## 一种基于嵌入式超算平台的车载网络原型系统

目录

[1. 引言 2](#_Toc68905981)

[1.1 背景与思路 2](#_Toc68905982)

[1.2 项目准备 3](#_Toc68905983)

[2. 功能描述 3](#_Toc68905984)

[3. 总体架构 3](#_Toc68905985)

[3.1 FPGA加速卡总体设计 4](#_Toc68905986)

[3.2 FPGA I/O卡总体设计 5](#_Toc68905987)

[4. 详细设计 6](#_Toc68905988)

[4.1 关键数据结构格式设计 6](#_Toc68905989)

[4.1.1 基于UDP的通信消息格式设计 6](#_Toc68905990)

[4.1.2 消息格式设计 6](#_Toc68905991)

[4.2 FPGA加速卡详细设计 7](#_Toc68905992)

[4.2.1 Controller模块详细设计 7](#_Toc68905993)

[4.2.1.1 Controller模块整体框架 8](#_Toc68905994)

[4.2.1.2 Controller模块接口设计 8](#_Toc68905995)

[4.2.2 Camera Adapter模块详细设计 9](#_Toc68905996)

[4.2.2.1 Camera Adapter模块整体框架 9](#_Toc68905997)

[4.2.2.2 Camera Adapter模块接口设计 9](#_Toc68905998)

[4.2.2.3 Camera Adapter模块逻辑设计 10](#_Toc68905999)

[4.2.3 Monitor Adapter模块详细设计 11](#_Toc68906000)

[4.2.3.1 Monitor Adapter模块整体框架 11](#_Toc68906001)

[4.2.3.2 Monitor Adapter模块接口设计 12](#_Toc68906002)

[4.2.3.3 Monitor Adapter模块逻辑设计 12](#_Toc68906003)

[4.2.4 I/O Switch模块详细设计 14](#_Toc68906004)

[4.2.4.1 I/O Switch模块整体框架 14](#_Toc68906005)

[4.2.4.2 I/O Switch模块接口设计 15](#_Toc68906006)

[4.2.4.3 I/O Switch模块逻辑设计 16](#_Toc68906007)

[4.2.4.3.1.1 Switch\_TX模块设计 16](#_Toc68906008)

[4.2.4.3.1.2 Switch\_RX模块设计 17](#_Toc68906009)

[4.2.5 D\_TreeAccels模详细设计 17](#_Toc68906010)

[4.2.5.1 D\_TreeAccels模块整体设计 17](#_Toc68906011)

[4.2.5.2 D\_TreeAccels接口设计 18](#_Toc68906012)

[4.2.5.3 D\_TreeAccels逻辑设计 18](#_Toc68906013)

[4.3 FPGA I/O卡详细设计 24](#_Toc68906014)

[4.3.5 Eth- Converter1模块详细设计 25](#_Toc68906015)

[4.3.5.1 Eth- Converter1整体框架 25](#_Toc68906016)

[4.3.5.2 Eth- Converter1模块接口 25](#_Toc68906017)

[4.3.5.2.2 DSM模块接口 25](#_Toc68906018)

[4.3.5.2.3 DRM模块接口 26](#_Toc68906019)

[4.3.5.3 Eth- Converter1模块设计 26](#_Toc68906020)

[4.3.6 Arbiter模块详细设计 28](#_Toc68906021)

[4.3.6.1 Arbiter整体框架 29](#_Toc68906022)

[4.3.6.2 Arbiter模块接口 29](#_Toc68906023)

[4.3.6.2.2 USM接口设计 30](#_Toc68906024)

[4.3.6.2.3 URM接口设计 30](#_Toc68906025)

[4.3.6.3 Arbiter模块逻辑设计 30](#_Toc68906026)

[4.3.7 Eth\_Converter2详细设计 32](#_Toc68906027)

[4.3.7.1 Eth\_Converter2整体框架 32](#_Toc68906028)

[3.2.1.1 Eth\_Converter2模块接口 33](#_Toc68906029)

[3.2.1.2 Eth\_Converter2模块逻辑设计 33](#_Toc68906030)

[4.3.8 HDMI converter详细设计 33](#_Toc68906031)

[4.3.9 CamControl模块 34](#_Toc68906032)

|  |  |  |  |
| --- | --- | --- | --- |
| 文档版本号 | 修改人 | 修改时间 | 备注 |
| 1.0 | 杨翔瑞，杨凌 | 2021.04.03 | 初始版本 |
| 1.1 | 韩楚浩 | 2021.04.19 | 增加加速器设计模块 |
|  |  |  |  |

该设计方案用于实现一种加速逻辑与I/O接口分离的基于级联决策树的目标识别机制，用于对摄像头所采集图像中的目标（人脸等）进行识别，并在外接显示器中进行显示与定位。其中，I/O设备（摄像头，显示器等）外接于拥有多类型接口的低端FPGA（AX7050开发板），核心加速算法运行于后端Vertex 7 690T终端开发板。两块板卡使用以太网互联，实现加速与业务逻辑分离的目标识别的演示应用。

### 引言

### 背景与思路

嵌入式超算是在体积、重量和功耗受限条件下，通过CPU，FPGA和DSP的异构计算为卫星、飞机和汽车等智能装备平台提供强大算力的重要手段。

2013年， TTTech公司针对集成模块化航电系统的设计需求，提出了面向关键任务执行的嵌入式云概念[1]，将虚拟化和云计算的思想引入装备平台计算中；2017年，中科院软件所首次提出了天基超算概念[2]，提出采用以太网互连多核CPU和FPGA，基于异构计算为软件定义卫星提供强大的算力资源。嵌入式超算系统是在嵌入式场景下，通过引入“FPGA即服务”，“敏捷交换”等技术，为新一代装备平台电子信息系统上多样化计算任务提供环境。总的来看，嵌入式场景下异构加速器应当具有具有高能效、低延迟、支持应用多样且易于开发等通用特点。另一方面，与大规模数据中心相比，嵌入式场景资源受限与实时业务种类丰富的特点又对加速器提出了特殊要求：

1. 深度优化需求：嵌入式场景的特点决定了应用需在资源受限条件下尽可能实现高性能处理。而这一般需要对应用与加速逻辑进行深度定制与优化。
2. 实时要求高：嵌入式平台通常拥有较丰富实时I/O设备（传感器，执行器），这要求加速器具有高速I/O和计算能力。
3. I/O接入类型多样化需求：嵌入式场景下传感器、执行器等接口接入类型丰富，加速器需能够易于兼容各类接口类型。



图1 基于嵌入式超算平台的车载网络原型系统

本目标识别框架即为在该场景下的一种典型应用，用于验证嵌入式超算场景下“FPGA即服务”的可行性。

### 项目准备

该方案基于https://github.com/WalkerLau/DetectHumanFaces，并对I/O逻辑与加速逻辑基于以太网进行解耦，从而支持I/O与加速逻辑相互独立的目标识别框架。

该系统利用基于FPGA的SoC架构，实现了对摄像头采集人脸数据的实时检测功能。将得到的人脸检测结果通过HDMI输出到显示器，在显示器上框出被检测到的人脸，并通过FPGA板载LED灯将人脸的数目以二进制的格式显示出来，同时，所检测到的人脸在图像中的位置、大小、检测时间等信息可以通过UART连接到计算机串口助手中显示。（具体设计方案见https://github.com/WalkerLau/DetectHumanFaces）

### 功能描述

### 总体架构

该架构将识别框架的I/O业务（视频采集，结果显示等）与加速逻辑以分布式方式部署在不同FPGA板卡上，并使用1Gbps以太网进行板卡之间互联，通过以太网接入的CPU板卡作为整个系统的控制平面。我们将在本章对FPGA加速卡与I/O卡总体设计与功能进行描述，并分别在第4章与第5章对FPGA加速卡与I/O卡逻辑进行详细设计。



图2 基于FPGA的目标识别设计架构图

### FPGA加速卡总体设计

FPGA加速卡拥有丰富的计算与存储资源，用于对I/O卡中传感器接收到的数据进行分析，并将所产生的指令或处理结果返回至I/O板中的执行器/监视器。在该项目中，FPGA加速卡的总体设计如图3所示。



图3 加速卡总体设计架构图

在“接收通路”上, MAC核用于接收来自I/O卡摄像机捕获的像素数据（封装在以太网报文中），并通过I/O\_Switch将数据交换至与Camera Adapter相连的Eth-AXI转接模块中。该Eth-AXI模块将用于对接收到的以太网报文进行解封装，并将像素数据以AXI4格式（32位）写入DDR的对应位置中。当Camera Adapter完成一帧图像写入后，会通知Controller模块。

当Controller模块被唤醒后，将会开始执行图像处理的控制流程，并以循环的方式触发算法加速模块运行图像识别算法完成目标检测。得到检测结果后，Controller将会产生一个摄像头控制报文用于触发位于I/O卡的摄像头重新收集一次视频帧（该报文由Controller还是I/O Switch产生目前待定）。

而在“发送/显示通路”上，如何保证图像的读写不冲突是需要解决的关键问题。仿照[1], 采用读写地址分离与交替变换的方式进行图像信息在DDR中的读写操作。简单来说，即为同一时刻DDR中存储两帧图像，读逻辑与写逻辑模块相同时间段内采用奇偶交替的方式仿真这两帧图像。

### FPGA I/O卡总体设计

I/O卡基于FPGA平台实现，用于数据的获取与输出。需要将摄像头获取的数据通过以太网传输到加速卡，同时将加速卡处理完的数据通过以太网接受并转换到HDMI显示器进行显示。为了能够更好地利用硬件资源，I/O卡统一处理与输入输出相关地设备接口。所以本章按照从摄像头的数据获取、处理以及传输和以太网数据的获取、处理和显示的流程，分别介绍I/O卡中的子模块及其详细设计。



图 4 I/O卡总体框图

上图为I/O卡的整体架构图，Camera接口出来的图像数据进入Eth-converter1模块进行数据的整合，然后被包装成UDP报文格式，通过Arbiter模块后发送给加速卡进行图像的识别加速计算。完成识别加速计算后的数据同样地通过MAC发送给Arbiter，Arbiter模块接收到数据后根据其报文包头信息将数据段发送给Eth-converter2模块进行处理。Eth-converter2模块接收到RGB565格式地像素信息后将其数据整合，根据index和像素点数量信息判断行同步信号和场同步信号，后以一个像素点为单位向下发送给缓冲器，后由HDMO converter将其中地像素点数据以及行、场同步信号转换成相应地HDMI接口信号，最后完成显示功能。

对于摄像头地初始化，同样地通过网络传输UDP报文进行，MAC接收到包含配置信息的UDP报文后，将其传送给Arbiter，Arbiter接收到后解析器报文头信息，判断其是发送给Eth-converter1的配置信息后，将后续的报文数据发送给Eth-converter1模块进行处理。Eth-converter接收到数据后根据具体的数据生成配置信号，然后发送给CamControl模块进行处理，CamControl模块内保存了与配置相关的所有命令，其通过I2C接口将配置命令发送给摄像头。以上就是整个I/O卡的工作流程。

### 详细设计

### 关键数据结构格式设计

### 基于UDP的通信消息格式设计

该架构中，我们使用UDP/IP进行数据平面数据传输，同时使用“in-band”方式支持各板卡间控制平面通信（寄存器配置，设备启停等）。



图4 基于UDP的像素数据格式设计

板卡间像素数据基于UDP报文格式进行传输，其数据格式如图4所示。其中type字段表示传输数据类型，使用0x01表示该数据为像素数据。index字段表示当前报文的1024B payload所携带的512个像素点在完整帧中的编号。padding字段用于补齐偏移便于对payload字段进行解析。需要注意的是，我们使用UDP源与目的端口(0x0000, 0x1111)标记摄像头回传数据，源与目的端口(0x1111, 0x0000)标记从加速卡到I/O卡的回传像素数据。

带内控制通路数据message格式与数据通路格式类似，区别为控制通路为变长格式，其定义如图5所示。其中type字段表示消息类型，目前定义控制报文的消息类型为0x00。



图5 基于UDP的控制报文格式设计

### 消息格式设计

### FPGA加速卡详细设计



图6 FPGA加速卡总体模块连接关系图

1. Controller通过“摄像机配置消息”初始化摄像头；
2. 初始状态下，Controller向MAC发送“摄像机请求消息”，从而触发I/O板摄像头采集像素信息（仅触发I/O端发送一帧数据）并通过以太网进行发送；
3. 加速卡接收到特定的UDP报文后，经由I/O\_Switch模块解析后转为256b的AXI-S数据类型输入至Camera Adapter模块，该模块记录当前报文携带payload的像素点index，并使用32b AXI接口将payload中的像素数据依次写入DDR；
4. 当该过程完成后（一帧图像写入），Camera Adapter模块向Controller模块发送1b的valid信号，告知Controller当前可对图像进行分析；
5. Controller接收到该信号后，将通过开关信号（inst）首先触发加速模块工作，同时通过read ready信号告知加速模块当前可读取DDR对应框中数据进行目标检测，另外Controller还会通过write address index信号将当前加速器模块需要处理的DDR数据的地址发送至加速模块；
6. 加速器模块根据地址获取DDR中对应数据并完成目标检测。完成后将布尔值返回给Controller模块用于在图像中绘制框图（标注已识别目标的位置）。

本节下述部分将对加速板各模块进行详细设计。

### Controller模块详细设计

Controller是加速板的核心控制模块，用于协同数据传输与加速器的运行。

【控制命令】

读写地址20b ---> [51:32]

Data字段32b ----> [31:0]

### Controller模块整体框架

Controller与Camera Adapter, Monitor Adapter, CPU Agent以及加速核心模块D\_Tree\_Accels互联，并协同整个加速板卡逻辑运行。

1. 首先Controller通过frm\_trigg信号(由almost\_finish&monitor\_adapter\_finish控制)触发CPU\_Agent，由CPU\_Agent构造报文触发远端camera发送一帧图像到DDR奇（偶）位置；
2. 当camera数据传输完成后，Controller通过ignite\_acc调度加速器执行识别算法；
3. 当加速器完成当前图像检测后，通过finish\_acc触发Controller。Controller通过对odd\_even\_ctrl信号取反从而对奇偶位置cursor进行切换；
4. Controller通过read\_valid信号触发monitor向远端发送一帧图像信息。

上述操作循环执行。



图7 Controller模块连接关系图

### Controller模块接口设计

表一 Controller模块输入输出信号定义表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名称** | **方向** | **宽度** | **信号描述** |
| frame\_trigger | output | 1 | 来自Controller的帧采集触发信号 |
| frame\_trigger\_ack | input | 1 | 来自CPU\_Agent的帧采集确认信号 |
| wirte\_finish | input | 1 | 来自CA的帧写入DDR完成信号 |
| odd\_even\_ctrl | output | 1 | 来自Controller的奇偶帧标识信号 |
| read\_valid | output | 1 | 来自Controller的帧读出标识信号 |
| read\_finish | input | 1 | 来自MA的帧读出完成信号 |
| finish\_ready | output | 1 | 来自controller的处理完成ready信号 |
| finish\_acc | input | 1 | 来自加速器的处理完成信号 |
| ignite\_acc | output | 1 | 来自controller的加速器开始执行信号 |
| ignite\_ready | input | 1 | 来自加速器的开始执行ready信号 |

### Camera Adapter模块详细设计

### Camera Adapter模块整体框架

Camera Adapter模块接收来自I/O Switch的包含像素数据的以太网报文，将以太网头剥离后，根据message头中的index数据将像素点通过AXI Interconnect写入DDR中。其整体框图如图8所示，其与 I/O Switch模块采用256位AXI-S总线连接，与AXI-Interconnect模块采用32b AXI写总线连接。



图8 Camera Adapter模块连接关系图

### Camera Adapter模块接口设计

表二 Camera Adapter模块输入输出信号定义表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名称** | **方向** | **宽度** | **信号描述** |
| s\_axis\_tdata | output | 32 | 来自I/O Switch的32位数据信号 |
| s\_axis\_tuser | output | 1 | 来自I/O Switch的1位带外metadata |
| s\_axis\_tvalid | output | 1 | 来自I/O Switch的有效信号 |
| s\_axis\_tkeep | output | 1 | 来自I/O Switch的有效字节标记信号 |
| s\_axis\_tlast | output | 1 | 来自I/O Switch的数据尾标识信号 |
| s\_axis\_tready | input | 1 | 来自Camera Adapter的反压信号 |
| m\_axi\_areset\_out\_n | input | 8 | Rst信号 |
| m\_axi\_aclk | input | 1 | 时钟信号 |
| m\_axi\_awid | output | 1 | 写地址ID，支持以group形式写地址 |
| m\_axi\_awaddr | output | 32 | 来自CA的写地址 |
| m\_axi\_awlen | output | 4 | 来自CA的burst length,（在burst模式下） |
| m\_axi\_awsize | output | 3 | 来自CA的一次传输的burst size |
| m\_axi\_awburst | output | 2 | 来自CA的burst类型信号 |
| m\_axi\_awlock | output | 2 | 来自CA的传输附加信息的信号 |
| m\_axi\_awcache | output | 1 | 来自CA的cache类型信号 |
| m\_axi\_awprot | output | 1 | 来自CA的保护类型信号 |
| m\_axi\_awqos | output | 1 |  |
| m\_axi\_awvalid | output | 1 | 来自CA的写地址有效信号 |
| m\_axi\_awready | input | 1 | 来自AXI-Interconnect的反压信号 |
| m\_axi\_wdata | output | 32 | 来自CA的写数据信号 |
| m\_axi\_wstrb | output | 4 | 来自CA的写数据频闪信号 |
| m\_axi\_wlast | output | 1 | 来自CA的写数据截止信号 |
| m\_axi\_wvalid | output | 1 | 来自CA的写数据有效信号 |
| m\_axi\_wready | input | 1 | 来自AXI-Interconnect的反压信号 |
| m\_axi\_bid | input | 1 | 来自A-I的写响应ID信号，需与awid匹配 |
| m\_axi\_bresp | input | 2 | 来自A-I的写响应，有4种状态 |
| m\_axi\_bvalid | input | 1 | 来自A-I的写响应有效信号 |
| m\_axi\_bready | output | 1 | 来自CA的写响应反压信号 |
| odd\_even\_ctrl | input | 1 | 来自controller的DDR写奇偶控制信号 |
| write\_finish | output | 1 | 来自CA的DDR写入完成信号 |

### Camera Adapter模块逻辑设计

Camera Adapter接收来自I/O Switch的32b AXIS输出信号，剥离message头后，根据像素点index信号将的数据转为32b的基于AXI4的像素数据，并将其写入DDR中。而由于AXI-Interconnect模块的存在（存在仲裁与反压机制），Camera Adapter需要使用FIFO/RAM读来自I/O Switch的数据进行缓存。因此，Camera Adapter模块的逻辑设计可通过两个通过FIFO连接独立状态机描述，分别控制FIFO的写入与FIFO的读出。



图9 Camera Adapter状态机1设计图

IDLE\_S：Camera Adapter状态机1的初始状态，若AXI-S输入valid信号被至高，则跳转至Depre\_S状态，准备处理报文头部新；

Depre\_S: Depre\_S连续丢弃AXI-S的前12拍数据，并在第13拍数据中提取index参数，确定当前payload需要DDR的位置并记录该值，然后将index 写入index FIFO中。该过程完成后跳转至Process\_S状态；

Process\_S：将从第14拍起始的数据推入数据FIFO中，完成全部1024B payload的处理后判断当前是否当前index是否等于599。若是，则跳转至IDLE\_S状态，否则跳转NEXT\_S状态；

NEXT\_S：为一帧数据未传输完成的中间状态（跳转条件与IDLE完全相同），主要用于debug时监测模块运行状态。



图10 Camera Adapter状态机2设计图

IDLE\_C：初始状态，当检测到数据FIFO非空且当前AXI4 ready置位时，跳转至DDRW\_C2/DDRW\_C2状态准备进行DDR写操作。需要注意的是，该模块通过来自Controller的1b odd\_even\_ctrl控制信号确定具体跳转至哪个状态；

DDRW\_C1：读取FIFO中32b像素数据并将offset\_cnt累加1，并将AXI4的写地址修改为index\_cnt x 256 + offset\_cnt（需要参考原设计奇偶乒乓地址生成机制），进行一次AXI4写地址操作，若FIFO依然非空，则循环执行此操作，直至index\_cnt = 599或FIFO非空，则跳转回IDLE状态。（DDRW\_C2与DDRW\_C1机制相同）

### Monitor Adapter模块详细设计

### Monitor Adapter模块整体框架

Monitor Adapter模块用于在Controller的控制下从DDR中读取处理后的像素数据，并将其发送至I/O Switch模块从而发送至I/O卡进行显示。其处理流程基本与Camera Adapter相反，其设计框图如图11所示：



图11 Monitor Adapter模块连接关系图

### Monitor Adapter模块接口设计

表三 Camera Adapter模块输入输出信号定义表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名称** | **方向** | **宽度** | **信号描述** |
| m\_axis\_tdata | output | 256 | 来自MA的256位数据信号 |
| m\_axis\_tuser | output | 1 | 来自MA的1位带外metadata |
| m\_axis\_tvalid | output | 1 | 来自MA的有效信号 |
| m\_axis\_tkeep | output | 1 | 来自MA的有效字节标记信号 |
| m\_axis\_tlast | output | 1 | 来自MA的数据尾标识信号 |
| m\_axis\_tready | input | 1 | 来自I/O Switch的反压信号 |
| m\_axi\_areset\_out\_n | input | 8 | Rst信号 |
| s\_axi\_aclk | input | 1 | 时钟信号 |
| s\_axi\_awid | input | 1 | 写地址ID，支持以group形式写地址 |
| s\_axi\_awaddr | input | 32 | 来自A-I的写地址 |
| s\_axi\_awlen | input | 4 | 来自A-I的burst length,（在burst模式下） |
| s\_axi\_awsize | input | 3 | 来自A-I的一次传输的burst size |
| s\_axi\_awburst | input | 2 | 来自A-I的burst类型信号 |
| s\_axi\_awlock | input | 2 | 来自A-I的传输附加信息的信号 |
| s\_axi\_awcache | input | 1 | 来自A-I的cache类型信号 |
| s\_axi\_awprot | input | 1 | 来自A-I的保护类型信号 |
| s\_axi\_awqos | input | 1 |  |
| s\_axi\_awvalid | input | 1 | 来自A-I的写地址有效信号 |
| s\_axi\_awready | output | 1 | 来自MA的反压信号 |
| s\_axi\_wdata | input | 32 | 来自A-I的写数据信号 |
| s\_axi\_wstrb | input | 4 | 来自A-I的写数据频闪信号 |
| s\_axi\_wlast | input | 1 | 来自A-I的写数据截止信号 |
| s\_axi\_wvalid | input | 1 | 来自A-I的写数据有效信号 |
| s\_axi\_wready | output | 1 | 来自MA的反压信号 |
| s\_axi\_bid | output | 1 | 来自MA的写响应ID信号，需与awid匹配 |
| s\_axi\_bresp | output | 2 | 来自MA的写响应，有4种状态 |
| s\_axi\_bvalid | output | 1 | 来自MA的写响应有效信号 |
| s\_axi\_bready | input | 1 | 来自A-I的写响应反压信号 |
| odd\_even\_ctrl | input | 1 | 来自controller的DDR写奇偶控制信号 |
| read\_valid | input | 1 | 来自controller的DDR读开始信号 |
| Read\_finish | output | 1 | 来自MA的读结束信号 |

### Monitor Adapter模块逻辑设计

Monitor Adapter首先接收来自controller的读开始消息，然后从DDR中依次读取数据并写入数据FIFO中，并进行计数。当计数达到1024B时（一个完整pkt payload），更新一次index-cnt值，直到读出完整一帧数据。当FIFO的读逻辑部分检查到FIFO非空并且index-cnt被更新后，其将会构造对应报文头（同时将对应的index-cnt写入）。并将数据FIFO中数据读出1024B，送至I/O Switch模块。因此，与Camera Adapter相似，Monitor Adapter也使用两个状态机分别控制数据FIFO的读写逻辑。



图12 Monitor Adapter状态机1设计图

状态机1用于控制数据FIFO的写逻辑部分。

IDLE\_S: 状态机1的起始状态，当收到来自controller的read\_start信号后，根据odd\_even\_ctrl值计算读地址的起始位置，并跳转到DDRR状态，准备进行DDR读操作，并将index\_rec寄存器重置为0（用于构造message头的index字段）；

DDRR\_S：该状态下，该模块将会从DDR由当前初地址寄存器作为地址开始，连续读取1024B像素数据，并将index\_rec寄存器值累加1，之后跳转到IDXA状态；

IDXA\_S: 该状态下更新当前初地址寄存器值（r\_read\_addr = r\_read\_addr + (1024B/4B)）检查index\_rec的值，若index\_rec != 599, 切换另一个数据FIFO，并跳转回DDRR状态继续向另一个FIFO继续写入1024B数据；若index\_rec == 599，则证明完成一帧数据写入，跳转回IDLE状态等待read\_start信号。



图13 Monitor Adapter状态机2设计图

状态机2用于控制数据FIFO的读出逻辑。

IDLE\_C：状态机2的起始状态，检查来自controller的odd\_even\_ctrl信号取反后对应FIFO当前是否为满（1024B），若该FIFO当前满，则跳转至HDR\_C状态；

HDR\_C: 构造并暂存向I/O Switch模块发送的报文头部分，构造完成后跳转至READ\_C状态；

READ\_C：从当前对应数据FIFO中连续读取报文并连同报文头进行发送。其中，消息头发送需要2拍（64B），报文体发送需要32拍（1024B），发送完成后跳转至UPDATE状态；

UPDATE\_C：（空状态，用于debug），直接跳转回IDLE状态。

### I/O Switch模块详细设计

### I/O Switch模块整体框架

I/O Switch用于将1G MAC核与FPGA加速板的核心逻辑进行互联。在TX通路上，I/O Switch模块将来自Monitor Adapter与Controller模块的2路位宽为256位的AXI-Stream信号转为1路8位的MAC输出信号；在RX通道上，I/O Switch模块接收来自MAC核的8位数据，并将其转为32b数据并将其发送至Camera Adapter模块（由于IMG封装在payload字段，因此待前13拍处理完成后，Camera Adapter才提取数据作为像素点写入DDR中）。因此，I/O Switch的模块连接关系图如图14所示。



图14 I/O Switch模块连接关系图



图15 I/O Switch内部模块连接关系图

### I/O Switch模块接口设计

表四 I/O Switch模块输入输出信号定义表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名称** | **方向** | **宽度** | **信号描述** |
| m1\_axis\_tdata | output | 8 | 来自I/O Switch的8位数据信号 |
| m1\_axis\_tuser | output | 1 | 来自I/O Switch的1位带外metadata |
| m1\_axis\_tvalid | output | 1 | 来自I/O Switch的有效信号 |
| m1\_axis\_tkeep | output | 1 | 来自I/O Switch的有效字节标记信号 |
| m1\_axis\_tlast | output | 1 | 来自I/O Switch的数据尾标识信号 |
| m1\_axis\_tready | input | 1 | 来自MAC的反压信号 |
| s1\_axis\_tdata | input | 8 | 来自MAC的8位数据信号 |
| s1\_axis\_tuser | input | 1 | 来自MAC的1位带外metadata |
| s1\_axis\_tvalid | input | 1 | 来自MAC的有效信号 |
| s1\_axis\_tkeep | input | 1 | 来自MAC的有效字节标记信号 |
| s1\_axis\_tlast | input | 1 | 来自MAC的数据尾标识信号 |
| s1\_axis\_tready | output | 1 | 来自I/O Switch的反压信号 |
| s2\_axis\_tdata | input | 256 | 来自Monitor Adapter的256位数据信号 |
| s2\_axis\_tuser | input | 1 | 来自Monitor Adapter的1位带外metadata |
| s2\_axis\_tvalid | input | 1 | 来自Monitor Adapter的有效信号 |
| s2\_axis\_tkeep | input | 1 | 来自Monitor Adapter的有效字节标记信号 |
| s2\_axis\_tlast | input | 1 | 来自Monitor Adapter的数据尾标识信号 |
| s2\_axis\_tready | output | 1 | 来自I/O Switch的反压信号 |
| s2\_axis\_tdata | input | 256 | 来自Controller的256位数据信号 |
| s2\_axis\_tuser | input | 1 | 来自Controller的1位带外metadata |
| s2\_axis\_tvalid | input | 1 | 来自Controller的有效信号 |
| s2\_axis\_tkeep | input | 1 | 来自Controller的有效字节标记信号 |
| s2\_axis\_tlast | input | 1 | 来自Controller的数据尾标识信号 |
| s2\_axis\_tready | output | 1 | 来自I/O Switch的反压信号 |
| m2\_axis\_tdata | output | 256 | 来自I/O Switch的256位数据信号 |
| m2\_axis\_tuser | output | 1 | 来自I/O Switch的1位带外metadata |
| m2\_axis\_tvalid | output | 1 | 来自I/O Switch的有效信号 |
| m2\_axis\_tkeep | output | 1 | 来自I/O Switch的有效字节标记信号 |
| m2\_axis\_tlast | output | 1 | 来自I/O Switch的数据尾标识信号 |
| m2\_axis\_tready | input | 1 | 来自Camera Adapter的反压信号 |

### I/O Switch模块逻辑设计

由图15所示，I/O Switch模块根据TX与RX通路区别被划分为Switch\_TX与Switch\_RX两个子模块。其中Switch\_TX接收2路256b的AXIS输入信号，将其转为1路8b的输出信号，并输出至MAC模块；Switch\_RX接收1路8b的AXIS输入信号，将其转为1路32b的输出信号，并输出到Camera Adapter模块。因此，本小节分别对Switch\_TX与Switch\_RX进行详细设计。

### Switch\_TX模块设计

根据上述功能描述，Switch\_TX应当包含两个子功能：1)完成2路到1路的数据多路选择（MUX模块）；2)完成256b AXIS输入信号到8b AXIS输出信号的转换（TX\_Converter模块）。

因此，Switch\_TX的子模块设计可入下图16所示。其中，MUX模块使用两个同步FIFO与Round-Robin调度机制完成2路转1路的工作，而MUX串接TX\_Converter模块，通过FIFO将1路256位AXI-S数据转为8位AXI-S输出数据。



图16 Switch\_TX内部模块连接关系图

**MUX**: MUX模块2路到1路转换，其状态机设计如图10所示。其中IDLE\_S循环检查两个FIFO的empty位，若当前轮询到的FIFO非空，则跳转至READ\_S状态将当前FIFO中的第一个packet读出，当该packet被完整读出后，切换当前需检查的FIFO的index，并返回IDLE\_S状态循环检查FIFO的empty位。



图17 MUX状态机设计图

**TX\_Converter**:由于X\_Converter模块所完成任务与方法较为直观，遂不再赘述设计细节。

### Switch\_RX模块设计

根据上述功能描述，Switch\_RX主要完成输入256b AXI-S数据到输出8b AXI-S数据的转化。该工作仅通过控制FIFO的读逻辑即可实现，细节在此不再赘述。

### D\_TreeAccels模详细设计

### 4.2.5.1 D\_TreeAccels模块整体设计

D\_TreeAccels的主要功能是实现决策树算法，对图像进行检测，其外部的连接如下图：

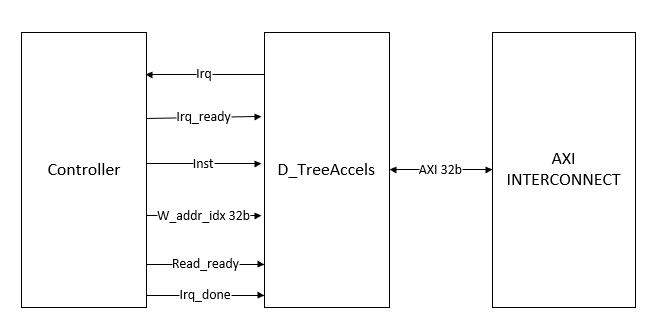


图18 D\_TreeAccels模块的外部连接

D\_TreeAccels与Controller相连，Controller控制D\_TreeAccels的执行；同时D\_TreeAccels向下通过AXI总线与DDR相连，DDR中存储着需要检测的图像信息。具体执行过程如下：

1. 首先整个系统启动时Controller通过Inst信号置1来唤醒D\_TreeAccels。
2. 当Read\_ready信号由0变为1之后，说明Controller通知D\_TreeAccels要处理的图像已经在DDR中存放好。而后D\_TreeAccels从W\_addr\_idx中读取要处理的信息在DDR中的地址。
3. 按照上一步在 W\_addr\_idx中读取的地址，从DDR中取出数据，并进行人脸检测。
4. 当D\_TreeAccels完成一张照片的人脸检测之后，且Irq\_ready=1时，向Controller发送中断请求Irq，告知Controller执行完毕。
5. Controller收到中断请求之后，马上协调另一幅图存入DDR，当完成该工作之后，就把给D\_TreeAccels的Irq\_done信号置1，控制D\_TreeAccels对第二张图像进行计算。如此循环往复。

### 4.2.5.2 D\_TreeAccels接口设计

表五 D\_TreeAccels模块输入输出信号定义表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名称** | **方向** | **宽度** | **信号描述** |
| Inst | Input | 1 | 来自Controller的模块启动信号 |
| Read\_ready | Input | 1 | 来自Controller的地址是否可读判断信号 |
| W\_addr\_idx | Input | 32 | 来自Controller的地址信息 |
| Irq\_ready | Input | 1 | 来自Controller的是否可以接受请求信号 |
| Irq | output | 1 | 来自D\_TreeAccels的运算完成信息 |
| Irq\_done | Input | 1 | 来自Controller的下一帧图像是否准备好信号 |
| AXI | Inout | 32 | 与AXI INTERCONNECT 的交互信息 |

### 4.2.5.3 D\_TreeAccels逻辑设计

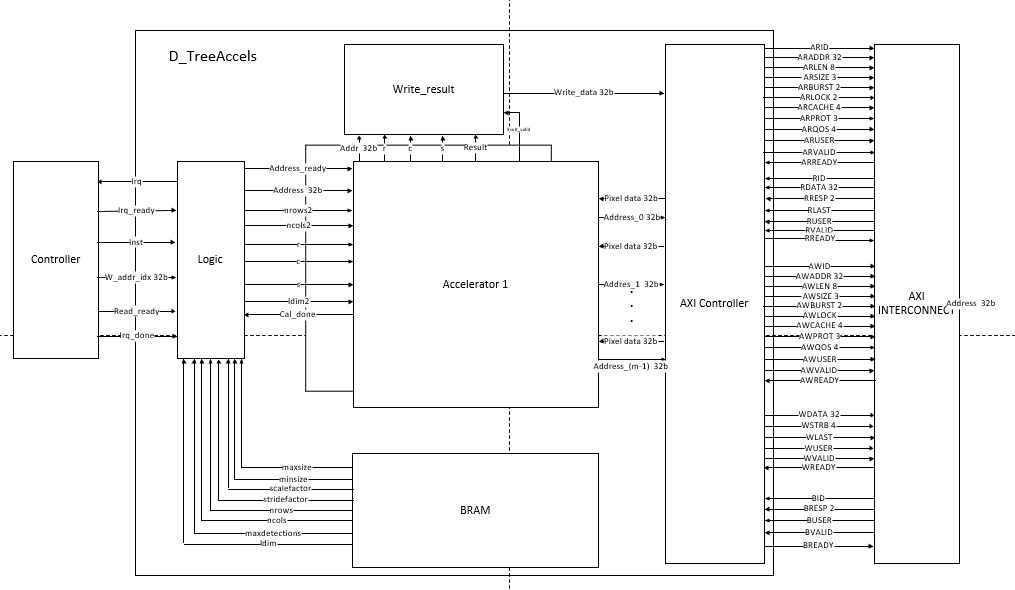


图19 D\_TreeAccels模块的内部逻辑图

D\_TreeAccels模块内部的设计如图19所示。其中Logic和Accelerator两个模块共同完成pico算法的内容，BRAM用来存储一些初始化的信息，Write\_result用来给检测出来的人脸画框，AXI Controller负责与AXI总线进行交互，获取DDR中的数据。具体逻辑如下：

对于每一张要检测的图像，Logic运行pico算法中的移窗操作。而对于每一个窗口，pico需要在其中检测是否存在人脸，这个检测是基于训练好的决策树来做的，而这部分工作由Accelerator来运行。当Accelerator算出当前图像中的人脸坐标信息之后，会将这些信息送给Write\_result模块，由该模块根据坐标信息修改DDR中的数据，为人脸“画框”。

同时，为了提高系统的并行度，我们采用双加速器的设计，使运算速度更快。

#### 4.2.5.1.1 Logic模块详细设计

##### 4.2.5.1.1.1Logic模块总体框架

Logic子模块主要是与Controller进行交互，并完成pico人脸检测算法的移窗部分。其外部连接如图20所示。

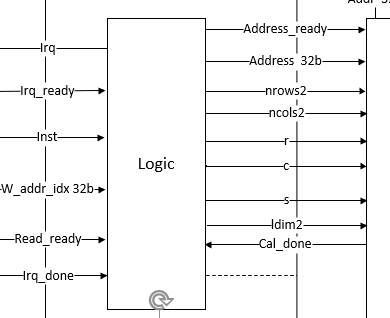


图20 Logic模块外部连接

##### 4.2.5.1.1.2Logic模块接口设计

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名称** | **方向** | **宽度** | **信号描述** |
| Inst | Input | 1 | 来自Controller的模块启动信号 |
| Read\_ready | Input | 1 | 来自Controller的地址是否可读判断信号 |
| W\_addr\_idx | Input | 32 | 来自Controller的地址信息 |
| Irq\_ready | Input | 1 | 来自Controller的是否可以接受请求信号 |
| Irq | output | 1 | 来自D\_TreeAccels的运算完成信息 |
| Irq\_done | Input | 1 | 来自Controller的下一帧图像是否准备好信号 |
| Address\_ready | Output | 1 | 来自Logic的告诉Accelerator地址有效的信号 |
| Address | Output | 32 | 来自Logic的告诉Accelerator的地址 |
| nrows2 | Output | 32 | 来自Logic的参数 |
| ncols2 | Output | 32 | 来自Logic的参数 |
| r | Output | 32 | 来自Logic的参数 |
| c | Output | 32 | 来自Logic的参数 |
| s | Output | 32 | 来自Logic的参数 |
| ldim2 | Output | 32 | 来自Logic的参数 |
| Cal\_done | Input | 1 | 来自Accelerator的运算完成信号 |

##### 4.2.5.1.1.3Logic模块逻辑设计

Logic模块主要完成的就是与Controller的交互和pico算法的移窗部分，其状态机如图21所：

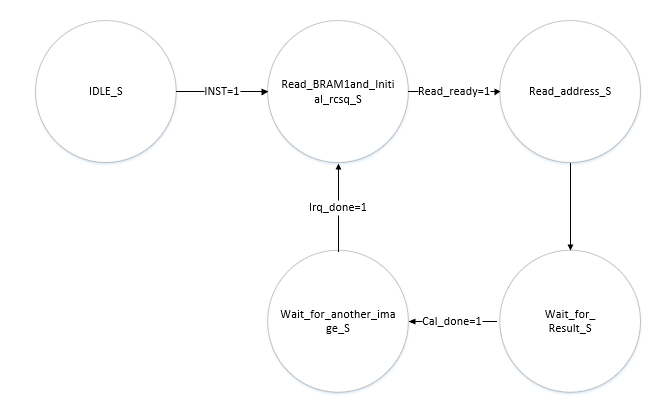


图21 Logic模块状态机

IDLE\_S；空状态，当Controller将Inst信号置为1的时候，状态跳转为Read\_BRAM1\_and\_initial\_rcsq\_S；

Read\_BRAM1\_and\_initial\_rcsq\_S：从BRAM1里面读取相关的配置信息，并且对处理本张图像的信息进行初始化，当Controller将Read\_ready信号置为1之后，状态跳转为Read\_address\_S；

Read\_address\_S：Logic从Controller读取图像在DDR中的地址，随后状态机跳转为Wait\_for\_Result\_S；

Wait\_for\_Result\_S：将图像在DDR中的地址以及所需要的各种参数传给Accelerator，等待Accelerator的执行结果；当Accelerator将Cal\_done信号置为1时，状态机跳转为Wait\_for\_another\_image\_S；

Wait\_for\_another\_image\_S：本张图像处理完毕，向Controller发送中断请求下一张图的信息；当Controller将Irq\_done信号置为1时，状态机跳转为Read\_BRAM1\_and\_initial\_rcsq\_S，对下一张图像进行处理。

#### 4.2.5.1.2Accelerator模块详细设计

##### 4.2.5.1.2.1Accelerator模块整体框架

Accelerator模块主要是用来对Logic确定的移窗进行决策树的逻辑运算，从而判断出该移窗中是否有人脸。其内部具体的设计如图22所示。

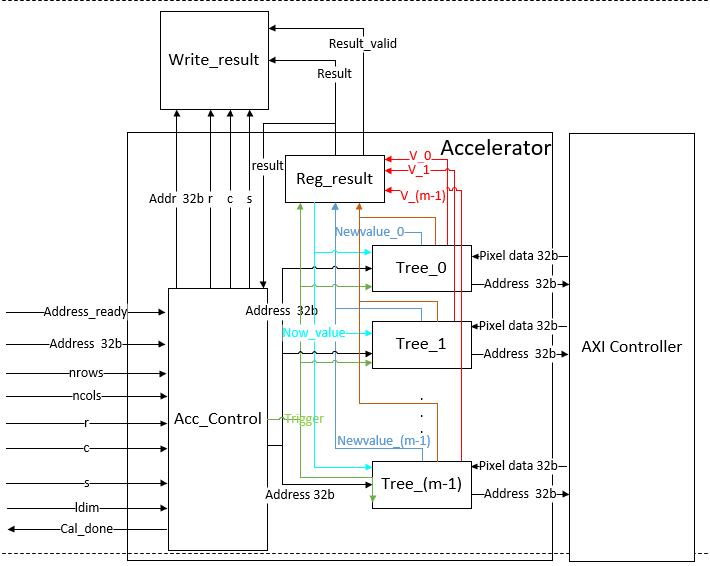


图22 Accelerator模块内部设计

##### 4.2.5.1.2.2Accelerator模块接口设计

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名称** | **方向** | **宽度** | **信号描述** |
| Address | Input | 32 | 来自Logic的告诉Accelerator的地址 |
| nrows2 | Input | 32 | 来自Logic的参数 |
| ncols2 | Input | 32 | 来自Logic的参数 |
| r | Input | 32 | 来自Logic的参数 |
| c | Input | 32 | 来自Logic的参数 |
| s | Input | 32 | 来自Logic的参数 |
| ldim2 | Input | 32 | 来自Logic的参数 |
| Cal\_done | Output | 1 | 来自Accelerator的运算完成信号 |
| Addr | Output | 32 | 来自Accelerator的地址信息 |
| R | Output | 32 | 来自Accelerator的图像坐标信息 |
| C | Output | 32 | 来自Accelerator的图像坐标信息 |
| S | Output | 32 | 来自Accelerator的图像大小信息 |
| Pixel data | Input | 32 | 来自AXI Controller的图像信息 |
| Address\_0 | Ouput | 32 | 来自Accelerator的图像地址信息 |

由于在Accelerator中实现了并行操作，所以会有多对Pixel data和Address\_n信号，在表中没有一一列举

##### 4.2.5.1.2.3 Accelerator模块逻辑设计

###### 4.2.5.1.2.3.1 Acc\_Control子模块详细设计

Acc\_Control子模块负责接收从Logic模块传输过来的信号，并初始化各个参数，读取图像在DDR中的地址，以及进行是否越界的检查；Tree\_0至Tree\_(m-1)则是并行的模块，可以同时计算通过m棵树后该窗口所得分数，提高并行度，节省时间。

Acc\_Control的状态机设计如图23：

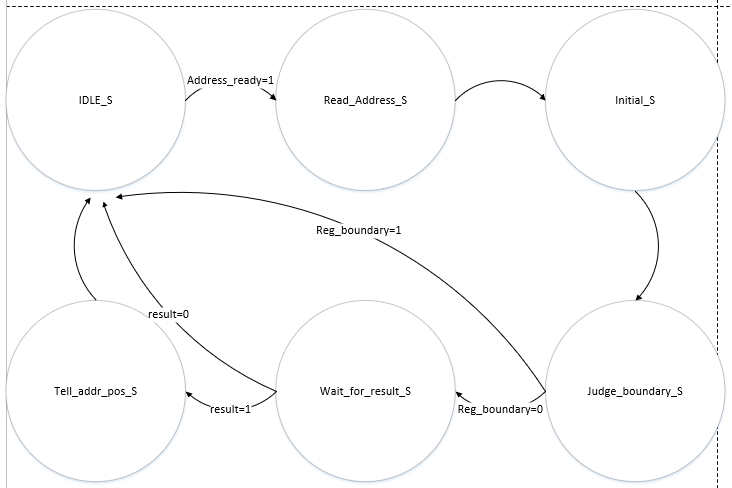


图23 Acc\_Control模块状态机

IDLE\_S：当Logic给的Address\_ready=1时，状态机跳转为Read\_Address\_S

Read\_Address\_S：读取图像的地址，随后状态机跳转为Initial\_S状态

Initial\_S：进行参数的初始化，随后跳转为Judge\_boundary\_S；

Judge\_boundary\_S：实现pico算法中的边界检查功能，如果Reg\_boundary=1，说明超界，则该窗口的图像处理无效，状态机跳转为IDLE\_S状态；若Reg\_boundary=0，说明未超界，则状态机跳转为Wait\_for\_result\_S；

Wait\_for\_result\_S：Acc\_Control触发Tree工作；若result=0，说明没有检测到人脸，状态机跳转回IDLE\_S；result=1，说明检测到人脸，状态机则跳转为Tell\_addr\_pos\_S；

Tell\_addr\_pos\_S：Acc\_Control给Write\_result提供图像的地址和坐标信息，方便使Write\_result模块画框，随后状态机跳转为IDLE\_S。

###### 4.2.5.1.2.3.2 Tree子模块详细设计

Tree模块的功能是运算pico算法中一棵决策树的内容。具体分为两块工作，一是要确定本次执行的哪棵树的运算，从本模块配属的BRAM中读取相关的数据；二是要根据图像的地址在DDR中读取图像的像素数据，从而进行运算。其内部结构如下：

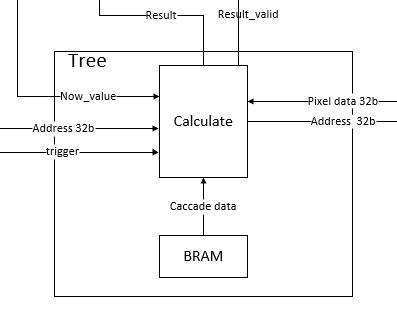


图24 Tree模块内部图

其中Calculate模块是进行运算的部分，BRAM中存储着其要用到的决策树信息。Calculate模块的状态机如下：

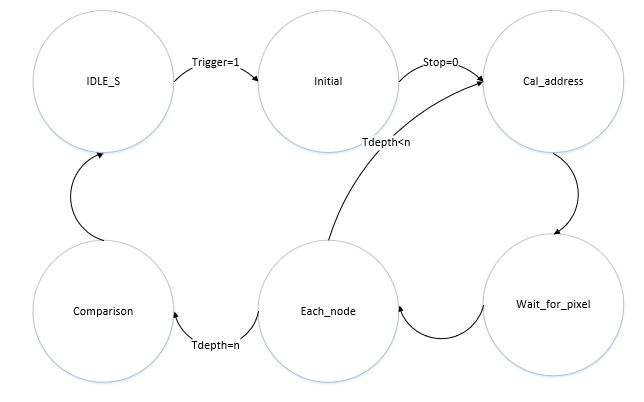


图25 Calculate模块状态机

IDLE\_S：Trigger=1时，状态机跳转位Initial状态，

Initial：进行参数的初始化，将常用数值放到寄存器中，计算本轮所需要的决策树信息的地址；当stop信号为0时，状态跳转为Cal\_address；

Cal\_address：进行图像地址的计算；随后状态机跳转为Wait\_for\_pixel；

Wait\_for\_pixel：与DDR进行交互，读取DDR中的图像信息；而后状态机跳转为Each\_node；

Each\_node：进行每个节点的判断；如果此时决策树还没有到叶子节点，那么状态机跳转为Cal\_address，进行下一轮的判断。若已经到达叶子节点，将叶子节点的值记入寄存器，状态机跳转为comparison;

Comparison：接收Reg\_write给出的value值，与在本棵树中的得分相加，相加之后的值为new\_value，而后将new\_value与本棵树的阈值进行比较，如果小于阈值，则将result置为0，表示未检测到人脸；如果大于阈值，表示检测到人脸。随后状态机跳转到IDLE\_S。

###### 4.2.5.1.2.3.3 Reg\_result子模块详细设计

Reg\_result模块主要用来记录该移窗图像的得分情况。该模块中会有一个寄存器，用来记录通过树的个数。

Reg\_write的状态机如图26所示。

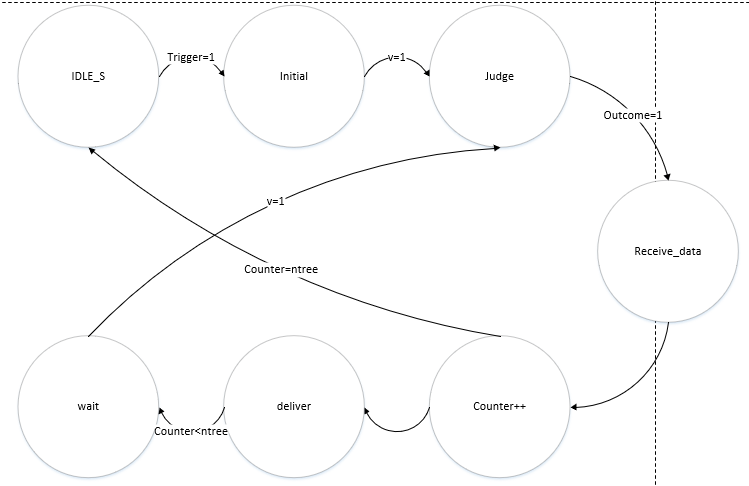


图26 Reg\_write状态机

IDLE\_S：起始状态，当Trigger=1时，状态机跳转为Initial；

Initial：初始化状态，将counter设置为0，将now\_value设置为0；当v=1时，条状为Judge状态；

Judge：检查outcome信号，如果outcome=1，状态跳转为Receive\_data；

Receive\_data：接受来自new\_value的数据，状态跳转为Counter++;

Counter++：将计数器加一；如果Counter=ntree，说明已经到达叶子节点，可以直接判定是否通过了该决策树，状态跳转为IDLE\_S；否则，状态跳转为deliver；

Deliver：将now\_value赋值为new\_value，发送给各个tree模块；若counter<ntree，状态条状为wait；

Wait：等待下一个v=1，此时状态机跳转为Judge。

##### 4.2.5.1.2.2 Write\_result模块详细设计

Write\_result模块实现画框功能，其状态机如图27。

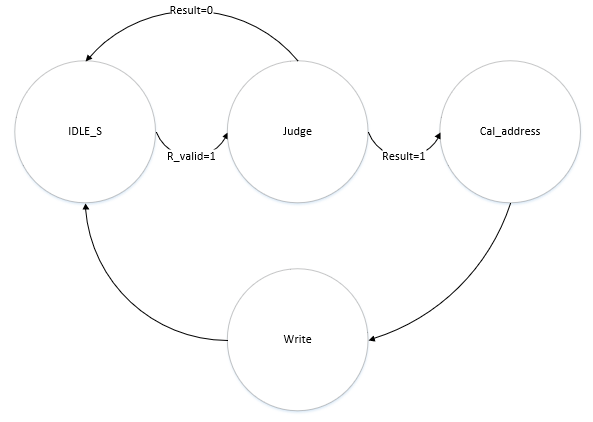


图27 Write\_result模块状态机

IDLE\_S：空状态，当R\_valid=1时，状态跳转为Judge；

Judge：判断是否检测到人脸；当Result=0时，状态跳转为IDLE\_S；当Result=1时，状态跳转为Cal\_address；

Cal\_address：计算地址，状态机跳转为Write；

Write：通过总线与DDR交互，画框，状态跳转为IDLE\_S。

* + - * 1. AXI Controller模块详细设计

AXI Controller模块主要负责整个系统与DDR的交互。具体的，通过轮询的方式进行，逻辑较为简单，但要遵循AXI协议。

### FPGA I/O卡详细设计

该模块将从缓存中读出的像素点数据包装成UDP格式的报文进行传输，需要将以太网头、IP头、UDP头、消息类型、数据帧索引等头信息，然后再通过以太网进行发送。同时将CPU发出的包含摄像头控制信息的UDP报文解析出来，并将控制信息传输给摄像头。

### Eth- Converter1模块详细设计

### Eth- Converter1整体框架



图28 Eth-Converter1模块总体框图

### Eth- Converter1模块接口

Eth- Converter1模块主要由两个子模块组成，分别是DSM模块和DRM模块其接口定义分别如下表所示。

表五 Eth-Converter1模块输入输出信号定义表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名称** | **方向** | **宽度** | **信号描述** |
| clk | input | 1 | 时钟信号 |
| rst | input | 1 | 复位信号，高有效 |
| Y2…Y9 | Input | 8 | 摄像头像素信息，每拍传输八位。 |
| Vsync | Input | 1 | 摄像头发送的帧同步信号，表示一张图像完成了传输。 |
| lut\_index | Input | 10 | 命令的类型。 |
| D2…D9 | Output | 8 | 通过网络发送的图像数据信号。 |
| Dvalid | Output | 1 | 输出数据有效信号 |
| R2…R9 | Input | 8 | 通过网络接收的UDP数据 |
| Rvalid | Input | 1 | 输入数据有效信号 |
| Req | Input | 1 | MAC ready信号 |

### DSM模块接口

表六 DSM子模块输入输出信号定义表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名称** | **方向** | **宽度** | **信号描述** |
| clk | input | 1 | 时钟信号 |
| rst | input | 1 | 复位信号，高有效 |
| Y2…Y9 | Input | 8 | 摄像头像素信息，每拍传输八位。 |
| Vsync | Input | 1 | 摄像头发送的帧同步信号，表示一张图像完成了传输。 |
| D2…D9 | Output | 8 | 通过网络发送的图像数据信号。 |
| Dvalid | Output | 1 | 输出数据有效信号 |
| Req | Input | 1 | MAC ready信号 |
| Req | Input | 1 | DRM确认能够发送的信号。 |

### DRM模块接口

表七DRM子模块输入输出信号定义表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名称** | **方向** | **宽度** | **信号描述** |
| clk | input | 1 | 时钟信号 |
| rst | input | 1 | 复位信号，高有效 |
| R2…R9 | Input | 8 | 通过网络接收的UDP数据 |
| Rvalid | Input | 1 | 输入数据有效信号 |
| lut\_index | Output | 10 | 命令的类型。 |
| Req | Output | 1 | DRM根据请求确认能够发送的信号。 |

### Eth- Converter1模块设计

Eth- Converter1模块一方面通过摄像头的图像数据包装成UDP格式发送给加速卡，另一方面完成摄像头配置信息报文的接收与分析。

UDP协议报文格式主要如下：



图29 UDP协议报文格式

在第一个方面，即将图像数据包装成UDP报文格式方面，首先对于固定的信息，IP首部、UDP首部和Index信息，将这些信息保存在固定的寄存器中，将图像像素信息每拍八位保存在缓存之中，当缓存存满后通过仲裁及MAC发送给加速卡，同时对index信息加一，当加到600完成传输后，自动复位至0，因为每一帧图像的完整传输需要600个数据报文，刚好能够传输完成，同时为了保证每一帧图像的正确完整传输，将摄像头的帧同步信号接入该模块，当每一帧图像传输完成后，会通过该信号告知该模块，该模块也会将index信息重置，

在另一个方面，其通过MAC接收后通过八位的数据通路传输下来，通过仲裁器后传输至Eth-Converter模块，进入该模块后是已经自动去除报文头部信息后的数据信息，将该信息保存在缓存中，然后根据数据信息发送信号给CamControl模块，CamControl模块内存储了与具体控制相关的命令，其接收到相应的信息后完成相应的动作，完成后CamControl模块会发送完成的信息给Eth- Converter1模块，然后再发送下一条指令的相关信息，直至全部的配置动作完成。

整个过程将分为两个子模块完成，一个处理发送信息，名为DSM（data sending module）模块，一个处理接收信息，名为DRM（data receiving module）模块。

DSM模块：DSM模块主要负责发送数据的整合，将摄像头每次八位的数据整合成为UDP报文所需的1024B的大小，然后一起发送，因此在这里采用双FIFO缓冲的设计，FIFO的大小设置为1024B，位宽为8bit。例如每次摄像头的数据都进入FIFO1，当FIFO1满后，摄像头来的数据开始进入另一个FIFO2，FIFO1的数据开始通过仲裁器后进行发送，这里需要注意的是，在发送数据之前还需要将报文的头部信息先发送出去。每次发送完一个FIFO后，index数据加一，发送完成后等待下一个FIFO的数据满。其状态转移图如下所示



图30 DSM子模块状态转移图

其主要设计结构如下图所示



图31 DSM子模块设计框图

REQ\_S状态查看总线状态，检查是否满足发送条件，SENDH\_S状态发送头部信息，SENDD\_S状态发送数据信息，SENDE\_S为结束状态，主要完成index的更新工作。在控制模块中，其发送请求主要控制实现的目的有两个，其一是实现双FIFO缓冲的功能，其二是实现加速卡对于图像的请求控制功能，即加速卡发出发送图像的请求后，I/O卡要在接收到消息后发送下一帧完整的图像数据，其余的图像数据都予以忽略，不予发送。

DRM模块主要将CPU发出的包含摄像头控制信息的UDP数据段报文解析出来，然后再将相应的信息发送给命令执行部件CamControl进行完成。同时，对于加速卡提出的发送新一帧图像的请求信息，会与DSM模块进行通信，倘若这是新一帧的图像，且FIFO满了，则统一DSM模块的发送请求，倘若没有加速卡的发送请求信息，则拒绝DSM模块的发送请求。其主要的状态转移图如下所示：



图32 DRM子模块状态转移图

### Arbiter模块详细设计

Arbiter模块完成UDP报文的接收，报文头的分析，以及分发的功能，同时完成来自DSM模块数据的UDP报文发送任务。主要的功能非为两个部分，其一为发送UDP报文的功能，其二为接收UDP的报文并分发给Eth-Converter1和Eth-Converter2两个模块的功能，因此在顶层设计中，主要设计两个子模块完成相应的功能。

### Arbiter整体框架



图33 Arbiter模块总体结构框图

### Arbiter模块接口

表八 Arbiter 模块输入输出信号定义表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名称** | **方向** | **宽度** | **信号描述** |
| clk | input | 1 | 时钟信号 |
| rst | input | 1 | 复位信号，高有效 |
| D2…D9 | Input | 8 | 通过网络发送的图像数据信号。 |
| Dvalid | Intput | 1 | 输出数据有效信号 |
| Req | Output | 1 | MAC ready信号 |
| Tx\_ready | Input | 1 | MAC ready信号，表明现在可以发送。 |
| Tx\_mac | Output | 8 | 发送数据。 |
| Tx\_valid | Output | 1 | 发送数据有效信号。 |
| Rx\_mac | Input | 8 | 接收数据。 |
| Rx\_valid | Input | 1 | 接收数据有效信号。 |
| R2…R9 | Output | 8 | 发送给Eth-converter1的信号。 |
| Rvalid | Output | 1 | 输出数据有效信号 |
| C2…C9 | Output | 8 | 发送给Eth-converter2的信号。 |
| Cvalid | Output | 1 | 输入数据有效信号 |

### USM接口设计

表九 USM子模块输入输出信号定义表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名称** | **方向** | **宽度** | **信号描述** |
| clk | input | 1 | 时钟信号 |
| rst | input | 1 | 复位信号，高有效 |
| D2…D9 | Input | 8 | 通过网络发送的图像数据信号。 |
| Dvalid | Intput | 1 | 输出数据有效信号 |
| Req | Output | 1 | MAC ready信号 |
| Tx\_ready | Input | 1 | MAC ready信号，表明现在可以发送。 |
| Tx\_mac | Output | 8 | 发送数据。 |
| Tx\_valid | Output | 1 | 发送数据有效信号。 |

### URM接口设计

表十 URM子模块输入输出信号定义表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名称** | **方向** | **宽度** | **信号描述** |
| clk | input | 1 | 时钟信号 |
| rst | input | 1 | 复位信号，高有效 |
| Rx\_mac | Input | 8 | 接收数据。 |
| Rx\_valid | Input | 1 | 接收数据有效信号。 |
| R2…R9 | Output | 8 | 发送给Eth-converter1的信号。 |
| Rvalid | Output | 1 | 输出数据有效信号 |
| C2…C9 | Output | 8 | 发送给Eth-converter2的信号。 |
| Cvalid | Output | 1 | 输入数据有效信号 |

### Arbiter模块逻辑设计

对于USM(udp send module)模块，其完成UDP报文的发送任务，同时检测MAC模块的状态，观察其是否可用，并将信息反馈给下面的DSM模块，因为DSM模块在数据准备好后也需要检查MAC是否准备好接收数据，因此对于这一模块的数据，基本是直通的状态。



图34 USM子模块示意图

对于URM(UDP Receive Module)模块，其完成UDP报文的接收工作和UDP报文头的解析和UDP保温数据的分发工作。每当有新的UDP报文来的时候，其接收报文数据并根据报文头信息将后续的数据传输给相应的端口。其状态转移图如下所示：



图35 URM子模块状态转移图

根据以上的分析内容，具体将模块设计如下：



图36 URM子模块结构示意图

首先根据State状态对数据进行分发，倘若State在IDLE\_S或者HEADR\_S状态，则不分发头部信息，头部信息由State模块进行分析和处理，当状态跳转为TO1\_S或者TO2\_S时，则将后续相应的数据流直接传输给相应的Eth\_Converter1模块或者Eth\_Converter2模块。这里需要注意的是index部分的信息是保留在数据部分内发送给Eth\_Converter模块进行下一步处理的，而不是在这里进行截断处理。

### Eth\_Converter2详细设计

### Eth\_Converter2整体框架

Eth\_Converter2模块主要完成加速卡完成识别后的图像数据的接收和数据的转换工作，最后输出像素点的RGB888格式。模块框图如图所示：



图37 Eth\_Converter2模块结构框图

### Eth\_Converter2模块接口

表十一 Eth\_Converter2模块输入输出信号定义表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名称** | **方向** | **宽度** | **信号描述** |
| clk | input | 1 | 时钟信号 |
| rst | input | 1 | 复位信号，高有效 |
| C2…C9 | Input | 8 | 像素数据 |
| Cvalid | Output | 1 | 像素数据有效信号 |
| Vsync | Output | 1 | 场同步信号 |
| Hsync | Output | 1 | 行同步信号 |
| P0…P15 | Output | 16 | 像素数据 |

### Eth\_Converter2模块逻辑设计

由以上的内容可知，Eth\_Converter2模块主要完成两个功能，其一是接收UDP数据段，其二是将每一个像素点从RGB565格式转换成RGB888格式，同时根据index以及发送的像素信息生成行场同步信号，并输出给pixel buffer模块。由于格式转换的过程就是将数据通过补零的方式转成固定8位的数据，因此此处仍然保留16位的像素数据。当信号进入Eth\_Converter2模块后，首先会被解析头部信息，后续的数据会进入FIFO之中，FIFO的大小为1024B，且宽度为16bits，即一个像素点的数据占用一个FIFO行，当一个数据包的数据接收完毕后，会产生FIFO\_FULL信号，进而触发进入下一级的piexel buffer。同时控制模块也会根据头部的index信息向下一级发送每一个像素点的行同步信号和场同步信号。模块的状态转移图如下图所示：



图38 Eth\_Converter2模块状态转移图

在FRAME0\_S状态，其接收的是一帧图像的第一个UDP报文数据，这个状态时需要对模块内的一些状态进行初始化，以便完成一帧完整图像的处理过程。SEND0\_S状态是向下发送第一个报文数据，同时在发送数据前还需要对于行、场同步信号进行发送，以使得显示器的初始点在左上角，然后再进行向下发送和显示。SEND1\_S状态即完成后续数据的处理，主要注意后续根据发送的像素点的数量完成行同步信号的生成，直至完成所有一帧图像的显示。

### HDMI converter详细设计

复用模块

将像素点信息转换驱动HDMI显示。

### CamControl模块

复用模块

CamControl模块是为了解析发送给摄像头的配置信息报文，并发送相应的配置信号给摄像头的一个模块，对于摄像头的一方，其通过I2C接口完成配置。