Digital Instrument Multi Effects Processing Unit

CSEE W4840 Embedded System Design

Ziqian Deng (zd2285), Longyi Li (ll3527), Yifan Zhan (yz4149), Yuqi Zhu (yz4137)

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Contents

1 Overview

1.1 Introduction

Since long ago, live musicians are no longer satisfied with the original sound their instruments make; they seek ways to modify how they sound through effects pedals, where the sound signals are processed in certain ways that fit the music and make them sound unique.

The 1940s saw the first standalone guitar effects unit from DeArmond. The unit is capable of producing tremolo effects by passing guitar signals through a water-based electrolytic fluid, mentioned in a brief history of guitar effects. Then several effect pedals such as delay, echo, fuzz, and distortion emerged. Those pedals were driven by purely analog transistor circuits. In the 1980s, a rise in digitized rack mount effects units occurred. Instead of standalone stompboxes, the digitized counterparts were capable of producing multiple effects in one box and storing presets for quick loading. Nowadays, the multi-effects processing units have gone through significant improvements in sound quality, delay, size and looks.

Therefore, the proposed multi-effects signal processor used DSP to realize the following effects: cabinet IR (discrete convolution), limiter effect, equalizer effect, delay effect.

1.2 Milestones

Starting from one basic effect (cabinet IR or equivalent) to test the functionality of the ADC, DAC, and FIFOs, then gradually adding effect modules, connected by Avalon-ST, and finally implementing the user interface, controlling the modules using Avalon-MM.

For each effect module, the following steps will be required:

- Matlab simulation of effect algorithms
- Simulation from behavioral HDL codes
- On-board tests

The milestones are:

- Complete an ADC-DAC signal path
- Implement effects in verilog
- Implement a user interface

1.3 Work Statement

Contribution of each member to this project is stated as follows:

Ziqian Deng - FIR module, delay module and clip module
Longyi Li - Top level of the design and pre-amplifier hardware
Yifan Zhan - VGA user interface
Yuqi Zhu - Biquad module

2 Implementation

2.1 System Architecture

The proposed system consists of two major parts including the hardware audio data path and software control implemented on the DE1-SOC.

Audio samples are sourced from the line-in input of the WM8731 Codec on the DE-1 SOC board. The ADC on the WM8731 sends serial PCM audio data in 16-bit 48kHz Left Justified format to the FPGA. A deserializer is used to convert the audio data into a parallel format and store them into FIFO registers to be processed. Following the FIFO will be the various signal processing hardware, connected by Avalon Streaming Interface, including a discrete convolution block, a series of biquads, a limiter, and a delay block. The blocks are connected in series and there is a bypass control for each block to turn the effect on or off in the software. The parameters used in the blocks such as gain, biquad coefficients, and discrete convolution coefficients will be configured through the Avalon Memory-Mapped bus by the software. At the end of the Datapath will be a serializer to convert the audio data into the 16-bit 48kHz Left Justified format supported by the DAC of the WM8731.

To configure the effects, a VGA user interface is implemented using sprites with various sliders to control parameters such as mixing ratio and delay length. The software receives user input from a USB keyboard, controls the display content, and configures parameters such as biquad coefficients in the audio data path.

Additional analog hardware, the Impedance Matching Pre-Amplifier, is required to connect the guitar to the WM7831's Line-In due to the guitar's high output impedance and low output voltage swing. The analog hardware is a two-stage amplifier with a high input impedance at the first stage and a programmable gain at the second stage.

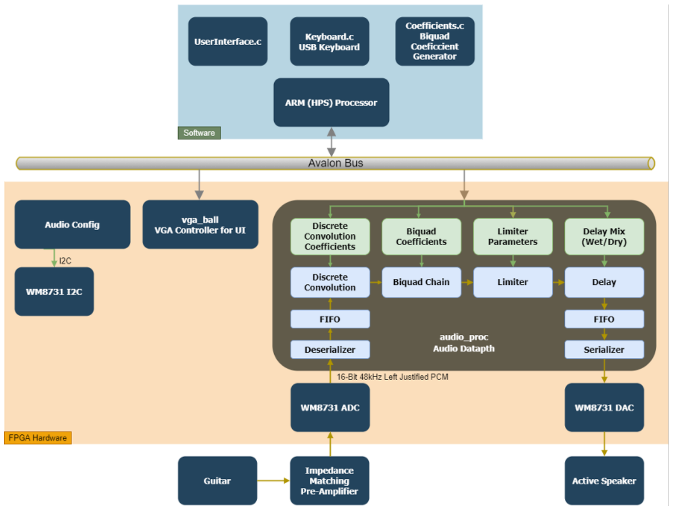


Figure 1: System Architecture

2.2 Hardware

Four guitar effects are implemented in this project including Cabinet IR, EQ, hard-clip, and delay. Other hardware required includes vga ball and i2s interface.

2.2.1 Analog Pre-Amplifier

An analog pre-amplifier is required for connecting the guitar to the DE1. The amplifier has a unity gain first stage with high input impedance (450k) and a second stage with low output impedance and some gain programmable by dip switches.

Single 12V supply is used with AC coupling at the input and output. 6V common mode voltage exist at the intermediate.

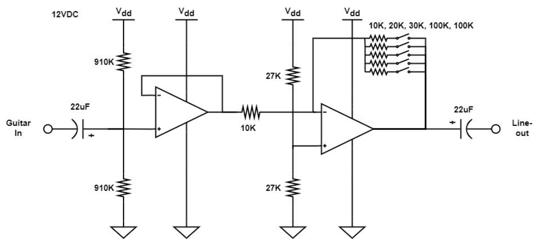


Figure 2: Pre-Amp Schematic

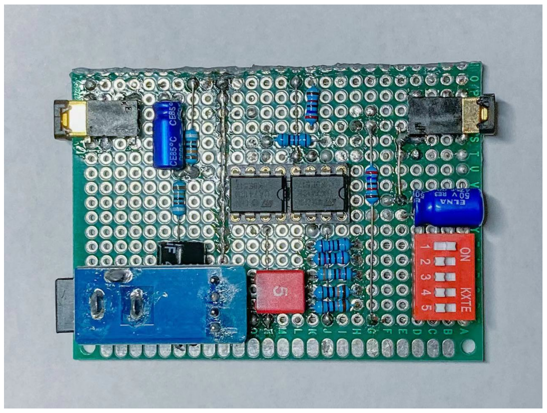


Figure 3: Pre-Amp Hardware

2.2.2 I2S Interface

The I2S interface converts the Left-Justified serial data from the ADC of the WM8731 to 16-bit parallel format and stores audio samples as signed integers in FIFO. After the audio has been processed by other blocks, the I2S interface module serializes the data back into I2S format for the DAC of the WM8731.

As shown in Figure 4, the I2S module includes an Avalon-ST source from the ADC and an Avalon-ST sink to the DAC. The ADCDAT and DACDAT signals are I2S serial signals connected to ADC and DAC, respectively. The BCLK and LRCLK signals are generated by the WM8731 which is in master mode. The BCLK ticks for every bit of new data, and the LRCLK ticks to indicate left/right channels.

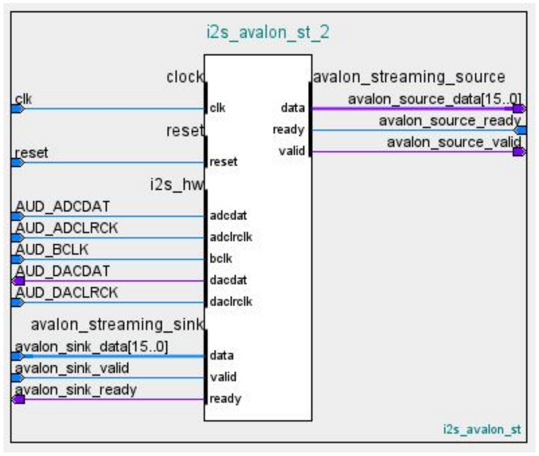


Figure 4: I2S Interface Module

2.2.3 Cabinet IR

The Cabinet IR effect is the discrete convolution of the audio signal with an impulse response signal to recreate the sound of a particular speaker cabinet. In this project, a pipelined FIR filter is used to perform discrete convolution with a set of coefficients set by software with the input audio signal.

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] \tag{1}$$

Figure 5 shows FIR module. It includes an Avalon-ST sink for data input and an Avalon-ST source for data output. It also includes an Avalon-MM interface for reloading FIR coefficients using the HPS and user space program.

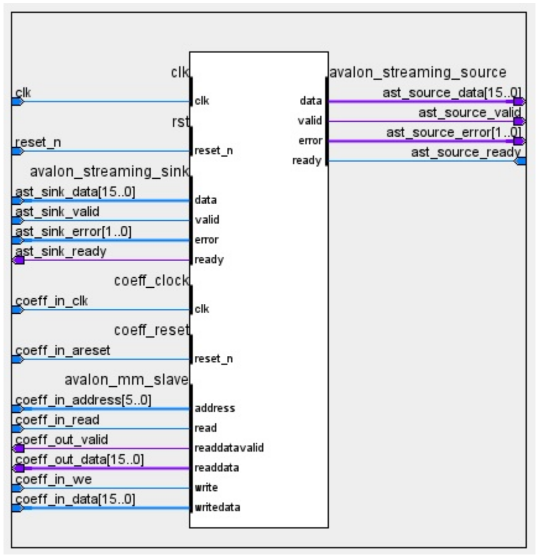


Figure 5: FIR Module

2.2.4 Biquad Chain

Biquads are second order IIR filter units that are used to create analog filters such as peaking, shelving, etc. Each biquad has five coefficients, as shown in Equation 2. The time-domain difference equation implemented in Verilog for each biquad is shown in Equation 3. A three-band EQ is implemented with a chain of such biquads to adjust bass, mid, and treble.

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 Z^{-1} + b_2 Z^{-2}}{1 + a_1 Z^{-1} + a_2 Z^{-2}} \tag{2}$$

$$y[n] = \left(\frac{b_0}{a_0}\right) x[n] + \left(\frac{b_1}{a_0}\right) x[n-1] + \left(\frac{b_2}{a_0}\right) x[n-2] - \left(\frac{a_1}{a_0}\right) y[n-1] - \left(\frac{a_2}{a_0}\right) y[n-2] \tag{3}$$

The biquad module is shown in Figure 6. Four biquads are included in this module. An Avalon-ST sink is used for data input and an Avalon-ST source for data output. It also includes an Avalon-MM interface for reloading biquad coefficients using the HPS and user space program. There is a total of four biquads, with five coefficients for each biquad.

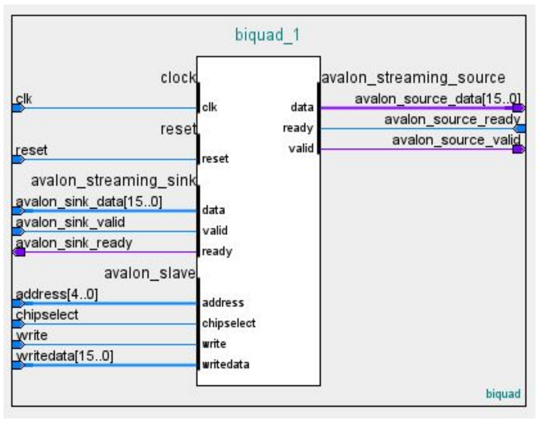


Figure 6: Biquad Module

2.2.5 Limiter

The limiter clips the audio signal above a certain threshold, adding distortion, which is desired in electric guitars. Two forms of clipping are implemented. The first is a hard clip, which places a hard limit on the signal's amplitude. The second is a soft clip, which is implemented by combining the hardclip with a low pass biquad to remove the sharp edges.

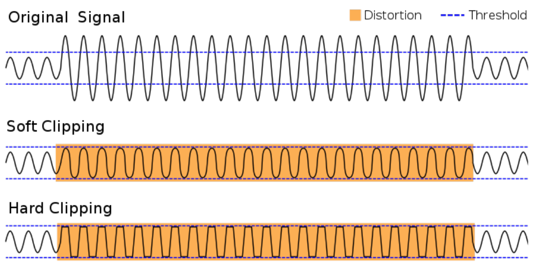


Figure 7: Limiter

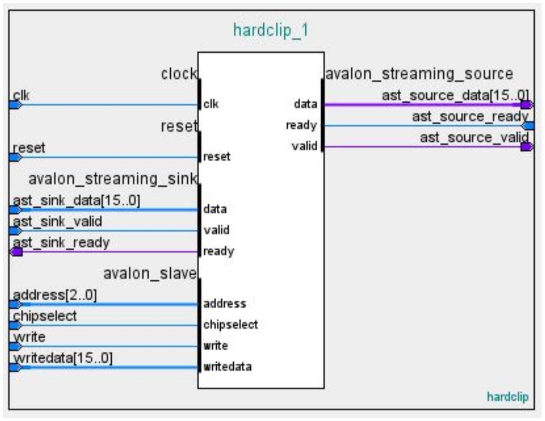


Figure 8: Hardclip Module

2.2.6 Delay

The delay block implements the data path shown in Figure 10. Some audio data is stored in FIFO to be delayed (wet) and are mixed with samples that have not been delayed (dry). The mixing factor K determines how much wet signal is added to the output, which can be set by software/UI.

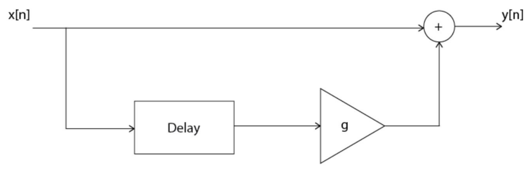


Figure 9: Delay Effect

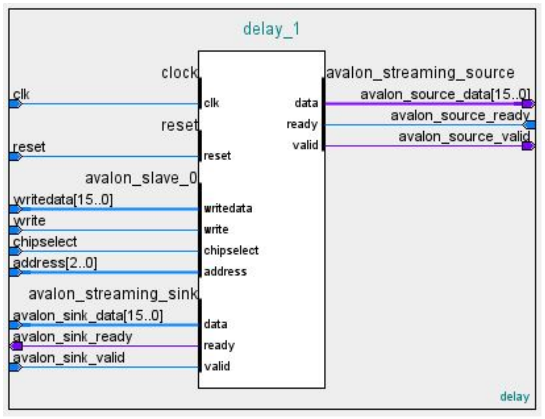


Figure 10: Delay Module

2.3 VGA

The VGA user interface is designed using photoshop and converted to a memory initialization file (mif) using matlab. The background image is a 640*480 image with only 8 bits of colors to save memory and avoid using off-chip SDRAM. The sliders and switches are smaller sprites also stored in mif files.

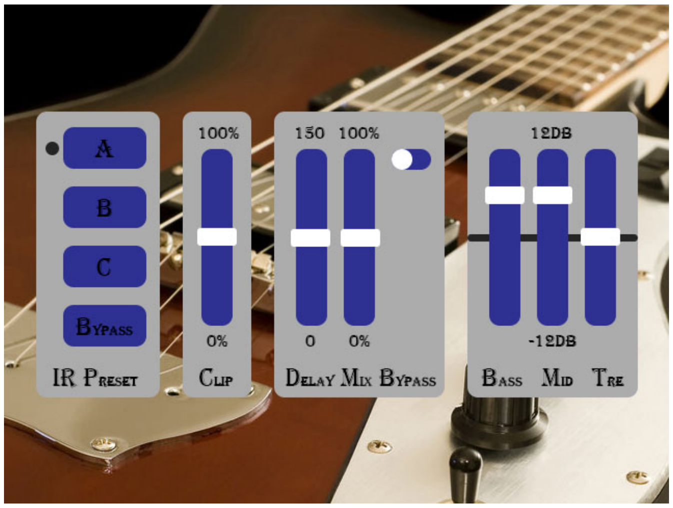
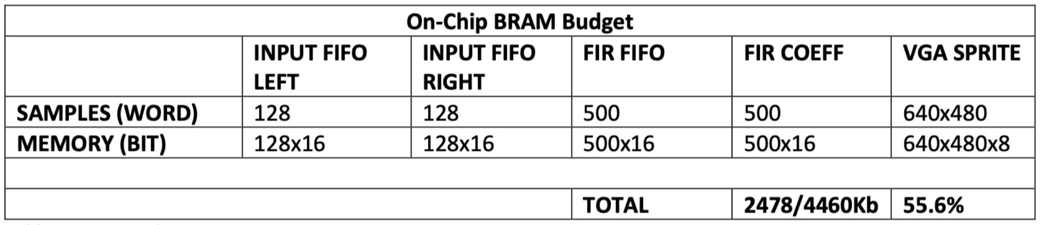


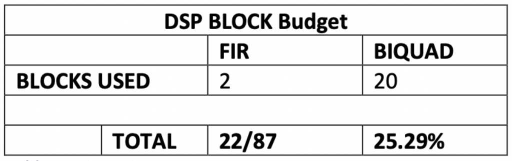
Figure 11: VGA UI

3 Resource Budget



On-Chip BRAM Budget					
	INPUT FIFO LEFT	**INPUT FIFO RIGHT**	**FIR FIFO**	**FIR COEFF**	**VGA SPRITE**
SAMPLES (WORD)	128	128	500	500	640x480
MEMORY (BIT)	128x16	128x16	500x16	500x16	640x480x8
			TOTAL	**2478/4460Kb**	**55.6%**

Figure 12: BRAM Budget



DSP BLOCK Budget			
		FIR	**BIQUAD**
BLOCKS USED		2	20
	TOTAL	**22/87**	**25.29%**

Figure 13: DSP BLOCK Budget

4 Hardware/Software Interface

Each effect block has Avalon Memory-Mapped registers.

4.1 Cabinet IR

- Address: 0x0000_1000 - 0x0000_13ff
- 500 FIR Coefficients

4.2 Biquad Chain

- Address: 0x0000_0100 - 0x0000_013f
- Reg 0˜4: BIQUAD 1 COEFF
- Reg 5˜9: BIQUAD 2 COEFF
- Reg 10˜14: BIQUAD 3 COEFF
- Reg 15˜19: BIQUAD 4 COEFF
- Reg 20: BYPASS

4.3 Limiter

- Address: 0x0000_0010 - 0x0000_001f
- Reg 0: CLIP THRESHOLD POS
- Reg 1: CLIP THRESHOLD NEG

4.4 Delay

- Address: 0x0000_0020 - 0x0000_002f
- Reg 0: BYPASS
- Reg 1: DELAY LENGTH
- Reg 2: MIX

4.5 VGA Ball

- Address: 0x0000_2000 - 0x0000_200f

- Reg 0: BYPASS SWITCH POSITION
- Reg 1: PRESET SELECT POSITION
- Reg 2: SLIDER 1 POSITION
- Reg 3: SLIDER 2 POSITION
- Reg 4: SLIDER 3 POSITION
- Reg 5: SLIDER 4 POSITION
- Reg 6: SLIDER 5 POSITION
- Reg 7: SLIDER 6 POSITION

A Design Files

A.1 Hardware

A.1.1 soc_system.sv

```
// ============================================================================
// Copyright (c) 2013 by Terasic Technologies Inc.
// ============================================================================
//
// Modified 2022
//
// Permission:
//
//   Terasic grants permission to use and modify this code for use
//   in synthesis for all Terasic Development Boards and Altera
//   Development Kits made by Terasic.  Other use of this code,
//   including the selling ,duplication, or modification of any
//   portion is strictly prohibited.
//
// Disclaimer:
//
//   This VHDL/Verilog or C/C++ source code is intended as a design
//   reference which illustrates how these types of functions can be
//   implemented.  It is the user's responsibility to verify their
//   design for consistency and functionality through the use of
//   formal verification methods.  Terasic provides no warranty
//   regarding the use or functionality of this code.
//
// ============================================================================
//
//  Terasic Technologies Inc

//  9F., No.176, Sec.2, Gongdao 5th Rd, East Dist, Hsinchu City, 30070. Taiwan
//
//
//                     web: http://www.terasic.com/
//                     email: support@terasic.com
module soc_system_top(

 ///////// ADC /////////
 inout          ADC_CS_N,
 output         ADC_DIN,
 input          ADC_DOUT,
 output         ADC_SCLK,

 ///////// AUD /////////
 input          AUD_ADCDAT,
 inout          AUD_ADCLRCK,
 inout          AUD_BCLK,
 output         AUD_DACDAT,
 inout          AUD_DACLRCK,
 output         AUD_XCK,

 ///////// CLOCK2 /////////
 input          CLOCK2_50,

 ///////// CLOCK3 /////////
 input          CLOCK3_50,

 ///////// CLOCK4 /////////
 input          CLOCK4_50,

 ///////// CLOCK /////////
 input          CLOCK_50,

 ///////// DRAM /////////
 output [12:0] DRAM_ADDR,
 output [1:0]  DRAM_BA,
 output        DRAM_CAS_N,
 output        DRAM_CKE,
 output        DRAM_CLK,
 output        DRAM_CS_N,
 inout [15:0]  DRAM_DQ,
 output        DRAM_LDQM,
 output        DRAM_RAS_N,
 output        DRAM_UDQM,
 output        DRAM_WE_N,

 ///////// FAN /////////
 output        FAN_CTRL,

 ///////// FPGA /////////
 output        FPGA_I2C_SCLK,
 inout         FPGA_I2C_SDAT,

 ///////// GPIO /////////
 inout [35:0]  GPIO_0,
 inout [35:0]  GPIO_1,

 ///////// HEX0 /////////
 output [6:0]  HEX0,

 ///////// HEX1 /////////
 output [6:0]  HEX1,

 ///////// HEX2 /////////
 output [6:0]  HEX2,

 ///////// HEX3 /////////
 output [6:0]  HEX3,

 ///////// HEX4 /////////
 output [6:0]  HEX4,

 ///////// HEX5 /////////
 output [6:0]  HEX5,

 ///////// HPS /////////
 inout         HPS_CONV_USB_N,
 output [14:0] HPS_DDR3_ADDR,
 output [2:0]  HPS_DDR3_BA,
 output        HPS_DDR3_CAS_N,
 output        HPS_DDR3_CKE,
 output        HPS_DDR3_CK_N,
 output        HPS_DDR3_CK_P,
 output        HPS_DDR3_CS_N,
 output [3:0]  HPS_DDR3_DM,
 inout [31:0]  HPS_DDR3_DQ,
 inout [3:0]   HPS_DDR3_DQS_N,
 inout [3:0]   HPS_DDR3_DQS_P,
 output        HPS_DDR3_ODT,
 output        HPS_DDR3_RAS_N,
 output        HPS_DDR3_RESET_N,
 input           HPS_DDR3_RZQ,
 output          HPS_DDR3_WE_N,
 output          HPS_ENET_GTX_CLK,
 inout           HPS_ENET_INT_N,
 output          HPS_ENET_MDC,
 inout           HPS_ENET_MDIO,
 input           HPS_ENET_RX_CLK,
 input  [3:0]    HPS_ENET_RX_DATA,
 input           HPS_ENET_RX_DV,
 output  [3:0]   HPS_ENET_TX_DATA,
 output          HPS_ENET_TX_EN,
 inout           HPS_GSENSOR_INT,
 inout           HPS_I2C1_SCLK,
 inout           HPS_I2C1_SDAT,
 inout           HPS_I2C2_SCLK,
 inout           HPS_I2C2_SDAT,
 inout           HPS_I2C_CONTROL,
 inout           HPS_KEY,
 inout           HPS_LED,
 inout           HPS_LTC_GPIO,
 output          HPS_SD_CLK,
 inout           HPS_SD_CMD,
 inout  [3:0]    HPS_SD_DATA,
 output          HPS_SPIM_CLK,
 input           HPS_SPIM_MISO,
 output          HPS_SPIM_MOSI,
 inout           HPS_SPIM_SS,
 input           HPS_UART_RX,
 output          HPS_UART_TX,
 input           HPS_USB_CLKOUT,
 inout  [7:0]    HPS_USB_DATA,
 input           HPS_USB_DIR,
 input           HPS_USB_NXT,
 output          HPS_USB_STP,

 ///////// IRDA /////////
 input           IRDA_RXD,
 output          IRDA_TXD,

 ///////// KEY /////////
 input  [3:0]    KEY,

 ///////// LEDR /////////
 output  [9:0]   LEDR,

 ///////// PS2 /////////
 inout           PS2_CLK,
 inout           PS2_CLK2,
 inout           PS2_DAT,
 inout           PS2_DAT2,

 ///////// SW /////////
 input  [9:0]    SW,

 ///////// TD /////////
 input           TD_CLK27,
 input  [7:0]    TD_DATA,
 input           TD_HS,
 output          TD_RESET_N,
 input           TD_VS,


 ///////// VGA /////////
 output [7:0]  VGA_B,
 output        VGA_BLANK_N,
 output        VGA_CLK,
 output [7:0]  VGA_G,
 output        VGA_HS,
 output [7:0]  VGA_R,
 output        VGA_SYNC_N,
 output        VGA_VS
);

   soc_system soc_system0(
     .clk_clk                      ( CLOCK_50 ),
     .reset_reset_n                ( 1),

     .hps_ddr3_mem_a               ( HPS_DDR3_ADDR ),
     .hps_ddr3_mem_ba              ( HPS_DDR3_BA ),
     .hps_ddr3_mem_ck              ( HPS_DDR3_CK_P ),
     .hps_ddr3_mem_ck_n            ( HPS_DDR3_CK_N ),
     .hps_ddr3_mem_cke             ( HPS_DDR3_CKE ),
     .hps_ddr3_mem_cs_n            ( HPS_DDR3_CS_N ),
     .hps_ddr3_mem_ras_n           ( HPS_DDR3_RAS_N ),
     .hps_ddr3_mem_cas_n           ( HPS_DDR3_CAS_N ),
     .hps_ddr3_mem_we_n            ( HPS_DDR3_WE_N ),
     .hps_ddr3_mem_reset_n         ( HPS_DDR3_RESET_N ),
     .hps_ddr3_mem_dq              ( HPS_DDR3_DQ ),
     .hps_ddr3_mem_dqs             ( HPS_DDR3_DQS_P ),
     .hps_ddr3_mem_dqs_n           ( HPS_DDR3_DQS_N ),
     .hps_ddr3_mem_odt             ( HPS_DDR3_ODT ),
     .hps_ddr3_mem_dm              ( HPS_DDR3_DM ),
     .hps_ddr3_oct_rzqin           ( HPS_DDR3_RZQ ),

     .hps_hps_io_emac1_inst_TX_CLK ( HPS_ENET_GTX_CLK ),
     .hps_hps_io_emac1_inst_TXD0   ( HPS_ENET_TX_DATA[0] ),
     .hps_hps_io_emac1_inst_TXD1   ( HPS_ENET_TX_DATA[1] ),
     .hps_hps_io_emac1_inst_TXD2   ( HPS_ENET_TX_DATA[2] ),
     .hps_hps_io_emac1_inst_TXD3   ( HPS_ENET_TX_DATA[3] ),
     .hps_hps_io_emac1_inst_RXD0   ( HPS_ENET_RX_DATA[0] ),
     .hps_hps_io_emac1_inst_MDIO   ( HPS_ENET_MDIO  ),
     .hps_hps_io_emac1_inst_MDC    ( HPS_ENET_MDC   ),
     .hps_hps_io_emac1_inst_RX_CTL ( HPS_ENET_RX_DV ),
     .hps_hps_io_emac1_inst_TX_CTL ( HPS_ENET_TX_EN ),
     .hps_hps_io_emac1_inst_RX_CLK ( HPS_ENET_RX_CLK ),
     .hps_hps_io_emac1_inst_RXD1   ( HPS_ENET_RX_DATA[1]  ),
     .hps_hps_io_emac1_inst_RXD2   ( HPS_ENET_RX_DATA[2]  ),
     .hps_hps_io_emac1_inst_RXD3   ( HPS_ENET_RX_DATA[3]  ),

     .hps_hps_io_sdio_inst_CMD     ( HPS_SD_CMD          ),
     .hps_hps_io_sdio_inst_D0      ( HPS_SD_DATA[0]      ),
     .hps_hps_io_sdio_inst_D1      ( HPS_SD_DATA[1]      ),
     .hps_hps_io_sdio_inst_CLK     ( HPS_SD_CLK          ),
     .hps_hps_io_sdio_inst_D2      ( HPS_SD_DATA[2]      ),
     .hps_hps_io_sdio_inst_D3      ( HPS_SD_DATA[3]      ),

     .hps_hps_io_usb1_inst_D0      ( HPS_USB_DATA[0]     ),
     .hps_hps_io_usb1_inst_D1      ( HPS_USB_DATA[1]     ),
     .hps_hps_io_usb1_inst_D2      ( HPS_USB_DATA[2]     ),
     .hps_hps_io_usb1_inst_D3      ( HPS_USB_DATA[3]     ),
     .hps_hps_io_usb1_inst_D4      ( HPS_USB_DATA[4]     ),
     .hps_hps_io_usb1_inst_D5      ( HPS_USB_DATA[5]     ),
     .hps_hps_io_usb1_inst_D6      ( HPS_USB_DATA[6]     ),
     .hps_hps_io_usb1_inst_D7      ( HPS_USB_DATA[7]     ),
     .hps_hps_io_usb1_inst_CLK      ( HPS_USB_CLKOUT      ),
     .hps_hps_io_usb1_inst_STP      ( HPS_USB_STP         ),
     .hps_hps_io_usb1_inst_DIR      ( HPS_USB_DIR         ),
     .hps_hps_io_usb1_inst_NXT      ( HPS_USB_NXT         ),

     .hps_hps_io_spim1_inst_CLK     ( HPS_SPIM_CLK  ),
     .hps_hps_io_spim1_inst_MOSI    ( HPS_SPIM_MOSI ),
     .hps_hps_io_spim1_inst_MISO    ( HPS_SPIM_MISO ),
     .hps_hps_io_spim1_inst_SS0     ( HPS_SPIM_SS   ),

     .hps_hps_io_uart0_inst_RX      ( HPS_UART_RX     ),
     .hps_hps_io_uart0_inst_TX      ( HPS_UART_TX     ),

     .hps_hps_io_i2c0_inst_SDA      ( HPS_I2C1_SDAT     ),
     .hps_hps_io_i2c0_inst_SCL      ( HPS_I2C1_SCLK     ),

     .hps_hps_io_i2c1_inst_SDA      ( HPS_I2C2_SDAT     ),
     .hps_hps_io_i2c1_inst_SCL      ( HPS_I2C2_SCLK     ),

     .hps_hps_io_gpio_inst_GPIO09   ( HPS_CONV_USB_N ),
     .hps_hps_io_gpio_inst_GPIO35   ( HPS_ENET_INT_N ),
     .hps_hps_io_gpio_inst_GPIO40   ( HPS_LTC_GPIO ),

     .hps_hps_io_gpio_inst_GPIO48   ( HPS_I2C_CONTROL ),
     .hps_hps_io_gpio_inst_GPIO53   ( HPS_LED ),
     .hps_hps_io_gpio_inst_GPIO54   ( HPS_KEY ),
     .hps_hps_io_gpio_inst_GPIO61   ( HPS_GSENSOR_INT ),



        .i2c_SDAT                     (FPGA_I2C_SDAT),                     //       i2c.SDAT
        .i2c_SCLK                     (FPGA_I2C_SCLK),                     //          .SCLK
        .i2s_hw_adcdat                (AUD_ADCDAT),                //    i2s_hw.adcdat
        .i2s_hw_adclrclk              (AUD_ADCLRCK),              //          .adclrclk
        .i2s_hw_bclk                  (AUD_BCLK),                  //          .bclk
        .i2s_hw_dacdat                (AUD_DACDAT),                //          .dacdat
        .i2s_hw_daclrclk              (AUD_DACLRCK),              //          .daclrclk
        .i2s_mclk_clk                 (AUD_XCK),                 // i2s_mclk.clk
        .vga_r (VGA_R),
        .vga_g (VGA_G),
        .vga_b (VGA_B),
        .vga_clk (VGA_CLK),
        .vga_hs (VGA_HS),
        .vga_vs (VGA_VS),
        .vga_blank_n (VGA_BLANK_N),
        .vga_sync_n (VGA_SYNC_N),
  );

   // The following quiet the "no driver" warnings for output
   // pins and should be removed if you use any of these peripherals

   assign ADC_CS_N = SW[1] ? SW[0] : 1'bZ;
   assign ADC_DIN = SW[0];
   assign ADC_SCLK = SW[0];

   assign DRAM_ADDR = { 13{ SW[0] } };
   assign DRAM_BA = { 2{ SW[0] } };
   assign DRAM_DQ = SW[1] ? { 16{ SW[0] } } : 16'bZ;
   assign {DRAM_CAS_N, DRAM_CKE, DRAM_CLK, DRAM_CS_N,
           DRAM_LDQM, DRAM_RAS_N, DRAM_UDQM, DRAM_WE_N} = { 8{SW[0]} };

   assign FAN_CTRL = SW[0];


   assign GPIO_0 = SW[1] ? { 36{ SW[0] } } : 36'bZ;
   assign GPIO_1 = SW[1] ? { 36{ SW[0] } } : 36'bZ;

   assign HEX0 = { 7{ SW[1] } };
   assign HEX1 = { 7{ SW[2] } };
   assign HEX2 = { 7{ SW[3] } };
   assign HEX3 = { 7{ SW[4] } };
   assign HEX4 = { 7{ SW[5] } };
   assign HEX5 = { 7{ SW[6] } };

   assign IRDA_TXD = SW[0];

   assign LEDR = { 10{SW[7]} };

   assign PS2_CLK = SW[1] ? SW[0] : 1'bZ;
   assign PS2_CLK2 = SW[1] ? SW[0] : 1'bZ;
   assign PS2_DAT = SW[1] ? SW[0] : 1'bZ;
   assign PS2_DAT2 = SW[1] ? SW[0] : 1'bZ;

   assign TD_RESET_N = SW[0];



endmodule
```

A.1.2 vga_ball.sv

```
/*
 * Avalon memory-mapped peripheral that generates VGA
 *
 * Stephen A. Edwards
 * Columbia University
 */

module vga_ball(input logic        clk,
                input logic        reset,

                //                input logic bypass,
                input logic [7:0]  writedata,
                input logic        write,
                input              chipselect,
                input logic [2:0]  address,

                output logic [7:0] VGA_R, VGA_G, VGA_B,
                output logic       VGA_CLK, VGA_HS, VGA_VS,
                                   VGA_BLANK_n,
                output logic       VGA_SYNC_n);

   logic [10:0]    hcount;
   logic [9:0]     vcount;


         logic      bypass;
         logic [9:0] sel_x=40;//select for FIR
         logic [9:0] sel_y;//config by sw

         logic [9:0] sw_x=373;//
         logic [9:0] sw_y=139;//

         logic [9:0] slider1_x=187;//slider for clipping
```

```
        logic [9:0] slider1_y;//config by sw

        logic [9:0] slider2_x=278;//slider
        logic [9:0] slider2_y;//config by sw

        logic [9:0] slider3_x=324;//slider
        logic [9:0] slider3_y;//config by sw


        logic [9:0] slider4_x=464;//slider
        logic [9:0] slider4_y;//config by sw

        logic [9:0] slider5_x=510;//slider
        logic [9:0] slider5_y;//config by sw

        logic [9:0] slider6_x=556;//slider
        logic [9:0] slider6_y;//config by sw




   vga_counters counters(.clk50(clk), .*);


   always_ff @(posedge clk)
     if (reset) begin
        sel_y <= 132;
        slider1_y <= 232;
        slider2_y <= 289;
        slider3_y <= 255;
        slider4_y <= 150;
        slider5_y <= 255;
        slider6_y <= 150;
        bypass<=1;
     end else if (chipselect && write)
       case (address)
         3'h0 : bypass <= writedata;
         3'h1 : sel_y <= writedata;
         3'h2 : slider1_y <= writedata;
         3'h3 : slider2_y <= writedata;
         3'h4 : slider3_y <= writedata;
         3'h5 : slider4_y <= writedata;
         3'h6 : slider5_y <= writedata;
         3'h7 : slider6_y <= writedata;

       endcase

logic [9:0]hcnt;
logic [18:0] bgaddr;
logic [7:0] BG,SWON,SWOFF;
logic [7:0] seladdr;
logic [7:0] S1;
always_comb  begin
hcnt=hcount[10:1];
bgaddr=vcount*640+hcnt;
end
BG      B1 (
        .address ( bgaddr ),
        .clock ( clk ),
        .q (BG  )
        );

```

```
SEL      SEL1 (
         .address ( (hcnt-sel_x)+(vcount-sel_y)*13 ),
         .clock ( clk ),
         .q (S1  )
         );



SWON     SW1 (
         .address ( (hcnt-sw_x)+(vcount-sw_y)*38 ),
         .clock ( clk ),
         .q (SWON  )
         );

SWOFF    SW2 (
         .address ( (hcnt-sw_x)+(vcount-sw_y)*38 ),
         .clock ( clk ),
         .q (SWOFF  )
         );



   always_ff @(posedge clk) begin
         if(reset) begin


         end
         else begin
                 if (VGA_BLANK_n ) begin
                 if(hcnt>=slider1_x&&hcnt<slider1_x+36&&
                    vcount>=slider1_y&&vcount<slider1_y+17)begin
                         VGA_R<=255;
                         VGA_G<=255;
                         VGA_B<=255;


                 end
                 else if(hcnt>=slider2_x&&hcnt<slider2_x+36&&
                         vcount>=slider2_y&&vcount<slider2_y+17)begin
                         VGA_R<=255;
                         VGA_G<=255;
                         VGA_B<=255;
                 end
                 else if(hcnt>=slider3_x&&hcnt<slider3_x+36&&
                         vcount>=slider3_y&&vcount<slider3_y+17)begin
                         VGA_R<=255;
                         VGA_G<=255;
                         VGA_B<=255;

                 end
                 else if(hcnt>=slider4_x&&hcnt<slider4_x+36&&
                         vcount>=slider4_y&&vcount<slider4_y+17)begin
                         VGA_R<=255;
                         VGA_G<=255;
                         VGA_B<=255;

                 end
                 else    if(hcnt>=slider5_x&&hcnt<slider5_x+36&&
                            vcount>=slider5_y&&vcount<slider5_y+17)begin
                         VGA_R<=255;
                         VGA_G<=255;
                         VGA_B<=255;
```

```

                end
                else if(hcnt>=slider6_x&&hcnt<slider6_x+36&&
                        vcount>=slider6_y&&vcount<slider6_y+17)begin
                        VGA_R<=255;
                        VGA_G<=255;
                        VGA_B<=255;

                end
                else if(hcnt>=sel_x&&hcnt<sel_x+13&&vcount>=sel_y&&vcount<sel_y+13)begin
                VGA_R[7:5]<=S1[2:0];
                VGA_G[7:5]<=S1[5:3];
                VGA_B[7:6]<=S1[7:6];
                end
                else if(hcnt>=sw_x&&hcnt<sw_x+38&&vcount>=sw_y&&vcount<sw_y+20)begin
                if (bypass)begin
                        VGA_R[7:5]<=SWOFF[2:0];
                        VGA_G[7:5]<=SWOFF[5:3];
                        VGA_B[7:6]<=SWOFF[7:6];
                        end else begin
                        VGA_R[7:5]<=SWON[2:0];
                        VGA_G[7:5]<=SWON[5:3];
                        VGA_B[7:6]<=SWON[7:6];
                        end

                end
                else begin
                VGA_R[7:5]<=BG[2:0];
                VGA_R[4:0]<=5'b0;

                VGA_G[7:5]<=BG[5:3];
                VGA_G[4:0]<=5'b0;

                VGA_B[7:6]<=BG[7:6];
                VGA_B[5:0]<=6'b0;
                end
                end
          end
   end

endmodule

module vga_counters(
 input logic         clk50, reset,
 output logic [10:0] hcount,  // hcount[10:1] is pixel column
 output logic [9:0]  vcount,  // vcount[9:0] is pixel row
 output logic        VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n);

/*
 * 640 X 480 VGA timing for a 50 MHz clock: one pixel every other cycle
 *
 * HCOUNT 1599 0             1279       1599 0
 *             _______________              ________
 * ___________|    Video      |____________|  Video
 *
 *
 * |SYNC| BP |<-- HACTIVE -->|FP|SYNC| BP |<-- HACTIVE
 *       _______________________      _____________
 * |____|       VGA_HS          |____|
 */
   // Parameters for hcount
   parameter HACTIVE      = 11'd 1280,
             HFRONT_PORCH = 11'd 32,
             HSYNC        = 11'd 192,
             HBACK_PORCH  = 11'd 96,
             HTOTAL       = HACTIVE + HFRONT_PORCH + HSYNC +
                            HBACK_PORCH; // 1600

   // Parameters for vcount
   parameter VACTIVE      = 10'd 480,
             VFRONT_PORCH = 10'd 10,
             VSYNC        = 10'd 2,
             VBACK_PORCH  = 10'd 33,
             VTOTAL       = VACTIVE + VFRONT_PORCH + VSYNC +
                            VBACK_PORCH; // 525

   logic endOfLine;

   always_ff @(posedge clk50 or posedge reset)
     if (reset)          hcount <= 0;
     else if (endOfLine) hcount <= 0;
     else                hcount <= hcount + 11'd 1;

   assign endOfLine = hcount == HTOTAL - 1;

   logic endOfField;

   always_ff @(posedge clk50 or posedge reset)
     if (reset)          vcount <= 0;
     else if (endOfLine)
       if (endOfField)   vcount <= 0;
       else              vcount <= vcount + 10'd 1;

   assign endOfField = vcount == VTOTAL - 1;

   // Horizontal sync: from 0x520 to 0x5DF (0x57F)
   // 101 0010 0000 to 101 1101 1111
   assign VGA_HS = !( (hcount[10:8] == 3'b101) &
                      !(hcount[7:5] == 3'b111));
   assign VGA_VS = !( vcount[9:1] == (VACTIVE + VFRONT_PORCH) / 2);

   assign VGA_SYNC_n = 1'b0; // For putting sync on the green signal; unused

   // Horizontal active: 0 to 1279     Vertical active: 0 to 479
   // 101 0000 0000   1280              01 1110 0000  480
   // 110 0011 1111   1599              10 0000 1100  524
   assign VGA_BLANK_n = !( hcount[10] & (hcount[9] | hcount[8]) ) &
                        !( vcount[9] | (vcount[8:5] == 4'b1111) );

   /* VGA_CLK is 25 MHz
    *             __    __    __
    * clk50    __|  |__|  |__|
    *
    *             _____       __
    * hcount[0]__|     |_____|
    */
   assign VGA_CLK = hcount[0]; // 25 MHz clock: rising edge sensitive

endmodule
```

A.1.3 i2s_avalon_st.v

```
/*****************************************************************************
 *                           Constant Declarations                           *
 ****************************************************************************/


/****************************************************************************
 *                 Internal Wires and Registers Declarations                 *
 ****************************************************************************/

module i2s_avalon_st #(
parameter DW                                             = 15, //16 bit audio
parameter BIT_COUNTER_INIT      = 5'd15
)(
  input clk,
  input reset,

  input wire [DW:0] avalon_sink_data,
  input wire avalon_sink_valid,
  output wire avalon_sink_ready,

  output wire [DW:0] avalon_source_data,
  output wire avalon_source_valid,
  input wire avalon_source_ready,

input      wire                         AUD_ADCDAT,
input           wire                            AUD_ADCLRCK,
input           wire                            AUD_BCLK,
input           wire                            AUD_DACLRCK,
output          wire                            AUD_DACDAT

  );
reg [8:0] validcounter; //128max
always@(posedge clk) begin
        if(reset)
                validcounter<=0;
        else begin
                validcounter=validcounter+1;
        end
end
assign avalon_sink_ready=audio_out_allowed;
assign avalon_source_valid=audio_in_available&&validcounter==7'd0;
// Internal Wires
wire                                            bclk_rising_edge;
wire                                            bclk_falling_edge;

wire                                            adc_lrclk_rising_edge;
wire                                            adc_lrclk_falling_edge;

wire            signed  [DW: 0] new_left_channel_audio;
wire            signed  [DW: 0] new_right_channel_audio;

wire                    [ 7: 0] left_channel_read_available;
wire                    [ 7: 0] right_channel_read_available;
wire                                            dac_lrclk_rising_edge;
wire                                            dac_lrclk_falling_edge;

wire                    [ 7: 0] left_channel_write_space;
wire                    [ 7: 0] right_channel_write_space;


// Internal Registers
reg                                             done_adc_channel_sync;
reg                                             read_interrupt_en;
reg                                             clear_read_fifos;
```

```verilog
reg                                             read_interrupt;

reg                                             done_dac_channel_sync;
reg                                             write_interrupt_en;
reg                                             write_interrupt;
reg                                             audio_in_available;
reg                                             audio_out_allowed;

// State Machine Registers


/*****************************************************************************
 *                         Finite State Machine(s)                           *
 *****************************************************************************/


/*****************************************************************************
 *                             Sequential Logic                              *
 *****************************************************************************/

always @(posedge clk)
begin
        if (reset == 1'b1)
                done_dac_channel_sync <= 1'b0;
        else if (dac_lrclk_falling_edge == 1'b1)
                done_dac_channel_sync <= 1'b1;
end
always @(posedge clk)
begin
        if (reset == 1'b1)
                done_adc_channel_sync <= 1'b0;
        else if (adc_lrclk_rising_edge == 1'b1)
                done_adc_channel_sync <= 1'b1;
end

always @ (posedge clk)
begin
        if (reset == 1'b1)
                audio_in_available <= 1'b0;
        else if ((left_channel_read_available[7] | left_channel_read_available[6])
                        & (right_channel_read_available[7] | right_channel_read_available[6]))
                audio_in_available <= 1'b1;
        else
                audio_in_available <= 1'b0;
end
always @ (posedge clk)
begin
        if (reset == 1'b1)
                audio_out_allowed <= 1'b0;
        else if ((left_channel_write_space[7] | left_channel_write_space[6])
                        & (right_channel_write_space[7] | right_channel_write_space[6]))
                audio_out_allowed <= 1'b1;
        else
                audio_out_allowed <= 1'b0;
end
/*****************************************************************************
 *                            Combinational Logic                            *
 *****************************************************************************/


/*****************************************************************************
 *                              Internal Modules                             *
 *****************************************************************************/

altera_up_clock_edge Bit_Clock_Edges (
	// Inputs
	.clk				(clk),
	.reset			(reset),

	.test_clk		(AUD_BCLK),

	// Bidirectionals

	// Outputs
	.rising_edge	(bclk_rising_edge),
	.falling_edge	(bclk_falling_edge)
);

altera_up_clock_edge ADC_Left_Right_Clock_Edges (
	// Inputs
	.clk				(clk),
	.reset			(reset),

	.test_clk		(AUD_ADCLRCK),

	// Bidirectionals

	// Outputs
	.rising_edge	(adc_lrclk_rising_edge),
	.falling_edge	(adc_lrclk_falling_edge)
);

altera_up_clock_edge DAC_Left_Right_Clock_Edges (
	// Inputs
	.clk				(clk),
	.reset			(reset),

	.test_clk		(AUD_DACLRCK),

	// Bidirectionals

	// Outputs
	.rising_edge	(dac_lrclk_rising_edge),
	.falling_edge	(dac_lrclk_falling_edge)
);


altera_up_audio_in_deserializer Audio_In_Deserializer (
	// Inputs
	.clk (clk),
	.reset (reset),
	.bit_clk_rising_edge (bclk_rising_edge),
	.bit_clk_falling_edge (bclk_falling_edge),
	.left_right_clk_rising_edge	(adc_lrclk_rising_edge),
	.left_right_clk_falling_edge (adc_lrclk_falling_edge),
	.done_channel_sync (done_adc_channel_sync),
	.serial_audio_in_data (AUD_ADCDAT),
	.read_left_audio_data_en (avalon_source_valid&&avalon_source_ready),
	.read_right_audio_data_en (avalon_source_valid&&avalon_source_ready),

	// Bidirectionals

	// Outputs
	.left_audio_fifo_read_space	(left_channel_read_available),
        .right_audio_fifo_read_space (right_channel_read_available),

        //.left_channel_data (new_left_channel_audio),
        .left_channel_data (avalon_source_data),//only send lch to avalon st
        .right_channel_data     (new_right_channel_audio)
);
defparam
        Audio_In_Deserializer.DW = DW,
        Audio_In_Deserializer.BIT_COUNTER_INIT = BIT_COUNTER_INIT;

altera_up_audio_out_serializer Audio_Out_Serializer (
        // Inputs
        .clk (clk),
        .reset (reset),

        .bit_clk_rising_edge (bclk_rising_edge),
        .bit_clk_falling_edge (bclk_falling_edge),
        .left_right_clk_rising_edge (done_dac_channel_sync & dac_lrclk_rising_edge),
        .left_right_clk_falling_edge (done_dac_channel_sync & dac_lrclk_falling_edge),

        .left_channel_data (avalon_sink_data),

        .left_channel_data_en (avalon_sink_valid&&avalon_sink_ready),

        .right_channel_data     (avalon_sink_data),

        .right_channel_data_en (avalon_sink_valid&&avalon_sink_ready),


        // Outputs
        .left_channel_fifo_write_space  (left_channel_write_space),
        .right_channel_fifo_write_space (right_channel_write_space),

        .serial_audio_out_data                          (AUD_DACDAT)
);
//END of ALTERA IP
endmodule
```

A.1.4 hardclip.v

```
`timescale 1 ps / 1 ps
module hardclip (
//AVALON ST
                input  wire         clk,                //clk.clk
                input  wire         reset,              //havent done any reset logic yet
                input  wire signed [15:0] ast_sink_data,  //avalon_streaming_sink.data
                input  wire         ast_sink_valid,     //.valid
                output reg         ast_sink_ready,      // .ready
                output reg signed [15:0] ast_source_data,  // avalon_streaming_source.data
                output reg         ast_source_valid,    // .valid
                input  wire         ast_source_ready,   //.ready

//AVALON MM
                input wire signed [15:0]  writedata,
                input wire          write,
                input               chipselect,
                input wire [2:0]   address

        );

//AVALON ST
        reg signed [15:0] tp;
        reg signed [15:0] tn;


always @(posedge clk) begin

        if(ast_sink_data>tp)
                ast_source_data<=tp;
        else if(ast_sink_data<tn)
                ast_source_data<=tn;
        else
                ast_source_data<=ast_sink_data;


        ast_source_valid<=ast_sink_valid;
        ast_sink_ready<=ast_source_ready;
        end


//AVALON MM
always @(posedge clk) begin
     if (reset) begin
             tp <= 16'd1000;
             tn <= -16'd1000;

     end else if (chipselect && write)
       case (address)
         3'h0 : tp <= writedata;
         3'h1 : tn <= writedata;
       endcase
end


endmodule
```

A.1.5 delay.v

```
`timescale 1ns/1ns
module delay
  (

  //AVALON ST
   input clk,
   input reset,
   input avalon_source_ready,
   output reg avalon_source_valid,
   output reg signed [DATA_WIDTH-1:0] avalon_source_data,

   output reg avalon_sink_ready,
   input avalon_sink_valid,
   input signed [DATA_WIDTH-1:0] avalon_sink_data,
//AVALON MM
                input wire signed [15:0]  writedata,
                input wire          write,
                input               chipselect,
                input wire [2:0]  address
   );


//AVALON MM REG
reg signed [15:0] mix;//adjust gain by software >>>1=-6dB, 2=12dB, etc.
wire signed [15:0] s_mix;
assign s_mix=mix;
```

```
reg [15:0] MAX_DELAY; //amount of samples to delay (no larger than 8192)
reg [15:0] bypass;//bypass
wire signed [31:0] result; //amount of samples to delay (no larger than 8192)
assign result = fifo_reg*s_mix;
parameter DATA_WIDTH = 16;
reg signed [DATA_WIDTH-1:0] in_reg;
wire signed [DATA_WIDTH-1:0] fifo_reg;
reg signed [15:0] mixin=16'd16384;
wire signed [31:0] indat=avalon_sink_data*mixin;
always @(*)begin
                in_reg = avalon_sink_data;
                avalon_source_data = |bypass?avalon_sink_data:result[31:16]+indat[30:15];
                //avalon_source_data = fifo_reg+avalon_sink_data;
end

//BEGIN AVALON MM
always @(posedge clk) begin

     if (reset) begin
                        mix<=16'd30000;//adjust gain by software >>>1=-6dB, 2=12dB, etc.
                        MAX_DELAY<= 6000; //amount of samples to delay
                        bypass<=16'd0;//default to bypass
     end else if (chipselect && write)
       case (address)
         3'h0 : bypass <= writedata;
         3'h1 : MAX_DELAY <= writedata;
         3'h2 : mix<=writedata;//adjust gain by software >>>1=-6dB, 2=12dB, etc.

       endcase
end



//END AVALON MM

  reg [15:0] ptr; //8192 samples =2^13
  reg [15:0] ptrr;

  always @(posedge clk)begin

 avalon_sink_ready<=avalon_source_ready;
 avalon_source_valid<=avalon_sink_valid;
    if (reset) begin
      ptr <= 0;
      ptrr <= 1;
end
else
         if (avalon_source_valid&&avalon_sink_ready)begin
      if (ptr+1 == MAX_DELAY)begin
        ptr <= 0;
        ptrr <= 1;
end
      else begin
        ptr <= ptr+1;
        ptrr <= ptrr+1;
        end
        end
 end
delayram        delayram_inst (
        .clock ( clk ),
        .data ( in_reg ),
        .rdaddress ( ptrr[12:0]),
        .rden (1),
        .wraddress ( ptr[12:0] ),
        .wren ( avalon_sink_valid),
        .q ( fifo_reg )
        );

endmodule
```

A.1.6 biquad.v

```
module biquad_ast #(
  parameter DW = 16
  /* coefficients */


  /*
a low pass filter for testing Q=0.7071, fc=1000Hz
b0_qq = 15
b1_qq = 31
b2_qq = 15
a1_qq = -31709
a2_qq = 15386


hpf 1000hz 0.7071
b0_qq =15869
b1_qq = -31740
b2_qq = 15869
a1_qq = -31709
a2_qq =  15386
                 */
  )(
  input clk,
  input reset,

  input wire signed [DW-1:0] avalon_sink_data,
  input wire avalon_sink_valid,
  output  reg avalon_sink_ready,

  output wire signed [DW-1:0] avalon_source_data,
  output reg avalon_source_valid,
  input wire avalon_source_ready,


  input wire signed [15:0]  b0_int,
  input wire signed [15:0]  b1_int,
  input wire signed [15:0]  b2_int,
  input wire signed [15:0]  a1_int,
  input wire signed [15:0]  a2_int,

  input wire bypass
  );

  wire signed [DW-1:0] input_int;
  assign input_int=avalon_sink_data;

  wire signed [DW-1:0] output_int;

  reg signed [DW-1:0] input_pipe1;
  reg signed [DW-1:0] input_pipe2;
  reg signed [DW-1:0] output_pipe1;
  reg signed [DW-1:0] output_pipe2;

  reg signed [DW + DW-1:0] input_b0;
  reg signed [DW + DW-1:0] input_b1;
  reg signed [DW + DW-1:0] input_b2;
  reg signed [DW + DW-1:0] output_a1;
  reg signed [DW + DW-1:0] output_a2;
  wire signed [DW + DW-1:0] output_2int;


  always @(posedge clk) begin
            avalon_source_valid <= avalon_sink_valid;
       avalon_sink_ready<=avalon_source_ready;
if (reset) begin
      input_pipe1 <= 0;
      input_pipe2 <= 0;
      output_pipe1 <= 0;
      output_pipe2 <= 0;
    end else
      if (avalon_sink_valid&&avalon_sink_valid) begin
        input_pipe1 <= input_int;
        input_pipe2 <= input_pipe1;
        output_pipe1 <= output_int;
        output_pipe2 <= output_pipe1;
      end

end
  always @(posedge clk)begin
      if (reset) begin
      input_b0 <= 0;
      input_b1 <= 0;
      input_b2 <= 0;
      output_a1 <= 0;
      output_a2 <= 0;
                end else begin
      input_b0 <= input_int * b0_int;
      input_b1 <= input_pipe1 * b1_int;
      input_b2 <= input_pipe2 * b2_int;
      output_a1 <= output_pipe1 * a1_int;
      output_a2 <= output_pipe2 * a2_int;
    end
  end
  assign output_2int = input_b0 + input_b1 + input_b2 - output_a1 - output_a2;
  assign output_int = output_2int[27:12];

  assign avalon_source_data = bypass?avalon_sink_data:output_int;

endmodule


module biquad (
  input clk,
  input reset,

  input wire signed [15:0] avalon_sink_data,
  input wire avalon_sink_valid,
  output wire avalon_sink_ready,

  output wire signed [15:0] avalon_source_data,
  output wire avalon_source_valid,
  input wire avalon_source_ready,

    //AVALON MM
  input wire signed [15:0]  writedata,
  input wire       write,
  input wire                chipselect,
  input wire [4:0]  address

  );
        reg signed [15:0] i0_b0_int;
        reg signed [15:0] i0_b1_int;
        reg signed [15:0] i0_b2_int;
        reg signed [15:0] i0_a1_int;
        reg signed [15:0] i0_a2_int;

        reg signed [15:0] i1_b0_int;
        reg signed [15:0] i1_b1_int;
        reg signed [15:0] i1_b2_int;
        reg signed [15:0] i1_a1_int;
        reg signed [15:0] i1_a2_int;

        reg signed [15:0] i2_b0_int;
        reg signed [15:0] i2_b1_int;
        reg signed [15:0] i2_b2_int;
        reg signed [15:0] i2_a1_int;
        reg signed [15:0] i2_a2_int;

        reg signed [15:0] i3_b0_int;
        reg signed [15:0] i3_b1_int;
        reg signed [15:0] i3_b2_int;
        reg signed [15:0] i3_a1_int;
        reg signed [15:0] i3_a2_int;
        wire signed [15:0] i0srcdat;
        wire signed [15:0] i1srcdat;
        wire signed [15:0] i2srcdat;

        reg [15:0] bypass;
biquad_ast i0(
  .clk(clk),
  .reset(reset),

  .avalon_sink_data(avalon_sink_data),
  .avalon_sink_valid(avalon_sink_valid),
  .avalon_sink_ready(avalon_sink_ready),

  .avalon_source_data(i0srcdat),
  .avalon_source_valid(i0srcv),
  .avalon_source_ready(i0srcr),

  .b0_int(i0_b0_int),
  .b1_int(i0_b1_int),
  .b2_int(i0_b2_int),
  .a1_int(i0_a1_int),
  .a2_int(i0_a2_int),
  .bypass(bypass[0])
);

biquad_ast i1(
  .clk(clk),
  .reset(reset),

  .avalon_sink_data(i0srcdat),
  .avalon_sink_valid(i0srcv),
  .avalon_sink_ready(i0srcr),

```

```verilog
  .avalon_source_data(i1srcdat),
  .avalon_source_valid(i1srcv),
  .avalon_source_ready(i1srcr),

  .b0_int(i1_b0_int),
  .b1_int(i1_b1_int),
  .b2_int(i1_b2_int),
  .a1_int(i1_a1_int),
  .a2_int(i1_a2_int),
  .bypass(bypass[0])
);

biquad_ast i2(
  .clk(clk),
  .reset(reset),

  .avalon_sink_data(i1srcdat),
  .avalon_sink_valid(i1srcv),
  .avalon_sink_ready(i1srcr),

  .avalon_source_data(i2srcdat),
  .avalon_source_valid(i2srcv),
  .avalon_source_ready(i2srcr),

  .b0_int(i2_b0_int),
  .b1_int(i2_b1_int),
  .b2_int(i2_b2_int),
  .a1_int(i2_a1_int),
  .a2_int(i2_a2_int),
  .bypass(bypass[0])
);

biquad_ast i3(
  .clk(clk),
  .reset(reset),

  .avalon_sink_data(i2srcdat),
  .avalon_sink_valid(i2srcv),
  .avalon_sink_ready(i2srcr),

  .avalon_source_data(avalon_source_data),
  .avalon_source_valid(avalon_source_valid),
  .avalon_source_ready(avalon_source_ready),

  .b0_int(i3_b0_int),
  .b1_int(i3_b1_int),
  .b2_int(i3_b2_int),
  .a1_int(i3_a1_int),
  .a2_int(i3_a2_int),
  .bypass(bypass[0])
);

//BEGIN AVALON MM
always @(posedge clk) begin
     if (reset) begin //default to bypass
                        bypass<=1;

                        i0_b0_int<=16'd4096;
                        i0_b1_int<= 16'd0;
                        i0_b2_int<=16'd0;
                        i0_a1_int<= 16'd0;
                        i0_a2_int<=16'd0;

                        i1_b0_int<=16'd4096;
                        i1_b1_int<=16'd0;
                        i1_b2_int<=16'd0;
                        i1_a1_int<=16'd0;
                        i1_a2_int<=16'd0;

                        i2_b0_int<=16'd4096;
                        i2_b1_int<=16'd0;
                        i2_b2_int<=16'd0;
                        i2_a1_int<=16'd0;
                        i2_a2_int<=16'd0;

                   i3_b0_int<=16'd4096;
                        i3_b1_int<=16'd0;
                        i3_b2_int<=16'd0;
                        i3_a1_int<=16'd0;
                        i3_a2_int<=16'd0;
     end else if (chipselect && write)
       case (address)
         5'h0 : i0_b0_int  <= writedata;
         5'h1 : i0_b1_int  <= writedata;
         5'h2 : i0_b2_int<=writedata;
         5'h3 : i0_a1_int  <= writedata;
         5'h4 : i0_a2_int<=writedata;

         5'h5 : i1_b0_int  <= writedata;
         5'h6 : i1_b1_int  <= writedata;
         5'h7 : i1_b2_int<=writedata;
         5'h8 : i1_a1_int  <= writedata;
         5'h9 : i1_a2_int<=writedata;

         5'd10 : i2_b0_int  <= writedata;
         5'd11 : i2_b1_int  <= writedata;
         5'd12 : i2_b2_int<=writedata;
         5'd13 : i2_a1_int  <= writedata;
         5'd14 : i2_a2_int<=writedata;


         5'd15 : i3_b0_int  <= writedata;
         5'd16 : i3_b1_int  <= writedata;
         5'd17 : i3_b2_int<=writedata;
         5'd18 : i3_a1_int  <= writedata;
         5'd19 : i3_a2_int<=writedata;

         5'd20 : bypass<=writedata;
         endcase
end
//END AVALON MM
endmodule
```

A.2 Software

A.2.1 Makefile

```
ifneq (${KERNELRELEASE},)

# KERNELRELEASE defined: we are being compiled as part of the Kernel
        obj-m :=delay.o hardclip.o biquad.o fir.o vga_ball.o

else

```

```
# We are being compiled as a module: use the Kernel build system

        KERNEL_SOURCE := /usr/src/linux-headers-$(shell uname -r)
        PWD := $(shell pwd)



default: module hello
hello: hello.o usbkeyboard.o
        cc $(CFLAGS) -o hello hello.o usbkeyboard.o -lusb-1.0 -pthread -lm

hello.o: hello.c usbkeyboard.h
usbkeyboard.o: usbkeyboard.c usbkeyboard.h

module:
        ${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} modules

clean:
        ${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} clean
        ${RM} hello

usbkeyboard.o : usbkeyboard.c usbkeyboard.h


TARFILES = Makefile README delay.h delay.c hello.c hardclip.h hardclip.c
           biquad.h biquad.c fir.h fir.c vga_ball.h vga_ball.c usbkeyboard.h usbkeyboard.c
TARFILE = lab3-sw.tar.gz
.PHONY : tar
tar : $(TARFILE)

$(TARFILE) : $(TARFILES)
        tar zcfC $(TARFILE) .. $(TARFILES:%=lab3-sw/%)

endif
```

A.2.2 biquad.c

```
/* * Device driver for the variable biquad module
 *
 * A Platform device implemented using the misc subsystem
 *
 * Stephen A. Edwards
 * Columbia University
 *
 * References:
 * Linux source: Documentation/driver-model/platform.txt
 *               drivers/misc/arm-charlcd.c
 * http://www.linuxforu.com/tag/linux-device-drivers/
 * http://free-electrons.com/docs/
 *
 * "make" to build
 * insmod biquad.ko
 *
 * Check code style with
 * checkpatch.pl --file --no-tree biquad.c
 */

#include <linux/module.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/version.h>
#include <linux/kernel.h>
```

```
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/slab.h>
#include <linux/io.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include "biquad.h"

#define DRIVER_NAME "biquad"

/* Device registers */


int z=0;
/*
 * Information about our device
 */
struct biquad_dev {
        struct resource res; /* Resource: our registers */
        void __iomem *virtbase; /* Where registers can be accessed in memory */
        biquad_param_t x;
} dev;

/*
 * Write segments of a single digit
 * Assumes digit is in range and the device information has been set up
 */
static void write_biquad_param(biquad_param_t *x)
{
        for (z=0;z<21;z++){
        iowrite16(x->biquadcoeff[z], (dev.virtbase+2*z) );
        }
    dev.x = *x;

}

/*
 * Handle ioctl() calls from userspace:
 * Read or write the segments on single digits.
 * Note extensive error checking of arguments
 */
static long biquad_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
{
        biquad_arg_t vla;

        switch (cmd) {


        case BIQUAD_WRITE_PARAM:
                if (copy_from_user(&vla, (biquad_arg_t *) arg,
                                   sizeof(biquad_arg_t)))
                        return -EACCES;
                write_biquad_param(&vla.biquadparam);
                break;


        case BIQUAD_READ_PARAM:
                vla.biquadparam = dev.x;
                if (copy_to_user((biquad_arg_t *) arg, &vla,
                                 sizeof(biquad_arg_t)))
				return -EACCES;
			break;

	default:
			return -EINVAL;
	}

	return 0;
}

/* The operations our device knows how to do */
static const struct file_operations biquad_fops = {
	.owner		= THIS_MODULE,
	.unlocked_ioctl = biquad_ioctl,
};

/* Information about our device for the "misc" framework -- like a char dev */
static struct miscdevice biquad_misc_device = {
	.minor		= MISC_DYNAMIC_MINOR,
	.name		= DRIVER_NAME,
	.fops		= &biquad_fops,
};

/*
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
static int __init biquad_probe(struct platform_device *pdev)
{
	biquad_param_t beige = {{ 4096,0,0,0,0,4096,0,0,0,0,4096,0,0,0,0,4096,0,0,0,0,0 }};
	int ret;

	/* Register ourselves as a misc device: creates /dev/biquad */
	ret = misc_register(&biquad_misc_device);

	/* Get the address of our registers from the device tree */
	ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
	if (ret) {
		ret = -ENOENT;
		goto out_deregister;
	}

	/* Make sure we can use these registers */
	if (request_mem_region(dev.res.start, resource_size(&dev.res),
			       DRIVER_NAME) == NULL) {
		ret = -EBUSY;
		goto out_deregister;
	}

	/* Arrange access to our registers */
	dev.virtbase = of_iomap(pdev->dev.of_node, 0);
	if (dev.virtbase == NULL) {
		ret = -ENOMEM;
		goto out_release_mem_region;
	}

	/* Set an initial color */
	write_biquad_param(&beige);

	return 0;

out_release_mem_region:
```

```c
        release_mem_region(dev.res.start, resource_size(&dev.res));
out_deregister:
        misc_deregister(&biquad_misc_device);
        return ret;
}

/* Clean-up code: release resources */
static int biquad_remove(struct platform_device *pdev)
{
        iounmap(dev.virtbase);
        release_mem_region(dev.res.start, resource_size(&dev.res));
        misc_deregister(&biquad_misc_device);
        return 0;
}

/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id biquad_of_match[] = {
        { .compatible = "csee4840,biquad-1.0" },
        {},
};
MODULE_DEVICE_TABLE(of, biquad_of_match);
#endif

/* Information for registering ourselves as a "platform" driver */
static struct platform_driver biquad_driver = {
        .driver = {
                .name   = DRIVER_NAME,
                .owner  = THIS_MODULE,
                .of_match_table = of_match_ptr(biquad_of_match),
        },
        .remove = __exit_p(biquad_remove),
};

/* Called when the module is loaded: set things up */
static int __init biquad_init(void)
{
        pr_info(DRIVER_NAME ": init\n");
        return platform_driver_probe(&biquad_driver, biquad_probe);
}

/* Calball when the module is unloaded: release resources */
static void __exit biquad_exit(void)
{
        platform_driver_unregister(&biquad_driver);
        pr_info(DRIVER_NAME ": exit\n");
}

module_init(biquad_init);
module_exit(biquad_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("LILONGYINIUBI, Columbia University");
MODULE_DESCRIPTION("biquad driver");
```

A.2.3 biquad.h

```c
#ifndef _BIQUAD_H
#define _BIQUAD_H

#include <linux/ioctl.h>

```

```
typedef struct {
        short signed int biquadcoeff[21];
} biquad_param_t;

typedef struct {
  biquad_param_t biquadparam;
} biquad_arg_t;

#define BIQUAD_MAGIC 'q'

/* ioctls and their arguments */
#define BIQUAD_WRITE_PARAM _IOW(BIQUAD_MAGIC, 1, biquad_arg_t *)
#define BIQUAD_READ_PARAM  _IOR(BIQUAD_MAGIC, 2, biquad_arg_t *)

#endif
```

A.2.4 eq_coeff.c

```
/*
 * Academic License - for use in teaching, academic research, and meeting
 * course requirements at degree granting institutions only.  Not for
 * government, commercial, or other organizational use.
 * File: eq_coeff.c
 *
 * MATLAB Coder version            : 5.2
 * C/C++ source code generated on  : 29-Apr-2022 14:21:01
 */

/* Include Files */
#include "eq_coeff.h"
#include "rt_nonfinite.h"
#include "rt_nonfinite.h"
#include <math.h>
#include "biquad.h"
#include <stdio.h>

/* Function Declarations */
static double rt_powd_snf(double u0, double u1);

static double rt_roundd_snf(double u);

/* Function Definitions */
/*
 * Arguments    : double u0
 *                double u1
 * Return Type  : double
 */
static double rt_powd_snf(double u0, double u1)
{
  double d;
  double d1;
  double y;
  if (rtIsNaN(u0) || rtIsNaN(u1)) {
    y = rtNaN;
  } else {
    d = fabs(u0);
    d1 = fabs(u1);
    if (rtIsInf(u1)) {
      if (d == 1.0) {
        y = 1.0;
      } else if (d > 1.0) {
        if (u1 > 0.0) {
          y = rtInf;
        } else {
          y = 0.0;
        }
      } else if (u1 > 0.0) {
        y = 0.0;
      } else {
        y = rtInf;
      }
    } else if (d1 == 0.0) {
      y = 1.0;
    } else if (d1 == 1.0) {
      if (u1 > 0.0) {
        y = u0;
      } else {
        y = 1.0 / u0;
      }
    } else if (u1 == 2.0) {
      y = u0 * u0;
    } else if ((u1 == 0.5) && (u0 >= 0.0)) {
      y = sqrt(u0);
    } else if ((u0 < 0.0) && (u1 > floor(u1))) {
      y = rtNaN;
    } else {
      y = pow(u0, u1);
    }
  }
  return y;
}

/*
 * Arguments    : double u
 * Return Type  : double
 */
static double rt_roundd_snf(double u)
{
  double y;
  if (fabs(u) < 4.503599627370496E+15) {
    if (u >= 0.5) {
      y = floor(u + 0.5);
    } else if (u > -0.5) {
      y = u * 0.0;
    } else {
      y = ceil(u - 0.5);
    }
  } else {
    y = u;
  }
  return y;
}

/*
 * Arguments    : double bass
 *                double mid
 *                double treble
 *                short b[9]
 *                short a[9]
 * Return Type  : void
 */
void eq_coeff(double bass, double mid, double treble, biquad_param_t bi[])
{
  double G;
```

```
    double V0;
    double a1;
    double a2;
    double b0;
    double b1;
    double b2;
    double b_a1;
    double b_a2;
    double b_b0;
    double b_b1;
    double b_b2;
    double c_a1;
    double c_a2;
    double c_b0;
    double c_b1;
    double c_b2;
    short i;
    short a[9];
    short b[9];

    /* bass, mid, treble are integer inputs with 146-285 range */
    G = (bass - 215.5) * 2.0 * 12.0 / -139.0;
    b0 = (mid - 215.5) * 2.0 * 12.0 / -139.0;
    /*  */
    /*  Derive coefficients for a shelving filter with a given amplitude and */
    /*  cutoff frequency.  All coefficients are calculated as described in  */
    /*  Zolzer's DAFX book (p. 50 -55).   */
    /*  */
    /*  Usage:     [B,A] = shelving(G, Fc, Fs, Q, type); */
    /*  */
    /*             G is the logrithmic gain (in dB) */
    /*             FC is the center frequency */
    /*             Fs is the sampling rate */
    /*             Q adjusts the slope be replacing the sqrt(2) term */
    /*             type is a character string defining filter type */
    /*                  Choices are: 'Bass_Shelf' or 'Treble_Shelf' */
    /*  */
    /*  Author:    Jeff Tackett 08/22/05 */
    /*  */
    /* Error Check */
    V0 = rt_powd_snf(10.0, G / 20.0);
    /* sqrt(2) */
    /* Invert gain if a cut */
    if (V0 < 1.0) {
      V0 = 1.0 / V0;
    }
    /* %%%%%%%%%%%%%%%%%%% */
    /*     BASS BOOST */
    /* %%%%%%%%%%%%%%%%%%% */
    if (G > 0.0) {
      G = sqrt(V0) * 1.4142135623730949 * 0.009817792466785694;
      b_b0 = ((G + 1.0) + V0 * 9.6389048920873918E-5) / 1.0139808443080136;
      b1 = 2.0 * (V0 * 9.6389048920873918E-5 - 1.0) / 1.0139808443080136;
      b2 = ((1.0 - G) + V0 * 9.6389048920873918E-5) / 1.0139808443080136;
      a1 = -1.9722337291952663;
      a2 = 0.972613969313063;
      /* %%%%%%%%%%%%%%%%%%% */
      /*     BASS CUT */
      /* %%%%%%%%%%%%%%%%%%% */
    } else if (G < 0.0) {
      G = (1.4142135623730949 * sqrt(V0) * 0.009817792466785694 + 1.0) +
          V0 * 9.6389048920873918E-5;
      b_b0 = 1.0139808443080136 / G;
      b1 = -1.9998072219021583 / G;
      b2 = 0.98621193378982808 / G;
      a1 = 2.0 * (V0 * 9.6389048920873918E-5 - 1.0) / G;
      a2 = ((1.0 - 1.4142135623730949 * sqrt(V0) * 0.0098177924667856 94) +
            V0 * 9.6389048920873918E-5) /
           G;
      /* %%%%%%%%%%%%%%%%%%% */
      /*    TREBLE BOOST */
      /* %%%%%%%%%%%%%%%%%%% */
    } else {
      b_b0 = V0;
      b1 = 0.0;
      b2 = 0.0;
      a1 = 0.0;
      a2 = 0.0;
    }
    /* return values */
    G = (treble - 215.5) * 2.0 * 12.0 / -139.0;
    /*  */
    /*  Derive coefficients for a shelving filter with a given amplitude and */
    /*  cutoff frequency.  All coefficients are calculated as described in  */
    /*  Zolzer's DAFX book (p. 50 -55).   */
    /*  */
    /*  Usage:     [B,A] = shelving(G, Fc, Fs, Q, type); */
    /*  */
    /*             G is the logrithmic gain (in dB) */
    /*             FC is the center frequency */
    /*             Fs is the sampling rate */
    /*             Q adjusts the slope be replacing the sqrt(2) term */
    /*             type is a character string defining filter type */
    /*                  Choices are: 'Bass_Shelf' or 'Treble_Shelf' */
    /*  */
    /*  Author:    Jeff Tackett 08/22/05 */
    /*  */
    /* Error Check */
    V0 = rt_powd_snf(10.0, G / 20.0);
    /* sqrt(2) */
    /* Invert gain if a cut */
    if (V0 < 1.0) {
      V0 = 1.0 / V0;
    }
    /* %%%%%%%%%%%%%%%%%%% */
    /*     BASS BOOST */
    /* %%%%%%%%%%%%%%%%%%% */
    if (G > 0.0) {
      G = 1.4142135623730949 * sqrt(V0) * 0.15838444032453627;
      c_b0 = ((V0 + G) + 0.025085630936916591) / 1.2490750545127478;
      b_b1 = 2.0 * (0.025085630936916591 - V0) / 1.2490750545127478;
      b_b2 = ((V0 - G) + 0.025085630936916591) / 1.2490750545127478;
      b_a1 = -1.5610180758007182;
      b_a2 = 0.64135153805756318;
      /* %%%%%%%%%%%%%%%%%%% */
      /*    TREBLE CUT */
      /* %%%%%%%%%%%%%%%%%%% */
    } else if (G < 0.0) {
      G = (V0 + 1.4142135623730949 * sqrt(V0) * 0.15838444032453627) +
          0.025085630936916591;
      c_b0 = 1.2490750545127478 / G;
      b_b1 = -1.9498287381261667 / G;
      b_b2 = 0.80109620736108533 / G;
      G = 1.4142135623730949 / sqrt(V0) * 0.15838444032453627;
      c_a2 = (G + 1.0) + 0.025085630936916591 / V0;
      b_a1 = 2.0 * (0.025085630936916591 / V0 - 1.0) / c_a2;
      b_a2 = ((1.0 - G) + 0.025085630936916591 / V0) / c_a2;
      /* %%%%%%%%%%%%%%%%%%% */
      /*    All-Pass */
      /* %%%%%%%%%%%%%%%%%%% */
    } else {
      c_b0 = V0;
      b_b1 = 0.0;
      b_b2 = 0.0;
      b_a1 = 0.0;
      b_a2 = 0.0;
    }
    /* return values */
    /*  */
    /*  Derive coefficients for a shelving filter with a given amplitude and */
    /*  cutoff frequency.  All coefficients are calculated as described in  */
    /*  Zolzer's DAFX book (p. 50 -55).   */
    /*  */
    /*  Usage:     [B,A] = shelving(G, Fc, Fs, Q); */
    /*  */
    /*             G is the logrithmic gain (in dB) */
    /*             FC is the center frequency */
    /*             Fs is the sampling rate */
    /*             Q adjusts the slope be replacing the sqrt(2) term */
    /*  */
    /*  Author:    Yuqi Zhu 03/30/22 */
    /*  */
    V0 = rt_powd_snf(10.0, b0 / 20.0);
    /* %%%%%%%%%%%%%%%%%%% */
    /*    BOOST */
    /* %%%%%%%%%%%%%%%%%%% */
    if (b0 > 0.0) {
      G = V0 * 0.03929010700766964 / 0.70710678118654757;
      b0 = ((G + 1.0) + 0.0015437125086741311) / 1.0571083147060107;
      c_b1 = -1.8890330793945247;
      c_b2 = ((1.0 - G) + 0.0015437125086741311) / 1.0571083147060107;
      c_a1 = -1.8890330793945247;
      c_a2 = 0.89487434461663562;
      /* %%%%%%%%%%%%%%%%%%% */
      /*    CUT */
      /* %%%%%%%%%%%%%%%%%%% */
    } else if (b0 < 0.0) {
      G = (0.055564602197336582 / V0 + 1.0) + 0.0015437125086741311;
      b0 = 1.0571083147060107 / G;
      c_b1 = -1.9969125749826517 / G;
      c_b2 = 0.94597911031133752 / G;
      c_a1 = c_b1;
      c_a2 = ((1.0 - 0.03929010700766964 / V0 / 0.70710678118654757) +
              0.0015437125086741311) /
             G;
      /* %%%%%%%%%%%%%%%%%%% */
      /*    All-Pass */
      /* %%%%%%%%%%%%%%%%%%% */
    } else {
      b0 = V0;
      c_b1 = 0.0;
      c_b2 = 0.0;
      c_a1 = 0.0;
      c_a2 = 0.0;
    }
    /* return values */
    a[0] = 4096;
    G = rt_roundd_snf(a1 * 4096.0);
    if (G < 32768.0) {
      i = (short)G;
    } else if (G >= 32768.0) {
      i = MAX_int16_T;
    } else {
      i = 0;
    }
    a[1] = i;
    G = rt_roundd_snf(a2 * 4096.0);
    if (G < 32768.0) {
      if (G >= -32768.0) {
        i = (short)G;
      } else {
        i = MIN_int16_T;
      }
    } else if (G >= 32768.0) {
      i = MAX_int16_T;
    } else {
      i = 0;
    }
    a[2] = i;
    a[3] = 4096;
    G = rt_roundd_snf(c_a1 * 4096.0);
    if (G < 32768.0) {
      i = (short)G;
    } else {
      i = 0;
    }
    a[4] = i;
    G = rt_roundd_snf(c_a2 * 4096.0);
    if (G < 32768.0) {
      if (G >= -32768.0) {
        i = (short)G;
      } else {
        i = MIN_int16_T;
      }
    } else {
      i = 0;
    }
    a[5] = i;
    a[6] = 4096;
    G = rt_roundd_snf(b_a1 * 4096.0);
    if (G < 32768.0) {
      i = (short)G;
    } else {
      i = 0;
    }
    a[7] = i;
    G = rt_roundd_snf(b_a2 * 4096.0);
    if (G < 32768.0) {
      i = (short)G;
    } else {
      i = 0;
    }
    a[8] = i;
    G = rt_roundd_snf(b_b0 * 4096.0);
    if (G < 32768.0) {
      i = (short)G;
    } else if (G >= 32768.0) {
      i = MAX_int16_T;
    } else {
      i = 0;
    }
    b[0] = i;
    G = rt_roundd_snf(b1 * 4096.0);
    if (G < 32768.0) {
      i = (short)G;
    } else if (G >= 32768.0) {
      i = MAX_int16_T;
    } else {
      i = 0;
    }
    b[1] = i;
    G = rt_roundd_snf(b2 * 4096.0);
    if (G < 32768.0) {
      if (G >= -32768.0) {
        i = (short)G;
      } else {
        i = MIN_int16_T;
      }
    } else if (G >= 32768.0) {
      i = MAX_int16_T;
    } else {
      i = 0;
    }
    b[2] = i;
    G = rt_roundd_snf(b0 * 4096.0);
    if (G < 32768.0) {
      i = (short)G;
    } else if (G >= 32768.0) {
      i = MAX_int16_T;
    } else {
      i = 0;
    }
    b[3] = i;
    G = rt_roundd_snf(c_b1 * 4096.0);
    if (G < 32768.0) {
      i = (short)G;
    } else {
      i = 0;
    }
    b[4] = i;
    G = rt_roundd_snf(c_b2 * 4096.0);
    if (G < 32768.0) {
      if (G >= -32768.0) {
        i = (short)G;
      } else {
        i = MIN_int16_T;
      }
    } else {
      i = 0;
    }
    b[5] = i;
    G = rt_roundd_snf(c_b0 * 4096.0);
    if (G < 32768.0) {
      i = (short)G;
    } else if (G >= 32768.0) {
      i = MAX_int16_T;
    } else {
      i = 0;
    }
    b[6] = i;
```

```c
  G = rt_roundd_snf(b_b1 * 4096.0);
  if (G < 32768.0) {
    if (G >= -32768.0) {
      i = (short)G;
    } else {
      i = MIN_int16_T;
    }
  } else {
    i = 0;
  }
  b[7] = i;
  G = rt_roundd_snf(b_b2 * 4096.0);
  if (G < 32768.0) {
    if (G >= -32768.0) {
      i = (short)G;
    } else {
      i = MIN_int16_T;
    }
  } else if (G >= 32768.0) {
    i = MAX_int16_T;
  } else {
    i = 0;
  }
  b[8] = i;
  (bi[0].biquadcoeff)[0] = b[0];
  (bi[0].biquadcoeff)[1] = b[1];
  (bi[0].biquadcoeff)[2] = b[2];
  (bi[0].biquadcoeff)[3] = a[1];
  (bi[0].biquadcoeff)[4] = a[2];
  (bi[0].biquadcoeff)[5] = b[3];
  (bi[0].biquadcoeff)[6] = b[4];
  (bi[0].biquadcoeff)[7] = b[5];
  (bi[0].biquadcoeff)[8] = a[4];
  (bi[0].biquadcoeff)[9] = a[5];
  (bi[0].biquadcoeff)[10] = b[6];
  (bi[0].biquadcoeff)[11] = b[7];
  (bi[0].biquadcoeff)[12] = b[8];
  (bi[0].biquadcoeff)[13] = a[7];
  (bi[0].biquadcoeff)[14] = a[8];
  printf("Biquad:␣");
  for (int i = 0; i <15; i++) printf("%d␣", (bi[0].biquadcoeff)[i]);
  printf("\n");
}

/*
 * File trailer for eq_coeff.c
 *
 * [EOF]
 */
```

A.2.5 eq_coeff.h

```c
/*
 * Academic License - for use in teaching, academic research, and meeting
 * course requirements at degree granting institutions only.  Not for
 * government, commercial, or other organizational use.
 * File: eq_coeff.h
 *
 * MATLAB Coder version            : 5.2
 * C/C++ source code generated on  : 29-Apr-2022 14:21:01
 */

```

```
#ifndef EQ_COEFF_H
#define EQ_COEFF_H

/* Include Files */
#include "rtwtypes.h"
#include <stddef.h>
#include <stdlib.h>
#include "biquad.h"

#ifdef __cplusplus
extern "C" {
#endif

/* Function Declarations */
extern void eq_coeff(double bass, double mid, double treble, biquad_param_t bi[]);

#ifdef __cplusplus
}
#endif

#endif
/*
 * File trailer for eq_coeff.h
 *
 * [EOF]
 */
```

A.2.6 delay.c

```
// Device driver for the  delay module


#include <linux/module.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/slab.h>
#include <linux/io.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include "delay.h"

#define DRIVER_NAME "delay"

/* Device registers */
#define DELAY_BYPASS(x) (x) //bypass, active high
#define DELAY_LENGTH(x) ((x)+2) //number of samples to delay
#define DELAY_MIX(x) ((x)+4) //mixing, 1=-6dB, 2=-12dB, etc.


/*
 * Information about our device
 */
struct delay_dev {
        struct resource res; /* Resource: our registers */
        void __iomem *virtbase; /* Where registers can be accessed in memory */
        delay_param_t x;
} dev;

/*
 * Write segments of a single digit
 * Assumes digit is in range and the device information has been set up
 */
static void write_delay_param(delay_param_t *x)
{
        iowrite16(x->bypass, DELAY_BYPASS(dev.virtbase) );
        iowrite16(x->length, DELAY_LENGTH(dev.virtbase) );
        iowrite16(x->mix, DELAY_MIX(dev.virtbase) );
        dev.x = *x;
}

/*
 * Handle ioctl() calls from userspace:
 * Read or write the segments on single digits.
 * Note extensive error checking of arguments
 */
static long delay_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
{
        delay_arg_t vla;

        switch (cmd) {


        case DELAY_WRITE_PARAM:
                if (copy_from_user(&vla, (delay_arg_t *) arg,
                                   sizeof(delay_arg_t)))
                        return -EACCES;
                write_delay_param(&vla.delayparam);
                break;


        case DELAY_READ_PARAM:
                vla.delayparam = dev.x;
                if (copy_to_user((delay_arg_t *) arg, &vla,
                                 sizeof(delay_arg_t)))
                        return -EACCES;
                break;

        default:
                return -EINVAL;
        }

        return 0;
}

/* The operations our device knows how to do */
static const struct file_operations delay_fops = {
        .owner          = THIS_MODULE,
        .unlocked_ioctl = delay_ioctl,
};

/* Information about our device for the "misc" framework -- like a char dev */
static struct miscdevice delay_misc_device = {
        .minor          = MISC_DYNAMIC_MINOR,
        .name           = DRIVER_NAME,
        .fops           = &delay_fops,
};

/*
```

```
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
static int __init delay_probe(struct platform_device *pdev)
{
        delay_param_t beige = { 1, 4096, 1 };
        int ret;

        /* Register ourselves as a misc device: creates /dev/delay */
        ret = misc_register(&delay_misc_device);

        /* Get the address of our registers from the device tree */
        ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
        if (ret) {
                ret = -ENOENT;
                goto out_deregister;
        }

        /* Make sure we can use these registers */
        if (request_mem_region(dev.res.start, resource_size(&dev.res),
                               DRIVER_NAME) == NULL) {
                ret = -EBUSY;
                goto out_deregister;
        }

        /* Arrange access to our registers */
        dev.virtbase = of_iomap(pdev->dev.of_node, 0);
        if (dev.virtbase == NULL) {
                ret = -ENOMEM;
                goto out_release_mem_region;
        }

        /* Set an initial color */
        write_delay_param(&beige);

        return 0;

out_release_mem_region:
        release_mem_region(dev.res.start, resource_size(&dev.res));
out_deregister:
        misc_deregister(&delay_misc_device);
        return ret;
}

/* Clean-up code: release resources */
static int delay_remove(struct platform_device *pdev)
{
        iounmap(dev.virtbase);
        release_mem_region(dev.res.start, resource_size(&dev.res));
        misc_deregister(&delay_misc_device);
        return 0;
}

/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id delay_of_match[] = {
        { .compatible = "csee4840,delay-1.0" },
        {},
};
MODULE_DEVICE_TABLE(of, delay_of_match);
#endif

```

```c
/* Information for registering ourselves as a "platform" driver */
static struct platform_driver delay_driver = {
        .driver = {
                .name    = DRIVER_NAME,
                .owner   = THIS_MODULE,
                .of_match_table = of_match_ptr(delay_of_match),
        },
        .remove = __exit_p(delay_remove),
};

/* Called when the module is loaded: set things up */
static int __init delay_init(void)
{
        pr_info(DRIVER_NAME ":␣init\n");
        return platform_driver_probe(&delay_driver, delay_probe);
}

/* Calball when the module is unloaded: release resources */
static void __exit delay_exit(void)
{
        platform_driver_unregister(&delay_driver);
        pr_info(DRIVER_NAME ":␣exit\n");
}

module_init(delay_init);
module_exit(delay_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("LILONGYINIUBI,␣Columbia␣University");
MODULE_DESCRIPTION("delay␣driver");
```

A.2.7 delay.h

```c
#ifndef _DELAY_H
#define _DELAY_H

#include <linux/ioctl.h>

typedef struct {
        unsigned short bypass, length, mix;
} delay_param_t;

typedef struct {
  delay_param_t delayparam;
} delay_arg_t;

#define DELAY_MAGIC 'q'

/* ioctls and their arguments */
#define DELAY_WRITE_PARAM _IOW(DELAY_MAGIC, 1, delay_arg_t *)
#define DELAY_READ_PARAM  _IOR(DELAY_MAGIC, 2, delay_arg_t *)

#endif
```

A.2.8 fir.c

```c
// Device driver for the variable fir module


#include <linux/module.h>
#include <linux/init.h>
#include <linux/errno.h>
```

```
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/slab.h>
#include <linux/io.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include "fir.h"

#define DRIVER_NAME "fir"

/* Device registers */


int z=0;
/*
 * Information about our device
 */
struct fir_dev {
        struct resource res; /* Resource: our registers */
        void __iomem *virtbase; /* Where registers can be accessed in memory */
        fir_param_t x;
} dev;

/*
 * Write segments of a single digit
 * Assumes digit is in range and the device information has been set up
 */
static void write_fir_param(fir_param_t *x)
{
        for (z=0;z<500;z++){
        iowrite16(x->fircoeff[z], (dev.virtbase+2*z) );
        }
    dev.x = *x;

}

/*
 * Handle ioctl() calls from userspace:
 * Read or write the segments on single digits.
 * Note extensive error checking of arguments
 */
static long fir_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
{
        fir_arg_t vla;

        switch (cmd) {


        case FIR_WRITE_PARAM:
                if (copy_from_user(&vla, (fir_arg_t *) arg, sizeof(fir_arg_t)))
                        return -EACCES;
                write_fir_param(&vla.firparam);
                break;


        case FIR_READ_PARAM:
                vla.firparam = dev.x;
                if (copy_to_user((fir_arg_t *) arg, &vla, sizeof(fir_arg_t)))
```

```c
                        return -EACCES;
                break;

        default:
                return -EINVAL;
        }

        return 0;
}

/* The operations our device knows how to do */
static const struct file_operations fir_fops = {
        .owner          = THIS_MODULE,
        .unlocked_ioctl = fir_ioctl,
};

/* Information about our device for the "misc" framework -- like a char dev */
static struct miscdevice fir_misc_device = {
        .minor          = MISC_DYNAMIC_MINOR,
        .name           = DRIVER_NAME,
        .fops           = &fir_fops,
};

/*
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
static int __init fir_probe(struct platform_device *pdev)
{
        fir_param_t beige =    {{   15369, 32767, 26902, ..., -356, -359 }}

        int ret;

        /* Register ourselves as a misc device: creates /dev/fir */
        ret = misc_register(&fir_misc_device);

        /* Get the address of our registers from the device tree */
        ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
        if (ret) {
                ret = -ENOENT;
                goto out_deregister;
        }

        /* Make sure we can use these registers */
        if (request_mem_region(dev.res.start, resource_size(&dev.res),
                               DRIVER_NAME) == NULL) {
                ret = -EBUSY;
                goto out_deregister;
        }

        /* Arrange access to our registers */
        dev.virtbase = of_iomap(pdev->dev.of_node, 0);
        if (dev.virtbase == NULL) {
                ret = -ENOMEM;
                goto out_release_mem_region;
        }

        /* Set an initial color */
        write_fir_param(&beige);

        return 0;

```

```
out_release_mem_region:
        release_mem_region(dev.res.start, resource_size(&dev.res));
out_deregister:
        misc_deregister(&fir_misc_device);
        return ret;
}

/* Clean-up code: release resources */
static int fir_remove(struct platform_device *pdev)
{
        iounmap(dev.virtbase);
        release_mem_region(dev.res.start, resource_size(&dev.res));
        misc_deregister(&fir_misc_device);
        return 0;
}

/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id fir_of_match[] = {
        { .compatible = "fir,fir-21.1" },
        {},
};
MODULE_DEVICE_TABLE(of, fir_of_match);
#endif

/* Information for registering ourselves as a "platform" driver */
static struct platform_driver fir_driver = {
        .driver = {
                .name   = DRIVER_NAME,
                .owner  = THIS_MODULE,
                .of_match_table = of_match_ptr(fir_of_match),
        },
        .remove = __exit_p(fir_remove),
};

/* Called when the module is loaded: set things up */
static int __init fir_init(void)
{
        pr_info(DRIVER_NAME ":␣init\n");
        return platform_driver_probe(&fir_driver, fir_probe);
}

/* Calball when the module is unloaded: release resources */
static void __exit fir_exit(void)
{
        platform_driver_unregister(&fir_driver);
        pr_info(DRIVER_NAME ":␣exit\n");
}

module_init(fir_init);
module_exit(fir_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("LILONGYINIUBI,␣Columbia␣University");
MODULE_DESCRIPTION("fir␣driver");
```

A.2.9 fir.h

```
#ifndef _FIR_H
#define _FIR_H

#include <linux/ioctl.h>
```

```c

typedef struct {
        short signed int fircoeff[500];
} fir_param_t;

typedef struct {
  fir_param_t firparam;
} fir_arg_t;

#define FIR_MAGIC 'q'

/* ioctls and their arguments */
#define FIR_WRITE_PARAM _IOW(FIR_MAGIC, 1, fir_arg_t *)
#define FIR_READ_PARAM  _IOR(FIR_MAGIC, 2, fir_arg_t *)

#endif
```

A.2.10 hardclip.c

```c
// Device driver for the variable hardclip module


#include <linux/module.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/slab.h>
#include <linux/io.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include "hardclip.h"

#define DRIVER_NAME "hardclip"

/* Device registers */
#define HARDCLIP_POSCLIP(x) ((x))
#define HARDCLIP_NEGCLIP(x) ((x)+2)


/*
 * Information about our device
 */
struct hardclip_dev {
        struct resource res; /* Resource: our registers */
        void __iomem *virtbase; /* Where registers can be accessed in memory */
        hardclip_param_t x;
} dev;

/*
 * Write segments of a single digit
 * Assumes digit is in range and the device information has been set up
 */
static void write_hardclip_param(hardclip_param_t *x)
{
        iowrite16(x->posclip, HARDCLIP_POSCLIP(dev.virtbase) );
        iowrite16(x->negclip, HARDCLIP_NEGCLIP(dev.virtbase) );
        dev.x = *x;
```

```c
}

/*
 * Handle ioctl() calls from userspace:
 * Read or write the segments on single digits.
 * Note extensive error checking of arguments
 */
static long hardclip_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
{
        hardclip_arg_t vla;

        switch (cmd) {


        case HARDCLIP_WRITE_PARAM:
                if (copy_from_user(&vla, (hardclip_arg_t *) arg,
                                   sizeof(hardclip_arg_t)))
                        return -EACCES;
                write_hardclip_param(&vla.hardclipparam);
                break;


        case HARDCLIP_READ_PARAM:
                vla.hardclipparam = dev.x;
                if (copy_to_user((hardclip_arg_t *) arg, &vla,
                                 sizeof(hardclip_arg_t)))
                        return -EACCES;
                break;

        default:
                return -EINVAL;
        }

        return 0;
}

/* The operations our device knows how to do */
static const struct file_operations hardclip_fops = {
        .owner          = THIS_MODULE,
        .unlocked_ioctl = hardclip_ioctl,
};

/* Information about our device for the "misc" framework -- like a char dev */
static struct miscdevice hardclip_misc_device = {
        .minor          = MISC_DYNAMIC_MINOR,
        .name           = DRIVER_NAME,
        .fops           = &hardclip_fops,
};

/*
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
static int __init hardclip_probe(struct platform_device *pdev)
{
        hardclip_param_t beige = { 32767, -32767 };
        int ret;

        /* Register ourselves as a misc device: creates /dev/hardclip */
        ret = misc_register(&hardclip_misc_device);

        /* Get the address of our registers from the device tree */
        ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
        if (ret) {
                ret = -ENOENT;
                goto out_deregister;
        }

        /* Make sure we can use these registers */
        if (request_mem_region(dev.res.start, resource_size(&dev.res),
                               DRIVER_NAME) == NULL) {
                ret = -EBUSY;
                goto out_deregister;
        }

        /* Arrange access to our registers */
        dev.virtbase = of_iomap(pdev->dev.of_node, 0);
        if (dev.virtbase == NULL) {
                ret = -ENOMEM;
                goto out_release_mem_region;
        }

        /* Set an initial color */
        write_hardclip_param(&beige);

        return 0;

out_release_mem_region:
        release_mem_region(dev.res.start, resource_size(&dev.res));
out_deregister:
        misc_deregister(&hardclip_misc_device);
        return ret;
}

/* Clean-up code: release resources */
static int hardclip_remove(struct platform_device *pdev)
{
        iounmap(dev.virtbase);
        release_mem_region(dev.res.start, resource_size(&dev.res));
        misc_deregister(&hardclip_misc_device);
        return 0;
}

/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id hardclip_of_match[] = {
        { .compatible = "csee4840,hardclip-1.0" },
        {},
};
MODULE_DEVICE_TABLE(of, hardclip_of_match);
#endif

/* Information for registering ourselves as a "platform" driver */
static struct platform_driver hardclip_driver = {
        .driver = {
                .name   = DRIVER_NAME,
                .owner  = THIS_MODULE,
                .of_match_table = of_match_ptr(hardclip_of_match),
        },
        .remove = __exit_p(hardclip_remove),
};

/* Called when the module is loaded: set things up */
static int __init hardclip_init(void)
```

```
{
        pr_info(DRIVER_NAME ":␣init\n");
        return platform_driver_probe(&hardclip_driver, hardclip_probe);
}

/* Calball when the module is unloaded: release resources */
static void __exit hardclip_exit(void)
{
        platform_driver_unregister(&hardclip_driver);
        pr_info(DRIVER_NAME ":␣exit\n");
}

module_init(hardclip_init);
module_exit(hardclip_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("LILONGYINIUBI,␣Columbia␣University");
MODULE_DESCRIPTION("hardclip␣driver");
```

A.2.11 hardclip.h

```
#ifndef _HARDCLIP_H
#define _HARDCLIP_H

#include <linux/ioctl.h>

typedef struct {
        short signed int posclip, negclip;
} hardclip_param_t;

typedef struct {
  hardclip_param_t hardclipparam;
} hardclip_arg_t;

#define HARDCLIP_MAGIC 'q'

/* ioctls and their arguments */
#define HARDCLIP_WRITE_PARAM _IOW(HARDCLIP_MAGIC, 1, hardclip_arg_t *)
#define HARDCLIP_READ_PARAM  _IOR(HARDCLIP_MAGIC, 2, hardclip_arg_t *)

#endif
```

A.2.12 hello.c

```
/*
 * Userspace program that communicates with the delay device driver
 * through ioctls
 *
 * Stephen A. Edwards
 * Columbia University
 */

#include <stdio.h>
#include "delay.h"
#include "hardclip.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include "biquad.h"
```

```
#include "fir.h"
#include "vga_ball.h"
#include "eq_coeff.h"

#include <stdlib.h>
#include "usbkeyboard.h"
#include <pthread.h>
#include "keycode.h"
#define KEY_ENTER 0x28
#define KEY_SPACE 0x2c
#define KEY_BACKSPACE 0x2a
#define KEY_RIGHT 0x4f
#define KEY_LEFT 0x50
#define KEY_1 0x1e
#define KEY_2 0x1f
#define KEY_3 0x20
#define KEY_4 0x21
#define KEY_Q 0x14
#define KEY_A 0x04
#define KEY_W 0x1a
#define KEY_S 0x16
#define KEY_E 0x08
#define KEY_D 0x07
#define KEY_R 0x15
#define KEY_F 0X09
#define KEY_T 0x17
#define KEY_G 0x0a
#define KEY_Y 0x1c
#define KEY_H 0x0b
#define KEY_X 0x1b
#define KEY_Z 0x1d
#define KEY_B 0x05
#define KEY_N 0x11

int delay_fd;
int hardclip_fd;
int biquad_fd;
int fir_fd;
int vga_ball_fd;


struct libusb_device_handle *keyboard;
uint8_t endpoint_address;


uint8_t prevkey_1;
uint8_t prevkey_2;

//preset locations
short unsigned int  preset1=132;
short unsigned int  preset2=188;
short unsigned int  preset3=245;
short unsigned int  preset4=304;
char clippreset=5;

//slider locations
short unsigned int  slider1=215;
short unsigned int  slider2=215;
short unsigned int  slider3=215;
short unsigned int  slider4=215;
short unsigned int  slider5=215;
short unsigned int  slider6=215;
```

```



//LUTS
static const char filename_delay[] = "/dev/delay";
delay_param_t delayparam =     { 0x0000, 4000, 8192  };//bypass, samples, mix
static const char filename_hardclip[] = "/dev/hardclip";
static const hardclip_param_t hardclipparam[] = {//lookup table for hardclip threshold
    {32767,-32767 },   {2750,-2750 },    {2500,-2500 },     {2250,-2250 },     {2000,-2000 },
{1750,-1750 },   {1500,-1500 },     {1250,-1250 },     {1000,-1000 },     {750,-750 },
{500,-500 },  };
static const char filename_biquad[] = "/dev/biquad";
biquad_param_t biquadparam[] = {
    {{ 1024, 0, 0, 0, 0,1024,0,0,0,0, 1024,0,0,0,0, 1024,0,0,0,0,1 }},
    {{ 32767,0,0,0,0,32767,0,0,0,0,32767,0,0,0,0,32767,0,0,0,0,0 }}

  };
static const char filename_fir[] = "/dev/fir";
static const fir_param_t firparam[] = {
    {{  15369, 32767, 26902, ..., 81, 13}},
    {{  0, 0, 0, ..., 81, 13}},
    {{  15369, 32767, 26902, ..., 81, 13}},
    {{  0, 32767, 0, ..., 0, 0}},
  };
static const char filename[] = "/dev/vga_ball";
vga_ball_color_t colors= { 0,215, 215,215,215,215,215,215};

void setup(void){

    if ( (delay_fd = open(filename_delay, O_RDWR)) == -1) {
    fprintf(stderr, "could␣not␣open␣%s\n", filename_delay);
    return -1;
  }
  if ( (hardclip_fd = open(filename_hardclip, O_RDWR)) == -1) {
    fprintf(stderr, "could␣not␣open␣%s\n", filename_hardclip);
    return -1;
  }

  if ( (biquad_fd = open(filename_biquad, O_RDWR)) == -1) {
    fprintf(stderr, "could␣not␣open␣%s\n", filename_biquad);
    return -1;
  }
  if ( (fir_fd = open(filename_fir, O_RDWR)) == -1) {
    fprintf(stderr, "could␣not␣open␣%s\n", filename_fir);
    return -1;
  }
  if ( (vga_ball_fd = open(filename, O_RDWR)) == -1) {
    fprintf(stderr, "could␣not␣open␣%s\n", filename);
    return -1;
  }
}

void set_background_color(const vga_ball_color_t *c)
{
  vga_ball_arg_t vla;
  vla.background = *c;
  if (ioctl(vga_ball_fd, VGA_BALL_WRITE_BACKGROUND, &vla)) {
      perror("ioctl(VGA_BALL_SET_BACKGROUND)␣failed");
      return;
  }
  printf("Sliders:␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d\n", c->bypass, c->sely,
  c->sly1,c->sly2,c->sly3,c->sly4,c->sly5,c->sly6);
```

```
}

void set_hardclip_param(const hardclip_param_t *c)
{
  hardclip_arg_t hardclipvla;
  hardclipvla.hardclipparam = *c;
  if (ioctl(hardclip_fd, HARDCLIP_WRITE_PARAM, &hardclipvla)) {
      perror("ioctl(HARDCLIP_WRITE_PARAM)␣failed");
      return;
  }
}

void set_delay_param(const delay_param_t *c)
{
  delay_arg_t delayvla;
  delayvla.delayparam = *c;
  if (ioctl(delay_fd, DELAY_WRITE_PARAM, &delayvla)) {
      perror("ioctl(DELAY_WRITE_PARAM)␣failed");
      return;
  }
}
void set_biquad_param(const biquad_param_t *c)
{
  biquad_arg_t biquadvla;
  biquadvla.biquadparam = *c;
  if (ioctl(biquad_fd, BIQUAD_WRITE_PARAM, &biquadvla)) {
      perror("ioctl(BIQUAD_WRITE_PARAM)␣failed");
      return;
  }
}
void set_fir_param(const fir_param_t *c)
{
  fir_arg_t firvla;
  firvla.firparam = *c;
  if (ioctl(fir_fd, FIR_WRITE_PARAM, &firvla)) {
      perror("ioctl(FIR_WRITE_PARAM)␣failed");
      return;
  }
}


int main()
{
        struct usb_keyboard_packet packet;
        int transferred;
        setup();

        /* Open the keyboard */
        if ( (keyboard = openkeyboard(&endpoint_address)) == NULL ) {
                fprintf(stderr, "Did␣not␣find␣a␣keyboard\n");
                exit(1);
        }
        set_biquad_param(biquadparam);

        /* Look for and handle keypresses */
        for (;;) {
                libusb_interrupt_transfer(keyboard, endpoint_address,
                        (unsigned char *) &packet, sizeof(packet),
                &transferred, 0);
                if (transferred == sizeof(packet)) {

                        if (packet.keycode[0] == 0 )
```

```
{
}
else if (packet.keycode[0] == prevkey_1 && packet.keycode[1] == 0)
{
}
else if ( packet.keycode[0] == prevkey_2 )
{

}
else if (packet.keycode[0] == 0x29) { // if press esc

}
else  if (packet.keycode[0] == KEY_1) { //if enter pressed
        colors.sely=preset1;
        set_background_color(&colors);
        set_fir_param(&firparam[0]);

}
else  if (packet.keycode[0] == KEY_2) { //if enter pressed
        colors.sely=preset2;
        set_background_color(&colors);
        set_fir_param(&firparam[1]);

}
else  if (packet.keycode[0] == KEY_3) { //if enter pressed
        colors.sely=preset3;
        set_background_color(&colors);
        set_fir_param(&firparam[2]);

}
else  if (packet.keycode[0] == KEY_4) { //if enter pressed
        colors.sely=preset4;
        set_background_color(&colors);
        set_fir_param(&firparam[3]);

}
else  if (packet.keycode[0] == KEY_BACKSPACE) { //if backspace

}

else  if (packet.keycode[0] == KEY_Q) { //HARDCLIP++
        if(slider1>146){
                slider1-=14;
        clippreset-=1;}
        colors.sly1=slider1;
        set_background_color(&colors);
        set_hardclip_param(&hardclipparam[clippreset]);

}
else  if (packet.keycode[0] == KEY_A) {//HARDCLIP--
        if(slider1<285){
                slider1+=14;
                                        clippreset+=1;}

        colors.sly1=slider1;
        set_background_color(&colors);
        set_hardclip_param(&hardclipparam[clippreset]);

}

else  if (packet.keycode[0] == KEY_W) { //DELAY++
        if(slider2>146){
```

```
                        slider2-=14;
                        delayparam.length+=700;
                }
                colors.sly2=slider2;
                set_background_color(&colors);
                set_delay_param(&delayparam);
        }
        else  if (packet.keycode[0] == KEY_S) { //DELAY--
                if(slider2<285){
                        slider2+=14;
                        delayparam.length-=700;
                }
                colors.sly2=slider2;
                set_background_color(&colors);
                set_delay_param(&delayparam);
        }
        else  if (packet.keycode[0] == KEY_B) { //BYPASS DELAY
                delayparam.bypass=1;
                colors.bypass=0;
                set_background_color(&colors);
                set_delay_param(&delayparam);
        }
        else  if (packet.keycode[0] == KEY_N) { //ENABLE DELAY
                delayparam.bypass=0;
                colors.bypass=1;
                set_background_color(&colors);
                set_delay_param(&delayparam);
        }

        else  if (packet.keycode[0] == KEY_E) { //MIX++
                if(slider3>146){
                        slider3-=14;
                        delayparam.mix+=1500;
                }
                colors.sly3=slider3;
                set_background_color(&colors);
                set_delay_param(&delayparam);
        }
        else  if (packet.keycode[0] == KEY_D) { //MIX--
                if(slider3<285){
                        slider3+=14;
                        delayparam.mix-=1500;
                }
                colors.sly3=slider3;
                set_delay_param(&delayparam);
                set_background_color(&colors);
        }

        else  if (packet.keycode[0] == KEY_R) { //BASS++
                if(slider4>146){
                        slider4-=14;
                        eq_coeff((double)slider4, (double)slider5,
            (double)slider6, biquadparam);
                }
                colors.sly4=slider4;
                set_background_color(&colors);
                set_biquad_param(biquadparam);
        }
        else  if (packet.keycode[0] == KEY_F) { //BASS--
                if(slider4<285){
                        eq_coeff((double)slider4, (double)slider5,
            (double)slider6, biquadparam);
```

```
                }
                colors.sly4=slider4;
                set_background_color(&colors);
                set_biquad_param(biquadparam);
        }

        else  if (packet.keycode[0] == KEY_T) { //MID++
                if(slider5>146){
                        slider5-=14;
                        eq_coeff((double)slider4, (double)slider5,
            (double)slider6, biquadparam);
                }
                colors.sly5=slider5;
                set_background_color(&colors);
                set_biquad_param(biquadparam);
        }
        else  if (packet.keycode[0] == KEY_G) { //MID--
                if(slider5<285){
                        slider5+=14;
                        eq_coeff((double)slider4, (double)slider5,
            (double)slider6, biquadparam);
                }
                colors.sly5=slider5;
                set_background_color(&colors);
                set_biquad_param(biquadparam);
        }

        else  if (packet.keycode[0] == KEY_Y) { //TREBLE++
                if(slider6>146){
                        slider6-=14;
                        eq_coeff((double)slider4, (double)slider5,
            (double)slider6, biquadparam);
                }
                colors.sly6=slider6;
                set_background_color(&colors);
                set_biquad_param(biquadparam);
        }
        else  if (packet.keycode[0] == KEY_H) { //TREBLE--
                if(slider6<285){
                        slider6+=14;
                        eq_coeff((double)slider4, (double)slider5,
            (double)slider6, biquadparam);
                }
                colors.sly6=slider6;
                set_background_color(&colors);
                set_biquad_param(biquadparam);
        }

        else  if (packet.keycode[0] == KEY_Z) { //BYPASS EQ
                biquadparam[0].biquadcoeff[20]=1;
                set_background_color(&colors);
                set_biquad_param(biquadparam);
        }
        else  if (packet.keycode[0] == KEY_X) { //ENABLE EQ
                biquadparam[0].biquadcoeff[20]=0;
                set_background_color(&colors);
                set_biquad_param(biquadparam);
        }


        else  if (packet.keycode[0] == KEYU) { //BYPASS EQ
                if(biquadparam[0].biquadcoeff[15] < 16000)
```

```
                                                biquadparam[0].biquadcoeff[15] =
                                          (short)(((float)biquadparam[0].biquadcoeff[15]) * 1.2);
                                        set_background_color(&colors);
                                        set_biquad_param(biquadparam);
                                }
                                else  if (packet.keycode[0] == KEYJ) { //ENABLE EQ
                                        if(biquadparam[0].biquadcoeff[15] > 100)
                                                biquadparam[0].biquadcoeff[15] =
                                           (short)(((float)biquadparam[0].biquadcoeff[15]) / 1.2);
                                        set_background_color(&colors);
                                        set_biquad_param(biquadparam);
                                }




                                prevkey_1 = packet.keycode[0];
                                prevkey_2 = packet.keycode[1];
                        }
                }


                return 0;
}
```

A.2.13 usbkeyboard.c

```
#include "usbkeyboard.h"

#include <stdio.h>
#include <stdlib.h>

/* References on libusb 1.0 and the USB HID/keyboard protocol
 *
 * http://libusb.org
 * http://www.dreamincode.net/forums/topic/148707-introduction-to-using-libusb-10/
 * http://www.usb.org/developers/devclass_docs/HID1_11.pdf
 * http://www.usb.org/developers/devclass_docs/Hut1_11.pdf
 */

/*
 * Find and return a USB keyboard device or NULL if not found
 * The argument con
 *
 */
struct libusb_device_handle *openkeyboard(uint8_t *endpoint_address) {
  libusb_device **devs;
  struct libusb_device_handle *keyboard = NULL;
  struct libusb_device_descriptor desc;
  ssize_t num_devs, d;
  uint8_t i, k;

  /* Start the library */
  if ( libusb_init(NULL) < 0 ) {
    fprintf(stderr, "Error:␣libusb_init␣failed\n");
    exit(1);
  }

  /* Enumerate all the attached USB devices */
  if ( (num_devs = libusb_get_device_list(NULL, &devs)) < 0 ) {
    fprintf(stderr, "Error:␣libusb_get_device_list␣failed\n");
    exit(1);
  }

  /* Look at each device, remembering the first HID device that speaks
     the keyboard protocol */

  for (d = 0 ; d < num_devs ; d++) {
    libusb_device *dev = devs[d];
    if ( libusb_get_device_descriptor(dev, &desc) < 0 ) {
      fprintf(stderr, "Error:␣libusb_get_device_descriptor␣failed\n");
      exit(1);
    }

    if (desc.bDeviceClass == LIBUSB_CLASS_PER_INTERFACE) {
      struct libusb_config_descriptor *config;
      libusb_get_config_descriptor(dev, 0, &config);
      for (i = 0 ; i < config->bNumInterfaces ; i++)
        for ( k = 0 ; k < config->interface[i].num_altsetting ; k++ ) {
          const struct libusb_interface_descriptor *inter =
            config->interface[i].altsetting + k ;
          if ( inter->bInterfaceClass == LIBUSB_CLASS_HID &&
               inter->bInterfaceProtocol == USB_HID_KEYBOARD_PROTOCOL) {
            int r;
            if ((r = libusb_open(dev, &keyboard)) != 0) {
              fprintf(stderr, "Error:␣libusb_open␣failed:␣%d\n", r);
              exit(1);
            }
            if (libusb_kernel_driver_active(keyboard,i))
              libusb_detach_kernel_driver(keyboard, i);
            libusb_set_auto_detach_kernel_driver(keyboard, i);
            if ((r = libusb_claim_interface(keyboard, i)) != 0) {
              fprintf(stderr, "Error:␣libusb_claim_interface␣failed:␣%d\n", r);
              exit(1);
            }
            *endpoint_address = inter->endpoint[0].bEndpointAddress;
            goto found;
          }
        }
    }
  }

 found:
  libusb_free_device_list(devs, 1);

  return keyboard;
}
```

A.2.14 usbkeyboard.h

```
#ifndef _USBKEYBOARD_H
#define _USBKEYBOARD_H

#include <libusb-1.0/libusb.h>

#define USB_HID_KEYBOARD_PROTOCOL 1

/* Modifier bits */
#define USB_LCTRL  (1 << 0)
#define USB_LSHIFT (1 << 1)
#define USB_LALT   (1 << 2)
#define USB_LGUI   (1 << 3)
#define USB_RCTRL  (1 << 4)
#define USB_RSHIFT (1 << 5)
```

```
#define USB_RALT   (1 << 6)
#define USB_RGUI   (1 << 7)

struct usb_keyboard_packet {
  uint8_t modifiers;
  uint8_t reserved;
  uint8_t keycode[6];
};

/* Find and open a USB keyboard device.  Argument should point to
   space to store an endpoint address.  Returns NULL if no keyboard
   device was found. */
extern struct libusb_device_handle *openkeyboard(uint8_t *);

#endif
```

A.2.15 vga_ball.c

```
//Device driver for the VGA video generator


#include <linux/module.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/slab.h>
#include <linux/io.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include "vga_ball.h"

#define DRIVER_NAME "vga_ball"

/* Device registers */
#define BYPASS(x) (x) //bypass switch on/off (0/1)
#define SELY(x) ((x)+2) //FIR PRESET SELECT LOC Y
#define SLY1(x) ((x)+4)//SLIDER 1 CLIP
#define SLY2(x) ((x)+6)//SLIDER 2 DELAY
#define SLY3(x) ((x)+8)//SLIDER 3 MIX
#define SLY4(x) ((x)+10)//SLIDER 4 BASS
#define SLY5(x) ((x)+12)//SLIDER 5 MID
#define SLY6(x) ((x)+14)//SLIDER 6 TRE

/*
 * Information about our device
 */
struct vga_ball_dev {
        struct resource res; /* Resource: our registers */
        void __iomem *virtbase; /* Where registers can be accessed in memory */
        vga_ball_color_t background;
} dev;

/*
 * Write segments of a single digit
 * Assumes digit is in range and the device information has been set up
 */
static void write_background(vga_ball_color_t *background)
```

```
{
        iowrite16(background->bypass, BYPASS(dev.virtbase) );
        iowrite16(background->sely, SELY(dev.virtbase) );
        iowrite16(background->sly1, SLY1(dev.virtbase) );
        iowrite16(background->sly2, SLY2(dev.virtbase) );
        iowrite16(background->sly3, SLY3(dev.virtbase) );
        iowrite16(background->sly4, SLY4(dev.virtbase) );
        iowrite16(background->sly5, SLY5(dev.virtbase) );
        iowrite16(background->sly6, SLY6(dev.virtbase) );
        dev.background = *background;
}

/*
 * Handle ioctl() calls from userspace:
 * Read or write the segments on single digits.
 * Note extensive error checking of arguments
 */
static long vga_ball_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
{
        vga_ball_arg_t vla;

        switch (cmd) {


        case VGA_BALL_WRITE_BACKGROUND:
                if (copy_from_user(&vla, (vga_ball_arg_t *) arg,
                                   sizeof(vga_ball_arg_t)))
                        return -EACCES;
                write_background(&vla.background);
                break;



        case VGA_BALL_READ_BACKGROUND:
                vla.background = dev.background;
                if (copy_to_user((vga_ball_arg_t *) arg, &vla,
                                 sizeof(vga_ball_arg_t)))
                        return -EACCES;
                break;

        default:
                return -EINVAL;
        }

        return 0;
}

/* The operations our device knows how to do */
static const struct file_operations vga_ball_fops = {
        .owner          = THIS_MODULE,
        .unlocked_ioctl = vga_ball_ioctl,
};

/* Information about our device for the "misc" framework -- like a char dev */
static struct miscdevice vga_ball_misc_device = {
        .minor          = MISC_DYNAMIC_MINOR,
        .name           = DRIVER_NAME,
        .fops           = &vga_ball_fops,
};

/*
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
static int __init vga_ball_probe(struct platform_device *pdev)
{
        vga_ball_color_t beige = { 0,215, 215,215,215,215,215,215        };
        int ret;

        /* Register ourselves as a misc device: creates /dev/vga_ball */
        ret = misc_register(&vga_ball_misc_device);

        /* Get the address of our registers from the device tree */
        ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
        if (ret) {
                ret = -ENOENT;
                goto out_deregister;
        }

        /* Make sure we can use these registers */
        if (request_mem_region(dev.res.start, resource_size(&dev.res),
                               DRIVER_NAME) == NULL) {
                ret = -EBUSY;
                goto out_deregister;
        }

        /* Arrange access to our registers */
        dev.virtbase = of_iomap(pdev->dev.of_node, 0);
        if (dev.virtbase == NULL) {
                ret = -ENOMEM;
                goto out_release_mem_region;
        }

        /* Set an initial color */
        write_background(&beige);

        return 0;

out_release_mem_region:
        release_mem_region(dev.res.start, resource_size(&dev.res));
out_deregister:
        misc_deregister(&vga_ball_misc_device);
        return ret;
}

/* Clean-up code: release resources */
static int vga_ball_remove(struct platform_device *pdev)
{
        iounmap(dev.virtbase);
        release_mem_region(dev.res.start, resource_size(&dev.res));
        misc_deregister(&vga_ball_misc_device);
        return 0;
}

/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id vga_ball_of_match[] = {
        { .compatible = "csee4840,vga_ball-1.0" },
        {},
};
MODULE_DEVICE_TABLE(of, vga_ball_of_match);
#endif

/* Information for registering ourselves as a "platform" driver */
```

```
static struct platform_driver vga_ball_driver = {
        .driver = {
                .name    = DRIVER_NAME,
                .owner   = THIS_MODULE,
                .of_match_table = of_match_ptr(vga_ball_of_match),
        },
        .remove = __exit_p(vga_ball_remove),
};

/* Called when the module is loaded: set things up */
static int __init vga_ball_init(void)
{
        pr_info(DRIVER_NAME ": init\n");
        return platform_driver_probe(&vga_ball_driver, vga_ball_probe);
}

/* Calball when the module is unloaded: release resources */
static void __exit vga_ball_exit(void)
{
        platform_driver_unregister(&vga_ball_driver);
        pr_info(DRIVER_NAME ": exit\n");
}

module_init(vga_ball_init);
module_exit(vga_ball_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Stephen A. Edwards, Columbia University");
MODULE_DESCRIPTION("VGA ball driver");
```

A.2.16 vga_ball.h

```
#ifndef _VGA_BALL_H
#define _VGA_BALL_H

#include <linux/ioctl.h>

typedef struct {
        short unsigned int bypass, sely, sly1,sly2,sly3,sly4,sly5,sly6;
} vga_ball_color_t;



typedef struct {
  vga_ball_color_t background;
} vga_ball_arg_t;

#define VGA_BALL_MAGIC 'q'

/* ioctls and their arguments */
#define VGA_BALL_WRITE_BACKGROUND _IOW(VGA_BALL_MAGIC, 1, vga_ball_arg_t *)
#define VGA_BALL_READ_BACKGROUND  _IOR(VGA_BALL_MAGIC, 2, vga_ball_arg_t *)

#endif
```