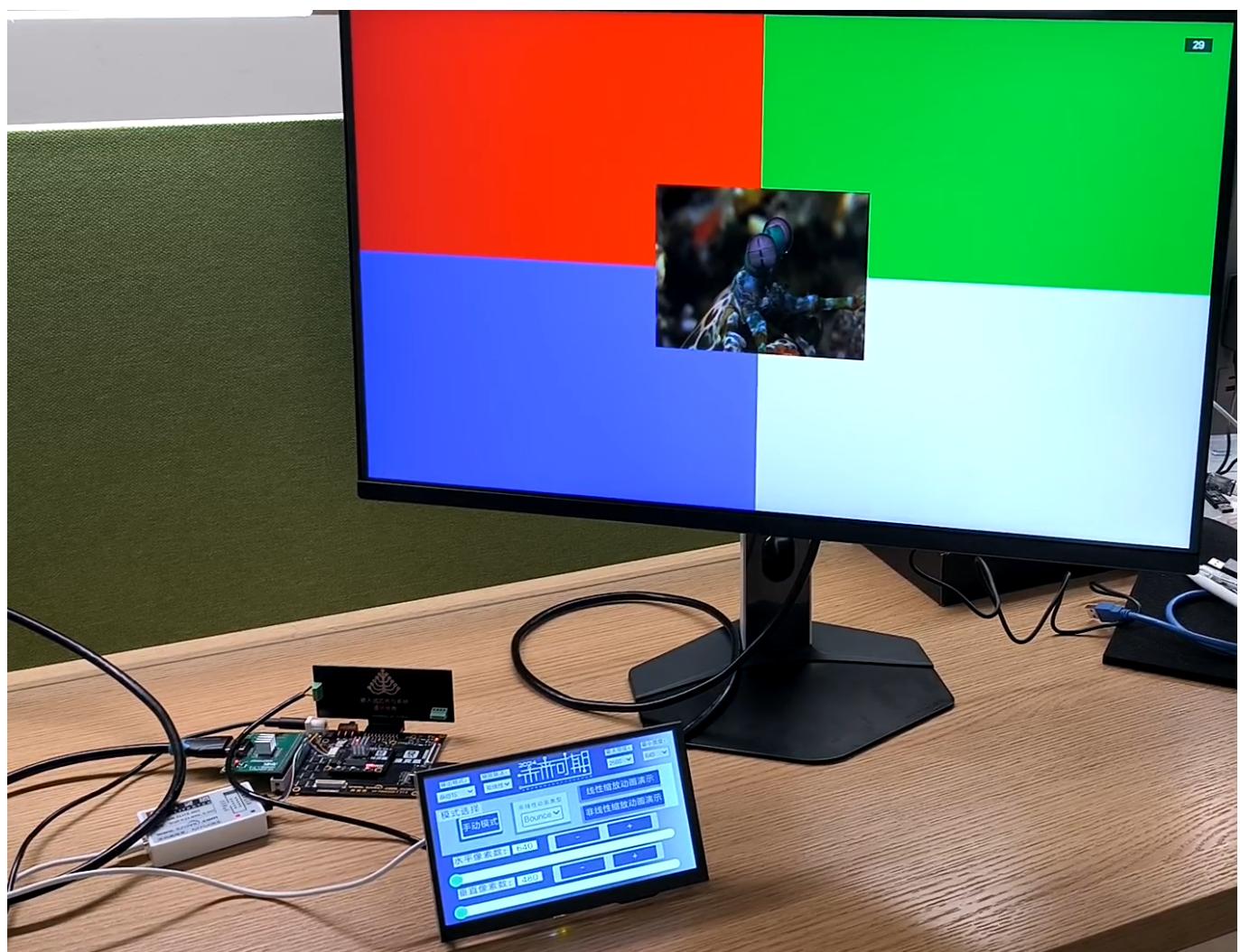




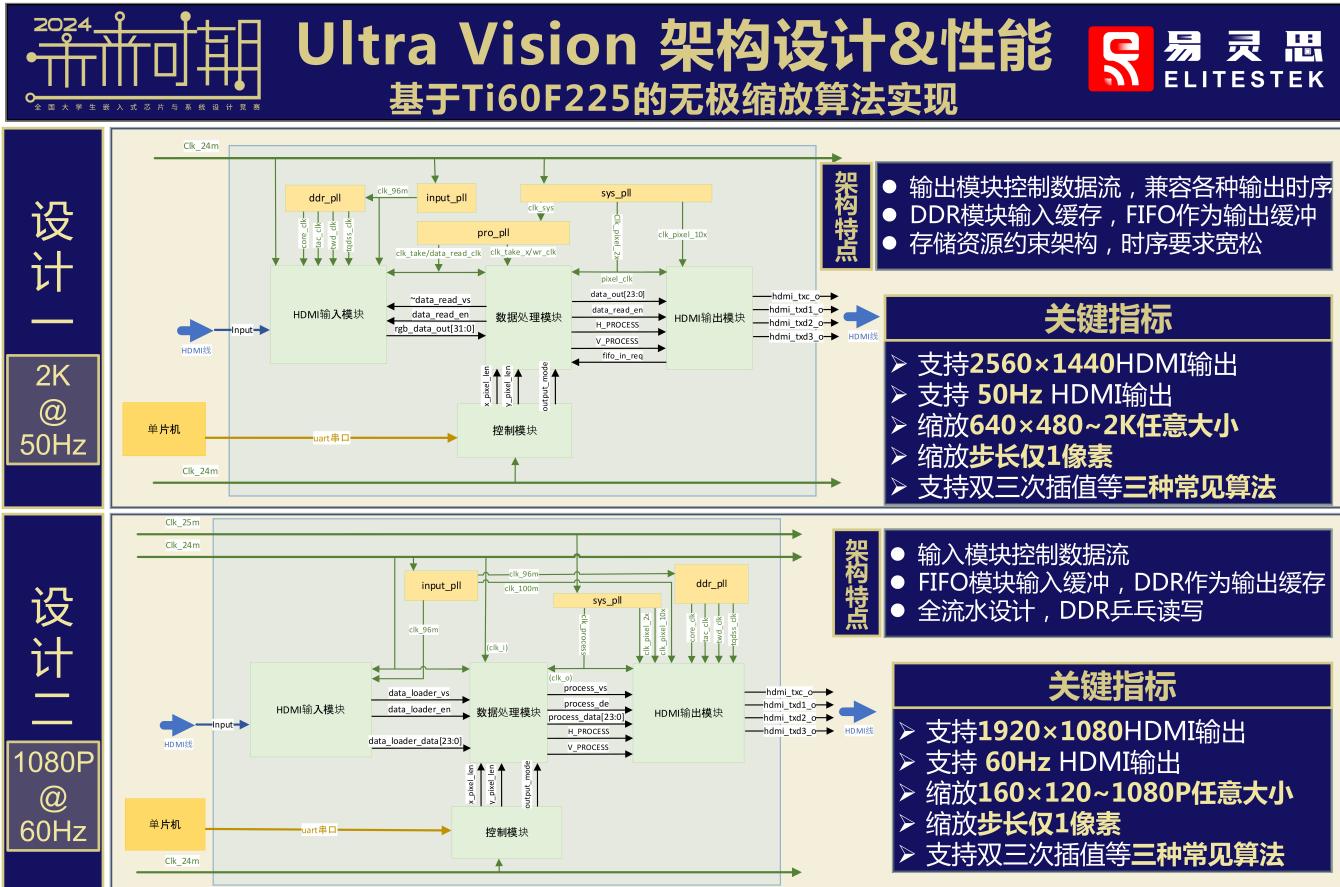
路演演示视频：【FPGA创新设计竞赛】2024年国一+易灵思企业杯获奖作品——基于Ti60F225的无极缩放算法实现

## 作品实物图



## Ultra-Vision 作品海报

架构设计与性能：



作品亮点：

**2024 可期** 全国大学生嵌入式芯片与系统设计竞赛

# Ultra Vision 作品亮点

## 基于Ti60F225的无极缩放算法实现

**易灵思 ELITESTEK**

**作品亮点**

- 支持HDMI视频输入输出
- 最高2K分辨率输出
- 最高60Hz刷新率输出
- 单片机作为上位机控制缩放
- 通过图形界面人机交互
- 内置多种非线性缩放演示动画
- 双三次插值算法参数实时调整
- 三种插值算法实时切换
- DSP资源利用率100%

**内置算法对比**

双三次 | 双线性 | 最近邻

**图形界面**

- 控制方式多样、友好、准确：按键、滑块、预设动画
- 深入算法，可调参数丰富：双三算法系数可调
- 四种预设缩放演示动画：线性、Bounce、Circ、Quint
- 界面精美流畅：丰富控件、大量使用过渡动画
- 高兼容性：兼容多输出格式，两个设计，一屏控制

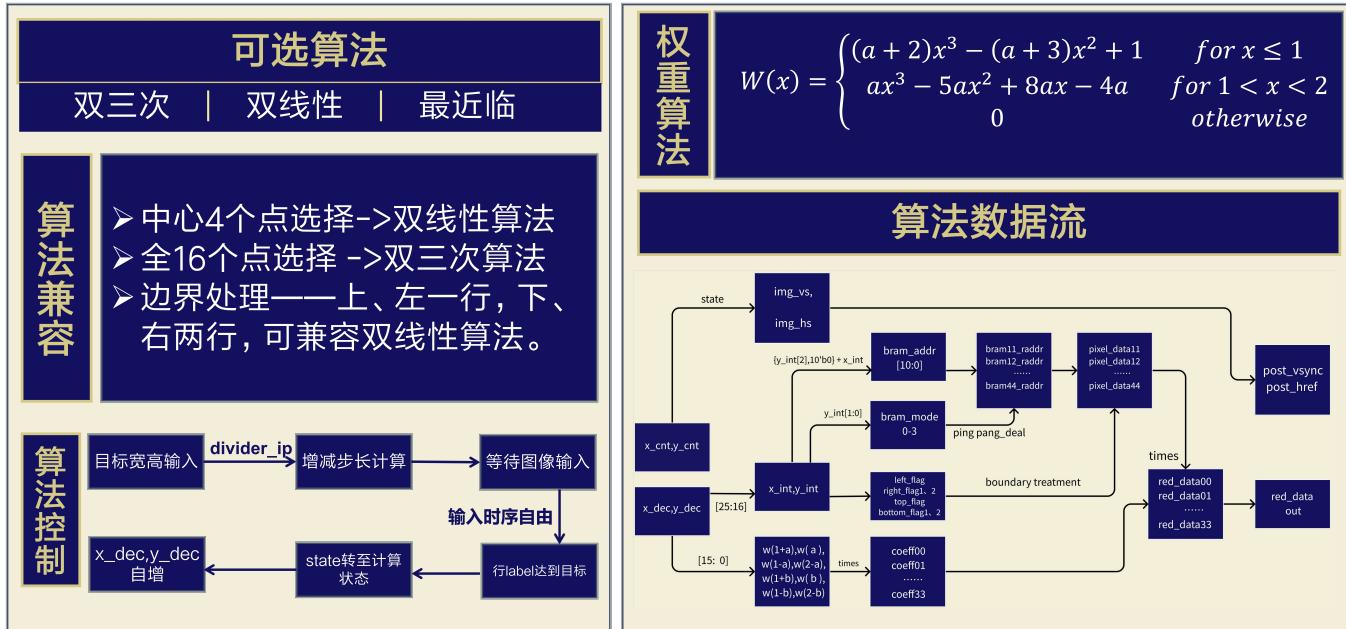
模式选择

手动模式	非线性动画类型	线性缩放动画演示
Bounce	非线性缩放动画演示	

水平像素数： 640 - +

垂直像素数： 480 - +

算法细节：



# 1. 作品简介

## 1.1 关键性能指标

**视频输入输出格式:**

输入格式: 640×480@60Hz

输出格式: 1080P@60Hz \ 2K@50Hz (分别使用两种架构)

**无极缩放步长:**

1像素 (宽和高的缩放步长均为1像素, 可以分别调整)

**无极缩放范围:**

1080P@60Hz 格式下: 160×120~1920×1080中的任意宽高组合

2K@50Hz 格式下: 640×480~2560×1440中的任意宽高组合

**系统输入输出延时:**

1080P@60Hz 格式下: 16.7ms

2K@50Hz 格式下: 可忽略不计

**内置缩放动画展示:**

线性动画: 线性动画

非线性动画: Bounce、Circ、Quint

## 1.2 主要创新点

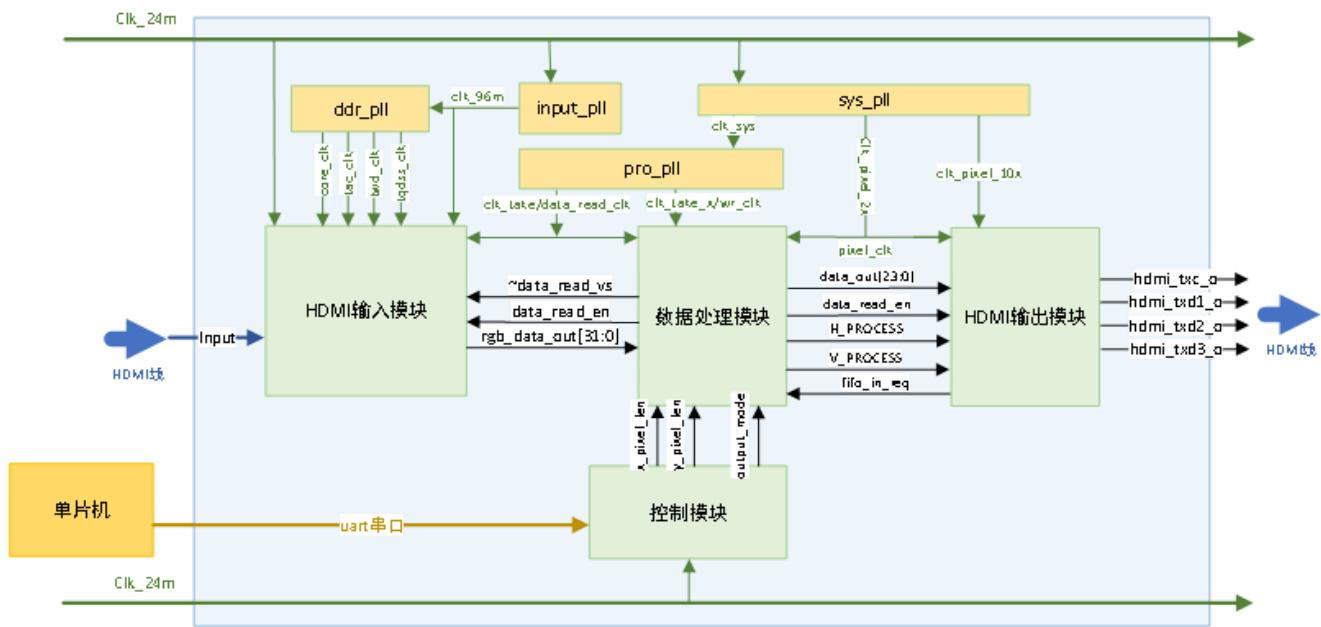
1. 输出格式为2K@50Hz的系统实现了算法直接输出给HDMI输出模块进行显示的架构，无需回写至DDR再输出显示，可以获得更高的输出带宽，从而支持更大的缩放范围的同时取得更高的输出帧率即640×480~2560×1440@50Hz。若没有采用这一架构，则输入模块读写DDR的操作会与算法输出回写DDR的操作冲突，造成系统流水中断，影响输出帧率。
2. 系统使用一套缩放算法同时实现了双三次插值、双线性插值和最近邻插值，可以在人机交互界面随时切换这三种缩放算法，并可以改变算法所采用的权重参数。
3. 使用LVGL图形库和STM32单片机构建人机交互界面，实现缩放大小的控制与显示，并内置预设缩放动画，以供作品展示，缩放动画流畅无卡帧坏帧。

## 2. 系统组成及功能说明

由于本项目是通过两个不同的架构分别实现的2K@50Hz和1080P@60Hz两种不同格式的视频输出，以下将把以2K@50Hz为输出的架构称为设计1，把以1080P@60Hz为输出的架构称为设计2。设计1和设计2之间的控制模块和插值算法模块保持一致，不会重复介绍。下面将对设计1和设计2分别进行详细的介绍。

### 2.1 设计1介绍

#### 2.1.1 设计1整体介绍



##### 2.1.1.1 控制模块

为了给本系统提供更多元的控制方式、更友好的人机交互方式，这里选择通过串口通信的方式为数据处理模块传输目标放大尺寸和选择使用的缩放算法。控制上位机是STM32H743XIH6单片机，其通过LVGL图形库构建的GUI实现人机交互，可以实时调整和显示缩放后的图形尺寸，并且内置了四个预设缩放动画供现场展示。

##### 2.1.1.2 HDMI输入模块

本系统输入信号采用HDMI信号线传输，信号采取VGA标准。由于购买的FPGA底板上没有HDMI输入接口，我们使用奥唯思官方提供的HDMI输入子卡进行HDMI信号输入与解码。

子卡的主要功能由ADV7611芯片实现，ADV7611为Analog Devices的一款低功耗HDMI接收器，可以将HDMI输入信号实时解码，输出RGB888信号、行信号、场信号、数据使能信号与像素时钟等所需信号。在Efinity中配置好对应引脚后，即可将HDMI信号输入FPGA中。

ADV7611芯片的初始化寄存器数据由I2C总线传输，初始化时，需向其对应寄存器组中写入HDMI传输所需的EDID信息，使电脑（信号源）能够选择VGA\_640×480@60Hz的输入时序。

Ti60F225核心板上有一颗DDR，FPGA通过Axi4总线与其进行数据传输。DDR作为内存，其存储空间极大，且支持地址访问，适合作为输入或输出缓冲区。在本设计中，DDR作为输入缓冲，将输入信号的一整帧RGB信号（640×480×24bit）存储在里面，便于后续读取。

### 2.1.1.3 数据处理模块

这个模块主要实现无极缩放的功能，模块包括数据修饰模块与算法模块。数据修饰模块从DDR中读取RGB数据并还原行、场、使能信号（DDR中存储的是纯像素数据）以向算法模块传递行、列、数据使能信息。算法模块则采取双三次、双线性插值或最近邻插值的方法进行视频图像的缩放并输出。

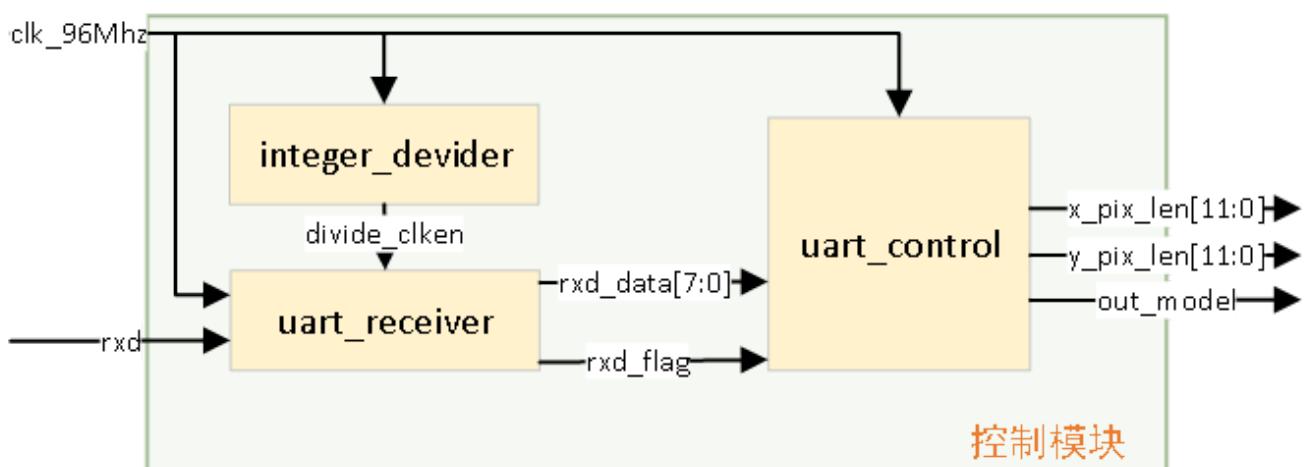
### 2.1.1.4 HDMI输出模块

输出模块包括输出缓冲模块，输出渲染模块以及HDMI输出驱动模块。由于输出在未放大到最大时，输出画面居中显示在显示器中央，这就需要对输出渲染模块对画面未占据的画布进行填充，保证HDMI输出驱动模块在不同放大尺寸下有着稳定的输出时序。

这种情况下，在每一帧中，输出的有效数据（即放大后的输入视频图像）在时间轴上的分布与插值算法输出的有效数据在时间轴上的分布不同，所以在插值算法模块输出与输出渲染模块输入之间需要有一个缓冲模块，这里我们使用异步FIFO来实现。

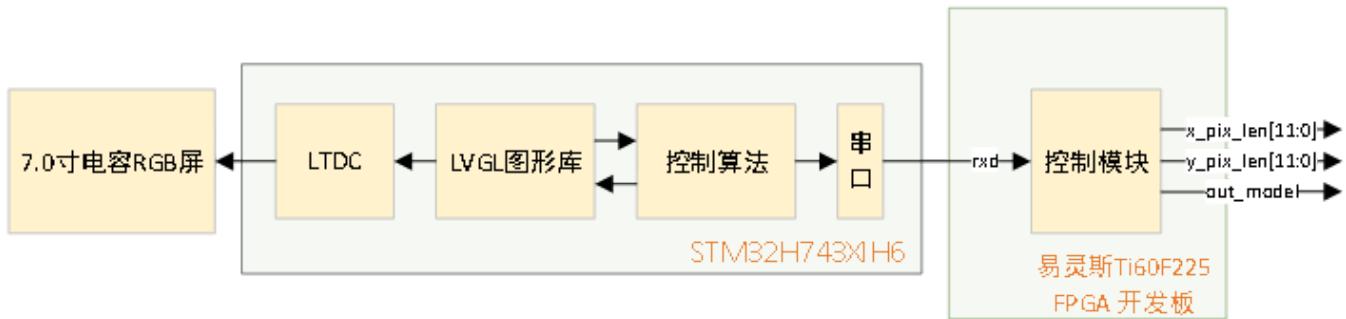
## 2.1.2 设计1各模块介绍

### 2.1.2.1 控制模块（设计1）

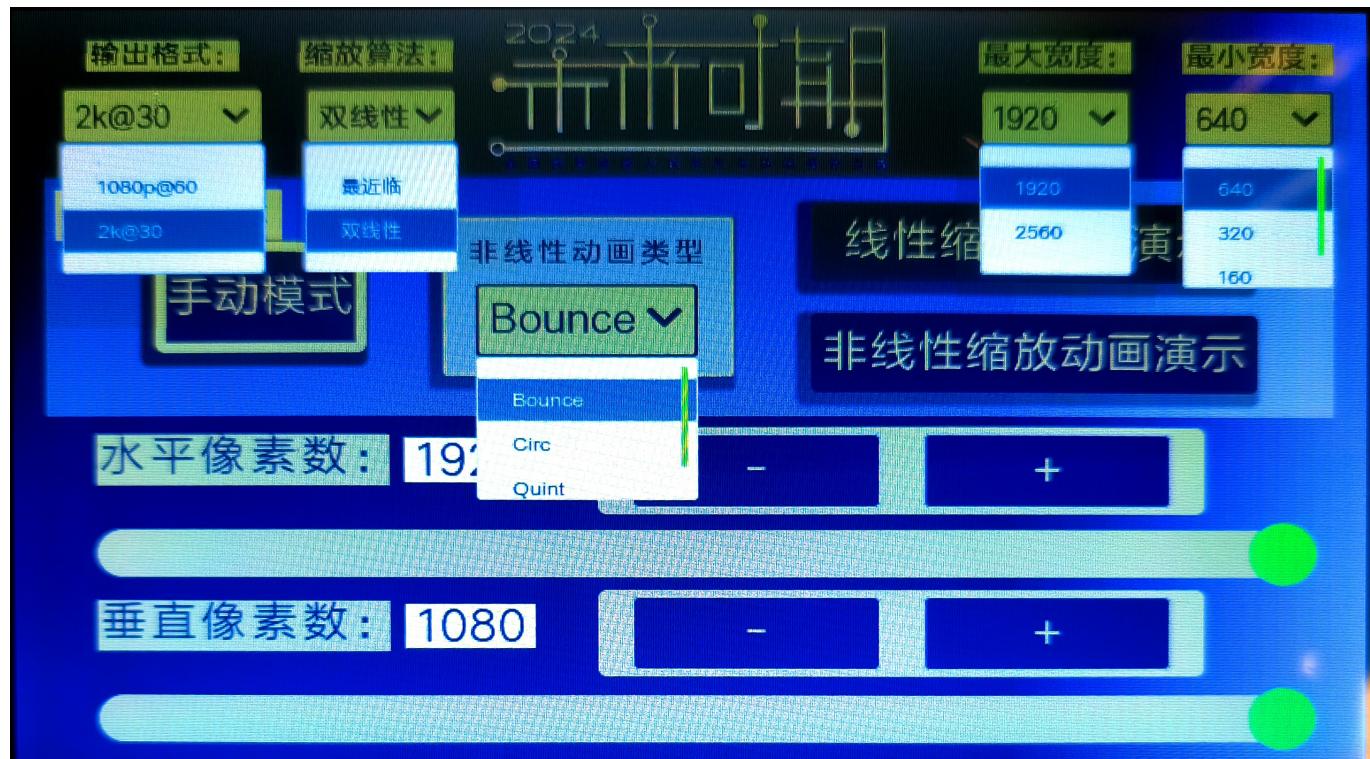


控制模块的结构框图如上图所示，其由integer\_devider、uart\_receiver和uart\_control这三个子模块组成，其中integer\_devider将输入时钟进行分频以匹配串口通信时的波特率；uart\_receiver负责接受串口的串行输入并以字节为单位输出收到的信息；uart\_control可以解码收到的信息，该模块可以从中解码出缩放尺寸中的长和宽（以

像素为单位) 以及缩放所用的算法, 为了加强控制模块的鲁棒性避免通信受到干扰, 这里还为数据包设置了额外的报头校验。



控制模块接受来自上位机的串口通信, 这里使用H743XIH6高性能单片机作为上位机, 负责用户图形界面的渲染和串口发送控制信息, 其中的控制算法负责处理不同控制模式之间的切换和四种不同缩放动画的实时计算, 确保在不同控制模式和缩放动画之间切换时仍保持输出动画的连续和流畅。

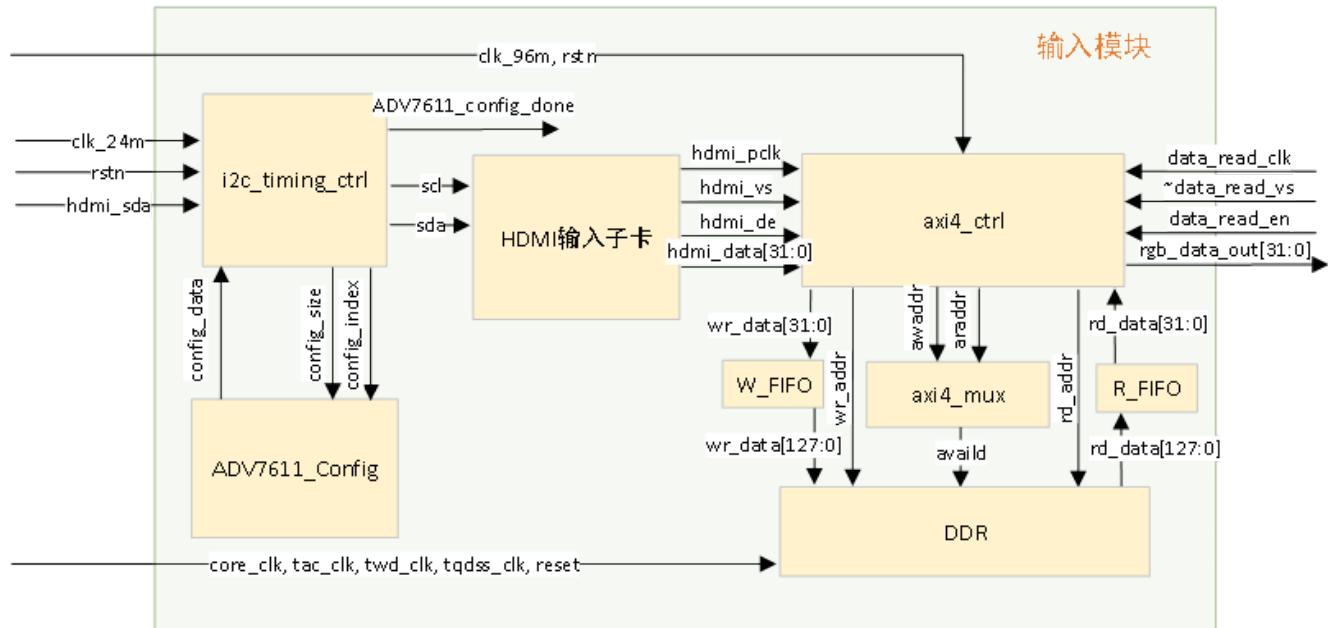


人机交互图形界面的实拍图如上图所示, 图形界面的渲染帧率为30Hz, 并搭配精心设计的模式切换动画, 可以提供极佳的交互体验。界面中可以调整视频的输出格式(实际也需要手动切换至对应输出格式的FPGA固件才可生效), 调整缩放采用的插值算法(图中为两种算法可选, 实际还有双三次插值算法), 调整可缩放的最大宽度和最小宽度, 以及演示的非线性动画类型。

在手动模式下, 可以手动点按(单次调整)/长按(连续变化)水平方向和垂直方向上的加减按键实现单个像素大小的缩放长、宽尺寸调整, 也可以手动拖动滑块条大幅度的调整缩放长、宽尺寸。

在线性缩放动画演示模式下, 单片机会以线性的方式连续改变缩放大小, 并实时显示出来。而在非线性动画演示模式下, 单片机会以非线性的方式连续改变缩放大小, 非线性变化过程共有3个预设选项, 即Bounce、Cric、Quint这三个常见的CCS缩放动画(Cascading Style Sheets Animations), 使项目更贴近实际使用场景, 与此同时, 图形界面也会实时显示当前的缩放尺寸大小。

### 2.1.2.2 HDMI输入模块 (设计1)



HDMI输入模块由i2c时序控制模块（i2c\_timing\_ctrl），ADV7611寄存器配置模块（ADV7611\_Config），HDMI输入子卡（硬件），DDR（硬件）以及AXI4总线模块（axi4\_ctrl, axi4\_mux）组成。

其中，ADV7611寄存器配置模块中存储着HDMI信号传输所需的EDID配置信息以及ADV7611芯片其他寄存器信息。i2c时序控制模块依据index信号和size信号按顺序从其中读取寄存器配置，并按照i2c总线协议输入进子卡的ADV7611芯片中，完成初始化，并将ADV7611初始化完成信号置高。时钟与复位信号由锁相环输入。

```
104:     LUT_DATA = {8'h6c, 8'd54, 8'hD8};  
105:     LUT_DATA = {8'h6c, 8'd55, 8'h09}; //25.2Mpc/clk  
106:     LUT_DATA = {8'h6c, 8'd56, 8'h80};  
107:     LUT_DATA = {8'h6c, 8'd57, 8'hA0};  
108:     LUT_DATA = {8'h6c, 8'd58, 8'h20}; //640-800  
109:     LUT_DATA = {8'h6c, 8'd59, 8'hE0};  
110:     LUT_DATA = {8'h6c, 8'd60, 8'h2D};  
111:     LUT_DATA = {8'h6c, 8'd61, 8'h10}; //480-525  
112:     LUT_DATA = {8'h6c, 8'd62, 8'h10}; //16  
113:     LUT_DATA = {8'h6c, 8'd63, 8'h20}; //32  
114:     LUT_DATA = {8'h6c, 8'd64, 8'hA2}; //10,2  
115:     LUT_DATA = {8'h6c, 8'd65, 8'h00}; //00  
116:     LUT_DATA = {8'h6c, 8'd66, 8'h80}; //scale  
117:     LUT_DATA = {8'h6c, 8'd67, 8'hE0};  
118:     LUT_DATA = {8'h6c, 8'd68, 8'h21};  
119:     LUT_DATA = {8'h6c, 8'd69, 8'h00};  
120:     LUT_DATA = {8'h6c, 8'd70, 8'h00};  
121:     LUT_DATA = {8'h6c, 8'd71, 8'h1E}; //5AF block
```

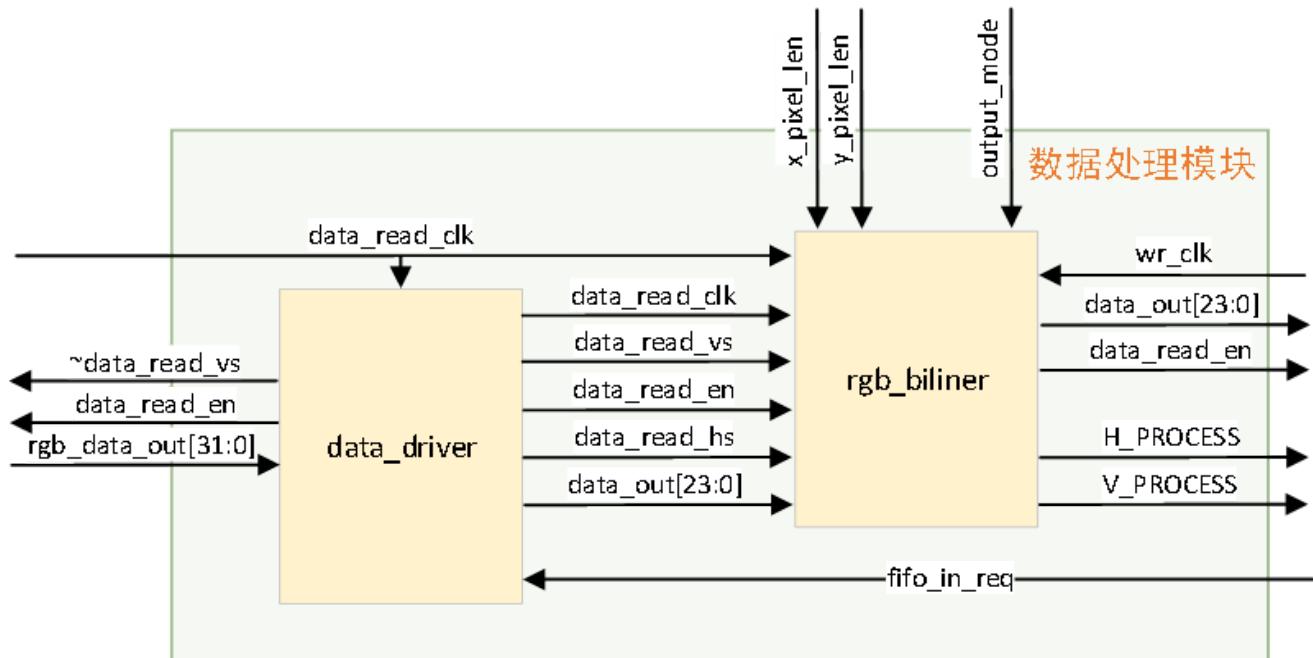
之后，HDMI输入子卡将输入的HDMI信号解码，输出像素时钟（hdmi\_pclk）、场信号（hdmi\_vs）、数据有效信号（hdmi\_de）以及像素RGB888信号（hdmi\_data[31:0]），其中场信号hdmi\_vs用于指示一帧视频图像的结束，写地址切换回初始地址。像素RGB888信号高8位补零，32位以对齐DDR输入位宽。通过AXI总线，将有效

数据同步存储进DDR中。在读数据一侧，AXI总线接收读时钟（data\_read\_clk），反读场信号（~data\_read\_vs），读有效（data\_read\_en），并同步输出有效像素信号。场信号用于指示一帧的结束。

AXI4总线通过突发传输的方式进行DDR的读写，突发位宽为128位，通过两个FIFO进行数据缓冲。axi4\_mux为多路选择器模块，将读写操作分离，避免了相互干扰。时钟与复位信号由锁相环输入。

DDR通过Efinity提供的ip核进行配置。时钟与复位信号由锁相环输入。由于DDR存储空间足以存储整帧画面，支持地址访问，且可以异步并发读写，作为数据缓冲区非常方便。

### 2.1.2.3 数据处理模块（设计1）



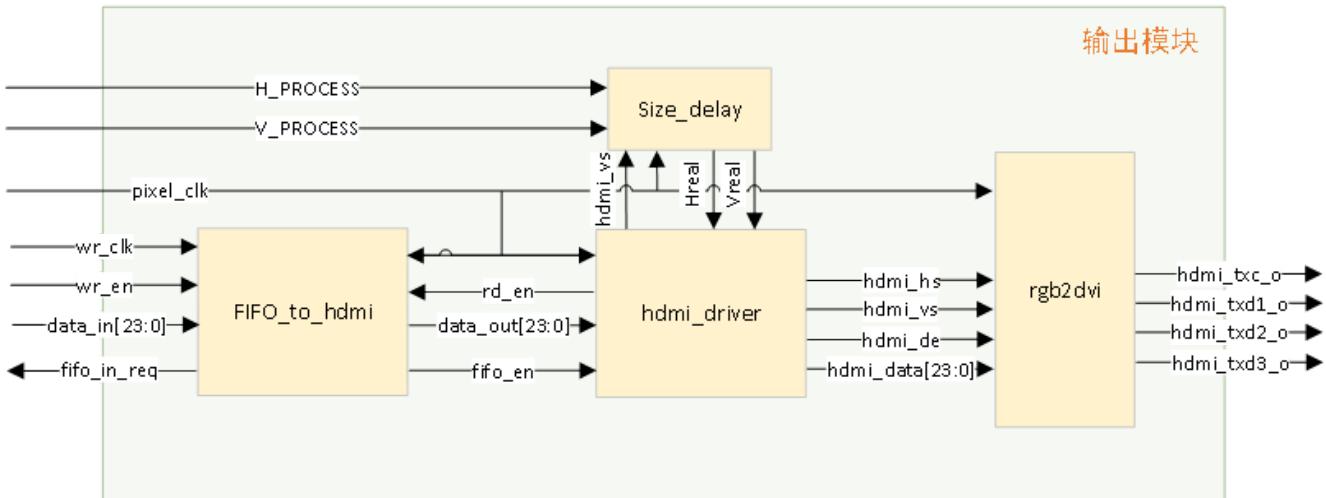
数据处理模块由数据修饰模块（data\_driver）与插值算法模块（rgb\_biliner）组成。由于DDR中存储的为纯数据，故需要数据修饰模块重新标志其行，场信号。数据修饰模块通过行列计数器，将从DDR中读取的信号按640\*480重新标志行，场信号，标记每一行，每一帧的开始与结束，便于插值算法有序运行。数据修饰模块输出完成一行数据后进入待机状态（计数器停止计数）；当输出FIFO传递的数据请求信号（fifo\_in\_req）为高且在待机时，计数器开始计数直到完成一行。

插值算法模块是将输入的640\*480图像按照给定的长度和宽度进行放大。在这个模块中，我们设计了异步时钟的双三次、双线性插值算法和最近邻算法。因此算法的input需要有RGB三通道的数据输入，输入的行信号，场信号，放大后图像的长，宽，以及模式选择的信号共8个信号。由于接下来过程的需要，这里的输出将RGB三个通道进行合并，并输出对应的行场信号。对于每一个通道，算法的处理都是相同的，我们在这里以R通道为例。

#### 2.1.2.3.1 双三次插值算法原理

[转到Algorithm文件夹下](#)

### 2.1.2.4 HDMI输出模块（设计1）



HDMI输出模块由输出缓冲模块（FIFO\_to\_hdmi），输出渲染模块（hdmi\_driver），HDMI差分输出驱动模块（rgb2dvi）以及尺寸对齐模块（Size\_delay）组成。

在不同的缩放尺寸下，为了保持输出时序的稳定性，需要将画面填充至输出分辨率大小（2560\*1440），输出渲染模块从输出缓冲FIFO中读取放大后的视频图像信息，通过行列计数器，填充满画布且按VGA时序规定填充消隐区，满足HDMI输出要求。

由于中间流水，数据进入输出缓冲到数据从输出缓冲中读出有若干时钟周期延时，且整张图像的大小远大于缓冲区深度。插值算法的目标缩放尺寸先于输出渲染的目标缩放尺寸变化，且时间差不超过一帧。在进行某一帧的输出渲染时，目标长宽应保持稳定，故以输出场信号作为触发（代表某一帧渲染完成），同步将输出目标长宽更新为算法目标缩放尺寸。

```

always @(posedge clk_pixel_2x or negedge rstn_pixel) begin//write frame next
  if(!rstn_pixel)begin
    hreal <= 640;
    vreal <= 480;
  end else begin
    if(!trig_vs_lcd)begin//write frame next
      hreal <= H_PROCESS;
      vreal <= V_PROCESS;
    end
  end
end
  
```

由于对DDR及AXI总线的分时复用可能会导致帧率降低，且输入侧HDMI输入子卡本身存在限制，输入信号的帧率固定为60，这可能会产生时间资源的冲突，且DDRip核较为复杂，所以没有对DDR进行复用，而是采用大FIFO作为缓冲区。

当输出缓冲FIFO中数据量首次超过某个值时，fifo\_en置高，输出渲染开始；当FIFO中数据量小于某个值时，fifo\_in\_req请求输入信号置高，将有一行信号输入进插值算法中，插值算法将连续输出输入一整行所产生的输出（保证插值算法无误）进入FIFO。FIFO驱动由Efinity提供的IP核实现。

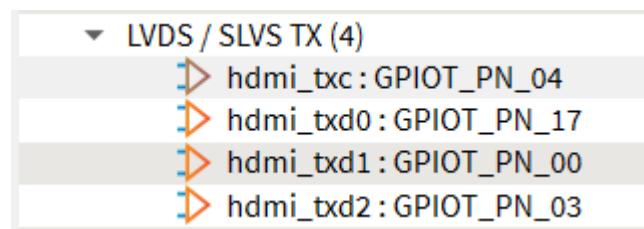
在一帧画面的像素信息中，我们的插值算法的有效数据-时间分布与输出渲染的有效数据-时间分布不同，具体地说，插值算法的每行的有效数据分布在行信号有效时间轴的左侧，每行均存在有效数据，行有效均匀分布于场有效时间轴中；输出渲染的行有效集中在场有效时间轴中，有效数据集中在行有效时间轴中。输出的画面占画布的比例越小，二者的分布不一致越严重，致使缩小能力，放大能力，帧率不可兼得，所以这里我们根据缩

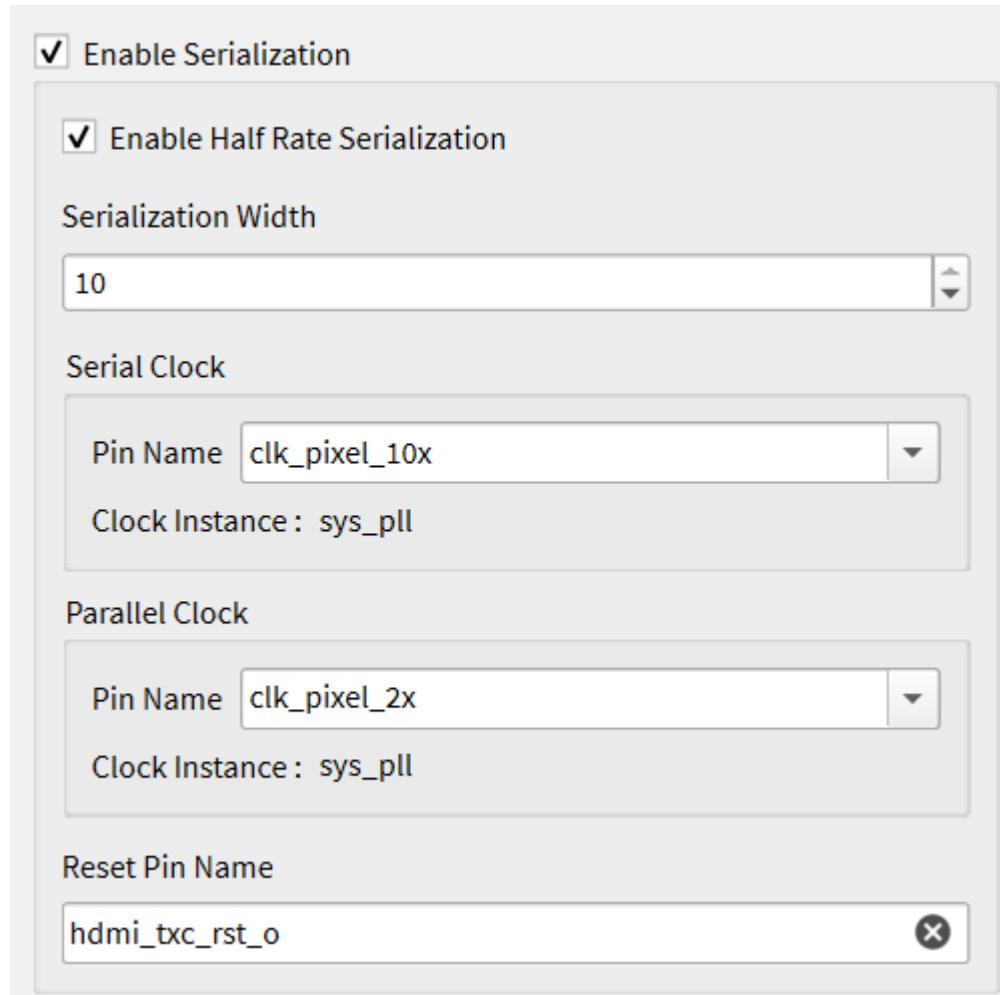
放后图像的长宽所在的范围动态的确定FIFO写入和读出的时机，最终使视频流输出稳定在640×480~2560×1440放大，50Hz的状态。

```
`ifdef VGA_2k
`define H_FRONT 12'd8
`define H_SYNC 12'd32
`define H_BACK 12'd40
`define H_DISP 12'd2560
`define H_TOTAL 12'd2640

`define V_FRONT 12'd27
`define V_SYNC 12'd8
`define V_BACK 12'd6
`define V_DISP 12'd1440
`define V_TOTAL 12'd1481
`endif
```

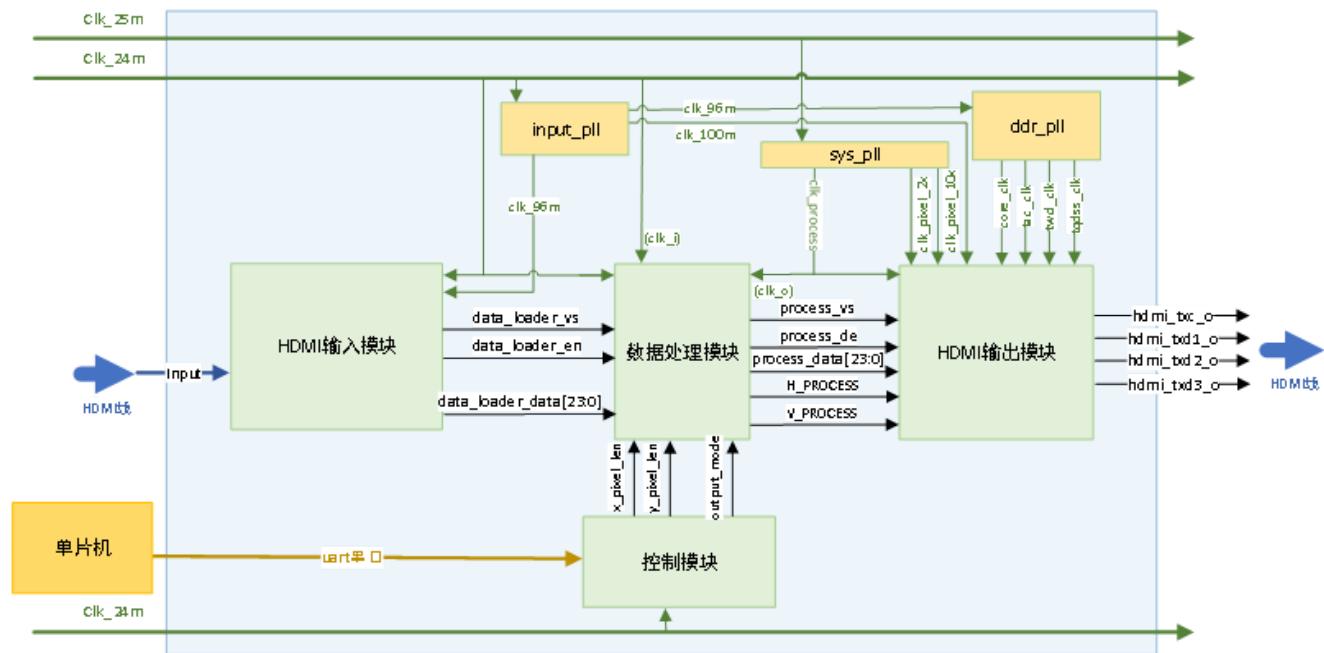
HDMI差分输出驱动模块通过Efinity提供的LVDS进行并—串转换与差分转换，将添加了消隐区的一帧图像通过底板上的HDMI输出接口输出到显示器或采集卡上显示。





## 2.2 设计2介绍

### 2.2.1 设计2整体介绍



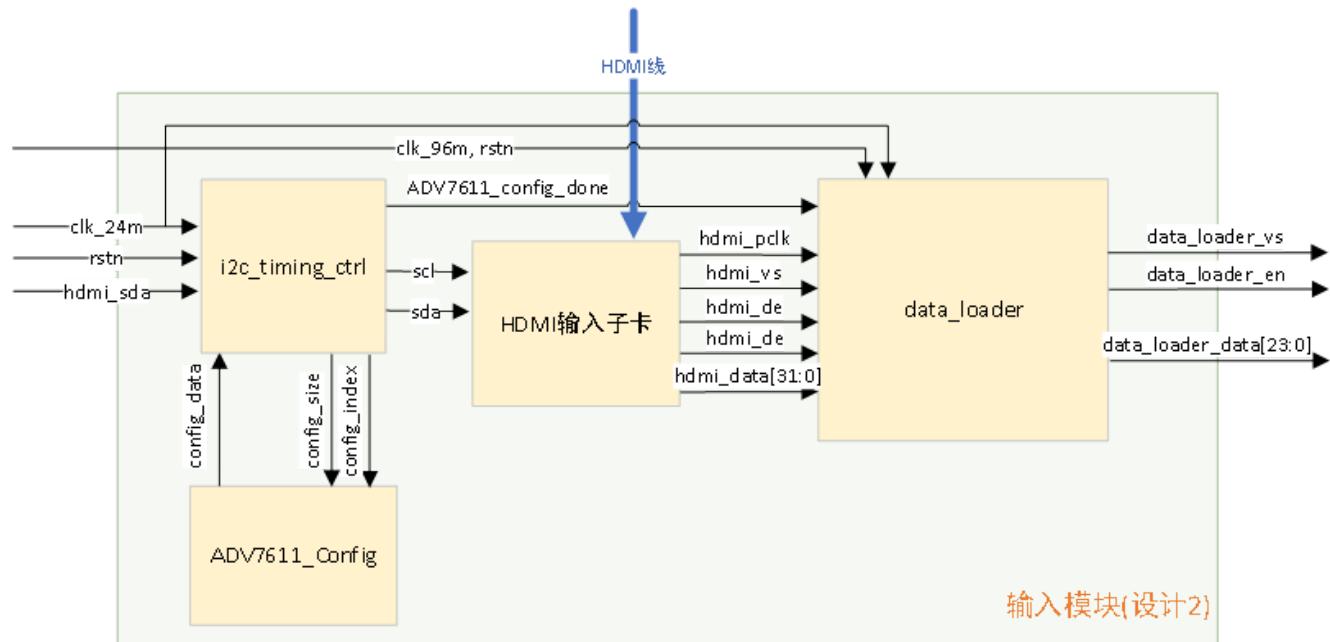
整体结构图与1类似，同样分为输入模块，数据处理模块，控制模块，输出模块。与设计1相比，其不同点在于将DDR作为输出端缓冲，FIFO作为输入端缓冲。由于DDR与FIFO作为缓冲区的特点不同，数据流处理不同，两种结构也具有不同的性能参数与瓶颈。

## 2.2.2 各模块介绍

### 2.2.2.1 控制模块 (设计2)

设计2的控制模块与设计1保持一致，此处不再重复。

### 2.2.2.2 HDMI输入模块 (设计2)

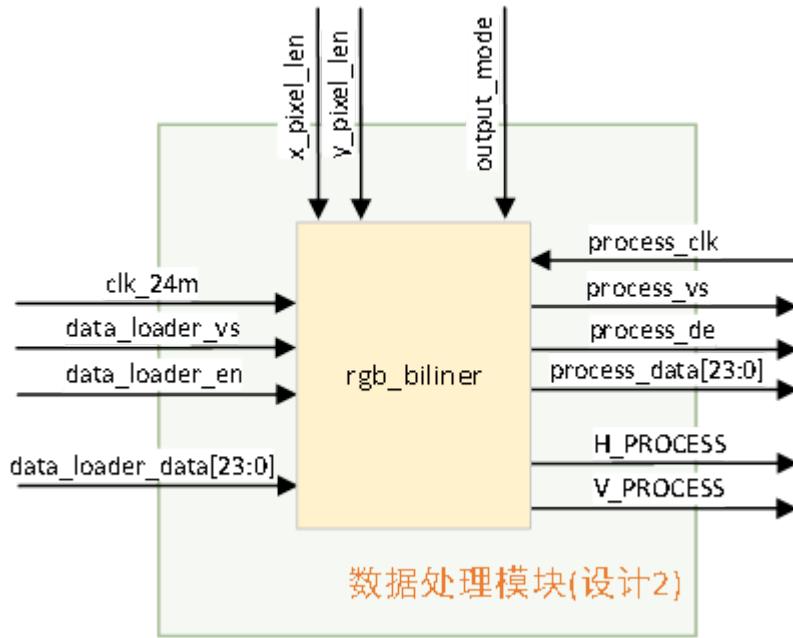


HDMI输入模块由I<sub>2</sub>C时序控制模块 (i2c\_timing\_ctrl) , ADV7611寄存器配置模块 (ADV7611\_Config) , HDMI输入子卡 (硬件) 以及输入缓冲及修饰模块 (data\_loader) 组成。

其中，HDMI输入子卡及其配置，I<sub>2</sub>C总线驱动与设计1相同，不再赘述。区别在于输入缓冲模块由DDR及配套AXI总线换为了由FIFO实现缓冲的data\_loader模块。

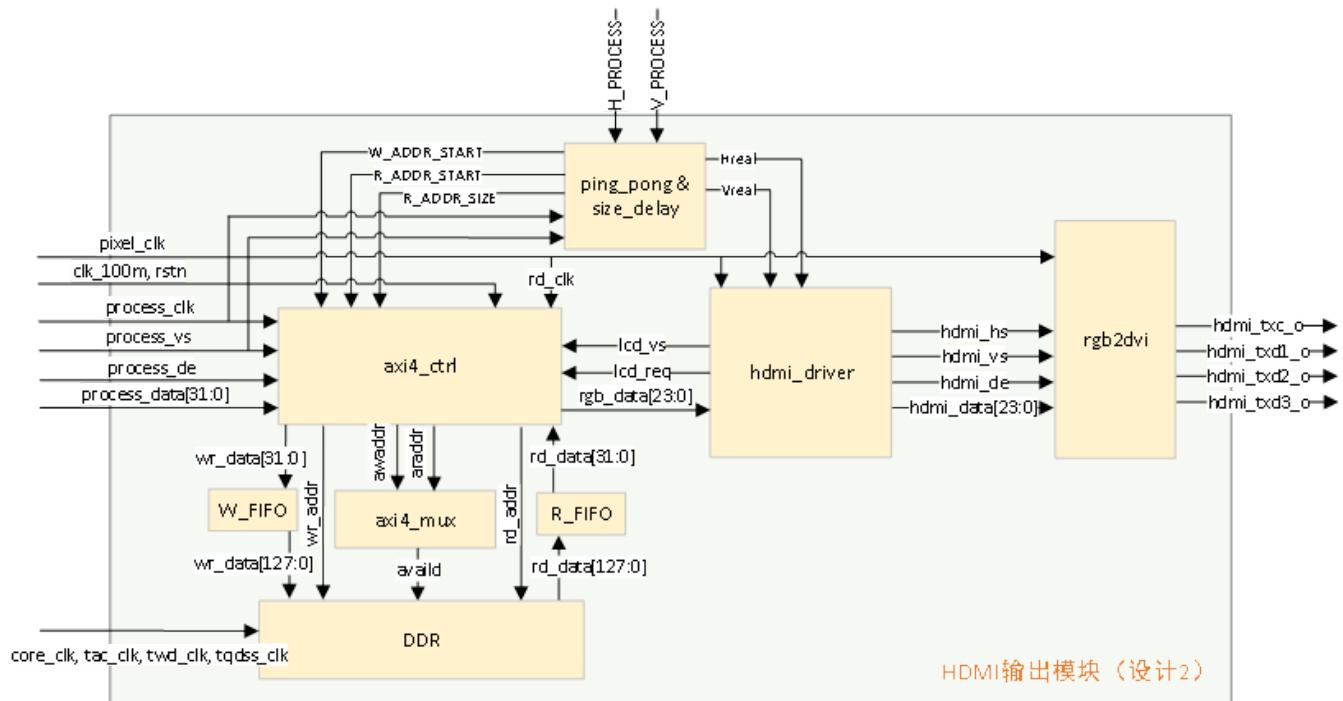
data\_loader直接获取ADV7611的输入数据并将有效的像素数据写入FIFO中，并在算法输入时钟的驱动下从FIFO中读出像素数据并生成对应的行场信号输出给算法数据处理模块。添加这一FIFO的目的是使数据输入更加集中一点，延长两场之间的数据空闲时间，给数据处理模块留出足够的场间间隔来进行长宽尺寸的更新（从控制模块获取）和一些比例常数的计算。

### 2.2.2.3 数据处理模块 (设计2)



数据处理模块仅由插值算法模块（`rgb_biliner`）组成。由于HDMI输入模块中，`data_loader`模块已经还原了行、场信号，设计1中的`data_driver`模块不再需要。除时钟频率外，这里的算法模块与设计1的实现相同，不再赘述。

#### 2.2.2.4 HDMI输出模块（设计2）



HDMI输出模块由DDR（硬件）以及AXI4总线模块（`axi4_ctrl`, `axi4_mux`），输出渲染模块（`hdmi_driver`），HDMI差分输出驱动模块（`rgb2dvi`）以及读写乒乓及尺寸对齐模块（`ping_pong & size_delay`）组成。

与设计1不同的是，输出缓冲由FIFO变为DDR。作为输出缓冲，DDR中存储的一帧视频图像的大小将由缩放尺寸决定，可能处于不断改变的状态，故这里对DDR采取乒乓读写方式：将DDR存储空间分为两个BLOCK，在某一帧的读写过程中，算法输出的数据写入BLOCK2/BLOCK1中，渲染画布并显示的数据从BLOCK1/BLOCK2中读出。若缺少这一部分，当缩放尺寸改变的瞬间，显示的画面将会由于读写过程中尺寸改变而产生错位。

BLOCK的切换由代表一帧HDMI输入/插值算法输出结束的process\_vs下降沿同步触发。这里输出时钟pixel\_clk为150M，折算为60.6帧，比60帧（理论输入/插值算法输出帧率）略快。当HDMI输出计数器计满一帧图像后停止计数，直到process\_vs下降沿到来时重新开始计数。这样将使得每一慢帧中系统的工作状态一致，避免了时钟误差导致的错位。

同时乒乓及尺寸对齐模块将上一帧的缩放尺寸保留，因为采取乒乓读写，输出的视频图像对应的尺寸是上一帧的插值算法的输出尺寸。故将插值算法对应的缩放尺寸延后一帧作为输出显示的尺寸。

输入进入DDR的一帧数据结束由process\_vs场信号标明，无需如R\_ADDR\_SIZE指定尺寸。

```
//////////  
always @(posedge clk_pixel or negedge rstn_sys) begin//write frame next  
    if(!rstn_sys)begin  
        hreal <= 640;  
        vreal <= 480;  
        bf_hreal <= 640;  
        bf_vreal <= 480;  
        frame_count <= 0;  
    end else if(!trig_vs)begin//write frame next  
        hreal <= H_PROCESS;  
        vreal <= V_PROCESS;  
        bf_hreal <= hreal;  
        bf_vreal <= vreal;  
        frame_count <= ~frame_count;  
    end  
end  
//////////  
//or  
  
reg delay_vs=1'b0;  
wire trig_vs;  
always @(posedge clk_pixel or negedge rstn_sys)begin  
    if(!rstn_sys)  
        delay_vs <= 1'b0;  
    else begin  
        delay_vs <= process_vs;  
    end  
end  
assign trig_vs = (process_vs) || (~delay_vs); //negetive trig  
assign BLOCK = 32'h0800_0000;//23+2+1=26bit  
assign W_ADDR_START = frame_count ? BLOCK : 0;  
assign R_ADDR_START = frame_count ? 0 : BLOCK;  
assign R_ADDR_SIZE = hreal*vreal*4;//or  
//////////
```

输出渲染模块(hdmi\_driver)与HDMI差分输出驱动模块(rgb2dvi)功能实现与设计1基本相同。通过行列计数器进行画布填充以及消隐区填充，通过Efinity提供的LVDS进行并—串转换与差分转换，将添加了消隐区的一帧图像通过底板上的HDMI输出接口输出到显示器或采集卡上显示。最终性能参数为160×120~1920×1080, 60Hz输出。

```
// 1920 * 1080
`ifdef VGA_1920_1080_60FPS_148_5MHz
`define H_FRONT 12'd88
`define H_SYNC 12'd44
`define H_BACK 12'd148
`define H_DISP 12'd1920
`define H_TOTAL 12'd2200

`define V_FRONT 12'd4
`define V_SYNC 12'd5
`define V_BACK 12'd36
`define V_DISP 12'd1080
`define V_TOTAL 12'd1125
`endif
```

## Stargazers over time

