EE 552 Project
Smart Audio Equalizer
Final Report

David R. Bull
bull@ee.ualberta.ca

Dustin R. Demontigny
demontig@ee.ualberta.ca

March 27, 2003

Declaration of Content

The design elements of this project and report are entirely the original work of the authors and have not been submitted for credit in any other course, except for the following:

- We will be using VHDL code from [5] - Inputting and Outputting Stereo Signals Through the Codec.

David R. Bull

Dustin Demontigny

Date

Abstract

This report describes the design and implementation of a smart audio equalizer system. The equalizer is a 2-channel, 10 band audio equalizer using the XSA-100 + Xstend prototyping board. The stereo audio codec on the XStend board digitizes the analog audio and constructs a digital bitstream. The audio equalizer can be customized by the user to best suit his or her individual hearing characteristics. The system optimizes the audio signal generated in real time to best suit each listener. The hardware components used are described in detail as well as the current VHDL source code including simulation waveforms and test benches.

Contents

1 Achievements **1**

2 Introduction **1**

3 Design Concept **2**

4 Stereo Audio Codec **3**
- 4.1 Codec Interface 3
- 4.2 Clock Generator 4

5 Filter Subsystem **4**

6 RAM Loader **5**

7 User Interface **5**
- 7.1 Description 5
- 7.2 Implementation 6

8 Arithmetic Unit **7**
- 8.1 Description 7
- 8.2 Implementation 7

9 FPGA Requirements **11**

10 Experiments and Characterization **12**

11 External Sources **14**

12 Datasheets **16**

13 Design Verification **18**

14 Design Hierarchy **55**

15 VHDL Code **57**

16 Test Benches **130**

17 Schematics **155**

18 Inputting/Outputting Stereo Signals Through the Codec **158**

1 Achievements

The first design of our digital filter module was not very good. The old design instantiated a multiplier and adder for each filter tap and consumed a large portion of the logic resources. A second filter design implements a larger filter with higher resolution at the cost of 1/3 the hardware. This was a very good achievement.

The new filter design works beautifully in simulation, unfortunately, at this point we have been unable to get it working on the board. We have spent many hours debugging the problem but have not yet reached a solution. We are hopeful that we will get something working before the end of the term.

We were successful, however, in getting the first design of our digital filter to produce valid output given a set of test data. This experiment is outlined in the testing section of this report.

The current version of the user interface successfully reads the inputs from 3 pushbuttons to control the state and gain settings of the smart equalizer system. The user toggles between the preset value using the incr or decr pushbuttons. The system can be reset with the third push button. The preset number is displayed using the seven segment LED display on the left side of the XStend board. The gain vector is sent to the arithmetic unit.

An arithmetic unit has been added to the smart equalizer system since the creation of the simulation report. This section of the code evaluates the coefficients needed by the filter block using the tap and gain values. This section has no connection with the outside world. It serves as the interface between the user I/O and the filter block. The description of operation is described in more detail in the following sections. Test benches used to simulate the arithmetic unit are attached in the Design and Verification section.

2 Introduction

The human ear responds differently to sounds at various frequencies. In general, the ear is most sensitive to sounds at frequencies near 1kHz and becomes increasingly insensitive as the the frequency deviates from this range[1]. Using an equalizer we can amplify the components of the signal we have difficulty hearing. Our smart equalizer will allow the user to configure the system to best suit his or her individual hearing characteristics providing the user with the best listening experience.

3 Design Concept

The proposed project will incorporate the XSA-100 + XStend project board, with an on-board stereo audio codec, to implement a 2-channel, 10-band audio equalizer. The gain for each band will be adjusted manually by the user. The top level diagram of the system is shown in Figure 1.

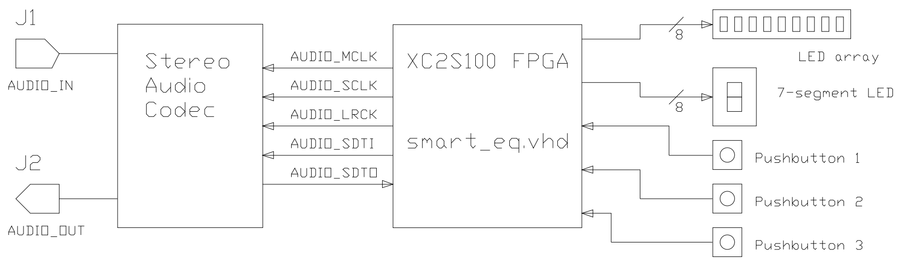

Figure 1: Top Level Diagram

The system inputs a stereo audio signal into the stereo codec. The codec runs off of three clock signals provided by the circuitry in the FPGA; *mclk*, *sclk* and *lrck*. The digitized audio samples are fed into the FPGA through the *sdout* pin. The top level entity in the FPGA, *smart_eq.vhd*, will filter the input samples and send them serially to the audio codec via the *sdin* pin. The reconstructed analog audio signal is output from the audio codec.

User input will be obtained with the use of pushbuttons and dip-switches. There are four on-board pushbuttons, only three of which are accessible for general purpose input, and 12 dip-switches. Since only three pushbuttons is insufficient for this system, we will use the dipswitches to set a “mode” for which the pushbuttons will operate. This way, we can effectively create

$3 * 2^n$ pushbutton inputs, using 3 bushbuttons and n dip-switches. We will only use one dip-switch.

A 7-segment LED display and an 8-segment LED bargraph will be used do display the status of the system and to guide the user.

The equalizer system is divided into three main sections:

1. A codec interface (*codec_intfc.vhd*)
2. A filter module (*filter.vhd*)
3. A user interface (*user_io.vhd*)

The operation of these subsystems is described in the sections that follow.

4 Stereo Audio Codec

The XStend board includes a 20-bit stereo audio codec that digitizes a two channel analog signal while simultaneously reconstructing a two channel analog signal. A high level diagram of the codec chip is shown in [5]. The codec chip includes a series of 20-bit shift registers used to serially transfer sample values to and from the FPGA. The codec interface will also include a series of 20-bit shift registers as well as circuitry to read and write to them.

4.1 Codec Interface

The XStend Board V1.3.2 User Manual includes documentation and VHDL code for a codec interface circuit (*codec_intfc.vhd*) used to send and received data to and from the stereo audio codec. This document is included in Appendix 18. Our system will incorporate this codec interface circuit.

The codec interface is composed of two modules:

- A clock generator module (*clk_gen.vhd*) which generates and outputs the codec clock signals, and
- A channel module (*channel.vhd*) which contains the shift registers and control circuitry for a single serial stream of input/output data.

These two modules are integrated together to form the codec interface circuit.

The sample code was revised to change the reset signal from active-high asynchronous to active-low synchronous, since the reset signal will be triggered using an active-low pushbutton. Also, the *bit_cntr* signal was included as an output of the *codec_intfc* module so that it can be used in the filter module. The revised code is included in Appendix 15.

4.2 Clock Generator

Three clock signals must be provided to the codec chip.

1. *mclk* - master clock to codec
2. *sclk* - serial data clock
3. *lrck* - left/right codec channel select

For sampling frequency f_s, the clock signals are related as follows:

$mclk = 256f_s$
$sclk = 64f_s$
$lrck = f_s$

Taking $mclk = 100MHz/8 = 12.5MHz$, we get a sampling frequency of $f_s = 48.8KHz$.

5 Filter Subsystem

Consider the simplified schematic on the following page. The filter is composed of two counters (counter and counterd), two single port distributed RAMs (distram), a signed multiplier (fmult) and an accumulator (facc). One RAM holds the filter coefficients and the other holds the sample values. The RAM holding the sample values is twice the size of the RAM holding the filter coefficients. The top half holds the sample values for the left channel and the bottom half holds the sample values for the right channel. The input to the coefficient RAM is fed by a RAM loader, which is discussed in the following section. The sample RAM inputs digitized samples from the codec interface.

The counters hold the RAM addresses. When an audio sample is received from the codec interface it is loaded into the sample RAM and the address counters are enabled. The RAMs loop throught and multiply the coefficients with the corresponding sample values. The result is stored in the accumulator. The accumulator is reset after the value at the output of the filter is read.

6 RAM Loader

A device which we call a RAM loader is used to load the filter coefficients into the RAM. See the following page for a simplified schematic. A slightly modified RAM loader is also used to initialize the contents of the sample RAM. The RAM loaders use several handshaking signals. When data is ready to be loaded into the RAM the *data_in_rdy* signal is asserted. The RAM loader requests control of the RAM control signals (address, write enable, etc.) by asserting *cont_req*. The RAM is granted control when the *cont_ack* signal is asserted. And finally, when the RAM loader has finished it asserts both the *cont_rel* and *data_in_ack* signals.

7 User Interface

7.1 Description

The smart equalizer has a user interface that allows interaction between the equalizer parameters and the outside world. The hardware components used in the user interface include 3 pushbuttons and a seven segment LED display, which are all accessible on the XStend board.

The user can select a preset value for the band gains. Two pushbuttons are used to cycle up or down between the desired preset values. Preset 0 will create an equalizer with a flat response (unity gain). Preset 1 will correspond to user 1 (i.e.Dave's setting), preset 2 to user 2 and so on. The current preset will be displayed on the 7 segment display.

The user interface manipulates input data from the outside world and stores band gain values in a register for use in the filter subsystem. It is important to note here that the User Interface is not channel dependent. In other words it applies the same gain to the left channel as it does to the right channel.

7.2 Implementation

The user interface incorporates four main components: a debouncer, a preset counter, a preset decoder and a bcd decoder. These four components are instantiated structurally in one VHDL file user_io.vhd which is the top level for the user interface. The project package file, smart_eq_pack.vhd holds all of the constants and component declarations used in the user interface.

The debounce entity described in debouncer3.vhd removes unwanted transients from the pushbutton inputs creating stable pulses of varying width. It does this by counting the number of clock cycles that occur. When the number of clock cycles (count) reaches the required value set by the constant (debounce_delay), the input signals are mapped to the output. This method works well as long as the debounce delay is greater than the maximum bounce duration of the pushbuttons. It is important to mention here that a debounce_delay of 7 was used for simulation purposes whereas a debounce duration of 1023 will be used when implementing the design on the FPGA. The outputs from the debouncer are sent to the input of the preset counter.

The preset counter is decribed in preset_counter.vhd. This component reads the input pushbuttons to set the desired preset value. It works similar to an up down counter. When (incr) is pressed the preset value is incremented by one. Similarily, when (decr) is pressed the preset value is decremented by one. A synchronous reset button is used to set the preset value to zero. A very important requirement is to allow the preset value to increment or decrement only once for every press cycle. This requirement is fullfilled by using (press) variables to be used on the pushbuttons. These variables act as a flags to indicate when the pushbutton signals goes low. These flags are then checked whenever the pushbuttons are not low. If the flags are set, then the preset value is incremented or decremented accordingly and the flag is reset to zero. It is important to note here that this system relies on only one pushbutton being pressed at one time. This requirement is acceptable considering the manner in which the user interface is used.

The output from the preset counter is sent to two decoders: a bcd decoder and a preset decoder. These are described in files bcd_decoder.vhd and preset_decoder.vhd respectively. The bcd decoder takes the value of the preset counter and converts it to a decimal number display on the seven segment LED display. The preset decoder converts the value of the preset counter to a vector of gain values. This vector is sent to the arithmetic unit to calculate the resulting filter coefficients for the filter block.The configuration of the LED segments and the preset values are stored in the package file smart_eq_pack.vhd and can be changed or updated as needed.

8 Arithmetic Unit

8.1 Description

The smart equalizer uses an arithmetic unit to calculate the filter coefficients. The inputs to the arithmetic unit include the gain values for all bands from the user interface and the hard coded filter taps listed in the top level package file smart_eq_pack.vhd. The arithmetic unit calculates the values of each of the filter coefficients and sends them to the filter block eq. The arithmetic unit has no external FPGA connections.

8.2 Implementation

The arithmetic unit uses five components: two temporary registers, a multiplier, an accumulator and a register controller . These five components are instantiated structurally in one VHDL file arith_unit.vhd. The project package file, smart_eq_pack.vhd holds all of the constants and component declarations used in the arithmetic unit.

The first temporary register is used to shift the gain vector input and output the require gain of each band, one at a time. The signal aload inputs the whole vector into the register. The clock signal shifts the register to the right by one gain value (usually 4 bits). The output is the least significant 4 bits, which is equivalent to the gain of the selected band.

Every time the register switches from one band to the next, the coef_value changes. This is performed using a process inside the arithmetic unit that increments the tap number. The value of signal coef_value is equal to the tap_number element in the tap array declared in the package file. Essentially this performs as an on chip ROM device.

The multiplier takes the coef_value from the on chip ROM and multiplies it by the band gain. The output of the multiplier has the same number of bits as the signal coef_value. It does this by simply truncating the output value by the number of bits in the band gain value.

The accumulator takes the output from the multiplier and creates a running sum of the signal vectors. It performs as many accumulations as there are bands (usually 10). Once it has performed these ten accumulations, it is cleared to zero and starts accumulating the values for the next coefficient. The output from the accumulator is sent to the second temporary register.

The second temporary register grabs the coefficient values from the accumulator and stores them in an array of vectors which are sent to the filter block. The array length is determined by the number of coefficients (number of filter taps plus one). The length of the coefficients is determined by the resolution of the filter (usually 12 bits).

The arithmetic unit is controlled by the entity reg_control.vhd. It performs similar to a state machine. It controls the loading and shifting of the first temporary register as wells as the accumulation function and the storage of the coefficients stored in the second temporary register. After each individual coefficient is loaded into the array of vectors in the second temporary register, it shifts the gain to the beginning (least significant four bits), clears the accumulator, and starts calculations for the next coefficient.

9 FPGA Requirements

The revised filter design, as predicted, has drasticly reduced the logic requirements for our project.

The first filter design implemented two discrete 6^{th} order FIR filters with 16-bit audio samples and 8-bit filter coefficients. Each filter contained 7 16×8 bit multipliers (one for each filter tap) and 7, 20-bit adders to sum the products. Obviously the resource requirements for this design were quite significant. This design used over 90% of the logic resources (including codec interface).

The new filter design implements two 10^{th} order FIR filters using circular buffers in RAM and a single multiplier and accumulator. We use 20-bit audio samples and 12-bit filter coefficients. The filter, codec interface and control circuitry for this design only consume 26% of the logic resources. In this case we have essentially doubled the size of the filter and we use only 1/3 the hardware.

The second module is the user interface. This module tests the functionality of the user input pushbuttons and dip switch and verifies proper operation using the seven segment display, LED bargraph. This module consumes 15% of the FPGA resources.

The user interface was also redesigned. The original architecture did not seperate the module into a controller and datapath. It consumed 27% of the FPGA resources, which was much more than originally scheduled. This required us to rethink the design and change the interface to toggle between preset values. The redesign, although costly in terms of timeline, resulted in a significant decrease in FPGA resources. The current version of the user interface (user_io.vhd), which uses a preset counter, uses 14% of the FPGA requirements. This is reasonable considering the size of the gain register. The user interface could be enhanced if time and space permits. Reducing the size of the user interface would be quite difficult. Lowering the debounce delay, optimizing the preset counter and using RAM to store the gain values could possibly reduce the FPGA resource requirements of the user interface.

An arithmetic unit has been added to the smart equalizer system since the writing of the simulation documentation. This module is used to store the tap values and evaluate the filter coefficients using the gain values. This unit stores the tap values in a ROM listed in the top level package file smart_eq_pack.vhd. The arithmetic unit uses 17% of the logic resources of the FPGA.

The top level smart equalizer system occupies 53% of the logic blocks of the FPGA. This is much less than the original filter and user interface design which took up 90% and 27% respectively. This is quite an improvements since the original filter was for a single band, and the user interface was slow and unreliable. The remaining 47% of the FPGA could be used to implement the enhancements listed in previous reports. Examples of these further enhancements include real time automatic optimization and VGA graphic equalizer display.

10 Experiments and Characterization

Since the new filter design uses RAM we tested the two different RAM types, distributed RAM and block RAM. Block RAM uses dedicated RAM blocks (there are 10 4K RAM blocks in the XC2S100 FPGA) and distributed RAM is implemented in the logic resources. Since we only use approximately 840 bits of RAM in total we chose to use distributed RAM. It was also easier to use the distributed RAM since the RAM can be read asynchronously. The implementation of the distributed RAM used minimal logic resources (i.e. only a few percent).

The structure of the user interface changed drastically during the development of the smart equalizer. Originally, the whole system including datapath and controller were linked into one file. The inputs were debounced in a seperate entity and instantiated structually in the top level. The original layout used 27% of the logic cells on the FPGA and operated quickly, although unreliably. For this reason, the datapath and controller of the user interface were seperated into datapath and controller sections in order to simply the layout and improve stability.

Two debounce mechanisms were designed in order to determine the best alternative. One used a counter that started counting once the signal went high. After the counter reached a certain value (say 1023) the output went high if the input remained high during the count sequence. Otherwise the output stayed low. This design is demonstrated in debouncer.vhd. This implementation used 17 logic blocks in the FPGA.

The second debounce mechanism used an enable signal that was pulsed high every couple thousand clock cycles. The input is latched to the output on the rising edge of this enable signal. This method proved to be much more efficient than the previous design using only 12 logic blocks. The VHDL code for this debouncer can be seen in debouncer2.vhd. Both debouncers are used during the simulation and testing of the project components to observe how they affect the reliability and operation of the system.

Both versions of the debouncer use the same amount of clock cycles to delay the input. This allowed us to determine that the second implementation was more efficient since it used less of the FPGA resources but operated at the same speed. The speed of the user interface isn't as critical as it's reliablity since it is used to manipulate the parameters of the filters in the equalizer.

The user interface was changed later to reduce FPGA requirements and speed up operation. The current version of the user interface occupies approximately 15% of the logic resources of the FPGA. This is just over half of the original version. Most of the large registers and MUX devices were removed. This limited the abilities of the user but simplified the design creating a mechanism that was more reliable. The reduced size on the IC is also a bonus.

During the simplification of the filter system, the coefficient requirements were changed. The updated filter only needs as many coefficients as there are taps. This reduces the previous amount by a factor equal to the number of bands (in our case 10). The new coefficients are calculated using the arithmetic unit. This unit occupies 17% of the FPGA which is not very much considering it stores all of the tap values and stores the resultant coefficient. The arithmentic unit provides the interface between the filter and the user interface.

11 External Sources

As mentioned previously, the XStend Board V1.3.2 Manual includes the VHDL code for a codec interface circuit. This module is used in our system with slight modifications. The modified VHDL source code (loopback.vhd, codec_intfc.vhd, clkgen.vhd and channel.vhd) can be found in Appendix 15.

Modifications to the sample code include changing the reset signal from active-high asynchronous to active-low synchronous, since the reset signal will be triggered using an active-low pushbutton. The *bit_cntr* signal was included as an output of the *codec_intfc* module so that it can be used in the filter module.

When compiling the sample code we came across an error. In the loopback.vhd file the signals *left_channel* and *right_channel* were only allocated 8 bits when they were expected to carry 20-bit signals. A simple correction of the vector indicies corrected the problem.

References

[1] Multimedia Signals and Systems, Mrinal Kr. Mandal, Kluwer Academic Publishers, 2003

[2] Signal Processing and Linear Systems, Lathi, B.P., Berkeley Cambridge Press, 1998.

[3] Discussions with Dr. Behrouz Nowrouzian, Electrical and Computer Engineering, University of Alberta.

[4] Implementation of an Audio Filter on an FPGA Board, Herbertz, Kay, March 2002.

[5] XStend Board V1.3.2 Manual, XESS Corporation, 2001.

[6] Xstend Board V2.0 Manual, XESS Corporation, 2002.

[7] Xilinx Synthesis Technology (XST) User Guide, Xilinx Inc.

12 Datasheets

The only device peripheral to the FPGA is the on-board stereo audio codec. The codec features a 20-bit analog to digital converter and digital to analog converter. The digitized audio samples are read from and written to serially. The coverpage for the codec datasheet is included on the following page. A schematic of the interface to the codec chip is included in Appendix 17.

13 Design Verification

Index of Test Cases

Case 1:	Board hardware is tested by displaying counter value on the on-board LED displays	19
Case 2:	On-board audio codec is tested using the loopback circuit provided in XStend V1.3.2 Manual	19
Case 3:	Loopback circuit is tested with real audio signals	19
Case 4:	Simulation of the codec interface circuit	19
Case 5:	Simulation of the codec interface circuit integrated with a single band FIR filter	21
Case 6:	Simulation and verification of debouncer3	23
Case 7:	Simulation and verification of preset counter	25
Case 8:	Simulation and verification of user interface (user_io)	27
Case 9:	Test bench verification of the filter module	29
Case 10:	Simulation verification of the RAM loaders	36
Case 11:	Simulation verification of the new filter	38
Case 12:	Test bench verification of the new filter	42
Case 13:	Test bench for the first temporary register	45
Case 14:	Test bench for the multiplier	47
Case 15:	Test bench for the accumulator	49
Case 16:	Test bench for the entire arithmetic unit	51
Case 17:	Debugging test bench for smart eq system	53

Case 1

The first experiment was to implement a 4-bit counter whose value is decoded through a BCD to 7-segment decoder and displayed on all on-board 7-segment displays. The circuit functioned properly with one exception. Segment 6 on one of the LED displays did not light up at all. It was discovered that in order to use the pin connected to this segment, the on-board CPLD needed to be reprogrammed. Since we do not know how to program the CPLD properly and since we do not need to use this particular LED display in our system, we did not investigate this problem any further.

Case 2

The second experiment was to implement the loopback circuit provided in the XStend V1.3.2 manual. The code from the manual was copied from the manual into a text file and compiled. There was, however, an error in this code that needed to be fixed which prevented a successful compilation. The signals *left_channel* and *right_channel* in the *loopback.vhd* file were only allocated 8 bits when they were expected to pass 20-bit signals. A simple correction of the signal indices corrected the problem. The code was also modified to change the active-high synchronous reset to active-low asynchronous, since an active-low synchronous pushbutton was used to generate the reset signal.

Case 3

To verify the operation of the loopback circuit we input an audio signal from a CD player and played the output audio from the codec on a stereo system. The loopback circuit worked properly as the audio was unaltered by the loopback circuitry.

Case 4

Simulation of the codec interface circuit was performed to analyze the timing requirements. The simulation waveform is included on the following page.

Case 5

Next a single-band digital filter was integrated with the loopback circuit. The input samples from the codec are passed through the filter, then sent back to the codec for reconstruction. Simulation of the integrated circuit confirmed proper operation, however, the hardware has not yet been tested with an audio signal. The simulation waveform is included on the following page.

Case 6 - Test Debouncer

The test_db test bench sends various inputs to the debouncer. This ensures that the output signals are only assigned the input values every n * clock_period seconds where n is equal to the constant debounce_delay, which is assigned in user interface package file. For simulation purposes a value of 7 was used for the debounce delay. In the actual implementation a value of 1023 will be used. This shouldn't affect the operation of the user interface since speed isn't a priority.

Case 7 - Test User I/O

The test_io test bench is used to test the entire user interface module. The test bench models noisy pushbutton inputs to see how the interface would react. It can be seen that the noisy inputs become smooth pulses of vary width depending on the timing of the inputs. These pulses trigger the preset counter to increment or decrement. The preset counter increments or decrements only once for every pulse. The two decoders are able to convert the preset value to display the hexadecimal digit on the seven segment display and store the gain value.

Case 8 - Test Preset Counter

The test_preset_counter test bench simulates the operation of the preset counter using variable pulse widths. It can be seen in the following waveform that the preset counter properly increments and decrements the preset value only oncer for every input pulse. The resulting output preset value will be stable as long as the inputs are not noisy. This requirement is fulfilled using the debouncer mechanism.

Case 9 - Test Filter Module - Old Design

The filt_test bench emulates the on-board stereo audio codec and generates the master clock and global reset signals. It is used to verify that the filtered samples output from the bandpass filter are correct.
We begin the experiment by generating 30 random 20-bit numbers using Microsoft Excel. The numbers are then processed using MATLAB to calculate the correct filter output samples. The matlab code is provided below.

```
% this matlab program generates the output of a FIR digital filter y(n)
% given input x(n) and filter coeffcients b(i)
% the input and output samples are written to xy.txt

clear;
b=[1 -6 18 101 18 -6 1];
x=[0 0 0 0 0 0 0];
x1=[  71  60 -86 -100 -24  22 14 119   82 0];
x2=[  42  59   6  -76 111 -66 -1  31 -117 39];
x3=[ -22 -52 126   99 105 121 71  92  -28 28];
xin=2^12*[x1 x2 x3];
for i=1:30,
   x=[xin(i) x(1:6)];
   y(i)=floor(sum(x.*b)/2048);
end
fid1=fopen('xin.txt','w');
fid2=fopen('y.txt','w');
fid3=fopen('xy.txt','w');
fprintf(fid3,'Input Samples        Output Samples\n\n');
for j=1:30,
   fprintf(fid1,'x(%i) = %9i\n',j-1,xin(j));
   fprintf(fid2,'y(%i) = %7i\n',j-1,y(j));
   fprintf(fid3,'x(%2i) = %9i,    y(%2i) = %7i\n',j-1,xin(j),j-1,y(j));
end
fclose(fid1);
fclose(fid2);
fclose(fid3);
```

The matlab program generates an output file *xy.txt* containing two columns. This file is included on page 31. The first column contains the input sample values and the second column contains the corresponding output sample values. The sample values are 20-bit signed numbers and are displayed in base 10 for clarity.

The randomly generated input samples were copied into the test bench file as an array of constants. The test bench loads the sample values in turn into a shift register on the rising edge of the *lrck_tb* signal so that the sample values are the same for both the left and right channels. This is not necessary, it is done to simplify the verification process. The samples are then serially shifted into the filter on the rising edge of the *sclk_tb* signal.

As the input samples are shifted out, the filtered samples from the previous input samples are shifted in. The filtered samples are loaded into an array on the rising edge of the *lrck_tb* signal.

The bandpass filter features an 'on/off' switch that toggles the filter coefficients between two predefined sets of values. For this test case the bandpass filter (bndfilt.vhd) is set in 'off' mode. The corresponding filter is a 6th order lowpass FIR filter with a normalized cutoff frequency of 0.8 (see bndfilt.vhd).

The waveforms generated from the simulated test bench are included on the following pages. Comparison to the results listed in the *xy.txt* file confirm that the filter is executing correctly.

```
Input Samples        Output Samples

x( 0) =    290816,    y( 0) =     142
x( 1) =    245760,    y( 1) =    -732
x( 2) =   -352256,    y( 2) =    1664
x( 3) =   -409600,    y( 3) =   17334
x( 4) =    -98304,    y( 4) =   12732
x( 5) =     90112,    y( 5) =  -19332
x( 6) =     57344,    y( 6) =  -24974
x( 7) =    487424,    y( 7) =   -6434
x( 8) =    335872,    y( 8) =    3848
x( 9) =         0,    y( 9) =    7008
x(10) =    172032,    y(10) =   27266
x(11) =    241664,    y(11) =   20338
x(12) =     24576,    y(12) =    2368
x(13) =   -311296,    y(13) =    9638
x(14) =    454656,    y(14) =   14944
x(15) =   -270336,    y(15) =   -1368
x(16) =     -4096,    y(16) =  -10974
x(17) =    126976,    y(17) =   17430
x(18) =   -479232,    y(18) =   -9054
x(19) =    159744,    y(19) =   -1464
x(20) =    -90112,    y(20) =    2516
x(21) =   -212992,    y(21) =  -21074
x(22) =    516096,    y(22) =    3376
x(23) =    405504,    y(23) =   -4760
x(24) =    430080,    y(24) =   -8440
x(25) =    495616,    y(25) =   26468
x(26) =    290816,    y(26) =   27584
x(27) =    376832,    y(27) =   26846
x(28) =   -114688,    y(28) =   28682
x(29) =    114688,    y(29) =   21340
```

Case 10 - RAM Loader

The following simulation verifies the operation of the RAM loaders used on the coefficient RAM and the sample RAM. First the coefficient RAM is loaded with 10 coefficients. Then the sample RAM is initialized by writing zero to all of the RAM locations. The annotated waveform is included on the following page.

Case 11 - FIR Filter - New Design

The following simulation verifies the operation of the new filter design. Six audio samples are sent through the filter (three for the left channel and three for the right). The filter uses the coefficients loaded in the previous simulation. The annotated waveform is included on the following page.

Case 12 - Filter Test Bench

The following simulation verifies the operation of the filter. A test bench is used that simulates the codec chip. The test bench generates the appropriate clock signals and sends and recieves audio samples to and from the filter serially. The input and output samples are stored in an array for easy comparison. For the purposes of easy comparison the digital filter coefficients were set such that the output of the filter is equal to the current input sample. Essentially, the filter acts like a loopback circuit for the purposes of this test. The simulation waveform is included on the following page.

Case 13 - Temporary Register Test Bench

The waveform on the following page illustrates how the temporary register at the input to the arithmetic unit functions. When the aload signal goes high, the gain value from the user interface is loaded in parallel to register. Otherwise, the data in the register is shifted to the right every rising edge of the clock cycle by 4 bits (the size of the gain value for each band). The least significant 4 bits of the register are sent to the output. This value is equal to the gain of the current band. The band values are therefore output from lowest band to highest band.

Case 14 - Multiplier Test Bench

The test bench for the multiplier simply checks to make sure the output corresponds to the input values multiplied. It also checks that the output is truncated to only output 12 bits (the size of our coefficients). The waveform on the following page shows that the multiplier works porperly.

Case 15 - Accumulator Test Bench

The waveform on the next page shows the proper operation of the accumulator section. The input value is added to the output every rising edge of the clock. The output asynchronously resets the to zero whenevery the clear signal goes high.

Case 16 - Arithmetic Unit Test Bench

The following simulation waveforms shows the operation of the entire arithmetic unit. The accumulator adds the values of the outputs from the multiplier and sends the output to be stored in temporary register every complete run through all band gains and tap values. This system works well but could be optimized to reduce the evaluation time of each coefficient.

Case 17 - Smart Eq Test Bench debugging

The following waveform shows how the smart eq test bench was used to debug system problems. As shown, the last element of the ceof_register is not stored. The index element was incremented by one to allow the last coefficient to be stored. The simulation was rerun and the change managed to fix the problem. This illustrates that modular design and testing techniques ease the troubleshooting and debugging steps in the project development.

14 Design Hierarchy

Included on the following page is a diagram of the design hierarchy.

15 VHDL Code

Index

accumulator.vhd	(simulated)	accumulates multiplier output in arithmetic unit	58
arith_unit.vhd	(simulated)	logic block used calculate the coefficient values	59
bcd_decoder.vhd	(simulated)	converts binary to decimal digit on 7 segment display	61
channel.vhd	(simulated)	data channel used in codec interface module	62
clkgen.vhd	(simulated)	clock generator for codec	66
codec.vhd	(compiled)	package used by codec modules	68
codec_intfc.vhd	(simulated)	interface module to codec	71
counter.vhd	(simulated)	up counter with definable wrap around value	75
counterd.vhd	(simulated)	down counter with definable wrap around value	77
debouncer3.vhd	(simulated)	uses clock delay to scan inputs and debounce signals	79
distram.vhd	(compiled)	single port distributed RAM	80
eq.vhd	(simulated)	entity containing filter, codec interface and control circuitry	81
facc.vhd	(compiled)	signed accumulator	91
firfilt.vhd	(simulated)	FIR filter	97
firpack.vhd	(compiled)	package used by filter module	92
fmult.vhd	(simulated)	multiplier: inputs and output variable width	100
fmultacc.vhd	(compiled)	multiplier accumulator for filter	101
multiplier.vhd	(simulated)	multiplies filter tap by gain value in arithmetic unit	103
preset_counter.vhd	(simulated)	toggles preset values in user I/O	104
preset_decoder.vhd	(simulated)	decodes preset values to filter gain values	106
ramload.vhd	(simulated)	RAM loader	107
ramload1.vhd	(simulated)	RAM loader	110
reg_control.vhd	(simulated)	controls data flow in arithmetic unit	113
smart_eq.vhd	(simulated)	top level entity for FPGA	115
smart_eq_pack.vhd	(compiled)	top level package file for smart_eq	117
temp_reg.vhd	(simulated)	shift register that holds gain values	126
temp_reg2.vhd	(simulated)	register that sends coefficient values to filter block	127
user_io.vhd	(simulated)	user interface module	128

```
-- EE 552 - Project
-- Dustin Demontigny
-- March 23, 2003
-- accumulator.vhd
-- used in converting coefficients using gain value
-- Reference design taken from XST User Manual
-- http://toolbox.xilinx.com/docsan/3_1i/data/fise/xst/xst.htm

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity accumulator is
  generic(input_size : integer := 12;
          output_size : integer := 12);
  port(clock, clr, reset : in  std_logic;
       data : in std_logic_vector(input_size-1 downto 0);
       sum : out std_logic_vector(output_size-1 downto 0));
end accumulator;

architecture archi of accumulator is
  signal temp: std_logic_vector(output_size-1 downto 0);
begin
  process (clock, clr)
  begin
    if clr = '1' or reset = '0' then
      temp <= (others => '0');
    elsif (clock'event and clock = '1') then
      temp <= temp + data;
    end if;
  end process;
  sum <= temp;
end archi;
```

```
-- EE 552 Project
-- Dustin Demontigny
-- March 10, 2003
-- arith_unit.vhd
-- arithmetic unit to calculate filter coefficent values

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

use work.smart_eq_pack.all;

entity arith_unit is
  port(clock, reset  : in std_logic;
       gain_register : in std_logic_vector(gain_reg-1 downto 0);
       new_gain, send_coefs : in std_logic;
       send_gain, new_coefs : out std_logic;
       coef_register : out cramarray);
end arith_unit;

architecture mixed of arith_unit is
  --signal tap_number : integer range 0 to taps*bands;
  signal load_gain, shift_gain : std_logic;
  signal clock_accum, clr_accum : std_logic;
  signal load_coef, shift_coefs : std_logic;
  signal band_gain : std_logic_vector(gain_length-1 downto 0);
  signal mult_out : std_logic_vector(coef_length-1 downto 0);
  signal accum_out : std_logic_vector(coef_length-1 downto 0);
  signal coef_value : std_logic_vector(coef_length-1 downto 0);
begin

  process(shift_gain)
    variable tap_number : integer := 0;
  begin
    if load_gain = '1' then
      tap_number := 0;
    elsif shift_gain'event and shift_gain = '1' then
      tap_number := tap_number + 1;
      if tap_number = 4 then
        tap_number := 0;
      end if;
    end if;
    coef_value <= tap_array(tap_number);
  end process;
  controller : reg_control
    port map(clock => clock,
             reset => reset,
             new_gain => new_gain,
             new_coefs => new_coefs,
             send_gain => send_gain,
             send_coefs => send_coefs,
             load_gain => load_gain,
             shift_gain => shift_gain,
             clock_accum => clock_accum,
             clr_accum => clr_accum,
             load_coef => load_coef,
             shift_coefs => shift_coefs);

  gain_shifter : temp_reg
    generic map(input_size => gain_reg,
                output_size => gain_length)
    port map(clock => shift_gain,
             ALOAD => load_gain,
             PI => gain_register,
             PO => band_gain);

  multiply_coeff : multiplier
    port map(A => coef_value,
             B => band_gain,
             RES => mult_out);

  accum_ceof : accumulator
    generic map(input_size => coef_length,
                output_size => coef_length)
    port map(clock => clock_accum,
             clr => clr_accum,
             reset => reset,
             data => mult_out,
             sum => accum_out);

  A_reg : temp_reg2
    port map(clock => shift_coefs,
             clr => reset,
             ALOAD => load_coef,
             PI => accum_out,
             PO => coef_register);

end mixed;
```

```
-- EE 552 - Project
-- Dustin Demontigny
-- February 11, 2003
-- bcd_decoder.vhd
-- converts binary to 7 segment display

library ieee;
use ieee.std_logic_1164.all;

-- digit values are assigned in package file
use work.smart_eq_pack.all;

entity bcd_decoder is
  port(
    binary_input   : in std_logic_vector(bits-1 downto 0);
    segment_output : out std_logic_vector(led_segments-1 downto 0));
end bcd_decoder;

architecture struct of bcd_decoder is
begin
-- select the digit that corresponds to the binary input
  with binary_input select
    segment_output <=
    digit_1 when "0000",
    digit_2 when "0001",
    digit_3 when "0010",
    digit_4 when "0011",
    digit_5 when "0100",
    digit_6 when "0101",
    digit_7 when "0110",
    digit_8 when "0111",
    digit_9 when "1000",
    digit_0 when "1001",
    digit_A when "1010",
    digit_b when "1011",
    digit_C when "1100",
    digit_d when "1101",
    digit_E when "1110",
    digit_F when "1111",
    digit_e when others; -- displays "e" for error
end struct;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use work.codec.all;

entity channel is
  generic
    (
      DAC_WIDTH     :     positive := 20;
      ADC_WIDTH     :     positive := 20
      );
  port
    (
-- interface I/O signals
      clk           : in  std_logic;    -- clock input
      reset         : in  std_logic;    -- synchronous active-high reset
      chan_on       : in  std_logic;
      bit_cntr      : in  std_logic_vector(5 downto 0);
      subcycle_cntr : in  std_logic_vector(1 downto 0);
      chan_sel      : in  std_logic;    -- select L/R channel for read/write
      rd            : in  std_logic;    -- read from the codec ADC
      wr            : in  std_logic;    -- write to the codec DAC
      adc_out       : out std_logic_vector(ADC_WIDTH-1 downto 0);  -- ADC output
      dac_in        : in  std_logic_vector(DAC_WIDTH-1 downto 0);  -- DAC input
      adc_out_rdy   : out std_logic;    -- ADC output is ready to be read
      adc_overrun   : out std_logic;    -- ADC overwritten before being read
      dac_in_rdy    : out std_logic;    -- DAC input is ready to be written
      dac_underrun  : out std_logic;    -- input to DAC arrived late
-- codec chip I/O signals
      sdin          : out std_logic;    -- serial output to codec DAC
      sdout         : in  std_logic     -- serial input from codec ADC
      );
end channel;

architecture channel_arch of channel is
  signal dac_shfreg     : std_logic_vector(DAC_WIDTH-1 downto 0);
  signal dac_empty      : std_logic;    -- DAC shift register is empty
  signal dac_wr         : std_logic;    -- the DAC channel has been written
  signal dac_wr_nxt     : std_logic;    -- the DAC channel has been written
  signal dac_in_rdy_int : std_logic;    -- internal version of dac_in_rdy
  signal adc_shfreg     : std_logic_vector(ADC_WIDTH-1 downto 0);
  signal adc_full       : std_logic;    -- ADC shift register is full

  signal adc_rd          : std_logic;   -- the ADC channel has been read
  signal adc_rd_nxt      : std_logic;   -- the ADC channel has been read
  signal adc_out_rdy_int : std_logic;   -- internal version adc_out_rdy
begin
-- receives data from codec ADC
  rcv_adc :
  process(clk)--, chan_on, subcycle_cntr, bit_cntr, adc_shfreg, sdout)
  begin
    if(clk'event and (clk = YES)) then
      if(reset = '1') then
        adc_shfreg   <= (others => '0');
        adc_full     <= NO;
      elsif((chan_on = YES) and (subcycle_cntr = 2)) then
        if(bit_cntr < ADC_WIDTH-1) then
          adc_full   <= NO;
          adc_shfreg <= adc_shfreg(ADC_WIDTH-2 downto 0) & sdout;
        elsif(bit_cntr = ADC_WIDTH-1) then
          adc_full   <= YES;
          adc_shfreg <= adc_shfreg(ADC_WIDTH-2 downto 0) & sdout;
        end if;
      end if;
    end if;
  end process;
  adc_out            <= adc_shfreg;

-- handle reading of ADC data from codec interface
  adc_rd_nxt      <= YES when (adc_full = YES and chan_sel = YES and rd = YES) or
                (adc_full = YES and adc_rd = YES)
                else NO;
  read_adc :
  process(clk)--, adc_rd_nxt)
  begin
    if(clk'event and clk = '1') then
      if(reset = YES) then
        adc_rd    <= NO;
      else
        adc_rd    <= adc_rd_nxt;
      end if;
    end if;
  end process;
-- ADC data is ready if register is full and hasn't been read yet
  adc_out_rdy_int <= YES when adc_full = YES and adc_rd = NO else NO;
  adc_out_rdy     <= adc_out_rdy_int;

-- detect and signal overwriting of data from the codec ADC channels
  detect_adc_overrun : process(clk)--, bit_cntr, chan_on, adc_out_rdy_int)
  begin
```

```
    if(clk'event and clk = '1') then
      if(reset = YES) then
        adc_overrun <= NO;
      elsif(bit_cntr = 1 and chan_on = YES and adc_out_rdy_int = YES) then
        adc_overrun <= YES;
      end if;
    end if;
  end process;

-- transmits data to codec DAC
  tx_dac : process(clk)--, chan_on, subcycle_cntr, bit_cntr, dac_shfreg)
  begin
    if(clk'event and clk = '1') then
      if(reset = YES) then
        dac_shfreg   <= (others => '0');
        dac_empty    <= YES;
      elsif(chan_sel = YES and wr = YES) then
        dac_shfreg   <= dac_in;
      elsif(chan_on = YES and subcycle_cntr = 2) then
        if(bit_cntr < DAC_WIDTH-1) then
          dac_empty  <= NO;
          dac_shfreg <= dac_shfreg(DAC_WIDTH-2 downto 0) & '0';
        elsif(bit_cntr = DAC_WIDTH-1) then
          dac_empty  <= YES;
          dac_shfreg <= dac_shfreg(DAC_WIDTH-2 downto 0) & '0';
        end if;
      end if;
    end if;
  end process;

-- output the serial data to the SDIN pin of the codec DAC
  sdin <= dac_shfreg(DAC_WIDTH-1) WHEN chan_on = YES ELSE '0';

-- handle writing of DAC data from codec interface
  dac_wr_nxt <= YES WHEN (dac_empty = YES AND chan_sel = YES AND wr = YES) OR
                (dac_empty = YES AND dac_wr = YES) ELSE NO;
  write_dac :
  PROCESS(clk)--, dac_wr_nxt)
  BEGIN
    IF(clk'event AND clk = '1') THEN
      IF(reset = YES) THEN
        dac_wr   <= NO;
      ELSE
        dac_wr   <= dac_wr_nxt;
      END IF;
    END IF;
  END PROCESS;
-- DAC is ready if register is empty and hasn't been written yet
  dac_in_rdy_int <= YES WHEN dac_empty = YES AND dac_wr = NO ELSE NO;
  dac_in_rdy     <= dac_in_rdy_int;

-- detect and signal underflow of data to the codec DAC channels
  detect_dac_underrun :
  PROCESS(clk)--, bit_cntr, chan_on, dac_in_rdy_int)
  BEGIN
    IF(clk'event AND clk = '1') THEN
      IF(reset = YES) THEN
        dac_underrun <= NO;
      ELSIF(bit_cntr = 1 AND chan_on = YES AND dac_in_rdy_int = YES) THEN
        dac_underrun <= YES;
      END IF;
    END IF;
  END PROCESS;
END channel_arch;
```

```
 library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use work.codec.all;

entity clkgen is
  generic
    (
      CHANNEL_DURATION :     positive := 128  -- must be 128
      );
  port
    (
-- interface I/O signals
      clk             : in  std_logic;  -- clock input
      reset           : in  std_logic;  -- synchronous active-high reset
-- codec chip clock signals
      mclk            : out std_logic;  -- master clock output to codec
      sclk            : out std_logic;  -- serial data clock to codec
      lrck            : out std_logic;  -- left/right codec channel select
      bit_cntr        : out std_logic_vector(5 downto 0);
      subcycle_cntr   : out std_logic_vector(1 downto 0)
      );
end clkgen;

architecture clkgen_arch of clkgen is
  signal lrck_int : std_logic;
  signal seq      : std_logic_vector(7 downto 0);
begin
  gen_clock : process(clk)--, seq, lrck_int)
  begin
    if (clk'event and clk = '1') then
      if(reset = YES) then               -- synchronous reset
        seq      <= (others => '0');
        lrck_int <= left;               -- start with left channel of codec
      elsif(seq = CHANNEL_DURATION-1) then
        seq      <= (others => '0');    -- reset sequencer every channel period
        lrck_int <= not(lrck_int);      -- toggle channel sel every period
      else
        seq      <= seq+1;
        lrck_int <= lrck_int;
      END IF;
    END IF;
  END PROCESS;
  lrck           <= lrck_int;           -- output the channel selector to the codec
  mclk           <= clk;                -- codec master clock equals input clock
  sclk           <= seq(1);             -- serial data shift clock = 1/4 master clock
  bit_cntr       <= seq(7 DOWNTO 2);
  subcycle_cntr  <= seq(1 DOWNTO 0);
END clkgen_arch;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_unsigned.all;

package codec is
  constant yes      : std_logic := '1';
  constant no       : std_logic := '0';
  constant ready    : std_logic := '1';
  constant overrun  : std_logic := '1';
  constant underrun : std_logic := '1';
  constant left     : std_logic := '0';
  constant right    : std_logic := '1';

  component clkgen
    generic
      (
        CHANNEL_DURATION :    positive := 128  -- must be 128
        );
    port
      (
-- interface I/O signals
        clk   : in std_logic;           -- clock input
        reset : in std_logic;           -- synchronous active-high reset
-- codec chip clock signals
        mclk          : out std_logic;  -- master clock output to codec
        sclk          : out std_logic;  -- serial data clock to codec
        lrck          : out std_logic;  -- left/right codec channel select
        bit_cntr      : out std_logic_vector(5 downto 0);
        subcycle_cntr : out std_logic_vector(1 downto 0)
        );
  end component;

  component channel
    generic
      (
        DAC_WIDTH : positive := 20;
        ADC_WIDTH : positive := 20
        );
    port
      (
-- interface I/O signals
        clk           : in  std_logic;  -- clock input
        reset         : in  std_logic;  -- synchronous active-high reset
        chan_on       : in  std_logic;
        bit_cntr      : in  std_logic_vector(5 downto 0);
        subcycle_cntr : in  std_logic_vector(1 downto 0);
        chan_sel      : in  std_logic;  -- select L/R channel for read/write
        rd            : in  std_logic;  -- read from the codec ADC
        wr            : in  std_logic;  -- write to the codec DAC
        adc_out       : out std_logic_vector(ADC_WIDTH-1 downto 0);  -- ADC output
        dac_in        : in  std_logic_vector(DAC_WIDTH-1 downto 0);  -- DAC input
        adc_out_rdy   : out std_logic;  -- ADC output is ready to be read
        adc_overrun   : out std_logic;  -- ADC overwritten before being read
        dac_in_rdy    : out std_logic;  -- DAC input is ready to be written
        dac_underrun  : out std_logic;  -- input to DAC arrived late
-- codec chip I/O signals
        sdin          : out std_logic;  -- serial output to codec DAC
        sdout         : in  std_logic   -- serial input from codec ADC
        );
  end component;

  component codec_intfc
    generic
      (
        DAC_WIDTH        :     positive := 20;
        ADC_WIDTH        :     positive := 20;
        CHANNEL_DURATION :     positive := 128  -- must be 128
        );
    port
      (
-- interface I/O signals
        clk              : in  std_logic;  -- clock input
        reset            : in  std_logic;  -- synchronous active-high reset
        lrsel            : in  std_logic;  -- select L/R channel for read/write
        rd               : in  std_logic;  -- read from the codec ADC
        wr               : in  std_logic;  -- write to the codec DAC
        ladc_out         : out std_logic_vector(ADC_WIDTH-1 downto 0);  -- L ADC
        radc_out         : out std_logic_vector(ADC_WIDTH-1 downto 0);  -- R ADC
        ldac_in          : IN  std_logic_vector(DAC_WIDTH-1 DOWNTO 0);  -- left DAC
        rdac_in          : IN  std_logic_vector(DAC_WIDTH-1 DOWNTO 0);  -- right DAC
        ladc_out_rdy     : OUT std_logic;  -- left ADC output ready to read
        radc_out_rdy     : OUT std_logic;  -- right ADC output ready to read
        adc_overrun      : OUT std_logic;  -- ADC overwritten before read
        ldac_in_rdy      : OUT std_logic;  -- left DAC in ready to be written
        rdac_in_rdy      : OUT std_logic;  --right DAC in ready to be written
        dac_underrun     : OUT std_logic;  -- DAC did not receive data in time
        bit_cntr         : out std_logic_vector(5 downto 0);
-- codec chip I/O signals
        mclk             : OUT std_logic;  -- master clock output to codec
        sclk             : OUT std_logic;  -- serial data clock to codec
        lrck            : OUT std_logic;  -- left/right codec channel select
        sdin            : OUT std_logic;  -- serial output to codec DAC
        sdout           : IN  std_logic  -- serial input from codec ADC
        );
  END COMPONENT;
END codec;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use work.codec.all;

entity codec_intfc is
  generic
    (
      DAC_WIDTH        :     positive := 20;
      ADC_WIDTH        :     positive := 20;
      CHANNEL_DURATION :     positive := 128  -- must be 128
      );
  port
    (
-- interface I/O signals
      clk              : in  std_logic;  -- clock input
      reset            : in  std_logic;  -- synchronous active-high reset
      lrsel            : in  std_logic;  -- select L/R channel for read/write
      rd               : in  std_logic;  -- read from the codec ADC
      wr               : in  std_logic;  -- write to the codec DAC
      ladc_out         : out std_logic_vector(ADC_WIDTH-1 downto 0);  -- L ADC
      radc_out         : out std_logic_vector(ADC_WIDTH-1 downto 0);  -- R ADC
      ldac_in          : in  std_logic_vector(DAC_WIDTH-1 downto 0);  -- left DAC
      rdac_in          : in  std_logic_vector(DAC_WIDTH-1 downto 0);  -- right DAC
      ladc_out_rdy     : out std_logic;  -- left ADC output ready to read
      radc_out_rdy     : out std_logic;  -- right ADC output ready to read
      adc_overrun      : out std_logic;  -- ADC overwritten before read
      ldac_in_rdy      : out std_logic;  -- left DAC in ready to be written
      rdac_in_rdy      : out std_logic;  --right DAC in ready to be written
      dac_underrun     : out std_logic;  -- DAC did not receive data in time
      bit_cntr         : out std_logic_vector(5 downto 0);
-- codec chip I/O signals
      mclk             : out std_logic;  -- master clock output to codec
      sclk             : out std_logic;  -- serial data clock to codec
      lrck             : out std_logic;  -- left/right codec channel select
      sdin             : out std_logic;  -- serial output to codec DAC
      sdout            : in  std_logic  -- serial input from codec ADC
      );
end codec_intfc;

architecture codec_intfc_arch of codec_intfc is
  signal mclk_int      : std_logic;     -- internal codec master clock
  signal lrck_int      : std_logic;     -- internal L/R codec channel select
  signal sclk_int      : std_logic;     -- internal codec data shift clock
  signal bit_cntr_int  : std_logic_vector(5 downto 0);
  signal subcycle_cntr : std_logic_vector(1 downto 0);
  signal lsdin         : std_logic;
  signal rsdin         : std_logic;
  signal ladc_overrun  : std_logic;
  signal radc_overrun  : std_logic;
  signal ldac_underrun : std_logic;
  signal rdac_underrun : std_logic;
  signal lchan_sel     : std_logic;
  signal rchan_sel     : std_logic;
  signal lchan_on      : std_logic;
  signal rchan_on      : std_logic;
begin

  u0 : clkgen
    generic map
    (
      CHANNEL_DURATION => CHANNEL_DURATION
      )
    port map
    (
      clk              => clk,
      reset            => reset,
      mclk             => mclk_int,
      sclk             => sclk_int,
      lrck             => lrck_int,
      bit_cntr         => bit_cntr_int,
      subcycle_cntr    => subcycle_cntr
      );
  lrck     <= not(lrck_int);            -- invert for inverter in XStend V1.3
  mclk     <= not(mclk_int);
  sclk     <= not(sclk_int);
  bit_cntr <= bit_cntr_int;

  lchan_sel <= YES when lrsel = left    else NO;
  lchan_on  <= YES when lrck_int = left else NO;
  u_left : channel
    generic map
    (
      DAC_WIDTH     => DAC_WIDTH,
      ADC_WIDTH     => ADC_WIDTH
      )
    port map
    (
      clk           => clk,
      reset         => reset,
    chan_on       => lchan_on,
    bit_cntr      => bit_cntr_int,
    subcycle_cntr => subcycle_cntr,
    chan_sel      => lchan_sel,
    rd            => rd,
    wr            => wr,
    adc_out       => ladc_out,
    dac_in        => ldac_in,
    adc_out_rdy   => ladc_out_rdy,
    adc_overrun   => ladc_overrun,
    dac_in_rdy    => ldac_in_rdy,
    dac_underrun  => ldac_underrun,
    sdin          => lsdin,
    sdout         => sdout
    );

rchan_sel <= YES WHEN lrsel = RIGHT    ELSE NO;
rchan_on  <= YES WHEN lrck_int = RIGHT ELSE NO;

u_right : channel
  GENERIC MAP
  (
    DAC_WIDTH     => DAC_WIDTH,
    ADC_WIDTH     => ADC_WIDTH
    )
  PORT MAP
  (
    clk           => clk,
    reset         => reset,
    chan_on       => rchan_on,
    bit_cntr      => bit_cntr_int,
    subcycle_cntr => subcycle_cntr,
    chan_sel      => rchan_sel,
    rd            => rd,
    wr            => wr,
    adc_out       => radc_out,
    dac_in        => rdac_in,
    adc_out_rdy   => radc_out_rdy,
    adc_overrun   => radc_overrun,
    dac_in_rdy    => rdac_in_rdy,
    dac_underrun  => rdac_underrun,
    sdin          => rsdin,
    sdout         => sdout
    );
  dac_underrun <= YES WHEN ldac_underrun = YES OR rdac_underrun = YES
                ELSE NO;
  adc_overrun  <= YES WHEN ladc_overrun = YES OR radc_overrun = YES
               ELSE NO;

-- generates the serial data output to the SDIN pin of the
-- codec DAC depending on which channel is being loaded
  sdin <= NOT(lsdin) WHEN lrck_int = LEFT ELSE NOT(rsdin);

END codec_intfc_arch;
```

```
-- Name:   counter.vhd
-- Author: David Bull
-- Description:
--   Up counter with definable wrap around value

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity counter is

  generic (
    width   : integer := 4;
    wrapval : integer := 2**4);

  port (
    clk   : in  std_logic;
    en    : in  std_logic;
    rst   : in  std_logic;
    count : out std_logic_vector(width-1 downto 0));

end counter;

architecture countarch of counter is

  constant wrapvalue : std_logic_vector(width-1 downto 0)
    := conv_std_logic_vector(wrapval,width);
  signal cnt : std_logic_vector(width-1 downto 0);

begin  -- countarch

  clockcount: process (clk)
  begin  -- process clockcount
    if clk'event and clk = '1' then
      if rst = '1' then
        cnt <= (others => '0');
      elsif en = '1' then
        if cnt = wrapval then
          cnt <= (others => '0');
        else
          cnt <= cnt + 1;
        end if;
      end if;
    end if;
  end process clockcount;

  count <= cnt;

end countarch;
```

```
-- Name:   counterd.vhd
-- Author: David Bull
-- Description:
--   Down counter with definable wrap around value.

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity counterd is

  generic (
    width   : integer := 4;
    wrapval : integer := 9);

  port (
    clk   : in  std_logic;
    en    : in  std_logic;
    rst   : in  std_logic;
    count : out std_logic_vector(width-1 downto 0));

end counterd;

architecture countdarch of counterd is

  constant wrapvalue : std_logic_vector(width-1 downto 0)
    := conv_std_logic_vector(wrapval,width);
  constant zero : std_logic_vector(width-1 downto 0)
    := conv_std_logic_vector(0,width);
  signal cnt : std_logic_vector(width-1 downto 0);

begin  -- countdarch

  clockcount: process (clk)
  begin  -- process clockcount
    if clk'event and clk = '1' then
      if rst = '1' then
        cnt <= (others => '0');
      elsif en = '1' then
        if cnt = zero then
          cnt <= wrapvalue;
        else
          cnt <= cnt - 1;
        end if;
      end if;
    end if;
  end process clockcount;

  count <= cnt;

end countdarch;
```

```
-- EE 552 - Project
-- Dustin Demontigny
-- February 11, 2003
-- debouncer3.vhd
-- Debounce mechanism for push buttons
-- uses D-flip flop with enable configuration

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

use work.smart_eq_pack.all;

entity debouncer3 is
  port(pb_input : in std_logic_vector(pb_number-1 downto 0);
    clock : in std_logic;
    db_output   : out std_logic_vector(pb_number-1 downto 0));
end debouncer3;

architecture DFF of debouncer3 is
begin
  process (clock)
    variable count : integer := 0;
  begin
    if clock'event and clock = '1' then
      count := count + 1;       -- counts the number of clock cycles
      if count = debounce_delay then-- when desired delay is reached
        db_output <= pb_input;  -- inputs are mapped to outputs
        count := 0;             -- and counting begins from zero
      end if;
    end if;
  end process;
end DFF;
```

```
-- Name:   distram.vhd
-- Author: David Bull
-- Description:
--   Single port distributed RAM

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity rambank is

  generic (
    portwidth : integer := 20;
    addwidth  : integer := 5);

  port (
    clk : in  std_logic;
    we  : in  std_logic;
    a   : in  std_logic_vector(addwidth-1 downto 0);
    di  : in  std_logic_vector(portwidth-1 downto 0);
    do  : out std_logic_vector(portwidth-1 downto 0));

end rambank;

architecture ramarch of rambank is

  type ram_type is array (2**addwidth-1 downto 0) of
    std_logic_vector (portwidth-1 downto 0);
  signal RAM    : ram_type;

begin
  process (clk)
  begin
    if (clk'event and clk = '1') then
      if (we = '1') then
        RAM(conv_integer(a)) <= di;
      end if;
    end if;
  end process;
  do <= RAM(conv_integer(a));
end ramarch;
```

```
-- Name:   eq.vhd
-- Author: David Bull
-- Date:   March 2003
-- Description:
--   This is the entity containing the digital filter, codec
--   interface, RAM loaders and control circuitry

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.firpack.all;
use work.codec.all;

entity eq is

  generic (
    cwidth    : integer := coeffwidth;
    swidth    : integer := samplewidth;
    caddwidth : integer := coeffaddwidth;
    saddwidth : integer := sampleaddwidth;
    filtorder : integer := filterorder);
  port (
    clk   : in  std_logic;
    rst   : in  std_logic;
    sdout : in  std_logic;
    sdin  : out std_logic;
    mclk  : out std_logic;
    sclk  : out std_logic;
    lrck  : out std_logic);

end eq;

architecture eqarch of eq is

  signal ladc_out     : std_logic_vector(swidth-1 downto 0);
  signal radc_out     : std_logic_vector(swidth-1 downto 0);
  signal ladc_out_rdy : std_logic;
  signal radc_out_rdy : std_logic;
  signal ldac_in      : std_logic_vector(swidth-1 downto 0);
  signal rdac_in      : std_logic_vector(swidth-1 downto 0);
  signal ldac_in_rdy  : std_logic;
  signal rdac_in_rdy  : std_logic;
  signal wr           : std_logic;
  signal adc_overrun  : std_logic;
  signal dac_underrun : std_logic;
signal coefframin   : std_logic_vector(cwidth-1 downto 0);
signal sampleramin  : std_logic_vector(swidth-1 downto 0);
signal coeffadden   : std_logic;
signal sampleadden  : std_logic;
signal coeffwe      : std_logic;
signal samplewe     : std_logic;
signal coeffrst     : std_logic;
signal samplerst    : std_logic;

signal sampleramin_int : std_logic_vector(swidth-1 downto 0);
signal coeffadden_int  : std_logic;
signal sampleadden_int : std_logic;
signal coeffwe_int     : std_logic;
signal samplewe_int    : std_logic;
signal coeffrst_int    : std_logic;
signal samplerst_int   : std_logic;

signal sampleramin_ext : std_logic_vector(swidth-1 downto 0);
signal coeffadden_ext  : std_logic;
signal sampleadden_ext : std_logic;
signal coeffwe_ext     : std_logic;
signal samplewe_ext    : std_logic;
signal coeffrst_ext    : std_logic;
signal samplerst_ext   : std_logic;

signal coeffintext  : std_logic;
signal sampleintext : std_logic;

signal clracc : std_logic;
signal accen  : std_logic;

signal poscount        : std_logic_vector(7 downto 0);
signal poscount_coeff  : std_logic_vector(7 downto 0);
signal poscount_sample : std_logic_vector(7 downto 0);
signal posen           : std_logic;
signal posrst          : std_logic;
signal posen_eq        : std_logic;
signal posrst_eq       : std_logic;
signal posen_coeff     : std_logic;
signal posrst_coeff    : std_logic;
signal posen_sample    : std_logic;
signal posrst_sample   : std_logic;
signal posen_ram       : std_logic;
signal posrst_ram      : std_logic;
signal filtout         : std_logic_vector(35 downto 0);
signal lrsample        : std_logic;
signal lrsample_int    : std_logic;
signal lrsample_coeff  : std_logic;
signal lrsample_sample : std_logic;
signal lrsample_ram    : std_logic;
signal lrsel1          : std_logic;

signal samplepos        : std_logic_vector(6 downto 0);
signal samplepos_coeff  : std_logic_vector(6 downto 0);
signal samplepos_sample : std_logic_vector(6 downto 0);

constant forderp1 : std_logic_vector(6 downto 0)
  := conv_std_logic_vector(filtorder+1, 7);
constant forderp2 : std_logic_vector(6 downto 0)
  := conv_std_logic_vector(filtorder+2, 7);
constant forderp3 : std_logic_vector(6 downto 0)
  := conv_std_logic_vector(filtorder+3, 7);
constant zero     : std_logic_vector(6 downto 0)
  := conv_std_logic_vector(0, 7);
constant one      : std_logic_vector(6 downto 0)
  := conv_std_logic_vector(1, 7);

signal cont_req_coeff  : std_logic;
signal cont_rel_coeff  : std_logic;
signal cont_ack_coeff  : std_logic;
signal cont_req_sample : std_logic;
signal cont_rel_sample : std_logic;
signal cont_ack_sample : std_logic;
signal rambusy         : std_logic;
signal coefframidle    : std_logic;
signal sampleramidle   : std_logic;

signal data_in_coeff     : cramarray;
signal data_in_rdy_coeff : std_logic;
signal data_in_ack_coeff : std_logic;

signal data_in_sample     : sramarray;
signal data_in_rdy_sample : std_logic;
signal data_in_ack_sample : std_logic;

signal prev_rst : std_logic;
signal pres_rst : std_logic;
```

```
begin  -- eqarch

  filter: firfilt
    generic map (
      cwidth    => cwidth,
      swidth    => swidth,
      caddwidth => caddwidth,
      saddwidth => saddwidth,
      filtorder => filtorder)
    port map (
      clk         => clk,
      coefframin  => coefframin,
      sampleramin => sampleramin,
      coeffadden  => coeffadden,
      sampleadden => sampleadden,
      coeffwe     => coeffwe,
      samplewe    => samplewe,
      coeffrst    => coeffrst,
      samplerst   => samplerst,
      clracc      => clracc,
      accen       => accen,
      lrsample    => lrsample,
      filtout     => filtout);

  codecintfc: codec_intfc
    generic map (
      DAC_WIDTH        => swidth,
      ADC_WIDTH        => swidth,
      CHANNEL_DURATION => 128)
    port map (
      clk          => clk,
      reset        => rst,
      lrsel        => lrsample,
      rd           => samplewe,
      wr           => wr,
      ladc_out     => ladc_out,
      radc_out     => radc_out,
      ldac_in      => ldac_in,
      rdac_in      => rdac_in,
      ladc_out_rdy => ladc_out_rdy,
      radc_out_rdy => radc_out_rdy,
      adc_overrun  => adc_overrun,
      ldac_in_rdy  => ldac_in_rdy,
      rdac_in_rdy  => rdac_in_rdy,
      dac_underrun => dac_underrun,
    mclk        => mclk,
    sclk        => sclk,
    lrck        => lrck,
    sdin        => sdin,
    sdout       => sdout);

sampleposcounter: counter
  generic map (
    width   => 8,
    wrapval => 2**8)
  port map (
    clk   => clk,
    en    => posen,
    rst   => posrst,
    count => poscount);

sampleposcounter2: counter
  generic map (
    width   => 8,
    wrapval => 2**8)
  port map (
    clk   => clk,
    en    => posen_coeff,
    rst   => posrst_coeff,
    count => poscount_coeff);

sampleposcounter3: counter
  generic map (
    width   => 8,
    wrapval => 2**8)
  port map (
    clk   => clk,
    en    => posen_sample,
    rst   => posrst_sample,
    count => poscount_sample);

rload: ramload
  generic map (
    swidth    => cwidth,
    filtorder => filtorder)
  port map (
    clk         => clk,
    rst         => rst,
    data_in     => data_in_coeff,
    data_in_rdy => data_in_rdy_coeff,
    data_in_ack => data_in_ack_coeff,
    data_out    => coefframin,
    we          => coeffwe_ext,
    adden       => coeffadden_ext,
    addrst      => coeffrst_ext,
    posen       => posen_coeff,
    posrst      => posrst_coeff,
    cont_req    => cont_req_coeff,
    cont_ack    => cont_ack_coeff,
    cont_rel    => cont_rel_coeff,
    samplepos   => samplepos_coeff);

rload1: ramload1
  generic map (
    swidth    => swidth,
    filtorder => filtorder)
  port map (
    clk         => clk,
    rst         => rst,
    data_in     => data_in_sample,
    data_in_rdy => data_in_rdy_sample,
    data_in_ack => data_in_ack_sample,
    data_out    => sampleramin_ext,
    we          => samplewe_ext,
    adden       => sampleadden_ext,
    addrst      => samplerst_ext,
    posen       => posen_sample,
    posrst      => posrst_sample,
    cont_req    => cont_req_sample,
    cont_ack    => cont_ack_sample,
    cont_rel    => cont_rel_sample,
    lrsample    => lrsample_sample,
    samplepos   => samplepos_sample);

samplepos        <= poscount(6 downto 0);
samplepos_coeff  <= poscount_coeff(6 downto 0);
samplepos_sample <= poscount_sample(6 downto 0);

coeffadden  <= coeffadden_int  when coeffintext = '0'  else coeffadden_ext;
coeffwe     <= coeffwe_int     when coeffintext = '0'  else coeffwe_ext;
coeffrst    <= coeffrst_int    when coeffintext = '0'  else coeffrst_ext;

coeffwe_int <= '0';

sampleramin <= sampleramin_int when sampleintext = '0' else sampleramin_ext;
```

```
sampleadden <= sampleadden_int when sampleintext = '0' else sampleadden_ext;
samplewe    <= samplewe_int    when sampleintext = '0' else samplewe_ext;
samplerst   <= samplerst_int   when sampleintext = '0' else samplerst_ext;

lrsample     <= lrsample_int   when lrsel1 = '1' else lrsample_sample;
posen        <= posen_eq;
posrst       <= posrst_eq;

changecontrol: process (clk)
begin  -- process changecontrol
  if clk'event and clk = '1' then
    if rst = '1' then
      lrsel1 <= '1';
      rambusy <= '0';
      cont_ack_coeff <= '0';
      cont_ack_sample <= '0';
      coeffintext <= '0';
      sampleintext <= '0';
    elsif cont_req_coeff = '1' and coefframidle = '1' and rambusy = '0' then
      lrsel1 <= '1';
      rambusy <= '1';
      cont_ack_coeff <= '1';
      coeffintext <= '1';
    elsif cont_req_sample = '1' and sampleramidle = '1' and rambusy = '0' then
      lrsel1 <= '0';
      rambusy <= '1';
      cont_ack_sample <= '1';
      sampleintext <= '1';
    elsif cont_rel_coeff = '1' then
      rambusy <= '0';
      coeffintext <= '0';
    elsif cont_rel_sample = '1' then
      lrsel1 <= '1';
      rambusy <= '0';
      sampleintext <= '0';
    else
      cont_ack_coeff <= '0';
      cont_ack_sample <= '0';
    end if;
  end if;
end process changecontrol;

sampleramin_int <= ladc_out when lrsample = '0' else radc_out;
ldac_in <= filtout(29 downto 10);
rdac_in <= filtout(29 downto 10);
```

```
countcontrol: process (rst,ladc_out_rdy,radc_out_rdy,samplepos)
begin  -- process countcontrol
  if rst = '1' then
    lrsample_int <= '0';
    samplewe_int <= '0';
    samplerst_int <= '1';
    coeffrst_int <= '1';
    posen_eq <= '0';
    posrst_eq <= '1';
    coefframidle <= '1';
    sampleramidle <= '1';
  elsif ladc_out_rdy = '1' then
    lrsample_int <= '0';
    samplewe_int <= '1';
    posrst_eq <= '0';
    posen_eq <= '1';
    coefframidle <= '0';
  elsif radc_out_rdy = '1' then
    lrsample_int <= '1';
    samplewe_int <= '1';
    posrst_eq <= '0';
    posen_eq <= '1';
    sampleramidle <= '0';
  elsif samplepos = forderp3 then
    posrst_eq <= '1';
    posen_eq <= '0';
    coefframidle <= '1';
    sampleramidle <= '1';
  else
    samplerst_int <= '0';
    coeffrst_int <= '0';
    samplewe_int <= '0';
    posrst_eq <= '0';
  end if;
end process countcontrol;

genenable: process (rst,samplepos,lrsample_int)
begin  -- process genenable
  if rst = '1' then
    coeffadden_int <= '0';
    sampleadden_int <= '0';
    wr <= '0';
  elsif samplepos = one then
    coeffadden_int <= '1';
    sampleadden_int <= '1';
  elsif samplepos = forderp1 then
    if lrsample_int = '1' then
      sampleadden_int <= '0';
    end if;
  elsif samplepos = forderp2 then
    coeffadden_int <= '0';
    sampleadden_int <= '0';
    if ldac_in_rdy = '1' then
      wr <= '1';
    elsif rdac_in_rdy = '1' then
      wr <= '1';
    end if;
  else
    wr <= '0';
  end if;
end process genenable;

accen <= coeffadden_int;
clracc <= not(accen);

loadram: process (clk)
begin  -- process loadram
  if clk'event and clk = '1' then
    prev_rst <= pres_rst;
    pres_rst <= rst;
    if pres_rst = '0' and prev_rst = '1' then
      data_in_rdy_coeff <= '1';
      data_in_rdy_sample <= '1';
    else
      data_in_rdy_coeff <= '0';
      data_in_rdy_sample <= '0';
    end if;
  end if;
end process loadram;

data_in_coeff(0) <= (10 => '1', others => '0');
data_in_coeff(1) <= (others => '0');
data_in_coeff(2) <= (others => '0');
data_in_coeff(3) <= (others => '0');
data_in_coeff(4) <= (others => '0');
data_in_coeff(5) <= (others => '0');
data_in_coeff(6) <= (others => '0');
data_in_coeff(7) <= (others => '0');
data_in_coeff(8) <= (others => '0');
```

```
  data_in_coeff(9) <= (others => '0');

  data_in_sample(0) <= (others => '0');
  data_in_sample(1) <= (others => '0');
  data_in_sample(2) <= (others => '0');
  data_in_sample(3) <= (others => '0');
  data_in_sample(4) <= (others => '0');
  data_in_sample(5) <= (others => '0');
  data_in_sample(6) <= (others => '0');
  data_in_sample(7) <= (others => '0');
  data_in_sample(8) <= (others => '0');
  data_in_sample(9) <= (others => '0');
  data_in_sample(10) <= (others => '0');
  data_in_sample(11) <= (others => '0');
  data_in_sample(12) <= (others => '0');
  data_in_sample(13) <= (others => '0');
  data_in_sample(14) <= (others => '0');
  data_in_sample(15) <= (others => '0');
  data_in_sample(16) <= (others => '0');
  data_in_sample(17) <= (others => '0');
  data_in_sample(18) <= (others => '0');
  data_in_sample(19) <= (others => '0');

end eqarch;
```

```
-- Name:   facc.vhd
-- Author: David Bull
-- Description:
--   Signed accumulator

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity facc is

  generic (
    inwidth  : integer := 32;
    outwidth : integer := 36);

  port (
    clk  : in  std_logic;
    en   : in  std_logic;
    rst  : in  std_logic;
    din  : in  std_logic_vector(inwidth-1 downto 0);
    dout : out std_logic_vector(outwidth-1 downto 0));

end facc;

architecture arch_facc of facc is

  signal temp : std_logic_vector(outwidth-1 downto 0);

begin

  clock: process (clk)
  begin  -- process clock
    if clk'event and clk = '1' then
      if rst = '1' then
        temp <= (others => '0');
      elsif en = '1' then
        temp <= temp + din;
      end if;
    end if;
  end process clock;

  dout <= temp;

end arch_facc;
```

```
-- Name:   firpack.vhd
-- Author: David Bull
-- Description:
--   Package containing components used in the filter

library ieee;
use ieee.std_logic_1164.all;

package firpack is

  constant coeffwidth     : integer := 12;  -- width of filter coefficients
  constant samplewidth    : integer := 20;  -- width of audio samples
  constant coeffaddwidth  : integer := 4;
  constant sampleaddwidth : integer := 5;
  constant filterorder    : integer := 9;
  constant adderwidth     : integer := 36;
  constant wrapvalue      : integer := 9;

  type cramarray is array (0 to filterorder)
    of std_logic_vector(coeffwidth-1 downto 0);
  type sramarray is array (0 to 2*filterorder+1)
    of std_logic_vector(samplewidth-1 downto 0);

  component rambank

  generic (
    portwidth : integer := 12;
    addwidth  : integer := 4);

  port (
    clk : in  std_logic;
    we  : in  std_logic;
    a   : in  std_logic_vector(addwidth-1 downto 0);
    di  : in  std_logic_vector(portwidth-1 downto 0);
    do  : out std_logic_vector(portwidth-1 downto 0));

  end component;

  component fadd

    generic (
      a1width : integer := 36;  -- adder width
      a2width : integer := 32);

    port (
    add1 : in  std_logic_vector(a1width-1 downto 0);  -- summand
    add2 : in  std_logic_vector(a2width-1 downto 0);  -- summand
    sum  : out std_logic_vector(a1width-1 downto 0));   -- sum

end component;

component fmult

  generic (
    m1width : integer := 20;          -- multiplicand width
    m2width : integer := 12);         -- multiplicand width

  port (
    mult1 : in  std_logic_vector(m1width-1 downto 0);  -- multiplicand
    mult2 : in  std_logic_vector(m2width-1 downto 0);  -- multiplicand
    prod  : out std_logic_vector(m1width+m2width-1 downto 0));  -- product

end component;

component counter

  generic (
    width   : integer := 4;
    wrapval : integer := 2**4);

  port (
    clk   : in  std_logic;
    en    : in  std_logic;
    rst   : in  std_logic;
    count : out std_logic_vector(width-1 downto 0));

end component;

component counterd

  generic (
    width   : integer := 4;
    wrapval : integer := 9);

  port (
    clk   : in  std_logic;
    en    : in  std_logic;
    rst   : in  std_logic;
    count : out std_logic_vector(width-1 downto 0));
```

```
end component;

component facc

  generic (
    inwidth  : integer := 32;
    outwidth : integer := 36);

  port (
    clk  : in  std_logic;
    en   : in  std_logic;
    rst  : in  std_logic;
    din  : in  std_logic_vector(inwidth-1 downto 0);
    dout : out std_logic_vector(outwidth-1 downto 0));

end component;

component fmultacc

  generic (
    sampwidth  : integer := 20;
    coeffwidth : integer := 12;
    prodwidth  : integer := 32;
    sumwidth   : integer := 36);
  port (
    clk  : in  std_logic;
    en   : in  std_logic;
    rst  : in  std_logic;
    in1  : in  std_logic_vector(19 downto 0);
    in2  : in  std_logic_vector(11 downto 0);
    out1 : out std_logic_vector(35 downto 0));

end component;

component firfilt

  generic (
    cwidth    : integer := coeffwidth;
    swidth    : integer := samplewidth;
    caddwidth : integer := coeffaddwidth;
    saddwidth : integer := sampleaddwidth;
    filtorder : integer := filterorder);

  port (
    clk             : in  std_logic;
    coefframin  : in  std_logic_vector(cwidth-1 downto 0);
    sampleramin : in  std_logic_vector(swidth-1 downto 0);
    coeffadden  : in  std_logic;
    sampleadden : in  std_logic;
    coeffwe     : in  std_logic;
    samplewe    : in  std_logic;
    coeffrst    : in  std_logic;
    samplerst   : in  std_logic;
    clracc          : in  std_logic;
    accen           : in  std_logic;
    lrsample        : in  std_logic;
    filtout         : out std_logic_vector(36-1 downto 0));

end component;

component ramload

  generic (
    swidth    : integer := 20;
    filtorder : integer := filterorder);

  port (
    clk         : in  std_logic;
    rst         : in  std_logic;
    data_in     : in  cramarray;
    data_in_rdy : in  std_logic;
    data_in_ack : out std_logic;
    data_out    : out std_logic_vector(swidth-1 downto 0);
    we          : out std_logic;
    adden       : out std_logic;
    addrst      : out std_logic;
    posen       : out std_logic;
    posrst      : out std_logic;
    cont_req    : out std_logic;
    cont_ack    : in  std_logic;
    cont_rel    : out std_logic;
    samplepos   : in  std_logic_vector(6 downto 0));

end component;

component ramload1

  generic (
    swidth    : integer := 20;
    filtorder : integer := filterorder);
    port (
      clk          : in  std_logic;
      rst          : in  std_logic;
      data_in      : in  sramarray;
      data_in_rdy  : in  std_logic;
      data_in_ack  : out std_logic;
      data_out     : out std_logic_vector(swidth-1 downto 0);
      we           : out std_logic;
      adden        : out std_logic;
      addrst       : out std_logic;
      posen        : out std_logic;
      posrst       : out std_logic;
      cont_req     : out std_logic;
      cont_ack     : in  std_logic;
      cont_rel     : out std_logic;
      lrsample     : out std_logic;
      samplepos    : in  std_logic_vector(6 downto 0));

  end component;

end firpack;
```

```
-- Name:   firfilt.vhd
-- Author: David Bull
-- Description:
--   This entity is the FIR filter.  It instantiates the RAMs,
--   counters, and multiplier-accumulator.

library ieee;
use ieee.std_logic_1164.all;
use work.firpack.all;

entity firfilt is

  generic (
    cwidth    : integer := coeffwidth;
    swidth    : integer := samplewidth;
    caddwidth : integer := coeffaddwidth;
    saddwidth : integer := sampleaddwidth;
    filtorder : integer := filterorder);

  port (
    clk             : in  std_logic;
    coefframin  : in  std_logic_vector(cwidth-1 downto 0);
    sampleramin : in  std_logic_vector(swidth-1 downto 0);
    coeffadden  : in  std_logic;
    sampleadden : in  std_logic;
    coeffwe     : in  std_logic;
    samplewe    : in  std_logic;
    coeffrst    : in  std_logic;
    samplerst   : in  std_logic;
    clracc          : in  std_logic;
    accen           : in  std_logic;
    lrsample        : in  std_logic;
    filtout         : out std_logic_vector(36-1 downto 0));

end firfilt;

architecture farch of firfilt is

  signal sampleadd_low   : std_logic_vector(saddwidth-2 downto 0);
  signal coeffadd        : std_logic_vector(caddwidth-1 downto 0);
  signal sampleadd       : std_logic_vector(saddwidth-1 downto 0);
  signal coefframout     : std_logic_vector(cwidth-1 downto 0);
  signal sampleramout    : std_logic_vector(swidth-1 downto 0);

begin  -- farch
```

```
coeffram: rambank
  generic map (
    portwidth => cwidth,
    addwidth  => caddwidth)
  port map (
    clk => clk,
    we  => coeffwe,
    a   => coeffadd,
    di  => coefframin,
    do  => coefframout);

sampleram: rambank
  generic map (
    portwidth => swidth,
    addwidth  => saddwidth)
  port map (
    clk => clk,
    we  => samplewe,
    a   => sampleadd,
    di  => sampleramin,
    do  => sampleramout);

coeffaddcounter: counter
  generic map (
    width   => caddwidth,
    wrapval => filtorder)
  port map (
    clk   => clk,
    en    => coeffadden,
    rst   => coeffrst,
    count => coeffadd);

sampleaddcounter: counterd
  generic map (
    width   => saddwidth-1,
    wrapval => filtorder)
  port map (
    clk   => clk,
    en    => sampleadden,
    rst   => samplerst,
    count => sampleadd_low);

sampleadd <= lrsample & sampleadd_low;
```

```
  multacc: fmultacc
    generic map (
      sampwidth  => swidth,
      coeffwidth => cwidth,
      prodwidth  => cwidth+swidth,
      sumwidth   => 36)
    port map (
      clk  => clk,
      en   => accen,
      rst  => clracc,
      in1  => sampleramout,
      in2  => coefframout,
      out1 => filtout);

end farch;
```

```vhdl
-- Name:    fmult.vhd
-- Author:  David Bull
-- Description:
--   signed multiplier

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity fmult is

  generic (
    m1width : integer := 20;            -- multiplicand width
    m2width : integer := 12);           -- multiplicand width

  port (
    mult1 : in  std_logic_vector(m1width-1 downto 0);  -- multiplicand
    mult2 : in  std_logic_vector(m2width-1 downto 0);  -- multiplicand
    prod  : out std_logic_vector(m1width+m2width-1 downto 0));  -- product

end fmult;

architecture arch_fmult of fmult is

begin  -- arch_fmult

  prod <= mult1 * mult2;

end arch_fmult;
```

```
-- Name:   fmultacc.vhd
-- Author: David Bull
-- Description:
--   multiplier accumulator for filter

library ieee;
use ieee.std_logic_1164.all;
use work.firpack.all;

entity fmultacc is

  generic (
    sampwidth  : integer := 20;
    coeffwidth : integer := 12;
    prodwidth  : integer := 32;
    sumwidth   : integer := 36);
  port (
    clk  : in  std_logic;
    en   : in  std_logic;
    rst  : in  std_logic;
    in1  : in  std_logic_vector(19 downto 0);
    in2  : in  std_logic_vector(11 downto 0);
    out1 : out std_logic_vector(35 downto 0));

end fmultacc;

architecture arhc of fmultacc is

  signal multout : std_logic_vector(31 downto 0);

begin  -- arhc

  mult: fmult
    generic map (
      m1width => sampwidth,
      m2width => coeffwidth)
    port map (
      mult1 => in1,
      mult2 => in2,
      prod  => multout);

  acc: facc
    generic map (
      inwidth  => prodwidth,
      outwidth => sumwidth)
    port map (
      clk   => clk,
      en    => en,
      rst   => rst,
      din   => multout,
      dout  => out1);

end arhc;
```

```
-- EE 552 - Project
-- Dustin Demontigny
-- March 18, 2003
-- multiplier.vhd
-- used in converting coefficients using gain value
-- Reference design taken from XST User Manual
-- http://toolbox.xilinx.com/docsan/3_1i/data/fise/xst/xst.htm

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

use work.smart_eq_pack.all;

entity multiplier is
  port(A : in std_logic_vector(coef_length-1 downto 0);
       B : in std_logic_vector(gain_length-1 downto 0);
       RES : out std_logic_vector(coef_length-1 downto 0));
end multiplier;

architecture archi of multiplier is
  signal hold : std_logic_vector(coef_length+gain_length-1 downto 0);
begin
  hold <= A * B;
  RES <= hold(coef_length+gain_length-1 downto gain_length);
end archi;
```

```
-- EE 552 Project
-- Dustin Demontigny
-- March 10, 2003
-- preset_counter.vhd
-- toggles between preset values for user interface

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

use work.smart_eq_pack.all;

entity preset_counter is
  port(clock : in std_logic;
       reset : in std_logic;
       incr  : in std_logic;
       decr  : in std_logic;
       preset_value : out std_logic_vector(bits-1 downto 0));
end preset_counter;

architecture mixed of preset_counter is
  signal hold : std_logic_vector(bits-1 downto 0);
begin
  toggle_presets : process(clock)
-- push button flags
    variable incr_pressed : integer := 0;
    variable decr_pressed : integer := 0;
  begin
    if clock'event and clock = '1' then
-- reset to preset 0
      if reset = '0' then
        hold <= "0000";
-- if either pushbutton is pressed, set the flag
      elsif incr = '0' then
        incr_pressed := 1;
      elsif decr = '0' then
        decr_pressed := 1;
-- if pushbuttons are not pressed check for flag
-- and increment or decrement preset value accordingly
      elsif incr_pressed = 1 then
        hold <= hold + "0001";
        incr_pressed := 0;
      elsif decr_pressed = 1 then
        hold <= hold - "0001";
        decr_pressed := 0;
      end if;
-- send the hold value to the output every rising clock edge
      preset_value <= hold;
    end if;
  end process;

end mixed;
```

```
-- EE 552 - Project
-- Dustin Demontigny
-- February 11, 2003
-- preset_decoder.vhd
-- converts preset value to gain/coefficient values

library ieee;
use ieee.std_logic_1164.all;

use work.smart_eq_pack.all;

entity preset_decoder is
  port(preset_value: in std_logic_vector(bits-1 downto 0);
       gain_register  : out std_logic_vector(gain_reg-1 downto 0));
end preset_decoder;

architecture struct of preset_decoder is
begin
  with preset_value select
    gain_register <=
    unity       when "0000", -- base value
    daves       when "0001", -- dave's preset
    dustins     when "0010", -- dustin's preset
    unity       when others;
end struct;
```

```
-- Name:   ramload.vhd
-- Author: David Bull
-- Description:
--   RAM loader

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.firpack.all;

entity ramload is

  generic (
    swidth    : integer := 20;
    filtorder : integer := filterorder);

  port (
    clk         : in  std_logic;
    rst         : in  std_logic;
    data_in     : in  cramarray;
    data_in_rdy : in  std_logic;
    data_in_ack : out std_logic;
    data_out    : out std_logic_vector(swidth-1 downto 0);
    we          : out std_logic;
    adden       : out std_logic;
    addrst      : out std_logic;
    posen       : out std_logic;
    posrst      : out std_logic;
    cont_req    : out std_logic;
    cont_ack    : in  std_logic;
    cont_rel    : out std_logic;
    samplepos   : in  std_logic_vector(6 downto 0));

end ramload;

architecture loadarch of ramload is

  signal data_rdy_int    : std_logic;
  signal data_in_rdy_old : std_logic;
  signal data_in_rdy_new : std_logic;

  constant forder   : std_logic_vector(6 downto 0)
    := conv_std_logic_vector(filtorder, 7);
  constant forderp1 : std_logic_vector(6 downto 0)
```

```
    := conv_std_logic_vector(filtorder+1, 7);

begin  -- loadarch

  genreq: process (clk)
  begin  -- process genreq
    if clk'event and clk = '1' then
      if rst = '1' then
        cont_req <= '0';
        data_in_rdy_old <= '0';
        data_in_rdy_new <= '0';
      else
        data_in_rdy_old <= data_in_rdy_new;
        data_in_rdy_new <= data_in_rdy;
        if data_in_rdy_new = '1' and data_in_rdy_old = '0' then
          cont_req <= '1';
        elsif cont_ack = '1' then
          cont_req <= '0';
        end if;
      end if;
    end if;
  end process genreq;

  timer: process (rst,cont_ack,samplepos)
  begin  -- process timer
    if rst = '1' then
      we <= '0';
      addrst <= '1';
      adden <= '0';
      posrst <= '1';
      posen <= '0';
      data_in_ack <= '0';
      cont_rel <= '0';
    elsif cont_ack = '1' then
      we <= '1';
      addrst <= '0';
      adden <= '1';
      posrst <= '0';
      posen <= '1';
    elsif samplepos = forder then
      addrst <= '1';
      adden <= '0';
      data_in_ack <= '1';
      cont_rel <= '1';
    elsif samplepos = forderp1 then
```

```
      we <= '0';
      posrst <= '1';
      posen <= '0';
      addrst <= '0';
      data_in_ack <= '0';
      cont_rel <= '0';
    else
      data_in_ack <= '0';
      cont_rel <= '0';
      addrst <= '0';
      posrst <= '0';
    end if;
  end process timer;

  output: process (samplepos)
  begin  -- process output
    if samplepos < forderp1 then
      data_out <= data_in(conv_integer(samplepos));
    end if;
  end process output;

end loadarch;
```

```
-- Name:   ramload1.vhd
-- Author: David Bull
-- Description:
--   RAM loader

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.firpack.all;

entity ramload1 is

  generic (
    swidth    : integer := 20;
    filtorder : integer := filterorder);

  port (
    clk         : in  std_logic;
    rst         : in  std_logic;
    data_in     : in  sramarray;
    data_in_rdy : in  std_logic;
    data_in_ack : out std_logic;
    data_out    : out std_logic_vector(swidth-1 downto 0);
    we          : out std_logic;
    adden       : out std_logic;
    addrst      : out std_logic;
    posen       : out std_logic;
    posrst      : out std_logic;
    cont_req    : out std_logic;
    cont_ack    : in  std_logic;
    cont_rel    : out std_logic;
    lrsample    : out std_logic;
    samplepos   : in  std_logic_vector(6 downto 0));

end ramload1;

architecture loadarch of ramload1 is

  signal data_rdy_int    : std_logic;
  signal data_in_rdy_old : std_logic;
  signal data_in_rdy_new : std_logic;

  constant forder     : std_logic_vector(6 downto 0)
    := conv_std_logic_vector(filtorder, 7);
  constant forderp1   : std_logic_vector(6 downto 0)
    := conv_std_logic_vector(filtorder+1, 7);
  constant fordert2   : std_logic_vector(6 downto 0)
    := conv_std_logic_vector(2*filtorder, 7);
  constant fordert2p1 : std_logic_vector(6 downto 0)
    := conv_std_logic_vector(2*filtorder+1, 7);
  constant fordert2p2 : std_logic_vector(6 downto 0)
    := conv_std_logic_vector(2*filtorder+2, 7);

begin  -- loadarch

  genreq: process (clk)
  begin  -- process genreq
    if clk'event and clk = '1' then
      if rst = '1' then
        cont_req <= '0';
        data_in_rdy_old <= '0';
        data_in_rdy_new <= '0';
      else
        data_in_rdy_old <= data_in_rdy_new;
        data_in_rdy_new <= data_in_rdy;
        if data_in_rdy_new = '1' and data_in_rdy_old = '0' then
          cont_req <= '1';
        elsif cont_ack = '1' then
          cont_req <= '0';
        end if;
      end if;
    end if;
  end process genreq;

  timer: process (rst,cont_ack,samplepos)
  begin  -- process timer
    if rst = '1' then
      lrsample <= '0';
      we <= '0';
      addrst <= '1';
      adden <= '0';
      posrst <= '1';
      posen <= '0';
      data_in_ack <= '0';
      cont_rel <= '0';
    elsif cont_ack = '1' then
      lrsample <= '0';
      we <= '1';
      addrst <= '0';
      adden <= '1';
      posrst <= '0';
      posen <= '1';
    elsif samplepos = forderp1 then
      lrsample <= '1';
    elsif samplepos = fordert2p1 then
      addrst <= '1';
      adden <= '0';
      data_in_ack <= '1';
      cont_rel <= '1';
    elsif samplepos = fordert2p2 then
      we <= '0';
      posrst <= '1';
      posen <= '0';
      addrst <= '0';
      data_in_ack <= '0';
      cont_rel <= '0';
    else
      data_in_ack <= '0';
      cont_rel <= '0';
      addrst <= '0';
      posrst <= '0';
    end if;
  end process timer;

  output: process (samplepos)
  begin  -- process output
    if samplepos < fordert2p2 then
      data_out <= data_in(conv_integer(samplepos));
    end if;
  end process output;

end loadarch;
```

```
-- EE 552 Project
-- Dustin Demontigny
-- March 21, 2003
-- reg_control.vhd
-- control the data flow to and from registers

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

use work.smart_eq_pack.all;

entity reg_control is
  port(clock, reset : in std_logic;
       new_gain, send_coefs : in std_logic;
       new_coefs, send_gain : out std_logic;
       load_gain, shift_gain : out std_logic;
       clock_accum, clr_accum : out std_logic;
       load_coef, shift_coefs : out std_logic);
end reg_control;

architecture mixed of reg_control is
  type state_type is (load, accum, shift, next_coef, pause);
  signal state: state_type ;
begin
  contol : process (clock,reset)
    variable bandcount : integer := 0;
    variable tapcount: integer := 0;
  begin
    if (reset ='0') then
      state <= load;
    elsif (clock = '1' and clock'event) then
      case state is
        when load =>
          new_coefs <= '1';
          load_gain <= '1';
          clr_accum <= '0';
          state <= accum;
        when accum =>
          new_coefs <= '0';
          shift_coefs <= '0';
          clock_accum <= '1';
          shift_gain <= '0';
          load_gain <= '0';
          clr_accum <= '0';
          state <= shift;
        when shift =>
          clock_accum <= '0';
          shift_gain <= '1';
          bandcount := bandcount + 1;
          if bandcount = bands then
            bandcount := 0;
            load_coef <= '1';
            state <= next_coef;
          else
            state <= accum;
          end if;
        when next_coef =>
          tapcount := tapcount + 1;
          load_coef <= '0';
          shift_coefs <= '1';
          state <= pause;
        when pause =>
          clr_accum <= '1';
          shift_coefs <= '0';
          if tapcount = taps then
            tapcount := 0; --coefs are ready here
            state <= load;
          else
            state <= accum;
          end if;
        when others =>
          state <= load;
      end case;
    end if;
  end process;
end mixed;
```

```
-- EE 552 Project
-- Dustin Demontigny
-- Dave Bull
-- February 28, 2003
-- smart_eq.vhd
-- Top Level of FPGA for Smart Equalizer project

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

use work.smart_eq_pack.all;

entity smart_eq is
  port (clock,reset: in std_logic; -- clock input to FPGA
        pb_1, pb_2 : in std_logic; -- 2 pushbuttons from XSA and XStend boards
        sdout    : in std_logic; -- serial data from codec
        preset_number : out std_logic_vector(led_segments-1 downto 0);
        sdin     : out std_logic);-- serial data sent to codec
end smart_eq;

architecture struct of smart_eq is
  signal gain_register : std_logic_vector(gain_reg-1 downto 0);
  signal send_gain, new_gain : std_logic;
  signal mclk, sclk, lrck      : std_logic;-- output clocks to codec
  signal send_coefs, new_coefs :  std_logic;
  signal coef_register : cramarray;
  signal notreset : std_logic;
begin

  notreset <= not reset;

  user_int : user_io
    port map(clock => clock,
             reset => reset,
             incr => pb_1,
             decr => pb_2,
             send_gain => send_gain,
             new_gain => new_gain,
             preset_number => preset_number,
             gain_register => gain_register);

  calculate_coefs : arith_unit
    port map(clock => clock,
             reset => reset,
             gain_register => gain_register,
             new_gain => new_gain,
             send_coefs => send_coefs,
             send_gain => send_gain,
             new_coefs => new_coefs,
             coef_register => coef_register);

  filter_block : eq
    port map(clk  => clock,
             rst   => notreset,
             mclk  =>mclk,
             lrck  =>lrck,
             sclk  =>sclk,
             sdout =>sdout,
             sdin  =>sdin,
             data_in_rdy_coeff => new_coefs,
             data_in_ack_coeff => send_coefs,
             data_in_coeff => coef_register);

end struct;
```

```
-- EE 552 Project
-- Dustin Demontigny
-- Dave Bull
-- March 10, 2003
-- smart_eq_pack.vhd
-- Package file for entire Smart EQ system

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

package smart_eq_pack is

-- declare dustin's multipurpose constants
  constant taps : integer := 4;
  constant bands : integer := 10;
  constant bits : integer := 4;
  constant pb_number : integer := 2;
  constant debounce_delay : integer := 7;
  constant coef_length : integer := 12;
  constant gain_reg : integer := 40;
  constant gain_length : integer := 4;

-- declare dave's multipurpose constants
  constant coeffwidth     : integer := coef_length; -- width of coefficients
  constant samplewidth    : integer := 20;  -- width of audio samples
  constant coeffaddwidth  : integer := 4;
  constant sampleaddwidth : integer := 5;
  constant filterorder    : integer := taps;
  constant adderwidth     : integer := 36;
  constant wrapvalue      : integer := taps;
  type cramarray is array (0 to filterorder) of
    std_logic_vector(coeffwidth-1 downto 0);
  type sramarray is array (0 to 2*filterorder+1) of
    std_logic_vector(samplewidth-1 downto 0);

-- declare digit constants for 7 segment display using following pattern
--  ---s6
-- |s5 |s4
--  ---s3
-- |s2 |s1
--  ---s0
-- which is the configuration for the XSA v1.1, v1.2 and Xstend v1.3.2
-- * see page 28 in XSA v1.1, 1.2 User Manual
```

```
  constant led_segments : integer := 7;
  constant digit_1 : std_logic_vector(led_segments-1 downto 0) := "1101101";
  constant digit_2 : std_logic_vector(led_segments-1 downto 0) := "0010100";
  constant digit_3 : std_logic_vector(led_segments-1 downto 0) := "0100100";
  constant digit_4 : std_logic_vector(led_segments-1 downto 0) := "1000101";
  constant digit_5 : std_logic_vector(led_segments-1 downto 0) := "0010010";
  constant digit_6 : std_logic_vector(led_segments-1 downto 0) := "0010000";
  constant digit_7 : std_logic_vector(led_segments-1 downto 0) := "0101101";
  constant digit_8 : std_logic_vector(led_segments-1 downto 0) := "0000000";
  constant digit_9 : std_logic_vector(led_segments-1 downto 0) := "0000101";
  constant digit_0 : std_logic_vector(led_segments-1 downto 0) := "0001000";
  constant digit_A : std_logic_vector(led_segments-1 downto 0) := "0000001";
  constant digit_b : std_logic_vector(led_segments-1 downto 0) := "1010000";
  constant digit_C : std_logic_vector(led_segments-1 downto 0) := "0010010";
  constant digit_d : std_logic_vector(led_segments-1 downto 0) := "1100000";
  constant digit_E : std_logic_vector(led_segments-1 downto 0) := "0010010";
  constant digit_F : std_logic_vector(led_segments-1 downto 0) := "0000011";
  constant digit_error : std_logic_vector(led_segments-1 downto 0) := "0000010";

-- define preset gain values
  constant unity   : std_logic_vector(gain_reg-1 downto 0)
    := "1000100010001000100010001000100010001000";
  constant daves   : std_logic_vector(gain_reg-1 downto 0)
    := "1011100101110101001100110101011110011011";
  constant dustins : std_logic_vector(gain_reg-1 downto 0)
    := "0011010101111001101110111001011101010011";
-- others added as needed

-- define tap array
  subtype tap_type is std_logic_vector(coef_length-1 downto 0);
  type tap_array_type is array (0 to taps*bands-1) of tap_type;
  constant tap_array : tap_array_type := tap_array_type'(
    tap_type'("000000010000"),
    tap_type'("000000100000"),
    tap_type'("000000110000"),
    tap_type'("000001000000"),
    tap_type'("000000010000"),
    tap_type'("000000100000"),
    tap_type'("000000110000"),
    tap_type'("000001000000"),
    tap_type'("000000010000"),
    tap_type'("000000100000"),
    tap_type'("000000110000"),
    tap_type'("000001000000"),
    tap_type'("000000010000"),
    tap_type'("000000100000"),
    tap_type'("000000110000"),
    tap_type'("000001000000"),
    tap_type'("000000010000"),
    tap_type'("000000100000"),
    tap_type'("000000110000"),
    tap_type'("000001000000"),
    tap_type'("000000010000"),
    tap_type'("000000100000"),
    tap_type'("000000110000"),
    tap_type'("000001000000"),
    tap_type'("000000010000"),
    tap_type'("000000100000"),
    tap_type'("000000110000"),
    tap_type'("000001000000"),
    tap_type'("000000010000"),
    tap_type'("000000100000"),
    tap_type'("000000110000"),
    tap_type'("000001000000"),
    tap_type'("000000010000"),
    tap_type'("000000100000"),
    tap_type'("000000110000"),
    tap_type'("000001000000"),
    tap_type'("000000010000"),
    tap_type'("000000100000"),
    tap_type'("000000110000"),
    tap_type'("000001000000"));


-- define test bench constants
  constant reset_delay : time := 200 ns;
  constant clock_period: time := 80 ns;
  constant bounce_delay : time := 18 ns;
  constant press_delay : time := 1000 ns;
  constant bounce_number : integer := 20;


-- declare user_io components
  component debouncer3
    port(pb_input : in std_logic_vector(pb_number-1 downto 0);
         clock  : in std_logic;
         db_output: out std_logic_vector(pb_number-1 downto 0));
  end component;

  component preset_counter
```

```
    port(clock : in std_logic;
         reset : in std_logic;
         incr  : in std_logic;
         decr  : in std_logic;
         preset_value : out std_logic_vector(bits-1 downto 0));
  end component;

  component preset_decoder
    port(preset_value: in std_logic_vector(bits-1 downto 0);
         gain_register  : out std_logic_vector(gain_reg-1 downto 0));
  end component;

  component BCD_decoder
    port(binary_input   : in std_logic_vector(bits-1 downto 0);
         segment_output : out std_logic_vector(led_segments-1 downto 0));
  end component;
-- end user_io components

-- begin arith_unit components
  component temp_reg
    generic(input_size : integer := 40;
            output_size : integer := 4);
    port(clock, aload : in  std_logic;
         PI : in std_logic_vector(input_size-1 downto 0);
         PO : out std_logic_vector(output_size-1 downto 0));
  end component;

  component temp_reg2
    port(clock, clr, aload : in  std_logic;
         PI : in std_logic_vector(coef_length-1 downto 0);
         PO : out cramarray);
  end component;

  component multiplier
    port(A : in std_logic_vector(coef_length-1 downto 0);
         B : in std_logic_vector(gain_length-1 downto 0);
         RES : out std_logic_vector(coef_length-1 downto 0));
  end component;

  component accumulator
    generic(input_size : integer := 12;
            output_size : integer := 12);
    port(clock, clr, reset : in  std_logic;
         data : in std_logic_vector(input_size-1 downto 0);
         sum : out std_logic_vector(output_size-1 downto 0));
  end component;

  component reg_control
    port(clock, reset : in std_logic;
         new_gain, send_coefs : in std_logic;
         new_coefs, send_gain : out std_logic;
         load_gain, shift_gain : out std_logic;
         clock_accum, clr_accum : out std_logic;
         load_coef, shift_coefs : out std_logic);
  end component;
-- end arith_unit components

-- declare filter block components
  component rambank
    generic (
      portwidth : integer := 12;
      addwidth  : integer := 4);
    port (
      clk : in  std_logic;
      we  : in  std_logic;
      a   : in  std_logic_vector(addwidth-1 downto 0);
      di  : in  std_logic_vector(portwidth-1 downto 0);
      do  : out std_logic_vector(portwidth-1 downto 0));
  end component;

  component fadd
    generic (
      a1width : integer := 36;  -- adder width
      a2width : integer := 32);
    port (
      add1 : in  std_logic_vector(a1width-1 downto 0);  -- summand
      add2 : in  std_logic_vector(a2width-1 downto 0);  -- summand
      sum  : out std_logic_vector(a1width-1 downto 0));   -- sum
  end component;

  component fmult
    generic (
      m1width : integer := 20;          -- multiplicand width
      m2width : integer := 12);         -- multiplicand width
    port (
      mult1 : in  std_logic_vector(m1width-1 downto 0);  -- multiplicand
      mult2 : in  std_logic_vector(m2width-1 downto 0);  -- multiplicand
      prod  : out std_logic_vector(m1width+m2width-1 downto 0));  -- product
  end component;
```

```
component counter
  generic (
    width   : integer := 4;
    wrapval : integer := 2**4);
  port (
    clk   : in  std_logic;
    en    : in  std_logic;
    rst   : in  std_logic;
    count : out std_logic_vector(width-1 downto 0));
end component;

component counterd
  generic (
    width   : integer := 4;
    wrapval : integer := 9);
  port (
    clk   : in  std_logic;
    en    : in  std_logic;
    rst   : in  std_logic;
    count : out std_logic_vector(width-1 downto 0));
end component;

component facc
  generic (
    inwidth  : integer := 32;
    outwidth : integer := 36);
  port (
    clk  : in  std_logic;
    en   : in  std_logic;
    rst  : in  std_logic;
    din  : in  std_logic_vector(inwidth-1 downto 0);
    dout : out std_logic_vector(outwidth-1 downto 0));
end component;

component fmultacc
  generic (
    sampwidth  : integer := 20;
    coeffwidth : integer := 12;
    prodwidth  : integer := 32;
    sumwidth   : integer := 36);
  port (
    clk  : in  std_logic;
    en   : in  std_logic;
    rst  : in  std_logic;
    in1  : in  std_logic_vector(19 downto 0);
    in2  : in  std_logic_vector(11 downto 0);
    out1 : out std_logic_vector(35 downto 0));
end component;

component firfilt
  generic (
    cwidth    : integer := coeffwidth;
    swidth    : integer := samplewidth;
    caddwidth : integer := coeffaddwidth;
    saddwidth : integer := sampleaddwidth;
    filtorder : integer := filterorder);
  port (
    clk            : in  std_logic;
    coefframin  : in  std_logic_vector(cwidth-1 downto 0);
    sampleramin : in  std_logic_vector(swidth-1 downto 0);
    coeffadden  : in  std_logic;
    sampleadden : in  std_logic;
    coeffwe     : in  std_logic;
    samplewe    : in  std_logic;
    coeffrst    : in  std_logic;
    samplerst   : in  std_logic;
    clracc      : in  std_logic;
    accen       : in  std_logic;
    lrsample    : in  std_logic;
    filtout     : out std_logic_vector(36-1 downto 0));
end component;

component ramload
  generic (
    swidth    : integer := 20;
    filtorder : integer := filterorder);
  port (
    clk         : in  std_logic;
    rst         : in  std_logic;
    data_in     : in  cramarray;
    data_in_rdy : in  std_logic;
    data_in_ack : out std_logic;
    data_out    : out std_logic_vector(swidth-1 downto 0);
    we          : out std_logic;
    adden       : out std_logic;
    addrst      : out std_logic;
    posen       : out std_logic;
    posrst      : out std_logic;
    cont_req    : out std_logic;
    cont_ack    : in  std_logic;
```

```
      cont_rel    : out std_logic;
--    lrsample    : out std_logic;
      samplepos   : in  std_logic_vector(6 downto 0));
  end component;

  component ramload1
    generic (
      swidth    : integer := 20;
      filtorder : integer := filterorder);
    port (
      clk         : in  std_logic;
      rst         : in  std_logic;
      data_in     : in  sramarray;
      data_in_rdy : in  std_logic;
      data_in_ack : out std_logic;
      data_out    : out std_logic_vector(swidth-1 downto 0);
      we          : out std_logic;
      adden       : out std_logic;
      addrst      : out std_logic;
      posen       : out std_logic;
      posrst      : out std_logic;
      cont_req    : out std_logic;
      cont_ack    : in  std_logic;
      cont_rel    : out std_logic;
      lrsample    : out std_logic;
      samplepos   : in  std_logic_vector(6 downto 0));
  end component;
-- end filter block components

-- begin smart_eq components
  component user_io
    port(clock, reset : in std_logic;
         incr, decr  : in std_logic;
         send_gain : in std_logic;
         new_gain : out std_logic;
         preset_number : out std_logic_vector(led_segments-1 downto 0);
         gain_register : out std_logic_vector(gain_reg-1 downto 0));
  end component;

  component arith_unit
    port(clock, reset  : in std_logic;
         gain_register : in std_logic_vector(gain_reg-1 downto 0);
         new_gain, send_coefs : in std_logic;
         send_gain, new_coefs : out std_logic;
         coef_register : out cramarray);
  end component;

  component eq
    generic (
      cwidth    : integer := coeffwidth;
      swidth    : integer := samplewidth;
      caddwidth : integer := coeffaddwidth;
      saddwidth : integer := sampleaddwidth;
      filtorder : integer := filterorder);
    port (
      clk   : in  std_logic;
      rst   : in  std_logic;
      sdout : in  std_logic;
      sdin  : out std_logic;
      mclk  : out std_logic;
      sclk  : out std_logic;
      lrck  : out std_logic;
      data_in_rdy_coeff : in std_logic;
      data_in_ack_coeff : out std_logic;
      data_in_coeff : in cramarray);
  end component;
-- end smart_eq components

-- declare top level of smart_eq
  component smart_eq
    port (clock,reset   : in std_logic; -- clock input to FPGA
          pb_1, pb_2    : in std_logic; -- 2 pushbuttons from XSA and XStend boards
          sdout         : in std_logic;
          preset_number : out std_logic_vector(led_segments-1 downto 0);
          sdin          : out std_logic);-- serial data sent to codec
  end component;
-- end top level of smart_eq

end smart_eq_pack;
```

```
-- EE 552 - Project
-- Dustin Demontigny
-- March 18, 2003
-- temp_reg.vhd
-- temporarily stores the gain value of each band
-- referenced from XST User Manual

library ieee;
use ieee.std_logic_1164.all;
use work.smart_eq_pack.all;

entity temp_reg is
  generic(input_size : integer := 40;
          output_size : integer := 4);
  port(clock, aload : in  std_logic;
       PI : in std_logic_vector(input_size-1 downto 0);
       PO : out std_logic_vector(output_size-1 downto 0));
end temp_reg;

architecture archi of temp_reg is
  signal temp: std_logic_vector(input_size-1 downto 0);
begin
  process (clock, aload)
  begin
    if (aload = '1') then
      temp <= PI;
    elsif (clock'event and clock='1') then
      temp <= temp(output_size-1 downto 0) &
              temp(input_size-1 downto output_size);
    end if;
  end process;
  PO <= temp(output_size-1 downto 0);
end archi;
```

```
-- EE 552 - Project
-- Dustin Demontigny
-- March 18, 2003
-- temp_reg2.vhd
-- stores the calculated coefficient values in a cramarray

library ieee;
use ieee.std_logic_1164.all;
use work.smart_eq_pack.all;

entity temp_reg2 is
  port(clock, clr, aload : in  std_logic;
       PI : in std_logic_vector(coef_length-1 downto 0);
       PO : out cramarray);
end temp_reg2;

architecture archi of temp_reg2 is
  signal temp: cramarray;
begin
  process (clock, aload, clr)
    variable coef_count : integer := 0;
  begin
    if clr = '0' then
      coef_count := 0;
    elsif (aload = '1') then
      if coef_count < taps+1 then
        temp(coef_count) <= PI;
      end if;
    elsif clock'event and clock = '1' then
      coef_count := coef_count + 1;
    end if;
  end process;
  PO <= temp;
end archi;
```

```vhdl
-- EE 552 Project
-- Dustin Demontigny
-- March 10, 2003
-- user_io.vhd
-- user interface using preset values

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

use work.smart_eq_pack.all;

entity user_io is
  port(clock, reset : in std_logic;
       incr, decr  : in std_logic;
       send_gain : in std_logic;
       new_gain : out std_logic;
       preset_number : out std_logic_vector(led_segments-1 downto 0);
       gain_register : out std_logic_vector(gain_reg-1 downto 0));
end user_io;

architecture mixed of user_io is
  signal db_incr, db_decr : std_logic;
  signal preset_value : std_logic_vector(bits-1 downto 0);
begin

  debounce_pbs : debouncer3
    port map(pb_input(0) => incr,
             pb_input(1) => decr,
             clock => clock,
             db_output(0) => db_incr,
             db_output(1) => db_decr);

  select_preset : preset_counter
    port map(clock => clock,
             reset => reset,
             incr  => db_incr,
             decr  => db_decr,
             preset_value => preset_value);

  decode_gains : preset_decoder
    port map(preset_value => preset_value,
             gain_register  => gain_register);
  decode_preset : bcd_decoder
    port map(binary_input => preset_value,
             segment_output => preset_number);

  gain_change : process(clock)
    variable new_gain_flag : integer := 0;
  begin
    if clock'event and clock = '1' then
      if db_incr = '0' or db_decr = '0' then
        new_gain_flag := 1;
        new_gain <= '0';
      elsif new_gain_flag = 1 and send_gain = '1' then
        new_gain_flag := 0;
        new_gain <= '1';
      end if;
    end if;
  end process;

end mixed;
```

16 Test Benches

Index

filt_testbench.vhd	(simulated)	generates master clock and global reset and simulates the codec chip to test the bandpass filter (firfilt.vhd)	131
test_accum.vhd	(simulated)	tests function of accumulator	137
test_arith_unit.vhd	(simulated)	simulates the arithmetic unit	138
test_db.vhd	(simulated)	used to test the debouncer3.vhd component	139
test_io.vhd	(simulated)	tests and simulates the user interface	141
test_mult.vhd	(simulated)	simulates the multiplier ALU	144
test_preset_counter.vhd	(simulated)	tests the functionality of the preset counter	146
test_reg_control.vhd	(simulated)	simulates the arithmetic unit controller	148
test_smart_eq.vhd	(simulated)	tests the operation of the whole smart equalizer system	150
test_temp_reg.vhd	(simulated)	simulates the operation of temp register	152
test_temp_reg2.vhd	(simulated)	simulates the operation of temp register2	153

```
-- Testbench for bandpass filter (eq.vhd)
-- Emulates the stereo audio codec
-- Supplies master clock (12.5 MHz) and global reset
--
-- Author: David Bull
-- Date: March 11, 2003

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.firpack.all;

entity filt_testbench is
end filt_testbench;

architecture testarch of filt_testbench is

  component eq

    generic (
      cwidth    : integer := coeffwidth;
      swidth    : integer := samplewidth;
      caddwidth : integer := coeffaddwidth;
      saddwidth : integer := sampleaddwidth;
      filtorder : integer := filterorder);
    port (
      clk   : in  std_logic;
      rst   : in  std_logic;
      sdout : in  std_logic;
      sdin  : out std_logic;
      mclk  : out std_logic;
      sclk  : out std_logic;
      lrck  : out std_logic);

  end component;

  constant dwidth : integer := 20;      -- sample width into filter
  constant bwidth : integer := 12;       -- coefficient width
  constant xwidth : integer := 20;      -- codec sample width

  constant clk_period      : time := 80 ns;  -- master clock
  constant clk_half_period : time := 40 ns;
  constant tsclk           : time := 100 ns;  -- propogation delay

  constant filton  : std_logic := '0';
  constant filtoff : std_logic := '1';

-- these constants are the samples that will be fed into the filter starting
-- with sample_in1
  constant sample_in1  : std_logic_vector(xwidth-1 downto 0)
    := conv_std_logic_vector( 816, 20);
  constant sample_in2  : std_logic_vector(xwidth-1 downto 0)
    := conv_std_logic_vector( 760, 20);
  constant sample_in3  : std_logic_vector(xwidth-1 downto 0)
    := conv_std_logic_vector(-352256, 20);
  constant sample_in4  : std_logic_vector(xwidth-1 downto 0)
    := conv_std_logic_vector(-409600, 20);
  constant sample_in5  : std_logic_vector(xwidth-1 downto 0)
    := conv_std_logic_vector( -98304, 20);
  constant sample_in6  : std_logic_vector(xwidth-1 downto 0)
    := conv_std_logic_vector(  90112, 20);
  constant sample_in7  : std_logic_vector(xwidth-1 downto 0)
    := conv_std_logic_vector(  57344, 20);
  constant sample_in8  : std_logic_vector(xwidth-1 downto 0)
    := conv_std_logic_vector( 487424, 20);
  constant sample_in9  : std_logic_vector(xwidth-1 downto 0)
    := conv_std_logic_vector( 335872, 20);
  constant sample_in10 : std_logic_vector(xwidth-1 downto 0)
    := conv_std_logic_vector(      0, 20);
  constant sample_in11 : std_logic_vector(xwidth-1 downto 0)
    := conv_std_logic_vector( 172032, 20);
  constant sample_in12 : std_logic_vector(xwidth-1 downto 0)
    := conv_std_logic_vector( 241664, 20);
  constant sample_in13 : std_logic_vector(xwidth-1 downto 0)
    := conv_std_logic_vector(  24576, 20);
  constant sample_in14 : std_logic_vector(xwidth-1 downto 0)
    := conv_std_logic_vector(-311296, 20);
  constant sample_in15 : std_logic_vector(xwidth-1 downto 0)
    := conv_std_logic_vector( 454656, 20);
  constant sample_in16 : std_logic_vector(xwidth-1 downto 0)
    := conv_std_logic_vector(-270336, 20);
  constant sample_in17 : std_logic_vector(xwidth-1 downto 0)
    := conv_std_logic_vector(  -4096, 20);
  constant sample_in18 : std_logic_vector(xwidth-1 downto 0)
    := conv_std_logic_vector( 126976, 20);
  constant sample_in19 : std_logic_vector(xwidth-1 downto 0)
    := conv_std_logic_vector(-479232, 20);
  constant sample_in20 : std_logic_vector(xwidth-1 downto 0)
    := conv_std_logic_vector( 159744, 20);
  constant sample_in21 : std_logic_vector(xwidth-1 downto 0)
```

```
    := conv_std_logic_vector( -90112, 20);
  constant sample_in22 : std_logic_vector(xwidth-1 downto 0)
    := conv_std_logic_vector(-212992, 20);
  constant sample_in23 : std_logic_vector(xwidth-1 downto 0)
    := conv_std_logic_vector( 516096, 20);
  constant sample_in24 : std_logic_vector(xwidth-1 downto 0)
    := conv_std_logic_vector( 405504, 20);
  constant sample_in25 : std_logic_vector(xwidth-1 downto 0)
    := conv_std_logic_vector( 430080, 20);
  constant sample_in26 : std_logic_vector(xwidth-1 downto 0)
    := conv_std_logic_vector( 495616, 20);
  constant sample_in27 : std_logic_vector(xwidth-1 downto 0)
    := conv_std_logic_vector( 290816, 20);
  constant sample_in28 : std_logic_vector(xwidth-1 downto 0)
    := conv_std_logic_vector( 376832, 20);
  constant sample_in29 : std_logic_vector(xwidth-1 downto 0)
    := conv_std_logic_vector(-114688, 20);
  constant sample_in30 : std_logic_vector(xwidth-1 downto 0)
    := conv_std_logic_vector( 114688, 20);

  signal clk_tb   : std_logic := '0';   -- internal signals
  signal rst_tb   : std_logic := '1';
  signal mclk_tb  : std_logic;
  signal lrck_tb  : std_logic;
  signal sclk_tb  : std_logic;
  signal sdout_tb : std_logic;
  signal sdin_tb  : std_logic;
--   signal onoff_tb : std_logic;

-- input and output samples will be serially shifted to/from codec
-- via the two following vectors
  signal data_in_vector  : std_logic_vector(xwidth-1 downto 0);
  signal data_out_vector : std_logic_vector(xwidth-1 downto 0);

-- array of input samples
  type samplearray is array (0 to 29) of std_logic_vector(xwidth-1 downto 0);
  signal in_array  : samplearray := (0 => sample_in1,
                                     1  => sample_in2,
                                     2  => sample_in3,
                                     3  => sample_in4,
                                     4  => sample_in5,
                                     5  => sample_in6,
                                     6  => sample_in7,
                                     7  => sample_in8,
                                     8  => sample_in9,
                                    9  => sample_in10,
                                    10 => sample_in11,
                                    11 => sample_in12,
                                    12 => sample_in13,
                                    13 => sample_in14,
                                    14 => sample_in15,
                                    15 => sample_in16,
                                    16 => sample_in17,
                                    17 => sample_in18,
                                    18 => sample_in19,
                                    19 => sample_in20,
                                    20 => sample_in21,
                                    21 => sample_in22,
                                    22 => sample_in23,
                                    23 => sample_in24,
                                    24 => sample_in25,
                                    25 => sample_in26,
                                    26 => sample_in27,
                                    27 => sample_in28,
                                    28 => sample_in29,
                                    29 => sample_in30);

-- output samples will be read into an array
  signal out_array : samplearray;

begin  -- testarch

-- instantiate the filter
  filter : eq
    port map (
      clk   => clk_tb,
      rst   => rst_tb,
      sdout => sdout_tb,
      sdin  => sdin_tb,
      mclk  => mclk_tb,
      lrck  => lrck_tb,
      sclk  => sclk_tb);

  rst_tb <= '0' after clk_period;       -- reset deasserted after 80ns
  clk_tb <= not(clk_tb) after clk_half_period;  -- master clock = 12.5 MHz

-- the following process clocks data serially into the filter.
-- samples from in_array are loaded one-by-one into sample_vector
-- then serially shifted into the filter through sdout_tb
-- new samples are loaded from in_array to sample_vector on the
-- rising edge of lrck_tb
-- data is shifted out through sdout_tb on the rising edge of sclk_tb.
  inputaudio: process

  variable sample_vector : std_logic_vector(xwidth-1 downto 0);
  variable bit_count     : integer := 19;
  variable sample_count  : integer := 0;

  begin  -- process inputdata
    wait until sclk_tb'event and sclk_tb = '1';
    if rst_tb = '1' then
      sample_count := 0;
      bit_count := 19;
      sample_vector := in_array(sample_count);
      data_out_vector <= in_array(sample_count);
      sdout_tb <= in_array(sample_count)(bit_count);
      bit_count := bit_count - 1;
    else
      if lrck_tb'event then
        sample_vector := in_array(sample_count);
        data_out_vector <= in_array(sample_count);
        bit_count := 19;
        if lrck_tb = '0' then
          sample_count := sample_count + 1;
        end if;
      end if;
      if bit_count < 0 then
        sdout_tb <= '0' after tsclk;
      else
        sdout_tb <= sample_vector(bit_count) after tsclk;
        bit_count := bit_count - 1;
      end if;
    end if;
  end process inputaudio;

-- the following process clocks data serially out of the filter.
-- the filtered samples are input serially through sdin_tb and read
-- into sample_vector
-- sample_vector is read into out_array on the falling edge of lrck_tb
  outputaudio: process (sclk_tb,lrck_tb)

  variable sample_vector : std_logic_vector(xwidth-1 downto 0);
  variable bit_count     : integer := 19;
  variable sample_count  : integer := 0;
  begin  -- process outputaudio
    if rst_tb = '1' then
      sample_count := 0;
      bit_count := 19;
    elsif lrck_tb'event then
      sample_vector := data_in_vector;
      out_array(sample_count) <= data_in_vector;
      bit_count := 19;
      if lrck_tb = '1' then
        sample_count := sample_count + 1;
      end if;
    elsif sclk_tb'event and sclk_tb = '0' and bit_count >= 0 then
      data_in_vector(bit_count) <= not(sdin_tb);
      bit_count := bit_count - 1;
    end if;
  end process outputaudio;
end testarch;
```

```
-- EE 552 Project
-- Dustin Demontigny
-- March 22, 2003
-- test_accum.vhd
-- Behavioral
-- test bench for accumulator

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

use work.user_io_pack.all;

entity test_accum is
end test_accum;

architecture behav of test_accum is
  signal clock, clr, reset : std_logic := '0';
  signal data : std_logic_vector(coef_length-1 downto 0);
  signal sum : std_logic_vector(coef_length-1 downto 0);
begin

  dut : accumulator
    port map(clock => clock,
             clr => clr,
             reset => reset,
             data => data,
             sum => sum);

  clock <= not clock after clock_period/2;

  stimulus : process
  begin
    clr <= '1';
    data <= "000000000010";
    wait for 200 ns;
    clr <= '0';
    wait;
  end process;

end behav;
```

```
-- EE 552 Project
-- Dustin Demontigny
-- March 8, 2003
-- test_arith_unit.vhd
-- Behavioral
-- test bench for arithmetic unit

library ieee;
use ieee.std_logic_1164.all;

-- timing constants are located in package file
use work.smart_eq_pack.all;

entity test_arith_unit is
end test_arith_unit;

architecture behav of test_arith_unit is
  signal clock, reset  : std_logic := '0';
  signal gain_register : std_logic_vector(gain_reg-1 downto 0) := (others => '0');
  signal new_gain, send_coefs : std_logic;
  signal send_gain, new_coefs : std_logic;
  signal coef_register        : cramarray;
begin

  dut : arith_unit
    port map(clock => clock,
             reset => reset,
             gain_register => gain_register,
             new_gain => new_gain,
             send_coefs => send_coefs,
             send_gain => send_gain,
             new_coefs => new_coefs,
             coef_register => coef_register);

  gain_register <= dustins;
  clock <= not clock after clock_period/2;

  stimulus: process
  begin
    reset <= '0';
    wait for 200 ns;
    reset <= '1';
    wait;
  end process;
end behav;
```

```
-- EE 552 Project
-- Dustin Demontigny
-- March 8, 2003
-- test_db.vhd
-- Behavioral
-- test bench for debouncer3

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

use work.user_io_pack.all;

entity test_db is
end test_db;

architecture behav of test_db is
-- declare inputs
  signal pb_input: std_logic_vector(pb_number-1 downto 0);
  signal clock : std_logic := '0';
  signal db_output : std_logic_vector(pb_number-1 downto 0);
begin
  dut : debouncer3
    port map(pb_input => pb_input,
             clock => clock,
             db_output => db_output);

  clock <= not clock after clock_period/2;

  stimulus : process
  begin
    pb_input <= "11";
    wait for 400 ns;
    pb_input <= "10";
    wait for 400 ns;
    pb_input <= "11";
    wait for 400 ns;
    pb_input <= "10";
    wait for 400 ns;
    pb_input <= "11";
    wait for 400 ns;
    pb_input <= "01";
    wait for 400 ns;
    pb_input <= "11";
    wait for 400 ns;
    pb_input <= "01";
    wait for 400 ns;
    pb_input <= "11";
    wait;
  end process;

end behav;
```

```
-- EE 552 Project
-- Dustin Demontigny
-- March 8, 2003
-- test_io.vhd
-- Behavioral
-- test bench for user I/O

library ieee;
use ieee.std_logic_1164.all;

-- timing constants are located in package file
use work.smart_eq_pack.all;

entity test_io is
end test_io;

architecture behav of test_io is
-- declare inputs
  signal reset, incr, decr : std_logic;
  signal clock : std_logic := '0';
  signal send_gain : std_logic;
-- declare outputs
  signal new_gain : std_logic;
  signal preset_number : std_logic_vector(led_segments-1 downto 0);
  signal gain_register : std_logic_vector(gain_reg-1 downto 0);
begin
  dut : user_io
    port map(reset => reset,
             incr => incr,
             decr => decr,
             clock => clock,
             send_gain => send_gain,
             new_gain => new_gain,
             preset_number => preset_number,
             gain_register => gain_register);

-- generate clock signal
  clock <= not clock after clock_period/2;

-- generate 2 push button stimulus
  push_buttons : process
  begin

-- initialize inputs
    reset <= '0';
    incr <= '1';
    decr <= '1';
    send_gain <= '0';
-- reset system
    wait for reset_delay;
    reset <= '1';
    wait for reset_delay;

-- generate two incr button presses
    for i in 0 to 1 loop
      for i in 0 to bounce_number loop
        incr <= not incr;
        wait for bounce_delay;
      end loop;
      incr <= '0';
      wait for press_delay;
      for i in 0 to bounce_number loop
        incr <= not incr;
        wait for bounce_delay;
      end loop;
      incr <= '1';
      wait for press_delay;
    end loop;

    wait for 200 ns;
    send_gain <= '1';
    wait for 400 ns;
    send_gain <= '0';
    wait for 200 ns;

-- generate two decr button presses
    for i in 0 to 1 loop
      for i in 0 to bounce_number loop
        decr <= not decr;
        wait for bounce_delay;
      end loop;
      decr <= '0';
      wait for press_delay;
      for i in 0 to bounce_number loop
        decr <= not decr;
        wait for bounce_delay;
      end loop;
      decr <= '1';
      wait for press_delay;
    end loop;
    wait for 200 ns;
    send_gain <= '1';
    wait for 400 ns;
    send_gain <= '0';
    wait;

  end process;

end behav;
```

```
-- EE 552 Project
-- Dustin Demontigny
-- March 22, 2003
-- test_mult.vhd
-- Behavioral
-- test bench for multiplier

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

use work.user_io_pack.all;

entity test_mult is
end test_mult;

architecture behav of test_mult is
  signal A : std_logic_vector(coef_length-1 downto 0);
  signal B : std_logic_vector(gain_length-1 downto 0);
  signal RES : std_logic_vector(coef_length-1 downto 0);
begin

  dut : multiplier
    port map(A => A,
             B => B,
             RES => RES);

  stimulus : process
  begin
    A <= "000011110000";
    B <= "0001";
    wait for 200 ns;
    B <= "0010";
    wait for 200 ns;
    B <= "0011";
    wait for 200 ns;
    B <= "0100";
    wait for 20 ns;
    B <= "0101";
    wait for 200 ns;
    B <= "0110";
    wait for 20 ns;
    B <= "0111";
    wait;
  end process;

end behav;
```

```
-- EE 552 Project
-- Dustin Demontigny
-- March 8, 2003
-- test_preset_counter.vhd
-- Behavioral
-- test bench for preset counter

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

use work.user_io_pack.all;

entity test_preset_counter is
end test_preset_counter;

architecture behav of test_preset_counter is
  signal clock : std_logic := '0';
  signal reset : std_logic;
  signal incr  : std_logic;
  signal decr  : std_logic;
  signal preset_value : std_logic_vector(bits-1 downto 0);
begin
  dut : preset_counter
    port map(clock => clock,
             reset => reset,
             incr  => incr,
             decr  => decr,
             preset_value => preset_value);

  clock <= not clock after clock_period/2;

  stimulus : process
  begin
    reset <= '0';
    incr <= '1';
    decr <= '1';
    wait for 200 ns;
    reset <= '1';
    wait for 200 ns;
    incr <= '0';
    wait for 300 ns;
    incr <= '1';
    wait for 200 ns;
    incr <= '0';
    wait for 700 ns;
    incr <= '1';
    wait for 200 ns;
    decr <= '0';
    wait for 900 ns;
    decr <= '1';
    wait for 200 ns;
    decr <= '0';
    wait for 600 ns;
    decr <= '1';
    wait;
  end process;

end behav;
```

```
-- EE 552 Project
-- Dustin Demontigny
-- March 8, 2003
-- test_reg_control.vhd
-- Behavioral
-- test bench for register controller

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

use work.smart_eq_pack.all;

entity test_reg_control is
end test_reg_control;

architecture behav of test_reg_control is
  signal clock, reset : std_logic := '0';
  signal new_gain, send_coefs : std_logic;
  signal load_gain, shift_gain : std_logic;
  signal clock_accum, clr_accum : std_logic;
  signal load_coef, shift_coefs : std_logic;
  signal coef_ready, send_gain : std_logic;
begin
  dut : reg_control
    port map(clock => clock,
             reset => reset,
             new_gain => new_gain,
             send_coefs => send_gain,
             load_gain => load_gain,
             shift_gain => shift_gain,
             clock_accum => clock_accum,
             clr_accum => clr_accum,
             load_coef => load_coef,
             shift_coefs => shift_coefs,
             send_gain => send_gain);

  clock <= not clock after clock_period/2;

  stimulus : process
  begin
    reset <= '0';
    wait for 200 ns;
    reset <= '1';
    wait;
  end process;

end behav;
```

```
-- EE 552 Project
-- Dustin Demontigny
-- March 8, 2003
-- test_smart_eq.vhd
-- Behavioral
-- test bench for top level smar_eq

library ieee;
use ieee.std_logic_1164.all;

-- timing constants are located in package file
use work.smart_eq_pack.all;

entity test_smart_eq is
end test_smart_eq;

architecture behav of test_smart_eq is
  signal clock,reset : std_logic := '0'; -- clock input to FPGA
  signal pb_1, pb_2  : std_logic; -- 2 pushbuttons from XSA and XStend boards
  signal sdout  : std_logic := '0';
  signal preset_number: std_logic_vector(led_segments-1 downto 0);
  signal sdin   : std_logic;-- serial data sent to codec
begin

  dut : smart_eq
    port map(clock => clock,
             reset => reset,
             pb_1 => pb_1,
             pb_2 => pb_2,
             sdout => sdout,
             preset_number => preset_number,
             sdin  => sdin);

-- generate clock signal
  clock <= not clock after clock_period/2;

  sdout <= not sdout after bounce_delay*31;

  process
  begin
-- initialize inputs
    reset <= '0';
    pb_1 <= '1';
    pb_2 <= '1';
-- reset system
```

```
    wait for reset_delay;
    reset <= '1';
    wait for reset_delay;

-- generate two incr button presses
    for i in 0 to 1 loop
      for i in 0 to bounce_number loop
        pb_1 <= not pb_1;
        wait for bounce_delay;
      end loop;
      pb_1 <= '0';
      wait for press_delay;
      for i in 0 to bounce_number loop
        pb_1 <= not pb_1;
        wait for bounce_delay;
      end loop;
      pb_1 <= '1';
      wait for press_delay;
    end loop;

-- generate two decr button presses
    for i in 0 to 1 loop
      for i in 0 to bounce_number loop
        pb_2 <= not pb_2;
        wait for bounce_delay;
      end loop;
      pb_2 <= '0';
      wait for press_delay;
      for i in 0 to bounce_number loop
        pb_2 <= not pb_2;
        wait for bounce_delay;
      end loop;
      pb_2 <= '1';
      wait for press_delay;
    end loop;

    wait;
  end process;

end behav;
```

```
-- EE 552 Project
-- Dustin Demontigny
-- March 8, 2003
-- test_temp_reg.vhd
-- Behavioral
-- test bench for temporary register

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

use work.user_io_pack.all;

entity test_temp_reg is
end test_temp_reg;

architecture behav of test_temp_reg is
  signal clock, ALOAD : std_logic := '0';
  signal PI : std_logic_vector(gain_reg-1 downto 0) := (others => '0');
  signal PO : std_logic_vector(gain_length-1 downto 0) := (others => '0');
begin

  dut : temp_reg
    port map(clock => clock,
             ALOAD  => ALOAD,
             PI  => PI,
             PO => PO);

  clock <= not clock after clock_period/2;

  stimulus : process
  begin
    PI <= dustins;
    ALOAD <= '0';
    wait for 200 ns;
    ALOAD <= '1';
    wait for 210 ns;
    ALOAD <= '0';

    wait;
  end process;

end behav;
```

```
-- EE 552 Project
-- Dustin Demontigny
-- March 8, 2003
-- test_temp2_reg.vhd
-- Behavioral
-- test bench for temporary register2

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

use work.smart_eq_pack.all;

entity test_temp_reg2 is
end test_temp_reg2;

architecture behav of test_temp_reg2 is
  signal clock, clr, ALOAD : std_logic := '0';
  signal PI : std_logic_vector(coef_length-1 downto 0) := (others => '0');
  signal PO : cramarray;
begin

  dut : temp_reg2
    port map(clock => clock,
             clr => clr,
             ALOAD  => ALOAD,
             PI  => PI,
             PO => PO);

  clock <= not clock after clock_period/2;

  stimulus : process
  begin
    PI <= "000001100000";
    clr <= '0';
    wait for 200 ns;
    clr <= '1';
    wait for 200 ns;
    aload <= '1';
    wait for 200 ns;
    aload <= '0';
    wait for 400 ns;
    aload <= '1';
    wait for 200 ns;
    aload <= '0';
    wait for 800 ns;
    clr <= '0';
    wait;
  end process;

end behav;
```

17 Schematics

The following page includes a schematic detailing the ciruitry associated with the on-board stereo audio codec. The codec is the only device used that is peripheral to the FPGA.

18 Inputting/Outputting Stereo Signals Through the Codec

The following document is taken from the XStend Board V1.3.2 User Manual[5]. It includes an example design using the stereo audio codec on the XStend board. We will be incorporating the codec_intfc module in our system, making modifications to the sample code as necessary.