

# 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

<b>1</b>	<b>Overview</b>	<b>2</b>
1.1	Introduction . . . . .	2
1.2	Milestones . . . . .	2
1.3	Work Statement . . . . .	2
<b>2</b>	<b>Implementation</b>	<b>2</b>
2.1	System Architecture . . . . .	2
2.2	Hardware . . . . .	3
2.2.1	Analog Pre-Amplifier . . . . .	3
2.2.2	I2S Interface . . . . .	3
2.2.3	Cabinet IR . . . . .	4
2.2.4	Biquad Chain . . . . .	5
2.2.5	Limiter . . . . .	5
2.2.6	Delay . . . . .	6
2.3	VGA . . . . .	6
<b>3</b>	<b>Resource Budget</b>	<b>7</b>
<b>4</b>	<b>Hardware/Software Interface</b>	<b>7</b>
4.1	Cabinet IR . . . . .	7
4.2	Biquad Chain . . . . .	7
4.3	Limiter . . . . .	7
4.4	Delay . . . . .	7
4.5	VGA Ball . . . . .	7
<b>A</b>	<b>Design Files</b>	<b>9</b>
A.1	Hardware . . . . .	9
A.1.1	soc_system.sv . . . . .	9
A.1.2	vga_ball.sv . . . . .	14
A.1.3	i2s_avalon_st.v . . . . .	18
A.1.4	hardclip.v . . . . .	22
A.1.5	delay.v . . . . .	23
A.1.6	biquad.v . . . . .	25
A.2	Software . . . . .	29
A.2.1	Makefile . . . . .	29
A.2.2	biquad.c . . . . .	30
A.2.3	biquad.h . . . . .	33
A.2.4	eq.coeff.c . . . . .	34
A.2.5	eq.coeff.h . . . . .	41
A.2.6	delay.c . . . . .	42
A.2.7	delay.h . . . . .	45
A.2.8	fir.c . . . . .	45
A.2.9	fir.h . . . . .	48
A.2.10	hardclip.c . . . . .	49
A.2.11	hardclip.h . . . . .	52
A.2.12	hello.c . . . . .	52
A.2.13	usbkeyboard.c . . . . .	59
A.2.14	usbkeyboard.h . . . . .	60
A.2.15	vga_ball.c . . . . .	61
A.2.16	vga_ball.h . . . . .	64

# 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.

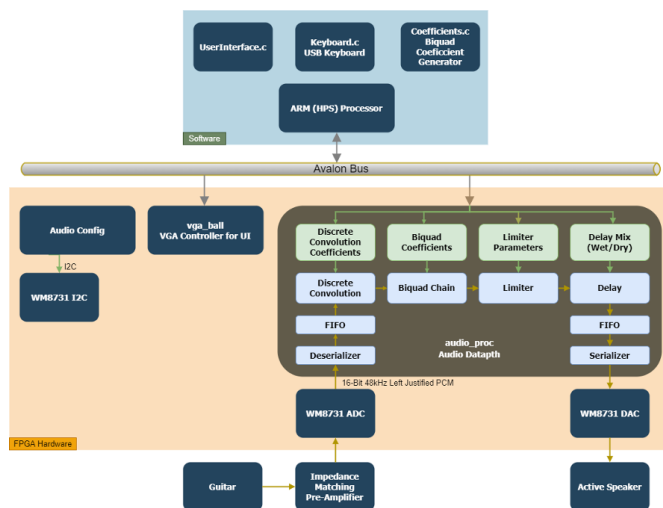


Figure 1: System Architecture

## 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.

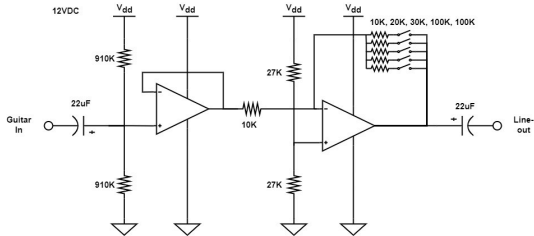


Figure 2: Pre-Amp Schematic

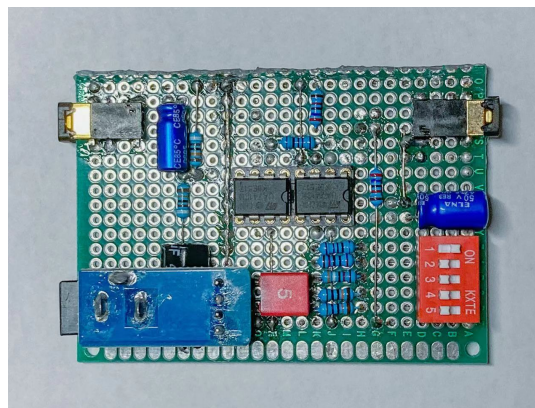


Figure 3: Pre-Amp Hardware

### 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.

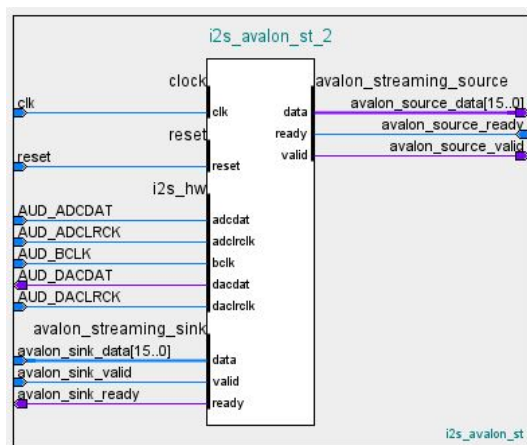


Figure 4: I2S Interface Module

### 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.

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

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.

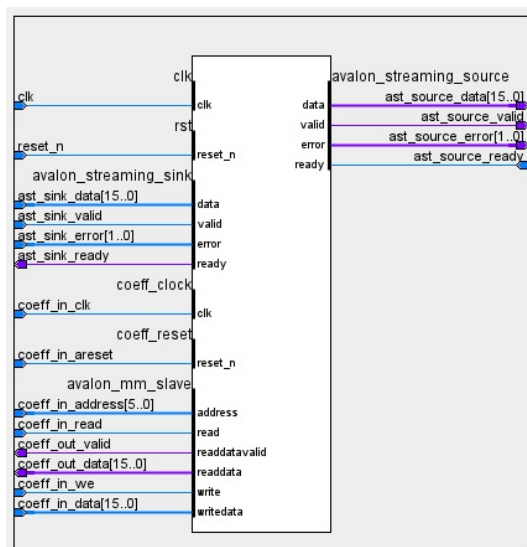


Figure 5: FIR Module

### 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.

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1Z^{-1} + b_2Z^{-2}}{1 + a_1Z^{-1} + a_2Z^{-2}} \quad (2)$$

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

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.

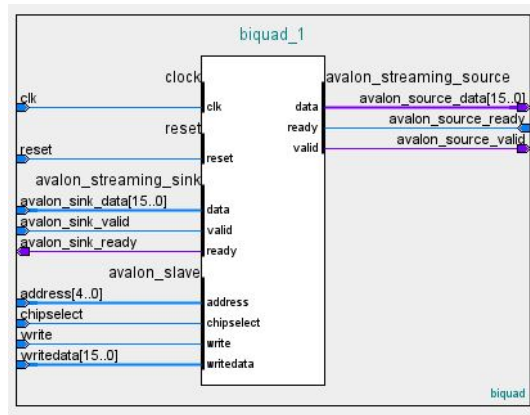


Figure 6: Biquad Module

### 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.

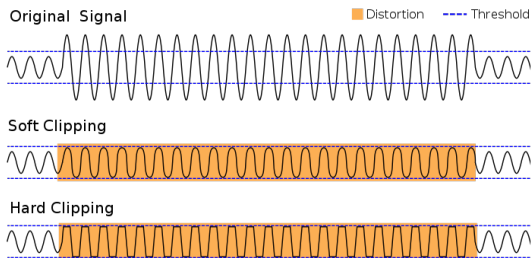


Figure 7: Limiter

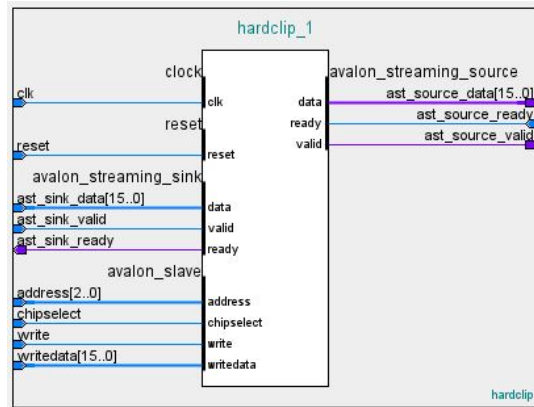


Figure 8: Hardclip Module

### 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.

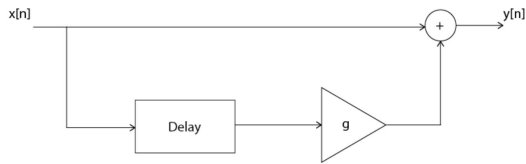


Figure 9: Delay Effect

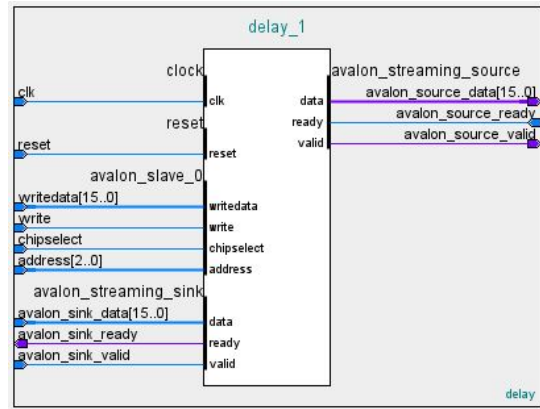


Figure 10: Delay Module

## 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.

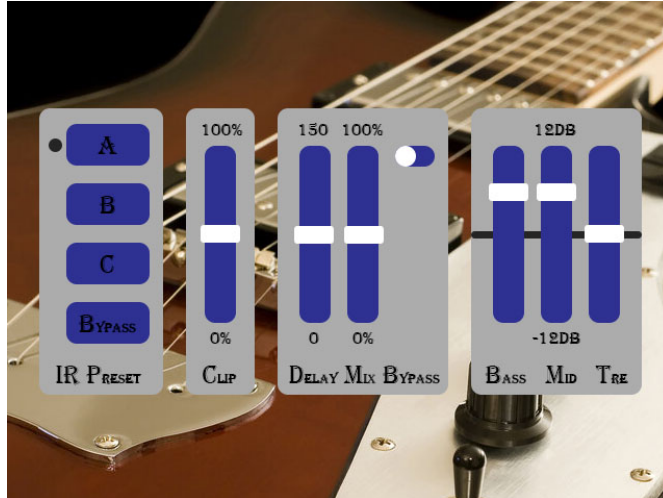


Figure 11: VGA UI

### 3 Resource Budget

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

Figure 12: BRAM Budget

DSP BLOCK Budget		
	FIR	BIQUAD
<b>BLOCKS USED</b>	2	20
<b>TOTAL</b>	<b>22/87</b>	<b>25.29%</b>

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

```
1 // =====
2 // Copyright (c) 2013 by Terasic Technologies Inc.
3 // =====
4 //
5 // Modified 2022
6 //
7 // Permission:
8 //
9 // Terasic grants permission to use and modify this code for use
10 // in synthesis for all Terasic Development Boards and Altera
11 // Development Kits made by Terasic. Other use of this code,
12 // including the selling ,duplication, or modification of any
13 // portion is strictly prohibited.
14 //
15 // Disclaimer:
16 //
17 // This VHDL/Verilog or C/C++ source code is intended as a design
18 // reference which illustrates how these types of functions can be
19 // implemented. It is the user's responsibility to verify their
20 // design for consistency and functionality through the use of
21 // formal verification methods. Terasic provides no warranty
22 // regarding the use or functionality of this code.
23 //
24 // =====
25 //
26 // Terasic Technologies Inc
27 //
28 // 9F., No.176, Sec.2, Gongdao 5th Rd, East Dist, Hsinchu City, 30070. Taiwan
29 //
30 //
31 // web: http://www.terasic.com/
32 // email: support@terasic.com
33 module soc_system_top(
34
35 /////////////// ADC ///////////////
36 inout      ADC_CS_N,
37 output     ADC_DIN,
38 input      ADC_DOUT,
39 output     ADC_SCLK,
40
41 /////////////// AUD ///////////////
42 input      AUD_ADCDAT,
43 inout      AUD_ADCLRCK,
44 inout      AUD_BCLK,
45 output     AUD_DACDAT,
46 inout      AUD_DACLCK,
47 output     AUD_XCK,
48
49 /////////////// CLOCK2 ///////////////
50 input      CLOCK2_50,
51
52 /////////////// CLOCK3 ///////////////
53 input      CLOCK3_50,
54
55 /////////////// CLOCK4 ///////////////
56 input      CLOCK4_50,
```

```

57
58 //////////////// CLOCK ////////////////
59 input          CLOCK_50 ,
60
61 //////////////// DRAM ////////////////
62 output [12:0] DRAM_ADDR ,
63 output [1:0]  DRAM_BA ,
64 output          DRAM_CAS_N ,
65 output          DRAM_CKE ,
66 output          DRAM_CLK ,
67 output          DRAM_CS_N ,
68 inout [15:0]  DRAM_DQ ,
69 output          DRAM_LDQM ,
70 output          DRAM_RAS_N ,
71 output          DRAM_UDQM ,
72 output          DRAM_WE_N ,
73
74 //////////////// FAN ////////////////
75 output          FAN_CTRL ,
76
77 //////////////// FPGA ////////////////
78 output          FPGA_I2C_SCLK ,
79 inout          FPGA_I2C_SDAT ,
80
81 //////////////// GPIO ////////////////
82 inout [35:0]  GPIO_0 ,
83 inout [35:0]  GPIO_1 ,
84
85 //////////////// HEX0 ////////////////
86 output [6:0]  HEX0 ,
87
88 //////////////// HEX1 ////////////////
89 output [6:0]  HEX1 ,
90
91 //////////////// HEX2 ////////////////
92 output [6:0]  HEX2 ,
93
94 //////////////// HEX3 ////////////////
95 output [6:0]  HEX3 ,
96
97 //////////////// HEX4 ////////////////
98 output [6:0]  HEX4 ,
99
100 //////////////// HEX5 ////////////////
101 output [6:0]  HEX5 ,
102
103 //////////////// HPS ////////////////
104 inout          HPS_CONV_USB_N ,
105 output [14:0]  HPS_DDR3_ADDR ,
106 output [2:0]  HPS_DDR3_BA ,
107 output          HPS_DDR3_CAS_N ,
108 output          HPS_DDR3_CKE ,
109 output          HPS_DDR3_CK_N ,
110 output          HPS_DDR3_CK_P ,
111 output          HPS_DDR3_CS_N ,
112 output [3:0]  HPS_DDR3_DM ,
113 inout [31:0]  HPS_DDR3_DQ ,
114 inout [3:0]  HPS_DDR3_DQS_N ,
115 inout [3:0]  HPS_DDR3_DQS_P ,
116 output          HPS_DDR3_ODT ,
117 output          HPS_DDR3_RAS_N ,
118 output          HPS_DDR3_RESET_N ,

```

```

119  input      HPS_DDR3_RZQ ,
120  output     HPS_DDR3_WE_N ,
121  output     HPS_ENET_GTX_CLK ,
122  inout      HPS_ENET_INT_N ,
123  output     HPS_ENET_MDC ,
124  inout      HPS_ENET_MDIO ,
125  input      HPS_ENET_RX_CLK ,
126  input [3:0] HPS_ENET_RX_DATA ,
127  input      HPS_ENET_RX_DV ,
128  output [3:0] HPS_ENET_TX_DATA ,
129  output     HPS_ENET_TX_EN ,
130  inout      HPS_GSENSOR_INT ,
131  inout      HPS_I2C1_SCLK ,
132  inout      HPS_I2C1_SDAT ,
133  inout      HPS_I2C2_SCLK ,
134  inout      HPS_I2C2_SDAT ,
135  inout      HPS_I2C_CONTROL ,
136  inout      HPS_KEY ,
137  inout      HPS_LED ,
138  inout      HPS_LTC_GPIO ,
139  output     HPS_SD_CLK ,
140  inout      HPS_SD_CMD ,
141  inout [3:0] HPS_SD_DATA ,
142  output     HPS_SPIM_CLK ,
143  input      HPS_SPIM_MISO ,
144  output     HPS_SPIM_MOSI ,
145  inout      HPS_SPIM_SS ,
146  input      HPS_UART_RX ,
147  output     HPS_UART_TX ,
148  input      HPS_USB_CLKOUT ,
149  inout [7:0] HPS_USB_DATA ,
150  input      HPS_USB_DIR ,
151  input      HPS_USB_NXT ,
152  output     HPS_USB_STP ,
153
154  //////////// IRDA ////////////
155  input      IRDA_RXD ,
156  output     IRDA_TXD ,
157
158  //////////// KEY ////////////
159  input [3:0] KEY ,
160
161  //////////// LEDR ////////////
162  output [9:0] LEDR ,
163
164  //////////// PS2 ////////////
165  inout      PS2_CLK ,
166  inout      PS2_CLK2 ,
167  inout      PS2_DAT ,
168  inout      PS2_DAT2 ,
169
170  //////////// SW ////////////
171  input [9:0] SW ,
172
173  //////////// TD ////////////
174  input      TD_CLK27 ,
175  input [7:0] TD_DATA ,
176  input      TD_HS ,
177  output     TD_RESET_N ,
178  input      TD_VS ,
179
180

```

```

181 /////////////// VGA ///////////////
182 output [7:0] VGA_B ,
183 output      VGA_BLANK_N ,
184 output      VGA_CLK ,
185 output [7:0] VGA_G ,
186 output      VGA_HS ,
187 output [7:0] VGA_R ,
188 output      VGA_SYNC_N ,
189 output      VGA_VS
190 );
191
192 soc_system soc_system0(
193     .clk_clk          ( CLOCK_50 ),
194     .reset_reset_n   ( 1),
195
196     .hps_dds3_mem_a   ( HPS_DDR3_ADDR ),
197     .hps_dds3_mem_ba  ( HPS_DDR3_BA ),
198     .hps_dds3_mem_ck  ( HPS_DDR3_CK_P ),
199     .hps_dds3_mem_ck_n ( HPS_DDR3_CK_N ),
200     .hps_dds3_mem_cke ( HPS_DDR3_CKE ),
201     .hps_dds3_mem_cs_n ( HPS_DDR3_CS_N ),
202     .hps_dds3_mem_ras_n ( HPS_DDR3_RAS_N ),
203     .hps_dds3_mem_cas_n ( HPS_DDR3_CAS_N ),
204     .hps_dds3_mem_we_n ( HPS_DDR3_WE_N ),
205     .hps_dds3_mem_reset_n ( HPS_DDR3_RESET_N ),
206     .hps_dds3_mem_dq   ( HPS_DDR3_DQ ),
207     .hps_dds3_mem_dqs  ( HPS_DDR3_DQS_P ),
208     .hps_dds3_mem_dqs_n ( HPS_DDR3_DQS_N ),
209     .hps_dds3_mem_odt  ( HPS_DDR3_ODT ),
210     .hps_dds3_mem_dm   ( HPS_DDR3_DM ),
211     .hps_dds3_oct_rzqin ( HPS_DDR3_RZQ ),
212
213     .hps_hps_io_emac1_inst_TX_CLK ( HPS_ENET_GTX_CLK ),
214     .hps_hps_io_emac1_inst_TXD0  ( HPS_ENET_TX_DATA [0] ),
215     .hps_hps_io_emac1_inst_TXD1  ( HPS_ENET_TX_DATA [1] ),
216     .hps_hps_io_emac1_inst_TXD2  ( HPS_ENET_TX_DATA [2] ),
217     .hps_hps_io_emac1_inst_TXD3  ( HPS_ENET_TX_DATA [3] ),
218     .hps_hps_io_emac1_inst_RXD0  ( HPS_ENET_RX_DATA [0] ),
219     .hps_hps_io_emac1_inst_MDIO  ( HPS_ENET_MDIO ),
220     .hps_hps_io_emac1_inst_MDC   ( HPS_ENET_MDC ),
221     .hps_hps_io_emac1_inst_RX_CTL ( HPS_ENET_RX_DV ),
222     .hps_hps_io_emac1_inst_TX_CTL ( HPS_ENET_TX_EN ),
223     .hps_hps_io_emac1_inst_RX_CLK ( HPS_ENET_RX_CLK ),
224     .hps_hps_io_emac1_inst_RXD1  ( HPS_ENET_RX_DATA [1] ),
225     .hps_hps_io_emac1_inst_RXD2  ( HPS_ENET_RX_DATA [2] ),
226     .hps_hps_io_emac1_inst_RXD3  ( HPS_ENET_RX_DATA [3] ),
227
228     .hps_hps_io_sdio_inst_CMD     ( HPS_SD_CMD ),
229     .hps_hps_io_sdio_inst_D0     ( HPS_SD_DATA [0] ),
230     .hps_hps_io_sdio_inst_D1     ( HPS_SD_DATA [1] ),
231     .hps_hps_io_sdio_inst_CLK    ( HPS_SD_CLK ),
232     .hps_hps_io_sdio_inst_D2     ( HPS_SD_DATA [2] ),
233     .hps_hps_io_sdio_inst_D3     ( HPS_SD_DATA [3] ),
234
235     .hps_hps_io_usb1_inst_D0     ( HPS_USB_DATA [0] ),
236     .hps_hps_io_usb1_inst_D1     ( HPS_USB_DATA [1] ),
237     .hps_hps_io_usb1_inst_D2     ( HPS_USB_DATA [2] ),
238     .hps_hps_io_usb1_inst_D3     ( HPS_USB_DATA [3] ),
239     .hps_hps_io_usb1_inst_D4     ( HPS_USB_DATA [4] ),
240     .hps_hps_io_usb1_inst_D5     ( HPS_USB_DATA [5] ),
241     .hps_hps_io_usb1_inst_D6     ( HPS_USB_DATA [6] ),
242     .hps_hps_io_usb1_inst_D7     ( HPS_USB_DATA [7] ),

```

```

243     .hps_hps_io_usb1_inst_CLK      ( HPS_USB_CLKOUT      ),
244     .hps_hps_io_usb1_inst_STP     ( HPS_USB_STP        ),
245     .hps_hps_io_usb1_inst_DIR     ( HPS_USB_DIR        ),
246     .hps_hps_io_usb1_inst_NXT     ( HPS_USB_NXT        ),
247
248     .hps_hps_io_spim1_inst_CLK     ( HPS_SPIM_CLK      ),
249     .hps_hps_io_spim1_inst_MOSI    ( HPS_SPIM_MOSI     ),
250     .hps_hps_io_spim1_inst_MISO    ( HPS_SPIM_MISO     ),
251     .hps_hps_io_spim1_inst_SSO     ( HPS_SPIM_SS       ),
252
253     .hps_hps_io_uart0_inst_RX      ( HPS_UART_RX       ),
254     .hps_hps_io_uart0_inst_TX      ( HPS_UART_TX       ),
255
256     .hps_hps_io_i2c0_inst_SDA      ( HPS_I2C1_SDAT     ),
257     .hps_hps_io_i2c0_inst_SCL      ( HPS_I2C1_SCLK     ),
258
259     .hps_hps_io_i2c1_inst_SDA      ( HPS_I2C2_SDAT     ),
260     .hps_hps_io_i2c1_inst_SCL      ( HPS_I2C2_SCLK     ),
261
262     .hps_hps_io_gpio_inst_GPI009   ( HPS_CONV_USB_N    ),
263     .hps_hps_io_gpio_inst_GPI035   ( HPS_ENET_INT_N    ),
264     .hps_hps_io_gpio_inst_GPI040   ( HPS_LTC_GPIO      ),
265
266     .hps_hps_io_gpio_inst_GPI048   ( HPS_I2C_CONTROL   ),
267     .hps_hps_io_gpio_inst_GPI053   ( HPS_LED            ),
268     .hps_hps_io_gpio_inst_GPI054   ( HPS_KEY            ),
269     .hps_hps_io_gpio_inst_GPI061   ( HPS_GSENSOR_INT   ),
270
271
272
273     .i2c_SDAT                       (FPGA_I2C_SDAT),           //      i2c.SDAT
274     .i2c_SCLK                       (FPGA_I2C_SCLK),           //      .SCLK
275     .i2s_hw_adcclk                   (AUD_ADCCLK),           //      i2s_hw.adcclk
276     .i2s_hw_adclrclk                 (AUD_ADCLRCK),        //      .adclrclk
277     .i2s_hw_bclk                     (AUD_BCLK),           //      .bclk
278     .i2s_hw_dacclk                   (AUD_DACCLK),           //      .dacclk
279     .i2s_hw_daclrclk                 (AUD_DACLCK),        //      .daclrclk
280     .i2s_mclk_clk                    (AUD_XCK),           // i2s_mclk.clk
281     .vga_r (VGA_R),
282     .vga_g (VGA_G),
283     .vga_b (VGA_B),
284     .vga_clk (VGA_CLK),
285     .vga_hs (VGA_HS),
286     .vga_vs (VGA_VS),
287     .vga_blank_n (VGA_BLANK_N),
288     .vga_sync_n (VGA_SYNC_N),
289 );
290
291 // The following quiet the "no driver" warnings for output
292 // pins and should be removed if you use any of these peripherals
293
294 assign ADC_CS_N = SW[1] ? SW[0] : 1'bZ;
295 assign ADC_DIN = SW[0];
296 assign ADC_SCLK = SW[0];
297
298 assign DRAM_ADDR = { 13{ SW[0] } };
299 assign DRAM_BA = { 2{ SW[0] } };
300 assign DRAM_DQ = SW[1] ? { 16{ SW[0] } } : 16'bZ;
301 assign {DRAM_CAS_N, DRAM_CKE, DRAM_CLK, DRAM_CS_N,
302        DRAM_LDQM, DRAM_RAS_N, DRAM_UDQM, DRAM_WE_N} = { 8{SW[0]} };
303
304 assign FAN_CTRL = SW[0];

```

```

305
306
307 assign GPIO_0 = SW[1] ? { 36{ SW[0] } } : 36'bZ;
308 assign GPIO_1 = SW[1] ? { 36{ SW[0] } } : 36'bZ;
309
310 assign HEX0 = { 7{ SW[1] } };
311 assign HEX1 = { 7{ SW[2] } };
312 assign HEX2 = { 7{ SW[3] } };
313 assign HEX3 = { 7{ SW[4] } };
314 assign HEX4 = { 7{ SW[5] } };
315 assign HEX5 = { 7{ SW[6] } };
316
317 assign IRDA_TXD = SW[0];
318
319 assign LEDR = { 10{SW[7]} };
320
321 assign PS2_CLK = SW[1] ? SW[0] : 1'bZ;
322 assign PS2_CLK2 = SW[1] ? SW[0] : 1'bZ;
323 assign PS2_DAT = SW[1] ? SW[0] : 1'bZ;
324 assign PS2_DAT2 = SW[1] ? SW[0] : 1'bZ;
325
326 assign TD_RESET_N = SW[0];
327
328
329
330 endmodule

```

### A.1.2 vga\_ball.sv

```

1  /*
2  * Avalon memory-mapped peripheral that generates VGA
3  *
4  * Stephen A. Edwards
5  * Columbia University
6  */
7
8  module vga_ball(input logic      clk,
9                 input logic      reset,
10
11                 //          input logic bypass,
12                 input logic [7:0] writedata,
13                 input logic      write,
14                 input            chipselect,
15                 input logic [2:0] address,
16
17                 output logic [7:0] VGA_R, VGA_G, VGA_B,
18                 output logic      VGA_CLK, VGA_HS, VGA_VS,
19                 output logic      VGA_BLANK_n,
20                 output logic      VGA_SYNC_n);
21
22  logic [10:0] hcount;
23  logic [9:0]  vcount;
24
25
26  logic      bypass;
27  logic [9:0] sel_x=40;//select for FIR
28  logic [9:0] sel_y;//config by sw
29
30  logic [9:0] sw_x=373;//
31  logic [9:0] sw_y=139;//
32
33  logic [9:0] slider1_x=187;//slider for clipping

```

```

34     logic [9:0] slider1_y;//config by sw
35
36     logic [9:0] slider2_x=278;//slider
37     logic [9:0] slider2_y;//config by sw
38
39     logic [9:0] slider3_x=324;//slider
40     logic [9:0] slider3_y;//config by sw
41
42
43     logic [9:0] slider4_x=464;//slider
44     logic [9:0] slider4_y;//config by sw
45
46     logic [9:0] slider5_x=510;//slider
47     logic [9:0] slider5_y;//config by sw
48
49     logic [9:0] slider6_x=556;//slider
50     logic [9:0] slider6_y;//config by sw
51
52
53
54
55     vga_counters counters(.clk50(clk), .*);
56
57
58     always_ff @(posedge clk)
59     if (reset) begin
60         sel_y <= 132;
61         slider1_y <= 232;
62         slider2_y <= 289;
63         slider3_y <= 255;
64         slider4_y <= 150;
65         slider5_y <= 255;
66         slider6_y <= 150;
67         bypass<=1;
68     end else if (chipselect && write)
69     case (address)
70         3'h0 : bypass <= writedata;
71         3'h1 : sel_y <= writedata;
72         3'h2 : slider1_y <= writedata;
73         3'h3 : slider2_y <= writedata;
74         3'h4 : slider3_y <= writedata;
75         3'h5 : slider4_y <= writedata;
76         3'h6 : slider5_y <= writedata;
77         3'h7 : slider6_y <= writedata;
78
79     endcase
80
81     logic [9:0]hcnt;
82     logic [18:0] bgaddr;
83     logic [7:0] BG,SWON,SWOFF;
84     logic [7:0] seladdr;
85     logic [7:0] S1;
86     always_comb begin
87         hcnt=hcount[10:1];
88         bgaddr=vcount*640+hcnt;
89     end
90     BG      B1 (
91         .address ( bgaddr ),
92         .clock ( clk ),
93         .q (BG )
94     );
95

```



```

96 SEL      SEL1 (
97         .address ( (hcnt-sel_x)+(vcount-sel_y)*13 ),
98         .clock ( clk ),
99         .q (S1 )
100        );
101
102
103
104 SWON     SW1 (
105         .address ( (hcnt-sw_x)+(vcount-sw_y)*38 ),
106         .clock ( clk ),
107         .q (SWON )
108        );
109
110 SWOFF    SW2 (
111         .address ( (hcnt-sw_x)+(vcount-sw_y)*38 ),
112         .clock ( clk ),
113         .q (SWOFF )
114        );
115
116
117
118 always_ff @(posedge clk) begin
119     if(reset) begin
120
121
122     end
123     else begin
124         if (VGA_BLANK_n ) begin
125             if(hcnt>=slider1_x&&hcnt<slider1_x+36&&
126                vcount>=slider1_y&&vcount<slider1_y+17) begin
127                 VGA_R<=255;
128                 VGA_G<=255;
129                 VGA_B<=255;
130
131
132             end
133             else if(hcnt>=slider2_x&&hcnt<slider2_x+36&&
134                vcount>=slider2_y&&vcount<slider2_y+17) begin
135                 VGA_R<=255;
136                 VGA_G<=255;
137                 VGA_B<=255;
138             end
139             else if(hcnt>=slider3_x&&hcnt<slider3_x+36&&
140                vcount>=slider3_y&&vcount<slider3_y+17) begin
141                 VGA_R<=255;
142                 VGA_G<=255;
143                 VGA_B<=255;
144             end
145             else if(hcnt>=slider4_x&&hcnt<slider4_x+36&&
146                vcount>=slider4_y&&vcount<slider4_y+17) begin
147                 VGA_R<=255;
148                 VGA_G<=255;
149                 VGA_B<=255;
150             end
151             else if(hcnt>=slider5_x&&hcnt<slider5_x+36&&
152                vcount>=slider5_y&&vcount<slider5_y+17) begin
153                 VGA_R<=255;
154                 VGA_G<=255;
155                 VGA_B<=255;
156             end
157         end
158     end
159 end

```

```

158
159     end
160     else if (hcnt >= slider6_x && hcnt < slider6_x + 36 &&
161             vcount >= slider6_y && vcount < slider6_y + 17) begin
162         VGA_R <= 255;
163         VGA_G <= 255;
164         VGA_B <= 255;
165
166     end
167     else if (hcnt >= sel_x && hcnt < sel_x + 13 && vcount >= sel_y && vcount < sel_y + 13) begin
168         VGA_R [7:5] <= S1 [2:0];
169         VGA_G [7:5] <= S1 [5:3];
170         VGA_B [7:6] <= S1 [7:6];
171     end
172     else if (hcnt >= sw_x && hcnt < sw_x + 38 && vcount >= sw_y && vcount < sw_y + 20) begin
173         if (bypass) begin
174             VGA_R [7:5] <= SWOFF [2:0];
175             VGA_G [7:5] <= SWOFF [5:3];
176             VGA_B [7:6] <= SWOFF [7:6];
177         end else begin
178             VGA_R [7:5] <= SWON [2:0];
179             VGA_G [7:5] <= SWON [5:3];
180             VGA_B [7:6] <= SWON [7:6];
181         end
182
183     end
184     else begin
185         VGA_R [7:5] <= BG [2:0];
186         VGA_R [4:0] <= 5'b0;
187
188         VGA_G [7:5] <= BG [5:3];
189         VGA_G [4:0] <= 5'b0;
190
191         VGA_B [7:6] <= BG [7:6];
192         VGA_B [5:0] <= 6'b0;
193     end
194 end
195 end
196 end
197
198 endmodule
199
200 module vga_counters(
201     input logic      clk50, reset,
202     output logic [10:0] hcount, // hcount[10:1] is pixel column
203     output logic [9:0] vcount, // vcount[9:0] is pixel row
204     output logic     VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n);
205
206 /*
207 * 640 X 480 VGA timing for a 50 MHz clock: one pixel every other cycle
208 *
209 * HCOUNT 1599 0                1279        1599 0
210 *
211 * -----|          Video          |-----|          Video          |
212 *
213 *
214 * |SYNC| BP |<-- HACTIVE -->|FP|SYNC| BP |<-- HACTIVE
215 *
216 * |_____|          VGA_HS          |_____|
217 */
218 // Parameters for hcount
219 parameter HACTIVE = 11'd 1280,

```

```

220         HFRONT_PORCH = 11'd 32,
221         HSYNC        = 11'd 192,
222         HBACK_PORCH  = 11'd 96,
223         HTOTAL       = HACTIVE + HFRONT_PORCH + HSYNC +
224                       HBACK_PORCH; // 1600
225
226 // Parameters for vcount
227 parameter VACTIVE    = 10'd 480,
228           VFRONT_PORCH = 10'd 10,
229           VSYNC        = 10'd 2,
230           VBACK_PORCH  = 10'd 33,
231           VTOTAL       = VACTIVE + VFRONT_PORCH + VSYNC +
232                       VBACK_PORCH; // 525
233
234 logic endOfLine;
235
236 always_ff @(posedge clk50 or posedge reset)
237     if (reset)          hcount <= 0;
238     else if (endOfLine) hcount <= 0;
239     else                hcount <= hcount + 11'd 1;
240
241 assign endOfLine = hcount == HTOTAL - 1;
242
243 logic endOfField;
244
245 always_ff @(posedge clk50 or posedge reset)
246     if (reset)          vcount <= 0;
247     else if (endOfLine)
248         if (endOfField) vcount <= 0;
249     else                vcount <= vcount + 10'd 1;
250
251 assign endOfField = vcount == VTOTAL - 1;
252
253 // Horizontal sync: from 0x520 to 0x5DF (0x57F)
254 // 101 0010 0000 to 101 1101 1111
255 assign VGA_HS = !( (hcount[10:8] == 3'b101) &
256                  !(hcount[7:5] == 3'b111));
257 assign VGA_VS = !( vcount[9:1] == (VACTIVE + VFRONT_PORCH) / 2);
258
259 assign VGA_SYNC_n = 1'b0; // For putting sync on the green signal; unused
260
261 // Horizontal active: 0 to 1279          Vertical active: 0 to 479
262 // 101 0000 0000 1280                  01 1110 0000 480
263 // 110 0011 1111 1599                  10 0000 1100 524
264 assign VGA_BLANK_n = !( (hcount[10] & (hcount[9] | hcount[8])) &
265                       !( vcount[9] | (vcount[8:5] == 4'b1111) ));
266
267 /* VGA_CLK is 25 MHz
268 *
269 * clk50      __|  |__|  |__|  |__|
270 *
271 *
272 * hcount[0]  _____|_____|
273 */
274 assign VGA_CLK = hcount[0]; // 25 MHz clock: rising edge sensitive
275
276 endmodule

```

### A.1.3 i2s\_avalon\_st.v

```

1  /*****
2  *                                     Constant Declarations                                     *

```

```

3  *****/
4
5
6  /* *****/
7  *          Internal Wires and Registers Declarations          *
8  *****/
9
10 module i2s_avalon_st #(
11     parameter DW                                = 15, //16 bit audio
12     parameter BIT_COUNTER_INIT                 = 5'd15
13 )(
14     input clk,
15     input reset,
16
17     input wire [DW:0] avalon_sink_data,
18     input wire avalon_sink_valid,
19     output wire avalon_sink_ready,
20
21     output wire [DW:0] avalon_source_data,
22     output wire avalon_source_valid,
23     input wire avalon_source_ready,
24
25     input      wire          AUD_ADCDAT,
26     input      wire          AUD_ADCLRCK,
27     input      wire          AUD_BCLK,
28     input      wire          AUD_DACLCK,
29     output     wire          AUD_DACDAT
30
31 );
32 reg [8:0] validcounter; //128max
33 always@(posedge clk) begin
34     if(reset)
35         validcounter<=0;
36     else begin
37         validcounter=validcounter+1;
38     end
39 end
40 assign avalon_sink_ready=audio_out_allowed;
41 assign avalon_source_valid=audio_in_available&&validcounter==7'd0;
42 // Internal Wires
43 wire          bclk_rising_edge;
44 wire          bclk_falling_edge;
45
46 wire          adc_lrclk_rising_edge;
47 wire          adc_lrclk_falling_edge;
48
49 wire          signed [DW: 0] new_left_channel_audio;
50 wire          signed [DW: 0] new_right_channel_audio;
51
52 wire          [ 7: 0] left_channel_read_available;
53 wire          [ 7: 0] right_channel_read_available;
54 wire          dac_lrclk_rising_edge;
55 wire          dac_lrclk_falling_edge;
56
57 wire          [ 7: 0] left_channel_write_space;
58 wire          [ 7: 0] right_channel_write_space;
59
60 // Internal Registers
61 reg          done_adc_channel_sync;
62 reg          read_interrupt_en;
63 reg          clear_read_fifos;

```

```

65 reg read_interrupt;
66
67 reg done_dac_channel_sync;
68 reg write_interrupt_en;
69 reg write_interrupt;
70 reg audio_in_available;
71 reg audio_out_allowed;
72
73 // State Machine Registers
74
75
76 /*****
77 * Finite State Machine(s) *
78 *****/
79
80
81 /*****
82 * Sequential Logic *
83 *****/
84
85 always @(posedge clk)
86 begin
87     if (reset == 1'b1)
88         done_dac_channel_sync <= 1'b0;
89     else if (dac_lrclk_falling_edge == 1'b1)
90         done_dac_channel_sync <= 1'b1;
91 end
92 always @(posedge clk)
93 begin
94     if (reset == 1'b1)
95         done_adc_channel_sync <= 1'b0;
96     else if (adc_lrclk_rising_edge == 1'b1)
97         done_adc_channel_sync <= 1'b1;
98 end
99
100 always @ (posedge clk)
101 begin
102     if (reset == 1'b1)
103         audio_in_available <= 1'b0;
104     else if ((left_channel_read_available[7] | left_channel_read_available[6])
105             & (right_channel_read_available[7] | right_channel_read_available[6]))
106         audio_in_available <= 1'b1;
107     else
108         audio_in_available <= 1'b0;
109 end
110 always @ (posedge clk)
111 begin
112     if (reset == 1'b1)
113         audio_out_allowed <= 1'b0;
114     else if ((left_channel_write_space[7] | left_channel_write_space[6])
115             & (right_channel_write_space[7] | right_channel_write_space[6]))
116         audio_out_allowed <= 1'b1;
117     else
118         audio_out_allowed <= 1'b0;
119 end
120 /*****
121 * Combinational Logic *
122 *****/
123
124
125 /*****
126 * Internal Modules *

```

```

127  *****/
128
129 altera_up_clock_edge Bit_Clock_Edges (
130     // Inputs
131     .clk                (clk),
132     .reset              (reset),
133
134     .test_clk          (AUD_BCLK),
135
136     // Bidirectionals
137
138     // Outputs
139     .rising_edge      (bclk_rising_edge),
140     .falling_edge     (bclk_falling_edge)
141 );
142
143 altera_up_clock_edge ADC_Left_Right_Clock_Edges (
144     // Inputs
145     .clk                (clk),
146     .reset              (reset),
147
148     .test_clk          (AUD_ADCLRCK),
149
150     // Bidirectionals
151
152     // Outputs
153     .rising_edge      (adc_lrclk_rising_edge),
154     .falling_edge     (adc_lrclk_falling_edge)
155 );
156
157 altera_up_clock_edge DAC_Left_Right_Clock_Edges (
158     // Inputs
159     .clk                (clk),
160     .reset              (reset),
161
162     .test_clk          (AUD_DACLRCK),
163
164     // Bidirectionals
165
166     // Outputs
167     .rising_edge      (dac_lrclk_rising_edge),
168     .falling_edge     (dac_lrclk_falling_edge)
169 );
170
171
172 altera_up_audio_in_deserializer Audio_In_Deserializer (
173     // Inputs
174     .clk (clk),
175     .reset (reset),
176     .bit_clk_rising_edge (bclk_rising_edge),
177     .bit_clk_falling_edge (bclk_falling_edge),
178     .left_right_clk_rising_edge (adc_lrclk_rising_edge),
179     .left_right_clk_falling_edge (adc_lrclk_falling_edge),
180     .done_channel_sync (done_adc_channel_sync),
181     .serial_audio_in_data (AUD_ADCDAT),
182     .read_left_audio_data_en (avalon_source_valid&&avalon_source_ready),
183     .read_right_audio_data_en (avalon_source_valid&&avalon_source_ready),
184
185     // Bidirectionals
186
187     // Outputs
188     .left_audio_fifo_read_space (left_channel_read_available),

```

```

189     .right_audio_fifo_read_space (right_channel_read_available),
190
191     //left_channel_data (new_left_channel_audio),
192     .left_channel_data (avalon_source_data), //only send lch to avalon st
193     .right_channel_data      (new_right_channel_audio)
194 );
195 defparam
196     Audio_In_Deserializer.DW = DW,
197     Audio_In_Deserializer.BIT_COUNTER_INIT = BIT_COUNTER_INIT;
198
199 altera_up_audio_out_serializer Audio_Out_Serializer (
200     // Inputs
201     .clk (clk),
202     .reset (reset),
203
204     .bit_clk_rising_edge (bclk_rising_edge),
205     .bit_clk_falling_edge (bclk_falling_edge),
206     .left_right_clk_rising_edge (done_dac_channel_sync & dac_lrclk_rising_edge),
207     .left_right_clk_falling_edge (done_dac_channel_sync & dac_lrclk_falling_edge),
208
209     .left_channel_data (avalon_sink_data),
210
211     .left_channel_data_en (avalon_sink_valid&&avalon_sink_ready),
212
213     .right_channel_data      (avalon_sink_data),
214
215     .right_channel_data_en (avalon_sink_valid&&avalon_sink_ready),
216
217     // Outputs
218     .left_channel_fifo_write_space (left_channel_write_space),
219     .right_channel_fifo_write_space (right_channel_write_space),
220
221     .serial_audio_out_data      (AUD_DACDAT)
222 );
223 //END of ALTERA IP
224 endmodule
225

```

#### A.1.4 hardclip.v

```

1  `timescale 1 ps / 1 ps
2  module hardclip (
3  //AVALON ST
4      input  wire      clk,                //clk.clk
5      input  wire      reset,              //havent done any reset logic yet
6      input  wire signed [15:0] ast_sink_data, //avalon_streaming_sink.data
7      input  wire      ast_sink_valid,     //.valid
8      output reg       ast_sink_ready,     //.ready
9      output reg signed [15:0] ast_source_data, // avalon_streaming_source.data
10     output reg       ast_source_valid,    //.valid
11     input  wire      ast_source_ready,    //.ready
12
13 //AVALON MM
14     input wire signed [15:0] writedata,
15     input wire      write,
16     input           chipselect,
17     input wire [2:0] address
18
19 );
20
21 //AVALON ST
22     reg signed [15:0] tp;

```

```

23         reg signed [15:0] tn;
24
25
26 always @(posedge clk) begin
27
28     if(ast_sink_data>tp)
29         ast_source_data<=tp;
30     else if(ast_sink_data<tn)
31         ast_source_data<=tn;
32     else
33         ast_source_data<=ast_sink_data;
34
35
36     ast_source_valid<=ast_sink_valid;
37     ast_sink_ready<=ast_source_ready;
38     end
39
40
41 //AVALON MM
42 always @(posedge clk) begin
43     if (reset) begin
44         tp <= 16'd1000;
45         tn <= -16'd1000;
46
47     end else if (chipselct && write)
48         case (address)
49             3'h0 : tp <= writedata;
50             3'h1 : tn <= writedata;
51         endcase
52     end
53
54
55 endmodule

```

### A.1.5 delay.v

```

1  `timescale 1ns/1ns
2  module delay
3      (
4
5      //AVALON ST
6      input clk,
7      input reset,
8      input avalon_source_ready,
9      output reg avalon_source_valid,
10     output reg signed [DATA_WIDTH-1:0] avalon_source_data,
11
12     output reg avalon_sink_ready,
13     input avalon_sink_valid,
14     input signed [DATA_WIDTH-1:0] avalon_sink_data,
15     //AVALON MM
16     input wire signed [15:0] writedata,
17     input wire             write,
18     input                 chipselct,
19     input wire [2:0]     address
20 );
21
22
23 //AVALON MM REG
24 reg signed [15:0] mix;//adjust gain by software >>>1=-6dB, 2=12dB, etc.
25 wire signed [15:0] s_mix;
26 assign s_mix=mix;

```



```

27 reg [15:0] MAX_DELAY; //amount of samples to delay (no larger than 8192)
28 reg [15:0] bypass; //bypass
29 wire signed [31:0] result; //amount of samples to delay (no larger than 8192)
30 assign result = fifo_reg*s_mix;
31 parameter DATA_WIDTH = 16;
32 reg signed [DATA_WIDTH-1:0] in_reg;
33 wire signed [DATA_WIDTH-1:0] fifo_reg;
34 reg signed [15:0] mixin=16'd16384;
35 wire signed [31:0] indat=avalon_sink_data*mixin;
36 always @(*)begin
37     in_reg = avalon_sink_data;
38     avalon_source_data = |bypass?avalon_sink_data:result[31:16]+indat[30:15];
39     //avalon_source_data = fifo_reg+avalon_sink_data;
40 end
41
42 //BEGIN AVALON MM
43 always @(posedge clk) begin
44
45     if (reset) begin
46         mix<=16'd30000; //adjust gain by software >>>1=-6dB, 2=12dB, etc.
47         MAX_DELAY<= 6000; //amount of samples to delay
48         bypass<=16'd0; //default to bypass
49     end else if (chipselect && write)
50         case (address)
51             3'h0 : bypass <= writedata;
52             3'h1 : MAX_DELAY <= writedata;
53             3'h2 : mix<=writedata; //adjust gain by software >>>1=-6dB, 2=12dB, etc.
54
55         endcase
56 end
57
58
59
60 //END AVALON MM
61
62 reg [15:0] ptr; //8192 samples =2^13
63 reg [15:0] ptrr;
64
65 always @(posedge clk)begin
66
67     avalon_sink_ready<=avalon_source_ready;
68     avalon_source_valid<=avalon_sink_valid;
69     if (reset) begin
70         ptr <= 0;
71         ptrr <= 1;
72     end
73     else
74         if (avalon_source_valid&&avalon_sink_ready)begin
75             if (ptr+1 == MAX_DELAY)begin
76                 ptr <= 0;
77                 ptrr <= 1;
78             end
79             else begin
80                 ptr <= ptr+1;
81                 ptrr <= ptrr+1;
82             end
83             end
84     end
85     delayram          delayram_inst (
86         .clock ( clk ),
87         .data ( in_reg ),
88         .rdaddress ( ptrr[12:0]),

```

```

89         .rden (1),
90         .waddress ( ptr[12:0] ),
91         .wren ( avalon_sink_valid),
92         .q ( fifo_reg )
93     );
94
95 endmodule

```

### A.1.6 biquad.v

```

1  module biquad_ast #(
2      parameter DW = 16
3      /* coefficients */
4
5
6      /*
7      a low pass filter for testing Q=0.7071, fc=1000Hz
8      b0_qq = 15
9      b1_qq = 31
10     b2_qq = 15
11     a1_qq = -31709
12     a2_qq = 15386
13
14
15     hpf 1000hz 0.7071
16     b0_qq =15869
17     b1_qq = -31740
18     b2_qq = 15869
19     a1_qq = -31709
20     a2_qq = 15386
21
22     */
23     )(
24     input clk,
25     input reset,
26
27     input wire signed [DW-1:0] avalon_sink_data,
28     input wire avalon_sink_valid,
29     output reg avalon_sink_ready,
30
31     output wire signed [DW-1:0] avalon_source_data,
32     output reg avalon_source_valid,
33     input wire avalon_source_ready,
34
35     input wire signed [15:0] b0_int,
36     input wire signed [15:0] b1_int,
37     input wire signed [15:0] b2_int,
38     input wire signed [15:0] a1_int,
39     input wire signed [15:0] a2_int,
40
41     input wire bypass
42 );
43
44 wire signed [DW-1:0] input_int;
45 assign input_int=avalon_sink_data;
46
47 wire signed [DW-1:0] output_int;
48
49 reg signed [DW-1:0] input_pipe1;
50 reg signed [DW-1:0] input_pipe2;
51 reg signed [DW-1:0] output_pipe1;
52 reg signed [DW-1:0] output_pipe2;

```

```

53
54 reg signed [DW + DW-1:0] input_b0;
55 reg signed [DW + DW-1:0] input_b1;
56 reg signed [DW + DW-1:0] input_b2;
57 reg signed [DW + DW-1:0] output_a1;
58 reg signed [DW + DW-1:0] output_a2;
59 wire signed [DW + DW-1:0] output_2int;
60
61
62 always @(posedge clk) begin
63     avalon_source_valid <= avalon_sink_valid;
64     avalon_sink_ready <= avalon_source_ready;
65 if (reset) begin
66     input_pipe1 <= 0;
67     input_pipe2 <= 0;
68     output_pipe1 <= 0;
69     output_pipe2 <= 0;
70 end else
71     if (avalon_sink_valid&&avalon_sink_valid) begin
72         input_pipe1 <= input_int;
73         input_pipe2 <= input_pipe1;
74         output_pipe1 <= output_int;
75         output_pipe2 <= output_pipe1;
76     end
77
78 end
79 always @(posedge clk)begin
80     if (reset) begin
81         input_b0 <= 0;
82         input_b1 <= 0;
83         input_b2 <= 0;
84         output_a1 <= 0;
85         output_a2 <= 0;
86     end else begin
87         input_b0 <= input_int * b0_int;
88         input_b1 <= input_pipe1 * b1_int;
89         input_b2 <= input_pipe2 * b2_int;
90         output_a1 <= output_pipe1 * a1_int;
91         output_a2 <= output_pipe2 * a2_int;
92     end
93 end
94 assign output_2int = input_b0 + input_b1 + input_b2 - output_a1 - output_a2;
95 assign output_int = output_2int[27:12];
96
97 assign avalon_source_data = bypass?avalon_sink_data:output_int;
98
99 endmodule
100
101
102 module biquad (
103     input clk,
104     input reset,
105
106     input wire signed [15:0] avalon_sink_data,
107     input wire avalon_sink_valid,
108     output wire avalon_sink_ready,
109
110     output wire signed [15:0] avalon_source_data,
111     output wire avalon_source_valid,
112     input wire avalon_source_ready,
113
114     //AVALON MM

```

```

115 input wire signed [15:0] writedata,
116 input wire          write,
117 input wire          chipselect,
118 input wire [4:0]    address
119
120 );
121     reg signed [15:0] i0_b0_int;
122     reg signed [15:0] i0_b1_int;
123     reg signed [15:0] i0_b2_int;
124     reg signed [15:0] i0_a1_int;
125     reg signed [15:0] i0_a2_int;
126
127     reg signed [15:0] i1_b0_int;
128     reg signed [15:0] i1_b1_int;
129     reg signed [15:0] i1_b2_int;
130     reg signed [15:0] i1_a1_int;
131     reg signed [15:0] i1_a2_int;
132
133     reg signed [15:0] i2_b0_int;
134     reg signed [15:0] i2_b1_int;
135     reg signed [15:0] i2_b2_int;
136     reg signed [15:0] i2_a1_int;
137     reg signed [15:0] i2_a2_int;
138
139     reg signed [15:0] i3_b0_int;
140     reg signed [15:0] i3_b1_int;
141     reg signed [15:0] i3_b2_int;
142     reg signed [15:0] i3_a1_int;
143     reg signed [15:0] i3_a2_int;
144     wire signed [15:0] i0srcdat;
145     wire signed [15:0] i1srcdat;
146     wire signed [15:0] i2srcdat;
147
148     reg [15:0] bypass;
149 biquad_ast i0(
150     .clk(clk),
151     .reset(reset),
152
153     .avalon_sink_data(avalon_sink_data),
154     .avalon_sink_valid(avalon_sink_valid),
155     .avalon_sink_ready(avalon_sink_ready),
156
157     .avalon_source_data(i0srcdat),
158     .avalon_source_valid(i0srcv),
159     .avalon_source_ready(i0srcr),
160
161     .b0_int(i0_b0_int),
162     .b1_int(i0_b1_int),
163     .b2_int(i0_b2_int),
164     .a1_int(i0_a1_int),
165     .a2_int(i0_a2_int),
166     .bypass(bypass[0])
167 );
168
169 biquad_ast i1(
170     .clk(clk),
171     .reset(reset),
172
173     .avalon_sink_data(i0srcdat),
174     .avalon_sink_valid(i0srcv),
175     .avalon_sink_ready(i0srcr),
176

```

```

177     .avalon_source_data(i1srcdat),
178     .avalon_source_valid(i1srcv),
179     .avalon_source_ready(i1srcr),
180
181     .b0_int(i1_b0_int),
182     .b1_int(i1_b1_int),
183     .b2_int(i1_b2_int),
184     .a1_int(i1_a1_int),
185     .a2_int(i1_a2_int),
186     .bypass(bypass[0])
187 );
188
189 biquad_ast i2(
190     .clk(clk),
191     .reset(reset),
192
193     .avalon_sink_data(i1srcdat),
194     .avalon_sink_valid(i1srcv),
195     .avalon_sink_ready(i1srcr),
196
197     .avalon_source_data(i2srcdat),
198     .avalon_source_valid(i2srcv),
199     .avalon_source_ready(i2srcr),
200
201     .b0_int(i2_b0_int),
202     .b1_int(i2_b1_int),
203     .b2_int(i2_b2_int),
204     .a1_int(i2_a1_int),
205     .a2_int(i2_a2_int),
206     .bypass(bypass[0])
207 );
208
209 biquad_ast i3(
210     .clk(clk),
211     .reset(reset),
212
213     .avalon_sink_data(i2srcdat),
214     .avalon_sink_valid(i2srcv),
215     .avalon_sink_ready(i2srcr),
216
217     .avalon_source_data(avalon_source_data),
218     .avalon_source_valid(avalon_source_valid),
219     .avalon_source_ready(avalon_source_ready),
220
221     .b0_int(i3_b0_int),
222     .b1_int(i3_b1_int),
223     .b2_int(i3_b2_int),
224     .a1_int(i3_a1_int),
225     .a2_int(i3_a2_int),
226     .bypass(bypass[0])
227 );
228
229 //BEGIN AVALON MM
230 always @(posedge clk) begin
231     if (reset) begin //default to bypass
232         bypass<=1;
233
234         i0_b0_int<=16'd4096;
235         i0_b1_int<= 16'd0;
236         i0_b2_int<=16'd0;
237         i0_a1_int<= 16'd0;
238         i0_a2_int<=16'd0;

```

```

239
240             i1_b0_int<=16'd4096;
241             i1_b1_int<=16'd0;
242             i1_b2_int<=16'd0;
243             i1_a1_int<=16'd0;
244             i1_a2_int<=16'd0;
245
246             i2_b0_int<=16'd4096;
247             i2_b1_int<=16'd0;
248             i2_b2_int<=16'd0;
249             i2_a1_int<=16'd0;
250             i2_a2_int<=16'd0;
251
252             i3_b0_int<=16'd4096;
253             i3_b1_int<=16'd0;
254             i3_b2_int<=16'd0;
255             i3_a1_int<=16'd0;
256             i3_a2_int<=16'd0;
257 end else if (chipselect && write)
258     case (address)
259         5'h0 : i0_b0_int <= writedata;
260         5'h1 : i0_b1_int <= writedata;
261         5'h2 : i0_b2_int<=writedata;
262         5'h3 : i0_a1_int <= writedata;
263         5'h4 : i0_a2_int<=writedata;
264
265         5'h5 : i1_b0_int <= writedata;
266         5'h6 : i1_b1_int <= writedata;
267         5'h7 : i1_b2_int<=writedata;
268         5'h8 : i1_a1_int <= writedata;
269         5'h9 : i1_a2_int<=writedata;
270
271         5'd10 : i2_b0_int <= writedata;
272         5'd11 : i2_b1_int <= writedata;
273         5'd12 : i2_b2_int<=writedata;
274         5'd13 : i2_a1_int <= writedata;
275         5'd14 : i2_a2_int<=writedata;
276
277
278         5'd15 : i3_b0_int <= writedata;
279         5'd16 : i3_b1_int <= writedata;
280         5'd17 : i3_b2_int<=writedata;
281         5'd18 : i3_a1_int <= writedata;
282         5'd19 : i3_a2_int<=writedata;
283
284         5'd20 : bypass<=writedata;
285     endcase
286 end
287 //END AVALON MM
288 endmodule

```

## A.2 Software

### A.2.1 Makefile

```

1 ifneq (${KERNELRELEASE},)
2
3 # KERNELRELEASE defined: we are being compiled as part of the Kernel
4     obj-m :=delay.o hardclip.o biquad.o fir.o vga_ball.o
5
6 else
7

```

```

8 # We are being compiled as a module: use the Kernel build system
9
10     KERNEL_SOURCE := /usr/src/linux-headers-$(shell uname -r)
11     PWD := $(shell pwd)
12
13
14
15 default: module hello
16 hello: hello.o usbkeyboard.o
17     cc $(CFLAGS) -o hello hello.o usbkeyboard.o -lusb-1.0 -pthread -lm
18
19 hello.o: hello.c usbkeyboard.h
20 usbkeyboard.o: usbkeyboard.c usbkeyboard.h
21
22 module:
23     ${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} modules
24
25 clean:
26     ${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} clean
27     ${RM} hello
28
29 usbkeyboard.o : usbkeyboard.c usbkeyboard.h
30
31
32 TARFILES = Makefile README delay.h delay.c hello.c hardclip.h hardclip.c
33     biquad.h biquad.c fir.h fir.c vga_ball.h vga_ball.c usbkeyboard.h usbkeyboard.c
34 TARFILE = lab3-sw.tar.gz
35 .PHONY : tar
36 tar : $(TARFILE)
37
38 $(TARFILE) : $(TARFILES)
39     tar zcfC $(TARFILE) .. $(TARFILES:%=lab3-sw/%)
40
41 endif

```

## A.2.2 biquad.c

```

1 /* * Device driver for the variable biquad module
2  *
3  * A Platform device implemented using the misc subsystem
4  *
5  * Stephen A. Edwards
6  * Columbia University
7  *
8  * References:
9  * Linux source: Documentation/driver-model/platform.txt
10  *     drivers/misc/arm-charlcd.c
11  * http://www.linuxforu.com/tag/linux-device-drivers/
12  * http://free-electrons.com/docs/
13  *
14  * "make" to build
15  * insmod biquad.ko
16  *
17  * Check code style with
18  * checkpatch.pl --file --no-tree biquad.c
19  */
20
21 #include <linux/module.h>
22 #include <linux/init.h>
23 #include <linux/errno.h>
24 #include <linux/version.h>
25 #include <linux/kernel.h>

```

```

26 #include <linux/platform_device.h>
27 #include <linux/miscdevice.h>
28 #include <linux/slab.h>
29 #include <linux/io.h>
30 #include <linux/of.h>
31 #include <linux/of_address.h>
32 #include <linux/fs.h>
33 #include <linux/uaccess.h>
34 #include "biquad.h"
35
36 #define DRIVER_NAME "biquad"
37
38 /* Device registers */
39
40
41 int z=0;
42 /*
43  * Information about our device
44  */
45 struct biquad_dev {
46     struct resource res; /* Resource: our registers */
47     void __iomem *virtbase; /* Where registers can be accessed in memory */
48     biquad_param_t x;
49 } dev;
50
51 /*
52  * Write segments of a single digit
53  * Assumes digit is in range and the device information has been set up
54  */
55 static void write_biquad_param(biquad_param_t *x)
56 {
57     for (z=0;z<21;z++){
58         iowrite16(x->biquadcoeff[z], (dev.virtbase+2*z) );
59     }
60     dev.x = *x;
61
62 }
63
64 /*
65  * Handle ioctl() calls from userspace:
66  * Read or write the segments on single digits.
67  * Note extensive error checking of arguments
68  */
69 static long biquad_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
70 {
71     biquad_arg_t vla;
72
73     switch (cmd) {
74
75
76     case BIQUAD_WRITE_PARAM:
77         if (copy_from_user(&vla, (biquad_arg_t *) arg,
78                             sizeof(biquad_arg_t)))
79             return -EACCES;
80         write_biquad_param(&vla.biquadparam);
81         break;
82
83
84     case BIQUAD_READ_PARAM:
85         vla.biquadparam = dev.x;
86         if (copy_to_user((biquad_arg_t *) arg, &vla,
87                             sizeof(biquad_arg_t)))

```



```

88             return -EACCES;
89         break;
90
91     default:
92         return -EINVAL;
93     }
94
95     return 0;
96 }
97
98 /* The operations our device knows how to do */
99 static const struct file_operations biquad_fops = {
100     .owner          = THIS_MODULE,
101     .unlocked_ioctl = biquad_ioctl,
102 };
103
104 /* Information about our device for the "misc" framework -- like a char dev */
105 static struct miscdevice biquad_misc_device = {
106     .minor          = MISC_DYNAMIC_MINOR,
107     .name           = DRIVER_NAME,
108     .fops           = &biquad_fops,
109 };
110
111 /*
112  * Initialization code: get resources (registers) and display
113  * a welcome message
114  */
115 static int __init biquad_probe(struct platform_device *pdev)
116 {
117     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 }};
118     int ret;
119
120     /* Register ourselves as a misc device: creates /dev/biquad */
121     ret = misc_register(&biquad_misc_device);
122
123     /* Get the address of our registers from the device tree */
124     ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
125     if (ret) {
126         ret = -ENOENT;
127         goto out_deregister;
128     }
129
130     /* Make sure we can use these registers */
131     if (request_mem_region(dev.res.start, resource_size(&dev.res),
132         DRIVER_NAME) == NULL) {
133         ret = -EBUSY;
134         goto out_deregister;
135     }
136
137     /* Arrange access to our registers */
138     dev.virtbase = of_iomap(pdev->dev.of_node, 0);
139     if (dev.virtbase == NULL) {
140         ret = -ENOMEM;
141         goto out_release_mem_region;
142     }
143
144     /* Set an initial color */
145     write_biquad_param(&beige);
146
147     return 0;
148
149 out_release_mem_region:

```

```

150         release_mem_region(dev.res.start, resource_size(&dev.res));
151 out_deregister:
152         misc_deregister(&biquad_misc_device);
153         return ret;
154     }
155
156     /* Clean-up code: release resources */
157     static int biquad_remove(struct platform_device *pdev)
158     {
159         iounmap(dev.virtbase);
160         release_mem_region(dev.res.start, resource_size(&dev.res));
161         misc_deregister(&biquad_misc_device);
162         return 0;
163     }
164
165     /* Which "compatible" string(s) to search for in the Device Tree */
166     #ifdef CONFIG_OF
167     static const struct of_device_id biquad_of_match[] = {
168         { .compatible = "csee4840,biquad-1.0" },
169         {}},
170     };
171     MODULE_DEVICE_TABLE(of, biquad_of_match);
172     #endif
173
174     /* Information for registering ourselves as a "platform" driver */
175     static struct platform_driver biquad_driver = {
176         .driver = {
177             .name      = DRIVER_NAME,
178             .owner     = THIS_MODULE,
179             .of_match_table = of_match_ptr(biquad_of_match),
180         },
181         .remove = __exit_p(biquad_remove),
182     };
183
184     /* Called when the module is loaded: set things up */
185     static int __init biquad_init(void)
186     {
187         pr_info(DRIVER_NAME ":_init\n");
188         return platform_driver_probe(&biquad_driver, biquad_probe);
189     }
190
191     /* Calball when the module is unloaded: release resources */
192     static void __exit biquad_exit(void)
193     {
194         platform_driver_unregister(&biquad_driver);
195         pr_info(DRIVER_NAME ":_exit\n");
196     }
197
198     module_init(biquad_init);
199     module_exit(biquad_exit);
200
201     MODULE_LICENSE("GPL");
202     MODULE_AUTHOR("LILONGYINIUBI, Columbia University");
203     MODULE_DESCRIPTION("biquad_driver");

```

### A.2.3 biquad.h

```

1  #ifndef _BIQUAD_H
2  #define _BIQUAD_H
3
4  #include <linux/ioctl.h>
5

```

```

6  typedef struct {
7      short signed int biquadcoeff[21];
8  } biquad_param_t;
9
10 typedef struct {
11     biquad_param_t biquadparam;
12 } biquad_arg_t;
13
14 #define BIQUAD_MAGIC 'q'
15
16 /* ioctls and their arguments */
17 #define BIQUAD_WRITE_PARAM _IOW(BIQUAD_MAGIC, 1, biquad_arg_t *)
18 #define BIQUAD_READ_PARAM  _IOR(BIQUAD_MAGIC, 2, biquad_arg_t *)
19
20 #endif

```

#### A.2.4 eq\_coeff.c

```

1  /*
2   * Academic License - for use in teaching, academic research, and meeting
3   * course requirements at degree granting institutions only. Not for
4   * government, commercial, or other organizational use.
5   * File: eq_coeff.c
6   *
7   * MATLAB Coder version      : 5.2
8   * C/C++ source code generated on : 29-Apr-2022 14:21:01
9   */
10
11 /* Include Files */
12 #include "eq_coeff.h"
13 #include "rt_nonfinite.h"
14 #include "rt_nonfinite.h"
15 #include <math.h>
16 #include "biquad.h"
17 #include <stdio.h>
18
19 /* Function Declarations */
20 static double rt_powd_snf(double u0, double u1);
21
22 static double rt_roundd_snf(double u);
23
24 /* Function Definitions */
25 /*
26  * Arguments      : double u0
27  *                  double u1
28  * Return Type    : double
29  */
30 static double rt_powd_snf(double u0, double u1)
31 {
32     double d;
33     double d1;
34     double y;
35     if (rtIsNaN(u0) || rtIsNaN(u1)) {
36         y = rtNaN;
37     } else {
38         d = fabs(u0);
39         d1 = fabs(u1);
40         if (rtIsInf(u1)) {
41             if (d == 1.0) {
42                 y = 1.0;
43             } else if (d > 1.0) {
44                 if (u1 > 0.0) {

```

```

45         y = rtInf;
46     } else {
47         y = 0.0;
48     }
49     } else if (u1 > 0.0) {
50         y = 0.0;
51     } else {
52         y = rtInf;
53     }
54     } else if (d1 == 0.0) {
55         y = 1.0;
56     } else if (d1 == 1.0) {
57         if (u1 > 0.0) {
58             y = u0;
59         } else {
60             y = 1.0 / u0;
61         }
62     } else if (u1 == 2.0) {
63         y = u0 * u0;
64     } else if ((u1 == 0.5) && (u0 >= 0.0)) {
65         y = sqrt(u0);
66     } else if ((u0 < 0.0) && (u1 > floor(u1))) {
67         y = rtNaN;
68     } else {
69         y = pow(u0, u1);
70     }
71 }
72 return y;
73 }
74
75 /*
76  * Arguments      : double u
77  * Return Type   : double
78  */
79 static double rt_roundd_snf(double u)
80 {
81     double y;
82     if (fabs(u) < 4.503599627370496E+15) {
83         if (u >= 0.5) {
84             y = floor(u + 0.5);
85         } else if (u > -0.5) {
86             y = u * 0.0;
87         } else {
88             y = ceil(u - 0.5);
89         }
90     } else {
91         y = u;
92     }
93     return y;
94 }
95
96 /*
97  * Arguments      : double bass
98  *                 double mid
99  *                 double treble
100 *                 short b[9]
101 *                 short a[9]
102 * Return Type   : void
103 */
104 void eq_coeff(double bass, double mid, double treble, biquad_param_t bi[])
105 {
106     double G;

```

```

107 double V0;
108 double a1;
109 double a2;
110 double b0;
111 double b1;
112 double b2;
113 double b_a1;
114 double b_a2;
115 double b_b0;
116 double b_b1;
117 double b_b2;
118 double c_a1;
119 double c_a2;
120 double c_b0;
121 double c_b1;
122 double c_b2;
123 short i;
124 short a[9];
125 short b[9];
126
127 /* bass, mid, treble are integer inputs with 146-285 range */
128 G = (bass - 215.5) * 2.0 * 12.0 / -139.0;
129 b0 = (mid - 215.5) * 2.0 * 12.0 / -139.0;
130 /* */
131 /* Derive coefficients for a shelving filter with a given amplitude and */
132 /* cutoff frequency. All coefficients are calculated as described in */
133 /* Zolzer's DAFX book (p. 50 -55). */
134 /* */
135 /* Usage: [B,A] = shelving(G, Fc, Fs, Q, type); */
136 /* */
137 /* G is the logarithmic gain (in dB) */
138 /* FC is the center frequency */
139 /* Fs is the sampling rate */
140 /* Q adjusts the slope by replacing the sqrt(2) term */
141 /* type is a character string defining filter type */
142 /* Choices are: 'Bass_Shelf' or 'Treble_Shelf' */
143 /* */
144 /* Author: Jeff Tackett 08/22/05 */
145 /* */
146 /* Error Check */
147 V0 = rt_powd_snf(10.0, G / 20.0);
148 /* sqrt(2) */
149 /* Invert gain if a cut */
150 if (V0 < 1.0) {
151     V0 = 1.0 / V0;
152 }
153 /* %%%%%%%%%%% */
154 /* BASS BOOST */
155 /* %%%%%%%%%%% */
156 if (G > 0.0) {
157     G = sqrt(V0) * 1.4142135623730949 * 0.009817792466785694;
158     b_b0 = ((G + 1.0) + V0 * 9.6389048920873918E-5) / 1.0139808443080136;
159     b1 = 2.0 * (V0 * 9.6389048920873918E-5 - 1.0) / 1.0139808443080136;
160     b2 = ((1.0 - G) + V0 * 9.6389048920873918E-5) / 1.0139808443080136;
161     a1 = -1.9722337291952663;
162     a2 = 0.972613969313063;
163     /* %%%%%%%%%%% */
164     /* BASS CUT */
165     /* %%%%%%%%%%% */
166 } else if (G < 0.0) {
167     G = (1.4142135623730949 * sqrt(V0) * 0.009817792466785694 + 1.0) +
168         V0 * 9.6389048920873918E-5;

```

```

169     b_b0 = 1.0139808443080136 / G;
170     b1 = -1.9998072219021583 / G;
171     b2 = 0.98621193378982808 / G;
172     a1 = 2.0 * (V0 * 9.6389048920873918E-5 - 1.0) / G;
173     a2 = ((1.0 - 1.4142135623730949 * sqrt(V0) * 0.009817792466785694) +
174           V0 * 9.6389048920873918E-5) /
175           G;
176     /* %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% */
177     /*     TREBLE BOOST */
178     /* %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% */
179 } else {
180     b_b0 = V0;
181     b1 = 0.0;
182     b2 = 0.0;
183     a1 = 0.0;
184     a2 = 0.0;
185 }
186 /* return values */
187 G = (treble - 215.5) * 2.0 * 12.0 / -139.0;
188 /* */
189 /* Derive coefficients for a shelving filter with a given amplitude and */
190 /* cutoff frequency. All coefficients are calculated as described in */
191 /* Zolzer's DAFX book (p. 50 -55). */
192 /* */
193 /* Usage:      [B,A] = shelving(G, Fc, Fs, Q, type); */
194 /* */
195 /*           G is the logarithmic gain (in dB) */
196 /*           FC is the center frequency */
197 /*           Fs is the sampling rate */
198 /*           Q adjusts the slope by replacing the sqrt(2) term */
199 /*           type is a character string defining filter type */
200 /*           Choices are: 'Bass_Shelf' or 'Treble_Shelf' */
201 /* */
202 /* Author:     Jeff Tackett 08/22/05 */
203 /* */
204 /* Error Check */
205 V0 = rt_powd_snf(10.0, G / 20.0);
206 /* sqrt(2) */
207 /* Invert gain if a cut */
208 if (V0 < 1.0) {
209     V0 = 1.0 / V0;
210 }
211 /* %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% */
212 /*     BASS BOOST */
213 /* %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% */
214 if (G > 0.0) {
215     G = 1.4142135623730949 * sqrt(V0) * 0.15838444032453627;
216     c_b0 = ((V0 + G) + 0.025085630936916591) / 1.2490750545127478;
217     b_b1 = 2.0 * (0.025085630936916591 - V0) / 1.2490750545127478;
218     b_b2 = ((V0 - G) + 0.025085630936916591) / 1.2490750545127478;
219     b_a1 = -1.5610180758007182;
220     b_a2 = 0.64135153805756318;
221     /* %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% */
222     /*     TREBLE CUT */
223     /* %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% */
224 } else if (G < 0.0) {
225     G = (V0 + 1.4142135623730949 * sqrt(V0) * 0.15838444032453627) +
226         0.025085630936916591;
227     c_b0 = 1.2490750545127478 / G;
228     b_b1 = -1.9498287381261667 / G;
229     b_b2 = 0.80109620736108533 / G;
230     G = 1.4142135623730949 / sqrt(V0) * 0.15838444032453627;

```

```

231     c_a2 = (G + 1.0) + 0.025085630936916591 / V0;
232     b_a1 = 2.0 * (0.025085630936916591 / V0 - 1.0) / c_a2;
233     b_a2 = ((1.0 - G) + 0.025085630936916591 / V0) / c_a2;
234     /* %%%%%%%%%%%%%%% */
235     /*     All-Pass */
236     /* %%%%%%%%%%%%%%% */
237 } else {
238     c_b0 = V0;
239     b_b1 = 0.0;
240     b_b2 = 0.0;
241     b_a1 = 0.0;
242     b_a2 = 0.0;
243 }
244 /* return values */
245 /* */
246 /* Derive coefficients for a shelving filter with a given amplitude and */
247 /* cutoff frequency. All coefficients are calculated as described in */
248 /* Zolzer's DAFX book (p. 50 -55). */
249 /* */
250 /* Usage:      [B,A] = shelving(G, Fc, Fs, Q); */
251 /* */
252 /*           G is the logarithmic gain (in dB) */
253 /*           FC is the center frequency */
254 /*           Fs is the sampling rate */
255 /*           Q adjusts the slope by replacing the sqrt(2) term */
256 /* */
257 /* Author:     Yuqi Zhu 03/30/22 */
258 /* */
259 V0 = rt_powd_snf(10.0, b0 / 20.0);
260 /* %%%%%%%%%%%%%%% */
261 /*     BOOST */
262 /* %%%%%%%%%%%%%%% */
263 if (b0 > 0.0) {
264     G = V0 * 0.03929010700766964 / 0.70710678118654757;
265     b0 = ((G + 1.0) + 0.0015437125086741311) / 1.0571083147060107;
266     c_b1 = -1.8890330793945247;
267     c_b2 = ((1.0 - G) + 0.0015437125086741311) / 1.0571083147060107;
268     c_a1 = -1.8890330793945247;
269     c_a2 = 0.89487434461663562;
270     /* %%%%%%%%%%%%%%% */
271     /*     CUT */
272     /* %%%%%%%%%%%%%%% */
273 } else if (b0 < 0.0) {
274     G = (0.055564602197336582 / V0 + 1.0) + 0.0015437125086741311;
275     b0 = 1.0571083147060107 / G;
276     c_b1 = -1.9969125749826517 / G;
277     c_b2 = 0.94597911031133752 / G;
278     c_a1 = c_b1;
279     c_a2 = ((1.0 - 0.03929010700766964 / V0 / 0.70710678118654757) +
280            0.0015437125086741311) /
281            G;
282     /* %%%%%%%%%%%%%%% */
283     /*     All-Pass */
284     /* %%%%%%%%%%%%%%% */
285 } else {
286     b0 = V0;
287     c_b1 = 0.0;
288     c_b2 = 0.0;
289     c_a1 = 0.0;
290     c_a2 = 0.0;
291 }
292 /* return values */

```

```

293 a[0] = 4096;
294 G = rt_roundd_snf(a1 * 4096.0);
295 if (G < 32768.0) {
296     i = (short)G;
297 } else if (G >= 32768.0) {
298     i = MAX_int16_T;
299 } else {
300     i = 0;
301 }
302 a[1] = i;
303 G = rt_roundd_snf(a2 * 4096.0);
304 if (G < 32768.0) {
305     if (G >= -32768.0) {
306         i = (short)G;
307     } else {
308         i = MIN_int16_T;
309     }
310 } else if (G >= 32768.0) {
311     i = MAX_int16_T;
312 } else {
313     i = 0;
314 }
315 a[2] = i;
316 a[3] = 4096;
317 G = rt_roundd_snf(c_a1 * 4096.0);
318 if (G < 32768.0) {
319     i = (short)G;
320 } else {
321     i = 0;
322 }
323 a[4] = i;
324 G = rt_roundd_snf(c_a2 * 4096.0);
325 if (G < 32768.0) {
326     if (G >= -32768.0) {
327         i = (short)G;
328     } else {
329         i = MIN_int16_T;
330     }
331 } else {
332     i = 0;
333 }
334 a[5] = i;
335 a[6] = 4096;
336 G = rt_roundd_snf(b_a1 * 4096.0);
337 if (G < 32768.0) {
338     i = (short)G;
339 } else {
340     i = 0;
341 }
342 a[7] = i;
343 G = rt_roundd_snf(b_a2 * 4096.0);
344 if (G < 32768.0) {
345     i = (short)G;
346 } else {
347     i = 0;
348 }
349 a[8] = i;
350 G = rt_roundd_snf(b_b0 * 4096.0);
351 if (G < 32768.0) {
352     i = (short)G;
353 } else if (G >= 32768.0) {
354     i = MAX_int16_T;

```



```

355 } else {
356     i = 0;
357 }
358 b[0] = i;
359 G = rt_roundd_snf(b1 * 4096.0);
360 if (G < 32768.0) {
361     i = (short)G;
362 } else if (G >= 32768.0) {
363     i = MAX_int16_T;
364 } else {
365     i = 0;
366 }
367 b[1] = i;
368 G = rt_roundd_snf(b2 * 4096.0);
369 if (G < 32768.0) {
370     if (G >= -32768.0) {
371         i = (short)G;
372     } else {
373         i = MIN_int16_T;
374     }
375 } else if (G >= 32768.0) {
376     i = MAX_int16_T;
377 } else {
378     i = 0;
379 }
380 b[2] = i;
381 G = rt_roundd_snf(b0 * 4096.0);
382 if (G < 32768.0) {
383     i = (short)G;
384 } else if (G >= 32768.0) {
385     i = MAX_int16_T;
386 } else {
387     i = 0;
388 }
389 b[3] = i;
390 G = rt_roundd_snf(c_b1 * 4096.0);
391 if (G < 32768.0) {
392     i = (short)G;
393 } else {
394     i = 0;
395 }
396 b[4] = i;
397 G = rt_roundd_snf(c_b2 * 4096.0);
398 if (G < 32768.0) {
399     if (G >= -32768.0) {
400         i = (short)G;
401     } else {
402         i = MIN_int16_T;
403     }
404 } else {
405     i = 0;
406 }
407 b[5] = i;
408 G = rt_roundd_snf(c_b0 * 4096.0);
409 if (G < 32768.0) {
410     i = (short)G;
411 } else if (G >= 32768.0) {
412     i = MAX_int16_T;
413 } else {
414     i = 0;
415 }
416 b[6] = i;

```

```

417 G = rt_roundd_snf(b_b1 * 4096.0);
418 if (G < 32768.0) {
419     if (G >= -32768.0) {
420         i = (short)G;
421     } else {
422         i = MIN_int16_T;
423     }
424 } else {
425     i = 0;
426 }
427 b[7] = i;
428 G = rt_roundd_snf(b_b2 * 4096.0);
429 if (G < 32768.0) {
430     if (G >= -32768.0) {
431         i = (short)G;
432     } else {
433         i = MIN_int16_T;
434     }
435 } else if (G >= 32768.0) {
436     i = MAX_int16_T;
437 } else {
438     i = 0;
439 }
440 b[8] = i;
441 (bi[0].biquadcoeff)[0] = b[0];
442 (bi[0].biquadcoeff)[1] = b[1];
443 (bi[0].biquadcoeff)[2] = b[2];
444 (bi[0].biquadcoeff)[3] = a[1];
445 (bi[0].biquadcoeff)[4] = a[2];
446 (bi[0].biquadcoeff)[5] = b[3];
447 (bi[0].biquadcoeff)[6] = b[4];
448 (bi[0].biquadcoeff)[7] = b[5];
449 (bi[0].biquadcoeff)[8] = a[4];
450 (bi[0].biquadcoeff)[9] = a[5];
451 (bi[0].biquadcoeff)[10] = b[6];
452 (bi[0].biquadcoeff)[11] = b[7];
453 (bi[0].biquadcoeff)[12] = b[8];
454 (bi[0].biquadcoeff)[13] = a[7];
455 (bi[0].biquadcoeff)[14] = a[8];
456 printf("Biquad:␣");
457 for (int i = 0; i <15; i++) printf("%d␣", (bi[0].biquadcoeff)[i]);
458 printf("\n");
459 }
460
461 /*
462  * File trailer for eq_coeff.c
463  *
464  * [EOF]
465  */

```

### A.2.5 eq\_coeff.h

```

1  /*
2  * Academic License - for use in teaching, academic research, and meeting
3  * course requirements at degree granting institutions only. Not for
4  * government, commercial, or other organizational use.
5  * File: eq_coeff.h
6  *
7  * MATLAB Coder version      : 5.2
8  * C/C++ source code generated on : 29-Apr-2022 14:21:01
9  */
10

```

```

11 #ifndef EQ_COEFF_H
12 #define EQ_COEFF_H
13
14 /* Include Files */
15 #include "rtwtypes.h"
16 #include <stddef.h>
17 #include <stdlib.h>
18 #include "biquad.h"
19
20 #ifdef __cplusplus
21 extern "C" {
22 #endif
23
24 /* Function Declarations */
25 extern void eq_coeff(double bass, double mid, double treble, biquad_param_t bi[]);
26
27 #ifdef __cplusplus
28 }
29 #endif
30
31 #endif
32 /*
33  * File trailer for eq_coeff.h
34  *
35  * [EOF]
36  */

```

## A.2.6 delay.c

```

1 // Device driver for the delay module
2
3
4 #include <linux/module.h>
5 #include <linux/init.h>
6 #include <linux/errno.h>
7 #include <linux/version.h>
8 #include <linux/kernel.h>
9 #include <linux/platform_device.h>
10 #include <linux/miscdevice.h>
11 #include <linux/slab.h>
12 #include <linux/io.h>
13 #include <linux/of.h>
14 #include <linux/of_address.h>
15 #include <linux/fs.h>
16 #include <linux/uaccess.h>
17 #include "delay.h"
18
19 #define DRIVER_NAME "delay"
20
21 /* Device registers */
22 #define DELAY_BYPASS(x) (x) //bypass, active high
23 #define DELAY_LENGTH(x) ((x)+2) //number of samples to delay
24 #define DELAY_MIX(x) ((x)+4) //mixing, 1=-6dB, 2=-12dB, etc.
25
26
27 /*
28  * Information about our device
29  */
30 struct delay_dev {
31     struct resource res; /* Resource: our registers */
32     void __iomem *virtbase; /* Where registers can be accessed in memory */
33     delay_param_t x;

```

```

34 } dev;
35
36 /*
37  * Write segments of a single digit
38  * Assumes digit is in range and the device information has been set up
39  */
40 static void write_delay_param(delay_param_t *x)
41 {
42     iowrite16(x->bypass, DELAY_BYPASS(dev.virtbase) );
43     iowrite16(x->length, DELAY_LENGTH(dev.virtbase) );
44     iowrite16(x->mix, DELAY_MIX(dev.virtbase) );
45     dev.x = *x;
46 }
47
48 /*
49  * Handle ioctl() calls from userspace:
50  * Read or write the segments on single digits.
51  * Note extensive error checking of arguments
52  */
53 static long delay_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
54 {
55     delay_arg_t vla;
56
57     switch (cmd) {
58
59     case DELAY_WRITE_PARAM:
60         if (copy_from_user(&vla, (delay_arg_t *) arg,
61                             sizeof(delay_arg_t)))
62             return -EACCES;
63         write_delay_param(&vla.delayparam);
64         break;
65
66     case DELAY_READ_PARAM:
67         vla.delayparam = dev.x;
68         if (copy_to_user((delay_arg_t *) arg, &vla,
69                             sizeof(delay_arg_t)))
70             return -EACCES;
71         break;
72     default:
73         return -EINVAL;
74     }
75     return 0;
76 }
77
78
79
80
81
82 /* The operations our device knows how to do */
83 static const struct file_operations delay_fops = {
84     .owner          = THIS_MODULE,
85     .unlocked_ioctl = delay_ioctl,
86 };
87
88 /* Information about our device for the "misc" framework -- like a char dev */
89 static struct miscdevice delay_misc_device = {
90     .minor          = MISC_DYNAMIC_MINOR,
91     .name           = DRIVER_NAME,
92     .fops           = &delay_fops,
93 };
94
95 /*

```

```

96  * Initialization code: get resources (registers) and display
97  * a welcome message
98  */
99  static int __init delay_probe(struct platform_device *pdev)
100 {
101     delay_param_t beige = { 1, 4096, 1 };
102     int ret;
103
104     /* Register ourselves as a misc device: creates /dev/delay */
105     ret = misc_register(&delay_misc_device);
106
107     /* Get the address of our registers from the device tree */
108     ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
109     if (ret) {
110         ret = -ENOENT;
111         goto out_deregister;
112     }
113
114     /* Make sure we can use these registers */
115     if (request_mem_region(dev.res.start, resource_size(&dev.res),
116         DRIVER_NAME) == NULL) {
117         ret = -EBUSY;
118         goto out_deregister;
119     }
120
121     /* Arrange access to our registers */
122     dev.virtbase = of_iomap(pdev->dev.of_node, 0);
123     if (dev.virtbase == NULL) {
124         ret = -ENOMEM;
125         goto out_release_mem_region;
126     }
127
128     /* Set an initial color */
129     write_delay_param(&beige);
130
131     return 0;
132
133 out_release_mem_region:
134     release_mem_region(dev.res.start, resource_size(&dev.res));
135 out_deregister:
136     misc_deregister(&delay_misc_device);
137     return ret;
138 }
139
140 /* Clean-up code: release resources */
141 static int delay_remove(struct platform_device *pdev)
142 {
143     iounmap(dev.virtbase);
144     release_mem_region(dev.res.start, resource_size(&dev.res));
145     misc_deregister(&delay_misc_device);
146     return 0;
147 }
148
149 /* Which "compatible" string(s) to search for in the Device Tree */
150 #ifdef CONFIG_OF
151 static const struct of_device_id delay_of_match[] = {
152     { .compatible = "csee4840,delay-1.0" },
153     {}
154 };
155 MODULE_DEVICE_TABLE(of, delay_of_match);
156 #endif
157

```

```

158 /* Information for registering ourselves as a "platform" driver */
159 static struct platform_driver delay_driver = {
160     .driver = {
161         .name     = DRIVER_NAME,
162         .owner    = THIS_MODULE,
163         .of_match_table = of_match_ptr(delay_of_match),
164     },
165     .remove = __exit_p(delay_remove),
166 };
167
168 /* Called when the module is loaded: set things up */
169 static int __init delay_init(void)
170 {
171     pr_info(DRIVER_NAME ":_init\n");
172     return platform_driver_probe(&delay_driver, delay_probe);
173 }
174
175 /* Calball when the module is unloaded: release resources */
176 static void __exit delay_exit(void)
177 {
178     platform_driver_unregister(&delay_driver);
179     pr_info(DRIVER_NAME ":_exit\n");
180 }
181
182 module_init(delay_init);
183 module_exit(delay_exit);
184
185 MODULE_LICENSE("GPL");
186 MODULE_AUTHOR("LILONGYINIUBI, Columbia University");
187 MODULE_DESCRIPTION("delay_driver");

```

### A.2.7 delay.h

```

1  #ifndef _DELAY_H
2  #define _DELAY_H
3
4  #include <linux/ioctl.h>
5
6  typedef struct {
7      unsigned short bypass, length, mix;
8  } delay_param_t;
9
10 typedef struct {
11     delay_param_t delayparam;
12 } delay_arg_t;
13
14 #define DELAY_MAGIC 'q'
15
16 /* ioctls and their arguments */
17 #define DELAY_WRITE_PARAM _IOW(DELAY_MAGIC, 1, delay_arg_t *)
18 #define DELAY_READ_PARAM  _IOR(DELAY_MAGIC, 2, delay_arg_t *)
19
20 #endif

```

### A.2.8 fir.c

```

1  // Device driver for the variable fir module
2
3
4  #include <linux/module.h>
5  #include <linux/init.h>
6  #include <linux/errno.h>

```

```

7  #include <linux/version.h>
8  #include <linux/kernel.h>
9  #include <linux/platform_device.h>
10 #include <linux/miscdevice.h>
11 #include <linux/slab.h>
12 #include <linux/io.h>
13 #include <linux/of.h>
14 #include <linux/of_address.h>
15 #include <linux/fs.h>
16 #include <linux/uaccess.h>
17 #include "fir.h"
18
19 #define DRIVER_NAME "fir"
20
21 /* Device registers */
22
23
24 int z=0;
25 /*
26  * Information about our device
27  */
28 struct fir_dev {
29     struct resource res; /* Resource: our registers */
30     void __iomem *virtbase; /* Where registers can be accessed in memory */
31     fir_param_t x;
32 } dev;
33
34 /*
35  * Write segments of a single digit
36  * Assumes digit is in range and the device information has been set up
37  */
38 static void write_fir_param(fir_param_t *x)
39 {
40     for (z=0;z<500;z++){
41         iowrite16(x->fircoeff[z], (dev.virtbase+2*z) );
42     }
43     dev.x = *x;
44 }
45
46 /*
47  * Handle ioctl() calls from userspace:
48  * Read or write the segments on single digits.
49  * Note extensive error checking of arguments
50  */
51
52 static long fir_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
53 {
54     fir_arg_t vla;
55
56     switch (cmd) {
57
58
59     case FIR_WRITE_PARAM:
60         if (copy_from_user(&vla, (fir_arg_t *) arg, sizeof(fir_arg_t)))
61             return -EACCES;
62         write_fir_param(&vla.firparam);
63         break;
64
65
66     case FIR_READ_PARAM:
67         vla.firparam = dev.x;
68         if (copy_to_user((fir_arg_t *) arg, &vla, sizeof(fir_arg_t)))

```

```

69             return -EACCES;
70         break;
71
72     default:
73         return -EINVAL;
74     }
75
76     return 0;
77 }
78
79 /* The operations our device knows how to do */
80 static const struct file_operations fir_fops = {
81     .owner          = THIS_MODULE,
82     .unlocked_ioctl = fir_ioctl,
83 };
84
85 /* Information about our device for the "misc" framework -- like a char dev */
86 static struct miscdevice fir_misc_device = {
87     .minor          = MISC_DYNAMIC_MINOR,
88     .name           = DRIVER_NAME,
89     .fops           = &fir_fops,
90 };
91
92 /*
93  * Initialization code: get resources (registers) and display
94  * a welcome message
95  */
96 static int __init fir_probe(struct platform_device *pdev)
97 {
98     fir_param_t beige =    {{ 15369, 32767, 26902, ..., -356, -359 }}
99
100     int ret;
101
102     /* Register ourselves as a misc device: creates /dev/fir */
103     ret = misc_register(&fir_misc_device);
104
105     /* Get the address of our registers from the device tree */
106     ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
107     if (ret) {
108         ret = -ENOENT;
109         goto out_deregister;
110     }
111
112     /* Make sure we can use these registers */
113     if (request_mem_region(dev.res.start, resource_size(&dev.res),
114         DRIVER_NAME) == NULL) {
115         ret = -EBUSY;
116         goto out_deregister;
117     }
118
119     /* Arrange access to our registers */
120     dev.virtbase = of_iomap(pdev->dev.of_node, 0);
121     if (dev.virtbase == NULL) {
122         ret = -ENOMEM;
123         goto out_release_mem_region;
124     }
125
126     /* Set an initial color */
127     write_fir_param(&beige);
128
129     return 0;
130

```



```

131 out_release_mem_region:
132     release_mem_region(dev.res.start, resource_size(&dev.res));
133 out_deregister:
134     misc_deregister(&fir_misc_device);
135     return ret;
136 }
137
138 /* Clean-up code: release resources */
139 static int fir_remove(struct platform_device *pdev)
140 {
141     iounmap(dev.virtbase);
142     release_mem_region(dev.res.start, resource_size(&dev.res));
143     misc_deregister(&fir_misc_device);
144     return 0;
145 }
146
147 /* Which "compatible" string(s) to search for in the Device Tree */
148 #ifdef CONFIG_OF
149 static const struct of_device_id fir_of_match[] = {
150     { .compatible = "fir,fir-21.1" },
151     {}
152 };
153 MODULE_DEVICE_TABLE(of, fir_of_match);
154 #endif
155
156 /* Information for registering ourselves as a "platform" driver */
157 static struct platform_driver fir_driver = {
158     .driver = {
159         .name     = DRIVER_NAME,
160         .owner    = THIS_MODULE,
161         .of_match_table = of_match_ptr(fir_of_match),
162     },
163     .remove = __exit_p(fir_remove),
164 };
165
166 /* Called when the module is loaded: set things up */
167 static int __init fir_init(void)
168 {
169     pr_info(DRIVER_NAME ":_init\n");
170     return platform_driver_probe(&fir_driver, fir_probe);
171 }
172
173 /* Calball when the module is unloaded: release resources */
174 static void __exit fir_exit(void)
175 {
176     platform_driver_unregister(&fir_driver);
177     pr_info(DRIVER_NAME ":_exit\n");
178 }
179
180 module_init(fir_init);
181 module_exit(fir_exit);
182
183 MODULE_LICENSE("GPL");
184 MODULE_AUTHOR("LILONGYINIUBI, Columbia University");
185 MODULE_DESCRIPTION("fir_driver");

```

## A.2.9 fir.h

```

1 #ifndef _FIR_H
2 #define _FIR_H
3
4 #include <linux/ioctl.h>

```

```

5
6 typedef struct {
7     short signed int fircoeff[500];
8 } fir_param_t;
9
10 typedef struct {
11     fir_param_t firparam;
12 } fir_arg_t;
13
14 #define FIR_MAGIC 'q'
15
16 /* ioctls and their arguments */
17 #define FIR_WRITE_PARAM _IOW(FIR_MAGIC, 1, fir_arg_t *)
18 #define FIR_READ_PARAM _IOR(FIR_MAGIC, 2, fir_arg_t *)
19
20 #endif

```

### A.2.10 hardclip.c

```

1 // Device driver for the variable hardclip module
2
3
4 #include <linux/module.h>
5 #include <linux/init.h>
6 #include <linux/errno.h>
7 #include <linux/version.h>
8 #include <linux/kernel.h>
9 #include <linux/platform_device.h>
10 #include <linux/miscdevice.h>
11 #include <linux/slab.h>
12 #include <linux/io.h>
13 #include <linux/of.h>
14 #include <linux/of_address.h>
15 #include <linux/fs.h>
16 #include <linux/uaccess.h>
17 #include "hardclip.h"
18
19 #define DRIVER_NAME "hardclip"
20
21 /* Device registers */
22 #define HARDCLIP_POSCLIP(x) ((x))
23 #define HARDCLIP_NEGCLIP(x) ((x)+2)
24
25
26 /*
27  * Information about our device
28  */
29 struct hardclip_dev {
30     struct resource res; /* Resource: our registers */
31     void __iomem *virtbase; /* Where registers can be accessed in memory */
32     hardclip_param_t x;
33 } dev;
34
35 /*
36  * Write segments of a single digit
37  * Assumes digit is in range and the device information has been set up
38  */
39 static void write_hardclip_param(hardclip_param_t *x)
40 {
41     iowrite16(x->posclip, HARDCLIP_POSCLIP(dev.virtbase) );
42     iowrite16(x->negclip, HARDCLIP_NEGCLIP(dev.virtbase) );
43     dev.x = *x;

```

```

44 }
45
46 /*
47  * Handle ioctl() calls from userspace:
48  * Read or write the segments on single digits.
49  * Note extensive error checking of arguments
50  */
51 static long hardclip_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
52 {
53     hardclip_arg_t vla;
54
55     switch (cmd) {
56
57
58     case HARDCLIP_WRITE_PARAM:
59         if (copy_from_user(&vla, (hardclip_arg_t *) arg,
60                             sizeof(hardclip_arg_t)))
61             return -EACCES;
62         write_hardclip_param(&vla.hardclipparam);
63         break;
64
65
66     case HARDCLIP_READ_PARAM:
67         vla.hardclipparam = dev.x;
68         if (copy_to_user((hardclip_arg_t *) arg, &vla,
69                             sizeof(hardclip_arg_t)))
70             return -EACCES;
71         break;
72
73     default:
74         return -EINVAL;
75     }
76
77     return 0;
78 }
79
80 /* The operations our device knows how to do */
81 static const struct file_operations hardclip_fops = {
82     .owner          = THIS_MODULE,
83     .unlocked_ioctl = hardclip_ioctl,
84 };
85
86 /* Information about our device for the "misc" framework -- like a char dev */
87 static struct miscdevice hardclip_misc_device = {
88     .minor          = MISC_DYNAMIC_MINOR,
89     .name           = DRIVER_NAME,
90     .fops           = &hardclip_fops,
91 };
92
93 /*
94  * Initialization code: get resources (registers) and display
95  * a welcome message
96  */
97 static int __init hardclip_probe(struct platform_device *pdev)
98 {
99     hardclip_param_t beige = { 32767, -32767 };
100     int ret;
101
102     /* Register ourselves as a misc device: creates /dev/hardclip */
103     ret = misc_register(&hardclip_misc_device);
104
105     /* Get the address of our registers from the device tree */

```

```

106     ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
107     if (ret) {
108         ret = -ENOENT;
109         goto out_deregister;
110     }
111
112     /* Make sure we can use these registers */
113     if (request_mem_region(dev.res.start, resource_size(&dev.res),
114                           DRIVER_NAME) == NULL) {
115         ret = -EBUSY;
116         goto out_deregister;
117     }
118
119     /* Arrange access to our registers */
120     dev.virtbase = of_iomap(pdev->dev.of_node, 0);
121     if (dev.virtbase == NULL) {
122         ret = -ENOMEM;
123         goto out_release_mem_region;
124     }
125
126     /* Set an initial color */
127     write_hardclip_param(&beige);
128
129     return 0;
130
131 out_release_mem_region:
132     release_mem_region(dev.res.start, resource_size(&dev.res));
133 out_deregister:
134     misc_deregister(&hardclip_misc_device);
135     return ret;
136 }
137
138 /* Clean-up code: release resources */
139 static int hardclip_remove(struct platform_device *pdev)
140 {
141     iounmap(dev.virtbase);
142     release_mem_region(dev.res.start, resource_size(&dev.res));
143     misc_deregister(&hardclip_misc_device);
144     return 0;
145 }
146
147 /* Which "compatible" string(s) to search for in the Device Tree */
148 #ifdef CONFIG_OF
149 static const struct of_device_id hardclip_of_match[] = {
150     { .compatible = "csee4840,hardclip-1.0" },
151     {}
152 };
153 MODULE_DEVICE_TABLE(of, hardclip_of_match);
154 #endif
155
156 /* Information for registering ourselves as a "platform" driver */
157 static struct platform_driver hardclip_driver = {
158     .driver = {
159         .name     = DRIVER_NAME,
160         .owner    = THIS_MODULE,
161         .of_match_table = of_match_ptr(hardclip_of_match),
162     },
163     .remove = __exit_p(hardclip_remove),
164 };
165
166 /* Called when the module is loaded: set things up */
167 static int __init hardclip_init(void)

```

```

168 {
169     pr_info(DRIVER_NAME ":_init\n");
170     return platform_driver_probe(&hardclip_driver, hardclip_probe);
171 }
172
173 /* Calball when the module is unloaded: release resources */
174 static void __exit hardclip_exit(void)
175 {
176     platform_driver_unregister(&hardclip_driver);
177     pr_info(DRIVER_NAME ":_exit\n");
178 }
179
180 module_init(hardclip_init);
181 module_exit(hardclip_exit);
182
183 MODULE_LICENSE("GPL");
184 MODULE_AUTHOR("LILONGYINIUBI, Columbia University");
185 MODULE_DESCRIPTION("hardclip_driver");

```

### A.2.11 hardclip.h

```

1 #ifndef _HARDCLIP_H
2 #define _HARDCLIP_H
3
4 #include <linux/ioctl.h>
5
6 typedef struct {
7     short signed int posclip, negclip;
8 } hardclip_param_t;
9
10 typedef struct {
11     hardclip_param_t hardclipparam;
12 } hardclip_arg_t;
13
14 #define HARDCLIP_MAGIC 'q'
15
16 /* ioctls and their arguments */
17 #define HARDCLIP_WRITE_PARAM _IOW(HARDCLIP_MAGIC, 1, hardclip_arg_t *)
18 #define HARDCLIP_READ_PARAM _IOR(HARDCLIP_MAGIC, 2, hardclip_arg_t *)
19
20 #endif

```

### A.2.12 hello.c

```

1 /*
2  * Userspace program that communicates with the delay device driver
3  * through ioctls
4  *
5  * Stephen A. Edwards
6  * Columbia University
7  */
8
9 #include <stdio.h>
10 #include "delay.h"
11 #include "hardclip.h"
12 #include <sys/ioctl.h>
13 #include <sys/types.h>
14 #include <sys/stat.h>
15 #include <fcntl.h>
16 #include <string.h>
17 #include <unistd.h>
18 #include "biquad.h"

```

```

19 #include "fir.h"
20 #include "vga_ball.h"
21 #include "eq_coeff.h"
22
23 #include <stdlib.h>
24 #include "usbkeyboard.h"
25 #include <pthread.h>
26 #include "keycode.h"
27 #define KEY_ENTER 0x28
28 #define KEY_SPACE 0x2c
29 #define KEY_BACKSPACE 0x2a
30 #define KEY_RIGHT 0x4f
31 #define KEY_LEFT 0x50
32 #define KEY_1 0x1e
33 #define KEY_2 0x1f
34 #define KEY_3 0x20
35 #define KEY_4 0x21
36 #define KEY_Q 0x14
37 #define KEY_A 0x04
38 #define KEY_W 0x1a
39 #define KEY_S 0x16
40 #define KEY_E 0x08
41 #define KEY_D 0x07
42 #define KEY_R 0x15
43 #define KEY_F 0x09
44 #define KEY_T 0x17
45 #define KEY_G 0x0a
46 #define KEY_Y 0x1c
47 #define KEY_H 0x0b
48 #define KEY_X 0x1b
49 #define KEY_Z 0x1d
50 #define KEY_B 0x05
51 #define KEY_N 0x11
52
53 int delay_fd;
54 int hardclip_fd;
55 int biquad_fd;
56 int fir_fd;
57 int vga_ball_fd;
58
59
60 struct libusb_device_handle *keyboard;
61 uint8_t endpoint_address;
62
63
64 uint8_t prevkey_1;
65 uint8_t prevkey_2;
66
67 //preset locations
68 short unsigned int preset1=132;
69 short unsigned int preset2=188;
70 short unsigned int preset3=245;
71 short unsigned int preset4=304;
72 char clippreset=5;
73
74 //slider locations
75 short unsigned int slider1=215;
76 short unsigned int slider2=215;
77 short unsigned int slider3=215;
78 short unsigned int slider4=215;
79 short unsigned int slider5=215;
80 short unsigned int slider6=215;

```

```

81
82
83
84 //LUTS
85 static const char filename_delay[] = "/dev/delay";
86 delay_param_t delayparam = { 0x0000, 4000, 8192 }; //bypass, samples, mix
87 static const char filename_hardclip[] = "/dev/hardclip";
88 static const hardclip_param_t hardclipparam[] = { //lookup table for hardclip threshold
89     {32767,-32767 }, {2750,-2750 }, {2500,-2500 }, {2250,-2250 }, {2000,-2000 },
90     {1750,-1750 }, {1500,-1500 }, {1250,-1250 }, {1000,-1000 }, {750,-750 },
91     {500,-500 }, };
92 static const char filename_biquad[] = "/dev/biquad";
93 biquad_param_t biquadparam[] = {
94     {{ 1024, 0, 0, 0, 0,1024,0,0,0,0, 1024,0,0,0,0, 1024,0,0,0,0,1 }},
95     {{ 32767,0,0,0,0,32767,0,0,0,0,32767,0,0,0,0,32767,0,0,0,0,0 }}
96 };
97 static const char filename_fir[] = "/dev/fir";
98 static const fir_param_t firparam[] = {
99     {{ 15369, 32767, 26902, ..., 81, 13}},
100    {{ 0, 0, 0, ..., 81, 13}},
101    {{ 15369, 32767, 26902, ..., 81, 13}},
102    {{ 0, 32767, 0, ..., 0, 0}},
103 };
104 static const char filename[] = "/dev/vga_ball";
105 vga_ball_color_t colors= { 0,215, 215,215,215,215,215,215};
106
107 void setup(void){
108     if ( (delay_fd = open(filename_delay, O_RDWR)) == -1) {
109         fprintf(stderr, "could not open %s\n", filename_delay);
110         return -1;
111     }
112     if ( (hardclip_fd = open(filename_hardclip, O_RDWR)) == -1) {
113         fprintf(stderr, "could not open %s\n", filename_hardclip);
114         return -1;
115     }
116
117     if ( (biquad_fd = open(filename_biquad, O_RDWR)) == -1) {
118         fprintf(stderr, "could not open %s\n", filename_biquad);
119         return -1;
120     }
121     if ( (fir_fd = open(filename_fir, O_RDWR)) == -1) {
122         fprintf(stderr, "could not open %s\n", filename_fir);
123         return -1;
124     }
125     if ( (vga_ball_fd = open(filename, O_RDWR)) == -1) {
126         fprintf(stderr, "could not open %s\n", filename);
127         return -1;
128     }
129 }
130
131 void set_background_color(const vga_ball_color_t *c)
132 {
133     vga_ball_arg_t vla;
134     vla.background = *c;
135     if (ioctl(vga_ball_fd, VGA BALL_WRITE_BACKGROUND, &vla)) {
136         perror("ioctl(VGA BALL_SET_BACKGROUND) failed");
137         return;
138     }
139     printf("Sliders: %d %d %d %d %d %d %d %d\n", c->bypass, c->sely,
140         c->sly1, c->sly2, c->sly3, c->sly4, c->sly5, c->sly6);

```

```

141 }
142
143 void set_hardclip_param(const hardclip_param_t *c)
144 {
145     hardclip_arg_t hardclipvla;
146     hardclipvla.hardclipparam = *c;
147     if (ioctl(hardclip_fd, HARDCLIP_WRITE_PARAM, &hardclipvla)) {
148         perror("ioctl(HARDCLIP_WRITE_PARAM) failed");
149         return;
150     }
151 }
152
153 void set_delay_param(const delay_param_t *c)
154 {
155     delay_arg_t delayvla;
156     delayvla.delayparam = *c;
157     if (ioctl(delay_fd, DELAY_WRITE_PARAM, &delayvla)) {
158         perror("ioctl(DELAY_WRITE_PARAM) failed");
159         return;
160     }
161 }
162 void set_biquad_param(const biquad_param_t *c)
163 {
164     biquad_arg_t biquadvla;
165     biquadvla.biquadparam = *c;
166     if (ioctl(biquad_fd, BIQUAD_WRITE_PARAM, &biquadvla)) {
167         perror("ioctl(BIQUAD_WRITE_PARAM) failed");
168         return;
169     }
170 }
171 void set_fir_param(const fir_param_t *c)
172 {
173     fir_arg_t firvla;
174     firvla.firparam = *c;
175     if (ioctl(fir_fd, FIR_WRITE_PARAM, &firvla)) {
176         perror("ioctl(FIR_WRITE_PARAM) failed");
177         return;
178     }
179 }
180
181
182 int main()
183 {
184     struct usb_keyboard_packet packet;
185     int transferred;
186     setup();
187
188     /* Open the keyboard */
189     if ( (keyboard = openkeyboard(&endpoint_address)) == NULL ) {
190         fprintf(stderr, "Did not find a keyboard\n");
191         exit(1);
192     }
193     set_biquad_param(biquadparam);
194
195     /* Look for and handle keypresses */
196     for (;;) {
197         libusb_interrupt_transfer(keyboard, endpoint_address,
198                                 (unsigned char *) &packet, sizeof(packet),
199                                 &transferred, 0);
200         if (transferred == sizeof(packet)) {
201
202             if (packet.keycode[0] == 0 )

```



```

203     {
204     }
205     else if (packet.keycode[0] == prevkey_1 && packet.keycode[1] == 0)
206     {
207     }
208     else if ( packet.keycode[0] == prevkey_2 )
209     {
210
211     }
212     else if (packet.keycode[0] == 0x29) { // if press esc
213
214     }
215     else if (packet.keycode[0] == KEY_1) { //if enter pressed
216         colors.sely=preset1;
217         set_background_color(&colors);
218         set_fir_param(&firparam[0]);
219
220     }
221     else if (packet.keycode[0] == KEY_2) { //if enter pressed
222         colors.sely=preset2;
223         set_background_color(&colors);
224         set_fir_param(&firparam[1]);
225
226     }
227     else if (packet.keycode[0] == KEY_3) { //if enter pressed
228         colors.sely=preset3;
229         set_background_color(&colors);
230         set_fir_param(&firparam[2]);
231
232     }
233     else if (packet.keycode[0] == KEY_4) { //if enter pressed
234         colors.sely=preset4;
235         set_background_color(&colors);
236         set_fir_param(&firparam[3]);
237
238     }
239     else if (packet.keycode[0] == KEY_BACKSPACE) { //if backspace
240
241     }
242
243     else if (packet.keycode[0] == KEY_Q) { //HARDCLIP++
244         if(slidebar1>146){
245             slider1-=14;
246             clippreset -=1;}
247         colors.sly1=slider1;
248         set_background_color(&colors);
249         set_hardclip_param(&hardclipparam[clippreset]);
250
251     }
252     else if (packet.keycode[0] == KEY_A) { //HARDCLIP--
253         if(slidebar1<285){
254             slider1+=14;
255
256                                     clippreset+=1;}
257
258         colors.sly1=slider1;
259         set_background_color(&colors);
260         set_hardclip_param(&hardclipparam[clippreset]);
261
262     }
263     else if (packet.keycode[0] == KEY_W) { //DELAY++
264         if(slidebar2>146){

```

```

265         slider2-=14;
266         delayparam.length+=700;
267     }
268     colors.sly2=slider2;
269     set_background_color(&colors);
270     set_delay_param(&delayparam);
271 }
272 else if (packet.keycode[0] == KEY_S) { //DELAY--
273     if(slider2<285){
274         slider2+=14;
275         delayparam.length-=700;
276     }
277     colors.sly2=slider2;
278     set_background_color(&colors);
279     set_delay_param(&delayparam);
280 }
281 else if (packet.keycode[0] == KEY_B) { //BYPASS DELAY
282     delayparam.bypass=1;
283     colors.bypass=0;
284     set_background_color(&colors);
285     set_delay_param(&delayparam);
286 }
287 else if (packet.keycode[0] == KEY_N) { //ENABLE DELAY
288     delayparam.bypass=0;
289     colors.bypass=1;
290     set_background_color(&colors);
291     set_delay_param(&delayparam);
292 }
293
294 else if (packet.keycode[0] == KEY_E) { //MIX++
295     if(slider3>146){
296         slider3-=14;
297         delayparam.mix+=1500;
298     }
299     colors.sly3=slider3;
300     set_background_color(&colors);
301     set_delay_param(&delayparam);
302 }
303 else if (packet.keycode[0] == KEY_D) { //MIX--
304     if(slider3<285){
305         slider3+=14;
306         delayparam.mix-=1500;
307     }
308     colors.sly3=slider3;
309     set_delay_param(&delayparam);
310     set_background_color(&colors);
311 }
312
313 else if (packet.keycode[0] == KEY_R) { //BASS++
314     if(slider4>146){
315         slider4-=14;
316         eq_coeff((double)slider4, (double)slider5,
317 (double)slider6, biquadparam);
318     }
319     colors.sly4=slider4;
320     set_background_color(&colors);
321     set_biquad_param(biquadparam);
322 }
323 else if (packet.keycode[0] == KEY_F) { //BASS--
324     if(slider4<285){
325         eq_coeff((double)slider4, (double)slider5,
326 (double)slider6, biquadparam);

```

```

327         }
328         colors.sly4=slider4;
329         set_background_color(&colors);
330         set_biquad_param(biquadparam);
331     }
332
333     else if (packet.keycode[0] == KEY_T) { //MID++
334         if(slider5>146){
335             slider5-=14;
336             eq_coeff((double)slider4, (double)slider5,
337 (double)slider6, biquadparam);
338         }
339         colors.sly5=slider5;
340         set_background_color(&colors);
341         set_biquad_param(biquadparam);
342     }
343     else if (packet.keycode[0] == KEY_G) { //MID--
344         if(slider5<285){
345             slider5+=14;
346             eq_coeff((double)slider4, (double)slider5,
347 (double)slider6, biquadparam);
348         }
349         colors.sly5=slider5;
350         set_background_color(&colors);
351         set_biquad_param(biquadparam);
352     }
353
354     else if (packet.keycode[0] == KEY_Y) { //TREBLE++
355         if(slider6>146){
356             slider6-=14;
357             eq_coeff((double)slider4, (double)slider5,
358 (double)slider6, biquadparam);
359         }
360         colors.sly6=slider6;
361         set_background_color(&colors);
362         set_biquad_param(biquadparam);
363     }
364     else if (packet.keycode[0] == KEY_H) { //TREBLE--
365         if(slider6<285){
366             slider6+=14;
367             eq_coeff((double)slider4, (double)slider5,
368 (double)slider6, biquadparam);
369         }
370         colors.sly6=slider6;
371         set_background_color(&colors);
372         set_biquad_param(biquadparam);
373     }
374
375     else if (packet.keycode[0] == KEY_Z) { //BYPASS EQ
376         biquadparam[0].biquadcoeff[20]=1;
377         set_background_color(&colors);
378         set_biquad_param(biquadparam);
379     }
380     else if (packet.keycode[0] == KEY_X) { //ENABLE EQ
381         biquadparam[0].biquadcoeff[20]=0;
382         set_background_color(&colors);
383         set_biquad_param(biquadparam);
384     }
385
386
387     else if (packet.keycode[0] == KEY_U) { //BYPASS EQ
388         if(biquadparam[0].biquadcoeff[15] < 16000)

```

```

389         biquadparam[0].biquadcoeff[15] =
390         (short)(((float)biquadparam[0].biquadcoeff[15]) * 1.2);
391         set_background_color(&colors);
392         set_biquad_param(biquadparam);
393     }
394     else if (packet.keycode[0] == KEYJ) { //ENABLE EQ
395         if(biquadparam[0].biquadcoeff[15] > 100)
396             biquadparam[0].biquadcoeff[15] =
397             (short)(((float)biquadparam[0].biquadcoeff[15]) / 1.2);
398         set_background_color(&colors);
399         set_biquad_param(biquadparam);
400     }
401
402
403
404
405     prevkey_1 = packet.keycode[0];
406     prevkey_2 = packet.keycode[1];
407 }
408 }
409
410
411     return 0;
412 }

```

### A.2.13 usbkeyboard.c

```

1  #include "usbkeyboard.h"
2
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  /* References on libusb 1.0 and the USB HID/keyboard protocol
7   *
8   * http://libusb.org
9   * http://www.dreamincode.net/forums/topic/148707-introduction-to-using-libusb-10/
10  * http://www.usb.org/developers/devclass\_docs/HID1\_11.pdf
11  * http://www.usb.org/developers/devclass\_docs/Hut1\_11.pdf
12  */
13
14  /*
15  * Find and return a USB keyboard device or NULL if not found
16  * The argument con
17  *
18  */
19  struct libusb_device_handle *openkeyboard(uint8_t *endpoint_address) {
20     libusb_device **devs;
21     struct libusb_device_handle *keyboard = NULL;
22     struct libusb_device_descriptor desc;
23     ssize_t num_devs, d;
24     uint8_t i, k;
25
26     /* Start the library */
27     if ( libusb_init(NULL) < 0 ) {
28         fprintf(stderr, "Error: libusb_init failed\n");
29         exit(1);
30     }
31
32     /* Enumerate all the attached USB devices */
33     if ( (num_devs = libusb_get_device_list(NULL, &devs)) < 0 ) {
34         fprintf(stderr, "Error: libusb_get_device_list failed\n");
35         exit(1);

```

```

36 }
37
38 /* Look at each device, remembering the first HID device that speaks
39 the keyboard protocol */
40
41 for (d = 0 ; d < num_devs ; d++) {
42     libusb_device *dev = devs[d];
43     if ( libusb_get_device_descriptor(dev, &desc) < 0 ) {
44         fprintf(stderr, "Error: libusb_get_device_descriptor failed\n");
45         exit(1);
46     }
47
48     if (desc.bDeviceClass == LIBUSB_CLASS_PER_INTERFACE) {
49         struct libusb_config_descriptor *config;
50         libusb_get_config_descriptor(dev, 0, &config);
51         for (i = 0 ; i < config->bNumInterfaces ; i++)
52             for (k = 0 ; k < config->interface[i].num_altsetting ; k++) {
53                 const struct libusb_interface_descriptor *inter =
54                     config->interface[i].altsetting + k ;
55                 if ( inter->bInterfaceClass == LIBUSB_CLASS_HID &&
56                     inter->bInterfaceProtocol == USB_HID_KEYBOARD_PROTOCOL) {
57                     int r;
58                     if ((r = libusb_open(dev, &keyboard)) != 0) {
59                         fprintf(stderr, "Error: libusb_open failed: %d\n", r);
60                         exit(1);
61                     }
62                     if (libusb_kernel_driver_active(keyboard, i))
63                         libusb_detach_kernel_driver(keyboard, i);
64                     libusb_set_auto_detach_kernel_driver(keyboard, i);
65                     if ((r = libusb_claim_interface(keyboard, i)) != 0) {
66                         fprintf(stderr, "Error: libusb_claim_interface failed: %d\n", r);
67                         exit(1);
68                     }
69                     *endpoint_address = inter->endpoint[0].bEndpointAddress;
70                     goto found;
71                 }
72             }
73     }
74 }
75
76 found:
77     libusb_free_device_list(devs, 1);
78
79     return keyboard;
80 }

```

#### A.2.14 usbkeyboard.h

```

1 #ifndef _USBKEYBOARD_H
2 #define _USBKEYBOARD_H
3
4 #include <libusb-1.0/libusb.h>
5
6 #define USB_HID_KEYBOARD_PROTOCOL 1
7
8 /* Modifier bits */
9 #define USB_LCTRL (1 << 0)
10 #define USB_LSHIFT (1 << 1)
11 #define USB_LALT (1 << 2)
12 #define USB_LGUI (1 << 3)
13 #define USB_RCTRL (1 << 4)
14 #define USB_RSHIFT (1 << 5)

```

```

15 #define USB_RALT    (1 << 6)
16 #define USB_RGUI   (1 << 7)
17
18 struct usb_keyboard_packet {
19     uint8_t modifiers;
20     uint8_t reserved;
21     uint8_t keycode[6];
22 };
23
24 /* Find and open a USB keyboard device.  Argument should point to
25    space to store an endpoint address.  Returns NULL if no keyboard
26    device was found. */
27 extern struct libusb_device_handle *openkeyboard(uint8_t *);
28
29 #endif

```

### A.2.15 vga\_ball.c

```

1  //Device driver for the VGA video generator
2
3
4  #include <linux/module.h>
5  #include <linux/init.h>
6  #include <linux/errno.h>
7  #include <linux/version.h>
8  #include <linux/kernel.h>
9  #include <linux/platform_device.h>
10 #include <linux/miscdevice.h>
11 #include <linux/slab.h>
12 #include <linux/io.h>
13 #include <linux/of.h>
14 #include <linux/of_address.h>
15 #include <linux/fs.h>
16 #include <linux/uaccess.h>
17 #include "vga_ball.h"
18
19 #define DRIVER_NAME "vga_ball"
20
21 /* Device registers */
22 #define BYPASS(x) (x) //bypass switch on/off (0/1)
23 #define SELY(x) ((x)+2) //FIR PRESET SELECT LOC Y
24 #define SLY1(x) ((x)+4)//SLIDER 1 CLIP
25 #define SLY2(x) ((x)+6)//SLIDER 2 DELAY
26 #define SLY3(x) ((x)+8)//SLIDER 3 MIX
27 #define SLY4(x) ((x)+10)//SLIDER 4 BASS
28 #define SLY5(x) ((x)+12)//SLIDER 5 MID
29 #define SLY6(x) ((x)+14)//SLIDER 6 TRE
30
31 /*
32  * Information about our device
33  */
34 struct vga_ball_dev {
35     struct resource res; /* Resource: our registers */
36     void __iomem *virtbase; /* Where registers can be accessed in memory */
37     vga_ball_color_t background;
38 } dev;
39
40 /*
41  * Write segments of a single digit
42  * Assumes digit is in range and the device information has been set up
43  */
44 static void write_background(vga_ball_color_t *background)

```

```

45 {
46     iowrite16(background->bypass, BYPASS(dev.virtbase) );
47     iowrite16(background->sely, SELY(dev.virtbase) );
48     iowrite16(background->sly1, SLY1(dev.virtbase) );
49     iowrite16(background->sly2, SLY2(dev.virtbase) );
50     iowrite16(background->sly3, SLY3(dev.virtbase) );
51     iowrite16(background->sly4, SLY4(dev.virtbase) );
52     iowrite16(background->sly5, SLY5(dev.virtbase) );
53     iowrite16(background->sly6, SLY6(dev.virtbase) );
54     dev.background = *background;
55 }
56
57 /*
58  * Handle ioctl() calls from userspace:
59  * Read or write the segments on single digits.
60  * Note extensive error checking of arguments
61  */
62 static long vga_ball_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
63 {
64     vga_ball_arg_t vla;
65
66     switch (cmd) {
67
68     case VGA BALL_WRITE_BACKGROUND:
69         if (copy_from_user(&vla, (vga_ball_arg_t *) arg,
70                             sizeof(vga_ball_arg_t)))
71             return -EACCES;
72         write_background(&vla.background);
73         break;
74
75
76
77     case VGA BALL_READ_BACKGROUND:
78         vla.background = dev.background;
79         if (copy_to_user((vga_ball_arg_t *) arg, &vla,
80                             sizeof(vga_ball_arg_t)))
81             return -EACCES;
82         break;
83
84     default:
85         return -EINVAL;
86     }
87
88     return 0;
89 }
90
91
92 /* The operations our device knows how to do */
93 static const struct file_operations vga_ball_fops = {
94     .owner          = THIS_MODULE,
95     .unlocked_ioctl = vga_ball_ioctl,
96 };
97
98 /* Information about our device for the "misc" framework -- like a char dev */
99 static struct miscdevice vga_ball_misc_device = {
100     .minor          = MISC_DYNAMIC_MINOR,
101     .name           = DRIVER_NAME,
102     .fops           = &vga_ball_fops,
103 };
104
105 /*
106  * Initialization code: get resources (registers) and display

```

```

107  * a welcome message
108  */
109  static int __init vga_ball_probe(struct platform_device *pdev)
110  {
111      vga_ball_color_t beige = { 0,215, 215,215,215,215,215,215 };
112      int ret;
113
114      /* Register ourselves as a misc device: creates /dev/vga_ball */
115      ret = misc_register(&vga_ball_misc_device);
116
117      /* Get the address of our registers from the device tree */
118      ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
119      if (ret) {
120          ret = -ENOENT;
121          goto out_deregister;
122      }
123
124      /* Make sure we can use these registers */
125      if (request_mem_region(dev.res.start, resource_size(&dev.res),
126                          DRIVER_NAME) == NULL) {
127          ret = -EBUSY;
128          goto out_deregister;
129      }
130
131      /* Arrange access to our registers */
132      dev.virtbase = of_iomap(pdev->dev.of_node, 0);
133      if (dev.virtbase == NULL) {
134          ret = -ENOMEM;
135          goto out_release_mem_region;
136      }
137
138      /* Set an initial color */
139      write_background(&beige);
140
141      return 0;
142
143  out_release_mem_region:
144      release_mem_region(dev.res.start, resource_size(&dev.res));
145  out_deregister:
146      misc_deregister(&vga_ball_misc_device);
147      return ret;
148  }
149
150  /* Clean-up code: release resources */
151  static int vga_ball_remove(struct platform_device *pdev)
152  {
153      iounmap(dev.virtbase);
154      release_mem_region(dev.res.start, resource_size(&dev.res));
155      misc_deregister(&vga_ball_misc_device);
156      return 0;
157  }
158
159  /* Which "compatible" string(s) to search for in the Device Tree */
160  #ifdef CONFIG_OF
161  static const struct of_device_id vga_ball_of_match[] = {
162      { .compatible = "csee4840,vga_ball-1.0" },
163      {}
164  };
165  MODULE_DEVICE_TABLE(of, vga_ball_of_match);
166  #endif
167
168  /* Information for registering ourselves as a "platform" driver */

```



```

169 static struct platform_driver vga_ball_driver = {
170     .driver = {
171         .name      = DRIVER_NAME,
172         .owner     = THIS_MODULE,
173         .of_match_table = of_match_ptr(vga_ball_of_match),
174     },
175     .remove = __exit_p(vga_ball_remove),
176 };
177
178 /* Called when the module is loaded: set things up */
179 static int __init vga_ball_init(void)
180 {
181     pr_info(DRIVER_NAME ":_init\n");
182     return platform_driver_probe(&vga_ball_driver, vga_ball_probe);
183 }
184
185 /* Calball when the module is unloaded: release resources */
186 static void __exit vga_ball_exit(void)
187 {
188     platform_driver_unregister(&vga_ball_driver);
189     pr_info(DRIVER_NAME ":_exit\n");
190 }
191
192 module_init(vga_ball_init);
193 module_exit(vga_ball_exit);
194
195 MODULE_LICENSE("GPL");
196 MODULE_AUTHOR("Stephen A. Edwards, Columbia University");
197 MODULE_DESCRIPTION("VGA_ball_driver");

```

#### A.2.16 vga\_ball.h

```

1  #ifndef _VGA BALL_H
2  #define _VGA BALL_H
3
4  #include <linux/ioctl.h>
5
6  typedef struct {
7      short unsigned int bypass, sely, sly1, sly2, sly3, sly4, sly5, sly6;
8  } vga_ball_color_t;
9
10
11
12 typedef struct {
13     vga_ball_color_t background;
14 } vga_ball_arg_t;
15
16 #define VGA BALL_MAGIC 'q'
17
18 /* ioctls and their arguments */
19 #define VGA BALL_WRITE_BACKGROUND _IOW(VGA BALL_MAGIC, 1, vga_ball_arg_t *)
20 #define VGA BALL_READ_BACKGROUND _IOR(VGA BALL_MAGIC, 2, vga_ball_arg_t *)
21
22 #endif

```