基于 ES8388 的音频回环实验

章节导读

ES8388 是一种高性能、低功耗、低成本的音频编解码器。它由两路 ADC，2 通道 DAC，话筒放大器、耳机放大器、数字音效、模拟混合和增益功能。ES8388 采用先进的多位 Δ∑调制技术实现数字与模拟之间的数据转换。多比特 Δ∑调制器使器件对时钟抖动和低带外噪声的灵敏度低。它应用于：MID，MP3， MP4，PMP，无线音频，数码相机，摄像机，GPS 领域，蓝牙，便携式音频设备。

本章将会从 ES8388 音频编码器讲起，并逐步介绍编码器的工作原理和配置方法。最后通过线路输入，将手机或者电脑输出的音频数据输出到 FPGA 中，经过音频回环后由耳机输出。

1.1 ES8388 音频编码器

ES8388 是由上海顺芯推出的一款高性能、低功耗、高性价比的音频编解码器，有 2 个 ADC 通道和 2 个 DAC 通道，麦克风放大器，耳机放大器，数字音效以及模拟混合和增益功能组成。

ES8388 的主要特性有：

- I2S 接口，支持最高 192k，24bit 音频播放
- DAC 信噪比 96dB；ADC 信噪比 95dB
- 支持主机和从机模式
- 支持立体声差分输入/麦克风输入
- 支持左右声道音量独立调节
- 支持 40mW 耳机输出，无爆音

ES8388 的控制通过 I2S 接口（即数字音频接口）同 FPGA 进行音频数据传输（支持音频接收和发送），通过 I2C 对寄存器进行配置。ES8388 的 I2S 接口，由 4 个引脚组成：

ASDOUT：ADC 数据输出

DSDIN：DAC 数据输入

LRC：数据左/右对齐时钟

SCLK：位时钟，用于同步

ES8388 可作为主机，输出 LRC 和 SCLK 时钟，也可以作为从机，接收 LRC 和 SCLK，本次实验，我们将 ES8388 作为主机。另外，ES8388 的 I2S 接口支持 4 种不同的音频数据模式：左（MSB）对齐模式、右（LSB）对齐模式、飞利浦（I2S）标准，我们这里将将使用飞利浦标准模式来传输 I2S 数据。

飞利浦（I2S）标准模式，数据在跟随 LRC 传输的 BCLK 的第二个上升沿传输 MSB，其它位一直到 LSB 按顺序传输。传输依赖于字长、BCLK 频率和采样率，在每个采样的 LSB 和下一个采样的 MSB 之间都应该有未用的 BCLK 周期。飞利浦标准模式的 I2S 数据传输协议如下所示：

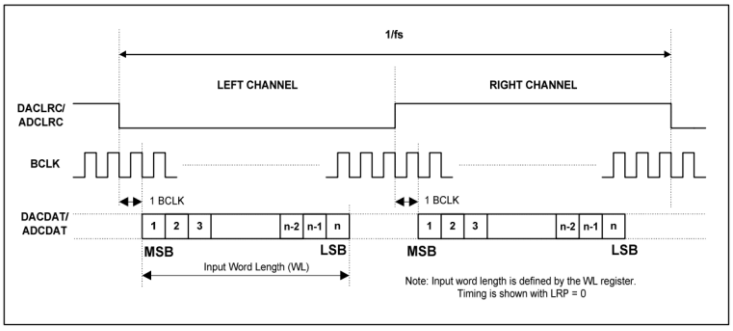

图 1-1 飞利浦标准模式 I2S 数据传输图

图中，fs 即音频信号的采样率，因此可以知道，LRC 的频率就是音频信号的采样率。另外 ES8388 还需要一个 MCLK，我们通过 PLL 产生之后提供给 ES8388 芯片。

ES8388 的整体结构如下图 1-2 所示。

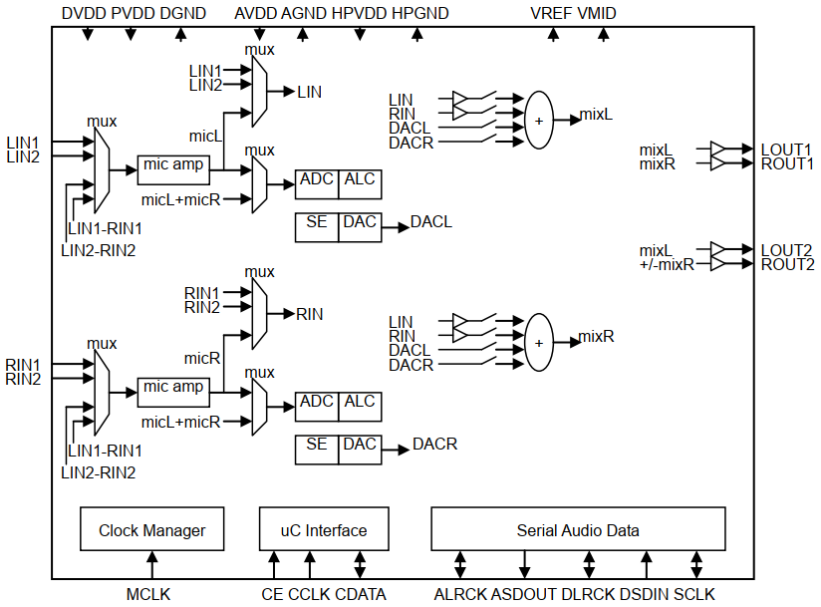

图 1-2 ES8388 框图

从上图可以看出，ES8388 内部有很多模拟开关，用来选择通道，同时还有一些运放调节器，用来设置增益和音量，这些都是通过配置相关寄存器来实现的。

1.1.1 模拟信号输入路径

ES8388 有两个灵活的立体声模拟输入通道，可以设置为线路输入、差分麦克风输入或单端麦克风输入。开发板中 ES8388 的硬件电路如下图 1-3 所示。

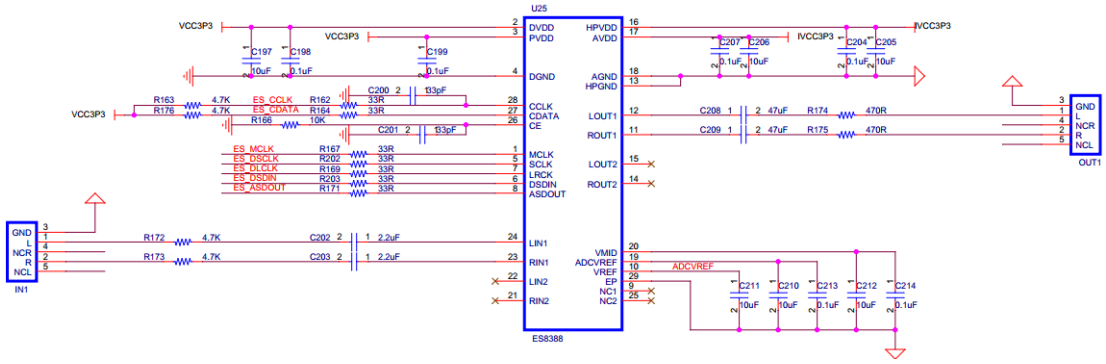

图 1-3 音频编解码硬件电路

从上图中，可以看出我们使用的是线路输入，左右声道的输入分别连接在了 LIN1 和 RIN1 上，输出分别连接在 LOUT1 和 ROUT1 上，根据原理图中设置的通道，然后去配置相关寄存器选择对应的输入\输出通道。

1.1.2 ES8388 寄存器说明

ES8388 支持通过 SPI 或者 I2C 去配置，但是根据我们的原理图来看，我们只能通过 I2C 去配置寄存器，配置时，首先需要知道对应的器件地址，ES8388 的器件地址为 0x20，下面将对部分寄存器进行说明。

1.1.2.1 寄存器 R3(03h)

ADC 电源管理控制寄存器，对于该寄存器的说明如下所示。

表 1-1 寄存器 03h 说明表

Bit name	Bit	Description
PdnAINL	7	0 – normal 1 – left analog input power down (default)
PdnAINR	6	0 – normal 1 – right analog input power down (default)
PdnADCL	5	0 – left ADC power up 1 – left ADC power down (default)

PdnADCR	4	0 – right ADC power up 1 – right ADC power down (default)
PdnMICB	3	0 – microphone bias power on 1 – microphone bias power down (high impedance output, default)
PdnADCBiasgen	2	0 – normal 1 – power down (default)
flashLP	1	0 – normal (default) 1 – flash ADC low power
int1LP	0	0 – normal (default) 1 – int1 low power
Bit name	Bit	Description
PdnAINL	7	0 – normal 1 – left analog input power down (default)
PdnAINR	6	0 – normal 1 – right analog input power down (default)
PdnADCL	5	0 – left ADC power up 1 – left ADC power down (default)
PdnADCR	4	0 – right ADC power up 1 – right ADC power down (default)
PdnMICB	3	0 – microphone bias power on 1 – microphone bias power down (high impedance output, default)
PdnADCBiasgen	2	0 – normal 1 – power down (default)
flashLP	1	0 – normal (default) 1 – flash ADC low power
int1LP	0	0 – normal (default) 1 – int1 low power

PdnAINL(bit7)和 PdnAINR(bit6)用于控制左右输入模拟通道的电源， 1 掉电，0 正常； PdnADCL(bit5)和 PdnADCR(bit4)用于控制左右通道 ADC 的电源， 1 掉电， 0 正常；我们必须保证这四位必须为 0，这样 ADC 部分才能开始输入的音频信号。

1.1.2.2 寄存器 04(04h)

DAC 电源管理控制寄存器，对于该寄存器的说明如下所示。

表 1-2 寄存器 04 说明表

Bit name	Bit	Description
PdnDACL	7	0 – left DAC power up 1 – left DAC power down (default)
PdnDACR	6	0 – right DAC power up 1 – right DAC power down (default)
LOUT1	5	0 – LOUT1 disabled (default) 1 – LOUT1 enabled
ROUT1	4	0 – ROUT1 disabled (default) 1 – ROUT1 enabled
LOUT2	3	0 – LOUT2 disabled (default) 1 – LOUT2 enabled

ROUT2	2	0 – ROUT2 disabled (default) 1 – ROUT2 enabled

PdnDACL(bit7)和 PdnDACR(bit6)分别用于左右声道 DAC 的电源控制， 1 掉电； 0 正常； LOUT1(bit5)和 ROUT1(bit4)分别用于控制通道 1 的左右声道输出是能， 1 使能， 0 禁止； LOUT2(bit3)和 ROUT2(bit2)分别用于控制通道 2 的左右声道输出是能， 1 使能， 0 禁止；根据原理图，我们可以看出，使用的是 LOUT1 和 ROUT1，所以我们需要将对应位置 1，所以需要向 03h 中写入 0x30。

1.1.2.3 寄存器 08(08h)

主模式控制器，对该寄存器的说明如下所示。

表 1-3 寄存器 08 说明表

Bit name	Bit	Description
MSC	7	0 – slave serial port mode 1 – master serial port mode (default)
MCLKDIV2	6	0 – MCLK not divide (default) 1 – MCLK divide by 2
BCLK_INV	5	0 – normal (default) 1 – BCLK inverted
BCLKDIV	4:0	00000 – master mode BCLK generated automatically based on the clock table (default) 00001 – MCLK/1　00010 – MCLK/2　00011 – MCLK/3 00100 – MCLK/4　00101 – MCLK/6　00110 – MCLK/8 00111 – MCLK/9　01000 – MCLK/11　01001 – MCLK/12 01010 – MCLK/16　01011 – MCLK/18　01100 – MCLK/22 01101 – MCLK/24　01110 – MCLK/33　01111 – MCLK/36 10000 – MCLK/44　10001 – MCLK/48　10010 – MCLK/66 10011 – MCLK/72　10100 – MCLK/5　10101 – MCLK/10 10110 – MCLK/15　10111 – MCLK/17　11000 – MCLK/20 11001 – MCLK/25　11010 – MCLK/30　11011 – MCLK/32 11100 – MCLK/34　Others – MCLK/4

MSC(bit7)用于控制接口模式，0 从模式，1 主模式；MCKDIV2(bit6)用于控制 MCLK 的 2 分频，0 不分频，1 二分频；BCLK_INV(bit5)用于控制 BCLK 的反相，0 不反相；1，反相；BCLKDIV(bit4~bit0)为主模式下，MCLK 和 BCLK 的比例，本次实验我们使用 ES8388 的主模式，bit7 设置为 1，BCLKDIV(bit4~bit0)设置为 00100，通过该寄存器就能设置 ES8388 输出的 BCLK 的时钟，BCLK 的时钟还需要根据 ADC 和 DAC 的采样位数进行判断是否满足要求，具体将会在下文中进行说明。

1.1.2.4 寄存器 10(0Ah)

ADC 控制器 2，对该寄存器的说明如下所示。

表 1-4 寄存器 10 说明表

Bit name	Bit	Description
LINSEL	7:6	Left channel input select 00 – LINPUT1 (default) 01 – LINPUT2 10 – reserved 11 – L-R differential (either LINPUT1-RINPUT1 or LINPUT2-RINPUT2, selected by DS)
RINSEL	5:4	Right channel input select 00 – RINPUT1 (default) 01 – RINPUT2 10 – reserved 11 – L-R differential (either LINPUT1-RINPUT1 or LINPUT2-RINPUT2, selected by DS)
DSSEL	3	0 – use one DS Reg11[7] (default) 1 – DSL=Reg11[7], DSR=Reg10[2]
DSR	2	Differential input select 0 – LINPUT1-RINPUT1 (default) 1 – LINPUT2-RINPUT2

需要用到的位有： LINSE(bit7:6)和 RINSE(bit5:4)分别选择左右输入通道，0 选择通道 1，1 选择通道 2，我们使用的通道 1，所以都设置为 0。

1.1.2.5 寄存器 12(0Ch)

ADC 控制器 4，对于该寄存器的说明如下所示。

表 1-5 寄存器 12 说明表

Bit name	Bit	Description
DATSEL	7:6	00 – left data = left ADC, right data = right ADC 01 – left data = left ADC, right data = left ADC 10 – left data = right ADC, right data = right ADC 11 – left data = right ADC, right data = left ADC
ADCLRP	5	I2S, left justified or right justified mode: 0 – left and right normal polarity 1 – left and right inverted polarity DSP/PCM mode: 0 – MSB is available on 2nd BCLK rising edge after ALRCK rising edge 1 – MSB is available on 1st BCLK rising edge after ALRCK rising edge
ADCWL	4:2	000 – 24-bit serial audio data word length 001 – 20-bit serial audio data word length 010 – 18-bit serial audio data word length 011 – 16-bit serial audio data word length 100 – 32-bit serial audio data word length
ADCWL	1:0	00 – I2S serial audio data format 01 – left justify serial audio data format 10 – right justify serial audio data format 11 – DSP/PCM mode serial audio data format

DATSEL(bit7:6)用于选择数据格式，设置为 00，左右边数据等于左右声道 ADC 数据；ADCLRP（bit5）在 I2S 模式下用于设置数据对其方式，一般设置为

0，正常极性； ADCWL(bit4:2)用于选择数据长度，我们设置 011，选择 16 位数据长度； ADCFORMAT(bit1:0)用于设置 ADC 数据格式，设置为 00，选择 I2S 数据格式。

1.1.2.6 寄存器 13(0Dh)

ADC 控制寄存器 5，对于该寄存器的说明如下。

表 1-6 寄存器 13 说明

Bit name	Bit	Description
ADCFsMode	5	0 – single speed mode (default) 1 – double speed mode
ADCFsRatio	4:0	Master mode ADC MCLK to sampling frequency ratio 00000 – 128　00001 – 192　00010 – 256 00011 – 384　00100 – 512　00101 – 576 00110 – 768 (default)　00111 – 1024 01000 – 1152　01001 – 1408　01010 – 1536 01011 – 2112　01100 – 2304　10000 – 125 10001 – 136　10010 – 250　10011 – 272 10100 – 375　10101 – 500　10110 – 544 10111 – 750　11000 – 1000　11001 – 1088 11010 – 1496　11011 – 1500 Other – reserved

ADCFsMode(bit7)用于设置 Fs 模式，0 单速模式，1 双倍速模式，一般设置为 0；ADCFsRatio(bit4:0)用于设置 ADC 的 MCLK 和 FS 的比率，我们设置 00000，即 128 倍关系，本次实验我们使用 PLL 产生 12.288M 的时钟给到 MCLK，这样得到 LRCK 为 12.288M/128=96K，ADC 和 DAC 的位数为 16，那么 BCLK 需要至少 96K*32（16*2）＝3.072M，前面寄存器 08 就是控制的 BCLK 的输出，所以我们在设置的时候，需要注意是否满足该要求，所以本次实验中将寄存器 08 的 bit[4:0]设置为 00100，此时 BCLK 就为 12.288M/4=3.072M，满足要求。

1.1.2.7 寄存器 23(17h)

DAC 控制寄存器 1，对于该寄存器的说明如下所示。

表 1-7 寄存器 23 说明

Bit name	Bit	Description
DACLRSWAP	7	0 – normal 1 – left and right channel data swap
DACLRP	6	I2S, left justified or right justified mode: 0 – left and right normal polarity 1 – left and right inverted polarity DSP/PCM mode: 0 – MSB is available on 2nd BCLK rising edge after ALRCK rising edge

		1 – MSB is available on 1st BCLK rising edge after ALRCK rising edgeLRCK Polarity
DACWL	5:3	000 – 24-bit serial audio data word length 001 – 20-bit serial audio data word length 010 – 18-bit serial audio data word length 011 – 16-bit serial audio data word length 100 – 32-bit serial audio data word length
DACFORMAT	2:1	00 – I2S serial audio data format 01 – left justify serial audio data format 10 – right justify serial audio data format 11 – DSP/PCM mode serial audio data format

DACLRSWAP(bit7)用于控制左右声道数据交换，0 正常，1 互换，一般设置为 0；DACLRP(bit6) 在 I2S 模式下用于设置数据对其方式，一般设置为 0，正常极性；DACWL(bit5:3)用于选择数据长度，我们设置 011，选择 16 位数据长度；ADCFORMAT(bit1:0)用于设置 DAC 数据格式，一般设置为 00，选择 I2S 数据格式。

1.1.2.8 寄存器 43(2Bh)

DAC 控制器 21，对该寄存器的说明如下。

表 1-8 寄存器 43 说明表

Bit name	Bit	Description
slrck	7	0 – DACLRC and ADCLRC separate (default) 1 – DACLRC and ADCLRC same
lrck_sel	6	Master mode, if slrck = 1 then 0 – use DAC LRCK (default) 1 – use ADC LRCK
offset_dis	5	0 – disable offset (default) 1 – enable offset
mclk_dis	4	0 – normal (default) 1 – disable MCLK input from PAD
adc_dll_pwd	3	0 – normal (default) 1 – ADC DLL power down
dac_dll_pwd	2	0 – normal (default) 1 – DAC DLL power down

这里我们只关心 slrck(bit7)这个位，用于控制 DACLRC 和 ADCLRC 是否共用，我们设置为 1，表示共用。

通过前面章节，我们对 ES8388 的部分寄存器设置进行了简要说明，ES8388 的配置还涉及到很多别的寄存器，详细的内容请自行查看 ES8388 的数据手册。

1.2 系统整体设计

本次实验最终需要实现的功能是通过手机或者电脑输出音频，传输至开发

板上的音频输入接口，经过转换之后，再由音频输出口进行输出，最终我们可以通过耳机听到手机或电脑传输过来的音频数据，系统的整体设计如下所示。

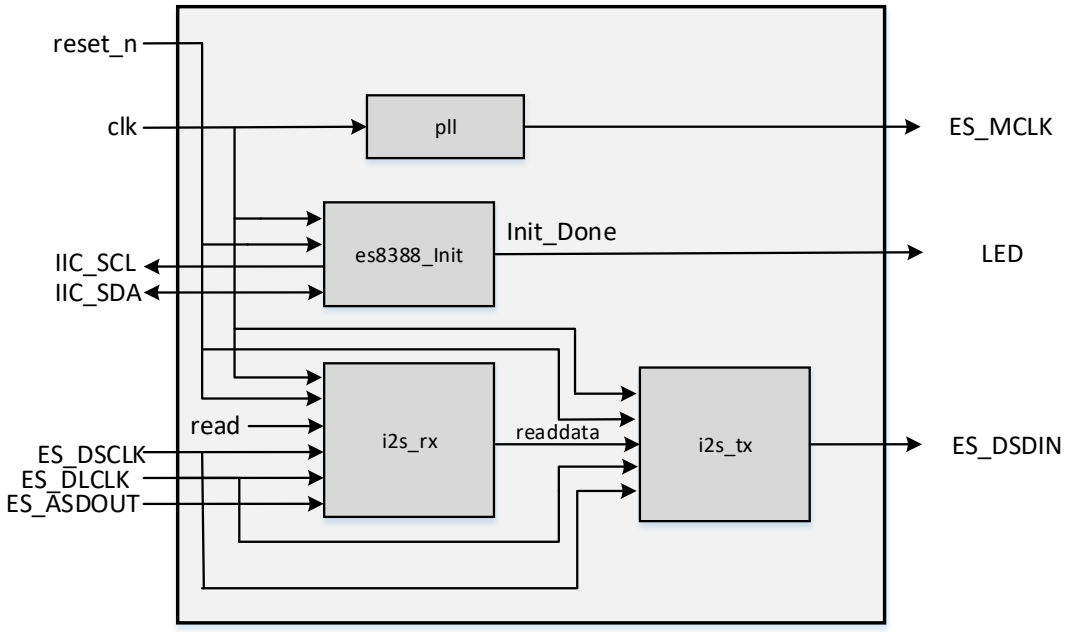

图 1-4 系统整体设计图

1.3 模块设计

下面将对本次实验需要设计的模块进行简要介绍。

1.3.1 PLL IP

我们在前面对 ES8388 音频编码器进行介绍的时候说过，需要通过 FPGA 输出一个 12.288MHz 的时候给到 ES8388，所以我们需要添加一个 PLL IP 核，该 IP 核的输入由开发板上的晶振提供为 50MHz，输出 12.288MHz 的时钟，PLL 的 IP 的配置如下所示。

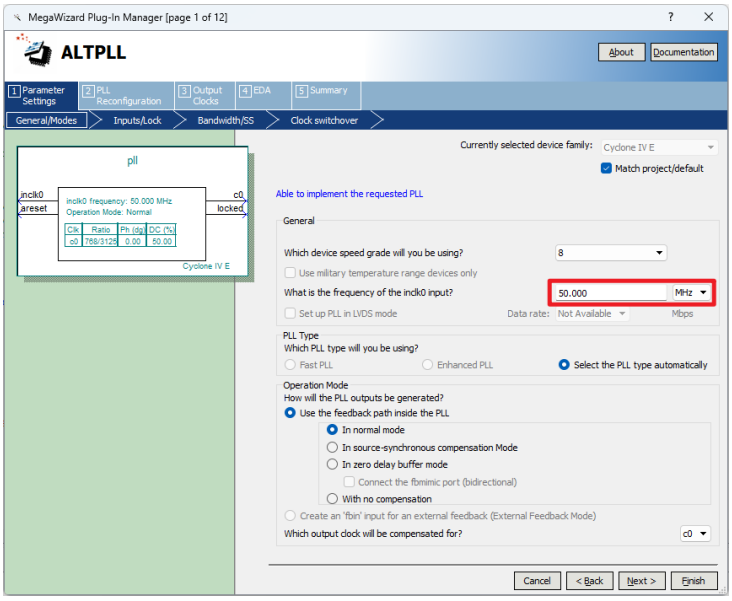

图 1-5PLL IP 配置界面 1

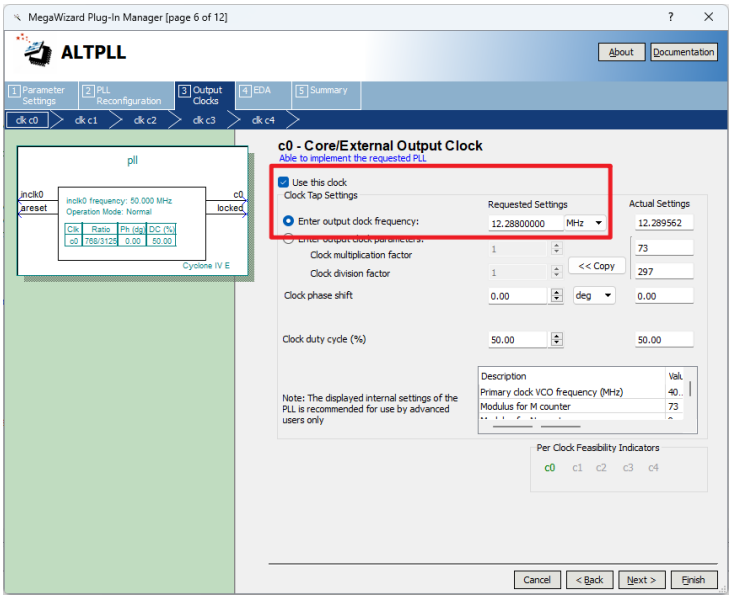

图 1-6 PLL 配置界面 2

1.3.2 es8388_Init

es8388_Init 模块的功能就是通过 I2C 初始化 ES8388 音频芯片，es8388_Init 模块的基本结构如下所示。

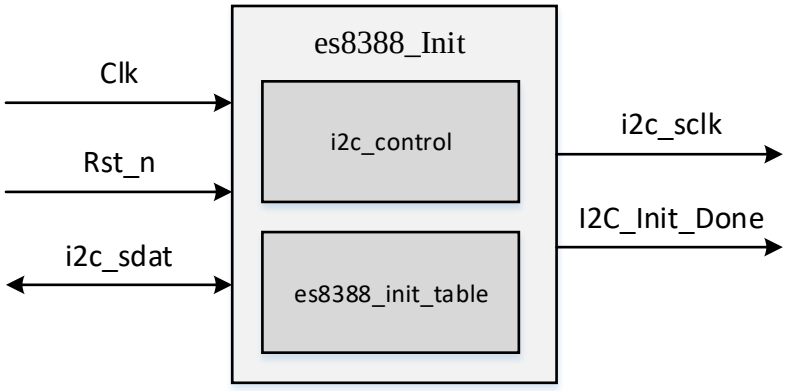

图 1-7 es8388_Init 模块基本框图

对上述模块的信号说明如下所示。

表 1-9 es8388_Init 模块信号说明表

信号名称	I/O	信号意义
Clk	I	模块工作时钟，50M
Rst_n	I	模块复位信号，低电平有效
I2C_Init_Done	O	初始化完成标志信号
i2c_sclk	O	I2C 时钟信号线
i2c_sdat	IO	I2C 数据信号线

es8388_Init 模块最主要的就是 I2C 控制模块 i2c_control 和 ES8388 寄存器配置模块 es8388_init_table，I2C 控制模块有专门的章节介绍过，这里将不再对其进行详细说明， es8388_init_table 模块的实现代码如下所示：

```
module es8388_init_table
#(parameter DATA_WIDTH=16, parameter ADDR_WIDTH=8)
(
    input [(ADDR_WIDTH-1):0] addr,
    input clk,
    output reg [(DATA_WIDTH-1):0] q,
    output [7:0]dev_id,
    output [7:0]lut_size
);

    reg [DATA_WIDTH-1:0] rom[2**ADDR_WIDTH-1:0];

    assign dev_id = 8'h20;      //es8388 IIC 接口器件地址
    assign lut_size = 8'd43;    //es8388 寄存器初始化数量

    //MCLK 时钟为 12.288M
    //Line IN
    always @ (*) begin
    rom[0 ] = 16'h00_80;       /* 软复位 ES8388 */
    rom[1 ] = 16'h00_16;
```

```
    rom[2 ] = 16'h01_58;
    rom[3 ] = 16'h01_50;
    rom[4 ] = 16'h02_F3;
    rom[5 ] = 16'h02_F0;
    rom[6 ] = 16'h2B_80; //ADC 和 DAC 使用相同的 LRCK bit[7] 为 1
    rom[7 ] = 16'h00_36;
    rom[8 ] = 16'h08_00; //主模式控制寄存器：bit[7]：1：主机模式,控制
MCLK/SCLK 的比例 bit【4：0】控制 SCLK 为 2.0148M
    rom[9 ] = 16'h03_09;//<-
    rom[10] = 16'h04_00;
    rom[11] = 16'h0D_02; //配置 MCLK 和频率的比率       ADCLRCK=MCLK/对应
比例（该寄存器配置[4：0]）512 16K
    rom[12] = 16'h18_02;
    rom[13] = 16'h05_00;
    rom[14] = 16'h06_C3;
    rom[15] = 16'h0A_00; //Select Analog input channel for ADC
(Lin1/Rin1)    LIN1:0X00                LIN2：0x52
    rom[16] = 16'h0B_02; //(Select LIN1and RIN1 as differential input
pairs)  LIN1:0X02    LIN2:0x82
    rom[17] = 16'h0C_0c; //ADC Control: [1:0]=00(I2S 模式); [4:2]:
011: 16bit(0x0c); 000:24(0x00); 001:20(0x04); 010:18(0x08);
100:32(0x10)
    rom[18] = 16'h17_18; //DAC Control: [2:1]=00(I2S);     [5:3]: 011:
16bit(0x18); 000:24(0x00); 001:20(0x08); 010:18(0x10); 100:32(0x20)
    rom[19] = 16'h10_00;
    rom[20] = 16'h11_00;
    rom[21] = 16'h1A_00;
    rom[22] = 16'h1B_00;
    rom[23] = 16'h09_88;//配置增益 L/R PGA 增益为+24b,adc 的数据选择为
left data = left adc 音频数据为 16bit
    rom[24] = 16'h12_11;//关闭 ALC
    rom[25] = 16'h13_C0;
    rom[26] = 16'h14_32;
    rom[27] = 16'h15_06;
    rom[28] = 16'h16_C3;
    rom[29] = 16'h27_B8;
    rom[30] = 16'h2A_B8;
    rom[31] = 16'h02_00;        //后面需要延时 500ms
    rom[32] = 16'h2E_1E;
    rom[33] = 16'h2F_1E;
    rom[34] = 16'h30_1E;
    rom[35] = 16'h31_1E;
    rom[36] = 16'h04_36;    //0x30:使用 OUT1 [4]:ROUT1 enable;
[5]:LOUT1 enable; 0x06:使用 OUT2 [2]:ROUT2 enable;[3]:LOUT2 enable
    rom[37] = 16'h26_00;
    rom[38] = 16'h03_09;
    rom[39] = 16'h2E_1E;
```

```
    rom[40] = 16'h2F_1E;
    rom[41] = 16'h30_1E;
    rom[42] = 16'h31_1E;
    end
    always @ (posedge clk)
    begin
        q <= rom[addr];
    end

endmodule
```

将上诉寄存表通过 I2C 控制模块进行配置，完整的代码请读者自行查看对应的工程。详细的寄存器配置含义请查看 ES8388 数据手册。

1.3.3 i2s_rx

I2S 接收模块，主要功能是将音频芯片内部 ADC 采集得到的串行数据解析出来存储至 FIFO 中，该模块的基本框图如下所示。

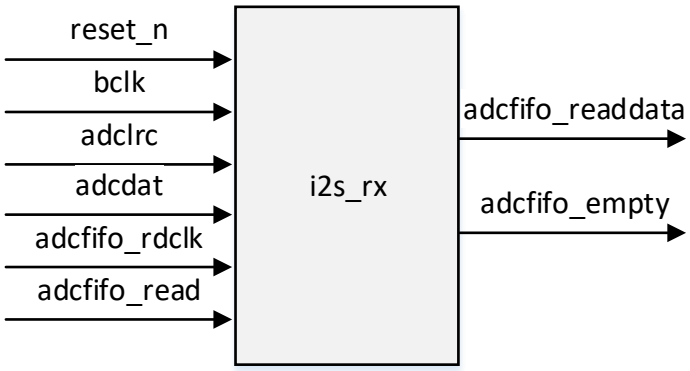

图 1-8 i2s_rx 模块基本框图

对上述模块的信号说明如下所示。

表 1-10 i2s_rx 信号说明表

信号名称	I/O	信号意义
reset_n	I	复位信号，低电平有效
bclk	I	音频接口的位时钟信号
adclrc	I	音频接口 ADC 的左/右时钟信号
adcdat	I	ADC 数字音频数据信号
adcfifo_rdclk	I	FIFO 的读时钟信号
adcfifo_read	I	FIFO 的读使能信号
adcfifo_empty	O	FIFO 的空标志信号
adcfifo_readdata	O	FIFO 读出的数据信号

我们在编写该模块之前，通过 ES8388 的数据手册可以得知，当使用 I2S 传输数据时，其时序图如下所示。

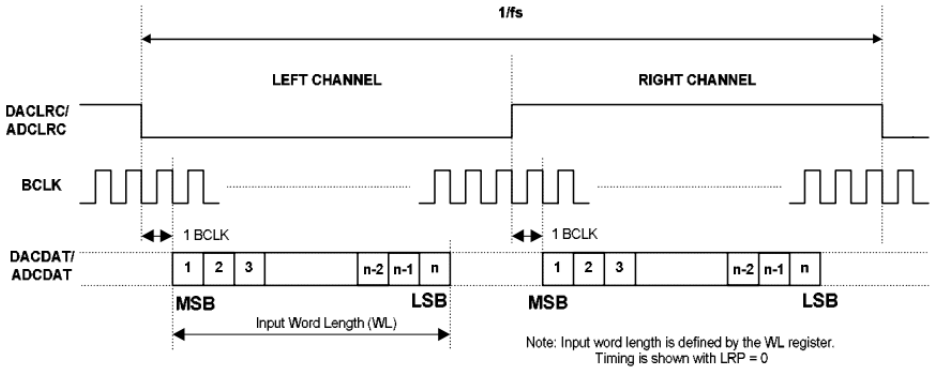

图 1-9 ES8388 I2S 传输数据时序图

本模块主要是针对 ADC 来说，当 ADCLRC 为低电平时，传输 LEFT CHANNEL 的数据，当 ADCLRC 为高电平时，传输 RIGHT CHANNEL 的数据，并且传输的数据和 ADCLRC 的边沿间隔一个时钟周期，先传输高位数据然后传输低位数据，根据上述时序图我们便可以编写本模块的代码。

首先，我们需要获得 ADCLRC 信号的上升沿和下降沿信号，将 ADCLRC 信号根据 BCLK 的时钟打两拍得到 adclrc_r0 和 adclrc_r1 信号，当 adclrc_r0 为低电平、adclrc_r1 为高电平时，说明检测到了 ADCLRC 信号的下降沿，当 adclrc_r0 为高电平、adclrc_r1 为低电平时，说明检测到了 ADCLRC 信号的上升沿，寄存 ADCLRC 信号的时候，同时将 ADCDAT 打两拍，这样在根据边沿信号解数据时才不会出现错位的情况，代码如下所示：

```
reg adclrc_nege;
reg adclrc_pose;
reg adclrc_r0;
reg adclrc_r1;
reg adcdat_r0;
reg adcdat_r1;
always @(posedge bclk) begin
    adclrc_r0 <= adclrc;
    adclrc_r1 <= adclrc_r0;
    adcdat_r0 <= adcdat;
    adcdat_r1 <= adcdat_r0;
end
always@(posedge bclk or negedge reset_n)
if(~reset_n) begin
    adclrc_nege <= 1'd0;
    adclrc_pose <= 1'd0;
end
else begin
    adclrc_nege <= adclrc_r1 & (!adclrc_r0);
    adclrc_pose <= (!adclrc_r1) & adclrc_r0;
```

```
end
```

然后根据边沿信号，依次将左通道和有通道的数据寄存器起来，存储至 FIFO 中，这里我们使用状态机实现，定义四个状态，分别是空闲状态、采集左通道数据状态、采集右通道数据状态、FIFO 的写数据状态，如下所示：

```
parameter state_idle        = 2'd0;   //空闲状态
parameter state_left_data   = 2'd1;   //采集左通道数据
parameter state_right_data  = 2'd2;   //采集右通道数据
parameter state_fifo_write  = 2'd3;   //FIFO 的写数据状态
```

当处于空闲状态的时候，当检测到了 ADCLRC 的下降沿的时候，进入左通道数据采集状态，并且由于先传输的是高位数据，所以我们在寄存的时候，需要将先接收到的数据放至最高位，我们这里设立 bit_cnt，用该值来表示左右两个通道每次传输一次的数据个数，比如本次设计我们设置 ES8388 寄存器时，使其左右两个通道每次传输 16 位的数据，两个通道加起来就是 32 位（0~31）的数据，这里通过 parameter DATA_WIDTH 来进行定义，当检测到了 ADCLRC 的下降沿的时候，还需要使 bit_cnt 等于 DATA_WIDTH – 1，代码如下所示：

```
state_idle:
begin
  adcfifo_write <= 1'd0;
  if(adclrc_nege)
  begin
      bit_cnt <= DATA_WIDTH - 1;
      state <= state_left_data;
  end
end
```

进入左通道数据采集状态时，将 bit_cnt 的值依次减 1，并将 ADCDAT 上的值放至 reg_wrfifo_data 对应的 bit_cnt 位上，当 bit_cnt 的值等于 DATA_WIDTH/2 – 1 的时候，表示此时左通道上的数据采集完成，跳转至采集右通道数据状态，代码如下所示：

```
state_left_data:
begin
    if(bit_cnt == (DATA_WIDTH/2 - 1)) //左通道数据采集完成
    begin
        if(adclrc_pose)        //进入左通道采集数据
        begin
            state <= state_right_data;
        end
    end
    else
    begin
        bit_cnt <= bit_cnt - 1'd1;
```

```
        reg_wrfifo_data[bit_cnt] = adcdat_r1;
    end
end
```

进入右通道数据采集状态时，bit_cnt 的值继续依次减 1，并将 ADCDAT 上的值放至 reg_wrfifo_data 对应的 bit_cnt 位上，当 bit_cnt == 5'd0 时，代表两个通道的数据采集完成，就需要跳转到写 FIFO 状态，将得到的 reg_wrfifo_data 的数据写入 FIFO 中，这里需要注意的是，bit_cnt 的值减为 0 的时候，此时最低位的数据还未写入至 reg_wrfifo_data 中，所以 bit_cnt 减为 0 时，还需要将此时 ADCDAT 上最后一位数据存放至 reg_wrfifo_data 的最低位中，具体实现代码如下所示：

```
state_right_data:
begin
    if(bit_cnt == 5'd0)          //32 位数据采集完成
    begin
        reg_wrfifo_data[bit_cnt] = adcdat_r1;
        state <= state_fifo_write;
    end
    else begin
        bit_cnt <= bit_cnt - 1'd1;
        reg_wrfifo_data[bit_cnt] = adcdat_r1;
        state <= state_right_data;
    end
end
```

最后进入 FIFO 的写数据状态，将 FIFO 的写使能信号拉高 adcfifo_write，并将得到的 reg_wrfifo_data 数据写入 FIFO 中，跳转至空闲状态，等待下一次的传输，代码如下所示：

```
state_fifo_write:
begin
  adcfifo_write <= 1'd1;
  adcfifo_writedata <= reg_wrfifo_data;
  state <= state_idle;
end
```

最后就是例化 FIFO 模块，这里的 FIFO 我们使用的是一个通用 FIFO，方便不同平台的代码移植，代码如下所示：

```
async_fifo #(
  .DATA_WIDTH(DATA_WIDTH),
  .ADDR_WIDTH(8),
  .FULL_AHEAD(1),
  .SHOWAHEAD_EN(0)
)adc_fifo
(
```

```
  .reset(~reset_n),
  //fifo wr
  .wrclk(bclk),
  .wren(adcfifo_write),
  .wrdata(adcfifo_writedata),
  .full(adcfifo_full),
  .almost_full(),
  .wrusedw(),
  //fifo rd
  .rdclk(adcfifo_rdclk),
  .rden(adcfifo_read),
  .rddata(adcfifo_readdata),
  .empty(adcfifo_empty),
  .rdusedw()
);
```

根据上述描述，i2s_rx 模块我们就设计完成了。

1.3.4 i2s_tx

I2S 发送模块的主要功能就是将 FIFO 读出的数据按照 I2S 协议格式传输出去，该模块的基本框图如下所示。

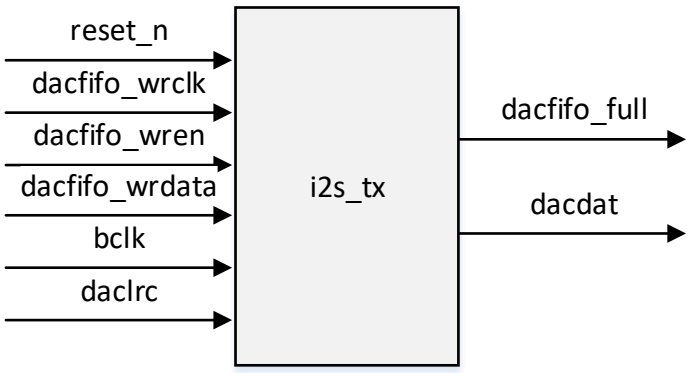

图 1-10 I2S 发送模块基本框图

对上述模块的信号说明如下所示。

表 1-11 i2s_tx 信号说明表

信号名称	I/O	信号意义
reset_n	I	复位信号，低电平有效
dacfifo_wrclk	I	FIFO 的写时钟信号
dacfifo_wren	I	FIFO 的写使能信号
dacfifo_wrdata	I	FIFO 的写数据信号
bclk	I	音频接口的位时钟信号
daclrc	I	音频接口 DAC 的左/右时钟信号

dacfifo_full	O	FIFO 的满标志信号
dacdat	O	DAC 数字音频数据信号

i2s_tx 模块同样也是根据图 1-9 的时序图完成数据的发送。首先获取 DACLRC 信号的边沿信号，如下所示：

```
reg daclrc_r0;
reg daclrc_nege;
reg daclrc_pose;
always@(posedge bclk) begin
    daclrc_r0 <= daclrc;
end
always@(posedge bclk or negedge reset_n)
if(~reset_n) begin
    daclrc_pose <= 1'd0;
    daclrc_nege <= 1'd0;
end
else begin
    daclrc_pose <= daclrc & (!daclrc_r0);
    daclrc_nege <= (!daclrc) & daclrc_r0;
end
```

然后当 FIFO 不为空并且 DACLRC 产生下降沿的时候，此时我们将 FIFO 的读使能信号拉高，将 FIFO 中存储的数据读出，如下所示：

```
assign dacfifo_rden = (~dacfifo_empty && daclrc_nege) ? 1'd1 : 1'd0;
```

我们同样使用状态机完成数据的发送，一共定义三个状态：空闲状态、发送左通道数据状态、发送右通道数据状态，如下所示：

```
parameter state_idle = 2'd0;  //空闲状态
parameter state_tx_left_data = 2'd1;  //发送左通道数据信号
parameter state_tx_right_data = 2'd2; //发送右通道数据信号
```

DACLRC 发送数据时是根据 BCLK 的上升沿发送出去的，那么我们数据变化必须在 BCLK 的下降沿，并且必须先发送高位然后发送低位，这里同样用 bit_cnt 的值来代表 DACLRC 需要发送的对应数据位，当处于空闲状态的时候，bit_cnt 的值等于 DATA_WIDTH - 1'd1，当检测到 DACLRAC 的下降沿开始发送左通道数据，当 bit_cnt == DATA_WIDTH/2 - 1'd1 的时候，左通道数据发送完成，跳转到发送右通道数据状态，当 bit_cnt == 0 的时候，数据发送完成，返回空闲状态，等待下一个数据的传输，整个状态机的代码如下所示：

```
always@(negedge bclk or negedge reset_n)
if(~reset_n)
begin
    state <= state_idle;
    bit_cnt <= 8'd0;
    dacdat <= 1'd0;
```

```
end
else
begin
    case(state)
        state_idle:
        begin
            bit_cnt <= DATA_WIDTH - 1'd1;
            dacdat <= 1'd0;
            if(daclrc_nege)        //开始发送左通道数据
            begin
                state <= state_tx_left_data;
                bit_cnt <= bit_cnt - 1'd1;
                dacdat <= dacfifo_rddata_r0[bit_cnt];
            end
        end

        state_tx_left_data:
        begin
            if(bit_cnt == DATA_WIDTH/2 - 1'd1)
            begin
                dacdat <= 1'd0;
                if(daclrc_pose) begin
                    state <= state_tx_right_data;
                    bit_cnt <= bit_cnt - 1'd1;
                    dacdat <= dacfifo_rddata_r0[bit_cnt];
                end
            end
            else
            begin
                bit_cnt <= bit_cnt - 1'd1;
                dacdat <= dacfifo_rddata_r0[bit_cnt];
            end
        end

        state_tx_right_data:
        begin
            if(bit_cnt == 0)
            begin
                state <= state_idle;
                bit_cnt <= DATA_WIDTH - 1'd1;
                dacdat <= dacfifo_rddata_r0[bit_cnt];
            end
            else
            begin
                bit_cnt <= bit_cnt - 1'd1;
                dacdat <= dacfifo_rddata_r0[bit_cnt];
            end
```

```
        end
    endcase
end
```

我们在发送数据的时候，并不是将 FIFO 读出的数据直接进行传输，而是根据 daclrc 的下降沿，将 FIFO 寄存的数据打了一拍，这样操作是因为 FIFO 读出的数据同样也是根据 DACLRC 的边沿来变化的，如果将此时读出的数据直接按位发出，那么传输数据的时候就会出错，结合图 1-11 所示的进行理解。从图中可以看出，当出现 daclrc_nege 信号之后，dacfifo_rden 信号也被拉高，bit_cnt 也会在 bclk 的下降沿发生变化，但是 FIFO 读出的数据在下一个 bclk 的上升沿才会输出，此时 bit_cnt 值已经减一，那么 dacdat 在输出数据的时候最高位将会出错，无法输出，所以我们在 DACLRC 下降沿，将上一个从 FIFO 读出的数据按照 DACLRC 的时钟进行打拍，那么只会让数据延迟一个 DACLRC 时钟，并不会导致数据出错，这样我们通过耳机听出的声音也是正确的，并不会出现噪声。

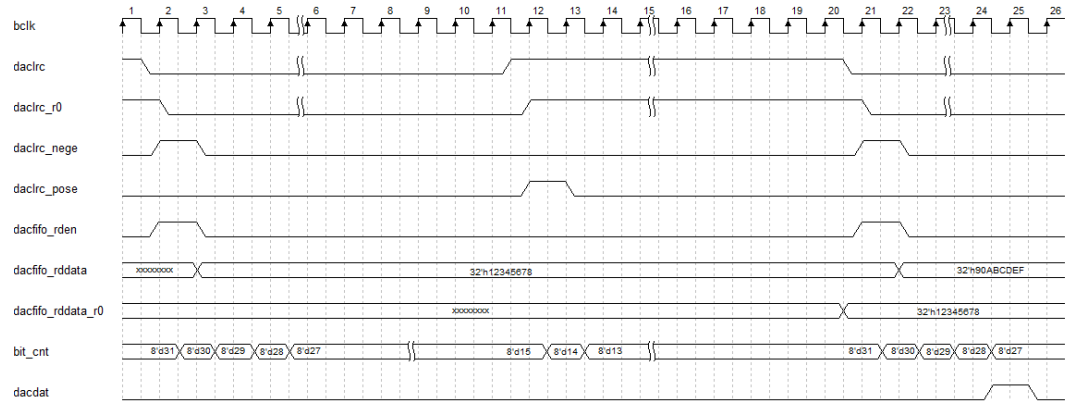

图 1-11 i2s_tx 模块信号波形说明图（DATA_WIDTH=32 为例）

根据上诉讲解，本次实验所涉及到的所有模块均已讲解完成，只需要在顶层模块中将各个模块进行例化，并将对应信号进行连接，在顶层中我们还需要对 ADCFIFO 的读使能信号和 DACFIFO 的写使能信号进行控制，当 ADCFIFO 不为空的时候，拉高读使能信号，将 ADCFIFO 中的数据读出，当 ADCFIFO 不为空和并且 DACFIFO 未满的情况下，将 ADCFIFO 中读出的数据存入 DACFIFO 中，最终将数据输出，如下所示：

```
always @ (posedge clk or negedge reset_n)
begin
    if (~reset_n)
    begin
        adcfifo_read <= 1'b0;
    end
    else if (~adcfifo_empty)
    begin
```

```
        adcfifo_read <= 1'b1;
    end
    else
    begin
        adcfifo_read <= 1'b0;
    end
end

always @ (posedge clk or negedge reset_n)
begin
    if(~reset_n)
        dacfifo_write <= 1'd0;
    else if(~dacfifo_full && (~adcfifo_empty)) begin
        dacfifo_write <= 1'd1;
        dacfifo_writedata <= adcfifo_readdata;
    end
    else begin
        dacfifo_write <= 1'd0;
    end
end
```

至此，本次实验相关的代码设计就讲解完成了，完整的代码请自行查看对应的源工程文件。

1.4 板级验证

1.4.1 引脚分配

本次实验对应的引脚分配表如下所示。

表 1-12 引脚分配表

引脚名称	AC620 开发板对应的引脚编号
clk	E1
reset_n	M16
iic_0_scl	D8
iic_0_sda	F7
led	A2
I2S_BCLK	B8
I2S_DI	A10
I2S_DO	F8
I2S_RCLK	B10
I2S_MCLK	A7

按照上述表中的内容进行引脚分配，然后编译工程，直至没有错误。

1.4.2 系统所需硬件

1. AC620 发板一块
2. 3.5mm 公对公音频线一根
3. 有线耳机一个
4. 电源线
5. 下载器

1.4.3 硬件连接及现象说明

将电源线和下载器依次连接至开发板上，然后就是连接耳机和音频线，用一根音频线连接开发板上的音频口和电脑上的音频口（或者带有耳机孔的手机），需要注意此时需要连接开发板上的音频输入口，开发板上有丝印（INI），对应 AC620 开发板右上角的蓝色音频口，然后用将耳机插入开发板上的音频输出口，开发板上有丝印（OUT），对应 AC620 开发板右上角的绿色音频口，整个硬件连接如所示。

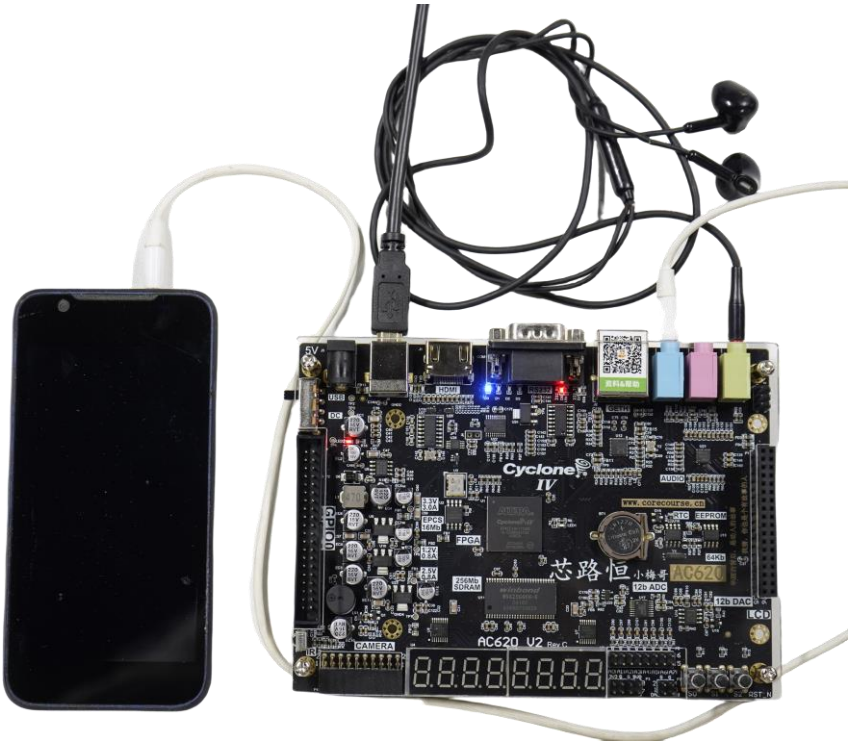

图 1-12 硬件连接图

连接完成之后，将生成的工程文件下载至开发板中，可以看到开发板上的 LED0 被点亮，此时说明音频芯片 ES8388 初始化完成，然后戴上耳机，打开电

脑或者手机上的音乐播放软件，如果从耳机中能听到音乐且声音清晰无噪音说明本次音频回环实验成功。

1.5 思考与总结

本章设计中由手机或电脑等设备输入到 ES8388 中为模拟音频信号，信号经过内部 ADC 的采集处理后转变为数字音频信号，并以 I2S 格式的数字音频接口输出给 I2S 接收模块，然后经由 I2S 输出模块将接收到的音频数据转换成 I2S 格式从数据接口重新传输给 ES8388 模块，此时的数据为数字音频信号，经过内部的 DAC 处理后会再度转变为模拟信号，最终再经由 ES8388 的耳机接口输出，实现音频数据的回环。

本次设计主要是对 ES8388 芯片寄存器的配置以及 I2S 发送和接收模块的设计，在官方手册的“CONFIGURATION REGISTER DEFINITION”一节中对各个寄存器的每一位和具体 I2S 时序都进行了详细说明，用户可以自行查阅。