第5章 综合设计

本章的内容主要是在前面实验的基础上做一些综合性的实验，将前面实现的两个或两个以上的实验结合在一起，并加以修改，实现多样化的接口设计，比如将音频、VGA 结合在一起实现音频、视频的综合性实验，或者将 VGA 和 USB 接口结合在一起用 USB 键盘控制 VGA 的显示等，目的是让学生能将学到的知识更加灵活地应用。

鉴于前面的实验基础，本章综合性实验的讲解只介绍实验的方案以及关键点，而不对实验的具体细节做详细描述，把它作为检验学生学习能力和创新能力的一部分。

5.1 USB 及 VGA 接口设计

5.1.1 实验目的及概述

1. 实验目的

(1) 了解和掌握 VGA 接口以及嵌入式主 USB 设备及其接口的原理。

(2) 学习和掌握在 EDK 开发平台上设计特定的 IP Core 以及 SOPC 的综合设计方法。

(3) 在前面章节的 VGA 以及 USB 接口的基础上，利用已有的 USB 键盘及 LCD 显示的工程，将前面用 VHDL 设计的 VGA 硬件模块作为嵌入式主 USB 接口工程中的一个 IP Core 加入其中，实现利用 USB 键盘输入的特定值控制图像的移动。

2. 实验概述

本实验主要内容包括：

(1) 利用前面章节设计的主 USB 设备接口的工程，在实验箱的 USB 接口接入 USB 键盘，输入 USB 键盘值，在实验箱的 LCD 上显示其输入值。

(2) 将前面章节用 VHDL 设计的 VGA 模块改成一个 IP Core 添加到嵌入式主 USB 设备模块中，添加 IP Core 模块的方法如 2.3.3 节所述。每当连接在嵌入式 USB 主设备上面的 USB 键盘输入时，除了将其中的值显示到 LCD 显示器上(此部分的设计为验证嵌入式主 USB 设备能正确接收来自 USB 键盘的信息)，同时，如果输入值为 W、S、A、D，则同时向 VGA 子模块通过系统总线发送消息，W、S、A、D 分别代表控制图像向上、下、左、右移动。

5.1.2 实验设计方案

实验设计框图如图 5-1 所示，其中 Self_0 为 VGA 图像显示硬件模块，将其作为一个自己设计的 IP Core 嵌入到系统中，通过系统总线获取来自 USB 键盘的控制数据。

1. 硬件设计关键点

硬件设计是在前面的主 USB 接口的工程上添加 VGA 的 IP Core，因为原来的 VGA 模

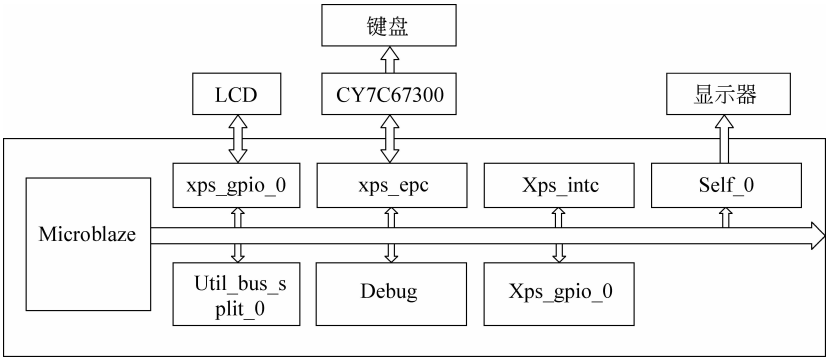

图 5-1　实验设计框图

块是一个独立的 VHDL 模块，要加到 USB 接口的工程上，就需要将 VGA 模块进行修改，方法是在 EDK 界面利用创建 IP Core 的子程序菜单 Create and Import Peripheral，在向导的指引下先创建一个 self 的 IP Core，用 self 模块作为 VGA 接口控制器模块。这样在 USB 的工程下有一个 PCORE 的目录，其中产生了两个文件：self. vhdl 和 userlogic. vhdl，将原 VGA 模块的 vhdl 代码添加到 userlogic. vhd 中并加以修改，同时修改 self. vhdl 的接口及端口映射信号(参考 2. 3. 3 节)。

User Logic 模板自身有 IPIC 总线一侧接口信号的部分，VGA 模块修改的方式是保留默认的 userlogic. vhd 的总线部分的端口信号，然后将 VGA 的端口加入到 userlogic. vhd 端口中，并将 VGA 的源代码加入，在源代码中加入接收总线信号并判断键盘值输入的代码，将接收到的特定键盘值作为移动图像的条件，但图像的基本显示不受键盘值的影响。

具体设计时只需要增加以下几个方面的内容：

(1) 将 VGA 的端口信号加入到 userlogic. vhd 端口中。

(2) 在系统时钟的上升沿以及写信号有效时读取 IPIC 总线上的数据并存到 slv_reg0、slv_reg1 寄存器中，这一部分的 VHDL 代码可以采用 userlogic. vhd 产生的源代码，不用做修改，这部分的代码如下：

```
slv_reg_write_sel <=Bus2IP_WrCE(0 to 1);
slv_reg_read_sel  <=Bus2IP_RdCE(0 to 1);
slv_write_ack     <=Bus2IP_WrCE(0) or Bus2IP_WrCE(1);
slv_read_ack      <=Bus2IP_RdCE(0) or Bus2IP_RdCE(1);
--implement slave model software accessible register(s)
SLAVE_REG_WRITE_PROC: process( Bus2IP_Clk) is
    begin
        if Bus2IP_Clk'event and Bus2IP_Clk ='1' then
        if Bus2IP_Reset ='1' then
            slv_reg0 <=(others =>'0');
            slv_reg1 <=(others =>'0');
        else
```

```
            case slv_reg_write_sel is
                when "10" =>
                    for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
                        if ( Bus2IP_BE(byte_index) ='1') then
                            slv_reg0(byte_index * 8 to byte_index * 8+7) <=
                            Bus2IP_Data(byte_index * 8 to byte_index * 8+7);
                        end if;
                    end loop;
                when "01" =>
                    for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
                        if ( Bus2IP_BE(byte_index) ='1') then
                            slv_reg1(byte_index * 8 to byte_index * 8+7) <=
                            Bus2IP_Data(byte_index * 8 to byte_index * 8+7);
                        end if;
                    end loop;
                when others =>null;
            end case;
        end if;
    end if;
end process SLAVE_REG_WRITE_PROC;
--implement slave model software accessible register(s) read mux
SLAVE_REG_READ_PROC: process( slv_reg_read_sel, slv_reg0, slv_reg1) is
begin
    case slv_reg_read_sel is
        when "10" =>slv_ip2bus_data <=slv_reg0;
        when "01" =>slv_ip2bus_data <=slv_reg1;
        when others =>slv_ip2bus_data <= (others =>'0');
    end case;
end process SLAVE_REG_READ_PROC;
------------------------------------------
--Example code to drive IP to Bus signals
------------------------------------------
IP2Bus_Data <=slv_ip2bus_data when slv_read_ack ='1' else
              (others =>'0');
IP2Bus_WrAck <=slv_write_ack;
IP2Bus_RdAck <=slv_read_ack;
IP2Bus_Error <='0';
```

(3) 将 VGA 的 VHDL 源代码加到 userlogic.vhd 中,因为 VGA 的 VHDL 源代码顶层代码下面包含了 3 个子程序,需要将这 3 个子程序引入,时钟分频的子程序可以改成直接用代码写,因为这部分比较简单,而引入原来用 IP Core 生成的 IP Core 反而麻烦。

(4) 增加在 VGA 像素时钟(PCLK)的上升沿从 slv_reg1 读取 USB 键盘输入值,并判断是否为控制图像的特定值,如果是,则根据该值改变图像的显示方式。图 5-2 是修改后的

User Logic 设计流程，这部分代码需要自己编写。

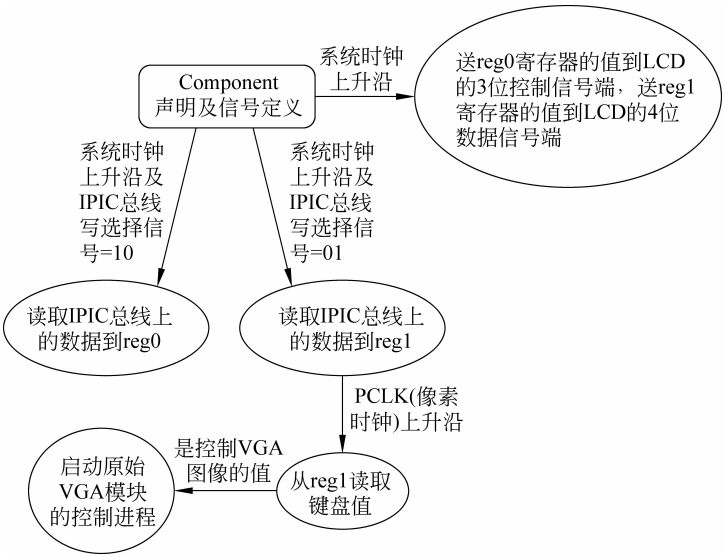

图 5-2　修改后的 User Logic 设计流程

2. 软件设计关键点

在设计软件时，CY7C67300 器件内部的应用程序不需要修改，可以用 USB 主设备接口的工程代码，因为这只涉及 USB 接口。

FPGA 内部固件程序则需要修改，修改的原则是：保留原来的 USB 接口及 LCD 显示部分的代码，增加将接收到的键盘值送到总线上的内容，这一步实际上是和 VGA 的硬件模块配合动作，VGA 的硬件模块一直在检测总线上有没有特定的键盘值，软件可以通过 C 语言的地址指针将键盘值送到与总线有关联的寄存器，这样，地址和键盘数据就会反映到总线上，硬件模块也可以捕捉到特定的键盘值。如果工程中没有其他设备在总线上传递数据，USB 键盘数据是唯一的数据，那么捕捉数据就不需要地址的配合，只需要和总线上的一些使能信号配合就可以了。FPGA 片内软件设计只需要修改 USB 主设备接口的主程序部分。FPGA 内部固件程序的流程如图 5-3 所示。

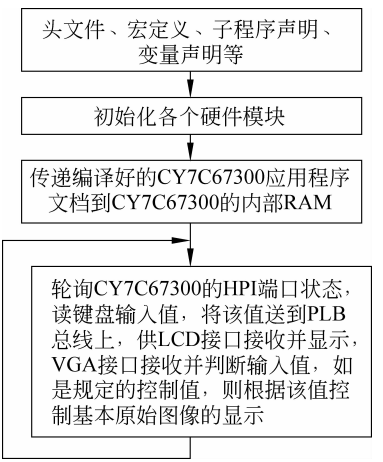

图 5-3　FPGA 片内软件设计流程图

5.1.3　实验

实验题：用 USB 键盘控制 VGA 显示图像及图像移动。

实验进阶：将 USB 键盘换成 USB 摄像头，将 USB 摄像头的摄像数据通过 VGA 接口显示在显示屏上。

5.2 音视频播放综合设计

5.2.1 实验目的及概述

1. 实验目的

(1) 了解和掌握音频、VGA 接口及播放、显示的原理和设计方法。

(2) 学习和掌握在 EDK 开发平台上设计特定的 IP Core 以及 SOPC 的综合设计方法。

(3) 在前面章节用 VHDL 设计 VGA 接口的基础上,将 VGA 接口的模块添加到一个基本的 SOPC 工程中,再用 VHDL 语言设计一个简单的音频模块(这个简易的音频模块与前面章节的音频接口不同),该模块可以通过实验板上的按钮播放事先存在 FPGA 内部 ROM 的音符,SOPC 的基本工程中有可用 GPIO 控制的 DIP 接口控制器、用 DIP 拨码开关控制音频的播放以及 VGA 图像的变化。

2. 实验概述

音频模块事先在 FPGA 的 ROM 中存入一段音频,通过 DIP 拨码开关可实现左、右声道两种播放模式。

视频模块事先在 FPGA 的 ROM 中存入一幅图像,通过 DIP 拨码开关可实现图片的平行移动、放大缩小两种显示方式。

在 EDK 设计框架下,融合音频、VGA 接口功能,实时检测开关状态,呈现不同音频输出与视频输出的组合模式。

5.2.2 实验设计方案

1. 硬件设计关键点

音视频接口分别用 VHDL 的方式实现,以 IP Core 的方式加入 EDK 的基本工程中,以子模块的方式并行工作。在 EDK 工程中,硬件上有 3 个主要模块:DIP_Mode、Video、Audio,如图 5-4 所示。音频、VGA 模块共同以 ROM 作为数据存储器。以 PLB 总线时钟作为初始时钟输入。图 5-5 为音视频子模块结构图。

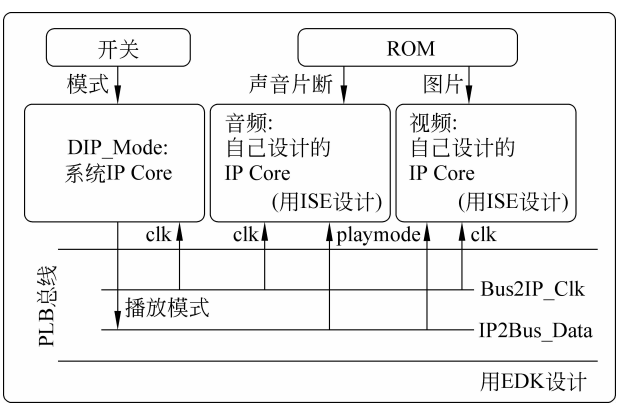

图 5-4 音视频接口硬件框架

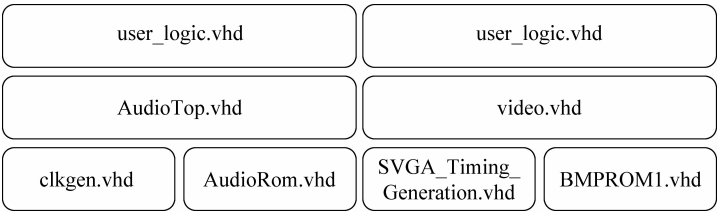

图 5-5　音视频子模块结构图

1）DIP_Mode 模块

在设计上 DIP_Mode 模块与 2.3.2 节添加一个系统的 GPIO 方式一样，用 GPIO 的输入端与实验板上的 DIP 开关连接，目的是通过系统检测 DIP 的开关状态以确定要实现的音视频功能。

2）Video 模块

Video 模块的实现与 5.1 节中 VGA 模块的实现方式相似，只是控制 VGA 图像移动的不是 USB 键盘，而是 DIP 开关，DIP 开关通过 GPIO 模块与总线连接。

3）Audio 模块

音频模块中，顶层模块下有 3 个子模块，AudioTop 模块为最顶层模块，ClkGen 模块为分频时钟生成模块，Par2Ser 模块为数据并转串模块，AudioROM 为存储音乐数据文件的 ROM 模块。为了减少子模块数量，后期完成时，可以将 Par2Ser 并串转换模块融合到 AudioTop 模块中。

各模块之间的关系如图 5-6 所示。

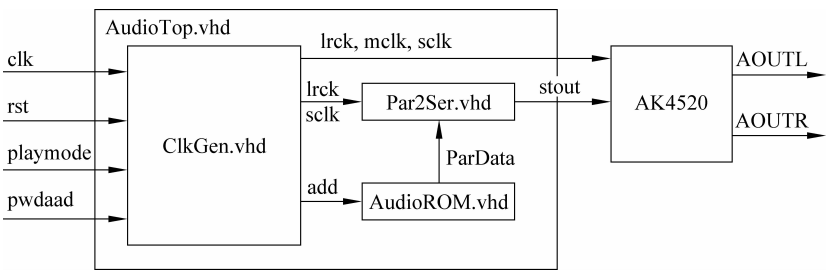

图 5-6　Audio 模块结构图

时钟模块的设计和 3.5 节中的音频接口设计一样。

AudioROM 模块是将一段音频数据存储在 FPGA 的 ROM 中，做法与 VGA 接口工程中的图像放在 FPGA 的 ROM 中是一样的，所以可以参考 VGA 接口的方式。关键的是需要录一段音频数据，然后将音频数据格式转换成与 AK4520 相匹配的数据格式（即 20 位、二补码的数字格式）。wav 格式就是根据二补码格式进行采样转换后按照一定的规律和格式编写的，因此可以取一段 wav 格式的音乐，如《蓝色多瑙河》，并打开查看它的内容（查看时，需要将文件转化为十六进制表示格式），打开后的部分内容如图 5-7 所示。根据 wav 标准协议，可以对图 5-8 的数据片段作以下的内容解释：从 NumChannels 字段得知本音乐为双声道音频，从 SampleRate 字段得知采样率为 44.1kHz，从 BitsPerSample 字段得知每个样本用 16 位数表示，这里虽然与 AK4520 所要求的 20 位有所不同，但是经过试验发现 16 位数

据同样可以使用。因此，实际的数字音频信息应从第三行 c 字段开始，每两个字段为一个数据段，按照左声道(低位)－左声道(高位)－右声道(低位)－右声道(高位)的顺序排列。在转换时，首先将第三行 c 列之前所有内容删除，再利用十六进制转二进制的程序将文件转换为二进制码，最后按照 AK4520 的要求，需要将高位放到前面。可以编写一段程序进行转换，主要功能是将高、低数据位的二进制码对换位置。

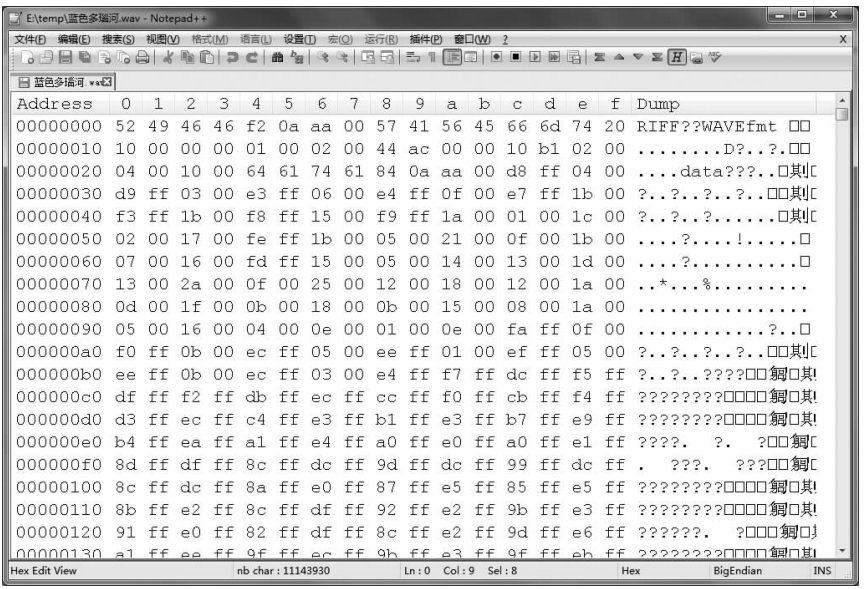

图 5-7 《蓝色多瑙河》数据片段

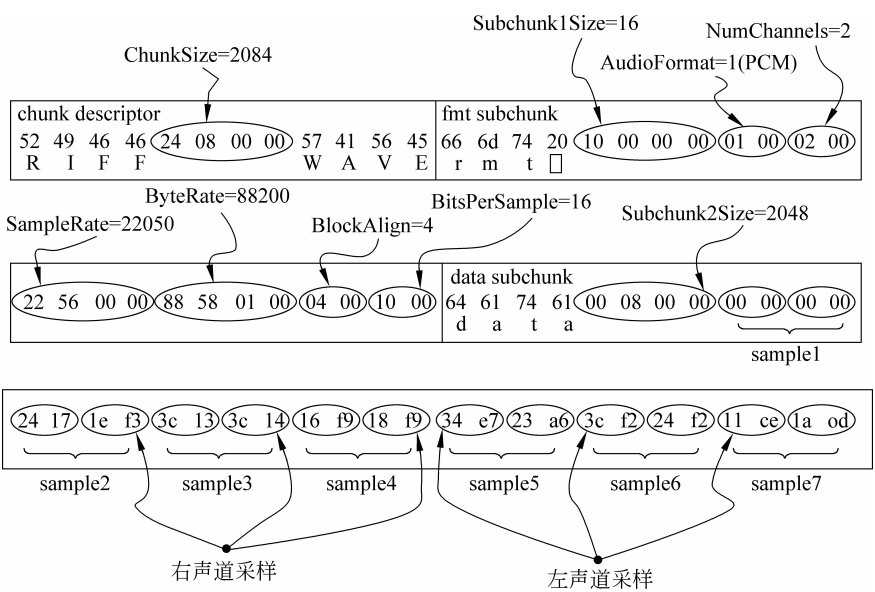

图 5-8 《蓝色多瑙河》数据片段分析

AudioTop 模块融合了 Par2Ser 模块，是整个系统的核心模块，其输入有 4 个，分别为 clk、reset、playmode 和 pwdaad；输出有 5 个，分别为 sdout、audio_lrck、audio_sclk、audio_

mclk 和 audio_mode。

(1) AudioROM 读取地址生成。

前面已经叙述过,在 clkgen 模块中,生成了一个 sublrck 信号,其频率为 lrck 的两倍,因此,在每个 sublrck 的上升沿对地址计数器加一,再利用 conv_integer_logicvector 函数即可生成新的地址,用来读取不同声道的数据。

(2) 并串转换模块。

并串转换模块以 sclk 时钟为驱动,判断依据为 clkgen 模块产生的 bit_cntr 计数器,当计数器数值小于 15 时,在每个 sclk 的上升沿读取一位并行数据,并赋值给串行数据输出信号。当计数器数值大于 16 时,串行数据输出保持为 0。

(3) playmode 左右声道选择。

利用 DIP 输入字段的对应关系,在不同模式下,选择性地对某一声道的数据进行清零。

下面是 AudioTop 模块的参考代码(融合了 Par2Ser 模块):

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.std_logic_arith.all;
entity audiotop is
    port (
        clk        : in  std_logic;        --100MHz
        reset      : in  std_logic;
        playmode   : in  std_logic_vector (2 downto 0);     系统端口声明
        pwdaad     : in  std_logic;
        sdout      : out std_logic;
        audio_lrck : out std_logic;
        audio_sclk : out std_logic;
        audio_mclk : out std_logic;
        audio_mode : out std_logic
        );
end audiotop;
architecture Behavioral of audiotop is
component clkgen is
    port (
        clk      : in  std_logic;          --100MHz
        reset    : in  std_logic;
        sclk     : out std_logic;
        mclk     : out std_logic;                          clkgen 子程序声明
        lrck     : out std_logic;
        bit_cntr : out integer range 0 to 31;
        sublrck  : out std_logic
    );
end component;
component AudioROM is
```

```
    port (
        clka  : in  std_logic;
        addra : in  std_logic_vector (14 downto 0);
        douta : out std_logic_vector (23 downto 0)
    );
end component;
--Signal for CLKGEN
signal sig_sclk, sig_lrck, sig_mclk, sig_sublrck: std_logic;
signal bitcounter: integer range 0 to 31;
--Signal for AudioROM
signal notecounter : integer range 0 to 19999:=0;
signal address     : std_logic_vector(14 downto 0);
signal music_tab   : std_logic_vector(23 downto 0);
--Signal for PAR2SER
signal sig_sdout: std_logic;
signal temp: std_logic;
signal playmode: std_logic_vector(2 downto 0);
begin
    audio_lrck <=sig_lrck;
    audio_sclk <=sig_sclk;
    audio_mclk <=sig_mclk;
    audio_mode <=not(pwdaad);
    u1: clkgen port map (clk, reset, sig_sclk, sig_mclk, sig_lrck, bitcounter,
    sig_sublrck);
    u2: AudioROM port map (clk, address, music_tab);
    process(sig_sublrck, clk, bitcounter, music_tab, playmode, sig_lrck)
    begin
        if sig_sublrck'event and sig_sublrck='1' then
            if notecounter <19999 then
                notecounter <=notecounter +1;
            else
                notecounter <=0;
            end if;
        end if;
        address <=conv_std_logic_vector(notecounter,15);
        if clk'event and clk='1' then
            if bitcounter >15 then
                temp <='0';
            else
                temp <=music_tab(15-bitcounter);
            end if;
            if playmode ="010" then    --Steoro
                sig_sdout <=temp;
```

AudioROM 子程序声明

信号声明

端口映射

产生 ROM 读取地址

进行并串转换

```
            elsif playmode ="001" then     --Left Track
                if sig_lrck ='1' then
                    sig_sdout <=temp;
                else
                    sig_sdout <='0';
                end if;
            elsif playmode ="100" then--Right Track
                if sig_lrck ='1' then
                    sig_sdout <='0';
                else
                    sig_sdout <=temp;
                end if;
            else
                sig_sdout <='0';
            end if;
        end if;
    end process;
    sdout <=sig_sdout;
end Behavioral;
```

依据不同的播放模式,对输出左右声道的数据进行置零

4) userlogic. vhdl

最后需要将 Audio 模块加入到 EDA 工程中,做法与 VGA 模块相同,其中的关键是需要增加判断播放音频的控制信号,该控制信号是从 DIP 开关输入,通过 GPIO 模块再通过总线传递过来。

2. 软件设计关键点

软件设计实际上是指 FPGA 内部 CPU 的固件代码,这一部分的设计也比较简单,可以沿用 2.3.2 节的工程代码,子程序保持不变,只是修改主程序部分。主程序部分的代码非常简单,只需要把 DIP 拨码开关的状态传递到总线即可。图 5-9 是主程序部分代码的框架。

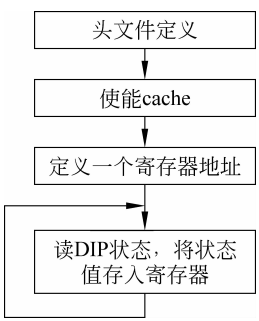

图 5-9 音频播放、VGA 显示软件流程图

5.2.3 实验

实验题:用 SOPC 方式通过 DIP 拨码开关控制音频的播放以及 VGA 图像的变化。

实验进阶:增加 SRAM 接口,将音、视频数据存放在 SRAM 中,通过按键控制音频的播放和图像的显示。

5.3 音频播放、处理以及 VGA 动态显示设计

5.3.1 实验目的与概述

1. 实验目的

(1) 了解和掌握 VGA 接口以及音频接口的原理。