***2022***



**接口技术 实验报告**

j0242087[1]

|  |  |
| --- | --- |
| 专 业： | 计算机科学与技术 |
| 班 级： | CS2207 |
| 学 号： | U202215561 |
| 姓 名： | 瞿明睿 |
| 电 话： | 13956929603 |
| 邮 件： | [3021018823@qq.com](mailto:3021018823@qq.com) |
| 完成日期： | 2024.12.19 |



目 录

[1 课程实验概述 2](#_Toc185525646)

[1.1 实验目的 2](#_Toc185525647)

[1.2 实验任务 2](#_Toc185525648)

[1.3 实验要求 2](#_Toc185525649)

[2 综合应用：音频输入输出接口 3](#_Toc185525650)

[2.1 设计目标 3](#_Toc185525651)

[2.2 总体方案 3](#_Toc185525652)

[2.3 硬件设计与实现 3](#_Toc185525653)

[3 总结与心得 14](#_Toc185525654)

[3.1 实验总结 14](#_Toc185525655)

[3.2 实验心得 14](#_Toc185525656)

[3.3 意见和建议 15](#_Toc185525657)

[致谢 16](#_Toc185525658)

# 课程实验概述

## 实验目的

《接口技术》是一门工程性和实践性都非常强的计算机专业课程。为了充分调动学生的学习积极性，体现研究性学习，提高学习效果，在各个理论知识点讲述的过程中，设计了有针对性的实验。实验注重将各章节知识有机结合，循环展开，逐步深入，使学生更加牢固的掌握接口技术，同时树立计算机的系统观和软硬件协同观，为后续计算机系统综合能力的培养奠定坚实的基础；同时，力求通过这些硬件类课程的学习同时提升学生软件设计水平。

课程实验属于设计型实验，不仅锻炼学生简单计算机系统的设计能力，而且通过具体接口的设计与实现，进行硬件的实现、故障分析与定位，以及接口的应用和系统集成，使学生在系统调试等环节得到综合锻炼，进一步提高学生分析和解决问题的能力。

## 实验任务

设计一个基于RVfpga\_SoC处理器系统的接口应用综合系统，该接口综合系统是一个具有一定实用价值的应用系统，在前面接口实验的基础上，在RVfpga\_SoC处理器系统中至少再添加一个新的外设模块（注：即前面实验没有使用过的，这个外设模块可以是Vivado自带的、也可以是现成的免费的IP模块、还可以是自定制的IP模块），完成硬件系统的搭建并生成比特流（bitstream），完成接口驱动程序和应用软件的设计，另外综合应用系统要能够充分展示同学们的想象力和创造性。

## 实验要求

完成实验任务并撰写设计报告（具体撰写内容见后面的模板！），提交能够生成综合应用系统硬件平台比特流（bitstream）的Vivado工程源程序、接口驱动程序和应用软件的源代码。

# 综合应用：音频输入输出接口

## 设计目标

这里明确我们的设计目标：我们需要实现的总体功能是，可以监听开发板上的麦克风收到的声音，总体来说我们实现的是一个简单的监听系统，从开发手册上我们可以得知：麦克风输入即MIC接口，Nexys4 DDR配备了一个MEMS麦克风ADMP421，它的输出为PDM脉冲密度调制信号。音频输出，AUD\_PWM接口：音频输出通过一个PWM信号实现，音频数据需要被转换为PWM信号。

## 总体方案

在这里我写出我的具体思路：

我们可以先实现一个并不是基于RVfpga的监听方案，了解一下具体的流程，以及我们如何处理PDM脉冲密度调制信号，PDM信号如何通过解调得到 PCM脉冲编码调制信号，如何根据统计的PDM计数值生成PWM信号。

再将我们的思路加入我们已经成熟的RVfpga方案，如何利用AXI总线实现数据传递。

## 硬件设计与实现

### 初步简单硬件设计

在板上直接实现，我们需要做到3个功能：

1.我们需要一个不同于系统的时钟，我们可以用采样的方式产生mic\_clk。

2.PDM处理：MIC的输出为PDM，我们可以通过统计每个PDM周期中MIC\_DATA信号为高的次数，并用计数器记录这些值。这里用一个周期为128的计数器，每个周期内如果MIC\_DATA为高，增加计数器值。在特定的起始点初始化 PWM\_thresh\_counter。PWM\_thresh\_counter 数组记录10个不同频段的计数值。

3.PWM 波形生成：根据统计的PDM计数值生成PWM信号使用一个周期为512的计数器led\_count，与统计的 PDM 值进行比较。如果计数器值小于 led\_threshold，输出 PWM 高电平，否则为低电平。led\_threshold 的值在每个大周期（5120）中，根据不同频段的 PWM\_thresh\_counter 动态更新。

提及一下具体的硬件:在板上麦克风芯片是Analog Device的ADMP421，通过手册可以看到A7芯片和它的连接非常简单，但是驱动方式是脉冲密度调制也就是PDM，在后面的简单实现和IP控制器中，我们把100MHz系统时钟降低40倍成2.5MHz使用。在PDM中，1对应正脉冲，0是负脉冲，保持1代表最大正值，保持0代表最大负值。在其中外界输入的模拟是通过delta-sigma调制实现，同时这里还有左右声道L/R SEL，高电平代表左声道，低电平代表右声道。

在板上音频输出驱动用的不是具体的芯片，这里看板子上也没有焊接相关芯片，具体来说使用的四阶低通滤波器，将PWM方波转换成相应的正弦波给3.5mm音频输出口。这里通过设计图纸可以发现AUD\_PWM连接到A7芯片的A11引脚，AUD\_SD连接到A7芯片的D12引脚，如果AUD\_SD为低电平不经过放大，高电平则使能放大器。

这里具体讲一下PWM方波转换输出的逻辑，在一个Pulse窗口内的高电平比例越大，那么模拟输出的电压值就越高。在这里需要明确：PWM波和PDM波不一样，是不可以直接使用PDM波的。

### 硬件实现

我们初步这样设计：100MHz系统时钟用来生成PWM波，512个时钟为一个周期，用一个阈值来控制PWM波中高电平的占比，再用系统时钟减缓40倍到2.5MHz（周期400ns）作为PDM波的读取时钟

通过MIC得到的PDM波读取128个样本的时间为400ns x 128 = 51.2us，PWM一个周期为10ns512=5.12us，正好是10倍，设计10个计数器交替计数。

以PDM波的128个采样计数为基准，定出10个参考点，每当PDM计数到这个参考点就清空相应计数器的值，重新计数，这样每当到达参考点时，相应计数器就记录了128个采样的高电平比例。10个参考点位置为floor(128\*i/10)={0, 12, 25, 38, 51, 64, 76, 89, 102, 115}

由于数字逻辑中只有整数，设置另一个计数器，频率是100MHz，以5120为周期（128个PDM采样周期），在十个PDM参考点的40倍即计数器完成计数时读取密度，乘以4倍（上限128转换到512），作为下一个PWM周期的阈值

input clk,

input rst,

output reg led,

input sd\_sw, //AUD\_SD\_input

output reg MIC\_CLK,

input MIC\_DATA,

output reg MIC\_LR\_SEL,

output reg AUD\_PWM,

output reg AUD\_SD

///////////////////////////////////////////

PDM处理：采样MIC\_DATA信号和处理

///////////////////////////////////////////

reg [7:0] pdm\_sample\_counter;

reg [9:0] pdm\_high\_count[9:0];

localparam [79:0] SAMPLE\_STARTS = {8'd115, 8'd102, 8'd89, 8'd76, 8'd64,

8'd51, 8'd38, 8'd25, 8'd12, 8'd0};

always @(posedge clk or posedge rst) begin

if (rst) begin

pdm\_sample\_counter <= 8'd0;

end else if (mic\_clk\_rising\_edge) begin

if (pdm\_sample\_counter == 8'd127) begin

pdm\_sample\_counter <= 8'd0;

end else begin

pdm\_sample\_counter <= pdm\_sample\_counter + 8'd1;

end

end

end

generate

genvar i;

for (i = 0; i < 10; i = i + 1) begin : PDM\_HIGH\_COUNTERS

always @(posedge clk or posedge rst) begin

if (rst) begin

pdm\_high\_count[i] <= 10'd0;

end else if (mic\_clk\_rising\_edge) begin

if (pdm\_sample\_counter == SAMPLE\_STARTS[8\*i +: 8]) begin

pdm\_high\_count[i] <= mic\_data ? 10'd1 : 10'd0;

end else begin

pdm\_high\_count[i] <= mic\_data ? pdm\_high\_count[i] + 10'd1 : pdm\_high\_count[i];

end

end

end

end

endgenerate

//////////////////////////////////////////////

PWM生成

//////////////////////////////////////////////

reg [15:0] pwm\_cycle\_counter;

reg [15:0] current\_threshold;

always @(posedge clk or posedge rst) begin

if (rst) begin

pwm\_cycle\_counter <= 16'd0;

current\_threshold <= 16'd0;

end else begin

pwm\_cycle\_counter <= (pwm\_cycle\_counter == 16'd5119) ? 16'd0 : pwm\_cycle\_counter + 16'd1;

case (pwm\_cycle\_counter)

16'd0 : current\_threshold <= {6'd0, pdm\_high\_count[0], 2'd0};

16'd480 : current\_threshold <= {6'd0, pdm\_high\_count[1], 2'd0};

16'd1000 : current\_threshold <= {6'd0, pdm\_high\_count[2], 2'd0};

16'd1520 : current\_threshold <= {6'd0, pdm\_high\_count[3], 2'd0};

16'd2040 : current\_threshold <= {6'd0, pdm\_high\_count[4], 2'd0};

16'd2560 : current\_threshold <= {6'd0, pdm\_high\_count[5], 2'd0};

16'd3040 : current\_threshold <= {6'd0, pdm\_high\_count[6], 2'd0};

16'd3560 : current\_threshold <= {6'd0, pdm\_high\_count[7], 2'd0};

16'd4080 : current\_threshold <= {6'd0, pdm\_high\_count[8], 2'd0};

16'd4600 : current\_threshold <= {6'd0, pdm\_high\_count[9], 2'd0};

endcase

end

end

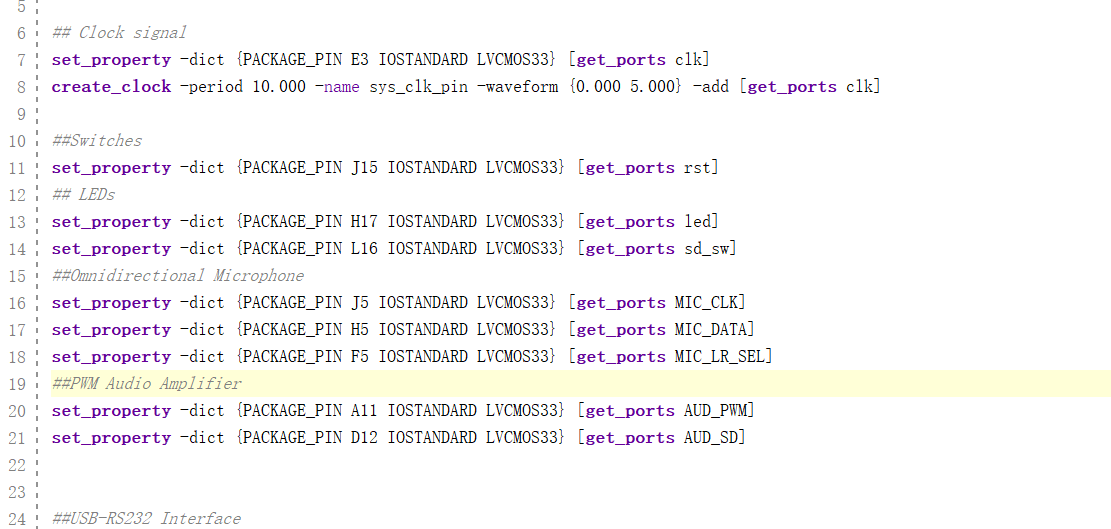


图2.1 初步实现的XDR文件

### 初步完成的问题以及整合进入RVfpga思路

从结果来看，实现的效果并不好，通过SD放大使能器前后都有非常大发白噪声，无法实现比较好的效果，简而言之就是没有滤波操作，仅通过计数高电平次数的方式来处理 PDM 数据，就是通过简单的积分器实现的转化。

为了实现方便MIC\_LR\_SEL信号始终为 0，没有在任何地方使用，也没有切换左右声道数据的逻辑。我们可以增加逻辑，每隔一定周期切换 MIC\_LR\_SEL，分别采样左声道和右声道数据。（但是实际上手册中并没有提及双声道stereo的部分，而且pwm本身就是单声道的方案）

如何整合进入RVfpga：

第一种思路：将初步设计的MIC成功封装作为一个AXI4总线的IP并通过块设计加入RVfpga，同时在XDR文件中成功绑定我们的端口，并且在PlatformIO中需要能够使用这个功能（理论上在硬件上也可以实现，作为一个单独的功能附加在模块周围也是可以的也许）

第二种思路：我们完全可以不利用块设计，在rvfpga顶层文件直接实例化我们的mic模块实现直接的移植，但是这样并不是直接利用了我们的AXI，所以不与采用。

### RVfpga实现

IP实现：这里我们在最初的简单实现中可以发现，我们需要几个接口：系统时钟和重置信号、一位led表示SW状态，一个SW来提示是否用AUD\_SD，音频的接口：MIC\_CLK, MIC\_DATA, MIC\_LR\_SEL, AUD\_PWM, AUD\_PWM。这里和初步实现一样，修改我们的XDR文件（这里应该也是可以在IP EDIT页面中创建独立的XDR文件，只要板子定义是对的其实不会有问题）。

此外我们还需要能够控制这个模块，我们加入enable来通过AXI总线控制。

在IP顶层文件中加入我们需要的端口定义，如下图

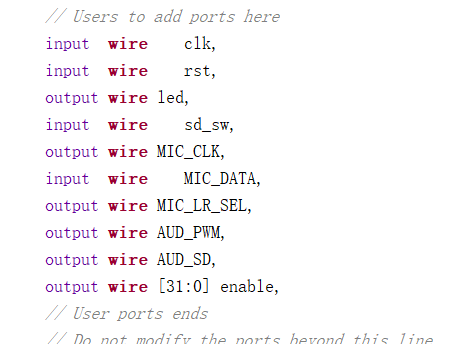


图2.2 IP顶层文件端口设计

在控制器中，我们需要修改我们的设计让这个模块工作的更加完整，在前面提到过，在mic采样的过程中我们偷懒仅仅是用一个声道的数据，这里作出修改。

//切换MIC\_LR\_SEL实现左右声道采样

always @(posedge clk or posedge rst) begin

if(rst) begin

MIC\_LR\_SEL <= 1'b0;

end

else begin

MIC\_LR\_SEL <= ~MIC\_LR\_SEL; //切换MIC\_LR\_SEL实现左右声道切换

end

end

我们之前并没有滤波，这里也应该加入滤波模块，这里我就偷懒使用了FIR低通滤波器来去除高频噪声，不对中频段做更多的修改。

// FIR滤波器

reg [15:0] mic\_data\_buffer[3:0];

wire [15:0] filtered\_data;

always @(posedge clk or posedge rst) begin

if (rst) begin

mic\_data\_buffer[0] <= 16'd0;

mic\_data\_buffer[1] <= 16'd0;

mic\_data\_buffer[2] <= 16'd0;

mic\_data\_buffer[3] <= 16'd0;

end

else if (MIC\_CLK\_posedge) begin

mic\_data\_buffer[0] <= {MIC\_DATA, 15'd0};

mic\_data\_buffer[1] <= mic\_data\_buffer[0];

mic\_data\_buffer[2] <= mic\_data\_buffer[1];

mic\_data\_buffer[3] <= mic\_data\_buffer[2];

end

end

// FIR滤波：简单的加权平均

assign filtered\_data = (mic\_data\_buffer[0] + mic\_data\_buffer[1] + mic\_data\_buffer[2] + mic\_data\_buffer[3]) >> 2;

由于我们使用了FIR滤波器，这里不能直接使用强度来复制给LED了，所以这里我们换源，用滤波后的数据赋值即可。

always @(posedge clk or posedge rst) begin

if(rst) begin

led\_threshold <= 16'h0;

MIC\_LR\_SEL <= 1'b0;

end

else begin

led\_threshold <= filtered\_data[7:0]; // 使用新的滤波数据低8位作LED阈值

end

end

此外在最后我们的控制逻辑也需要修改，为了让整个流程不冲突（这里直接使用会发现enable和开关是冲突的）在最后AUDIO控制信号部分我们给出这样的定义

always @(posedge clk or posedge rst) begin

if(rst) begin

AUD\_SD <= 1'b0;

AUD\_PWM <= 1'b0;

end

else if(enable[0])begin

AUD\_SD <= enable[0];

AUD\_PWM <= led;

end

else begin

AUD\_SD <= sd\_sw;

AUD\_PWM <= led;

end

end

这里我们设计好以我们的AXI控制为高控制权，开关仅仅在没有主动通过软件控制的时候起作用。

在修改完控制器具体实现后，在顶层文件中实例化我们的控制模块

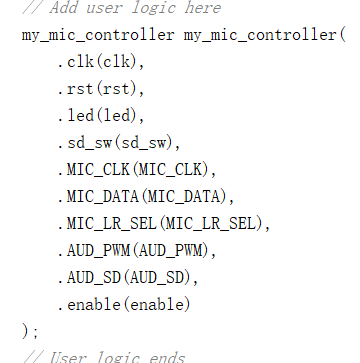


图2.3 顶层文件实例化

加入好的块设计如下图：

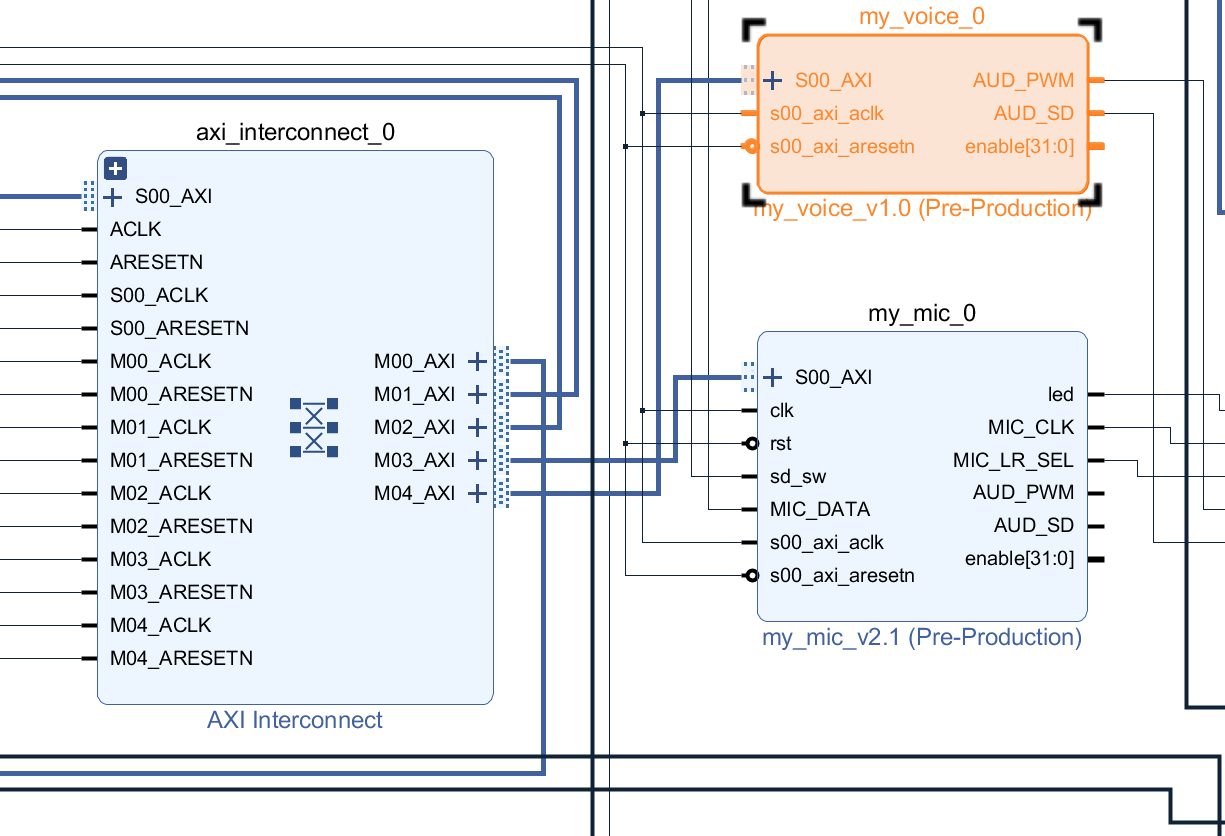


图2.4 块设计示意图

在rvfpga顶层文件中作出修改，我们调用的几个端口和初步实现的一致。

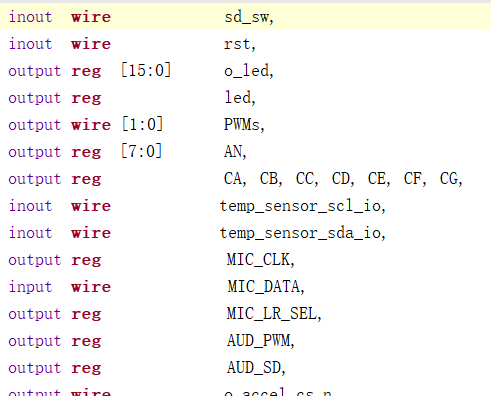


图2.5 rvfpga顶层文件修改

### 声音方面RVfpga探索

这里可以发现如果没有输入源，其实RVfpga可以实现主动的pwm，正如我们所做的pwm实验当中可以主动生成一个pwm信号，当然可以发现通过主动生成pwm信号也是可以转化成一个可以发声的信号源，这里我简单实现了一下但是并不完整。

首先我们需要设计好我们的音色的值，这个值的计算基本上是通过**十二平均律** 的理论来计算的，将一个八度的音频范围均匀分成12个音符的标准调律方式，我们有频率为440 Hz的音符为A4，基于这个基准我们就能了解到其他的音符是如何计算的。公式是所需时钟周期数=音符周期除以每个时钟周期的时间。

`define C5 523 // 中音C（523 Hz）

`define G5 784 // 高音G（784 Hz）

`define A5 880 // 高音A（880 Hz）

`define F5 698 // 高音F（698 Hz）

`define E5 659 // 中音E（659 Hz）

`define D5 587 // 中音D（587 Hz）

那么用100mhz系统时钟的时候，我们可以定义好每个音符的周期

`define C5\_PERIOD 191000 // 中音C周期

`define G5\_PERIOD 127500 // 高音G周期

`define A5\_PERIOD 113640 // 高音A周期

`define F5\_PERIOD 143160 // 高音F周期

`define E5\_PERIOD 151200 // 中音E周期

`define D5\_PERIOD 170010 // 中音D周期

我们在累加通过计数器，根据音符的周期计算，控制PWM输出（这个计算方案和前面的方式一样）

always@(posedge clk or negedge rst) begin

if(!rst)

AUD\_PWM <= 0;

else

AUD\_PWM <= (cnt < cnt\_top / 2) ? 1 : 0; //前半段高电平，后半段低电平

end

最后通过enable传值得到AUD\_SD的控制端。只保留了《小星星》旋律中的一小段音符，长度较短。

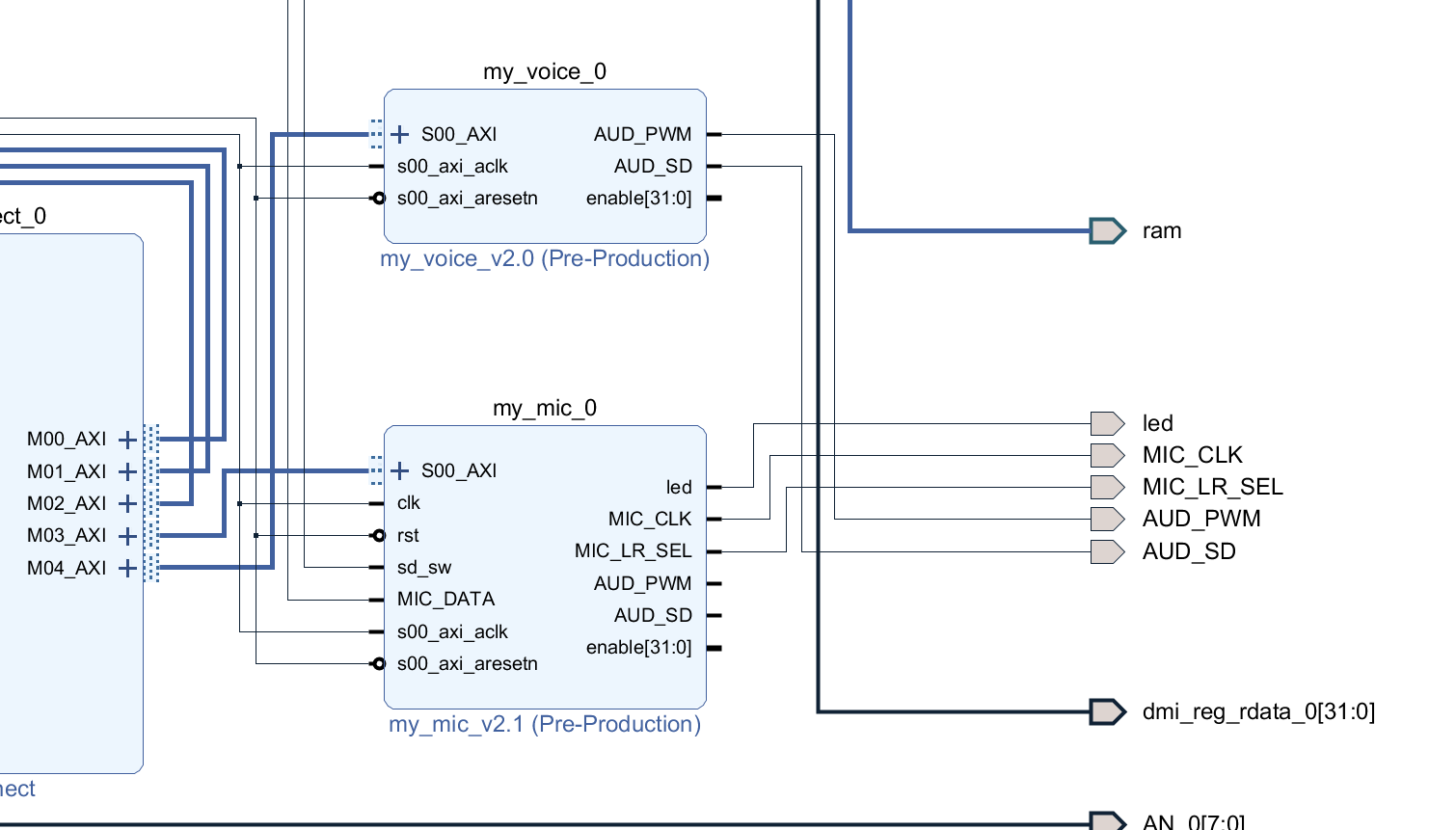


图2.6 IP总图

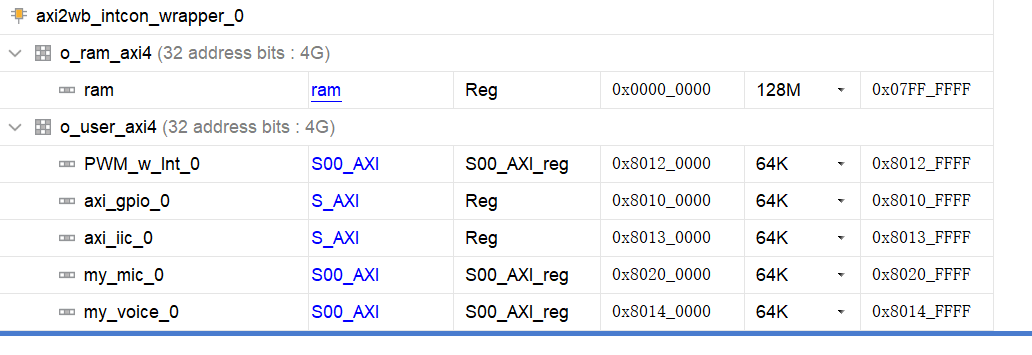


图2.7 地址分配总图

给这个模块地址为0x80140000，在platformIO中只需要做到在给这个地址使能值即可。

理论上是可以实现的但是有些问题，听不到声音，有可能是因为define音符的值有些问题。

### PlatformIO设计

#define VOICE 0x80140000

#define LISTEN 0x80200000

只需要在需要的时候给出赋值即可。

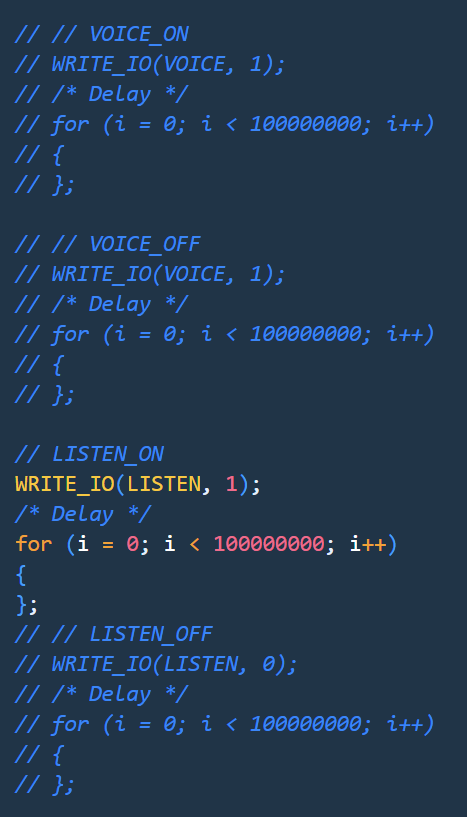


图2.8 PlatformIO 简单控制

# 总结与心得

## 实验总结

通过本次实验，成功设计并实现了一个基于RVfpga\_SoC处理器的音频输入输出接口系统。该系统能够实现音频信号的采集和播放，具有一定的实用价值。通过实验，加深了对FPGA和接口技术的理解，提高了硬件设计和调试能力。

## 实验心得

在整个实验的过程中，我对于fpga本身有了更加深刻的认识，尤其是现在作为一个迟交报告的学生，从一个初步实现的demo到能够将一个实现的简单项目移植，再到修改接入总线，其实还是挺困难的，说实话我并不觉得这是一个简单的问题，还是需要理解，学习很多东西的（尤其是项目中PDM转PWM的处理方式实在有点难，不过其他很多地方可以查阅手册解决），能够实现一个很简单的功能我已经很高兴了。

在做前面的单独的实验，我遇到一些难以理解的问题，也在老师的帮助寻找下发现了PlatformIO平台下编译的C程序也不是随心所欲的，在程序的宏定义方面有些限制，对于定义好的宏也不是随便使用，这也教会了我寻找问题根源的方法，如何通过控制变量的方式去寻找导致问题的办法，也许有很多问题结合导致了问题的产生。换板、换比特流、换机器到最后确定是程序的问题，还是花了大半节课寻找这样的问题，还是很有收获的。

在整个接口的实验过程中，我了解了很多没有了解到的知识，接口还是与我们的生活息息相关，了解过很多名词，不管是英特尔做的雷电接口还是统一规范过后的USB，学习这门课让我对这些名词也有了一些可能的印象，毕竟没有教学，但是却能够让我对接口有一个基本的认识。

使用vivado并不是很容易，即使我在上个学期选修过verilog，使用过vivado，但是RVfpga完全是一个崭新的领域，我感觉我还没有怎么了解就做实验其实也算是稀里糊涂开始做了，但是具体实验难度并没有那么大，通过实验的简单实现我也是对这些概念有了更加深刻的认识。

无论如何，从现在来看还是很有意思的一门课，我在网络上寻找到不少人使用其他板子做的项目，我也学习了他们的代码，看看他人的实现，还是很有趣的。

## 意见和建议

我觉得最好能够实现新的vivado适配（虽然好像不太好做）。

PlatformIO的搭建问题，最好能够有一个错误的QA这样大家碰到问题的时候就不会抓耳挠腮。

# 致谢

特别感谢Imagination Community对本次实验中的支持和贡献，正因为有这些业界公司无私的教学支持才能让我们这些学生在学习中走的更远。

|  |
| --- |
| 一、原创性声明 |
| 本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。  特此声明！  **作者签字:** |
| 二、对课程实验的学术评语（教师填写） |
|  |
| 三、对课程实验的评分（教师填写） |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | 评分项目  （分值） | 工具应用  （10分） | 实验过程  （70分） | 报告撰写  （20分） | 最终评定  （100分） | | 得分 |  |  |  |  | |
| **指导教师签字:** |