

云 MO 监控

宋羽;

李书扬;

吴子静;

第一部分 设计概述

1.1 设计目的

我们的作品 云 MO 监控 对应的赛题为：**基于高云 FPGA 的网络视频监控系统**，因此，我们的首要目标在于充分发挥 Sipeed Tang Primer 20K 板卡的性能和资源，构建一能够迎合满足赛题需求、同时致力创新发挥，实现较为优秀的网络视频监控系统。我们主要着力于：

1. **以太网发送**：利用底板 RMii 网卡芯片实现 UDP 以太网传输
2. **摄像头适配**：利用 OV5640 实现视频采集
3. **视频处理**：使用高云软核+自主编写 Video Processing 模块实现

1.2 应用领域

云 MO 监控系统具有广泛的应用领域，能够通过通过配套使用兼容的镜头模块、同时通过配置 CMOS 寄存器，能够达到不同的**画幅、视角区域**，具有较为良好的**视角可定制性**。能够灵活**适用于各类监控场景**。包括但不限于：家庭安防（一般画幅、分辨能力）、商业区域监控（广画幅、低分辨能力）、仓库管理监控（窄画幅、高分辨能力）等。

同时，云 MO 监控系统突出了其独特的“**云墨模式**”特点。通过巧妙地结合板卡处理，系统能够通过仅保留视频灰度，巧妙地压缩数据，从而提高了网络传输效率和容错能力，优化了监控图像的流畅度。这一创新特点将监控系统的性能提升与数据传输的高可靠性相结合，同时能够借此实现二值化、边缘检测等，能够较好地**适应光纤较弱、画面干扰较大等情况**的时候，仍能通过板卡自主判断、获取可靠的物品信息。因此，也可以用在特种作业、暗光环境等场所。

1.3 主要技术特点

云 MO 监控系统的技术特点体现在多个方面。

1. 首先，系统利用高云 FPGA 板卡实现了高性能视频采集+摄像头寄存器高度定制化，能够追踪特定场景、配置相应参数，确保监控图像质量。
2. 其次，通过支持多路摄像头接入，系统提供了更丰富的监控视角和更全面的场景覆盖。同时，我们不仅在理论支撑中可以通过座子复用、开关切换的方式进行视

角的切换；同时，也通过完整的 UDP 网络协议实现，让单终端能够通过上位机设置同时访问不同的视角。

3. 再次，在网络传输方面，支持通用 RJ45 百兆网口，使得数据传输更为迅速和可靠。同时技术成熟、并且帧率达到了该接口条件下的较高水平（稳定在 $19fps$ ）。
4. 最后，在数据处理方面，系统实现了“云墨模式”，能够将图片进行二值化、灰度处理、基本人体识别、边缘检测等，同时应用中值滤波软核等方式、实现了对数据的高度精准和有效处理，为用户提供了更为清晰和有用的监控信息。

1.4 关键性能指标

在关键性能指标部分，为服务于上述设计目的等内容，我们最终实现如下性能指标：

1. **20fps** 视频传输，在使用 CMOS 原生 5-6-5 RGB 进行颜色发送时，能够具有较好、稳定的流畅度。
2. **标准 VGA 清晰度** 实现了基于标准 640×480 分辨率的帧结构
3. **多上位机画面接收** 通过完整的 UDP 协议和底层协议支持，能够在单个终端当中，实现多画幅接收，借助多下位机实现多路收发。
4. **满速百兆以太网**：能够通过满速的百兆以太网进行数据图像传输。

1.5 主要创新点

1. **在底层实现完整的 ARP+UDP 协议、自动网络链路获取**，支持自由收发，可以通过多上位机和地址设置实现多摄像头同时工作、显示实现。
2. **基本图像处理**，自行构建图像处理接口软核，使用标准的 vga 传输时序对于图像处理的过程进行封装，能够较好地添加、删改新的软核；并加入按键状态机进入模式切换。
3. **通用滤波模块**：提供通用的滤波、减少资源消耗，实现较好的二值化、边缘识别等图像处理效果。
4. **单键操作**：使用单按键加状态机进行模式切换，增加易用性、减少操作难度。

第二部分 系统组成及功能说明

2.1 整体介绍

根据设计，本组绘制了如下系统框图，并将在之后根据框图进行各部分的详细讲解：

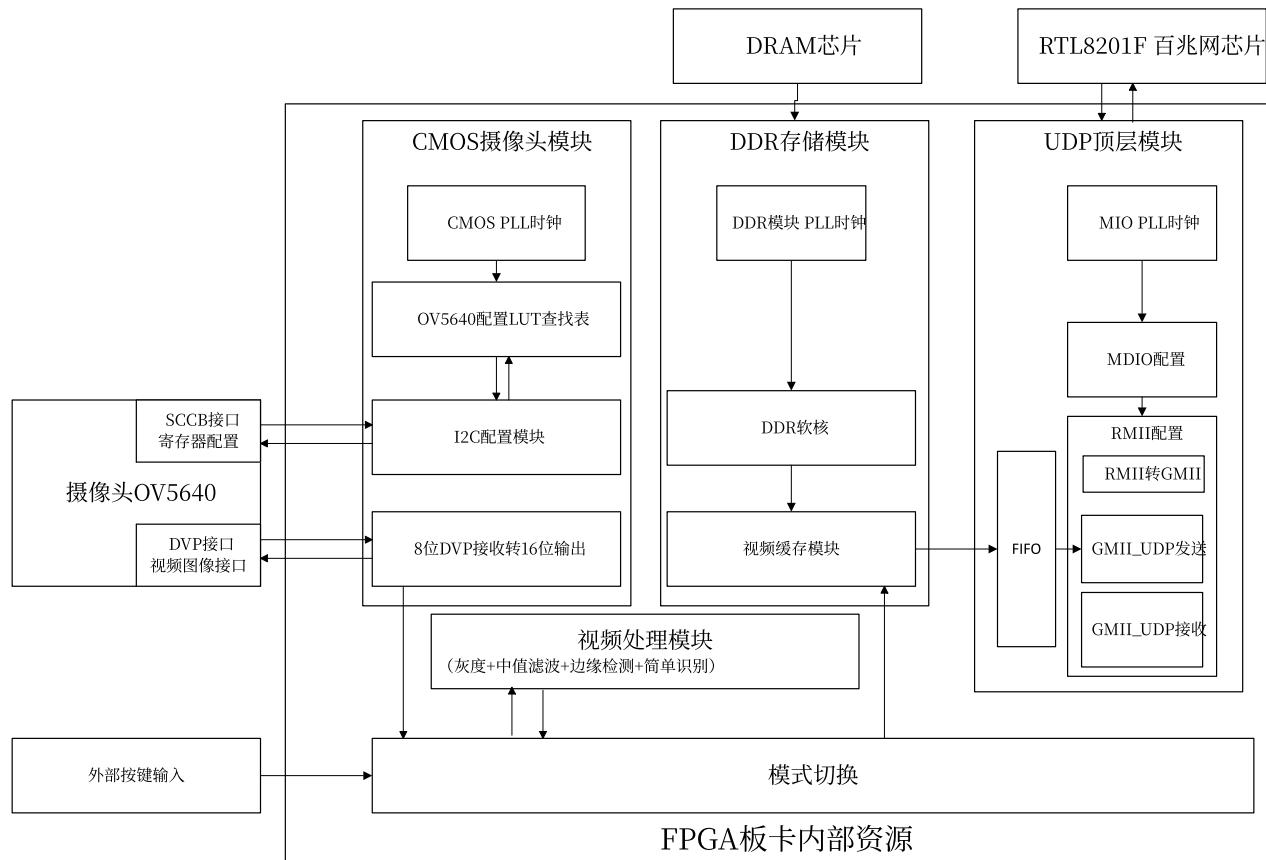


图 2.1 图 1：总体系统框图

可能乍一看这个系统框图会显得比较庞大、难以分辨每一部分，但其实每个部分的功能相当清晰和明确。在接下来的整体说明当中，我将整体介绍使用到的外设和作用。

2.1.1 框图一览

根据如上图 2.1, 可以较好地看到我们的系统主要依托 FPGA 芯片和几个基本的片外外设——其中，外设在框图中的位置为 *FPGA* 板卡内资源的大框之外的四个小框，即为：

表 2.1 表 1：片外外设和对应接口模块一览

片外外设名称	外部按键输入	摄像头 OV5640	DRAM 芯片	RTL8201F 百兆网芯片
片内模块对应	模式切换	CMOS 摄像头模块	DDR 存储模块	UDP 顶层模块
外设实现功能	使用 单按键 对于模式进行切换	配置外部摄像头/转存摄像头捕获数据	使用 DRAM 生成视频缓存	配置以太网向上位机发送
代码中模块名称	module_shift	cmos_8_16bit 和 i2c_config	DDR3_Memory_Interface_Top_inst	udp_top

2.1.1.1 主要外设

根据表 2.1 所示，面对各个外设、FPGA 板卡主要完成的任务大略如下：

1. **外部按键输入**: 通过常用的按键滤波、消抖操作，实现使用单键切换状态、及时进入下一状态的操作。
2. **摄像头 OV5640**: 本次我们使用 CMOS 芯片 *OV5640* 作为采集视频信号的摄像头芯片，其拥有 SCCB 接口和 DVP 接口提供配置和数据传输。
3. **DRAM 芯片**: 本次使用的底板中，拥有一块 DDR3 的 DRAM，结合高云提供的 *Video_Frame_Buffer* 软核，构建一个含有 3 帧的视频缓存。使用近似于 VGA 标准的同步信号进行数据的传输，用于接收并行视频输入数据，然后缓存至存储器，并同时输出并行视频数据，从而实现帧缓存的功能。[1]
4. **RTL8201F**: 使用底板提供的 *RTL8201F* 百兆以太网芯片，用于实现物理层的接口转换和信号调理。它支持 10/100M 自适应以太网速率，并具备自动协商功能，能够根据连接的设备自动选择最佳的速率。该芯片通过提供标准的 MII（介质独立接口）和 RMII（简化的介质独立接口）接口，与主控芯片（如单片机）进行通信。[2]

2.1.1.2 外设实物图

综合本次外设和实际应用情况，我们在这里给出实物图、并标注了如上四个外设所在的位置如：图 2.2

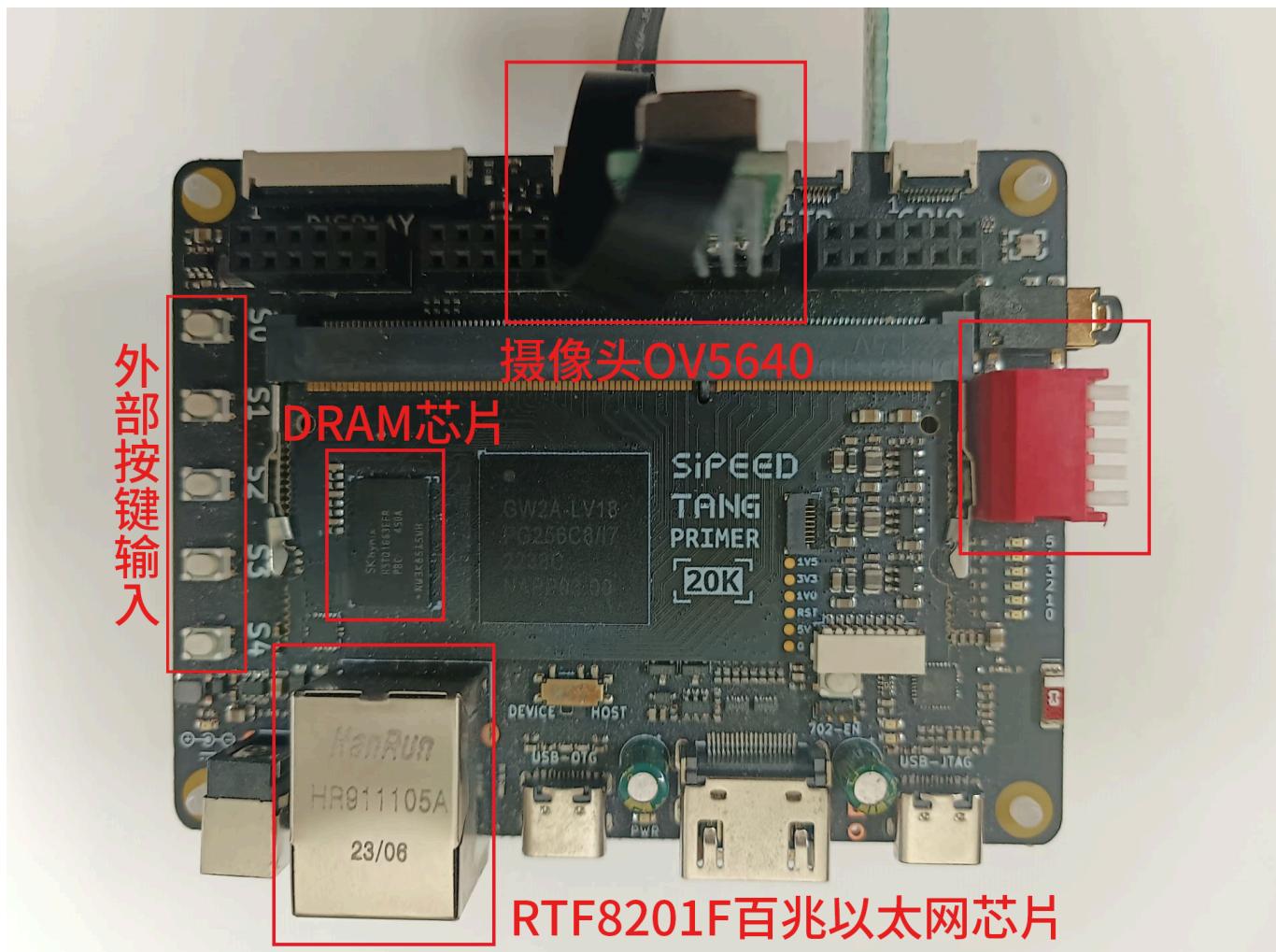


图 2.2 图 2: 主要功能外设标注

2.1.2 内部模块

由于熟练使用外设即可完成大部分基础功能，内部模块的撰写主要作用为：增强和融合。

在连接部分当中，一般情况下，多为外部外设/软核的输出的位宽不一致、因此会采取 FIFO 等方式、重新设计数据位宽度、让生成与等待异步的数据能够被较好地放入另一个时钟当中。因此、这一部分由于出现在两个外设支持模块相互连接的地方、故被不再被单独列出。

而在增强部分中，我们主要只考虑 **视频处理模块**：在这一部分中，我们会进行需要进行的工作，完成如 表 2.2 所示的全部功能，

表 2.2 表 2: 视频处理模块功能

功能名称	灰度处理	二值化	边缘检测
------	------	-----	------

2.1.2.1 视频上位机模块

为实现功能，我们构建了如图 2.3 所示的上位机模块：

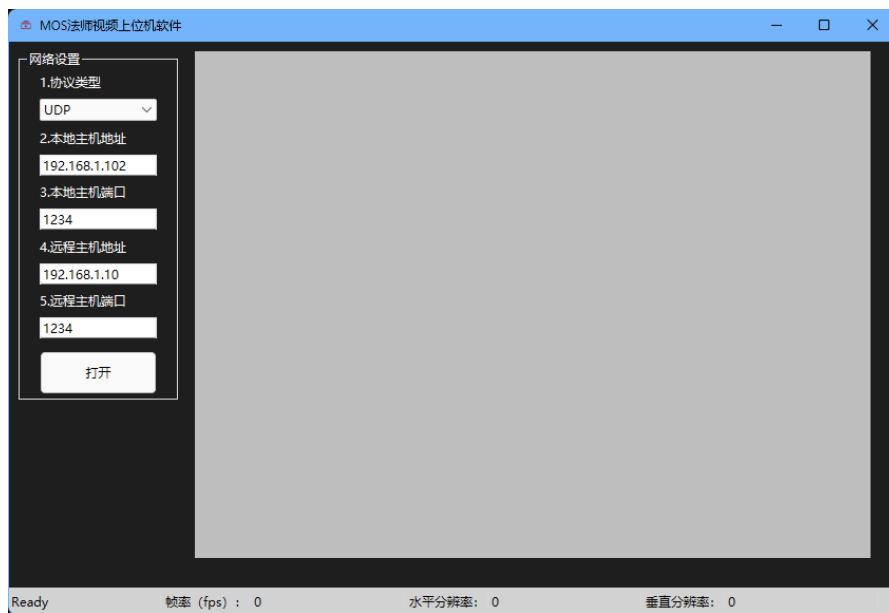


图 2.3 图 3：MOS 法师上位机软件

在这个上位机中，能够通过较为简单地调节左侧的各项设置进行接收模式的修改，同时在软件的下方，也具有较为简洁明了的帧率指示、分辨率指示。

上位机使用的接收 UDP 数据格式为：

1. **帧头**：如果该次发送为首次发送，则会十六进制发送 0xf05aa50f 四个字节,进行表示：这个 UDP 报文为需要被上位机解析的报文。而上位机也会继续解析余下这一帧的全部报文。
2. **分辨率信息**：如果该次发送为首次发送，则会接着帧头按照 16 进制转换发送分辨率信息 640×480 , 表示本帧的分辨率。
3. **帧内容**：以太网会按照行内容发送帧内容，每行的长度为 640 个标准 RGB5-6-5 的数据。

因此，如果是首行，则会发送长度为：1288 的数据，如果是后续的 479 行，则会发送数据长度为 1280 的数据、因为少了一个分辨率的信息。可以使用 Wireshark 软件进行以太帧的捕获，UDP 报文情况详见图 2.4：

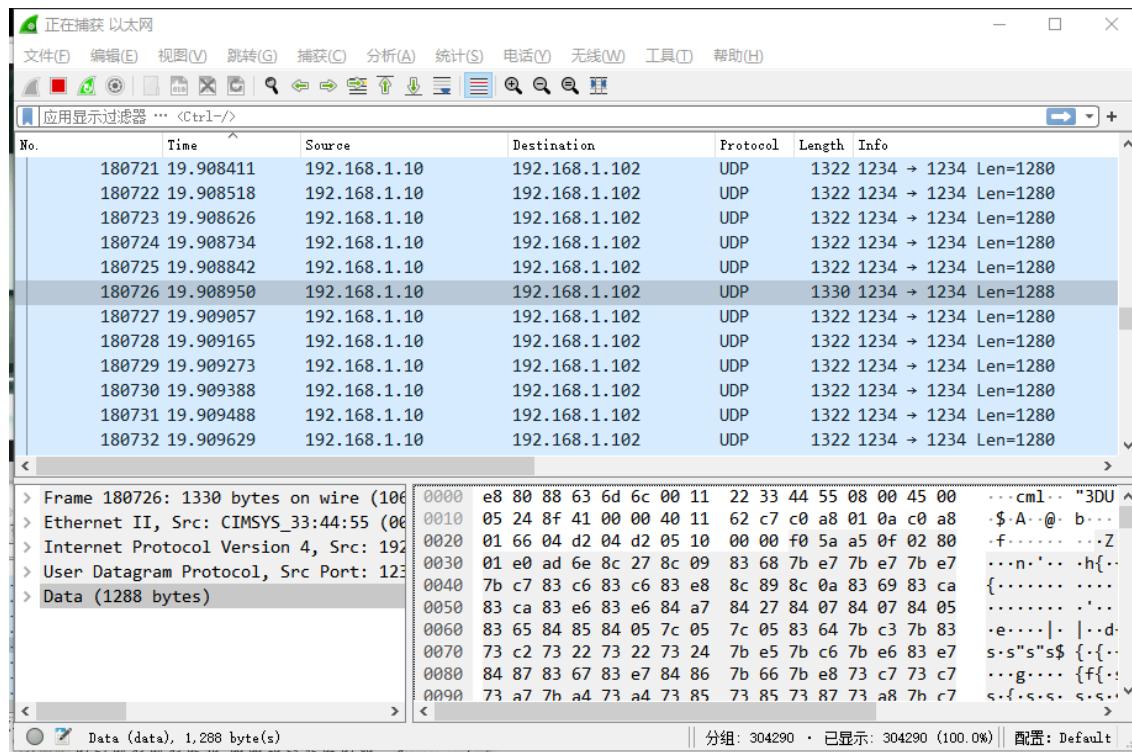


图 2.4 图 4：以太网抓帧 UDP 报文表

2.2 各模块介绍

2.2.1 UDP 模块

udp_top 模块是一个多功能的数字系统，涵盖了时钟管理、数据缓冲和以太网通信功能。

首先我们来看它的物理接口：在以太网芯片当中，用以连接以太网 MAC 层和 PHY 芯片的常用接口有：MII、RMII、SMII、GMII、RGMII。[3]

在本次 UDP 的实验当中、我们使用的芯片 RTL8201F 使用的是以太网当中的 RMII 接口，图 2.5 为 RMII 接口连接示意图。

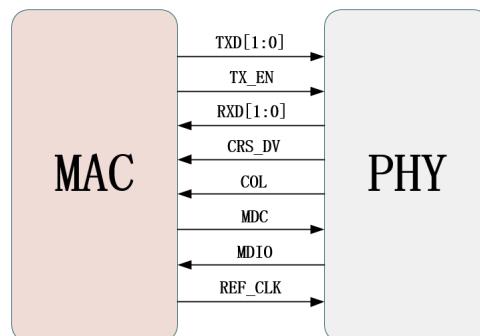


图 2.5 图 5：RMII 物理接口示意

可以从图 2.5 当中看出，RMII 只需 7 根通信，在功能上却能够与 MII 是相同的。因此，我们构建如下的硬件框图结构 图 2.6 让 RMII 能够发送 UDP 报文：

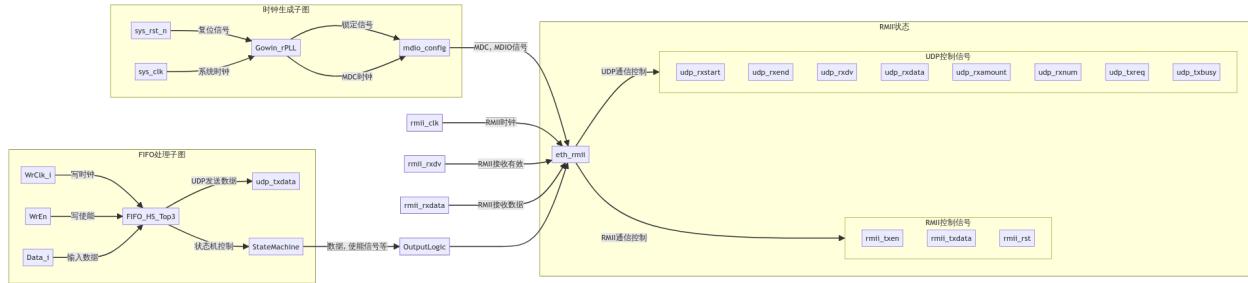


图 2.6 图 6: UDP 模块框图

可以看到，RMII 模块的主要工作有两个：

- 控制 UDP 通信：**其中，通信的 UDP 部分使用 RMII 转 GMII 的模块进行完成——原因在于 GMII 拥有 8 条并行数据线，而 RMII 只有两条。使用 GMII 数据线转 RMII 可以有效地减少发送需要的时钟周期，在 PLL 时钟较低的同时就能够达到较高的发送速率。
- 管理 RMII 状态：**向其他上游逻辑模块，如控制 FIFO 的状态机 图 2.7 进行状态汇报，管理状态。

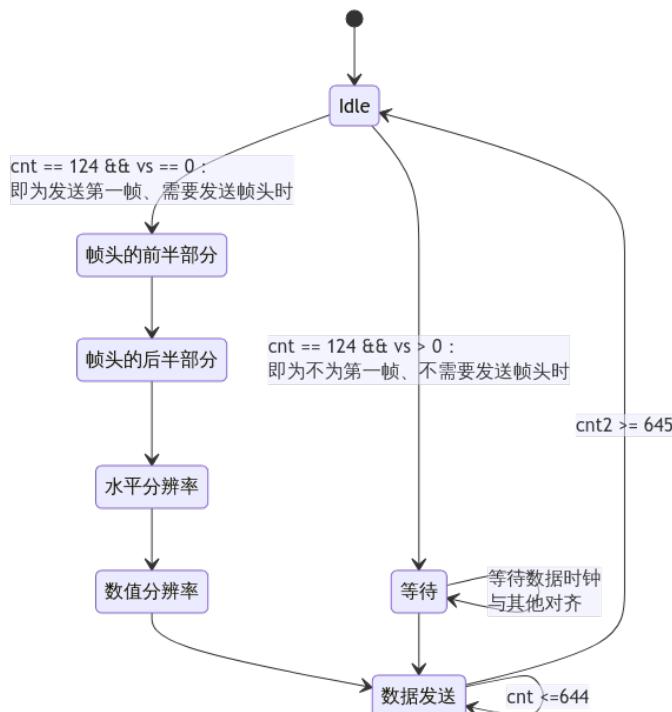


图 2.7 图 7: UDP 模块状态转移图

在状态转移中可以看到，在发送时，主要在数据发送状态会从 FIFO 当中获取来自 IP 核缓存的以太网信息。

- 而在 **头一帧** 当中，则会在几个状态之中，发送此帧需要的各种帧头信息。
- 在其他帧当中，则会在等待时钟对齐之后，直接将数据发送到 UDP 模块当中。

注意：进入到 Idle 状态的条件为：当从 RMII 模块传回的信息表示 RMII 不忙碌时。

2.2.2 CMOS 图像收集模块

在 CMOS 模块当中，我们使用 OV5640 作为 CMOS，其上有两种总线接口，作用如下：

- SCCB 接口**：时序上和 I^2C 接口一致，作用为配置 *OV5640* 芯片的相关功能即向摄像头寄存器写入对应指令。由于需要配置较多寄存器，对于 *OV5640* 摄像头具体配置信息，这里不再列举，具体的有，比如：修改输出图像的格式、画幅等。
- DVP 接口**：DVP (Digital Video Port) 是非差分信号的并口传输，在本次实验当中、数据位为有 8bit，是。在配置寄存器之后，按照较为标准的 VGA 时序等进行发送、并使用同步信号等判断是否已经接收完成一帧。[4]

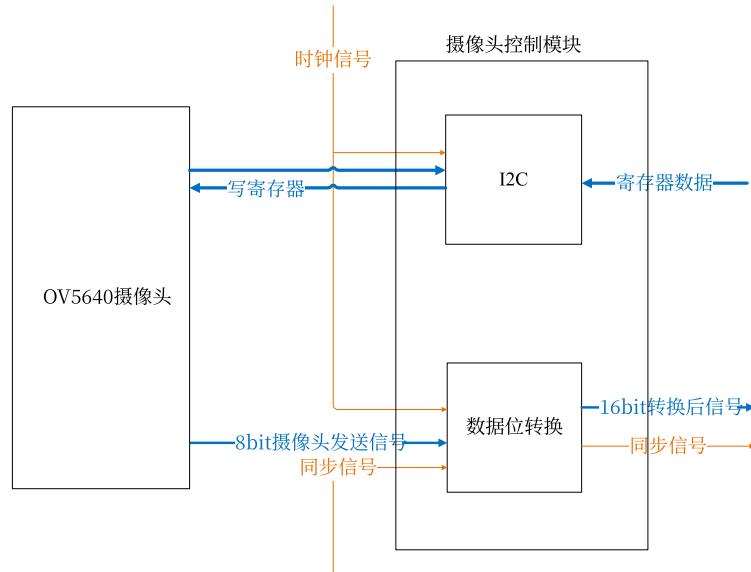


图 2.8 图 8: CMOS 配置和数据传输框图

由于这里 DVP 使用的位宽为硬件限制的 8 位，而如果使用高云的硬核作为视频缓存 Buffer，则至少需要使用 16bit 的数据位宽。因此需要使用 **数据位转换** 模块让数据能够进入缓存。

2.2.3 图像存储/数据流转移模块

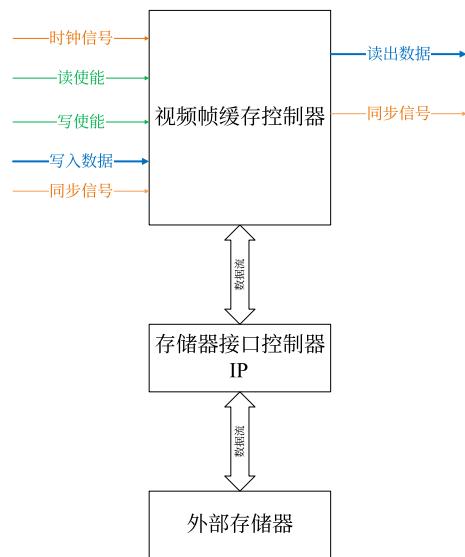


图 2.9 图 9: 存储模块系统框图

输入行缓存控制电路接收并行视频输入数据，然后缓存到输入 FIFO 行缓存中。当 FIFO 中数据存储到预先设定的阈值时，就向仲裁控制器发出写请求。当仲裁控制器响应请求，给与存储器控制权，则开始发送写数据，地址和命令。[1]

2.2.4 图像处理模块

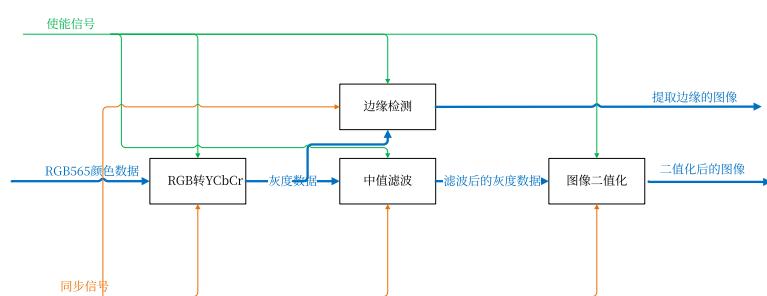


图 2.10 图 10: 图像数据处理模块

2.2.4.1 二值化部分

图像二值化 是将像素点的灰度值设为最大（白色）或最小（黑色），以显示明显的黑白效果。以 8bit 灰度图像为例，二值化通过选取阈值将亮度高于阈值的像素点设为白色（255），低于阈值的设为黑色（0），突出图像的特征。^[5]

二值化的实现方法有两种：手动指定阈值和自适应阈值（如 OTSU 算法）。手动方法计算量小，速度快但适用性有限；自适应方法适用性强，能清晰显示图像轮廓，但计算更复杂。本作品采用手动阈值方法，适用于特定物体与背景灰度差异显著的情况。

在本次作品当中，我们使用 OV5640 摄像头采集 RGB565 数据，转化为 Ycbcr 格式后进行二值化。

其模块连接如 图 2.11 所示：

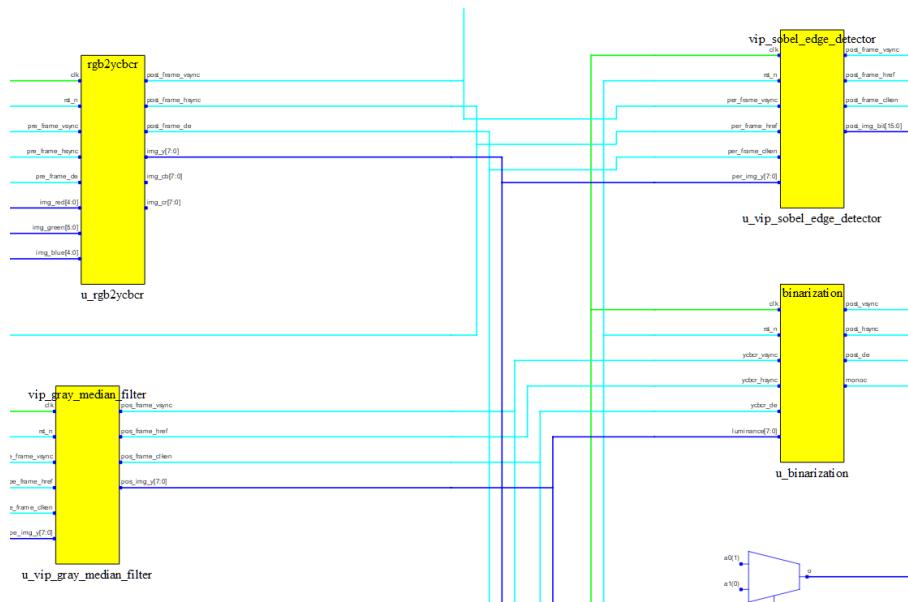


图 2.11 图 11: VIP 内部连接图

在对于二值化图像的发送中，并没有刻意对图像进行压缩，这也降低了误码对于发送的影响。虽然同时也较为浪费带宽。

2.2.4.2 边缘检测部分

在边缘检测部分，主要依托于二值化打下的二值化图像的基础进行处理，根据连接图 图 2.11 可以看出，在二值化后，图像会先进入一个中值滤波，随后进入 SOBEL 边缘检测算法模块，进行边缘检测的计算。

SOBEL 算法的实现可见 式 7.1，中值滤波的实现可见。这里放入附录当中。

2.2.5 模式切换模块

在模式切换模块当中，我们主要使用按键信号作为状态信号切换的标志，其实现也较为简单，只要通过对于下降沿的检测、简单滤波，判断是否释放即可实现。

在状态切换之后，直接输出选择需要输出的信号即可。

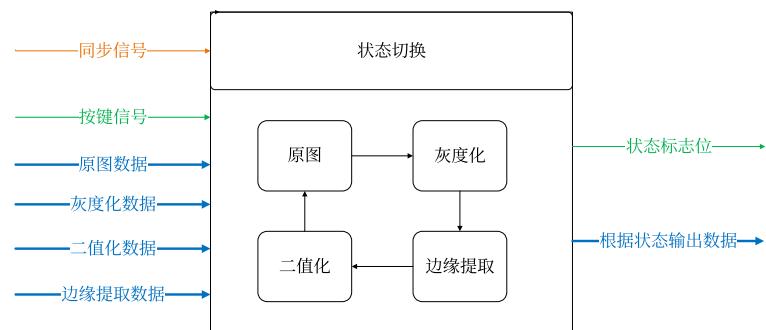


图 2.12 图 12：模式切换模块框图

第三部分 完成情况及性能参数

我们将赛题对于性能指标的基础和扩展要求、结合我们自身对于项目的理解分列的条目进行整合，并且列表见表 3.1：

表 3.1 表 1：关键性能指标

项目名称	实现情况	是否完成	指标详情
使用高云 FPGA 板卡	使用板卡为： <i>Tang Primer 20K</i>	✓	高云 FPGA (<i>GW2A-LV18PG256C8/I7</i>)
实现摄像头视频传输功能	使用 MIPI 接口实现摄像头并能对应不同场景进行寄存器配置	✓	CMOS 芯片 (<i>OV5640</i>)
网络传输能力	使用底板板载网卡：实现百兆以太网功能	✓	百兆网 (使用芯片 <i>RTL8201F</i>)
理论视频传输分辨率	综合考虑帧率，使用标清等清晰度传输	○	图像尺寸为 640 × 480
视频传输帧率	为保证百兆正常观看，使帧率设置为较为流畅的帧率	✓	帧率为 $\geq 19\text{fps}$
视频编码方式	由于加入特殊图像处理后不足，使用标准的 5-6-5 RGB 编码方式	✗	原始编码 但抗干扰能力强
多路传输	可通过完整的 UDP 在两块板卡上设置不同 IP 实现	✓	一主多从实现
图像增强功能	提供二值化、简单目标识别 边缘识别 滤波处理等功能	✓	提供多种
可扩展性	可定制画幅 空余大量 IO 有通用视频处理接口模块	○	资源足够，未做尝试

第四部分 总结

4.1 可扩展之处

在项目的实施中，尽管我们取得了一系列令人满意的成果，但我们也深刻认识到一些未能达到的目标，这为未来的可扩展性提供了一些有益的方向。以下是一些未能实现的目标以及对应的可扩展性考虑：

- 更高级的图像检测功能：尽管我们实现了基本的人体检测功能，但在未来的扩展中，可以考虑集成更高级的图像检测算法，如物体识别、行为分析等，以提升监控系统的智能化水平。
- 单 FPGA 多路摄像头：当前的设计虽然支持多路摄像头接入，但考虑到监控系统可能需要更大规模的部署，未来的扩展方向可以包括优化硬件设计，实现单一 FPGA 处理多路摄像头输入，提高系统整体的可伸缩性。
- SD 卡存储：在实际应用中，对于长时间的监控数据存储是一个重要考虑因素。未来的系统可扩展性可以包括将存储介质扩展至 SD 卡，以满足更大容量的数据存储需求。
- 更高分辨率和图像压缩：针对监控系统对于图像清晰度和网络传输效率的不断追求，未来可考虑进一步提高系统支持的分辨率和优化图像压缩算法，以适应更高要求的监控场景。

4.2 心得体会

在项目的完成过程中，我们不仅仅实现了基本的视频监控功能，更加深入地思考并解决了实际应用中可能遇到的挑战。以下是我们在项目中的一些心得体会：

- 创新与实用的平衡：通过引入云墨模式、通用滤波模块等创新特性，我们在提高系统性能的同时，注重了实际应用的实用性。创新点的引入并非为了炫技，而是为了解决实际场景中可能遇到的问题，提高系统的适用性。
- 用户体验至上：在设计阶段我们特别关注了用户体验，通过单键操作、模式切换状态机等设计，使系统更加易用。用户可以方便地切换模式，增加了系统的操作便捷性，提升了用户体验。
- 丰富的互联网和厂商支持必不可少：高云优秀的器件资源和 IP 核支持为我们的项目提供了强大的基础，极大地降低了我们的工作难度和压力。我们能够充分利用高

云半导体提供的先进技术，更专注于项目的创新和功能实现，使我们的设计更加高效、稳定。而网络上关于高云的资料也不少，让我们能够快速迁移。

在总结这一阶段的工作时，我们对于项目的成果感到欣喜，同时也意识到在不断发展的技术领域中，我们需要不断学习和创新，以更好地满足未来监控系统的需求。这次比赛是一个宝贵的经验，让我们更深刻地理解了团队协作和项目管理的重要性。希望我们的努力能够为未来的监控系统领域带来一些有益的启示。

第五部分 参考文献

- [1] Gowin, 《Gowin Video Frame Buffer》. 见于: 2023 年 11 月 15 日. [在线]. 载于: http://www.gowinsemi.com.cn/enrollment_view.aspx?TypeId=67&Id=742&FId=t27:67:27#IP
- [2] 普通网友, 《PHY 芯片 RTL8201F 的基本原理及单片机应用-CSDN 博客》. 见于: 2023 年 11 月 15 日. [在线]. 载于: https://blog.csdn.net/UoEmacs_Lisp/article/details/133032262
- [3] 野火, 《19. 以太网数据回环实验 — [野火]FPGA Verilog 开发实战指南——基于 Altera EP4CE10 征途 Pro 开发板 文档》. 见于: 2023 年 11 月 16 日. [在线]. 载于: https://doc.embedfire.com/fpga/altera/ep4ce10_pro/zh/latest/code/enet.html
- [4] 电子小白菜, 《CMOS Image Sensor 的 DVP 接口硬件设计》. 见于: 2023 年 11 月 15 日. [在线]. 载于: <https://zhuanlan.zhihu.com/p/141140876>
- [5] 正点原子, 《【正点原子 FPGA 连载】第二十六章基于 OV5640 的二值化实验领航者 ZYNQ 之嵌入式开发指南》. 见于: 2023 年 11 月 16 日. [在线]. 载于: <https://zhuanlan.zhihu.com/p/217535018>

第六部分 附录

6.1 边缘检测算法推理

6.1.1.1 SOBEL 算法

在边缘检测算法当中，我们使用的是较为经典的 SOBEL 算法：

1. 先求 x, y 方向的梯度 dx, dy
2. 然后求出近似梯度

$$G = dx^2 + dy^2 \quad (7.1)$$

然后开根号，也可以为了分别计算近似为

$$G = | dx | + | dy | \quad (7.2)$$

3. 最后根据 G 的值，来判断该点是不是边缘点，是的话，就将该点的像素复制为 255，否则为 0，当然 0 或 255 可以自己随意指定，也可以是其他两个易于区分的像素值。

6.1.1.2 中值滤波算法

同时，为了保证 SOBEL 算法的实现，我们进行了中值滤波的引入：

中值滤波方法是，对待处理的当前像素，选择一个模板，该模板为其邻近的若干个像素组成，对模板的像素由小到大进行排序，再用模板的中值来替代原像素的值的方法。

当我们使用 3×3 窗口后获取邻域中的 9 个像素，就需要对 9 个像素值进行排序，为了提高排序效率，排序算法思想如下图所示：

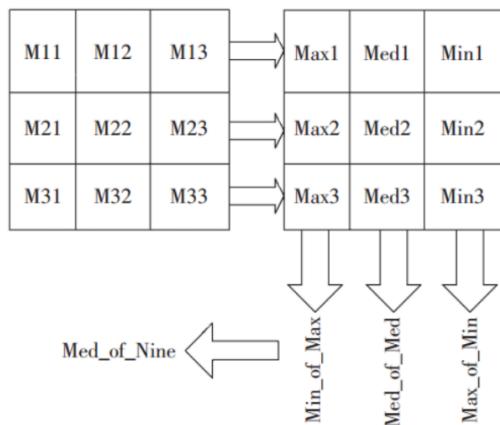


图 7.1 图 1：中值滤波思想

1. 对窗内的每行像素按降序排序，得到最大值、中间值和最小值；
2. 把三行的最小值相比较，取其中的最大值；
3. 把三行的最大值相比较，取其中的最小值；
4. 把三行的中间值相比较，再取一次中间值；
5. 把前面得到的三个值再做一次排序，获得的中值即该窗口的中值。

6.2 重要代码

1. 在这里，我们仅仅展示 top 模块当中的代码，以保证报告中能对我们的作品有一个较为全面且容易理解的系统构建展示：

```

module top (
    input    clk,
    input    rst_n,
    // inout      cmos_scl,    //cmos i2c clock
    // inout      cmos_sda,    //cmos i2c data
    input    cmos_vsync, //cmos vsync
    input    cmos_href, //cmos hsync refrence,data valid
    input    cmos_pclk, //cmos pixel clock
    output   cmos_xclk, //cmos externel clock
    input [7:0] cmos_db, //cmos data
    output   cmos_rst_n, //cmos reset
    output   cmos_pwdn, //cmos power down

    output [1:0] state_led,

    output [14-1:0] ddr_addr, //ROW_WIDTH=14
    output [3-1:0] ddr_bank, //BANK_WIDTH=3
    output   ddr_cs,
    output   ddr_ras,
    output   ddr_cas,
    output   ddr_we,
    output   ddr_ck,
    output   ddr_ck_n,
    output   ddr_cke,
    output   ddr_odt,
    output   ddr_reset_n,
    output [2-1:0] ddr_dm, //DM_WIDTH=2
    inout   [16-1:0] ddr_dq, //DQ_WIDTH=16
    inout   [2-1:0] ddr_dqs, //DQS_WIDTH=2
    inout   [2-1:0] ddr_dqs_n, //DQS_WIDTH=2
    //RMII 接口信号
    input wire    rmii_clk,
    input wire    rmii_rxdv,
    input wire [1:0] rmii_rxdata,
```

```

output wire      rmii_txen,
output wire [ 1:0] rmii_txdata,
output wire      rmii_RST,
inout wire      mdc_SDL, //SDL
inout wire      mdio_SDA, //SDA

input wire keyshift
);

//memory interface
wire      memory_clk;
wire      dma_clk;
wire      DDR_pll_lock;
wire      cmd_ready;
wire [ 2:0] cmd;
wire      cmd_en;
wire [ 5:0] app_burst_number;
wire [ ADDR_WIDTH-1:0] addr;
wire      wr_data_rdy;
wire      wr_data_en; //
wire      wr_data_end; //
wire [ DATA_WIDTH-1:0] wr_data;
wire [DATA_WIDTH/8-1:0] wr_data_mask;
wire      rd_data_valid;
wire      rd_data_end; //unused
wire [ DATA_WIDTH-1:0] rd_data;
wire      init_calib_complete;

//According to IP parameters to choose
`define WR_VIDEO_WIDTH_16
`define DEF_WR_VIDEO_WIDTH 16

`define RD_VIDEO_WIDTH_16
`define DEF_RD_VIDEO_WIDTH 16

`define USE_THREE_FRAME_BUFFER 1

`define DEF_ADDR_WIDTH 28
`define DEF_SRAM_DATA_WIDTH 128
//=====
//=====SRAM parameters
parameter ADDR_WIDTH      = `DEF_ADDR_WIDTH; //存储单元是 byte, 总容量=2^27*16bit = 2Gbit,增加 1 位
rank 地址, {rank[0],bank[2:0],row[13:0],column[9:0]}
parameter DATA_WIDTH       = `DEF_SRAM_DATA_WIDTH; //与生成 DDR3IP 有关, 此 ddr3 2Gbit, x16, 时钟比

```

例 1:4，则固定 128bit

```
parameter WR_VIDEO_WIDTH = `DEF_WR_VIDEO_WIDTH;
parameter RD_VIDEO_WIDTH = `DEF_RD_VIDEO_WIDTH;

wire      video_clk; //video pixel clock

wire      off0_syn_de;
wire [RD_VIDEO_WIDTH-1:0] off0_syn_data;

wire [ 15:0] cmos_16bit_data;
wire      cmos_16bit_clk;
wire [ 15:0] write_data;

wire [ 9:0] lut_index;
wire [ 31:0] lut_data;

assign cmos_xclk = cmos_clk;
assign cmos_pwdn = 1'b0;
assign cmos_rst_n = 1'b1;
assign write_data = {cmos_16bit_data[4:0], cmos_16bit_data[10:5], cmos_16bit_data[15:11]};

//状态指示灯
// assign state_led[3] =
assign state_led[1] = rst_n; //复位指示灯
assign state_led[0] = init_calib_complete; //DDR3 初始化指示灯

// wire cmos_clk;
wire out_de;
wire cmos_sdl;
wire cmos_sda;
wire mdc;
wire mdio;
assign mdc_sdl = cmos_sdl;
assign mdio_sda = cmos_sda;

//generate the CMOS sensor clock and the SDRAM controller clock
cmos_pll cmos_pll_m0 (
    .clkin (clk),
    .clkout(cmos_clk)
);

mem_pll mem_pll_m0 (
    .clkin (clk),
    .clkout(memory_clk),
    .lock (DDR_pll_lock)
```

```
);

//I2C master controller
i2c_config i2c_config_m0 (
    .rst      (~rst_n),
    .clk      (clk),
    .clk_div_cnt (16'd500),
    .i2c_addr_2byte(1'b1),
    .lut_index  (lut_index),
    .lut_dev_addr (lut_data[31:24]),
    .lut_reg_addr (lut_data[23:8]),
    .lut_reg_data (lut_data[7:0]),
    .error     (),
    .done      (),
    .i2c_scl   (cmos_sdl),
    .i2c_sda   (cmos_sda)
);

//configure look-up table
lut_ov5640_rgb565_640_480 lut_ov5640_rgb565 (
    .lut_index(lut_index),
    .lut_data (lut_data)
);

//CMOS sensor 8bit data is converted to 16bit data
cmos_8_16bit cmos_8_16bit_m0 (
    .rst  (~rst_n),
    .pclk (cmos_pclk),
    .pdata_i(cmos_db),
    .de_i  (cmos_href),
    .pdata_o(cmos_16bit_data),
    .hblank (cmos_16bit_wr),
    .de_o   (cmos_16bit_clk)
);
wire post_frame_de;
wire post_frame_vsync;
wire [15:0] post_rgb;
wire post_frame_de1;
wire post_frame_vsync1;
wire [15:0] post_rgb1;
wire post_frame_de2;
wire post_frame_vsync2;
wire [15:0] post_rgb2;
//图像处理模块
vip u_vip (
```

```

//module clock
.clk      (cmos_pclk),    // 时钟信号
.rst_n    (rst_n),        // 复位信号 (低有效)

//图像处理前的数据接口
.pre_frame_vsync(cmos_vsync),
.pre_frame_hsync(cmos_href),
.pre_frame_de (cmos_16bit_wr),
.pre_rgb    (write_data),

//图像处理后的数据接口
.post_frame_vsync(post_frame_vsync), // 场同步信号
.post_frame_hsync(),               // 行同步信号
.post_frame_de (post_frame_de),   // 数据输入使能
.post_rgb     (post_rgb),

.post_frame_vsync1(post_frame_vsync1), // 场同步信号
.post_frame_hsync1(),               // 行同步信号
.post_frame_de1 (post_frame_de1),   // 数据输入使能
.post_rgb1    (post_rgb1),

.post_frame_vsync2(post_frame_vsync2), // 场同步信号
.post_frame_hsync2(),               // 行同步信号
.post_frame_de2 (post_frame_de2),   // 数据输入使能
.post_rgb2    (post_rgb2)

);

wire de_out;
wire vs_out;
wire [15:0] data_out;
module_shift u_module_shift (
    .clk  (cmos_16bit_clk),
    .clk2 (cmos_pclk),
    .rstn (rst_n),
    .keyshift(keyshift),

    .de_1  (cmos_16bit_wr),
    .vs_1  (cmos_vsync),
    .data_1 (write_data),
    .de_2  (post_frame_de),
    .vs_2  (post_frame_vsync),
    .data_2 (post_rgb),
    .de_3  (post_frame_de1),
    .vs_3  (post_frame_vsync1),

```

```

    .data_3 (post_rgb1),
    .de_4  (post_frame_de2),
    .vs_4  (post_frame_vsync2),
    .data_4 (post_rgb2),

    .de_out (de_out),
    .vs_out (vs_out),
    .data_out(data_out)
);

Video_Frame_Buffer_Top Video_Frame_Buffer_Top_inst (
    .l_rst_n (init_calib_complete), //rst_n      ),
    .l_dma_clk(dma_clk),          //sram_clk    ),
`ifdef USE_THREE_FRAME_BUFFER
    .l_wr_halt(1'd1),           //1:halt, 0:no halt
    .l_rd_halt(1'd1),           //1:halt, 0:no halt
`endif

// video gary data input
    .l_vin0_clk   (cmos_16bit_clk),
    .l_vin0_vs_n  (~vs_out),     //只接收负极性
    .l_vin0_de    (de_out),
    .l_vin0_data   (data_out),
    .O_vin0_fifo_full(),

// video data output
    .l_vout0_clk   (rmii_clk),
    .l_vout0_vs_n  (~out_vs),    //只接收负极性
    .l_vout0_de    (out_de),
    .O_vout0_den   (off0_syn_de),
    .O_vout0_data   (off0_syn_data),
    .O_vout0_fifo_empty(),

// ddr write request
    .l_cmd_ready    (cmd_ready),
    .O_cmd         (cmd),        //0:write; 1:read
    .O_cmd_en      (cmd_en),
    .O_app_burst_number (app_burst_number),
    .O_addr        (addr),       // [ADDR_WIDTH-1:0]
    .l_wr_data_rdy  (wr_data_rdy),
    .O_wr_data_en   (wr_data_en),  //
    .O_wr_data_end   (wr_data_end),  //
    .O_wr_data      (wr_data),     // [DATA_WIDTH-1:0]
    .O_wr_data_mask  (wr_data_mask),

```

```

    .l_rd_data_valid  (rd_data_valid),
    .l_rd_data_end    (rd_data_end),      //unused
    .l_rd_data        (rd_data),         //DATA_WIDTH-1:0]
    .l_init_calib_complete(init_calib_complete)
);

DDR3MI DDR3_Memory_Interface_Top_inst (
    .clk          (rmii_clk),
    .memory_clk   (memory_clk),
    .pll_lock     (DDR_pll_lock),
    .rst_n        (rst_n),             //rst_n
    .app_burst_number (app_burst_number),
    .cmd_ready    (cmd_ready),
    .cmd          (cmd),
    .cmd_en       (cmd_en),
    .addr         (addr),
    .wr_data_rdy  (wr_data_rdy),
    .wr_data      (wr_data),
    .wr_data_en   (wr_data_en),
    .wr_data_end  (wr_data_end),
    .wr_data_mask (wr_data_mask),
    .rd_data      (rd_data),
    .rd_data_valid (rd_data_valid),
    .rd_data_end  (rd_data_end),
    .sr_req       (1'b0),
    .ref_req      (1'b0),
    .sr_ack       (),
    .ref_ack      (),
    .init_calib_complete(init_calib_complete),
    .clk_out      (dma_clk),
    .burst        (1'b1),

    // mem interface
    .ddr_rst     (),
    .O_ddr_addr  (ddr_addr),
    .O_ddr_ba    (ddr_bank),
    .O_ddr_cs_n  (ddr_cs),
    .O_ddr_ras_n (ddr_ras),
    .O_ddr_cas_n (ddr_cas),
    .O_ddr_we_n  (ddr_we),
    .O_ddr_clk   (ddr_ck),
    .O_ddr_clk_n (ddr_ck_n),
    .O_ddr_cke   (ddr_cke),
    .O_ddr_odt   (ddr_odt),
    .O_ddr_reset_n(ddr_reset_n),

```

```
.O_ddr_dqm (ddr_dm),
.IO_ddr_dq (ddr_dq),
.IO_ddr_dqs (ddr_dqs),
.IO_ddr_dqs_n (ddr_dqs_n)
);
```

```
udp_top udp_top (
    .sys_rst_n(rst_n),
    .sys_clk (clk),
    .rmii_clk(rmii_clk),
    .rmii_rxdv(),
    .rmii_rxdata(),
    .rmii_txen(rmii_txen),
    .rmii_txdata(rmii_txdata),
    .rmii_RST(rmii_RST),
    .mdc(mdc),
    .mdio(mdio),
    .WrClk_i(rmii_clk),
    .WrEn(off0_syn_de),
    .Data_i(off0_syn_data),
    .de(out_de),
    .outvs(out_vs)
);
endmodule
```

1. 具体模块代码详见压缩文档，且基本功能易于实现、故不具体展示。

2. 代码生成的 RTL 图像见 图 7.2：

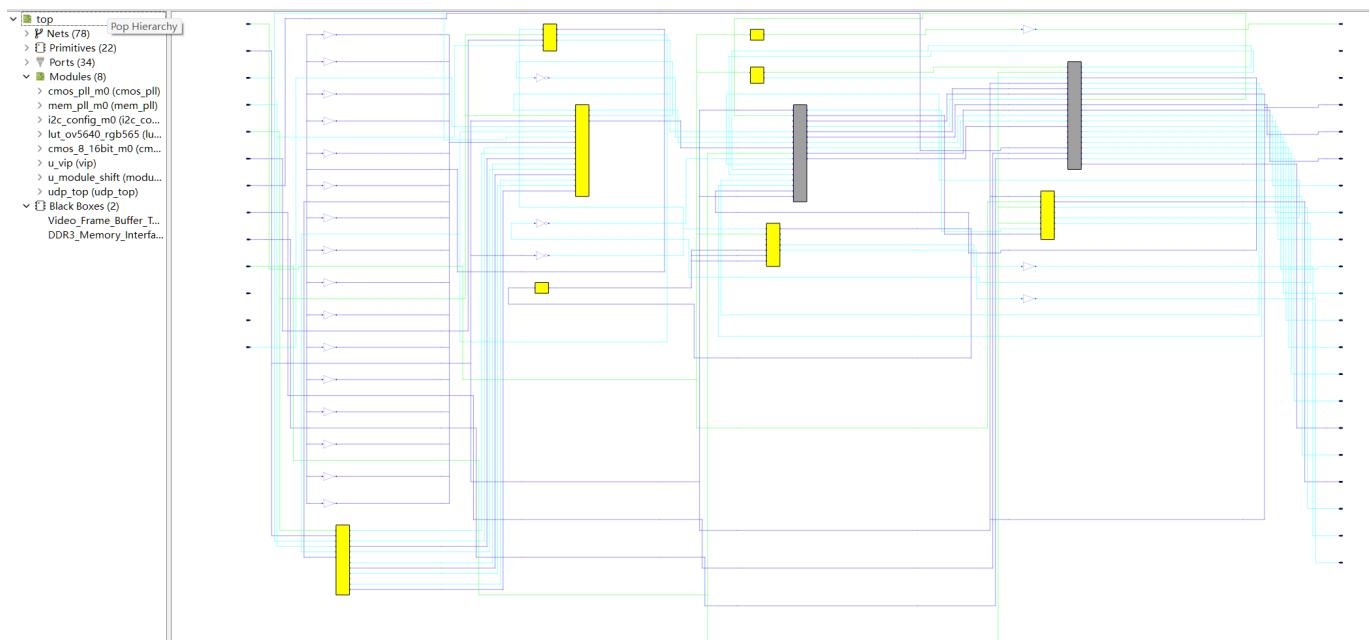


图 7.2 图 2: RTL 图像

6.3 工程资源使用报告

- 时钟资源:

Clock Summary:

Clock Name	Type	Period	Frequency(Hz)	Rise	Fall	Source	Master	Object
clk	Base	37.037	27.0	0.000	18.519			clk_ibuf/1
cmos_pdk	Base	10.000	100.0	0.000	5.000			cmos_pdk_ibuf/1
rmi_clk	Base	10.000	100.0	0.000	5.000			rmi_clk_ibuf/1
cmos_8_16bit_m0/cmos_16bit_clk_1	Base	10.000	100.0	0.000	5.000			cmos_8_16bit_m0/de_o_s0/Q
udp_top/your_instance/fifo_inst/n4_6	Base	10.000	100.0	0.000	5.000			udp_top/your_instance/fifo_inst/n4_s2/O
udp_top/your_instance/fifo_inst/n9_6	Base	10.000	100.0	0.000	5.000			udp_top/your_instance/fifo_inst/n9_s2/O
udp_top/your_instance/fifo_inst/wfull_val	Base	10.000	100.0	0.000	5.000			udp_top/your_instance/fifo_inst/wfull_val_s12/F
gw_gao_inst_0/u_icom_top/n19_6	Base	10.000	100.0	0.000	5.000			gw_gao_inst_0/u_icom_top/n19_s2/O
gw_gao_inst_0/u_la0_top/n15_6	Base	10.000	100.0	0.000	5.000			gw_gao_inst_0/u_la0_top/n15_s2/O
u_module_shift/n6_6	Base	10.000	100.0	0.000	5.000			u_module_shift/n6_s2/O
u_module_shift/n39_11	Base	10.000	100.0	0.000	5.000			u_module_shift/n39_s7/F
cmos_pll_m0/rpll_inst/CLKOUT/default_gen_clk	Generated	41.667	24.0	0.000	20.833	clk_ibuf/1	clk	cmos_pll_m0/rpll_inst/CLKOUT
cmos_pll_m0/rpll_inst/CLKOUTP/default_gen_clk	Generated	41.667	24.0	0.000	20.833	clk_ibuf/1	clk	cmos_pll_m0/rpll_inst/CLKOUTP
cmos_pll_m0/rpll_inst/CLKOUTD/default_gen_clk	Generated	83.333	12.0	0.000	41.667	clk_ibuf/1	clk	cmos_pll_m0/rpll_inst/CLKOUTD
cmos_pll_m0/rpll_inst/CLKOUTD3/default_gen_clk	Generated	125.000	8.0	0.000	62.500	clk_ibuf/1	clk	cmos_pll_m0/rpll_inst/CLKOUTD3
mem_pll_m0/rpll_inst/CLKOUT/default_gen_clk	Generated	2.511	398.3	0.000	1.255	clk_ibuf/1	clk	mem_pll_m0/rpll_inst/CLKOUT
mem_pll_m0/rpll_inst/CLKOUTP/default_gen_clk	Generated	2.511	398.3	0.000	1.255	clk_ibuf/1	clk	mem_pll_m0/rpll_inst/CLKOUTP
mem_pll_m0/rpll_inst/CLKOUTD/default_gen_clk	Generated	5.022	199.1	0.000	2.511	clk_ibuf/1	clk	mem_pll_m0/rpll_inst/CLKOUTD
mem_pll_m0/rpll_inst/CLKOUTD3/default_gen_clk	Generated	7.533	132.8	0.000	3.765	clk_ibuf/1	clk	mem_pll_m0/rpll_inst/CLKOUTD3
udp_top/u_rpl/rpll_inst/CLKOUT/default_gen_clk	Generated	83.333	12.0	0.000	41.667	clk_ibuf/1	clk	udp_top/u_rpl/rpll_inst/CLKOUT
udp_top/u_rpl/rpll_inst/CLKOUTP/default_gen_clk	Generated	83.333	12.0	0.000	41.667	clk_ibuf/1	clk	udp_top/u_rpl/rpll_inst/CLKOUTP
udp_top/u_rpl/rpll_inst/CLKOUTD/default_gen_clk	Generated	500.000	2.0	0.000	250.000	clk_ibuf/1	clk	udp_top/u_rpl/rpll_inst/CLKOUTD
udp_top/u_rpl/rpll_inst/CLKOUTD3/default_gen_clk	Generated	250.000	4.0	0.000	125.000	clk_ibuf/1	clk	udp_top/u_rpl/rpll_inst/CLKOUTD3
DDR3_Memory_Interface_Top_inst/gw3_top/4/fdkdiv/CLKOUT.default_gen_clk	Generated	10.044	99.6	0.000	5.022	mem_pll_m0/rpll_inst/CLKOUT	mem_pll_m0/rpll_inst/CLKOUT.default_gen_clk	DDR3_Memory_Interface_Top_inst/gw3_top/4/fdkdiv/CLKOUT

- 消耗的 FPGA 片上资源:

Resource Utilization Summary

Resource	Usage	Utilization
Logic	11940(7464 LUT, 672 ALU, 634 RAM16) / 20736	58%
Register	3159 / 16173	20%
--Register as Latch	19 / 16173	<1%
--Register as FF	3140 / 16173	20%
BSRAM	44 / 46	95%