding callback)

[out] hEnc	新创建的编码器的句柄
[in] pScheduler	指向调度程序对象的指针
[in] pAlloc	指向 AL_TAllocator 接口的指针
[in] pSettings	指向 AL_TEncSettings 结构的指针 （用来指定编码器参数）
[in] callback	完成帧编码时调用的回调

描述

创建一个新编码器并返回一个句柄。创建编码器对象时，编码格式是固定的。对于编码 H.264 和 H.265 流的应用，要从一种编码切换到另一种编码，编码器对象必须被销毁并使用不同的设置重新创建。VCU LogiCORE 的参数在可编程逻辑比特流中是固定的，应该选择来用于最苛刻的情况。

返回值

成功后，返回 AL_SUCCESS。出错时，返回 AL_ERROR、AL_ERR_NO_MEMORY 或 ioctl 的返回值。AL_ERROR 可能表示内存分配失败、无法打开
`/dev/allegroIP`，或未能将消息发布到编码器。来自 ioctl 的错误返回代码表示与器件驱动的交互失败。

请参阅

DmaAlloc_Create、 AL_Settings_SetDefaultParam、 AL_SchedulerMcu_Create、 AL_GetHardwareDriver、 AL_Encoder_Destroy、 AL_Allocator_Destroy、 AL_CB_EndEncoding

结构

AL_CB_EndEncoding

描述

发生以下任何一种情况时，将调用此回调：

类型	字段	描述
Function pointer(1)	func	在帧编码时调用到达流的末尾 (EOS)，或释放流缓存
void*	userParam	用户定义

注释：

1. void (*func)(void* pUserParam, AL_TBuffer* pStream, AL_TBuffer const* pSrc)

函数

void AL_Settings_SetDefaults(AL_TEncSettings* pSettings)

描述

为与 HEVC Main profile、51 级、Main tier、4:2:0、每通道 8 位、GOP length = 32、Target Bit Rate = 4,000,000、frame rate = 30 等对应的 pSetting 分配值。要查看完整的设置列表，请参阅“Settings.c”中的 AL_Settings_SetDefaults。

请参阅

Settings.c

函数

void AL_Settings_SetDefaultParam(AL_TEncSettings* pSettings)

描述

如果 pSettings->tChParam.eProfile 是 AVC，将 pSettings->tChParam.uMaxCuSize 设置为 4。这相当于 16×16。如果 HEVC and pSettings->tChParam.uCabacInitIdc > 1，则将其设置为 1。

请参阅

AL_Settings_CheckValidity

函数

int AL_Settings_CheckValidity(AL_TEncSettings* pSettings, AL_TEncChanParam * pChParam, FILE* pOut)

描述

如果 pOut 为 非 NULL，则会将列出 pSettings 中的无效设置的信息写入 pOut。

返回值

返回在 pSettings 中找到的错误数。

请参阅

AL_Settings_CheckCoherency

函数

int AL_Settings_CheckCoherency(AL_TEncSettings * pSettings, AL_TEncChanParam * pChParam, TFourCC tFourCC, FILE * pOut)

描述

检查并更正 pSettings 中的某些编码器参数。如果 pOut 为非 NULL,则会将列出 pSettings 中的无效设置的信息写入 pOut。

可能会更正以下设置：

```
eQpCtrlMode
eScalingList
iPrefetchLevel2
tChParam.eEntropyMode
tChParam.eOptions
tChParam.ePicFormat
tChParam.eProfile
tChParam.iCbPicQpOffset
tChParam.iCbSliceQpOffset
tChParam.iCrPicQpOffset
tChParam.tGopParam.uFreqIDR
tChParam.tGopParam.uFreqLT
tChParam.tGopParam.uGopLength
tChParam.tGopParam.uNumB
tChParam.tRCParam.uCPBSize
tChParam.tRCParam.uInitialRemDelay
tChParam.tRCParam.uMaxBitRate
tChParam.tRCParam.uTargetBitRate
tChParam.uCuQPDeltaDepth
tChParam.uLevel
tChParam.uMaxCuSize
tChParam.uMaxTuSize
tChParam.uMinCuSize
tChParam.uNumSlices
tChParam.uTier
uL2PSize
```

注释：并非所有设置更正都伴有警告消息。

返回值

0 如果没有连贯性。

如果发现不连贯，则显示不连贯的数量。

-1 如果发现了致命的不连贯性。

请参阅

Settings.c、 AL_Settings_CheckValidity

函数

int AL_GetMitigatedMaxNalSize(AL_TDimension tDim, AL_EChromaMode eMode, int iBitDepth)

描述

减轻的最坏情况 NAL 大小是 PCM 加上每行一个 slice。

返回值

返回一个 NAL 单位的最大大小，向上舍入到最接近的 32 的倍数。

请参阅

AL_TDimension, AL_EChromaMode、 AL_GetMaxNalSize

结构

AL_TBufPoolConfig

描述

用于配置 AL_TBufPool 的结构。

类型	字段	描述
uint32_t	uNumBuf	池中的缓存数量
size_t	zBufSize	将填充池的缓存的大小 （以字节为单位）
char const*	debugName	用于区分调试器中缓存的字符串
AL_TMetaData*	pMetaData	添加到池中的每个缓存的元数据

请参阅

AL_SrcMetaData_Create

函数

bool AL_Encoder_PutStreamBuffer(AL_HEncoder hEnc, AL_TBuffer* pStream)

[in] hEnc	编码器对象句柄
[in] pStream	指向给定编码器的流缓存的指针

描述

告诉编码器在哪里写入编码比特流。对于给定的编码器，此函数的调用次数不得超过 320 次。 pStream 实参必须有与 AL_TStreamMetaData 指针关联的指针添加到它。元数据可以由 AL_StreamMetaData_Create(sectionNumber、 uMaxSize) 创建，并添加 AL_Buffer_AddMetaData(pStream、 pMeta)，但会受以下约束条件限制：

- sectionNumber = AL_MAX_SECTION，如果需要可以更大 （例如在流中添加 SEI）

- uMaxSize 应为 32 位对齐

注释：不对太多缓存或重复缓存执行错误检查。

返回值

返回 true。

请参阅

AL_StreamMetaData_Create、 AL_Buffer_AddMetaData、 AL_Encoder_Create、 AL_Encoder_Destroy

函数

bool AL_Encoder_Process(AL_HEncoder hEnc, AL_TBuffer* pFrame, AL_TBuffer* pQpTable)

[in] hEnc	编码器对象的句柄
[in] pFrame	指向要进行编码的帧缓存指针
[in] pQpTable	如果启用了外部质量参数模式，则指向质量参数表 （为可选项）的指针。请参阅 AL_TEncSettings.eQpCtrlMode

描述

将帧缓存推送到编码器。 GOP 模式确定帧是否可以立即编码。在编码期间，应用不得更改与 pFrame 关联的数据。pFrame 缓存必须有关联的 AL_TSrcMetaData，用于描述 YUV 数据是如何存储在内存中的。下列限制与源元数据相关：

- 色度间距和亮度间距必须相等
- 色度间距必须为 32 位对齐
- 色度间距应不小于分辨率支持的最小间距
- 色度偏移不应落在色度块内
- FourCC 应与通道匹配 （请参阅 AL_EncGetSrcFourCC()）。

返回值

如果成功则返回 true，否则，返回 false。调用 AL_Encoder_GetLastError 以获取错误状态代码。

请参阅

AL_Encoder_ReleaseFrameBuffer、 AL_TEncSettings、 AL_CB_EndEncoding

函数

void AL_Encoder_Destroy(AL_HEncoder hEnc)

描述

遍历编码器上下文 hEnc->pCtx 并删除和清除各种结构和字段。释放与编码器 hEnc.相关的内存。

枚举

AL_EBufMode

描述

表示某些缓存操作的阻塞或非阻塞模式。如果使用 AL_BUF_MODE_BLOCK 或 AL_BUF_MODE_NON_BLOCK 以外的值调用期望 AL_EBufMode 的函数，则行为未定义。

AL_EBufMode 描述	值
AL_BUF_MODE_BLOCK	0
AL_BUF_MODE_NONBLOCK	1

请参阅

AL_Decoder_PushBuffer、 AL_GetWaitMode

函数

int AL_Encoder_AddSei(AL_HEncoder hEnc, AL_TBuffer *pStream,bool isPrefix, int iPayloadType, uint8_t *pPayload, int iPayloadSize)

[in] hEnc	编码器的句柄
[in] pStream	拥有流元数据所需的流缓存
[in] isPrefix	区分前缀和后缀 SEI
[in] iPayloadType	SEI 有效载荷类型。请查阅 ITU-T [in] pPayload 的附录 D.3。 SEI 有效载荷的原始数据
[in] iPayloadSize	原始数据有效负载的大小

描述

向流中添加 SEI 应在编码器对比特流进行编码后调用此函数。流中 SEI 的最大最终大小不能超过 2Ko。 SEI 有效载荷不需要反模拟 （这将由编码器完成）。

返回值

返回节 id。

函数

void AL_Encoder_NotifyUseLongTerm(AL_HEncoder hEnc)

描述

通知编码器将使用长期参考帧。这可以提升背景不变的用例的背景 （例如固定摄像机监视）质量。

函数

void AL_Encoder_NotifyIsLongTerm(AL_HEncoder hEnc)

描述

通知编码器下一参考图像是长期参考图像。

函数

bool AL_Encoder_SetGopLength(AL_HEncoder hEnc, int iGopLength)

描述

通知编码器 GOP 长度已更改。如果正在进行的 GOP 比新的 iGopLength 长，编码器将立即重启 GOP。否则，编码器在达到新长度时重新启动 GOP。 iGopLength 实参必须介于 1 和 1,000 之间 （包括 1 和 1,000）。

返回值

如果成功，则返回 true。如果不成功，则返回 false。调用 AL_Encoder_GetLastError 以获取错误代码。

函数

bool AL_Encoder_SetGopNumB(AL_HEncoder hEnc, int iNumB)

描述

通知编码器 I 帧和 P 帧之间的连续 B 帧的数量。

返回值

如果成功，则返回 true。如果不成功，则返回 false。调用 AL_Encoder_GetLastError 以获取错误代码。

函数

bool AL_Encoder_SetBitRate(AL_HEncoder hEnc, int iBitRate)

描述

以每秒位数通知编码器目标比特率。

返回值

如果成功，则返回 true。如果不成功，则返回 false。调用 AL_Encoder_GetLastError 以获取错误代码。

函数

bool AL_Encoder_SetFrameRate(AL_HEncoder hEnc, uint16_t uFrameRate, uint16_t uClkRatio)

描述

指示编码器改变编码帧速率，计算如下。

fps = iFrameRate × 1,000 ÷ iClkRatio

例如，当 uFrameRate = 60 且 uClkRatio = 1,001 时，帧速率将为 59.94 fps。

返回值

如果成功，则返回 true。如果不成功，则返回 false。调用 AL_Encoder_GetLastError 以获取错误代码。

函数

bool AL_Encoder_SetQP(AL_HEncoder hEnc, int iQP)

描述

更改下一个推送帧的量化参数。

返回值

如果成功，则返回 true。如果不成功，则返回 false。调用 AL_Encoder_GetLastError 以获取错误代码。

函数

void AL_Encoder_ReleaseRecPicture(AL_HEncoder hEnc, TRecPic* pRecPic)

描述

释放先前通过 AL_Encoder_GetRecPicture 获得的重建的缓存。

解码器流程

图 12-6 显示的是使用 VCU 控制软件 API 的示例。

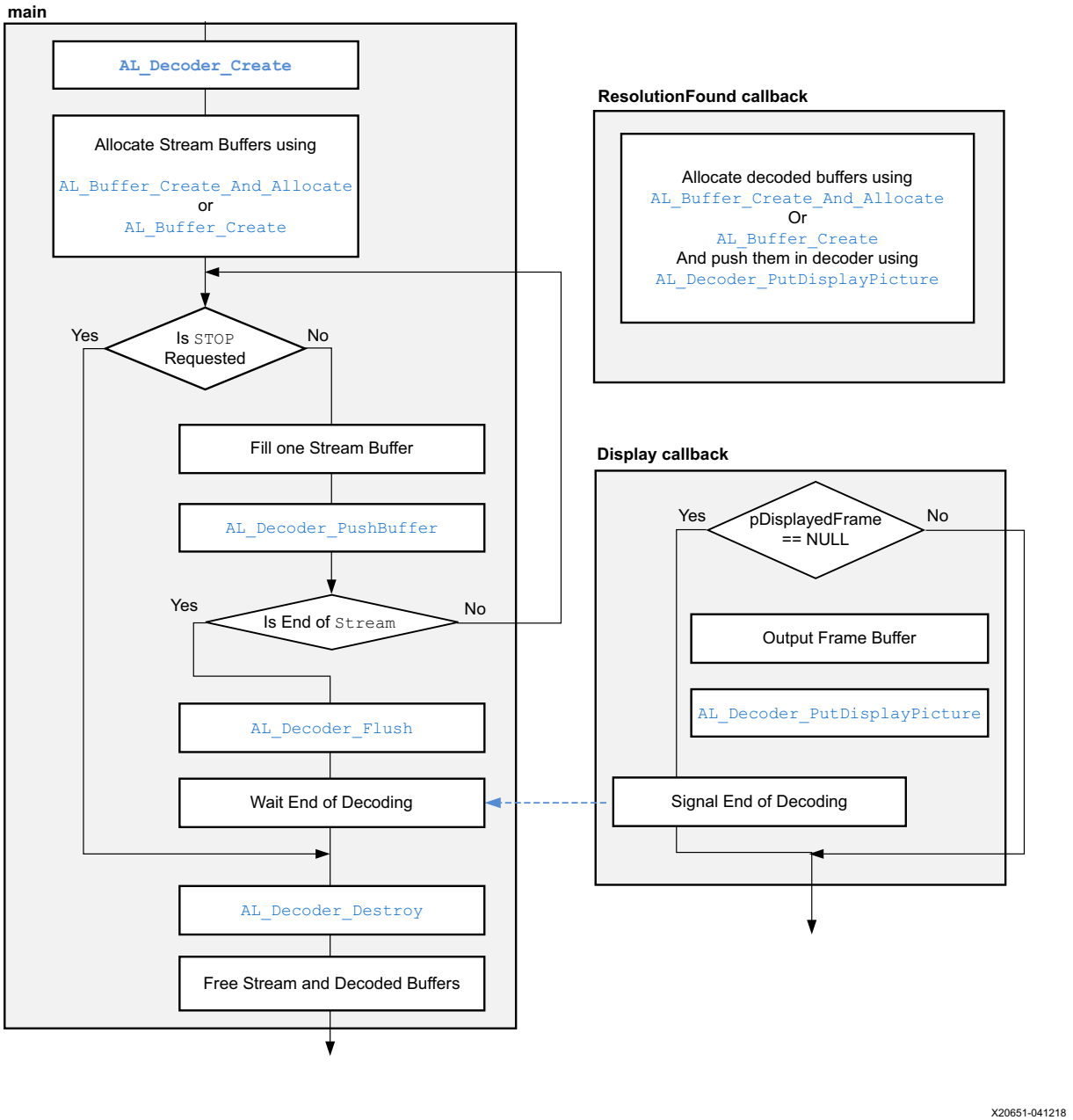

图 12-6：解码器 API 工作流程示例

解码器 API

解码器 API 定义在
https://github.com/Xilinx/vcu-ctrl-sw/blob/master/include/lib_decode。 API 记录如下。

示例应用 ctrlsw_encoder 演示了如何使用 API。

函数

AL_ERR AL_Decoder_Create(AL_HDecoder* hDec, AL_TIDecChannel* pDecChannel, AL_TAllocator* pAllocator, AL_TDecSettings* pSettings, AL_TDecCallBacks* pCB)

[out] hDec	新创建的解码器的句柄
[in] pDecChannel	指向调度程序结构的指针
[in] pAllocator	指向分配器的指针
[in] pSettings	指向解码器设置的指针
[in] pCB	指向编码器回调的指针

描述

创建解码器对象。 AL_TIDecChannel 对象由 AL_DecChannelMcu_Create 创建。 AL_TAllocator 对象由 DmaAlloc_Create 创建。 AL_TDecSettings 对象和 AL_TDecCallBacks 对象通过直接设置其字段进行初始化。

返回值

成功后，返回 AL_SUCCESS。否则，返回其中一个

AL_ERROR – 未识别的错误

AL_ERR_NO_MEMORY – 内存分配失败

AL_ERR_INIT_FAILED – 请参阅 CheckSettings

请参阅

AL_DecChannelMcu_Create、 `/dev/allegroDecodeIP`、 DmaAlloc_Create、 AssignSettings、 CheckSettings、 AL_TDecCallBacks

结构

AL_TDecCallBacks

描述

类型	字段	描述
AL_CB_EndDecoding	endDecodingCB	在帧被解码时调用
AL_CB_Display	displayCB	在准备好显示缓存时调用
AL_CB_ResolutionFound	resolutionFoundCB	在更改分辨率时调用
AL_CB_ParsedSei	parsedSeiCB	在解析 SEI 时调用

请参阅

AL_CB_EndDecoding、 AL_CB_Display、 AL_CB_ResolutionFound

结构

AL_CB_EndDecoding

描述

表 12-18：

类型	字段	描述
Function pointer(1)	func	在帧被解码时调用。出错时， pDecodedFrame 将为 NULL。使用 AL_Decoder_GetLastError 获取更多信息。
void*	userParam	用户定义

注释：

1. void (*func)(AL_TBuffer* pDecodedFrame, void* pUserParam)

请参阅

AL_TDecCallBacks、 AL_Decoder_GetLastError

结构

AL_CB_Display

描述

类型	字段	描述
Function pointer(1)	func	显示框架时调用。在流结束时， pDisplayedFrame 将为 NULL。使用 AL_Decoder_GetLastError 检查错误信息。
void*	userParam	用户定义

注释：

1. void (*func)(AL_TBuffer* pDisplayedFrame, AL_TInfoDecode* pInfo, void* pUserParam)

请参阅

AL_TDecCallBacks、 AL_Decoder_GetLastError、 AL_Decoder_PutDisplayPicture

结构

AL_CB_ParsedSei

描述

可以解析对分辨率 SEI 回调的定义。

类型	字段	描述
Function pointer(1)	func	在解析 SEI 时调用。解码器已经从有效载荷中去除了反模拟。有关 sei_payload 语法，请参见 ITU-T 的附录 D.3。
void*	userParam	用户定义

注释：

1. void (* func)(int iPayloadType, uint8_t* pPayload, int iPayloadSize, void* pUserParam);

结构

AL_CB_ResolutionFound

描述

指对分辨率更改回调的定义。

类型	字段	描述
Function pointer(1)	func	在第一次解码过程发生时仅调用了一次。解码器不支持更改流内的分辨率。使用 AL_Decoder_GetLastError 检查错误信息。
void*	userParam	用户定义

注释：

1. void (*func)(int BufferNumber, int BufferSize, AL_TStreamSettings const* pSettings, AL_TCropInfo const* pCropInfo, void* pUserParam)

请参阅

AL_TDecCallBacks、 AL_Decoder_GetLastError、 AL_Decoder_PutDisplayPicture

函数

AL_TIDecChannel* AL_DecChannelMcu_Create()

描述

返回指向新分配的 AL_TIDecChannel 对象的指针，如果分配失败，则返回 NULL。

结构

AL_TIDecChannel

结构

AL_CB_EndFrameDecoding

描述

当解码器完成对帧的解码时，调用该回调。此回调结构包含一个函数指针 （该函数指针指向一个获取用户参数和图像状态的函数）。图像状态是指向 AL_TDecPicStatus 的指针。默认回调是 AL_Default_Decoder_EndDecoding。

类型	字段	描述
Function pointer(1)	func	在帧被解码时调用
void*	userParam	用户定义

注释：

1. void (*func)(void* pUserParam, AL_TBuffer* pStream, AL_TDecPicStatus* pPicStatus)

请参阅

AL_Default_Decoder_EndDecoding

函数

void AL_Decoder_Destroy(AL_HDecoder hDec)

描述

释放与解码器 hDec 关联的资源。

请参阅

AL_Decoder_Create

函数

void AL_Decoder_PutDisplayPicture(AL_HDecoder hDec, AL_TBuffer* pDisplay)

描述

将显示缓存 pDisplay 添加到解码器的内部缓存池 （该缓存池用来输出解码图像）。最多允许 50 个显示缓存。根据编译器设置，超过此限制将导致置位违例；或者缓存将被清除，但不会添加到池中。

请参阅

AL_TFrmBufPool、 FRM_BUF_POOL_SIZE

函数

bool AL_Decoder_PreallocateBuffers(AL_HDecoder hDec)

描述

根据流设置和选定的编解码器分配内存，还会进行一些缓存初始化。

返回值

如果分配成功则返回 true，否则，返回 false。

请参阅

AL_Default_Decoder_PreallocateBuffers、 AL_Default_Decoder_AllocPool、 AL_Default_Decoder_AllocMv、 AL_PictMngr_Init

函数

void AL_Default_Decoder_EndDecoding(void* pUserParam, AL_TDecPicStatus* pStatus)

描述

解码器完成解码帧时调用的默认函数指针可执行各种显示缓存操作：

- 更新显示缓存 CRC
- 更新缓存指针、计数器和引用计数
- 释放未使用的帧解码存储器
- 表示帧已准备好可以显示了
- 如果需要重新启动解码器，则打印消息到 `stdout` 文件

请参阅

AL_PictMngr_EndDecoding、 AL_t_PictureManagerCallbacks、 AL_PictMngr_UnlockRefMvID

结构

AL_TDecPicStatus

描述

将计数器、 CRC 和标志与帧相关联。

类型	字段	描述
uint8_t	uFrmID	显示 FIFO 中的帧标识符
uint8_t	uMvID	运动矢量标识符
uint32_t	uNumLCU	最大编码单元 (LCU) 中的条目数
uint32_t	uNumBytes	未使用
uint32_t	uNumBins	未使用
uint32_t	uCRC	帧 CRC
bool	bConceal	这个帧被隐藏了？
bool	bHanged	解码器超时了？

结构

AL_TDecSettings

描述

类型	字段	描述
int	iStackSize	解码器处理的命令堆栈的大小
int	iBitDepth	输出位深度
uint8_t	uNumCore	所用解码器核数
uint32_t	uFrameRate	如果指定帧速率的语法元素不存在，则使用用户提供的帧速率

类型	字段	描述
uint32_t	uClkRatio	如果语法元素不提供，则使用用户提供的时钟比率
bool	bIsAvc	如果为 true，则选择 AVC 解码；如果为 false，则选择 HEVC 解码
bool	bParallelWPP	如果为 true，则启用波前并行处理。不支持。必须为 false。
uint8_t	uDDRWidth	解码器使用的 DDR 宽度。 16 或 32（取决于电路板设计）。
bool	bDisableCache	如果为 true，则禁用解码器高速缓存
bool	bLowLat	如果为 true，则使用低时延解码
bool	bForceFrameRate	如果为true，则用户定义的帧速率将覆盖流的帧速率
bool	bFrameBufferCompression	如果为 true，则应使用内部帧缓存压缩
AL_EFbStorageMode	eFBStorageMode	AL_FB_RASTER = 0 AL_FB_TILE_32x4 = 2 AL_FB_TILE_64x4 = 3
AL_EDecUnit	eDecUnit	子帧时延控制 AL_AU_UNIT = 0 AL_VCL_NAL_UNIT = 1
AL_EDpbMode	eDpbMode	低参考帧模式控制 AL_DPB_NORMAL = 0 AL_DPB_LOW_REF = 1
AL_TStreamSettings	tStream	解码器输出流

函数

bool AL_Decoder_PushBuffer(AL_HDecoder hDec, AL_TBuffer* pBuf, size_t uSize, AL_EBufMode eMode)

描述

将缓存推入解码器队列。生成传入的工作事件。缓存会尽快被解码。 uSize 实参指示要解码的缓存的字节数。 eMode 实参指定是否会阻塞。将使用指向解码帧缓存器和用户参数的指针来调用 AL_CB_EndDecoding 中提供的函数指针。在创建解码器对象时提供该函数指针和其他函数指针。注意：提供的缓存 pBuf 不得包含与之关联的 AL_TCircMetaData。

请参阅

AL_EBufMode, AL_CB_EndDecoding, AL_Decoder_Create

函数

void AL_Decoder_Flush(AL_HDecoder hDec)

描述

当流解析完成时，请求解码器刷新解码请求堆栈。

函数

void AL_Decoder_GetMaxBD(AL_HDecoder hDec)

描述

返回当前流配置信息支持的最大位深度。

函数

int32_t RndPitch(int32_t iWidth, uint8_t uBitDepth, AL_EFbStorageMode eFrameBufferStorageMode)

返回值

返回向上舍入的、与突发 DMA 对齐的间距 (pitch)。

函数

int32_t RndHeight(int32_t iHeight)

返回值

返回与最近的 64 位边界对齐的高度。

函数

int AL_GetNumLCU(AL_TDimension tDim, uint8_t uLCUSize)

描述

以 2uLCUSize 为单位表示由 tDim 定义的区域像素。 uLCUSize 实参必须是 4、 5 或 6，分别对应于 16×16、 32×32 或 64×64 的块。

返回值

返回帧中的 LCU 数。

函数

int AL_GetAllocSize_HevcCompData(AL_TDimension tDim, AL_EChromaMode eChromaMode)

[in] tDim	帧维度 （宽度、高度），以像素为单位
[in] eChromaMode	色度子采样模式

返回值

返回 HEVC 压缩缓存的大小 （LCU 报头 + MVDs + 残差）。

函数

int AL_GetAllocSize_AvcCompData(AL_TDimension tDim, AL_EChromaMode eChromaMode)

[in] tDim	帧维度 （宽度、高度），以像素为单位
[in] eChromaMode	色度子采样模式

返回值

返回 AVC 压缩缓存的大小 （LCU 报头 + MVDs + 残差）。

函数

int AL_GetAllocSize_DecCompMap(AL_TDimension tDim);

[in] tDim	帧维度 （宽度、高度），以像素为单位
[in] tDim	帧维度 （宽度、高度），以像素为单位

返回值

返回压缩映射缓存的最大大小 （即 LCU 偏移量和大小，以字节为单位）。

函数

int AL_GetAllocSize_HevcMV(AL_TDimension tDim)

[in] tDim	帧维度 （宽度、高度），以像素为单位
[in] tDim	帧维度 （宽度、高度），以像素为单位

返回值

返回共址 HEVC 运动向量缓存所需的大小 （以字节为单位）。

函数

int AL_GetAllocSize_AvcMV(AL_TDimension tDim)

[in] tDim	帧维度 （宽度、高度），以像素为单位
in] tDim	帧维度 （宽度、高度），以像素为单位

返回值

返回共址 AVC 运动向量缓存所需的大小 （以字节为单位）。

函数

int AL_GetAllocSize_Frame(AL_TDimension tDim, AL_EChromaMode eChromaMode, uint8_t uBitDepth, bool bFrameBufferCompression, AL_EFbStorageMode eFrameBufferStorageMode)

[in] tDim	帧维度 （宽度、高度），以像素为单位
[in] eChromaMode	色度模式
[in] uBitDepth	比特深度
[in] bFrameBufferCompression	指定是否需要压缩
[in] eFrameBufferStorageMode	帧缓存的存储模式

返回值

返回输出帧缓存所需的大小 （以字节为单位）。

函数

int AL_GetAllocSize_DecReference(AL_TDimension tDim, int iPitch AL_EChromaMode eChromaMode, AL_EFbStorageMode eFrameBufferStorageMode)

[in] tDim	帧维度 （宽度、高度），以像素为单位
[in] eChromaMode	色度子采样
[in] uBitDepth	图像样本的位大小
[in] eFrameBufferStorageMode	帧缓存的存储模式

返回值

返回参考帧缓存所需的大小 （以字节为单位）。

函数

uint32_t GetAllocSizeEP2(AL_TDimension tDim, uint8_t uMaxCuSize)

[in] tDim	帧的大小 （以像素为单位）
[in] uMaxCuSize	编码单元的最大大小

返回值

返回存储 QP Ctrl Encoder 参数缓存 (EP2) 所需的最大大小 （以字节为单位）。

函数

uint32_t AL_GetAllocSize(AL_TDimension tDim, uint8_t uBitDepth, AL_EChromaMode eChromaMode, AL_EFbStorageMode eStorageMode)

[in] tDim	帧的大小 （以像素为单位）
[in] uBitDepth	YUV 位深度
[in] eChromaMode	色度模式
[in] eStorageMode	源存储模式

返回值

返回 YUV 帧缓存所需的最大大小 （以字节为单位）。

函数

uint32_t GetAllocSize_Src(AL_TDimension tDim, uint8_t uBitDepth, AL_EChromaMode eChromaMode, AL_ESrcMode eSrcFmt)

[in] tDim	帧的大小 （以像素为单位）
[in] uBitDepth	YUV 位深度
[in] eChromaMode	色度模式
[in] eSrcFmt	硬件 IP 使用的源格式

返回值

返回 YUV 源帧缓存所需的大小 （以字节为单位）。

函数

int AL_CalculatePitchValue(int iWidth, uint8_t uBitDepth, AL_EFbStorageMode eStorageMode)

[in] iWidth	帧宽 （以像素为单位）
[in] uBitDepth	YUV 位深度
[in] eStorageMode	源存储模式

返回值

以字节为单位返回间距 (pitch) 值。

函数

AL_EFbStorageMode AL_GetSrcStorageMode(AL_ESrcMode eSrcMode)

[in] iWidth	帧宽 （以像素为单位）
[in] eSrcMode	源模式

返回值

返回源帧缓存存储模式。

函数

AL_ERR AL_Decoder_GetLastError(AL_HDecoder hDec)

[in] iWidth	帧宽 （以像素为单位）
[in] hEnc	解码器句柄

返回值

从解码器的内容结构中返回错误代码。 Thread-safe。

错误代码	描述
AL_ERR_INIT_FAILED	由于参数不一致而无法初始化解码器
AL_ERR_CHAN_CREATION_NO_CHANNEL_AVAILABLE	由于没有可用的通道，因此未创建通道
AL_ERR_CHAN_CREATION_RESOURCE_UNAVAILABLE	未创建通道，因为可用核的处理能力不足
AL_ERR_CHAN_CREATION_NOT_ENOUGH_CORES	未创建通道，因为可用来分散负载的处理核太少
AL_ERR_REQUEST_MALFORMED	由于请求格式错误，因此未创建通道
AL_ERR_NO_FRAME_DECODED	没有帧被解码

错误代码	描述
AL_ERR_RESOLUTION_CHANGE	不支持分辨率更改
AL_ERR_NO_MEMORY	检测到内存不足 （DMA、嵌入式内存或虚拟内存不足）

优化视觉质量

编码视频的质量主要取决于目标比特率函数和视频内容的类型。有极少数编码器参数可以用于调节编码器质量。要调试 HEVC/AVC 编码器的质量问题，执行以下步骤：

1. 根据动态量来调整 B 帧数 (Gop.NumB)，例如，可将静态场景或类似视频会议的内容增加到 2，或将动态量大或帧速率高的序列减少到 0
2. 当某些序列部分的复杂度和/或动态量较低时， VBR 速率控制模式可以提升平均质量
3. 在用于视频会议或不需要随机访问时，用 LOW_DELAY_P GOP 更换 IPP...GOP，并可以选择启用视频渐近刷新 (GDR) 的帧内刷新
4. 如果频繁更改场景，则可通过启用 ScnChgResilience 设置来减少场景更改转换后的伪像。
5. 如果系统可以检测到场景变化，则应该调用编码器的场景变化信号 API （即禁用 (ScnChgResilience)，以便编码器能够动态地调整编码参数和 GOP 模式。当使用控制软件测试应用时，可以在单独的输入文件 (CmdFile) 中提供场景更改信息。

提示：要提高视频质量，可考虑使用场景变化检测硬件。

6. 如果把最高 PSNR 数字而不是主观质量作为目标，则建议设置 QPCtrlMode = UNIFORM_QP 和 ScalingList = FLAT。
7. 使用 CONST_QP 速率控制算法来评估特定视频流的有效比特率范围。
8. 使用由 CONST_QP 速率控制算法获得的 QP 值 22、 27、 32、 37
9. 使用步骤 2 中的比特率范围，评估 CBR/VBR/低时延码率控制算法的 PSNR
10. 对 libx264 或 libx265 编码器运行类似的实验。

    以下是 CBR 的示例流水线：

    ```
    ffmpeg -s 1280x720 -pix_fmt yuv420p -framerate 50 -i /srv/data1/kvikrama/
    vcu_video_quality_runs/source/old_town_cross_420_720p50.yuv -c:v libx265 -preset
    medium -x265-params bframes=0:ref=1:keyint=30:rc
    lookahead=0:ipratio=1:pbratio=1:trellis=0 -b:v 1M -minrate 1M -maxrate 1M -bufsize
    2M   -frames 500   old_town_cross_420_720p50_x265.mp4
    ```

11. 使用 JCT-VC 通用测试条件评估指标来计算 BD 速率。
12. 如果 VCU 和 libx264/libx265 之间的 PSNR 存在差异，请优化以下参数查看影响：

MinQP、 MaxQP、 ScnChgResilience （应启用）、 NumSlices （设置为 1）。

用例的最佳 VCU 编码器参数

视频流

- 视频流用例需要所有图像都有非常稳定的比特率图表。
- 最好能在编码会话期间避免周期性大的帧内图像
- 低时延率控制 （硬件 RC）是视频流的首选控制率，它会试图使所有图像的帧大小保持一致。
- 最好能避免周期性帧内帧，而不是使用已启用视频帧内刷新的 low-delay-p (IPPPPP...) （gdr-mode =水平或垂直）
- VBR 不是首选的流媒体模式。

AVC 编码器性能设置

- 倾向于使用 8 个或更多个 slice 以获得更佳的 AVC 编码器性能。
- AVC 标准不支持块 (Tile) 模式处理 （该处理可导致循序处理 MB 行以进行熵编码）。

低比特率 AVC 编码

- 启用 profile=high 并将 qp-mode=auto 用于低比特率编码用例。
- 高配置信息可实现 8x8 变换，从而在低比特率下实现更好的视频质量。
- 启用 GopMode=low-delay-p 可提升低比特率用例的质量。

附录 A

调试

本附录包含有关赛灵思支持网站和调试工具上可用资源的详细信息。

提示：如果 IP 生成因错误而停止，则可能存在许可证问题。

在 Xilinx.com 上寻求帮助

为了在使用 Video Codec Unit 时帮助设计和调试进程，赛灵思支持网页提供了关键资源 （如产品文档、发行说明、答案记录、已知问题相关信息以及获取进一步产品支持的链接）。

文档

本产品指南是与 Video Codec Unit 相关的主要文件。本指南以及与所有有助于设计进程的产品相关的文档可以在赛灵活思支持网页上找到，可以在赛灵活思支持网页上找到，也用以通过赛灵思 Documentation Navigator 找到。

从下载页面下载赛灵思 Documentation Navigator。如需了解此工具和可用功能的详细信息，请在安装后打开联机帮助。

解决方案中心

如需了解设计周期各阶段有关器件、软件工具和 IP 等的技术支持，请参阅赛灵思解决方案中心。相关专题包括设计辅助、建议和故障排除提示等。

答复记录

答案记录包括有关常见问题的信息、有关如何解决这些问题的有用信息以及赛灵思产品的任何已知问题。我们每天都会创建和维护答案记录，确保用户可以访问最准确的信息。

可以通过赛灵思支持网页 （主页）上的“搜索支持”框找到此核的记录。要最大化搜索结果，请使用适当的关键字，例如：

- 产品名称
- 工具消息
- 所遇到问题的摘要

返回结果后，可以使用过滤器搜索来进一步定位结果。

Video Codec Unit 主答复记录

AR：66763

技术支持

赛灵思在赛灵思支持网页上提供有对此 LogiCORE™ IP 产品的技术支持，但这些技术支持主要针对的是产品文档中所描述的使用方法。如果您执行以下任何操作，赛灵思无法保证能及时提供功能或支持：

- 在文档中未定义的器件中实施解决方案。
- 在超出产品文档中允许的范围定制解决方案。
- 更改标记有“DO NOT MODIFY”的设计的任何部分。

要联系赛灵思技术支持，请导航至赛灵思支持网页。

调试工具

有许多工具可用于解决 Video Codec Unit 设计问题。至关重要的是要了解哪些工具可用于调试各种情况。

Vivado Design Suite 调试特性

Vivado® Design Suite 调试功能可以将逻辑分析器和虚拟 I/O 核直接插入到您的设计中。调试功能还让您可以设置触发条件，以便于在硬件中捕获应用和集成块端口信号。之后便可以对捕获的信号进行分析了。 Vivado IDE 中的这个功能用来对在赛灵思器件中运行的设计进行逻辑调试和验证。

Vivado 逻辑分析器与逻辑调试 IP 核一起使用，包括：

- ILA 2.0 （及更高版本）
- VIO 2.0 （及更高版本）

请参阅 《Vivado Design Suite 用户指南：编程和调试》(UG908) [参照 6]。

参考板

各种赛灵思开发板都支持 Video Codec Unit。这些电路板可用于进行原型设计并确定核可与系统通信。

- ZCU106

硬件调试

硬件问题各不相同，可能是链接开启问题，也可能是经过数小时测试后看到的问题。本节提供了常见问题的调试步骤。Vivado 调试功能是可用于硬件调试的宝贵资源。可以通过调试功能来探测以下各个部分中提到的信号名称，以便对特定问题进行调试。

一般检查

确保核的所有时序约束都已正确地纳入示例设计中，并在实现期间满足所有约束条件。

- 它是否适用于后置仿真和路径时序仿真？如果在硬件中看到问题但在时序仿真中没有出现问题，则可能表示存在 PCB 问题。确保所有时钟源都干净且处于活动状态。
- 如果在设计中使用 MMCM，请通过监控 `locked` 端口来确保所有 MMCM 都获得了锁定。
- 如果您的输出为 0，检查您的许可。

调试 VCU 的系统

对基于 VCU 的系统进行故障排除可能会很复杂。本附录提供了一个决策树，可以指导您有效地进行最有效的调查。故障排除步骤适用于能够执行实时捕获、编码、解码、传输和显示的基于 VCU 的系统。

调试流程

图 A-1 显示的是系统级调试流程。

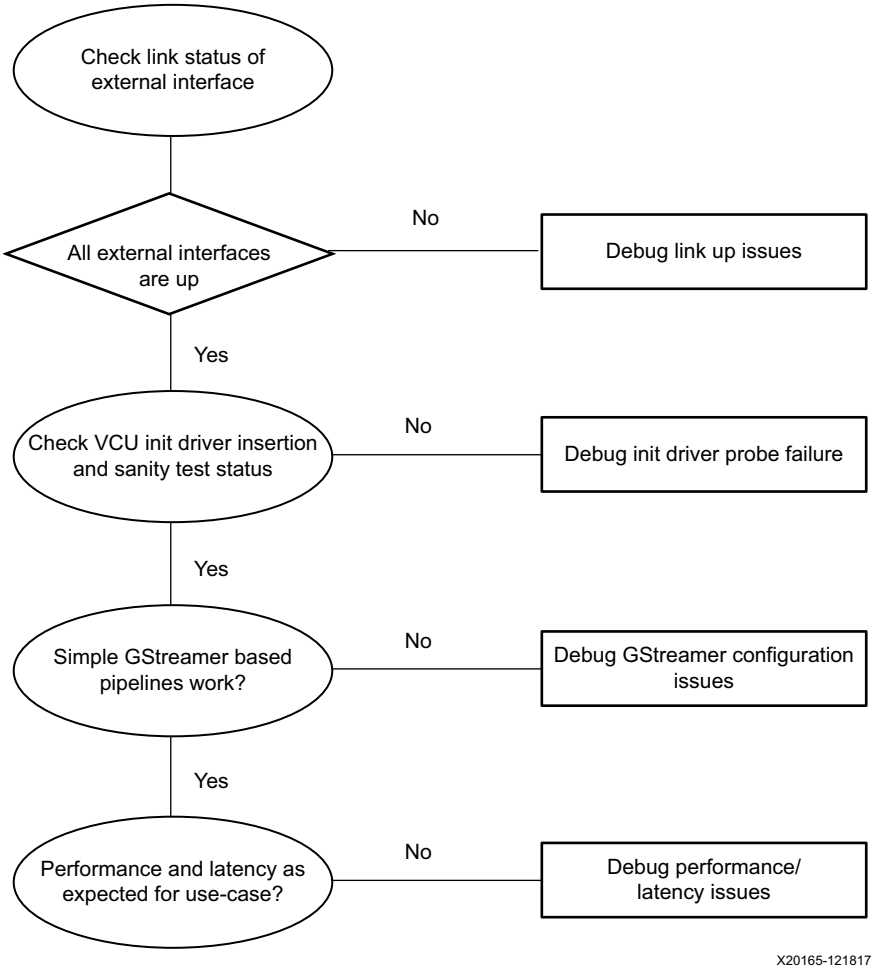

图 A-1：调试流程

故障排除

调试外部接口

如果您使用基于高速串行 I/O 的捕获和显示接口 （例如 HDMI、 MIPI、 SDI 等），请在调试上层之前确保物理链路已启动。请按以下步骤操作：

1. 监控指示心跳时钟的 LED （该心跳时钟是根据输入到收发器的参考时钟得出的）。如果心跳时钟不存在，检查 GT PLL 锁定状态。
2. 如果 PLL 未锁定，检查 Video PHY IP 配置和参考时钟约束。大多数情况下，参考时钟源自可编程时钟芯片。确保在可编程时钟芯片的设备树源文件中对频率进行的编程正确无误，并且其他 Linux 器件不会修改频率 （当多个组件之间共享时钟源节点的 `phandle` 时会发生这种情况）。
3. 如果捕获接口与 Video for Linux (v4l2) 框架兼容，使用 media-ctl API 验证链接拓扑和链接状态。

   ```
   xmedia-ctl –p –d /dev/mediaX
   ```

   mediaX 表示 v4l2 流水线中的流水线器件。如果您有多个 TPG/HDMI 流水线，它们显示为 `/dev/media0`、`/dev/media1` 等，链接状态会在相应的子器件节点中指示。例如，HDMI RXSS 节点在使用上述命令时指示其子器件属性中 HDMI 链路的链路状态。如果链接失败，则会显示“无链接”消息。否则，系统会检测到颜色格式适当的有效分辨率。

4. 如果显示界面与 DRM/KMS 框架兼容，请通过运行以下命令之一确保 DRM 正在连接：

   如果使用 PL 混合块，

   ```
   modetest –M xilinx_drm_mixer
   ```

   如果不使用 PL 混合器块，

   ```
   modetest –M xilinx_drm
   ```

   如果要显示模式以确保显示路径正常，使用以下命令之一：

   要使用 BG24 格式显示，

   ```
   modetest –M xilinx_drm_mixer –s <connector_id>@<crtc_id>:3840x2160-60@BG24
   ```

   要使用 NV12 格式显示

   ```
   modetest –M xilinx_drm_mixer –P <crtc_id>:3840x2160-60@NV12
   ```

   在继续使用不同的 VCU 用例之前，多次运行 `modetest` 命令以确保链接在不同分辨率下均处于稳定状态。

5. 如果捕获流水线 （基于 v4l2）或显示流水线 （基于 DRM/KMS）被损坏，那么 `media-ctl` 或 `modetest` 应用显示流水线已损坏且子器件连接得不正确。如果是这样，浏览启动日志以查看是否存在任何 v4l2 子器件驱动探测失败的情况。如果探测失败，检查一下您的 dts 文件，看看是否存在属性名称不匹配或属性缺失或不正确的情况。

调试 VCU 接口

要调试 VCU 接口，请执行以下步骤：

1. 确保在启动过程中 VCU init 驱动探测成功。在启动日志中，您可以看到以下消息

   ```
   xilinx-vcu <axilite address>.vcu: xvcu_probe: Probed successfully
   ```

2. 如果 PLL 无法锁定，VCU 初始化驱动会报告锁定状态失败的消息。确保在 IP 集成器块设计中 VCU LogiCORE IP 的 PLL 输入参考时钟频率设置得准确无误。确保 PLL 时钟源 （父时钟）在设备树节点中作为固定时钟源被正确地建模。确保 VCU init 驱动节点的属性正确无误，并且 PLL 参考时钟的 `phandle` 得以恰当传递。
3. 成功探测 VCU init 驱动后，使用以下命令检查 VCU 驱动：

```
lsmod
```

这样做应该会显示正在插入的 al5d、 al5e 和 Allegro 模块。

4. 检查是否分配了足够的 CMA。 VCU 操作需要至少 1000 MB 的 CMA。您可以使用以下命令检查可用的 CMA

```
cat /proc/meminfo
```

5. 如果 CMA 大小不足，请使用以下命令在 U-Boot 中增加大小：

```
cma=xxxxM
```

也可以在使用内核配置属性构建 Linux 内核时增加其大小。

6. 使用 VCU Control Software 示例应用运行 VCU 完整性测试。您可以使用 `ctrlsw_encoder` 和 `ctrlsw_decoder` 应用来运行进行可用性测试。

调试控制软件应用

要调试基于控制软件的应用 (`ctrlsw_encoder`， `ctrlsw_decoder`)，请执行以下步骤。

1. 将文件运行到基于文件的编码器和解码器示例应用，以确保其按预期运行。
   - H.264 解码 （文件到文件）

```
ctrlsw_decoder -avc -in input-avc-file.h264 -out ouput.yuv
```

   - H.265 解码 （文件到文件）

```
ctrlsw_decoder -hevc -in input-hevc-file.h265 -out ouput.yuv
```

   - 编码 （文件到文件）

```
ctrlsw_encoder –cfg encode_simple.cfg
```

应用退出并显示相应的错误消息。有关详细信息，请参阅错误说明表。

例如，如果 VCU 电源不足，无法以 120fps 的速度对分辨率为 4kp 的文件进行编码，则系统会显示以下消息：

```
ctrlsw_encoder -cfg AVC_test.cfg -i test_4Kp120_nv12.yuv -o /dev/null
Allegro DVT2 - AVC/HEVC Encoder Reference Software v1.0.41 - Copyright (C) 2018
Confidential material

Channel creation failed, processing power of the available cores insufficient
Codec error: Channel creation failed, processing power of the available cores
insufficient

To debug above error, change either Frame rate(4kp60) or resolution(1080p120) in cfg
file which can be handled by VCU and re-run.
```

如果 CMA 不足，应用退出并显示存储器错误。

例如：

```
ctrlsw_encoder -cfg AVC_test.cfg -i test_4Kp60_nv12.yuv -o /dev/null
Allegro DVT2 - AVC/HEVC Encoder Reference Software v1.0.41 - Copyright (C) 2018
Confidential material

[53.119829] cma: cma_alloc: alloc failed, req-size: 3078 pages, ret: -12
[53.126542] al5e a0100000.al5e: Can't alloc DMA buffer of size 12607488
GET_DMA_FD: Cannot allocate memory
[53.137916] cma: cma_alloc: alloc failed, req-size: 10522 pages, ret: -12
[53.144712] al5e a0100000.al5e: Failed internal buffers allocation, channel wasn't
created
Memory shortage detected (DMA, embedded memory or virtual memory)
```

```
Codec error: Memory shortage detected (DMA, embedded memory or virtual memory)
```

要缓解存储器错误，使用以下命令检查 CMATotal 和 CMAFree 是否足够：

```
cat /proc/meminfo
```

要么使用命令在 U-Boot 中增加 CMA 的大小：

```
cma=xxxxM
```

要么在使用内核配置属性构建 Linux 内核时增加其大小

2. 如果未生成输出文件且终端上未出现故障或错误消息，使用以下命令查看内核层的消息：

```
dmesg

  Check “al5e”, “al5d” keyword in the dmesg log for the error if any.
```

3. 如果应用停止响应，请使用“gdb”进行调试。为此，从控制软件 Makefile 中删除最优化，如下所示：

```
replace: CFLAGS+=-O3
   By CFLAGS+=-O0
```

然后使用以下命令运行应用：

```
gdb –args ctrlsw_encoder -cfg AVC_test.cfg -i test_4Kp60_nv12.yuv -o /dev/null
```

此操作会提供 gdb shell。键入“run”执行应用

```
(gdb)run
```

如果系统停止响应，使用 backtrace 函数查看开始停止响应的最后一个函数流。

```
(gdb)bt
```

调试 GStreamer 的应用

要调试基于 GStreamer 的接口，请执行以下步骤：

1. 通过运行捕获显示流水线以确保要显示的实时捕获会按预期工作。

```
gst-launch-1.0 -v v4l2src io-mode=4 device=/dev/video1 ! video/
x-raw,format=NV12,width=3840,height=2160, framerate=30/1 ! kmssink
driver-name=xilinx_drm_mixer
```

2. 如果您在“capture-> display”流水线中看到“streamon”错误，确保使用 hdmi 配置脚本中的 v4l2-ctl 和 media-ctl API 将格式设置为 NV12。下面给出一个实例：

```
v4l2-ctl -d /dev/video1 --set-fmt-video=width=3840,height=2160,pixelformat='NV12'
xmedia-ctl -d /dev/media1 -V "\"a0080000.scaler\":0  [fmt:RBG888_1X24/3840x2160
field:none]"
xmedia-ctl -d /dev/media1 -V "\"a0080000.scaler\":1  [fmt:VYYUYY8_1X24/3840x2160
field:none]"
```

3. 在流水线中运行带有 VCU 组件的流水线。在流水线中使用 omxh264/omxh265enc/dec 组件。
4. 有关 omxh264/omxh265enc/dec 支持的属性的列表，使用以下命令：

```
gst-inspect-1.0 <omxh264/omxh265enc/dec>
```

系统会列出支持的 VCU 编码器和解码器块的所有属性。使用流水线中的适当属性。

5. 先从已知的流水线示例着手。下面给出一个实例：

```
gst-launch-1.0 -v  \
```

```
v4l2src num-buffers=2100 device=/dev/video8 io-mode=4 \
! video/x-raw,format=NV12,width=3840,height=2160, framerate=30/1 \
! omxh265enc target-bitrate=70000 prefetch-buffer-size=630 \
 control-rate=2 gop-length=30 b-frames=0 \
! video/x-h265, profile=main,level=\(string\)5.1,tier=main \
! omxh265dec latency-mode=0 internal-entropy-buffers=2 \
! queue \
! fpsdisplaysink name=fpssink text-overlay=false \
video-sink="kmssink max-lateness=100000000 async=false \
sync=true driver-name=xilinx_drm_mixer" -v
```

调试性能问题

有关其他调试信息，请查阅 GStreamer 网站的“调试工具”页面：
https://gstreamer.freedesktop.org/documentation/tutorials/basic/debugging-tools.html。

如果问题是帧速率低或帧丢失，请按照以下步骤调试系统。

1. 使用 `fpsdisplaysink` 元素来报告帧速率和丢帧数 （请参阅上面的示例）
2. 检查将 VCU 连接到 PS DDR 的 HP 端口的 QoS 设置。检查待处理数据传输数的配置。请注意，对于 VCU 数据流量， QoS 应设置为 Best Effort (BE)，并且待处理数据传输数应设置为最大值 （AFI 端口：0xF）。
3. 检查设备树中是否已启用 SMMU。如果启用了SMMU，则禁用它，因为它会在数据传输完成时导致更长的时延。
4. 检查用户设计中是否使用了编码器缓存区 （预取缓存区）。如果没有使用，检查一下，看看编码器缓存区是否影响了性能。如果编码器缓存区的性能提高了，则有可能表示系统带宽有问题。使用以下示例流水线启用设计中的预取缓存区：

   ```
   gst-launch-1.0 videotestsrc ! omxh265enc prefetch-buffer=true ! fakesink
   ```

5. 尝试使用不同的编码器属性或解码器属性来查看性能下降是否与任何属性相关。避免流水线中的 B 帧，看看是否有任何性能改进。如果性能提高了，则有可能表示系统带宽有问题。降低比特率，看看帧速率是否有提升。如果降低目标比特率可以提供更好的吞吐量，则可能表明系统带宽有问题。
6. 在流水线运行时检查 CPU 利用率。较高的 CPU 利用率表明中断处理时间可能会产生影响，从而降低帧速率。
7. 尝试在数据路径中的两个 GStreamer 插件之间使用 `queue` 队列元素来检查是否有任何性能改进
8. 使用 DDR APM 和 VCU APM 检查 DDR 带宽利用率
9. 使用 gst-shark （基于 GStreamer 的工具）来验证性能并创建调度时间和交互图，以了解哪个元素将导致性能下降。

调试时延问题

如果问题出在端到端时延较高，请按照以下步骤调试系统：

1. 了解客户端流水线 (udpsrc) 中使用的抖动缓存区的数量。请注意， `rtpjitterbuffer` 的时延应与服务器流水线中的 CPB 大小相对应。通常，使用 LowLatency 速率控制 （或硬件速率控制）算法来维持较低的 CPB 缓存区以减少 `rtpjitterbuffer` 缓存时延是有用的。
2. 很多时候，时延与帧丢失有关。在测量时延之前，确保流水线中没有帧丢失。
3. 检查编码器流水线中填充数据设置的使用是否将导致更长的时延。如果是这样，在流水线中设置 filler-data=false 并检查时延。
4. 使用微调的内部熵缓存区数。请注意，内部熵缓存区设置会影响时延，因此需要使用最佳值。您可以更新此解码器属性先确保不会丢帧，然后再最优化流水线的时延时间。

接口调试

AXI4-Lite 接口

要验证接口是否正常工作，请尝试从没有全 0 的寄存器中读取默认值。输出 s_axi_arready 在读取地址有效时置位，而输出 s_axi_rvalid 在读取数据或响应有效时置位。如果界面没有响应，确保满足以下条件：

- s_axi_aclk 和 aclk 输入已连接并正常工作。
- 接口未保持复位状态，且 s_axi_areset 为低电平有效复位状态。
- 接口已启用， s_axi_aclken 为 active-High （如果使用）。
- 主核时钟正在正常工作，并且还启用了使能。
- 如果仿真已运行，请在仿真和/或调试功能捕获过程中验证波形正确且已访问 AXI4-Lite 接口。

AXI4-Stream 接口

如果未传输或接收数据，检查以下条件：

- 如果在置位 <interface_name>_tready 输入后传输 <interface_name>_tvalid 被卡在低位，则核无法发送数据。
- 如果接收 <interface_name>_tvalid 被卡在低位，则核不接收数据。
- 检查 aclk 输入是否已连接并正常工作。
- 检查是否遵循 AXI4-Stream 波形。
- 检查核配置。

XILINX

附录 B

附加资源与法律提示

赛灵思资源

如需了解答复记录、技术文档、下载以及论坛等支持性资源，请参阅赛灵思技术支持。

参考资料

以下技术文档是本产品指南非常实用的补充材料：

1. 《Vivado Design Suite 用户指南：采用 IP 集成器设计 IP 子系统》(UG994)
2. 《Vivado Design Suite 用户指南：采用 IP 进行设计》(UG896)
3. 《Vivado Design Suite 用户指南 I/O 管脚分配和时钟规划》(UG899)
4. 《Vivado Design Suite 用户指南：着手设计》(UG910)
5. 《Vivado Design Suite 用户指南：逻辑仿真》(UG900)
6. 《Vivado Design Suite 用户指南：编程和调试》(UG908)
7. 《Vivado Design Suite 用户指南：实现》(UG904)
8. 《Vivado Design Suite 用户指南：设计分析和收敛技术》(UG906)
9. 《LogiCORE IP AXI Interconnect 产品指南》(PG059)
10. 《LogiCORE IP UltraScale 架构的 FPGA 存储器 IP 产品指南》(PG150)
11. 《UltraScale 架构 SelectIO 资源用户指南》(UG571)
12. 《Zynq UltraScale+ MPSoC 量产勘误表》(EN285)
13. 《Zynq UltraScale+ 器件技术参考手册》(UG1085)
14. 《Zynq UltraScale+ MPSoC 寄存器参考》(UG1087)
15. 《Zynq UltraScale+ MPSoC 数据手册：简介》(DS891)
16. 《Zynq UltraScale+ MPSoC 数据手册：DC 和 AC 开关特性》(DS925)
17. 《LogiCORE IP Zynq UltraScale+ MPSoC Processing System 产品指南》(PG201)
18. 《PetaLinux 工具文档》(UG1144)
19. OpenMax Integration Layer (https://www.khronos.org/registry/OpenMAX-IL/specs/OpenMAX_IL_1_1_2_Specification.pdf)
20. GStreamer (https://gstreamer.freedesktop.org/features/)

培训资料

1. Vivado Design Suite Hands-on Introductory Workshop
2. Vivado Design Suite 工具流程

修订历史

下表列出了本文档的修订历史。

日期	版本	修订
2018 年 12 月 5 日	1.2	更新软 IP 寄存器表。 添加了编码器和解码器原理图。 添加了编码器和解码器缓存要求。 添加第 6 章"Zynq UltraScale+ EV 架构 Video Codec Unit DDR4 LogiCORE IP"。 添加了各种时延模式的详细信息。 在用例中添加了 Optimum VCU Encoder 参数。 更新软件要求。 更新了 VCU 开箱即用示例部分。 添加了以下新功能：GDR 内部刷新、长期参考图像、自适应 GOP、在编码器中插入 SEI 数据、 SEI 解码器 API、双通道编码、场景变化检测、隔行扫描视频。 更新了编码器输入参数。 更新了编码器设置参数。 添加了有关从实时流媒体源使用 ROI 和 LoadQP 函数的信息。 更新了控制软件调试信息和错误代码。 更新了寄存器和 GStreamer 元素。 分别用 ctrlsw_encode 和 ctrlsw_decoder 替换了 AL_Encoder.exe 和 AL_Decoder.exe 应用。 分别用 omx_encode 和 omx_decoder 替换了 omx_encoder.exe 和 omx_decoder.exe 应用。 将流支持从 8 增加到 32。
2018 年 6 月 6 日	1.1	常规更新。
2018 年 4 月 4 日	1.1	更新控制软件 API。 添加了配置 VCU 核的说明。 更新了支持的配置信息和选项。 将 gop-freq-idr 更改为 periodicity-idr。 对示例应用的使用情况进行了记录。
2017 年 12 月 20 日	1.0	重新组织了内容并更新了 API。
2017 年 10 月 4 日	1.0	初始版本。

请阅读：重要法律提示

本文向贵司/您所提供的信息（下称"资料"）仅在对赛灵思产品进行选择和使用时参考。在适用法律允许的最大范围内：（1）资料均按"现状"提供，且不保证不存在任何瑕疵，赛灵思在此声明对资料及其状况不作任何保证或担保，无论是明示、暗示还是法定的保证，包括但不限于对适销性、非侵权性或任何特定用途的适用性的保证；且（2）赛灵思对任何因资料发生的或与资料有关的（含对资料的使用）任何损失或赔偿（包括任何直接、间接、特殊、附带或连带损失或赔偿，如数据、利润、商誉的损失或任何因第三方行为造成的任何类型的损失或赔偿），均不承担责任，不论该等损失或者赔偿是何种类或性质，也不论是基于合同、侵权、过失或是其他责任认定原理，即便该损失或赔偿可以合理预见或赛灵思事前被告知有发生该损失或赔偿的可能。赛灵思无义务纠正资料中包含的任何错误，也无义务对资料或产品说明书发生的更新进行通知。未经赛灵思公司的事先书面许可，贵司/您不得复制、修改、分发或公开展示本资料。部分产品受赛灵思有限保证条款的约束，请参阅赛灵思销售条款：https://china.xilinx.com/legal.htm#tos；IP 核可能受赛灵思向贵司/您签发的许可证中所包含的保证与支持条款的约束。赛灵思产品并非为故障安全保护目的而设计，也不具备此故障安全保护功能，不能用于任何需要专门故障安全保护性能的用途。如果把赛灵思产品应用于此类特殊用途，贵司/您将自行承担风险和责任。请参阅赛灵思销售条款：https://china.xilinx.com/legal.htm#tos。

关于与汽车相关用途的免责声明

如将汽车产品（部件编号中含"XA"字样）用于部署安全气囊或用于影响车辆控制的应用（"安全应用"），除非有符合 ISO 26262 汽车安全标准的安全概念或冗余特性（"安全设计"），否则不在质保范围内。客户应在使用或分销任何包含产品的系统之前为了安全的目的全面地测试此类系统。在未采用安全设计的条件下将产品用于安全应用的所有风险，由客户自行承担，并且仅在适用的法律法规对产品责任另有规定的情况下，适用该等法律法规的规定。