## 基于圆形麦克风阵列和GW2A的声场摄像头系统

丛霄；沈俊杰；章轩然

**第一部分 设计概述**

* 1. 设计目的

基于麦克风阵列的声源定位技术可广泛用于视频会议，语音增强，智能机器人等领域，而声场摄像头系统可用于测量物体的位置与辐射的状态，可用于噪声源检测，局部放电检测，NVH优化等等，同时，由于半导体制造和制造过程中物联网的出现，预计在半导体制造行业，全球声学相机市场规模将出现可观的增长，尤其是在亚太地区等新兴经济体。此外，声学照相机可以直观看出噪声的位置，并且便于记录留证，解决取证难的问题。随着噪声法的推广，环境噪声监测也是未来发展的重要方向。

* 1. 应用领域

一、军事领域。本设备可以使用在军事领域，可用于士兵在野战、巷战等复杂场景中对潜在的敌人进行侦察。可以极大地提高士兵在作战场景中的侦察效率，减小潜在风险，提高作战的成功率。

二、城市街道噪声检测。本设备可用于城市街道噪声污染环境治理项目，用于检测城市街道等喧闹嘈杂环境中的噪声来源、噪声分布、噪声的成因分析等测试，助力城市噪声环境污染治理工作的有效进行

三、汽车NVH问题优化。利用声场摄像头，可以获取实时的噪声图像，通过可视化来研究声场各分布和车内共振模态、便于优化汽车NVH性能，辅助汽车音响系统的设计，对发动机、电机、轮胎、车厢内部设计的声学性能分析都有利于提高乘坐舒适性、保证产品的安全性、环保性。

* 1. 主要技术特点

1. 基于麦克风阵列的时延估计声源定位方法，使用FPGA实现了广义互相关时延估计法算法的实现
2. 硬件层面上实现了对摄像头数据的获取与显示，达到了几乎没有延迟的水准
3. 声场热力图显示，利用图像算法将声音的强度在位置上的分布按照伪色彩融合实际图像显示
4. 使用PICORV32软核对硬件电路进行辅助，进行串口调试与上位机的沟通
   1. 关键性能指标
5. 实现对1024\*768像素图像60FPS的的获取与显示
6. 麦克风采样率达到52.734KHz，对声音定位的精度达到厘米级别
7. 声场图像的融合，实现声场热力图的显示，同时添加了多种模式
   1. 主要创新点
8. 在fpga上实现了广义互相关函数的计算以及求极值，采用极低的硬件成本实现了复杂的计算方法（XCORR算法）
9. 声源定位的可视化，将声音一种听觉与视觉相结合，两种感官的结合使得声音的传播更加具象化
10. 实现多种伪彩色声场的显示，将图像算法和声音定位相结合
11. 使用更加先进且节省资源的数字麦克风，避免了使用模拟麦克风造成的外围电路复杂，误差大等问题
12. 系统结构简单，灵活性高，具有较大的实用价值

**第二部分 系统组成及功能说明**

* 1. 整体介绍

GW2A的系统应用：高云半导体 GW2A 系列 FPGA 产品是高云半导体晨熙家族第一代产品，内部资源丰富，具有高性能的 DSP 资源，高速 LVDS 接口以及丰富的 BSRAM存储器资源，这些内嵌的资源搭配精简的 FPGA 架构以及 55nm 工艺使GW2A 系列 FPGA 产品适用于高速低成本的应用场合。芯片架构如图2.1-1所示

图形用户界面

描述已自动生成

图2.1-1：GW2A芯片的结构框图

我们所采用的GW2A-LV18PG256C8/I7芯片具有2,0736LUTS,4个PLL资源，并且我们成功在芯片的移植了PICORV32这个RISC-V架构的MCU,运用其串口资源通过标准AHB总线传输数据，达到软核与FPGA通信的连接，系统整体框图如图2.1-2所示

声场摄像头系统主要由麦克风I2S数据采集模块，OV5640摄像头数据采集模块，HDMI显示模块，XCORR算法模块，PICORV32软核模块，图像融合算法模块组成。我们的XCORR算法为自己独创，首先通过逻辑分析仪采集7组麦克风阵列数据，带入matlab进行代码仿真，并且在Visual Studio上完成算法设计并验证，在云源软件上编写verilog代码完成设计。首先FPGA通过I2S协议，以52.734khz采样率采集声音数据，得到7组各方位麦克风阵列数据，依照时序进入SDBP的伪双端口B-SRAM进行存储，之后并行进入XCORR算法，根据TODA算法得到时延估计，同时我们调用摄像头完成采集数据，将数据通过video\_frame\_buffer IP核进行三帧缓存，传输给DDR3存储，PICORV32软核负责数据串口打印和系统模式切换，通过我们的串口上位机软件就能很好的分析数据以及更改系统模式，算法算得延迟估计之后，我们根据声音远场模型计算声源位置，并且通过声场成像算法将摄像头数据跟声场数据进行融合发送给hdmi显示模块，得到最终效果。



图2.1-2：作品主要流程图

* 1. 各模块介绍

①XCORR：

关于GCC-PHAT的两种实现（离散采样，非连续）

第一种是互相关算法：假设我们要计算1024个数据的正弦波的GCC-PHAT

我们首先会计算互相关函数R(SIN1,SIN2)，得到互相关函数应该是1024 \*2 -1个值，然后再对得到的这个数组进行FFT计算

sin\_fft=sin\_fft\*1./real(sin\_fft);

sin\_ifft=ifft(sin\_fft)

之后便会得到这个数组经过PHAT因子变换之后的广义互相关，对峰值进行检波，得到最大峰值对应的点的序号再进行频率采样模拟化就可以得到对应的延时。

delay\_number=100;

x1=([1024:-1:1])';

x2=([zeros(1,delay\_number),1024:-1:1+delay\_number])';

m=xcorr(x1,x2);

NFFT=length(m);

figure(1);

plot(-(NFFT-1)/2:(NFFT-1)/2,m,'g');

title('x2与x2的互相关函数图像');

n=fft(m);

n=ifft(n\*1./abs(n));

figure(2);

plot(-(NFFT-1)/2:(NFFT-1)/2,n,'r');

title('x2与x2的广义互相关函数图像');

第二种是直接求取功率谱：首先对两个信号（如果1024位）补0（补到2\*1024-1位）进行fft,得到他们的频谱H(SIN1),H(SIN2)，然后对这第二个信号进行共轭变换，之后对这两个信号进行点乘（.\*）之后

A=1./abs(P);

R1=ifft(A.\*P);

但是因为采样的问题，需要对得到的数据做一个类似matlab里面的shiftfft变换，也就是频谱左右交换，之后就会得到广义互相关图像，对图形进行最大峰值检波，得到最大峰值对应的点的序号再进行频率采样模拟化就可以得到对应的延时。

delay\_number=500;

x1=(1024:-1:1)';

x2=([zeros(1,delay\_number),1024:-1:1+delay\_number])';

NFFT=length(x1);

xk1=fft(x1,NFFT);

xk2=fft(x2,NFFT);

P=xk1.\*conj(xk2);

A=1./abs(P);

R1=ifft(A.\*P);

R2=fftshift(R1);

figure(2)

plot(-(NFFT)/2:(NFFT)/2-1,R2);

title('x2与x2的互相关函数图像');

我们的算法实践是首先根据输入的两个1024个硅麦数据进行依次先左移后右移的移位操作，依次将得到的序列进行累乘，然后串行比较数据大小，检测峰值，将得到的序列号传给图像声场融合模块，运用四组麦克风阵列数据，进行均值处理得到X，Y方向的声音延迟估计，根据声音远场模型，并且根据麦克风间距之间的距离估算，结合采样率分析，我们判断样本大致在正负30个样本点左右，所以只计算60组数据并且串行比较，极大的节省了代码与硬件资源。

图示

描述已自动生成

图2.2-1 远场模型的相位分析

图示

描述已自动生成

图2.2-2 近场模型的空间分析

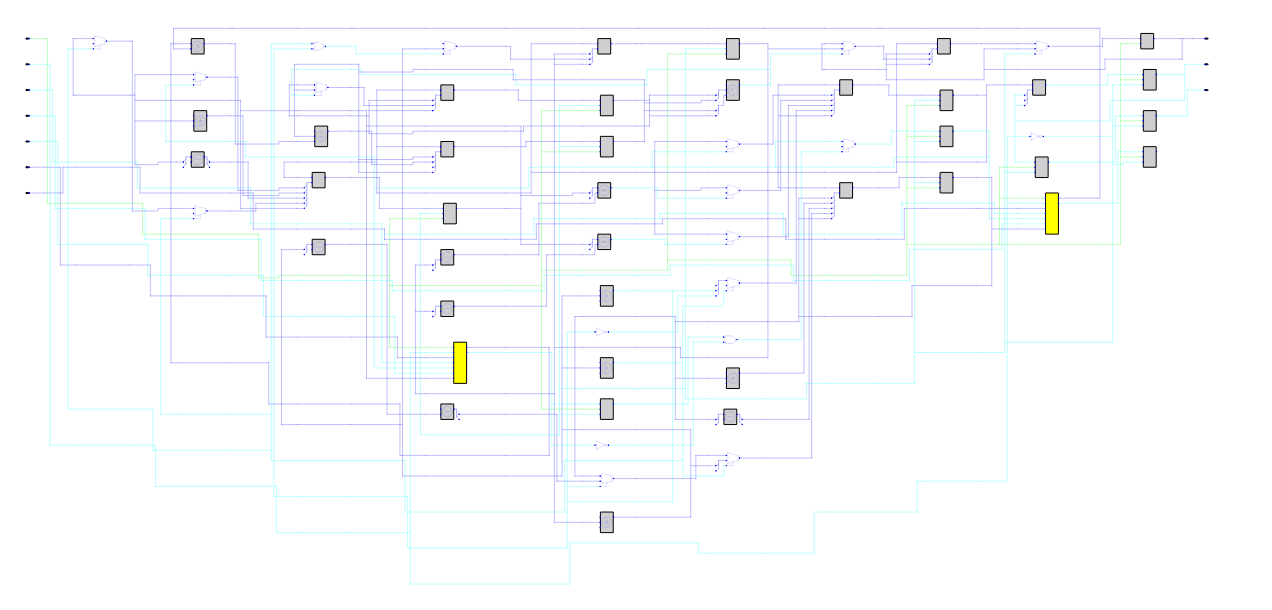


图2.2-3 XCORR算法的框图

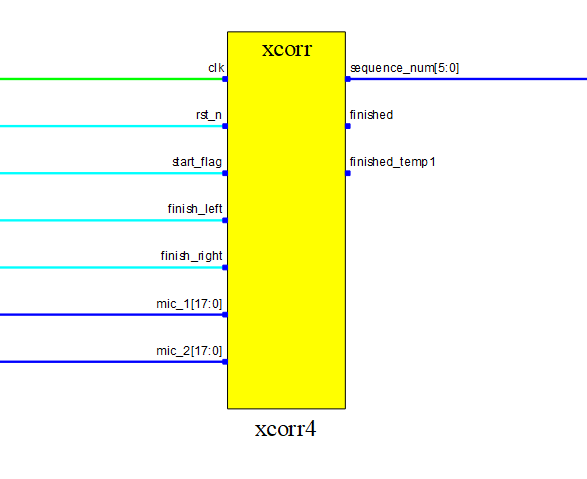


图2.2-4 XCORR算法

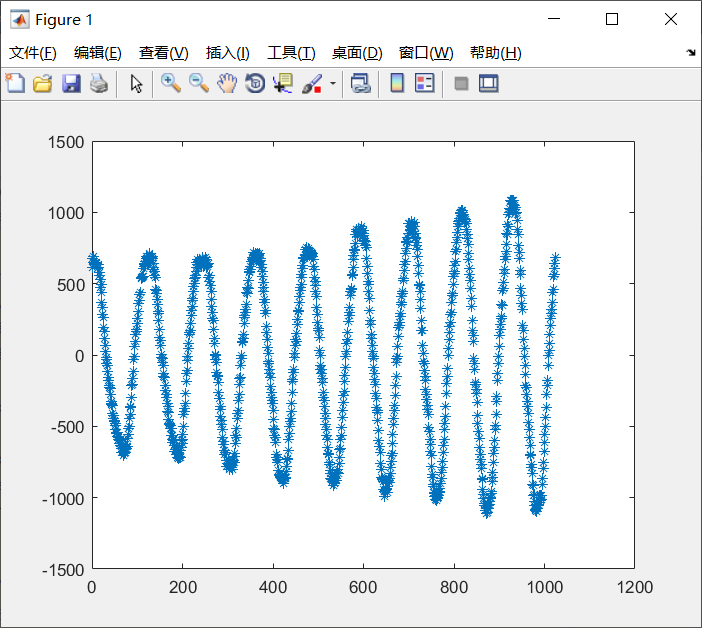
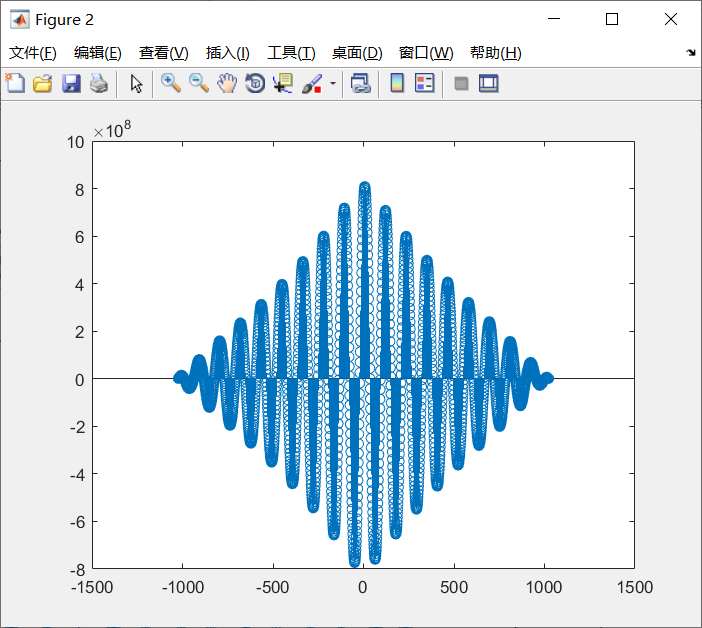


图2.2-5 MATLAB采集到的音频数据

  
图2.2-6 MATLAB采集到的音频数据做XCORR算法的波形图

②算法融合模块：X坐标采集我们选择SIPEED 6+1麦克风的0号与1号，3号与4号，Y坐标采集我们选用SIPEED 6+1麦克风的1号与3号，0号与4号，根据远场模型可知，我们可以将声音位置看做一个三维向量，跟我们两组麦克风之间形成的向量点乘，就可以得到声音单位向量的方向位置，再将XY坐标按照图像线性比例进行放大就可以得到估计到的坐标位置，而放大系数与两个麦克风之间的直线距离与摄像头距离麦克风模块的距离有关，在得到这两个缩放参数之后，根据我们制作MXL90640热成像仪的经验如图，我们将模式分为了：原图模式，灰度模式，金属2模式，伪彩2模式，将声场周围的像素点归一化到255，并且根据距离换算对应的声场热力图，并非简单的图片的叠加，切换这四个模式以外的其他模式只需添加一小段对于的色彩转换代码即可，非常的方便，切换模式通过软核PICORV32的串口切换，理论上模式数量无限大，图片给出了经典的模式的色彩图像，我们将图像跟需要显示的声场做一个二重叠加，将声场表现出透明性，效果如图所示



图2.2-7 声场成像表现出透明性

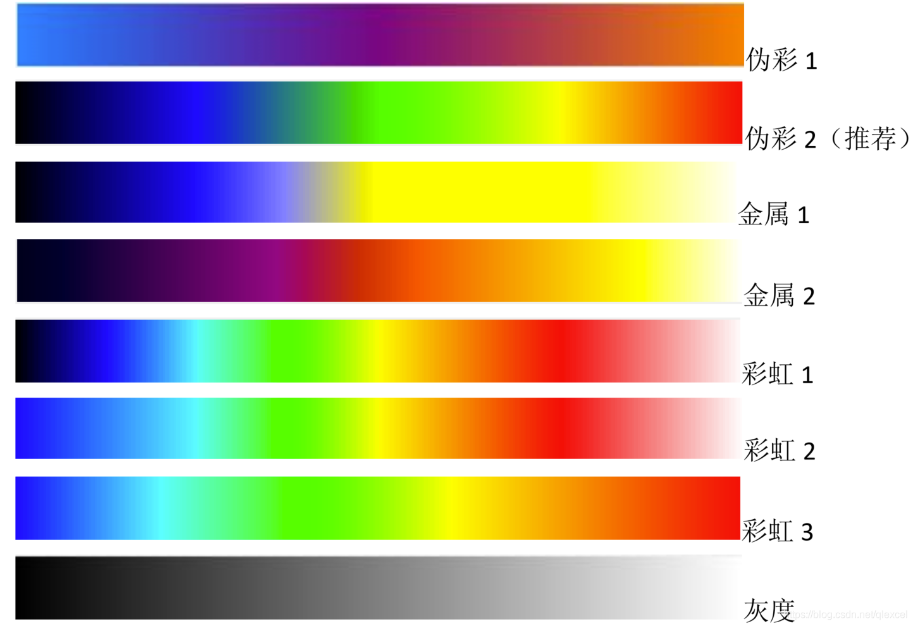


图2.2-8 经典的热成像色彩图

电视萤幕

描述已自动生成

图2.2-9 团队之前做过的热成像仪（MLX90640）

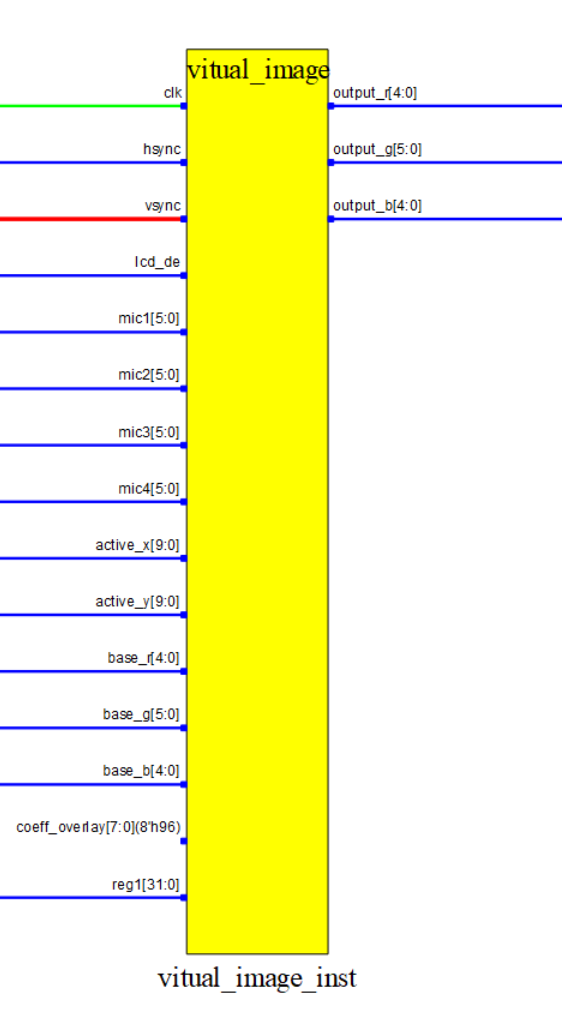


图2.2-10 虚拟成像接口

③HDMI(IP):HDMI显示器接口：

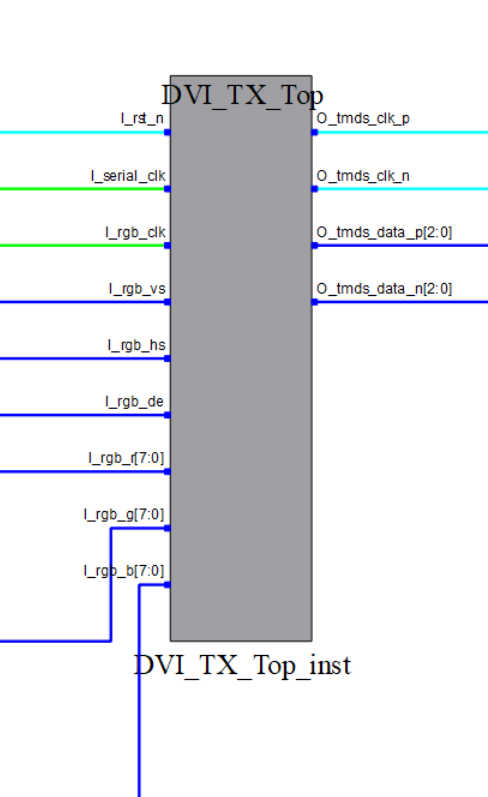


图2.2-11 DVI IP核

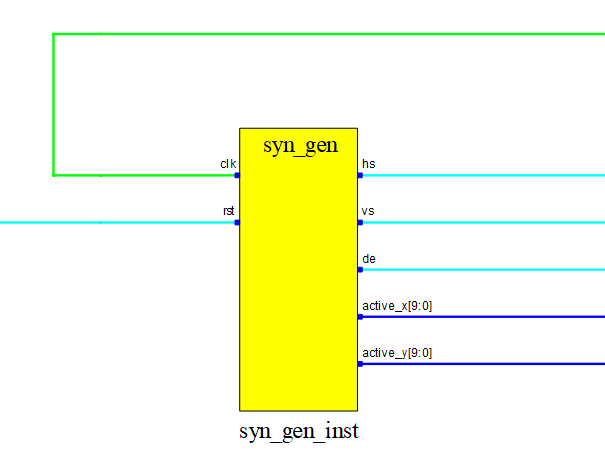


图2.2-12 HDMI时序生成

④OV5640摄像头接口：

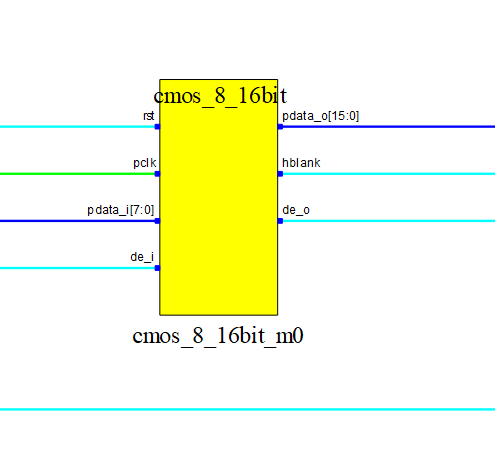


图2.2-13 摄像头输入转RGB565时序

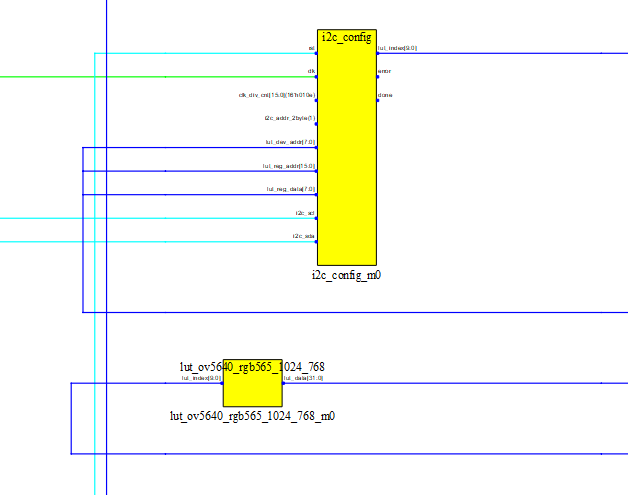


图2.2-14 摄像头SCCB时序

⑤硅麦I2S采集数据接口：

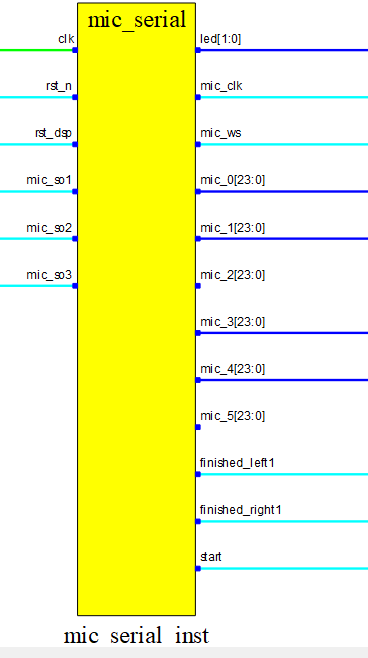


图2.2-15 麦克风数据采集

⑤PICORV32软核接口：

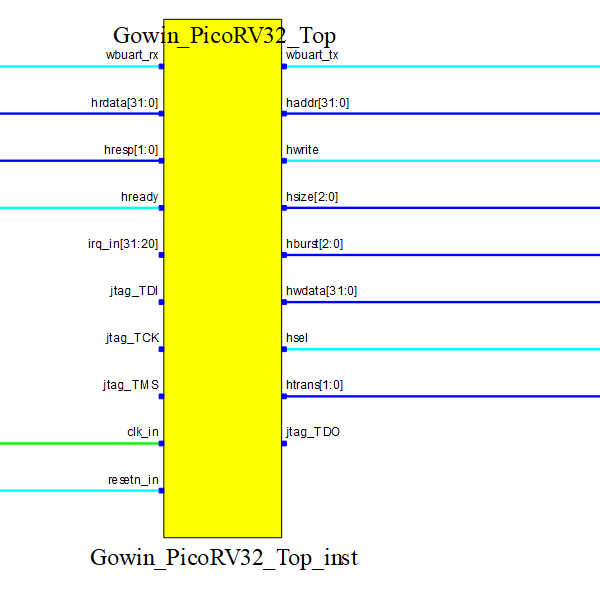


图2.2-16 高云picorv32(IP)

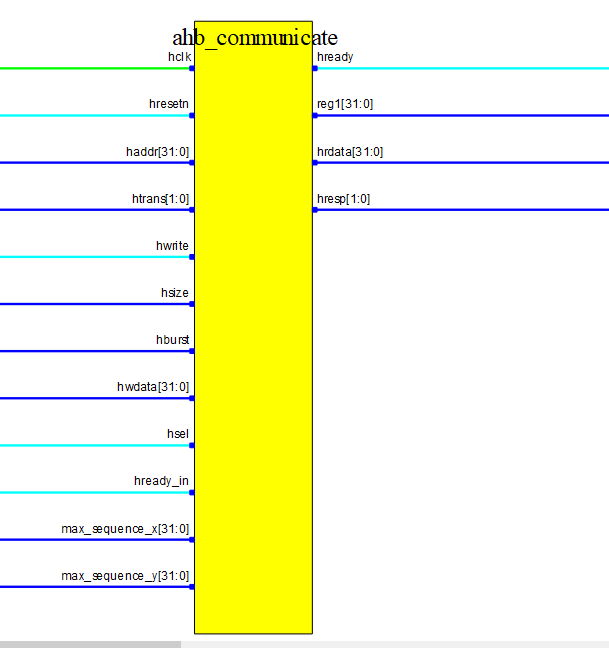


图2.2-17 AHB通信

**第三部分 完成情况及性能参数**

作品实拍：



原图显示：



灰度图显示：



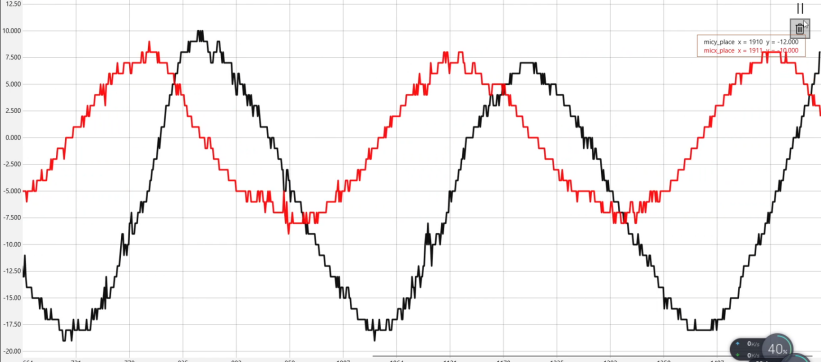
金属色显示：



伪彩色显示：



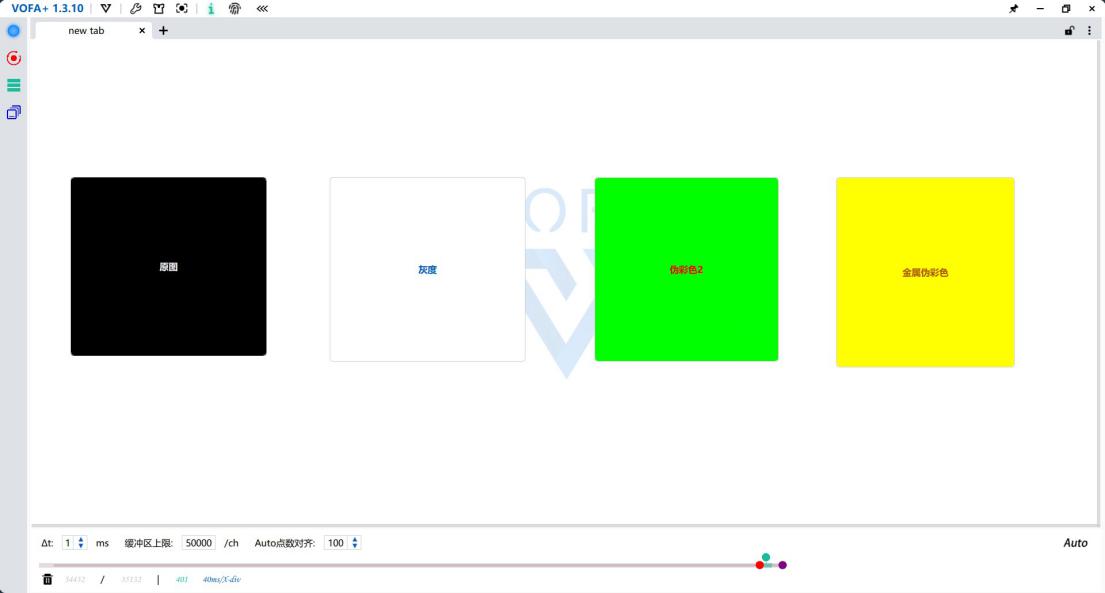
串口音频波形显示：



此为将声源绕麦克风阵列一周采集到的的X,Y轴坐标，是两个较为标准的相位差为90度的正弦波



串口上位机界面：



**第四部分 总结**

* 1. 可扩展之处

1. 可以采用近场模型进行声源定位的分析计算
2. 可以强化互相关算法比如引入经典GCC算法，GCC-ROTH,GCC-SCOT,GCC-ECKART等等算法以适应不同模型
   1. 心得体会

在实际生活中，声音定位是十分有意义的，比如语音降噪等，我们使用广义互相关函数实现算法。通过这次竞赛，我们极大地锻炼了自己的能力与扩展了自己的知识水平，通过对声音定位实际算法的了解与认识，我们对这方面有了更加深厚的兴趣，并将在比赛后不断加强学习不断完善自己。

**第五部分 参考文献**

[1]Bounded non-linear covariance based ESPRIT method for noncircular signals in presence of impulsive noise [J] . Jiacheng Zhang,Tianshuang Qiu,Shengyang Luan,Houjie Li. Digital Signal Processing . 2019

[2]A Low-Complexity Robust Beamforming Using Diagonal Unloading for Acoustic Source Localization [J] . IEEE/ACM Transactions on Audio, Speech and Language Processing （TASLP） . 2018 (3)

[3]Ship localization in Santa Barbara Channel using machine learning classifiers. [J] . Niu Haiqiang,Ozanich Emma,Gerstoft Peter. The Journal of the Acoustical Society of America . 2017 (5)

[4]Perception and prediction of apparent source width and listener envelopment in binaural spherical microphone array auralizations. [J] . Nowak Johannes,Klockgether Stefan. The Journal of the Acoustical Society of America . 2017 (3)

**第六部分 附录**

重要代码、推导过程等不便于在正文中体现的内容

Xcorr matlab 仿真代码如下

clc;clear;

load mic.mat

Fs=27000000/4/64;

s1=(mic((628914:629937)',5)/64);

s2=(mic((628914:629937)',4)/64);

% s2=[zeros(1,9),s2(10:1024)']';

[c,lags]=xcorr(s1,s2);

figure(2)

stem(lags,c)

ans\_test=(sum(s1.\*s2));

tea=gccphat(s1,s2);

N=1024;

figure(1)

plot(s1,'\*');

fd=fopen('F:\code\fpga\xcorr\_gcc\sim\sin33.txt','w');

for i=1:N

if(s1(i))<0

fprintf(fd,'%s\n',dec2bin(s1(i)+2^18,18));

else

fprintf(fd,'%s\n',dec2bin(s1(i),18));

end

end

fclose(fd);

XCORR verilog实现代码

module xcorr(input clk,

input rst\_n,

input start\_flag,

input finish\_left,

input finish\_right,

input signed[17:0]mic\_1,

input signed[17:0]mic\_2,

output reg[5:0]sequence\_num,

output reg finished,

output reg finished\_temp1);

reg finished\_temp;

wire data\_ok\_1\_flag;

wire data\_ok\_2\_flag;

reg data\_ok\_1\_flag\_temp;

reg data\_ok\_2\_flag\_temp;

reg data\_ok\_all\_flag;

reg [10:0]mic\_1\_cnt;

reg [10:0]mic\_2\_cnt;

reg [5:0]shift\_index;

reg signed[35:0] mul\_add\_result;

reg [10:0]address\_1;

reg [10:0]address\_2;

wire signed[17:0]data\_out\_1;

wire signed[17:0]data\_out\_2;

reg signed[35:0] mul\_add\_max;

parameter state\_0 = 3'b000 ;

parameter state\_1 = 3'b001 ;

parameter state\_2 = 3'b010 ;

parameter state\_3 = 3'b011 ;

parameter state\_4 = 3'b100 ;

parameter state\_5 = 3'b101 ;

reg [2:0]state;

initial begin

// data\_ok\_1\_flag = 0;

// data\_ok\_2\_flag = 0;

sequence\_num = 0;

data\_ok\_1\_flag\_temp = 0;

data\_ok\_2\_flag\_temp = 0;

data\_ok\_all\_flag = 0;

mic\_1\_cnt = 0;

mic\_2\_cnt = 0;

shift\_index = 0;

mul\_add\_result = 0;

address\_1 = 0;

address\_2 = 0;

mul\_add\_max = 0;

end

always @(posedge clk or negedge rst\_n) begin

if (!rst\_n)

begin

data\_ok\_1\_flag\_temp <= 0;

data\_ok\_2\_flag\_temp <= 0;

finished\_temp <= 0;

finished\_temp1 <= 0;

end

else

begin

data\_ok\_1\_flag\_temp <= data\_ok\_1\_flag;

data\_ok\_2\_flag\_temp <= data\_ok\_2\_flag;

finished\_temp <= finished;

finished\_temp1 <= finished\_temp;

end

end

always @(posedge clk or negedge rst\_n) begin

if (!rst\_n)

begin

data\_ok\_all\_flag = 0;

end

else if ((data\_ok\_1\_flag\_temp&&!data\_ok\_1\_flag))

begin

data\_ok\_all\_flag <= 1;

end

end

always @(posedge clk or negedge rst\_n) begin

if (!rst\_n)

begin

state = state\_0;

mic\_1\_cnt <= 0;

mic\_2\_cnt <= 0;

shift\_index = 0;

finished = 0;

mul\_add\_max = 0;

mul\_add\_result = 0;

end

else

begin

case (state)

state\_0://初始

begin

if (data\_ok\_all\_flag)

begin

mic\_1\_cnt <= 0;

state <= state\_1;

end

end

state\_1://左移

begin

if (mic\_1\_cnt == 1023-shift\_index)

state = state\_3;

//mul\_add\_result[shift\_index] = mul\_add\_result[shift\_index]+data\_out\_1\*data\_out\_2;

mul\_add\_result = mul\_add\_result+data\_out\_1\*data\_out\_2;

mic\_1\_cnt = mic\_1\_cnt+1;

mic\_2\_cnt = mic\_2\_cnt+1;

end

state\_2://右移

begin

if (mic\_1\_cnt == shift\_index-30)

state = state\_3;

mul\_add\_result = mul\_add\_result+data\_out\_1\*data\_out\_2;

mic\_1\_cnt = mic\_1\_cnt-1;

mic\_2\_cnt = mic\_2\_cnt-1;

// mul\_add\_result[shift\_index] = mul\_add\_result[shift\_index]+data\_out\_1\*data\_out\_2;

end

state\_3://shift\_index++

begin

if (mul\_add\_result>mul\_add\_max)

begin

mul\_add\_max = mul\_add\_result;

sequence\_num = shift\_index;

end

mul\_add\_result = 0;

shift\_index = shift\_index+1;

if (shift\_index <= 30)

begin

state <= state\_1;

mic\_1\_cnt <= 0;

mic\_2\_cnt <= shift\_index;

end

else if (shift\_index<61)

begin

state = state\_2;

mic\_1\_cnt <= 1023;

mic\_2\_cnt <= 1023-(shift\_index-30);

end

else

begin

shift\_index = 0;

state = state\_4;

end

end

state\_4://结束

begin

finished = 1;

end

state\_5:

begin

finished = 0;

end

endcase

end

end

mic\_data\_store mic\_data\_store\_inst1(

.clk(clk),

.rst\_n(rst\_n),

.mic\_1(mic\_1),

.start(start\_flag),

.output\_flag(data\_ok\_all\_flag),

.finish\_left\_or\_right(finish\_right),

.adb\_cnt(mic\_1\_cnt[9:0]),

.out\_data(data\_out\_1),

.output\_start\_flag(data\_ok\_1\_flag));

mic\_data\_store mic\_data\_store\_inst2(

.clk(clk),

.rst\_n(rst\_n),

.mic\_1(mic\_2),

.start(start\_flag),

.output\_flag(data\_ok\_all\_flag),

.finish\_left\_or\_right(finish\_left),

.adb\_cnt(mic\_2\_cnt[9:0]),

.out\_data(data\_out\_2),

.output\_start\_flag(data\_ok\_2\_flag));

endmodule

虚拟声场图像显示生成代码

module vitual\_image (input clk,

input hsync, // 行同步信号

input vsync, // 场同步信号

input lcd\_de,

input [5:0]mic1,

input [5:0]mic2,

input [5:0]mic3,

input [5:0]mic4,

input [9:0]active\_x, //video x position

input [9:0]active\_y,

input [4:0]base\_r,

input [5:0]base\_g,

input [4:0]base\_b,

input [7:0]coeff\_overlay, // overlay的叠加强度, 0~255

input [31:0]reg1,

output [4:0]output\_r,

output [5:0]output\_g,

output [4:0]output\_b);

//摄像头为1024x768 输入点阵大致为40 x40 首先对输入点阵进行处理

wire signed [6:0]mic\_cal\_1;

wire signed [6:0]mic\_cal\_2;

wire signed [6:0]mic\_cal\_3;

wire signed [6:0]mic\_cal\_4;

wire signed [7:0]mic\_cal\_x1;

wire signed [7:0]mic\_cal\_y1;

reg signed [7:0]mic\_cal\_x;

reg signed [7:0]mic\_cal\_y;

assign mic\_cal\_1 = $signed({1'd0,mic1})>30 ?$signed({1'd0,mic1})-30:(0-$signed({1'd0,mic1}));

assign mic\_cal\_2 = $signed({1'd0,mic2})>30 ?$signed({1'd0,mic2})-30:(0-$signed({1'd0,mic2}));

assign mic\_cal\_3 = $signed({1'd0,mic3})>30 ?$signed({1'd0,mic3})-30:(0-$signed({1'd0,mic3}));

assign mic\_cal\_4 = $signed({1'd0,mic4})>30 ?$signed({1'd0,mic4})-30:(0-$signed({1'd0,mic4}));

assign mic\_cal\_x1 = (mic\_cal\_1+mic\_cal\_2)/2;

assign mic\_cal\_y1 = (mic\_cal\_3+mic\_cal\_4)/(-2);

//assign mic\_cal\_x = mic\_cal\_x1[6:1];

//assign mic\_cal\_y = mic\_cal\_y1[6:1];

wire [7:0]coeff\_base; // base的叠加强度

assign coeff\_base = 8'd255 - coeff\_overlay;

reg [15:0]tmp\_r;

reg [15:0]tmp\_g;

reg [15:0]tmp\_b;

assign output\_r = tmp\_r[12:8];

assign output\_g = tmp\_g[13:8];

assign output\_b = tmp\_b[12:8];

wire [4:0]lcd\_r\_ov;

wire [5:0]lcd\_g\_ov;

wire [4:0]lcd\_b\_ov;

reg [10:0]lcd\_x;

reg [10:0]lcd\_y;

reg last\_hsync;

reg last\_vsync;

reg [14:0] prom\_address;

reg [23:0] fake\_color\_data;

reg [7:0]distance;

reg signed [22:0]dis\_sq;

always@(posedge clk) begin

if (lcd\_de == 0) begin

last\_hsync <= 0;

last\_vsync <= 0;

end

else begin

if (last\_vsync == 0 && vsync == 1) begin // vsync上升沿

lcd\_x <= 1'd0;

lcd\_y <= 1'd0;

mic\_cal\_x <= mic\_cal\_x1;

mic\_cal\_y <= mic\_cal\_y1;

end

else

if (last\_hsync == 1 && hsync == 0) begin // hsync上升沿

lcd\_x <= 1'd0;

lcd\_y <= lcd\_y + 1'd1;

end

else begin

lcd\_x <= lcd\_x + 1'd1;

end

last\_hsync = hsync;

last\_vsync = vsync;

end

end

parameter PIXEL\_NUM = 32'd1024;

assign {lcd\_r\_ov,lcd\_g\_ov,lcd\_b\_ov} = lcd\_de? {fake\_color\_data[23:19],fake\_color\_data[15:10],fake\_color\_data[7:3]}: 16'H0000;

always@(posedge clk) begin

if ($signed({1'd0,active\_x})>(512+70\*mic\_cal\_x))

begin

if ($signed({1'd0,active\_y})>(384+40\*mic\_cal\_y))

begin

dis\_sq = ($signed({1'd0,active\_x})-(512+70\*mic\_cal\_x))\*($signed({1'd0,active\_x})-(512+70\*mic\_cal\_x))+($signed({1'd0,active\_y})-(384+40\*mic\_cal\_y))\*($signed({1'd0,active\_y})-(384+40\*mic\_cal\_y));

end

else

begin

dis\_sq = ($signed({1'd0,active\_x})-(512+70\*mic\_cal\_x))\*($signed({1'd0,active\_x})-(512+70\*mic\_cal\_x))+((384+40\*mic\_cal\_y)-$signed({1'd0,active\_y}))\*((384+40\*mic\_cal\_y)-$signed({1'd0,active\_y}));

end

end

else

begin

if ($signed({1'd0,active\_y})>(384+40\*mic\_cal\_y))

begin

dis\_sq = ((512+70\*mic\_cal\_x)-$signed({1'd0,active\_x}))\*((512+70\*mic\_cal\_x)-$signed({1'd0,active\_x}))+($signed({1'd0,active\_y})-(384+40\*mic\_cal\_y))\*($signed({1'd0,active\_y})-(384+40\*mic\_cal\_y));

end

else

begin

dis\_sq = ((512+70\*mic\_cal\_x)-$signed({1'd0,active\_x}))\*((512+70\*mic\_cal\_x)-$signed({1'd0,active\_x}))+((384+40\*mic\_cal\_y)-$signed({1'd0,active\_y}))\*((384+40\*mic\_cal\_y)-$signed({1'd0,active\_y}));

end

end

if (dis\_sq>10000||(512+70\*mic\_cal\_x)<0||(512+70\*mic\_cal\_x)>1024||(384+40\*mic\_cal\_y)<0||(384+40\*mic\_cal\_y)>768)

distance = 0;

else

distance = 255-$unsigned(dis\_sq)\*255/10000;

case(reg1)

32'd0://原图

begin

tmp\_r = base\_r\*8'd255;

tmp\_g = base\_g\*8'd255;

tmp\_b = base\_b\*8'd255;

end

32'd1://gray

begin

fake\_color\_data[23:16] = distance;

fake\_color\_data[15:8] = distance;

fake\_color\_data[7:0] = distance;

tmp\_r = coeff\_base \* base\_r + coeff\_overlay \* lcd\_r\_ov;

tmp\_g = coeff\_base \* base\_g + coeff\_overlay \* lcd\_g\_ov;

tmp\_b = coeff\_base \* base\_b + coeff\_overlay \* lcd\_b\_ov;

end

32'd2://GCM\_Metal2

begin

if ((distance>= 0) && (distance<= 16))

fake\_color\_data[23:16] = 0;

else if ((distance>= 17) && (distance<= 140))

fake\_color\_data[23:16] = ((distance-16)\*255/(140-16));

else if ((distance>= 141) && (distance<= 255))

fake\_color\_data[23:16] = 255;

if ((distance>= 0) && (distance<= 101))

fake\_color\_data[15:8] = 0;

else if ((distance>= 102) && (distance<= 218))

fake\_color\_data[15:8] = ((distance-101)\*255/(218-101));

else if ((distance>= 219) && (distance<= 255))

fake\_color\_data[15:8] = 255;

if ((distance>= 0) && (distance<= 91))

fake\_color\_data[7:0] = 28+((distance-0)\*100/(91-0));

else if ((distance>= 92) && (distance<= 120))

fake\_color\_data[7:0] = ((120-distance)\*128/(120-91));

else if ((distance>= 129) && (distance<= 214))

fake\_color\_data[7:0] = 0;

else if ((distance>= 215) && (distance<= 255))

fake\_color\_data[7:0] = ((distance-214)\*255/(255-214));

tmp\_r = coeff\_base \* base\_r + coeff\_overlay \* lcd\_r\_ov;

tmp\_g = coeff\_base \* base\_g + coeff\_overlay \* lcd\_g\_ov;

tmp\_b = coeff\_base \* base\_b + coeff\_overlay \* lcd\_b\_ov;

end

32'd3://GCM\_Pseudo2

begin

if (distance <= 63)

begin

fake\_color\_data[23:16] = 0;

fake\_color\_data[15:8] = 0;

fake\_color\_data[7:0] = (distance\*255/64);

end

else if ((distance>= 64) && (distance<= 127))

begin

fake\_color\_data[23:16] = 0;

fake\_color\_data[15:8] = ((distance-64)\*255/64);

fake\_color\_data[7:0] = ((127-distance)\*255/64);

end

else if ((distance>= 128) && (distance<= 191))

begin

fake\_color\_data[23:16] = ((distance-128)\*255/64);

fake\_color\_data[15:8] = 255;

fake\_color\_data[7:0] = 0;

end

else if ((distance>= 192) && (distance<= 255))

begin

fake\_color\_data[23:16] = 255;

fake\_color\_data[15:8] = ((255-distance)\*255/64);

fake\_color\_data[7:0] = 0;

end

tmp\_r = coeff\_base \* base\_r + coeff\_overlay \* lcd\_r\_ov;

tmp\_g = coeff\_base \* base\_g + coeff\_overlay \* lcd\_g\_ov;

tmp\_b = coeff\_base \* base\_b + coeff\_overlay \* lcd\_b\_ov;

end

//32'd4:

endcase

end

endmodule