Massachusetts Institute of Technology

OMNIBOX

A Multi-Featured Audio Effects Module

Dylan Sherry and Devon Rosner

12/13/2011

6.111 Final Project

Abstract

Omnibox is an audio effects suite which supports a multitude of valuable musical effects as found on commercial pedal boxes, including some more complex and uncommon features. Omnibox supports a variety of effects including: equalization, long delay, tremolo, reverb, chorus, ring modulation, pedal-controlled wah, auto wah, panning, distortion, and a four-channel recorder/looper. Due to the power and versatility of the 6.111 labkit, Omnibox maintains a high quality 48 kHz audio stream. When these factors are considered in conjunction with a streamlined, intuitive physical layout and menu, it is clear that Omnibox is the system to choose for audio effects.

Table Of Contents

Section Authorship

The Abstract and Design Overview were the products of a team effort. Under Utility Modules and Effects Modules, each modules' section was written by the author or authors of that module, and revised by the group. Future Work and subsequent sections were written and revised by Dylan.

Design Overview

Omnibox is a multi-feature audio effects module enabling realtime digital effects on streaming audio. Using the module is simple: users connect an audio signal to the labkit's mic port and receive the modified signal via the labkit's headphone port. Users can interact with and alter the effects via a simple menu implemented with the labkit's alphanumeric display panel, buttons, and switches. When packaged as individual components, each of the effects supported by Omnibox retails for hundreds of dollars. Conversely, Omnibox is intended to provide high quality audio effects at a low price.

Omnibox is partitioned into four packages: the AC'97 codec (implemented in lab 5), the effects control package, the effects stream, and the ZBT 4 channel recorder/looper. The control module provides an interface between the labkit's buttons, switches, alphanumeric display, and the parameters that control each effect. The effects stream consists of a string of effects modules, where data streams through the modules in a predefined order. This live audio stream allows the Omnibox to maintain a high quality sampling rate around 48kHz. The ZBT SRAM provides enough memory space for a 4 channel recorder/looper that will output recorded audio simultaneously with the audio output from the effects stream. Additionally, for each audio sample received from the AC'97 codec, Omnibox has several hundred clock cycles to compute an output sample in time for the next input/output exchange, which grants our design incredible flexibility.

Regarding the layout of our modules, we recognize that the typical music effects layout has been fairly standardized, so the effects will be laid out in a minimally changeable fashion. From audio in to audio out, our effects will be as follows: distortion/compression, wah/auto wah, ring modulator, either reverb or chorus, tremolo, long delay, and equalizer. Both reverb and chorus use small uniform time blocks as delay, so only one will be on at a time.

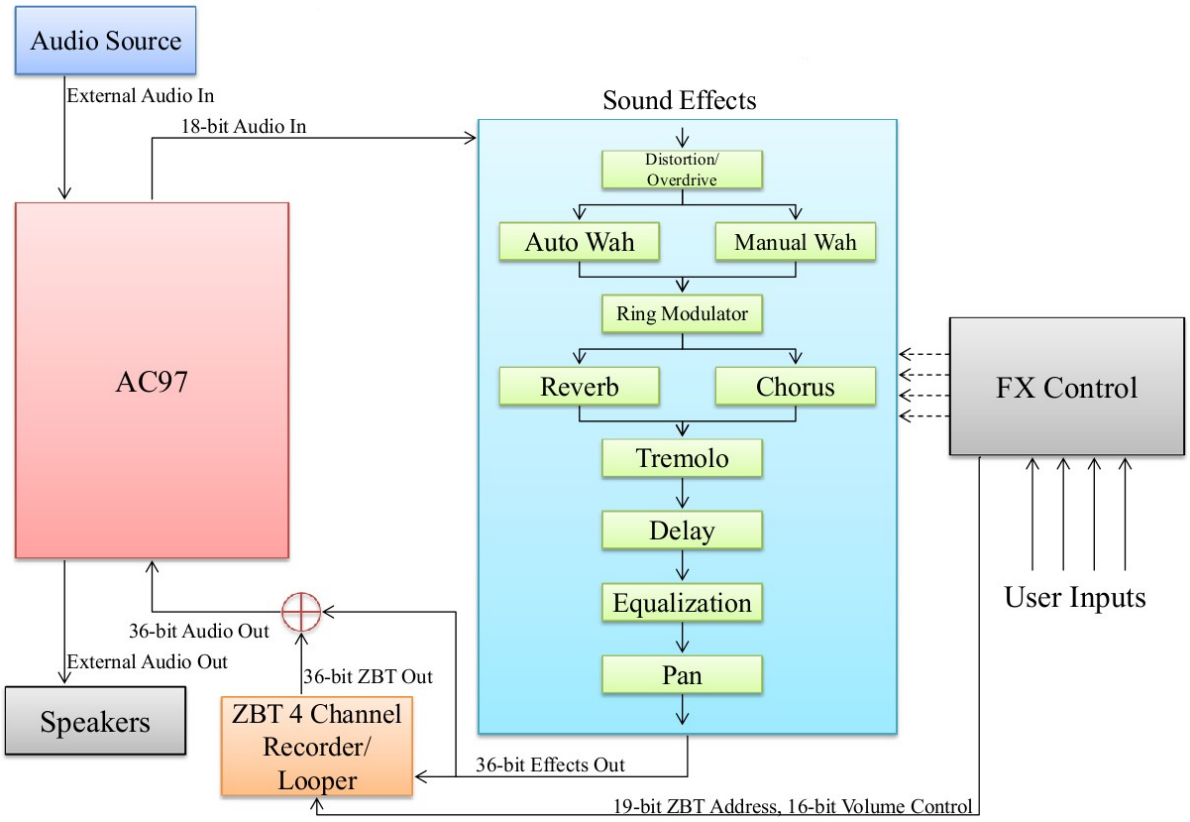


Figure 1: the high-level Omnibox block diagram

Modules

Utility Modules

The following modules are the packages which granted an added dimension of convenience to our interfaces, and allowed us to connect with the labkit's outputs and memory, and our hardware pedal. Unless otherwise noted, all modules accept as inputs clock, reset and ready signals.

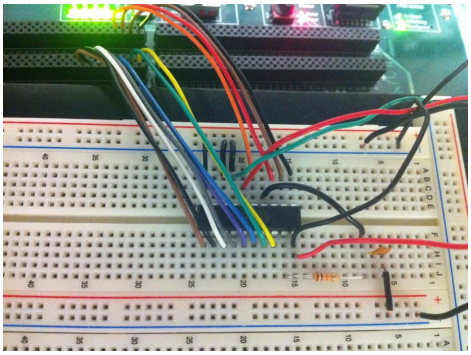
The "mybram" Module (6.111 Provided Code)
This module remained unchanged from the lab 5 code. It is used in the Long Delay, Reverb, and Chorus effects.

Display String (6.111 Provided Code, modified by Devon)
The display string module takes a string where every character is represented in 8-bit ASCII and maps the string to the 16 character led display on the labkit. It is used by the FX Controller to display the menu to the user. The only modification was to give it the ability to take non-ASCII formatted numbers and output them to the led display. Since the data input is still 8 bits per character, we decided to make the numbers display in hexadecimal. This means every displayed number greater than 4 bits must be split up into multiple characters.

For example, the 7 digit register ‘Example’ would be sent to the display string module as {5’d0,Example[6:4],4’d0, Example[3:0]}, since it must be represented by two characters in hexadecimal and each character must be eight bits in length.

Modules “lab5audio” and “lab5” (6.111 Provided Code, modified by Devon and Dylan)
The lab5audio and lab5 modules provided us with an easy communication link to the AC97. These modules, as the names imply, were taken from lab 5, but they had to be modified to take 18-bit quality audio instead of 8-bit audio, and to output stereo audio instead of mono. Changing to 18-bit audio only involved changing a few [7:0]’s to [17:0]. To get stereo output, a line of code had to be changed in lab5audio to take real input data into the left speaker instead of setting the left speaker equal to the right speaker, and a [7:0] had to be changed to [35:0] to output the 18-bit left and right channels. Some code also had to be added to support writing to the ZBT SRAM, mainly the clock deskewing/timing modules.

ADC Interface (Devon)
The ADC module is used to communicate between the labkit and the ADC0801. It follows the ADC0801 timing specifications in a simple three-state state machine. It first tells the ADC to store the value of the potentiometer onto the chip, then it tells the labkit to read this




Figures 2a and 2b: the ADC and pedal used in the Wah effect

value from the ADC, and finally it waits for the ADC’s reset signal. This module is only used with the manual wah position input.

Button Scroll (Devon)
Button scroll is a module used by the menu that allows the user to rapidly change a parameter’s value. This module is very useful to the user because it means he/she does not have to press up or down over 1000 times to completely scroll through the value range of one of the many 10-bit effect parameters. The button scroll module runs completely on a four-state state machine. When a button is not pressed, it stays in the BUT_OFF state waiting for the input button’s signal. Once a button is depressed, the state instantly transitions to BUT_ON where if the button is released before a third of a second, the module outputs a single clock pulse (which translates to a single +/- 1 to the effect parameter) and returns to BUT_OFF. However, if the button is held for more than a third of a second, it moves to BUT_ON_SLOW and sends another single clock pulse every third of a second for two

seconds or until the button is released. If the button is held for longer than two seconds, the state transitions to BUT_ON_FAST and the module outputs a single clock pulse every 1/100th of a second until the button is released and then returns to BUT_OFF.

The FirlowXXX and FirlowXXXcoeffs Modules (Devon)
For all 13 low-pass filters used in this project, the 251 coefficients for a given low-pass filter of frequency XXX can be found in its corresponding FirlowXXXcoeffs module. The coefficients were generated using the `fir1` method in MATLAB, and automatically parsed and molded into Verilog modules. Each frequency also has its own firlowXXX module that handles the convolution of the incoming audio signal with the FIR filter. Convolution is handled by continuously storing the most recent 251 input samples into an array, multiplying each of the entries by one of the coefficients from the firlowXXXcoeffs module, and adding all of them together. The effects that depend on firlowXXX and firlowXXXcoeffs are wah and the 5 band equalizer.

Signal Generator (Dylan)
Description: The SignalGen module is generates square, sin, triangle and sawtooth waves. The resulting waves have a theoretical range in frequency from around 10-5 Hz to 12,000 Hz. The module serves as a critical component of Reverb, Chorus and Ring Modulation, and proved to be a useful testing and debugging tool for Chorus, Distortion, Panning, Tremolo and Ring Modulation.

The SignalGen takes as inputs a bit indicating the domain of operation, a two-bit wave selector indicating which waveform should be produced, and an 11-bit “knob” value controlling the waveform’s period. The output comes in the form of an 18-bit signed number. For the square wave, the output alternates between an 18-bit signed 1 and 0; for the sine, triangle and sawtooth waves, the output oscillates between the minimum and maximum values of the specified domain.

SignalGen includes two modes of operation, each corresponding to a different output domain. When the domain selector bit is high, SignalGen outputs values between (2^18 - 1) and 0. This domain is used by Reverb to calculate a decay value profile, if enabled. When the domain selector bit is low, SignalGen outputs values ranging from (2^18 - 1) and (-2^18).

The SignalGen module relies on four component modules:

- square → `mybram_square`
- sin → `mybram_sin`
- triangle → `mybram_triangle`
- saw → `mybram_saw`

Each of these modules consists of a wrapper FSM which controls reading from a read-only BRAM module, named above. Originally generated in Python, the BRAM modules each consist of 256 samples at 8 bits each, for a total size of 1024 bits per read-only BRAM.

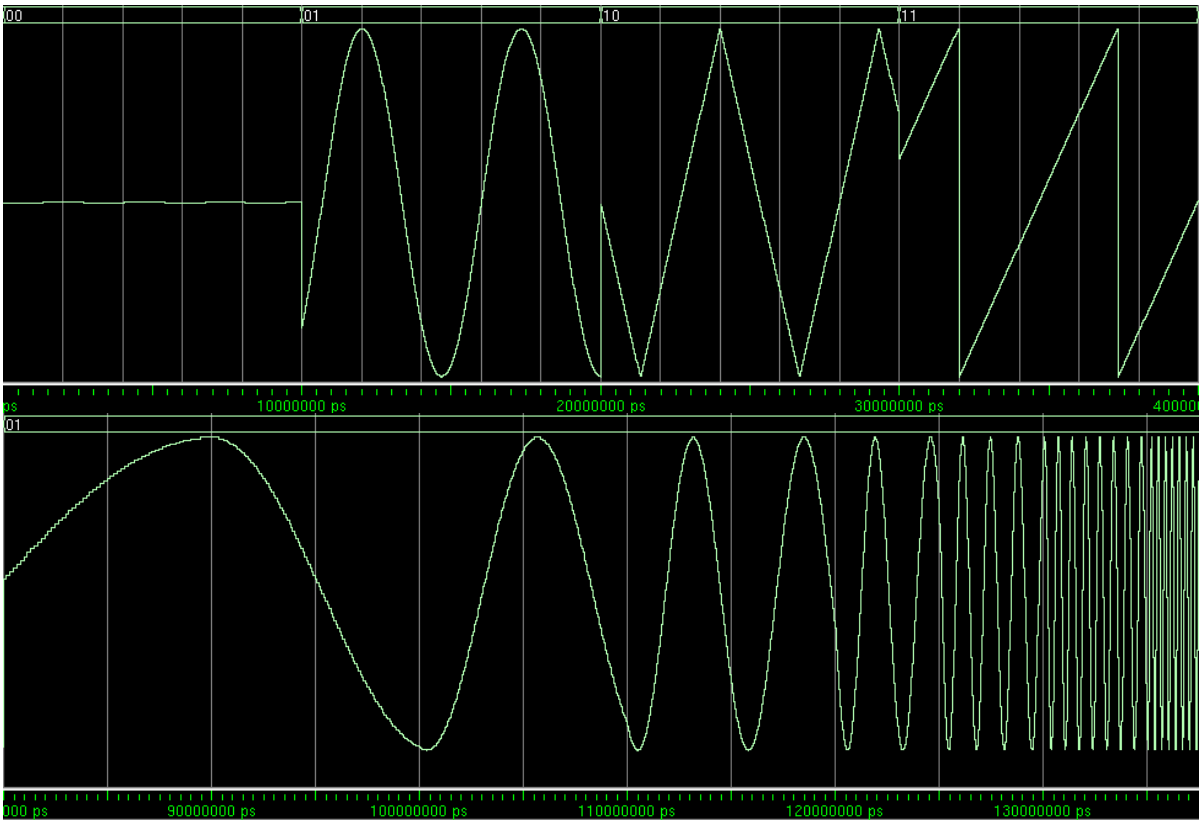


Figure 3a (top): SignalGen output of each waveform at 750 Hz (knobval=64)
Figure 3b (bottom): A sine wave with an exponentially increasing knobval, from 8 to 4095

knobval	overflow value = floor(knobval/64)	Frequency = (48,000 / (256 * overflow))
0 (min)	0	0 Hz
1	Mostly 0. 1 every 64th ready pulse	0.000085 Hz
64	Always 1	750 Hz
4095 (max)	Mostly 63. 62 every 64th ready pulse	12,000 Hz

Table 1: Frequencies and overflow values resulting from characteristic knobvals

SignalGen allows for the construction of a variable-frequency signal in the following manner. Upon each ready pulse, a large counter register is incremented by the current value of the knobval input. The resulting value is then downshifted by 6 bits, effectively dividing the counter by 64. The resulting value indicates how many times the counter would have overflowed if it were only 6 bits in width. Then, the BRAM address is then incremented by the overflow value, and the resulting data is driven to the output upon the next clock cycle.

A knobval of 0 corresponds to an overflow of 0, and thus a frequency of 0 Hz. When the knobval is exactly equal to 64, the overflow value is always 1, and the output is simply the next value in the read-only BRAM. The output will be given a new value on each ready pulse. When the knobval is greater than 64, the overflow value is always greater than or equal to 1, and can rise as far as 63, which means the address is usually incremented by a value greater than 1, achieving a speed-up effect. And when the knobval is less than 64, the overflow value will be either 0 or 1, depending on the point in time. This means that a new sample will only be fetched from the BRAM once every 64 ready pulses. The knobval is, in a sense, directly controlling the frequency of the wave.

Note that the overflow value is always between 63 and 0, which ensures the resulting signal will always consist of at least 4 data points from the BRAM, which is the minimum number of points required to communicate a rough sine wave. Normally a 4 or 5-point sine wave would be too messy to use. However, for knobvals near 1, the wave frequency produced is too high to be of practical use, so knobvals below about (or Hz) are avoided.

Effects Modules

The following modules are the packages which implemented the logic and computation supporting our core effects stream and user interface. Unless otherwise noted, all modules accept as inputs clock, reset and ready signals, and an 18-bit signed audio value, and output an 18-bit signed audio signal.

5 Band Equalizer (Devon)
The equalizer module uses a linear combination of a set of low-pass filters to produce five frequency bands that, when combined, span the range 0 Hz to 7,000 Hz. Each of these bands has its own user defined amplitude. This will give the user control over what frequencies are emphasized and the overall shape of the output tone.

This module takes five 5-bit volume inputs from the user. Each of these volumes is used to scale the volume output from one of the five frequency bands. To create the five frequency bands, the five low-pass filters with cutoff frequencies of 125 Hz, 275 Hz, 530 Hz, 1100 Hz, and 7000 Hz were used. These frequencies were chosen logarithmically because as frequency increases, the distance between note values also increases. The goal was to assign each frequency band approximately the same note range, and to do this, the frequencies had to be logarithmic. The first frequency band ranges from 0 Hz to 125 Hz. This is simply the output from the 125 Hz low-pass filter. The rest of the frequency bands were created by subtracting the output from one of the low-pass filters from the next higher frequency low-pass filter's output. For example, to create the second lowest frequency band, the output from the 125 Hz low-pass filter was subtracted from the output of the 275 Hz low-pass filter.

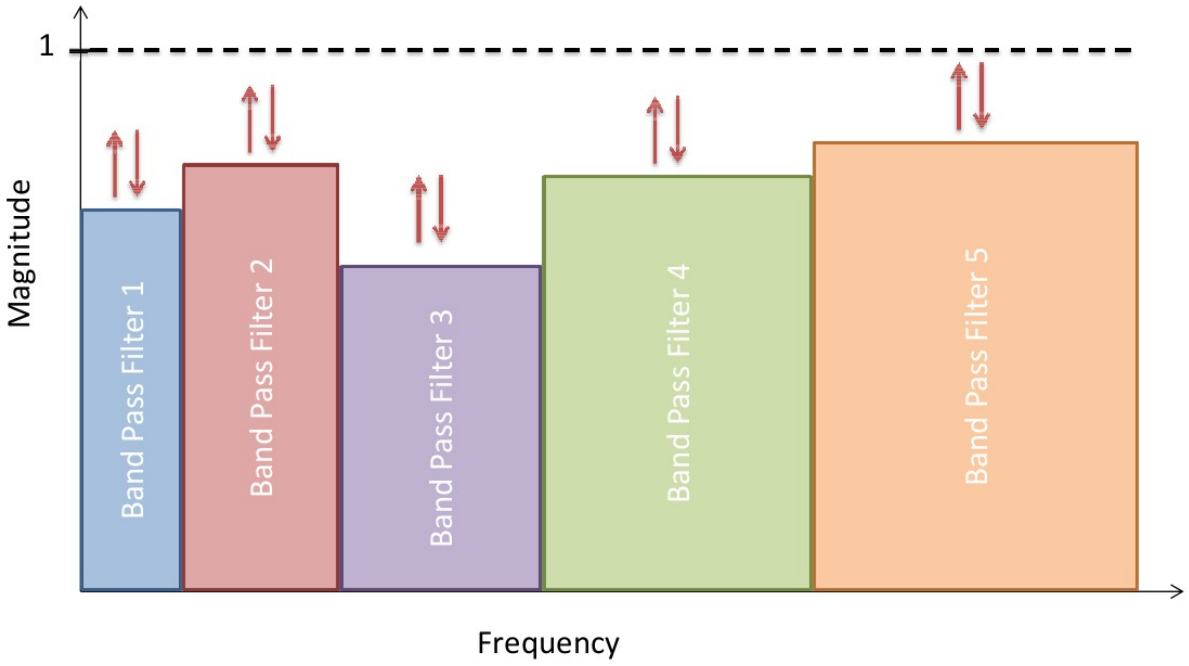


Figure 4: the Equalizer's five frequency domains

Each of the outputs from the five frequency bands is multiplied by its respective user defined 5-bit volume. The 18 highest order bits are taken from each of these multiplications to ensure that the output is less than or equal to the original volume of the low-pass filter's output. The five 18-bit values are added together to produce the final output audio of the equalizer. In addition to giving the user the ability to select and diminish the volume of specific frequency ranges, it also acts as noise filter and cleans up the audio signal tremendously.

Tremolo (Devon)
The tremolo module takes the input audio and produces a stuttering effect. This is done by outputting the exact audio input for half of a user defined period and outputting a quieter version of the audio input for the other half.

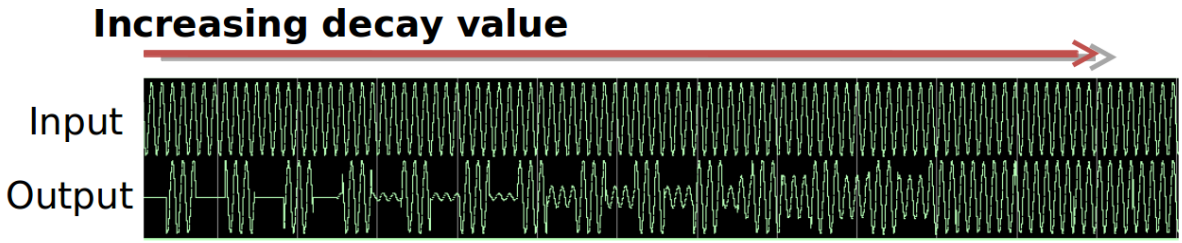


Figure 5: the effects of tremolo on a signal

This module takes a 10-bit decay value and 10-bit period from the user. Every ready pulse, a counter is incremented by one. Once the counter reaches the user defined period (which is multiplied by 26 so that the period is of audible length), a two-state state machine switches states and the counter resets. The first state of the state machine directly outputs the input audio. The second state of the state machine multiplies the input audio by the user defined 10-bit decay value, and outputs the highest order 18 bits of the resulting 28-bit value. The result of taking the highest order bits is a decayed value of the original 18-bit input. Since the decayed value of the input audio and the exact value of the input audio are switched as outputs every time the counter reaches the user defined period, and the user defined period is restricted to short time intervals, the resulting output audio has a stuttering effect.

Long Delay (Dylan)
Long delay is a repeated echo effect, allowing for audio to be replayed after a delay of up to almost 3 seconds. The incoming audio signal will still be played over the repeated audio, and is mixed into subsequent echoes. It accepts as inputs a 10-bit delay value and a 10-bit decay value. The delay value controls the high order bits of the address offset value used to fetch previous samples. The decay value determines how loud the delayed audio will be.

Long Delay uses an 18-bit BRAM with an address width of 16 bits, for a total of under 1.2Mb. To double the length of time the Long Delay effect can support, incoming audio is downsampled by a factor of 2, and the output is similarly upsampled by a factor of 2. On every other ready pulse (i.e., operating at 24000Hz), the BRAM address is shifted to point to the delayed value by subtracting the delay address offset, which is simultaneously buffered

to avoid adverse effects from glitchy or transitioning input. Two buffer states are inserted to ensure the value is successfully fetched, and to multiply the freshly retrieved delayed data by the decay value. Next, the address pointer is restored to its original value by adding the delay address offset, and is further incremented by one to point to a fresh location in memory. The scaled delayed sample is then added to the current audio input and written to the BRAM, and the FSM returns to its idle state for the next alternate ready pulse.

Reverb (Dylan)
Reverb possesses the same behavior as delay, but with a smaller BRAM capacity of just over 340ms. An instance of the Long Delay is instantiated with this smaller capacity. Reverb takes as inputs a wave decay enable bit, two-bit decay wavesel, a 11-bit decay knobval, a 10-bit decay value, and 10-bit delay value.

In an attempt to recreate various forms of decay such as spring reverb, hall reverb, and room reverb, Reverb supports nonlinear trends in its decay value by using the SignalGen module to create a fluctuating decay value. If the wave decay enable bit is low, the input decay value is fed directly to the small Long Delay instance. If the wave decay enable bit is high, the decay wavesel and the decay knobval are used to control the resulting decay waveform, which is input to the small Long Delay instance.

Distortion (Devon)
The distortion module simulates the sound of an incoming audio signal that is too loud and overdrives the amplifier. It does this by setting a threshold bar and clipping the input audio to that threshold. The audio is then multiplied by a gain to increase its volume. This causes the output to sound fuzzy, which is the desired effect of distortion. Clipping is handles in one of two ways. First, there is hard clipping. Hard clipping is simply setting all of the input audio with magnitude greater than the threshold to the threshold value. This creates a quasi-square

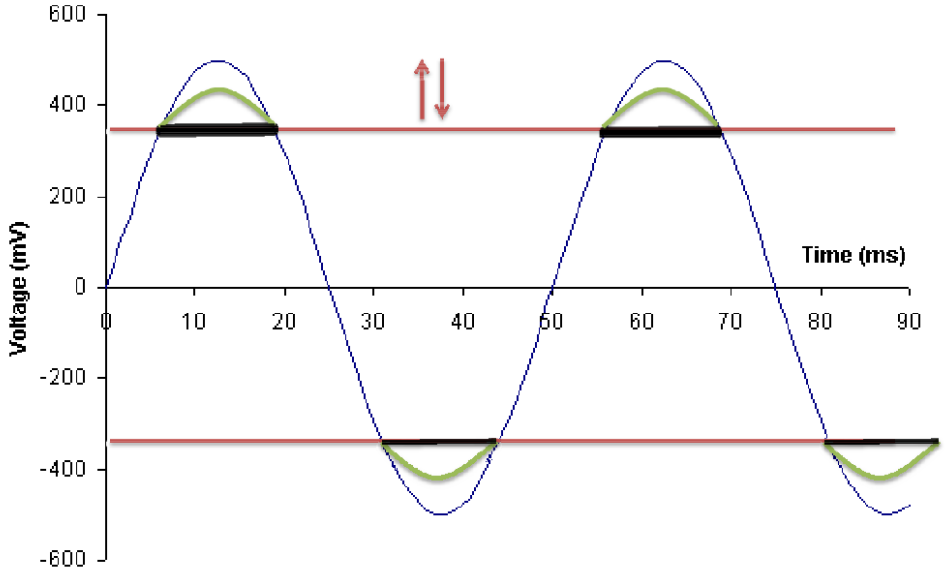


Figure 6: the effects of hard clipping (black) and soft clipping (green). The threshold value is shown here in red.

wave out of the input audio, which causes the effect to have a strong impact on the sound. Soft clipping is the second method of clipping. Instead of setting all of the audio above a threshold value to the threshold value, it is scaled down by a constant. This makes the signal maintain some of its basic shape. The downscaling of audio above the threshold still produces a distortion effect, just not as strong as in hard clipping.

This module takes a 7-bit threshold, 6-bit overdrive scalar, and 7-bit gain from the user. The 7-bit threshold is concatenated with 10 zeros following it to make the threshold values fall into the range of actual audio volume. A negative threshold, which is just the input threshold multiplied by negative one, is also created to allow for checking and clipping of input audio above the positive threshold and below the negative threshold. The threshold value gets subtracted from any audio above the threshold, and the resulting value is multiplied by the overdrive scalar. This value gets added back to the threshold value. When the overdrive scalar is set to zero this produces hard clipping, and when it is greater than zero it produces soft clipping. Finally, the clipped signal is multiplied by the gain, making the signal either quieter or louder.

ZBT Looper, Looper Control and Interface (Dylan and Devon)
The Looper module enables the recording and playback of four audio channels stored on the ZBT ram. It imposes a 2x downsampling rate yielding a length per channel of approximately 11 seconds. It contains one sub-module: the Looper Interface, which handles the channel logic and serves as an interface to drive the ZBT chip signals. The control signals of the Looper module come from the Looper Control module, which constitutes the user interface.

Looper Control
The Looper Control module takes as inputs record signals from channels 0-3 (buttons 0-3) and a play/stop signal (the enter button), and outputs a two-bit channel selector, a two-bit action signal, and a four-bit channel enable list for playback.

The design of the Looper Control FSM fully describes the functionality of the ZBT Looper

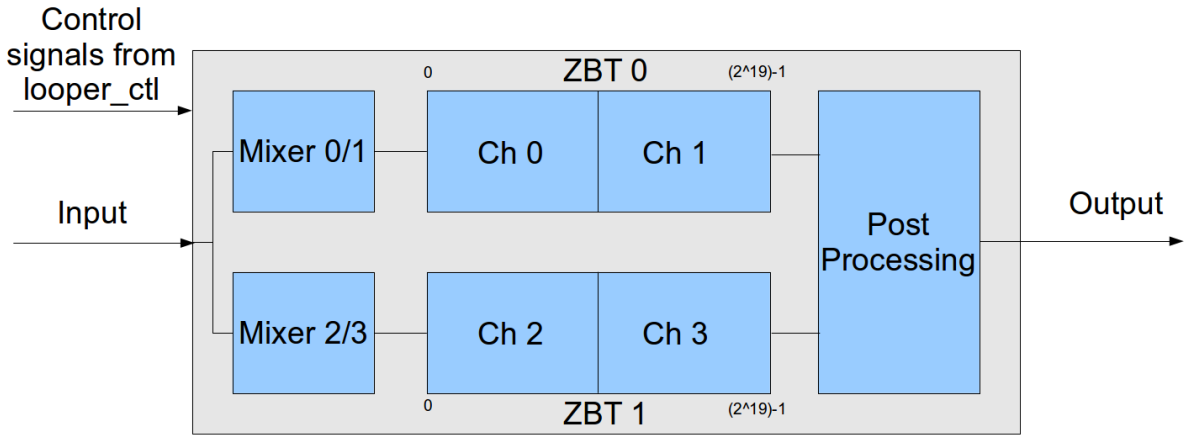


Figure 7: The ZBT Looper module is shown in gray, with the components of the Looper Interface shown in blue.

package by expressing the following interface: Each channel is recorded to individually by holding down the record/enable button assigned to the channel. The output from each channel, if any, is added with the original input to produce the Looper audio out, meaning that audio will still pass through the Looper during recording. The user can hold enter and press each of the record/enable buttons to enable or disable the corresponding channel. LEDs 0 through 3 display this information. Pressing the enter button toggles playback. LED 7 will blink if the Looper is in playback mode, remain on if the Looper is recording audio to a channel, and remain inactive otherwise. Finally, the entire Looper effect can be enabled or disabled via the menu option “OnOff,” along with the menu parameter “Loop” which specifies whether playback should cycle indefinitely or terminate after one iteration.

The Looper Control module's action signal specifies if the Looper should remain idle (00), enter playback mode (01), or record to the channel specified by the channel selector (10). The higher and lower order bits of the action signal are respectively connected to the write and read enables of the Looper Interface. Similarly, the channel enable values serve as selectors for four muxes in the Looper module which will drive the playback output volume for each channel to 0 if that channel's enable signal is low.

Looper
The Looper module is a convenience wrapper for the Looper Interface. It requires as inputs a signal specifying whether looping should occur on playback, the enter button, a two-bit channel selector and a two-bit action selector, a 16-bit volume vector detailing the 4-bit playback volume for each of the four channels, a 4-bit enable signal specifying which channels should be muted during playback, and 36 bits of read data for each of the ZBT chips, which comes from the top-level labkit. The 36 bits consist of an 18-bit left signal in the high order bits, followed by an 18-bit right signal. It outputs a 36-bit write data field, 19-bit address and 1-bit write enable for each ZBT chip, as well as an ultimate 36-bit audio output. Some state and address pointer information is also output for debugging purposes. The Looper module buffers the write and read enable signals from the action selector, driving them as inputs to the Looper Interface.

Looper Interface
The Looper Interface has almost identical inputs and outputs to the Looper module, except it requires a write and read enable signal as input instead of an action selector. It contains an FSM which provides the core logic necessary to retain four partitioned channels of audio, and for handling downsampling. It contains 11 states due to the need for several buffer states when changing the ZBT input address.

The Looper Interface maintains four 19-bit channel address “end” pointer values, each consisting of a 0 or 1 followed by an 18-bit register-based modular address. These pointers indicate how much data has been written to each channel. On every other ready pulse, if write enable is asserted and a particular channel selector value is provided, the Looper Interface increments the appropriate channel address and updates the address and write enable value of the correct chip. Two buffer cycles later, the Looper Interface returns to an idle state.

Otherwise, if write enable is low but read enable is high, the Looper Interface must fetch two data values from each ZBT chip, as there are two channels per chip. The Looper Interface first reads the data values for channels 0 and 2, the two lower order channels, followed by the values for channels 1 and 3. To accomplish this, it increments a second set of pointers reserved for playback, and updates each chip address to the values of those pointers for channels 0 and 2. After two buffer cycles, the data is saved and the chip addresses are updated to the freshly incremented playback pointers for channels 1 and 3. After two more buffer cycles, the four data values are multiplied by the channel volumes and summed to produce an output.

The Looper Interface contains logic for resetting the playback addresses. A two-bit counter is kept, and whenever a channel's playback pointer reaches its end pointer, the counter is incremented. When the counter reaches three, the longest channel has completed playback, and the playback pointers are reset. If looping is enabled, playback continues from the start. Or, if read enable has just transitioned from 0 to 1 and write enable is still 0, all playback addresses are driven to 0, and playback begins.

Similarly, if write enable has just transitioned from 0 to 1, the end pointer for the channel being written to is set to 0, “forgetting” any data which may have previously existed in the buffer.

Wah/Auto Wah (Devon)

The wah module applies a dynamic band-pass filter to the input audio signal. The band-pass filter’s frequency band can shifted by a set period or by an external potentiometer. Shifting the band-pass filter quickly over the input audio produces a sound similar to a human saying “wah” in the same pitch(es) as the audio input. This effect has two modes. The auto wah mode has a periodic band-pass filter that will only shift frequency when the input volume exceeds a certain threshold. Manual wah mode allows an external potentiometer controlled by the user to shift the bandpass filter.

This module takes 1-bit wah mode (manual or auto), 7-bit auto wah volume threshold, 2-bit bandpass frequency width, 6-bit auto wah period, and 5-bit manual band-pass position as inputs from the user. For the band-pass filters, 11 low-pass filters were used in linear combination. The 11 frequencies of these low-pass filters are 385 Hz, 450 Hz, 530 Hz, 620 Hz, 730 Hz, 840 Hz, 960 Hz, 1100 Hz, 1300 Hz, 1600 Hz, and 7000 Hz. With the exception of the first band-pass filter being only the 385 Hz low-pass filter, every other band-pass filter is created by subtracting the output of a lower frequency low-pass filter from a higher frequency low-pass filter. The frequency widths of these band-pass filters are determined by the 2-bit bandpass frequency width user input. If this input is zero, then the low-pass filter subtracted from the higher one is only one low-pass filter frequency range away. For example, the output of the 450 Hz low-pass filter is subtracted from 530 Hz low-pass filter since 530 Hz is the next highest low-pass filter from 450 Hz. If the user defines the width to be two, then the output of the 450 Hz low-pass filter is subtracted from the output of the 730 Hz low-pass filter since the 730 Hz low-pass filter is three frequency ranges higher than 450 Hz. The ability to choose a width gives the user control over how drastic the wah effect is.

When the user selects ‘manual wah’ mode they are choosing to control which band-pass filter is selected via an external potentiometer. The external potentiometer is connected to +5V and an ADC0801 which digitizes the analog data from the potentiometer and outputs the data as 8 bits to the board’s user in/outs. Since the manual band-pass position input is only 5 bits, the five high order bits are taken from the eight bit input. Using a python script, the band-pass filters are divided up as evenly as possibly into 32 cases to support the 5-bit input. This is especially useful since the width of the band-pass filters is dependent on the user input, which changes the total number of band-pass filters (11 when width is 0, 10 when width 1, 9 when width is 2, and 8 when width is 3). To make the manual wah as authentic as possible, we obtained a broken wah pedal. All electronics with the exception of the potentiometer were gutted from the pedal. The pedal mechanism allows the user to shift the potentiometer’s position and band-pass filter frequency position by tilting the pedal up and down, just like a real wah pedal.

The second mode of operation is ‘auto wah’ mode. When this mode is selected, the band-pass filter is shifted at a speed determined by the user input 6-bit auto wah period. Whether the band-pass filter is moving or not depends on the input volume. If the input volume’s magnitude is above the 7-bit auto wah volume threshold, then the band-pass filter will shift on the user-defined period, otherwise it will remain still. This is a necessary and useful feature because without it, the band-pass filter will shift when nothing is being played, so the listener will hear the audio signal noise being swept over by the band-pass filter.

Ring Modulator (Dylan)

Ring modulation, colloquially known as “MoogerFooger” in reference to synthesizer designer Moog Music, is an audio effect that multiplies the incoming audio by another signal. The signals supported by the Ring Modulator module are sine waves, square waves, saw-tooth waves, and triangle waves from the SignalGen module. The module accepts as inputs a two-bit wave selector, a 10-bit knobval, and a 6-bit amount.

The wave selector and knobval are fed into the SignalGen instantiation as control variables. The amount dictates how much of the output is ring modulated. On each ready pulse, the Ring Modulator performs the following operations, where each stage is pipelined: first, incoming audio is multiplied by the SignalGen output, and the appropriate downshift is performed. Then, the resulting value is multiplied by the input amount, and the input value is multiplied by the inverse of the amount. The sum of the two values is accordingly downshifted to produce an output value.

Panning (Devon)

The pan module splits the input mono audio into a stereo output. The two modes of pan are “ping pong” and “fade” pan. Ping pong pan plays the incoming audio through one speaker for half of a period and through the other speaker for the other half. Fade pan fades the audio signal gradually from one speaker to the other over a period.

This module takes an 8-bit period input and a 1-bit mode select input. When in ping pong

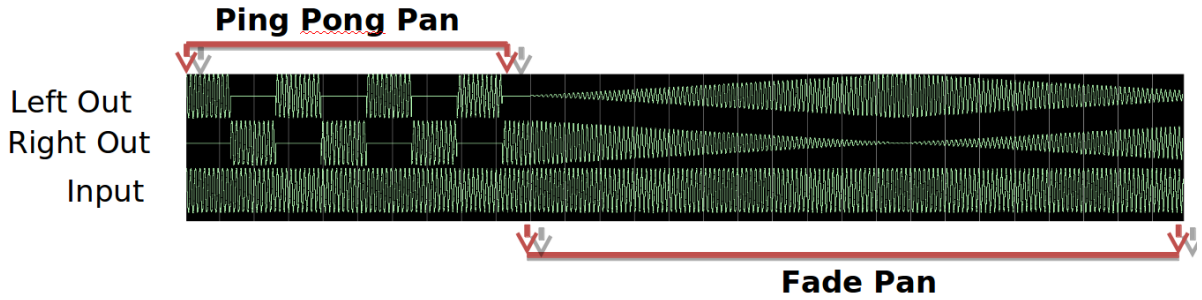


Figure 8: the two modes of pan

pan mode, a counter increases every clock pulse until it reaches the user-defined period. Once it reaches this value, the incoming audio is only sent as an output to one of the two speakers. The counter is reset, and the next time it reaches the user-defined period, it outputs only to the other speaker. This switching is done by a simple two-state state machine which switches state each time the counter equals the user-defined period. As the speakers alternate outputting the audio signal, the ping pong pan effect is produced.

Fade pan uses the counter as a scalar, and treats the period input as a speed input (greater values make the fade faster instead of slower). In this implementation of pan, the counter is always counting to a constant value, but the counter is increased by the user defined speed every clock cycle. So, the greater the input, the faster the counter reaches the constant. Also, once the counter increases to the constant value, it reverses direction, counts down to zero, and repeats this loop. This is done to simplify the scaling of the audio signal in the two speakers. In addition to their being a counter register, there is also an anti-counter. This value is the counter value subtracted from the constant. As the counter increases, the anti-counter decreases, and vice-versa. The volume of the audio in the left speaker is the most significant 18 bits of the counter multiplied by the input audio, whereas the volume of the audio in the right speaker is the most significant 18 bits of the anti-counter multiplied by the input audio. As the counter is increasing to the constant value, the audio in the left speaker is increasing while the audio in the right speaker is decreasing. As the counter is decreasing to zero, the audio in the left speaker is decreasing, while the audio in the right speaker is increasing. The gradual increase in volume of one speaker that occurs simultaneously with the gradual decrease in volume of the other speaker causes the fading pan effect.

Chorus (Dylan and Devon)

The chorus module makes one instrument sound like multiple instruments are playing the same audio sample. To accomplish this, a fixed delay is applied to the audio similar to that of reverb. These recent blocks will be repeated, but played back at a varying speeds. This will give the illusion that multiple instruments are playing, as two musicians are never exactly in tune or phase with each other.

This module takes a 2-bit wave select, 10-bit delay value, 8-bit low frequency oscillator (LFO) depth, 9-bit LFO rate, and 10-bit decay value as user inputs. An LFO is the means by which the playback of the recorded audio sample is played back at varying speeds. The LFO is generated by the signal generator function, and the waveform used for the LFO is chosen

by the wave select input. The LFO's magnitude depends on the depth input, and it's rate depends on the rate input. When subtracted from the address of the stored audio sample's BRAM during reading, the playback of the audio is slightly (or heavily if desired) modulated since the playback is not at the same rate as when the audio was recorded.

Playing back the audio at a varying rate, while recording at a constant rate was a challenge that was assisted by saving the previous write address into a register. When transitioning from reading to writing, the current write address is assigned the saved write address plus one.

FX Controller (Devon)

The FX controller takes the user input and provides the necessary values to all of the effects and ZBT interfacing modules. It is also responsible for relaying data to and from the ZBT SRAM and ADC. Using the alphanumeric display and debounced labkit directional buttons, the user is able to scroll through the menu of effects inside the FX controller and alter specific effects parameters as desired. Essentially, the FX controller takes everything we have created for our project and connects it with the labkit.

The menu system was created within the first week of the project to allow for an easy way to put effects on the board and see if they work/debug them. The menu is a giant state machine where every effect module is a different state which contains its own state machine where the parameters of the effect are the states. To scroll through the menu, the user presses up or

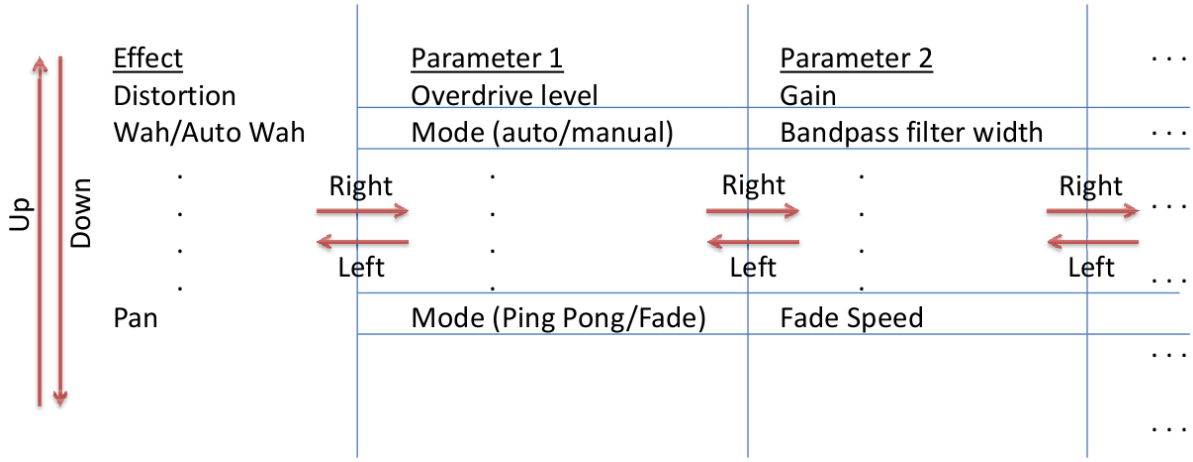


Figure 9: FX Controller menu operation

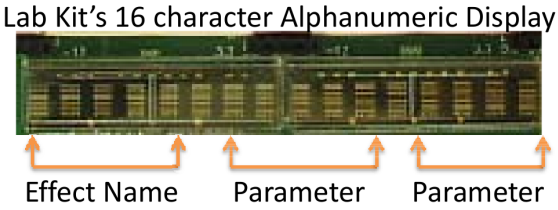


Figure 10: the FX Controller menu alphanumeric display

down to change effects, then presses right or left to enter the effect's inner parameter state machine. Once inside the inner state machine, pressing up or down changes the value of the parameter selected. The user can tell what effect they are on by looking at the five left-most characters on the 16-character led display. After a space, the next five characters are the parameter, and the last three characters on the display is the parameter value. Having this menu system early allowed us to put every new effect and feature directly onto the board and have full control of the parameter values.

In addition to the menu system, the FX controller wires up the effect stream, so it takes the input audio, scales it down by one half to prevent overdriving the output drivers when reverb or delay are enabled, passes it through the effect stream, through the four channel recorder/looper, and sends the output to the modified lab 5 labkit module.

Future Work

A common enhancement to synthesizers with Ring Modulator-like effects is the incorporation of two or three oscillators into the signal, each producing a different waveform with a distinct period. Such an addition would improve the creative range the effect could provide, and would only entail adding several states to the Ring Modulator module, and several entries in the FX Controller menu.

Many long delay pedals have an option to only repeat an echo once, making the effect more like a long distance echo and less like a fading looper. One simple method of achieving this goal is to add a separate track for recording echo values. This allows the effect to remain isolated in a separate location from the pure audio.

Most reverb boxes and simulators support many control parameters which produce distinctly recognizable characteristics. Our reverb module produced a beautiful sound, but was not as versatile and customizable as we may have hoped. Therefore, improving upon reverb would represent a major functional augmentation. Further research would be required in order to recognize feasible methods for achieving this.

The FX Control Menu is a better interface than those found in many commercial systems. Nevertheless, effects boxes traditionally make use of sliders and knobs to tune certain effects due to their tangible nature, easy access and universal intuition. The ability to assign sliders or knobs to specific effects would make a powerful addition to Omnibox's functionality, fluidity and ease of use. As we already utilize an analog value (the wah pedal) to control one of our effects, the only additions required would be A) an analog mux, to allow one ADC to sample from several sources, and B) a module to control the analog mux, and ensure that the retrieved values are used appropriately

Whether in software or hardware, musicians who have an array of effects processing units learn the specifics of each unit. Most units, especially those classified as multi-effect systems, have some means of constructing, saving and easily reloading valuable presets. Granting Omnibox users the option of saving favored configurations for select parameters could prove highly useful.

Conclusion

Omnibox is a powerful, high fidelity multi-feature audio effects module which enables the application of realtime digital effects to streaming audio. With a clean interface for inputting and receiving high quality 48 kHz audio, a simple switch array for enabling effects, and a streamlined effects menu with text and numerical output that grants easy access to core parameters, Omnibox presents a compelling package. The dynamic range of effects supported by Omnibox include distortion/compression, wah, auto wah, ring modulation, reverb, chorus, tremolo, long delay, five-band equalization, panning, and a four-channel recorder, looper and mixer. Additionally, the wah pedal facilitates full musical expression while using the wah effect. Omnibox provides a wide range of high quality audio effects at a low price, and is something every musician should own.

Acknowledgements

We would like to thank the lab staff (Prof. Stojanovic, Gim Hom, John Losh, Kevin Zheng and Krishna Settaluri) for countless hours of crucial help in lab, and their continued support on this and other 6.111 assignments. We would also like to offer our most sincere gratitude to the CI-M staff for their invaluable insight, advice and critiques, and for providing us with opportunities to grow as effective communicators. Thank you all for an excellent term!

Appendix A: Menu Functions

“Disto” Distortion	“Thresh” Distortion threshold value “Overd” Overdrive, amount by which to multiply initial signal if it exceeds the threshold “Gain” Amount by which to amplify compressed signal
“Wah” Wah	“Auto” Enable Auto-wah “Wah_P” auto-wah volume threshold “Width” auto-wah wave amplitude “Perio” period of waveform “Thresh” in manual wah, controlled by pedal.
“Moog” Ring Mod.	“Wave” wave selector “Amnt” amount of moog effect “Speed” frequency of waveform
“Revrb” Reverb	“Wavon” control decay value with waveform “Wave” wave selector “Delay” amount of delay “Decay” constant by which previous samples are multiplied “WvFrq” frequency of waveform
“Chor” Chorus	“ChoOn” activate LFO “Wave” wave selector “Delay” amount of delay, not including LFO “Depth” LFO amplitude “Wobbl” LFO frequency “Decay” constant by which previous samples are multiplied
“Tremo” Tremolo	“Decay” constant by which previous samples are multiplied “Perio” period of waveform
“Delay” Longdel	“Delay” amount of delay “Decay” constant by which previous samples are multiplied
“5BdEq” 5-band EQ	“LowV” Low range volume “LoMdV” Low-mid range volume “MidV” Mid-range volume “MdHiV” Mid-high range volume “HighV” High-range volume
“Pan” Pan	“Fade” enable/disable fading “Perio”/”Speed” If fade, control speed. Else, control period
“Loop” Looper	“OnOff” Enable/disable effect “Loop” Enable/disable looping playback “C0Vol” Channel 0 playback volume “C1Vol” Channel 1 playback volume “C2Vol” Channel 2 playback volume “C3Vol” Channel 3 playback volume
“MstrV” Volume	“Volum” master input volume

Appendix B: Core Code

```
module firlow125(
  input wire clock,reset,ready,
  input wire signed [17:0] x,
  output reg signed [17:0] y
);
        reg signed [17:0]big_array[250:0]; //change big_array[x:0] x to coeff #
        reg [7:0]index = 0;
        reg [7:0]offset = 0;
        wire signed [11:0]coeff;
        reg signed [29:0]accumulator = 0;
        reg go = 0;

  always @(posedge clock) begin
                if (reset) begin //reinitialize everything
                        offset <= 0;
                        index <= 0;
                        accumulator <= 0;
                        go <= 0;end
                else if (ready) begin //on ready insert x into big_array and reset index
                        big_array[offset] <= x;
                        index <= 0;
                        go <= 1;end
                else if (go == 1) begin //while accumulator is accumulating
                        if (index == 8'd250) begin //change 8'dx x to coeff #
                                index <= 0;
                                go <= 0;
                                offset <= offset + 1;
                                y <= accumulator[29:12] <<< 1;
                                accumulator <= 0;
                                end
                        else begin //accumulate procedure
                                index <= index + 1;
                                accumulator <= accumulator + coeff*big_array[offset - index];end
                        end
                else;
  end
  firlow125coeffs coeffs(index,coeff);
endmodule

module firlow125coeffs(
  input wire [7:0] index,
  output reg signed [11:0] coeff
);
  // tools will turn this into a 31x10 ROM
  always @(index)
    case (index)
        8'd0:  coeff = 12'sd1;
        8'd1:  coeff = 12'sd1;
        8'd2:  coeff = 12'sd1;
        8'd3:  coeff = 12'sd1;
        8'd4:  coeff = 12'sd1;
        8'd5:  coeff = 12'sd1;
        8'd6:  coeff = 12'sd1;
        8'd7:  coeff = 12'sd1;
        8'd8:  coeff = 12'sd1;
        8'd9:  coeff = 12'sd1;
        8'd10:  coeff = 12'sd1;
        8'd11:  coeff = 12'sd1;
        8'd12:  coeff = 12'sd1;
        8'd13:  coeff = 12'sd1;
        8'd14:  coeff = 12'sd1;
        8'd15:  coeff = 12'sd1;
        8'd16:  coeff = 12'sd1;
        8'd17:  coeff = 12'sd1;
        8'd18:  coeff = 12'sd1;
        8'd19:  coeff = 12'sd1;
        8'd20:  coeff = 12'sd1;
        8'd21:  coeff = 12'sd1;
        8'd22:  coeff = 12'sd1;
        8'd23:  coeff = 12'sd2;
        8'd24:  coeff = 12'sd2;
        8'd25:  coeff = 12'sd2;
        8'd26:  coeff = 12'sd2;
        8'd27:  coeff = 12'sd2;
        8'd28:  coeff = 12'sd2;
        8'd29:  coeff = 12'sd2;
        8'd30:  coeff = 12'sd2;
        8'd31:  coeff = 12'sd2;
        8'd32:  coeff = 12'sd2;
        8'd33:  coeff = 12'sd3;
        8'd34:  coeff = 12'sd3;
        8'd35:  coeff = 12'sd3;
        8'd36:  coeff = 12'sd3;
        8'd37:  coeff = 12'sd3;
        8'd38:  coeff = 12'sd3;
        8'd39:  coeff = 12'sd3;
        8'd40:  coeff = 12'sd4;
        8'd41:  coeff = 12'sd4;
```

```
8'd42:  coeff = 12'sd4;
8'd43:  coeff = 12'sd4;
8'd44:  coeff = 12'sd4;
8'd45:  coeff = 12'sd4;
8'd46:  coeff = 12'sd4;
8'd47:  coeff = 12'sd5;
8'd48:  coeff = 12'sd5;
8'd49:  coeff = 12'sd5;
8'd50:  coeff = 12'sd5;
8'd51:  coeff = 12'sd5;
8'd52:  coeff = 12'sd6;
8'd53:  coeff = 12'sd6;
8'd54:  coeff = 12'sd6;
8'd55:  coeff = 12'sd6;
8'd56:  coeff = 12'sd6;
8'd57:  coeff = 12'sd6;
8'd58:  coeff = 12'sd7;
8'd59:  coeff = 12'sd7;
8'd60:  coeff = 12'sd7;
8'd61:  coeff = 12'sd7;
8'd62:  coeff = 12'sd8;
8'd63:  coeff = 12'sd8;
8'd64:  coeff = 12'sd8;
8'd65:  coeff = 12'sd8;
8'd66:  coeff = 12'sd8;
8'd67:  coeff = 12'sd9;
8'd68:  coeff = 12'sd9;
8'd69:  coeff = 12'sd9;
8'd70:  coeff = 12'sd9;
8'd71:  coeff = 12'sd9;
8'd72:  coeff = 12'sd10;
8'd73:  coeff = 12'sd10;
8'd74:  coeff = 12'sd10;
8'd75:  coeff = 12'sd10;
8'd76:  coeff = 12'sd11;
8'd77:  coeff = 12'sd11;
8'd78:  coeff = 12'sd11;
8'd79:  coeff = 12'sd11;
8'd80:  coeff = 12'sd11;
8'd81:  coeff = 12'sd12;
8'd82:  coeff = 12'sd12;
8'd83:  coeff = 12'sd12;
8'd84:  coeff = 12'sd12;
8'd85:  coeff = 12'sd12;
8'd86:  coeff = 12'sd13;
8'd87:  coeff = 12'sd13;
8'd88:  coeff = 12'sd13;
8'd89:  coeff = 12'sd13;
8'd90:  coeff = 12'sd13;
8'd91:  coeff = 12'sd14;
8'd92:  coeff = 12'sd14;
8'd93:  coeff = 12'sd14;
8'd94:  coeff = 12'sd14;
8'd95:  coeff = 12'sd14;
8'd96:  coeff = 12'sd14;
8'd97:  coeff = 12'sd15;
8'd98:  coeff = 12'sd15;
8'd99:  coeff = 12'sd15;
8'd100:  coeff = 12'sd15;
8'd101:  coeff = 12'sd15;
8'd102:  coeff = 12'sd15;
8'd103:  coeff = 12'sd15;
8'd104:  coeff = 12'sd16;
8'd105:  coeff = 12'sd16;
8'd106:  coeff = 12'sd16;
8'd107:  coeff = 12'sd16;
8'd108:  coeff = 12'sd16;
8'd109:  coeff = 12'sd16;
8'd110:  coeff = 12'sd16;
8'd111:  coeff = 12'sd16;
8'd112:  coeff = 12'sd16;
8'd113:  coeff = 12'sd16;
8'd114:  coeff = 12'sd17;
8'd115:  coeff = 12'sd17;
8'd116:  coeff = 12'sd17;
8'd117:  coeff = 12'sd17;
8'd118:  coeff = 12'sd17;
8'd119:  coeff = 12'sd17;
8'd120:  coeff = 12'sd17;
8'd121:  coeff = 12'sd17;
8'd122:  coeff = 12'sd17;
8'd123:  coeff = 12'sd17;
8'd124:  coeff = 12'sd17;
8'd125:  coeff = 12'sd17;
8'd126:  coeff = 12'sd17;
8'd127:  coeff = 12'sd17;
8'd128:  coeff = 12'sd17;
8'd129:  coeff = 12'sd17;
8'd130:  coeff = 12'sd17;
8'd131:  coeff = 12'sd17;
8'd132:  coeff = 12'sd17;
8'd133:  coeff = 12'sd17;
```

```
8'd134:  coeff = 12'sd17;
8'd135:  coeff = 12'sd17;
8'd136:  coeff = 12'sd17;
8'd137:  coeff = 12'sd16;
8'd138:  coeff = 12'sd16;
8'd139:  coeff = 12'sd16;
8'd140:  coeff = 12'sd16;
8'd141:  coeff = 12'sd16;
8'd142:  coeff = 12'sd16;
8'd143:  coeff = 12'sd16;
8'd144:  coeff = 12'sd16;
8'd145:  coeff = 12'sd16;
8'd146:  coeff = 12'sd16;
8'd147:  coeff = 12'sd15;
8'd148:  coeff = 12'sd15;
8'd149:  coeff = 12'sd15;
8'd150:  coeff = 12'sd15;
8'd151:  coeff = 12'sd15;
8'd152:  coeff = 12'sd15;
8'd153:  coeff = 12'sd15;
8'd154:  coeff = 12'sd14;
8'd155:  coeff = 12'sd14;
8'd156:  coeff = 12'sd14;
8'd157:  coeff = 12'sd14;
8'd158:  coeff = 12'sd14;
8'd159:  coeff = 12'sd14;
8'd160:  coeff = 12'sd13;
8'd161:  coeff = 12'sd13;
8'd162:  coeff = 12'sd13;
8'd163:  coeff = 12'sd13;
8'd164:  coeff = 12'sd13;
8'd165:  coeff = 12'sd12;
8'd166:  coeff = 12'sd12;
8'd167:  coeff = 12'sd12;
8'd168:  coeff = 12'sd12;
8'd169:  coeff = 12'sd12;
8'd170:  coeff = 12'sd11;
8'd171:  coeff = 12'sd11;
8'd172:  coeff = 12'sd11;
8'd173:  coeff = 12'sd11;
8'd174:  coeff = 12'sd11;
8'd175:  coeff = 12'sd10;
8'd176:  coeff = 12'sd10;
8'd177:  coeff = 12'sd10;
8'd178:  coeff = 12'sd10;
8'd179:  coeff = 12'sd9;
8'd180:  coeff = 12'sd9;
8'd181:  coeff = 12'sd9;
8'd182:  coeff = 12'sd9;
8'd183:  coeff = 12'sd9;
8'd184:  coeff = 12'sd8;
8'd185:  coeff = 12'sd8;
8'd186:  coeff = 12'sd8;
8'd187:  coeff = 12'sd8;
8'd188:  coeff = 12'sd8;
8'd189:  coeff = 12'sd7;
8'd190:  coeff = 12'sd7;
8'd191:  coeff = 12'sd7;
8'd192:  coeff = 12'sd7;
8'd193:  coeff = 12'sd6;
8'd194:  coeff = 12'sd6;
8'd195:  coeff = 12'sd6;
8'd196:  coeff = 12'sd6;
8'd197:  coeff = 12'sd6;
8'd198:  coeff = 12'sd6;
8'd199:  coeff = 12'sd5;
8'd200:  coeff = 12'sd5;
8'd201:  coeff = 12'sd5;
8'd202:  coeff = 12'sd5;
8'd203:  coeff = 12'sd5;
8'd204:  coeff = 12'sd4;
8'd205:  coeff = 12'sd4;
8'd206:  coeff = 12'sd4;
8'd207:  coeff = 12'sd4;
8'd208:  coeff = 12'sd4;
8'd209:  coeff = 12'sd4;
8'd210:  coeff = 12'sd4;
8'd211:  coeff = 12'sd3;
8'd212:  coeff = 12'sd3;
8'd213:  coeff = 12'sd3;
8'd214:  coeff = 12'sd3;
8'd215:  coeff = 12'sd3;
8'd216:  coeff = 12'sd3;
8'd217:  coeff = 12'sd3;
8'd218:  coeff = 12'sd2;
8'd219:  coeff = 12'sd2;
8'd220:  coeff = 12'sd2;
8'd221:  coeff = 12'sd2;
8'd222:  coeff = 12'sd2;
8'd223:  coeff = 12'sd2;
8'd224:  coeff = 12'sd2;
8'd225:  coeff = 12'sd2;
```

```
        8'd226:  coeff = 12'sd2;
        8'd227:  coeff = 12'sd2;
        8'd228:  coeff = 12'sd1;
        8'd229:  coeff = 12'sd1;
        8'd230:  coeff = 12'sd1;
        8'd231:  coeff = 12'sd1;
        8'd232:  coeff = 12'sd1;
        8'd233:  coeff = 12'sd1;
        8'd234:  coeff = 12'sd1;
        8'd235:  coeff = 12'sd1;
        8'd236:  coeff = 12'sd1;
        8'd237:  coeff = 12'sd1;
        8'd238:  coeff = 12'sd1;
        8'd239:  coeff = 12'sd1;
        8'd240:  coeff = 12'sd1;
        8'd241:  coeff = 12'sd1;
        8'd242:  coeff = 12'sd1;
        8'd243:  coeff = 12'sd1;
        8'd244:  coeff = 12'sd1;
        8'd245:  coeff = 12'sd1;
        8'd246:  coeff = 12'sd1;
        8'd247:  coeff = 12'sd1;
        8'd248:  coeff = 12'sd1;
        8'd249:  coeff = 12'sd1;
        8'd250:  coeff = 12'sd1;
      default: coeff = 12'hXXX;
    endcase
endmodule

module firlow275(
  input wire clock,reset,ready,
  input wire signed [17:0] x,
  output reg signed [17:0] y
);
        reg signed [17:0]big_array[250:0]; //change big_array[x:0] x to coeff #
        reg [7:0]index = 0;
        reg [7:0]offset = 0;
        wire signed [11:0]coeff;
        reg signed [29:0]accumulator = 0;
        reg go = 0;

  always @(posedge clock) begin
                if (reset) begin //reinitialize everything
                        offset <= 0;
                        index <= 0;
                        accumulator <= 0;
                        go <= 0;end
                else if (ready) begin //on ready insert x into big_array and reset index
                        big_array[offset] <= x;
                        index <= 0;
                        go <= 1;end
                else if (go == 1) begin //while accumulator is accumulating
                        if (index == 8'd250) begin //change 8'dx x to coeff #
                                index <= 0;
                                go <= 0;
                                offset <= offset + 1;
                                y <= accumulator[29:12]<<<1;
                                accumulator <= 0;
                                end
                        else begin //accumulate procedure
                                index <= index + 1;
                                accumulator <= accumulator + coeff*big_array[offset - index];end
                        end
                else;
  end
  firlow275coeffs coeffs(index,coeff);
endmodule

module firlow275coeffs(
  input wire [7:0] index,
  output reg signed [11:0] coeff
);
  // tools will turn this into a 31x10 ROM
  always @(index)
    case (index)
        8'd0:  coeff = 12'sd0;
        8'd1:  coeff = 12'sd0;
        8'd2:  coeff = 12'sd0;
        8'd3:  coeff = 12'sd0;
        8'd4:  coeff = 12'sd0;
        8'd5:  coeff = 12'sd0;
        8'd6:  coeff = 12'sd0;
        8'd7:  coeff = 12'sd0;
        8'd8:  coeff = 12'sd0;
        8'd9:  coeff = 12'sd0;
        8'd10:  coeff = 12'sd0;
        8'd11:  coeff = 12'sd0;
        8'd12:  coeff = 12'sd0;
        8'd13:  coeff = 12'sd0;
        8'd14:  coeff = 12'sd0;
        8'd15:  coeff = 12'sd0;
        8'd16:  coeff = -12'sd1;
```

```
8'd17:  coeff = -12'sd1;
8'd18:  coeff = -12'sd1;
8'd19:  coeff = -12'sd1;
8'd20:  coeff = -12'sd1;
8'd21:  coeff = -12'sd1;
8'd22:  coeff = -12'sd1;
8'd23:  coeff = -12'sd1;
8'd24:  coeff = -12'sd1;
8'd25:  coeff = 12'sd0;
8'd26:  coeff = 12'sd0;
8'd27:  coeff = 12'sd0;
8'd28:  coeff = 12'sd0;
8'd29:  coeff = 12'sd0;
8'd30:  coeff = 12'sd0;
8'd31:  coeff = 12'sd0;
8'd32:  coeff = 12'sd0;
8'd33:  coeff = 12'sd0;
8'd34:  coeff = 12'sd0;
8'd35:  coeff = 12'sd0;
8'd36:  coeff = 12'sd0;
8'd37:  coeff = 12'sd0;
8'd38:  coeff = 12'sd0;
8'd39:  coeff = 12'sd0;
8'd40:  coeff = 12'sd0;
8'd41:  coeff = 12'sd0;
8'd42:  coeff = 12'sd0;
8'd43:  coeff = 12'sd0;
8'd44:  coeff = 12'sd1;
8'd45:  coeff = 12'sd1;
8'd46:  coeff = 12'sd1;
8'd47:  coeff = 12'sd1;
8'd48:  coeff = 12'sd1;
8'd49:  coeff = 12'sd1;
8'd50:  coeff = 12'sd2;
8'd51:  coeff = 12'sd2;
8'd52:  coeff = 12'sd2;
8'd53:  coeff = 12'sd2;
8'd54:  coeff = 12'sd2;
8'd55:  coeff = 12'sd3;
8'd56:  coeff = 12'sd3;
8'd57:  coeff = 12'sd3;
8'd58:  coeff = 12'sd3;
8'd59:  coeff = 12'sd4;
8'd60:  coeff = 12'sd4;
8'd61:  coeff = 12'sd4;
8'd62:  coeff = 12'sd4;
8'd63:  coeff = 12'sd5;
8'd64:  coeff = 12'sd5;
8'd65:  coeff = 12'sd5;
8'd66:  coeff = 12'sd6;
8'd67:  coeff = 12'sd6;
8'd68:  coeff = 12'sd6;
8'd69:  coeff = 12'sd7;
8'd70:  coeff = 12'sd7;
8'd71:  coeff = 12'sd7;
8'd72:  coeff = 12'sd8;
8'd73:  coeff = 12'sd8;
8'd74:  coeff = 12'sd9;
8'd75:  coeff = 12'sd9;
8'd76:  coeff = 12'sd9;
8'd77:  coeff = 12'sd10;
8'd78:  coeff = 12'sd10;
8'd79:  coeff = 12'sd11;
8'd80:  coeff = 12'sd11;
8'd81:  coeff = 12'sd11;
8'd82:  coeff = 12'sd12;
8'd83:  coeff = 12'sd12;
8'd84:  coeff = 12'sd13;
8'd85:  coeff = 12'sd13;
8'd86:  coeff = 12'sd14;
8'd87:  coeff = 12'sd14;
8'd88:  coeff = 12'sd14;
8'd89:  coeff = 12'sd15;
8'd90:  coeff = 12'sd15;
8'd91:  coeff = 12'sd16;
8'd92:  coeff = 12'sd16;
8'd93:  coeff = 12'sd16;
8'd94:  coeff = 12'sd17;
8'd95:  coeff = 12'sd17;
8'd96:  coeff = 12'sd18;
8'd97:  coeff = 12'sd18;
8'd98:  coeff = 12'sd18;
8'd99:  coeff = 12'sd19;
8'd100:  coeff = 12'sd19;
8'd101:  coeff = 12'sd20;
8'd102:  coeff = 12'sd20;
8'd103:  coeff = 12'sd20;
8'd104:  coeff = 12'sd21;
8'd105:  coeff = 12'sd21;
8'd106:  coeff = 12'sd21;
8'd107:  coeff = 12'sd21;
8'd108:  coeff = 12'sd22;
```

```
8'd109:  coeff = 12'sd22;
8'd110:  coeff = 12'sd22;
8'd111:  coeff = 12'sd22;
8'd112:  coeff = 12'sd23;
8'd113:  coeff = 12'sd23;
8'd114:  coeff = 12'sd23;
8'd115:  coeff = 12'sd23;
8'd116:  coeff = 12'sd23;
8'd117:  coeff = 12'sd24;
8'd118:  coeff = 12'sd24;
8'd119:  coeff = 12'sd24;
8'd120:  coeff = 12'sd24;
8'd121:  coeff = 12'sd24;
8'd122:  coeff = 12'sd24;
8'd123:  coeff = 12'sd24;
8'd124:  coeff = 12'sd24;
8'd125:  coeff = 12'sd24;
8'd126:  coeff = 12'sd24;
8'd127:  coeff = 12'sd24;
8'd128:  coeff = 12'sd24;
8'd129:  coeff = 12'sd24;
8'd130:  coeff = 12'sd24;
8'd131:  coeff = 12'sd24;
8'd132:  coeff = 12'sd24;
8'd133:  coeff = 12'sd24;
8'd134:  coeff = 12'sd23;
8'd135:  coeff = 12'sd23;
8'd136:  coeff = 12'sd23;
8'd137:  coeff = 12'sd23;
8'd138:  coeff = 12'sd23;
8'd139:  coeff = 12'sd22;
8'd140:  coeff = 12'sd22;
8'd141:  coeff = 12'sd22;
8'd142:  coeff = 12'sd22;
8'd143:  coeff = 12'sd21;
8'd144:  coeff = 12'sd21;
8'd145:  coeff = 12'sd21;
8'd146:  coeff = 12'sd21;
8'd147:  coeff = 12'sd20;
8'd148:  coeff = 12'sd20;
8'd149:  coeff = 12'sd20;
8'd150:  coeff = 12'sd19;
8'd151:  coeff = 12'sd19;
8'd152:  coeff = 12'sd18;
8'd153:  coeff = 12'sd18;
8'd154:  coeff = 12'sd18;
8'd155:  coeff = 12'sd17;
8'd156:  coeff = 12'sd17;
8'd157:  coeff = 12'sd16;
8'd158:  coeff = 12'sd16;
8'd159:  coeff = 12'sd16;
8'd160:  coeff = 12'sd15;
8'd161:  coeff = 12'sd15;
8'd162:  coeff = 12'sd14;
8'd163:  coeff = 12'sd14;
8'd164:  coeff = 12'sd14;
8'd165:  coeff = 12'sd13;
8'd166:  coeff = 12'sd13;
8'd167:  coeff = 12'sd12;
8'd168:  coeff = 12'sd12;
8'd169:  coeff = 12'sd11;
8'd170:  coeff = 12'sd11;
8'd171:  coeff = 12'sd11;
8'd172:  coeff = 12'sd10;
8'd173:  coeff = 12'sd10;
8'd174:  coeff = 12'sd9;
8'd175:  coeff = 12'sd9;
8'd176:  coeff = 12'sd9;
8'd177:  coeff = 12'sd8;
8'd178:  coeff = 12'sd8;
8'd179:  coeff = 12'sd7;
8'd180:  coeff = 12'sd7;
8'd181:  coeff = 12'sd7;
8'd182:  coeff = 12'sd6;
8'd183:  coeff = 12'sd6;
8'd184:  coeff = 12'sd6;
8'd185:  coeff = 12'sd5;
8'd186:  coeff = 12'sd5;
8'd187:  coeff = 12'sd5;
8'd188:  coeff = 12'sd4;
8'd189:  coeff = 12'sd4;
8'd190:  coeff = 12'sd4;
8'd191:  coeff = 12'sd4;
8'd192:  coeff = 12'sd3;
8'd193:  coeff = 12'sd3;
8'd194:  coeff = 12'sd3;
8'd195:  coeff = 12'sd3;
8'd196:  coeff = 12'sd2;
8'd197:  coeff = 12'sd2;
8'd198:  coeff = 12'sd2;
8'd199:  coeff = 12'sd2;
8'd200:  coeff = 12'sd2;
```

```
        8'd201:  coeff = 12'sd1;
        8'd202:  coeff = 12'sd1;
        8'd203:  coeff = 12'sd1;
        8'd204:  coeff = 12'sd1;
        8'd205:  coeff = 12'sd1;
        8'd206:  coeff = 12'sd1;
        8'd207:  coeff = 12'sd0;
        8'd208:  coeff = 12'sd0;
        8'd209:  coeff = 12'sd0;
        8'd210:  coeff = 12'sd0;
        8'd211:  coeff = 12'sd0;
        8'd212:  coeff = 12'sd0;
        8'd213:  coeff = 12'sd0;
        8'd214:  coeff = 12'sd0;
        8'd215:  coeff = 12'sd0;
        8'd216:  coeff = 12'sd0;
        8'd217:  coeff = 12'sd0;
        8'd218:  coeff = 12'sd0;
        8'd219:  coeff = 12'sd0;
        8'd220:  coeff = 12'sd0;
        8'd221:  coeff = 12'sd0;
        8'd222:  coeff = 12'sd0;
        8'd223:  coeff = 12'sd0;
        8'd224:  coeff = 12'sd0;
        8'd225:  coeff = 12'sd0;
        8'd226:  coeff = -12'sd1;
        8'd227:  coeff = -12'sd1;
        8'd228:  coeff = -12'sd1;
        8'd229:  coeff = -12'sd1;
        8'd230:  coeff = -12'sd1;
        8'd231:  coeff = -12'sd1;
        8'd232:  coeff = -12'sd1;
        8'd233:  coeff = -12'sd1;
        8'd234:  coeff = -12'sd1;
        8'd235:  coeff = 12'sd0;
        8'd236:  coeff = 12'sd0;
        8'd237:  coeff = 12'sd0;
        8'd238:  coeff = 12'sd0;
        8'd239:  coeff = 12'sd0;
        8'd240:  coeff = 12'sd0;
        8'd241:  coeff = 12'sd0;
        8'd242:  coeff = 12'sd0;
        8'd243:  coeff = 12'sd0;
        8'd244:  coeff = 12'sd0;
        8'd245:  coeff = 12'sd0;
        8'd246:  coeff = 12'sd0;
        8'd247:  coeff = 12'sd0;
        8'd248:  coeff = 12'sd0;
        8'd249:  coeff = 12'sd0;
        8'd250:  coeff = 12'sd0;
     default: coeff = 12'hXXX;
    endcase
endmodule

module firlow385(
  input wire clock,reset,ready,
  input wire signed [17:0] x,
  output reg signed [17:0] y
);
        reg signed [17:0]big_array[250:0]; //change big_array[x:0] x to coeff #
        reg [7:0]index = 0;
        reg [7:0]offset = 0;
        wire signed [11:0]coeff;
        reg signed [29:0]accumulator = 0;
        reg go = 0;

  always @(posedge clock) begin
                if (reset) begin //reinitialize everything
                        offset <= 0;
                        index <= 0;
                        accumulator <= 0;
                        go <= 0;end
                else if (ready) begin //on ready insert x into big_array and reset index
                        big_array[offset] <= x;
                        index <= 0;
                        go <= 1;end
                else if (go == 1) begin //while accumulator is accumulating
                        if (index == 8'd250) begin //change 8'dx x to coeff #
                                index <= 0;
                                go <= 0;
                                offset <= offset + 1;
                                y <= accumulator[29:12]<<<1;
                                accumulator <= 0;
                                end
                        else begin //accumulate procedure
                                index <= index + 1;
                                accumulator <= accumulator + coeff*big_array[offset - index];end
                        end
                else;
  end
  firlow385coeffs coeffs(index,coeff);
endmodule
```

```
module firlow385coeffs(
  input wire [7:0] index,
  output reg signed [11:0] coeff
);
  // tools will turn this into a 31x10 ROM
  always @(index)
    case (index)
         8'd0:  coeff = 12'sd0;
         8'd1:  coeff = 12'sd0;
         8'd2:  coeff = 12'sd0;
         8'd3:  coeff = 12'sd0;
         8'd4:  coeff = 12'sd0;
         8'd5:  coeff = 12'sd0;
         8'd6:  coeff = 12'sd0;
         8'd7:  coeff = 12'sd0;
         8'd8:  coeff = 12'sd0;
         8'd9:  coeff = 12'sd0;
         8'd10:  coeff = 12'sd0;
         8'd11:  coeff = 12'sd0;
         8'd12:  coeff = 12'sd0;
         8'd13:  coeff = 12'sd0;
         8'd14:  coeff = 12'sd0;
         8'd15:  coeff = 12'sd0;
         8'd16:  coeff = 12'sd0;
         8'd17:  coeff = -12'sd1;
         8'd18:  coeff = -12'sd1;
         8'd19:  coeff = -12'sd1;
         8'd20:  coeff = -12'sd1;
         8'd21:  coeff = -12'sd1;
         8'd22:  coeff = -12'sd1;
         8'd23:  coeff = -12'sd1;
         8'd24:  coeff = -12'sd1;
         8'd25:  coeff = -12'sd1;
         8'd26:  coeff = -12'sd1;
         8'd27:  coeff = -12'sd1;
         8'd28:  coeff = -12'sd1;
         8'd29:  coeff = -12'sd1;
         8'd30:  coeff = -12'sd1;
         8'd31:  coeff = -12'sd1;
         8'd32:  coeff = -12'sd2;
         8'd33:  coeff = -12'sd2;
         8'd34:  coeff = -12'sd2;
         8'd35:  coeff = -12'sd2;
         8'd36:  coeff = -12'sd2;
         8'd37:  coeff = -12'sd2;
         8'd38:  coeff = -12'sd2;
         8'd39:  coeff = -12'sd2;
         8'd40:  coeff = -12'sd2;
         8'd41:  coeff = -12'sd2;
         8'd42:  coeff = -12'sd2;
         8'd43:  coeff = -12'sd2;
         8'd44:  coeff = -12'sd2;
         8'd45:  coeff = -12'sd2;
         8'd46:  coeff = -12'sd2;
         8'd47:  coeff = -12'sd2;
         8'd48:  coeff = -12'sd2;
         8'd49:  coeff = -12'sd2;
         8'd50:  coeff = -12'sd2;
         8'd51:  coeff = -12'sd2;
         8'd52:  coeff = -12'sd2;
         8'd53:  coeff = -12'sd2;
         8'd54:  coeff = -12'sd2;
         8'd55:  coeff = -12'sd2;
         8'd56:  coeff = -12'sd1;
         8'd57:  coeff = -12'sd1;
         8'd58:  coeff = -12'sd1;
         8'd59:  coeff = -12'sd1;
         8'd60:  coeff = -12'sd1;
         8'd61:  coeff = 12'sd0;
         8'd62:  coeff = 12'sd0;
         8'd63:  coeff = 12'sd0;
         8'd64:  coeff = 12'sd0;
         8'd65:  coeff = 12'sd1;
         8'd66:  coeff = 12'sd1;
         8'd67:  coeff = 12'sd1;
         8'd68:  coeff = 12'sd2;
         8'd69:  coeff = 12'sd2;
         8'd70:  coeff = 12'sd3;
         8'd71:  coeff = 12'sd3;
         8'd72:  coeff = 12'sd4;
         8'd73:  coeff = 12'sd4;
         8'd74:  coeff = 12'sd5;
         8'd75:  coeff = 12'sd5;
         8'd76:  coeff = 12'sd6;
         8'd77:  coeff = 12'sd6;
         8'd78:  coeff = 12'sd7;
         8'd79:  coeff = 12'sd8;
         8'd80:  coeff = 12'sd8;
         8'd81:  coeff = 12'sd9;
         8'd82:  coeff = 12'sd9;
         8'd83:  coeff = 12'sd10;
```

```
8'd84:  coeff = 12'sd11;
8'd85:  coeff = 12'sd12;
8'd86:  coeff = 12'sd12;
8'd87:  coeff = 12'sd13;
8'd88:  coeff = 12'sd14;
8'd89:  coeff = 12'sd14;
8'd90:  coeff = 12'sd15;
8'd91:  coeff = 12'sd16;
8'd92:  coeff = 12'sd17;
8'd93:  coeff = 12'sd17;
8'd94:  coeff = 12'sd18;
8'd95:  coeff = 12'sd19;
8'd96:  coeff = 12'sd20;
8'd97:  coeff = 12'sd20;
8'd98:  coeff = 12'sd21;
8'd99:  coeff = 12'sd22;
8'd100:  coeff = 12'sd23;
8'd101:  coeff = 12'sd23;
8'd102:  coeff = 12'sd24;
8'd103:  coeff = 12'sd25;
8'd104:  coeff = 12'sd25;
8'd105:  coeff = 12'sd26;
8'd106:  coeff = 12'sd27;
8'd107:  coeff = 12'sd27;
8'd108:  coeff = 12'sd28;
8'd109:  coeff = 12'sd28;
8'd110:  coeff = 12'sd29;
8'd111:  coeff = 12'sd29;
8'd112:  coeff = 12'sd30;
8'd113:  coeff = 12'sd30;
8'd114:  coeff = 12'sd31;
8'd115:  coeff = 12'sd31;
8'd116:  coeff = 12'sd31;
8'd117:  coeff = 12'sd32;
8'd118:  coeff = 12'sd32;
8'd119:  coeff = 12'sd32;
8'd120:  coeff = 12'sd32;
8'd121:  coeff = 12'sd32;
8'd122:  coeff = 12'sd33;
8'd123:  coeff = 12'sd33;
8'd124:  coeff = 12'sd33;
8'd125:  coeff = 12'sd33;
8'd126:  coeff = 12'sd33;
8'd127:  coeff = 12'sd33;
8'd128:  coeff = 12'sd33;
8'd129:  coeff = 12'sd32;
8'd130:  coeff = 12'sd32;
8'd131:  coeff = 12'sd32;
8'd132:  coeff = 12'sd32;
8'd133:  coeff = 12'sd32;
8'd134:  coeff = 12'sd31;
8'd135:  coeff = 12'sd31;
8'd136:  coeff = 12'sd31;
8'd137:  coeff = 12'sd30;
8'd138:  coeff = 12'sd30;
8'd139:  coeff = 12'sd29;
8'd140:  coeff = 12'sd29;
8'd141:  coeff = 12'sd28;
8'd142:  coeff = 12'sd28;
8'd143:  coeff = 12'sd27;
8'd144:  coeff = 12'sd27;
8'd145:  coeff = 12'sd26;
8'd146:  coeff = 12'sd25;
8'd147:  coeff = 12'sd25;
8'd148:  coeff = 12'sd24;
8'd149:  coeff = 12'sd23;
8'd150:  coeff = 12'sd23;
8'd151:  coeff = 12'sd22;
8'd152:  coeff = 12'sd21;
8'd153:  coeff = 12'sd20;
8'd154:  coeff = 12'sd20;
8'd155:  coeff = 12'sd19;
8'd156:  coeff = 12'sd18;
8'd157:  coeff = 12'sd17;
8'd158:  coeff = 12'sd17;
8'd159:  coeff = 12'sd16;
8'd160:  coeff = 12'sd15;
8'd161:  coeff = 12'sd14;
8'd162:  coeff = 12'sd14;
8'd163:  coeff = 12'sd13;
8'd164:  coeff = 12'sd12;
8'd165:  coeff = 12'sd12;
8'd166:  coeff = 12'sd11;
8'd167:  coeff = 12'sd10;
8'd168:  coeff = 12'sd9;
8'd169:  coeff = 12'sd9;
8'd170:  coeff = 12'sd8;
8'd171:  coeff = 12'sd8;
8'd172:  coeff = 12'sd7;
8'd173:  coeff = 12'sd6;
8'd174:  coeff = 12'sd6;
8'd175:  coeff = 12'sd5;
```

```
        8'd176:  coeff = 12'sd5;
        8'd177:  coeff = 12'sd4;
        8'd178:  coeff = 12'sd4;
        8'd179:  coeff = 12'sd3;
        8'd180:  coeff = 12'sd3;
        8'd181:  coeff = 12'sd2;
        8'd182:  coeff = 12'sd2;
        8'd183:  coeff = 12'sd1;
        8'd184:  coeff = 12'sd1;
        8'd185:  coeff = 12'sd1;
        8'd186:  coeff = 12'sd0;
        8'd187:  coeff = 12'sd0;
        8'd188:  coeff = 12'sd0;
        8'd189:  coeff = 12'sd0;
        8'd190:  coeff = -12'sd1;
        8'd191:  coeff = -12'sd1;
        8'd192:  coeff = -12'sd1;
        8'd193:  coeff = -12'sd1;
        8'd194:  coeff = -12'sd1;
        8'd195:  coeff = -12'sd2;
        8'd196:  coeff = -12'sd2;
        8'd197:  coeff = -12'sd2;
        8'd198:  coeff = -12'sd2;
        8'd199:  coeff = -12'sd2;
        8'd200:  coeff = -12'sd2;
        8'd201:  coeff = -12'sd2;
        8'd202:  coeff = -12'sd2;
        8'd203:  coeff = -12'sd2;
        8'd204:  coeff = -12'sd2;
        8'd205:  coeff = -12'sd2;
        8'd206:  coeff = -12'sd2;
        8'd207:  coeff = -12'sd2;
        8'd208:  coeff = -12'sd2;
        8'd209:  coeff = -12'sd2;
        8'd210:  coeff = -12'sd2;
        8'd211:  coeff = -12'sd2;
        8'd212:  coeff = -12'sd2;
        8'd213:  coeff = -12'sd2;
        8'd214:  coeff = -12'sd2;
        8'd215:  coeff = -12'sd2;
        8'd216:  coeff = -12'sd2;
        8'd217:  coeff = -12'sd2;
        8'd218:  coeff = -12'sd2;
        8'd219:  coeff = -12'sd1;
        8'd220:  coeff = -12'sd1;
        8'd221:  coeff = -12'sd1;
        8'd222:  coeff = -12'sd1;
        8'd223:  coeff = -12'sd1;
        8'd224:  coeff = -12'sd1;
        8'd225:  coeff = -12'sd1;
        8'd226:  coeff = -12'sd1;
        8'd227:  coeff = -12'sd1;
        8'd228:  coeff = -12'sd1;
        8'd229:  coeff = -12'sd1;
        8'd230:  coeff = -12'sd1;
        8'd231:  coeff = -12'sd1;
        8'd232:  coeff = -12'sd1;
        8'd233:  coeff = -12'sd1;
        8'd234:  coeff = 12'sd0;
        8'd235:  coeff = 12'sd0;
        8'd236:  coeff = 12'sd0;
        8'd237:  coeff = 12'sd0;
        8'd238:  coeff = 12'sd0;
        8'd239:  coeff = 12'sd0;
        8'd240:  coeff = 12'sd0;
        8'd241:  coeff = 12'sd0;
        8'd242:  coeff = 12'sd0;
        8'd243:  coeff = 12'sd0;
        8'd244:  coeff = 12'sd0;
        8'd245:  coeff = 12'sd0;
        8'd246:  coeff = 12'sd0;
        8'd247:  coeff = 12'sd0;
        8'd248:  coeff = 12'sd0;
        8'd249:  coeff = 12'sd0;
        8'd250:  coeff = 12'sd0;
      default: coeff = 12'hXXX;
    endcase
endmodule

module firlow450(
  input wire clock,reset,ready,
  input wire signed [17:0] x,
  output reg signed [17:0] y
);
        reg signed [17:0]big_array[250:0]; //change big_array[x:0] x to coeff #
        reg [7:0]index = 0;
        reg [7:0]offset = 0;
        wire signed [11:0]coeff;
        reg signed [29:0]accumulator = 0;
        reg go = 0;

  always @(posedge clock) begin
```

```
                if (reset) begin //reinitialize everything
                        offset <= 0;
                        index <= 0;
                        accumulator <= 0;
                        go <= 0;end
                else if (ready) begin //on ready insert x into big_array and reset index
                        big_array[offset] <= x;
                        index <= 0;
                        go <= 1;end
                else if (go == 1) begin //while accumulator is accumulating
                        if (index == 8'd250) begin //change 8'dx x to coeff #
                                index <= 0;
                                go <= 0;
                                offset <= offset + 1;
                                y <= accumulator[29:12]<<<1;
                                accumulator <= 0;
                                end
                        else begin //accumulate procedure
                                index <= index + 1;
                                accumulator <= accumulator + coeff*big_array[offset - index];end
                        end
                else;
  end
  firlow450coeffs coeffs(index,coeff);
endmodule

module firlow450coeffs(
  input wire [7:0] index,
  output reg signed [11:0] coeff
);
  // tools will turn this into a 31x10 ROM
  always @(index)
    case (index)
        8'd0:  coeff = 12'sd0;
        8'd1:  coeff = 12'sd0;
        8'd2:  coeff = 12'sd0;
        8'd3:  coeff = 12'sd0;
        8'd4:  coeff = 12'sd0;
        8'd5:  coeff = 12'sd0;
        8'd6:  coeff = 12'sd0;
        8'd7:  coeff = 12'sd0;
        8'd8:  coeff = 12'sd0;
        8'd9:  coeff = 12'sd0;
        8'd10:  coeff = 12'sd0;
        8'd11:  coeff = 12'sd0;
        8'd12:  coeff = 12'sd0;
        8'd13:  coeff = 12'sd0;
        8'd14:  coeff = 12'sd0;
        8'd15:  coeff = 12'sd0;
        8'd16:  coeff = 12'sd0;
        8'd17:  coeff = 12'sd0;
        8'd18:  coeff = 12'sd0;
        8'd19:  coeff = 12'sd0;
        8'd20:  coeff = 12'sd0;
        8'd21:  coeff = 12'sd0;
        8'd22:  coeff = 12'sd0;
        8'd23:  coeff = 12'sd0;
        8'd24:  coeff = 12'sd0;
        8'd25:  coeff = 12'sd0;
        8'd26:  coeff = -12'sd1;
        8'd27:  coeff = -12'sd1;
        8'd28:  coeff = -12'sd1;
        8'd29:  coeff = -12'sd1;
        8'd30:  coeff = -12'sd1;
        8'd31:  coeff = -12'sd1;
        8'd32:  coeff = -12'sd1;
        8'd33:  coeff = -12'sd1;
        8'd34:  coeff = -12'sd1;
        8'd35:  coeff = -12'sd1;
        8'd36:  coeff = -12'sd2;
        8'd37:  coeff = -12'sd2;
        8'd38:  coeff = -12'sd2;
        8'd39:  coeff = -12'sd2;
        8'd40:  coeff = -12'sd2;
        8'd41:  coeff = -12'sd2;
        8'd42:  coeff = -12'sd2;
        8'd43:  coeff = -12'sd3;
        8'd44:  coeff = -12'sd3;
        8'd45:  coeff = -12'sd3;
        8'd46:  coeff = -12'sd3;
        8'd47:  coeff = -12'sd3;
        8'd48:  coeff = -12'sd3;
        8'd49:  coeff = -12'sd3;
        8'd50:  coeff = -12'sd3;
        8'd51:  coeff = -12'sd3;
        8'd52:  coeff = -12'sd3;
        8'd53:  coeff = -12'sd3;
        8'd54:  coeff = -12'sd3;
        8'd55:  coeff = -12'sd4;
        8'd56:  coeff = -12'sd3;
        8'd57:  coeff = -12'sd3;
        8'd58:  coeff = -12'sd3;
```

```
8'd59:  coeff = -12'sd3;
8'd60:  coeff = -12'sd3;
8'd61:  coeff = -12'sd3;
8'd62:  coeff = -12'sd3;
8'd63:  coeff = -12'sd3;
8'd64:  coeff = -12'sd3;
8'd65:  coeff = -12'sd2;
8'd66:  coeff = -12'sd2;
8'd67:  coeff = -12'sd2;
8'd68:  coeff = -12'sd1;
8'd69:  coeff = -12'sd1;
8'd70:  coeff = -12'sd1;
8'd71:  coeff = 12'sd0;
8'd72:  coeff = 12'sd0;
8'd73:  coeff = 12'sd1;
8'd74:  coeff = 12'sd1;
8'd75:  coeff = 12'sd2;
8'd76:  coeff = 12'sd2;
8'd77:  coeff = 12'sd3;
8'd78:  coeff = 12'sd4;
8'd79:  coeff = 12'sd4;
8'd80:  coeff = 12'sd5;
8'd81:  coeff = 12'sd6;
8'd82:  coeff = 12'sd7;
8'd83:  coeff = 12'sd7;
8'd84:  coeff = 12'sd8;
8'd85:  coeff = 12'sd9;
8'd86:  coeff = 12'sd10;
8'd87:  coeff = 12'sd11;
8'd88:  coeff = 12'sd12;
8'd89:  coeff = 12'sd13;
8'd90:  coeff = 12'sd14;
8'd91:  coeff = 12'sd15;
8'd92:  coeff = 12'sd16;
8'd93:  coeff = 12'sd17;
8'd94:  coeff = 12'sd18;
8'd95:  coeff = 12'sd19;
8'd96:  coeff = 12'sd20;
8'd97:  coeff = 12'sd21;
8'd98:  coeff = 12'sd22;
8'd99:  coeff = 12'sd23;
8'd100:  coeff = 12'sd24;
8'd101:  coeff = 12'sd25;
8'd102:  coeff = 12'sd26;
8'd103:  coeff = 12'sd27;
8'd104:  coeff = 12'sd27;
8'd105:  coeff = 12'sd28;
8'd106:  coeff = 12'sd29;
8'd107:  coeff = 12'sd30;
8'd108:  coeff = 12'sd31;
8'd109:  coeff = 12'sd32;
8'd110:  coeff = 12'sd32;
8'd111:  coeff = 12'sd33;
8'd112:  coeff = 12'sd34;
8'd113:  coeff = 12'sd34;
8'd114:  coeff = 12'sd35;
8'd115:  coeff = 12'sd36;
8'd116:  coeff = 12'sd36;
8'd117:  coeff = 12'sd37;
8'd118:  coeff = 12'sd37;
8'd119:  coeff = 12'sd37;
8'd120:  coeff = 12'sd38;
8'd121:  coeff = 12'sd38;
8'd122:  coeff = 12'sd38;
8'd123:  coeff = 12'sd38;
8'd124:  coeff = 12'sd38;
8'd125:  coeff = 12'sd38;
8'd126:  coeff = 12'sd38;
8'd127:  coeff = 12'sd38;
8'd128:  coeff = 12'sd38;
8'd129:  coeff = 12'sd38;
8'd130:  coeff = 12'sd38;
8'd131:  coeff = 12'sd37;
8'd132:  coeff = 12'sd37;
8'd133:  coeff = 12'sd37;
8'd134:  coeff = 12'sd36;
8'd135:  coeff = 12'sd36;
8'd136:  coeff = 12'sd35;
8'd137:  coeff = 12'sd34;
8'd138:  coeff = 12'sd34;
8'd139:  coeff = 12'sd33;
8'd140:  coeff = 12'sd32;
8'd141:  coeff = 12'sd32;
8'd142:  coeff = 12'sd31;
8'd143:  coeff = 12'sd30;
8'd144:  coeff = 12'sd29;
8'd145:  coeff = 12'sd28;
8'd146:  coeff = 12'sd27;
8'd147:  coeff = 12'sd27;
8'd148:  coeff = 12'sd26;
8'd149:  coeff = 12'sd25;
8'd150:  coeff = 12'sd24;
```

```
8'd151:  coeff = 12'sd23;
8'd152:  coeff = 12'sd22;
8'd153:  coeff = 12'sd21;
8'd154:  coeff = 12'sd20;
8'd155:  coeff = 12'sd19;
8'd156:  coeff = 12'sd18;
8'd157:  coeff = 12'sd17;
8'd158:  coeff = 12'sd16;
8'd159:  coeff = 12'sd15;
8'd160:  coeff = 12'sd14;
8'd161:  coeff = 12'sd13;
8'd162:  coeff = 12'sd12;
8'd163:  coeff = 12'sd11;
8'd164:  coeff = 12'sd10;
8'd165:  coeff = 12'sd9;
8'd166:  coeff = 12'sd8;
8'd167:  coeff = 12'sd7;
8'd168:  coeff = 12'sd7;
8'd169:  coeff = 12'sd6;
8'd170:  coeff = 12'sd5;
8'd171:  coeff = 12'sd4;
8'd172:  coeff = 12'sd4;
8'd173:  coeff = 12'sd3;
8'd174:  coeff = 12'sd2;
8'd175:  coeff = 12'sd2;
8'd176:  coeff = 12'sd1;
8'd177:  coeff = 12'sd1;
8'd178:  coeff = 12'sd0;
8'd179:  coeff = 12'sd0;
8'd180:  coeff = -12'sd1;
8'd181:  coeff = -12'sd1;
8'd182:  coeff = -12'sd1;
8'd183:  coeff = -12'sd2;
8'd184:  coeff = -12'sd2;
8'd185:  coeff = -12'sd2;
8'd186:  coeff = -12'sd3;
8'd187:  coeff = -12'sd3;
8'd188:  coeff = -12'sd3;
8'd189:  coeff = -12'sd3;
8'd190:  coeff = -12'sd3;
8'd191:  coeff = -12'sd3;
8'd192:  coeff = -12'sd3;
8'd193:  coeff = -12'sd3;
8'd194:  coeff = -12'sd3;
8'd195:  coeff = -12'sd4;
8'd196:  coeff = -12'sd3;
8'd197:  coeff = -12'sd3;
8'd198:  coeff = -12'sd3;
8'd199:  coeff = -12'sd3;
8'd200:  coeff = -12'sd3;
8'd201:  coeff = -12'sd3;
8'd202:  coeff = -12'sd3;
8'd203:  coeff = -12'sd3;
8'd204:  coeff = -12'sd3;
8'd205:  coeff = -12'sd3;
8'd206:  coeff = -12'sd3;
8'd207:  coeff = -12'sd3;
8'd208:  coeff = -12'sd2;
8'd209:  coeff = -12'sd2;
8'd210:  coeff = -12'sd2;
8'd211:  coeff = -12'sd2;
8'd212:  coeff = -12'sd2;
8'd213:  coeff = -12'sd2;
8'd214:  coeff = -12'sd2;
8'd215:  coeff = -12'sd1;
8'd216:  coeff = -12'sd1;
8'd217:  coeff = -12'sd1;
8'd218:  coeff = -12'sd1;
8'd219:  coeff = -12'sd1;
8'd220:  coeff = -12'sd1;
8'd221:  coeff = -12'sd1;
8'd222:  coeff = -12'sd1;
8'd223:  coeff = -12'sd1;
8'd224:  coeff = -12'sd1;
8'd225:  coeff = 12'sd0;
8'd226:  coeff = 12'sd0;
8'd227:  coeff = 12'sd0;
8'd228:  coeff = 12'sd0;
8'd229:  coeff = 12'sd0;
8'd230:  coeff = 12'sd0;
8'd231:  coeff = 12'sd0;
8'd232:  coeff = 12'sd0;
8'd233:  coeff = 12'sd0;
8'd234:  coeff = 12'sd0;
8'd235:  coeff = 12'sd0;
8'd236:  coeff = 12'sd0;
8'd237:  coeff = 12'sd0;
8'd238:  coeff = 12'sd0;
8'd239:  coeff = 12'sd0;
8'd240:  coeff = 12'sd0;
8'd241:  coeff = 12'sd0;
8'd242:  coeff = 12'sd0;
```

```
        8'd243:  coeff = 12'sd0;
        8'd244:  coeff = 12'sd0;
        8'd245:  coeff = 12'sd0;
        8'd246:  coeff = 12'sd0;
        8'd247:  coeff = 12'sd0;
        8'd248:  coeff = 12'sd0;
        8'd249:  coeff = 12'sd0;
        8'd250:  coeff = 12'sd0;
      default: coeff = 12'hXXX;
    endcase
endmodule

module firlow530(
  input wire clock,reset,ready,
  input wire signed [17:0] x,
  output reg signed [17:0] y
);
        reg signed [17:0]big_array[250:0]; //change big_array[x:0] x to coeff #
        reg [7:0]index = 0;
        reg [7:0]offset = 0;
        wire signed [11:0]coeff;
        reg signed [29:0]accumulator = 0;
        reg go = 0;

  always @(posedge clock) begin
                if (reset) begin //reinitialize everything
                        offset <= 0;
                        index <= 0;
                        accumulator <= 0;
                        go <= 0;end
                else if (ready) begin //on ready insert x into big_array and reset index
                        big_array[offset] <= x;
                        index <= 0;
                        go <= 1;end
                else if (go == 1) begin //while accumulator is accumulating
                        if (index == 8'd250) begin //change 8'dx x to coeff #
                                index <= 0;
                                go <= 0;
                                offset <= offset + 1;
                                y <= accumulator[29:12] <<< 1;
                                accumulator <= 0;
                                end
                        else begin //accumulate procedure
                                index <= index + 1;
                                accumulator <= accumulator + coeff*big_array[offset - index];end
                        end
                else;
  end
  firlow530coeffs coeffs(index,coeff);
endmodule

module firlow530coeffs(
  input wire [7:0] index,
  output reg signed [11:0] coeff
);
  // tools will turn this into a 31x10 ROM
  always @(index)
    case (index)
        8'd0:  coeff = 12'sd0;
        8'd1:  coeff = 12'sd0;
        8'd2:  coeff = 12'sd0;
        8'd3:  coeff = 12'sd0;
        8'd4:  coeff = 12'sd0;
        8'd5:  coeff = 12'sd0;
        8'd6:  coeff = 12'sd0;
        8'd7:  coeff = 12'sd0;
        8'd8:  coeff = 12'sd0;
        8'd9:  coeff = 12'sd1;
        8'd10:  coeff = 12'sd1;
        8'd11:  coeff = 12'sd1;
        8'd12:  coeff = 12'sd1;
        8'd13:  coeff = 12'sd1;
        8'd14:  coeff = 12'sd1;
        8'd15:  coeff = 12'sd1;
        8'd16:  coeff = 12'sd1;
        8'd17:  coeff = 12'sd1;
        8'd18:  coeff = 12'sd1;
        8'd19:  coeff = 12'sd1;
        8'd20:  coeff = 12'sd1;
        8'd21:  coeff = 12'sd1;
        8'd22:  coeff = 12'sd1;
        8'd23:  coeff = 12'sd1;
        8'd24:  coeff = 12'sd1;
        8'd25:  coeff = 12'sd1;
        8'd26:  coeff = 12'sd1;
        8'd27:  coeff = 12'sd1;
        8'd28:  coeff = 12'sd1;
        8'd29:  coeff = 12'sd0;
        8'd30:  coeff = 12'sd0;
        8'd31:  coeff = 12'sd0;
        8'd32:  coeff = 12'sd0;
        8'd33:  coeff = 12'sd0;
```

```
8'd34:  coeff = 12'sd0;
8'd35:  coeff = 12'sd0;
8'd36:  coeff = 12'sd0;
8'd37:  coeff = 12'sd0;
8'd38:  coeff = -12'sd1;
8'd39:  coeff = -12'sd1;
8'd40:  coeff = -12'sd1;
8'd41:  coeff = -12'sd1;
8'd42:  coeff = -12'sd1;
8'd43:  coeff = -12'sd1;
8'd44:  coeff = -12'sd2;
8'd45:  coeff = -12'sd2;
8'd46:  coeff = -12'sd2;
8'd47:  coeff = -12'sd2;
8'd48:  coeff = -12'sd3;
8'd49:  coeff = -12'sd3;
8'd50:  coeff = -12'sd3;
8'd51:  coeff = -12'sd3;
8'd52:  coeff = -12'sd4;
8'd53:  coeff = -12'sd4;
8'd54:  coeff = -12'sd4;
8'd55:  coeff = -12'sd4;
8'd56:  coeff = -12'sd4;
8'd57:  coeff = -12'sd5;
8'd58:  coeff = -12'sd5;
8'd59:  coeff = -12'sd5;
8'd60:  coeff = -12'sd5;
8'd61:  coeff = -12'sd5;
8'd62:  coeff = -12'sd5;
8'd63:  coeff = -12'sd5;
8'd64:  coeff = -12'sd5;
8'd65:  coeff = -12'sd5;
8'd66:  coeff = -12'sd5;
8'd67:  coeff = -12'sd5;
8'd68:  coeff = -12'sd5;
8'd69:  coeff = -12'sd5;
8'd70:  coeff = -12'sd5;
8'd71:  coeff = -12'sd4;
8'd72:  coeff = -12'sd4;
8'd73:  coeff = -12'sd4;
8'd74:  coeff = -12'sd3;
8'd75:  coeff = -12'sd3;
8'd76:  coeff = -12'sd2;
8'd77:  coeff = -12'sd2;
8'd78:  coeff = -12'sd1;
8'd79:  coeff = -12'sd1;
8'd80:  coeff = 12'sd0;
8'd81:  coeff = 12'sd1;
8'd82:  coeff = 12'sd2;
8'd83:  coeff = 12'sd3;
8'd84:  coeff = 12'sd4;
8'd85:  coeff = 12'sd5;
8'd86:  coeff = 12'sd6;
8'd87:  coeff = 12'sd7;
8'd88:  coeff = 12'sd8;
8'd89:  coeff = 12'sd9;
8'd90:  coeff = 12'sd10;
8'd91:  coeff = 12'sd11;
8'd92:  coeff = 12'sd13;
8'd93:  coeff = 12'sd14;
8'd94:  coeff = 12'sd15;
8'd95:  coeff = 12'sd17;
8'd96:  coeff = 12'sd18;
8'd97:  coeff = 12'sd19;
8'd98:  coeff = 12'sd21;
8'd99:  coeff = 12'sd22;
8'd100:  coeff = 12'sd23;
8'd101:  coeff = 12'sd25;
8'd102:  coeff = 12'sd26;
8'd103:  coeff = 12'sd28;
8'd104:  coeff = 12'sd29;
8'd105:  coeff = 12'sd30;
8'd106:  coeff = 12'sd31;
8'd107:  coeff = 12'sd33;
8'd108:  coeff = 12'sd34;
8'd109:  coeff = 12'sd35;
8'd110:  coeff = 12'sd36;
8'd111:  coeff = 12'sd37;
8'd112:  coeff = 12'sd38;
8'd113:  coeff = 12'sd39;
8'd114:  coeff = 12'sd40;
8'd115:  coeff = 12'sd41;
8'd116:  coeff = 12'sd42;
8'd117:  coeff = 12'sd42;
8'd118:  coeff = 12'sd43;
8'd119:  coeff = 12'sd44;
8'd120:  coeff = 12'sd44;
8'd121:  coeff = 12'sd44;
8'd122:  coeff = 12'sd45;
8'd123:  coeff = 12'sd45;
8'd124:  coeff = 12'sd45;
8'd125:  coeff = 12'sd45;
```

```
8'd126:  coeff = 12'sd45;
8'd127:  coeff = 12'sd45;
8'd128:  coeff = 12'sd45;
8'd129:  coeff = 12'sd44;
8'd130:  coeff = 12'sd44;
8'd131:  coeff = 12'sd44;
8'd132:  coeff = 12'sd43;
8'd133:  coeff = 12'sd42;
8'd134:  coeff = 12'sd42;
8'd135:  coeff = 12'sd41;
8'd136:  coeff = 12'sd40;
8'd137:  coeff = 12'sd39;
8'd138:  coeff = 12'sd38;
8'd139:  coeff = 12'sd37;
8'd140:  coeff = 12'sd36;
8'd141:  coeff = 12'sd35;
8'd142:  coeff = 12'sd34;
8'd143:  coeff = 12'sd33;
8'd144:  coeff = 12'sd31;
8'd145:  coeff = 12'sd30;
8'd146:  coeff = 12'sd29;
8'd147:  coeff = 12'sd28;
8'd148:  coeff = 12'sd26;
8'd149:  coeff = 12'sd25;
8'd150:  coeff = 12'sd23;
8'd151:  coeff = 12'sd22;
8'd152:  coeff = 12'sd21;
8'd153:  coeff = 12'sd19;
8'd154:  coeff = 12'sd18;
8'd155:  coeff = 12'sd17;
8'd156:  coeff = 12'sd15;
8'd157:  coeff = 12'sd14;
8'd158:  coeff = 12'sd13;
8'd159:  coeff = 12'sd11;
8'd160:  coeff = 12'sd10;
8'd161:  coeff = 12'sd9;
8'd162:  coeff = 12'sd8;
8'd163:  coeff = 12'sd7;
8'd164:  coeff = 12'sd6;
8'd165:  coeff = 12'sd5;
8'd166:  coeff = 12'sd4;
8'd167:  coeff = 12'sd3;
8'd168:  coeff = 12'sd2;
8'd169:  coeff = 12'sd1;
8'd170:  coeff = 12'sd0;
8'd171:  coeff = -12'sd1;
8'd172:  coeff = -12'sd1;
8'd173:  coeff = -12'sd2;
8'd174:  coeff = -12'sd2;
8'd175:  coeff = -12'sd3;
8'd176:  coeff = -12'sd3;
8'd177:  coeff = -12'sd4;
8'd178:  coeff = -12'sd4;
8'd179:  coeff = -12'sd4;
8'd180:  coeff = -12'sd5;
8'd181:  coeff = -12'sd5;
8'd182:  coeff = -12'sd5;
8'd183:  coeff = -12'sd5;
8'd184:  coeff = -12'sd5;
8'd185:  coeff = -12'sd5;
8'd186:  coeff = -12'sd5;
8'd187:  coeff = -12'sd5;
8'd188:  coeff = -12'sd5;
8'd189:  coeff = -12'sd5;
8'd190:  coeff = -12'sd5;
8'd191:  coeff = -12'sd5;
8'd192:  coeff = -12'sd5;
8'd193:  coeff = -12'sd5;
8'd194:  coeff = -12'sd4;
8'd195:  coeff = -12'sd4;
8'd196:  coeff = -12'sd4;
8'd197:  coeff = -12'sd4;
8'd198:  coeff = -12'sd4;
8'd199:  coeff = -12'sd3;
8'd200:  coeff = -12'sd3;
8'd201:  coeff = -12'sd3;
8'd202:  coeff = -12'sd3;
8'd203:  coeff = -12'sd2;
8'd204:  coeff = -12'sd2;
8'd205:  coeff = -12'sd2;
8'd206:  coeff = -12'sd2;
8'd207:  coeff = -12'sd1;
8'd208:  coeff = -12'sd1;
8'd209:  coeff = -12'sd1;
8'd210:  coeff = -12'sd1;
8'd211:  coeff = -12'sd1;
8'd212:  coeff = -12'sd1;
8'd213:  coeff = 12'sd0;
8'd214:  coeff = 12'sd0;
8'd215:  coeff = 12'sd0;
8'd216:  coeff = 12'sd0;
8'd217:  coeff = 12'sd0;
```

```
        8'd218:  coeff = 12'sd0;
        8'd219:  coeff = 12'sd0;
        8'd220:  coeff = 12'sd0;
        8'd221:  coeff = 12'sd0;
        8'd222:  coeff = 12'sd1;
        8'd223:  coeff = 12'sd1;
        8'd224:  coeff = 12'sd1;
        8'd225:  coeff = 12'sd1;
        8'd226:  coeff = 12'sd1;
        8'd227:  coeff = 12'sd1;
        8'd228:  coeff = 12'sd1;
        8'd229:  coeff = 12'sd1;
        8'd230:  coeff = 12'sd1;
        8'd231:  coeff = 12'sd1;
        8'd232:  coeff = 12'sd1;
        8'd233:  coeff = 12'sd1;
        8'd234:  coeff = 12'sd1;
        8'd235:  coeff = 12'sd1;
        8'd236:  coeff = 12'sd1;
        8'd237:  coeff = 12'sd1;
        8'd238:  coeff = 12'sd1;
        8'd239:  coeff = 12'sd1;
        8'd240:  coeff = 12'sd1;
        8'd241:  coeff = 12'sd1;
        8'd242:  coeff = 12'sd0;
        8'd243:  coeff = 12'sd0;
        8'd244:  coeff = 12'sd0;
        8'd245:  coeff = 12'sd0;
        8'd246:  coeff = 12'sd0;
        8'd247:  coeff = 12'sd0;
        8'd248:  coeff = 12'sd0;
        8'd249:  coeff = 12'sd0;
        8'd250:  coeff = 12'sd0;
      default: coeff = 12'hXXX;
    endcase
endmodule

module firlow620(
  input wire clock,reset,ready,
  input wire signed [17:0] x,
  output reg signed [17:0] y
);
        reg signed [17:0]big_array[250:0]; //change big_array[x:0] x to coeff #
        reg [7:0]index = 0;
        reg [7:0]offset = 0;
        wire signed [11:0]coeff;
        reg signed [29:0]accumulator = 0;
        reg go = 0;

  always @(posedge clock) begin
                if (reset) begin //reinitialize everything
                        offset <= 0;
                        index <= 0;
                        accumulator <= 0;
                        go <= 0;end
                else if (ready) begin //on ready insert x into big_array and reset index
                        big_array[offset] <= x;
                        index <= 0;
                        go <= 1;end
                else if (go == 1) begin //while accumulator is accumulating
                        if (index == 8'd250) begin //change 8'dx x to coeff #
                                index <= 0;
                                go <= 0;
                                offset <= offset + 1;
                                y <= accumulator[29:12] <<< 1;
                                accumulator <= 0;
                                end
                        else begin //accumulate procedure
                                index <= index + 1;
                                accumulator <= accumulator + coeff*big_array[offset - index];end
                        end
                else;
  end
  firlow620coeffs coeffs(index,coeff);
endmodule

module firlow620coeffs(
  input wire [7:0] index,
  output reg signed [11:0] coeff
);
  // tools will turn this into a 31x10 ROM
  always @(index)
    case (index)
        8'd0:  coeff = 12'sd0;
        8'd1:  coeff = 12'sd0;
        8'd2:  coeff = 12'sd0;
        8'd3:  coeff = 12'sd0;
        8'd4:  coeff = 12'sd0;
        8'd5:  coeff = 12'sd0;
        8'd6:  coeff = 12'sd0;
        8'd7:  coeff = 12'sd0;
        8'd8:  coeff = 12'sd0;
```

```
8'd9:  coeff = 12'sd0;
8'd10:  coeff = 12'sd0;
8'd11:  coeff = 12'sd0;
8'd12:  coeff = 12'sd0;
8'd13:  coeff = 12'sd0;
8'd14:  coeff = 12'sd0;
8'd15:  coeff = 12'sd0;
8'd16:  coeff = 12'sd0;
8'd17:  coeff = 12'sd0;
8'd18:  coeff = 12'sd1;
8'd19:  coeff = 12'sd1;
8'd20:  coeff = 12'sd1;
8'd21:  coeff = 12'sd1;
8'd22:  coeff = 12'sd1;
8'd23:  coeff = 12'sd1;
8'd24:  coeff = 12'sd1;
8'd25:  coeff = 12'sd1;
8'd26:  coeff = 12'sd1;
8'd27:  coeff = 12'sd1;
8'd28:  coeff = 12'sd1;
8'd29:  coeff = 12'sd1;
8'd30:  coeff = 12'sd1;
8'd31:  coeff = 12'sd1;
8'd32:  coeff = 12'sd1;
8'd33:  coeff = 12'sd2;
8'd34:  coeff = 12'sd2;
8'd35:  coeff = 12'sd2;
8'd36:  coeff = 12'sd2;
8'd37:  coeff = 12'sd1;
8'd38:  coeff = 12'sd1;
8'd39:  coeff = 12'sd1;
8'd40:  coeff = 12'sd1;
8'd41:  coeff = 12'sd1;
8'd42:  coeff = 12'sd1;
8'd43:  coeff = 12'sd1;
8'd44:  coeff = 12'sd1;
8'd45:  coeff = 12'sd1;
8'd46:  coeff = 12'sd0;
8'd47:  coeff = 12'sd0;
8'd48:  coeff = 12'sd0;
8'd49:  coeff = 12'sd0;
8'd50:  coeff = -12'sd1;
8'd51:  coeff = -12'sd1;
8'd52:  coeff = -12'sd1;
8'd53:  coeff = -12'sd2;
8'd54:  coeff = -12'sd2;
8'd55:  coeff = -12'sd2;
8'd56:  coeff = -12'sd3;
8'd57:  coeff = -12'sd3;
8'd58:  coeff = -12'sd4;
8'd59:  coeff = -12'sd4;
8'd60:  coeff = -12'sd4;
8'd61:  coeff = -12'sd5;
8'd62:  coeff = -12'sd5;
8'd63:  coeff = -12'sd5;
8'd64:  coeff = -12'sd6;
8'd65:  coeff = -12'sd6;
8'd66:  coeff = -12'sd6;
8'd67:  coeff = -12'sd7;
8'd68:  coeff = -12'sd7;
8'd69:  coeff = -12'sd7;
8'd70:  coeff = -12'sd7;
8'd71:  coeff = -12'sd7;
8'd72:  coeff = -12'sd7;
8'd73:  coeff = -12'sd7;
8'd74:  coeff = -12'sd7;
8'd75:  coeff = -12'sd7;
8'd76:  coeff = -12'sd7;
8'd77:  coeff = -12'sd7;
8'd78:  coeff = -12'sd6;
8'd79:  coeff = -12'sd6;
8'd80:  coeff = -12'sd5;
8'd81:  coeff = -12'sd5;
8'd82:  coeff = -12'sd4;
8'd83:  coeff = -12'sd3;
8'd84:  coeff = -12'sd2;
8'd85:  coeff = -12'sd1;
8'd86:  coeff = 12'sd0;
8'd87:  coeff = 12'sd1;
8'd88:  coeff = 12'sd2;
8'd89:  coeff = 12'sd3;
8'd90:  coeff = 12'sd5;
8'd91:  coeff = 12'sd6;
8'd92:  coeff = 12'sd8;
8'd93:  coeff = 12'sd9;
8'd94:  coeff = 12'sd11;
8'd95:  coeff = 12'sd12;
8'd96:  coeff = 12'sd14;
8'd97:  coeff = 12'sd16;
8'd98:  coeff = 12'sd18;
8'd99:  coeff = 12'sd19;
8'd100:  coeff = 12'sd21;
```

```
8'd101:  coeff = 12'sd23;
8'd102:  coeff = 12'sd25;
8'd103:  coeff = 12'sd27;
8'd104:  coeff = 12'sd29;
8'd105:  coeff = 12'sd31;
8'd106:  coeff = 12'sd32;
8'd107:  coeff = 12'sd34;
8'd108:  coeff = 12'sd36;
8'd109:  coeff = 12'sd38;
8'd110:  coeff = 12'sd39;
8'd111:  coeff = 12'sd41;
8'd112:  coeff = 12'sd43;
8'd113:  coeff = 12'sd44;
8'd114:  coeff = 12'sd45;
8'd115:  coeff = 12'sd47;
8'd116:  coeff = 12'sd48;
8'd117:  coeff = 12'sd49;
8'd118:  coeff = 12'sd50;
8'd119:  coeff = 12'sd51;
8'd120:  coeff = 12'sd51;
8'd121:  coeff = 12'sd52;
8'd122:  coeff = 12'sd52;
8'd123:  coeff = 12'sd53;
8'd124:  coeff = 12'sd53;
8'd125:  coeff = 12'sd53;
8'd126:  coeff = 12'sd53;
8'd127:  coeff = 12'sd53;
8'd128:  coeff = 12'sd52;
8'd129:  coeff = 12'sd52;
8'd130:  coeff = 12'sd51;
8'd131:  coeff = 12'sd51;
8'd132:  coeff = 12'sd50;
8'd133:  coeff = 12'sd49;
8'd134:  coeff = 12'sd48;
8'd135:  coeff = 12'sd47;
8'd136:  coeff = 12'sd45;
8'd137:  coeff = 12'sd44;
8'd138:  coeff = 12'sd43;
8'd139:  coeff = 12'sd41;
8'd140:  coeff = 12'sd39;
8'd141:  coeff = 12'sd38;
8'd142:  coeff = 12'sd36;
8'd143:  coeff = 12'sd34;
8'd144:  coeff = 12'sd32;
8'd145:  coeff = 12'sd31;
8'd146:  coeff = 12'sd29;
8'd147:  coeff = 12'sd27;
8'd148:  coeff = 12'sd25;
8'd149:  coeff = 12'sd23;
8'd150:  coeff = 12'sd21;
8'd151:  coeff = 12'sd19;
8'd152:  coeff = 12'sd18;
8'd153:  coeff = 12'sd16;
8'd154:  coeff = 12'sd14;
8'd155:  coeff = 12'sd12;
8'd156:  coeff = 12'sd11;
8'd157:  coeff = 12'sd9;
8'd158:  coeff = 12'sd8;
8'd159:  coeff = 12'sd6;
8'd160:  coeff = 12'sd5;
8'd161:  coeff = 12'sd3;
8'd162:  coeff = 12'sd2;
8'd163:  coeff = 12'sd1;
8'd164:  coeff = 12'sd0;
8'd165:  coeff = -12'sd1;
8'd166:  coeff = -12'sd2;
8'd167:  coeff = -12'sd3;
8'd168:  coeff = -12'sd4;
8'd169:  coeff = -12'sd5;
8'd170:  coeff = -12'sd5;
8'd171:  coeff = -12'sd6;
8'd172:  coeff = -12'sd6;
8'd173:  coeff = -12'sd7;
8'd174:  coeff = -12'sd7;
8'd175:  coeff = -12'sd7;
8'd176:  coeff = -12'sd7;
8'd177:  coeff = -12'sd7;
8'd178:  coeff = -12'sd7;
8'd179:  coeff = -12'sd7;
8'd180:  coeff = -12'sd7;
8'd181:  coeff = -12'sd7;
8'd182:  coeff = -12'sd7;
8'd183:  coeff = -12'sd7;
8'd184:  coeff = -12'sd6;
8'd185:  coeff = -12'sd6;
8'd186:  coeff = -12'sd6;
8'd187:  coeff = -12'sd5;
8'd188:  coeff = -12'sd5;
8'd189:  coeff = -12'sd5;
8'd190:  coeff = -12'sd4;
8'd191:  coeff = -12'sd4;
8'd192:  coeff = -12'sd4;
```

```
        8'd193:  coeff = -12'sd3;
        8'd194:  coeff = -12'sd3;
        8'd195:  coeff = -12'sd2;
        8'd196:  coeff = -12'sd2;
        8'd197:  coeff = -12'sd2;
        8'd198:  coeff = -12'sd1;
        8'd199:  coeff = -12'sd1;
        8'd200:  coeff = -12'sd1;
        8'd201:  coeff = 12'sd0;
        8'd202:  coeff = 12'sd0;
        8'd203:  coeff = 12'sd0;
        8'd204:  coeff = 12'sd0;
        8'd205:  coeff = 12'sd1;
        8'd206:  coeff = 12'sd1;
        8'd207:  coeff = 12'sd1;
        8'd208:  coeff = 12'sd1;
        8'd209:  coeff = 12'sd1;
        8'd210:  coeff = 12'sd1;
        8'd211:  coeff = 12'sd1;
        8'd212:  coeff = 12'sd1;
        8'd213:  coeff = 12'sd1;
        8'd214:  coeff = 12'sd2;
        8'd215:  coeff = 12'sd2;
        8'd216:  coeff = 12'sd2;
        8'd217:  coeff = 12'sd2;
        8'd218:  coeff = 12'sd1;
        8'd219:  coeff = 12'sd1;
        8'd220:  coeff = 12'sd1;
        8'd221:  coeff = 12'sd1;
        8'd222:  coeff = 12'sd1;
        8'd223:  coeff = 12'sd1;
        8'd224:  coeff = 12'sd1;
        8'd225:  coeff = 12'sd1;
        8'd226:  coeff = 12'sd1;
        8'd227:  coeff = 12'sd1;
        8'd228:  coeff = 12'sd1;
        8'd229:  coeff = 12'sd1;
        8'd230:  coeff = 12'sd1;
        8'd231:  coeff = 12'sd1;
        8'd232:  coeff = 12'sd1;
        8'd233:  coeff = 12'sd0;
        8'd234:  coeff = 12'sd0;
        8'd235:  coeff = 12'sd0;
        8'd236:  coeff = 12'sd0;
        8'd237:  coeff = 12'sd0;
        8'd238:  coeff = 12'sd0;
        8'd239:  coeff = 12'sd0;
        8'd240:  coeff = 12'sd0;
        8'd241:  coeff = 12'sd0;
        8'd242:  coeff = 12'sd0;
        8'd243:  coeff = 12'sd0;
        8'd244:  coeff = 12'sd0;
        8'd245:  coeff = 12'sd0;
        8'd246:  coeff = 12'sd0;
        8'd247:  coeff = 12'sd0;
        8'd248:  coeff = 12'sd0;
        8'd249:  coeff = 12'sd0;
        8'd250:  coeff = 12'sd0;
      default: coeff = 12'hXXX;
    endcase
endmodule

module firlow730(
  input wire clock,reset,ready,
  input wire signed [17:0] x,
  output reg signed [17:0] y
);
        reg signed [17:0]big_array[250:0]; //change big_array[x:0] x to coeff #
        reg [7:0]index = 0;
        reg [7:0]offset = 0;
        wire signed [11:0]coeff;
        reg signed [29:0]accumulator = 0;
        reg go = 0;

  always @(posedge clock) begin
                if (reset) begin //reinitialize everything
                        offset <= 0;
                        index <= 0;
                        accumulator <= 0;
                        go <= 0;end
                else if (ready) begin //on ready insert x into big_array and reset index
                        big_array[offset] <= x;
                        index <= 0;
                        go <= 1;end
                else if (go == 1) begin //while accumulator is accumulating
                        if (index == 8'd250) begin //change 8'dx x to coeff #
                                index <= 0;
                                go <= 0;
                                offset <= offset + 1;
                                y <= accumulator[29:12]<<<1;
                                accumulator <= 0;
                                end
```

```
                        else begin //accumulate procedure
                                index <= index + 1;
                                accumulator <= accumulator + coeff*big_array[offset - index];end
                        end
                else;
  end
  firlow730coeffs coeffs(index,coeff);
endmodule

module firlow730coeffs(
  input wire [7:0] index,
  output reg signed [11:0] coeff
);
  // tools will turn this into a 31x10 ROM
  always @(index)
    case (index)
        8'd0:  coeff = 12'sd0;
        8'd1:  coeff = 12'sd0;
        8'd2:  coeff = 12'sd0;
        8'd3:  coeff = 12'sd0;
        8'd4:  coeff = 12'sd0;
        8'd5:  coeff = 12'sd0;
        8'd6:  coeff = 12'sd0;
        8'd7:  coeff = 12'sd0;
        8'd8:  coeff = 12'sd0;
        8'd9:  coeff = -12'sd1;
        8'd10:  coeff = -12'sd1;
        8'd11:  coeff = -12'sd1;
        8'd12:  coeff = -12'sd1;
        8'd13:  coeff = -12'sd1;
        8'd14:  coeff = -12'sd1;
        8'd15:  coeff = -12'sd1;
        8'd16:  coeff = -12'sd1;
        8'd17:  coeff = -12'sd1;
        8'd18:  coeff = -12'sd1;
        8'd19:  coeff = -12'sd1;
        8'd20:  coeff = 12'sd0;
        8'd21:  coeff = 12'sd0;
        8'd22:  coeff = 12'sd0;
        8'd23:  coeff = 12'sd0;
        8'd24:  coeff = 12'sd0;
        8'd25:  coeff = 12'sd0;
        8'd26:  coeff = 12'sd0;
        8'd27:  coeff = 12'sd0;
        8'd28:  coeff = 12'sd0;
        8'd29:  coeff = 12'sd0;
        8'd30:  coeff = 12'sd0;
        8'd31:  coeff = 12'sd1;
        8'd32:  coeff = 12'sd1;
        8'd33:  coeff = 12'sd1;
        8'd34:  coeff = 12'sd1;
        8'd35:  coeff = 12'sd1;
        8'd36:  coeff = 12'sd1;
        8'd37:  coeff = 12'sd2;
        8'd38:  coeff = 12'sd2;
        8'd39:  coeff = 12'sd2;
        8'd40:  coeff = 12'sd2;
        8'd41:  coeff = 12'sd2;
        8'd42:  coeff = 12'sd2;
        8'd43:  coeff = 12'sd3;
        8'd44:  coeff = 12'sd3;
        8'd45:  coeff = 12'sd3;
        8'd46:  coeff = 12'sd3;
        8'd47:  coeff = 12'sd3;
        8'd48:  coeff = 12'sd3;
        8'd49:  coeff = 12'sd3;
        8'd50:  coeff = 12'sd3;
        8'd51:  coeff = 12'sd3;
        8'd52:  coeff = 12'sd2;
        8'd53:  coeff = 12'sd2;
        8'd54:  coeff = 12'sd2;
        8'd55:  coeff = 12'sd2;
        8'd56:  coeff = 12'sd1;
        8'd57:  coeff = 12'sd1;
        8'd58:  coeff = 12'sd1;
        8'd59:  coeff = 12'sd0;
        8'd60:  coeff = 12'sd0;
        8'd61:  coeff = -12'sd1;
        8'd62:  coeff = -12'sd1;
        8'd63:  coeff = -12'sd2;
        8'd64:  coeff = -12'sd3;
        8'd65:  coeff = -12'sd3;
        8'd66:  coeff = -12'sd4;
        8'd67:  coeff = -12'sd5;
        8'd68:  coeff = -12'sd5;
        8'd69:  coeff = -12'sd6;
        8'd70:  coeff = -12'sd6;
        8'd71:  coeff = -12'sd7;
        8'd72:  coeff = -12'sd8;
        8'd73:  coeff = -12'sd8;
        8'd74:  coeff = -12'sd8;
        8'd75:  coeff = -12'sd9;
```

```
8'd76:  coeff = -12'sd9;
8'd77:  coeff = -12'sd10;
8'd78:  coeff = -12'sd10;
8'd79:  coeff = -12'sd10;
8'd80:  coeff = -12'sd10;
8'd81:  coeff = -12'sd10;
8'd82:  coeff = -12'sd9;
8'd83:  coeff = -12'sd9;
8'd84:  coeff = -12'sd9;
8'd85:  coeff = -12'sd8;
8'd86:  coeff = -12'sd7;
8'd87:  coeff = -12'sd7;
8'd88:  coeff = -12'sd6;
8'd89:  coeff = -12'sd4;
8'd90:  coeff = -12'sd3;
8'd91:  coeff = -12'sd2;
8'd92:  coeff = 12'sd0;
8'd93:  coeff = 12'sd1;
8'd94:  coeff = 12'sd3;
8'd95:  coeff = 12'sd5;
8'd96:  coeff = 12'sd7;
8'd97:  coeff = 12'sd9;
8'd98:  coeff = 12'sd12;
8'd99:  coeff = 12'sd14;
8'd100:  coeff = 12'sd16;
8'd101:  coeff = 12'sd19;
8'd102:  coeff = 12'sd21;
8'd103:  coeff = 12'sd24;
8'd104:  coeff = 12'sd26;
8'd105:  coeff = 12'sd29;
8'd106:  coeff = 12'sd32;
8'd107:  coeff = 12'sd34;
8'd108:  coeff = 12'sd37;
8'd109:  coeff = 12'sd39;
8'd110:  coeff = 12'sd42;
8'd111:  coeff = 12'sd44;
8'd112:  coeff = 12'sd46;
8'd113:  coeff = 12'sd49;
8'd114:  coeff = 12'sd51;
8'd115:  coeff = 12'sd53;
8'd116:  coeff = 12'sd54;
8'd117:  coeff = 12'sd56;
8'd118:  coeff = 12'sd57;
8'd119:  coeff = 12'sd59;
8'd120:  coeff = 12'sd60;
8'd121:  coeff = 12'sd61;
8'd122:  coeff = 12'sd61;
8'd123:  coeff = 12'sd62;
8'd124:  coeff = 12'sd62;
8'd125:  coeff = 12'sd62;
8'd126:  coeff = 12'sd62;
8'd127:  coeff = 12'sd62;
8'd128:  coeff = 12'sd61;
8'd129:  coeff = 12'sd61;
8'd130:  coeff = 12'sd60;
8'd131:  coeff = 12'sd59;
8'd132:  coeff = 12'sd57;
8'd133:  coeff = 12'sd56;
8'd134:  coeff = 12'sd54;
8'd135:  coeff = 12'sd53;
8'd136:  coeff = 12'sd51;
8'd137:  coeff = 12'sd49;
8'd138:  coeff = 12'sd46;
8'd139:  coeff = 12'sd44;
8'd140:  coeff = 12'sd42;
8'd141:  coeff = 12'sd39;
8'd142:  coeff = 12'sd37;
8'd143:  coeff = 12'sd34;
8'd144:  coeff = 12'sd32;
8'd145:  coeff = 12'sd29;
8'd146:  coeff = 12'sd26;
8'd147:  coeff = 12'sd24;
8'd148:  coeff = 12'sd21;
8'd149:  coeff = 12'sd19;
8'd150:  coeff = 12'sd16;
8'd151:  coeff = 12'sd14;
8'd152:  coeff = 12'sd12;
8'd153:  coeff = 12'sd9;
8'd154:  coeff = 12'sd7;
8'd155:  coeff = 12'sd5;
8'd156:  coeff = 12'sd3;
8'd157:  coeff = 12'sd1;
8'd158:  coeff = 12'sd0;
8'd159:  coeff = -12'sd2;
8'd160:  coeff = -12'sd3;
8'd161:  coeff = -12'sd4;
8'd162:  coeff = -12'sd6;
8'd163:  coeff = -12'sd7;
8'd164:  coeff = -12'sd7;
8'd165:  coeff = -12'sd8;
8'd166:  coeff = -12'sd9;
8'd167:  coeff = -12'sd9;
```

```
        8'd168:  coeff = -12'sd9;
        8'd169:  coeff = -12'sd10;
        8'd170:  coeff = -12'sd10;
        8'd171:  coeff = -12'sd10;
        8'd172:  coeff = -12'sd10;
        8'd173:  coeff = -12'sd10;
        8'd174:  coeff = -12'sd9;
        8'd175:  coeff = -12'sd9;
        8'd176:  coeff = -12'sd8;
        8'd177:  coeff = -12'sd8;
        8'd178:  coeff = -12'sd8;
        8'd179:  coeff = -12'sd7;
        8'd180:  coeff = -12'sd6;
        8'd181:  coeff = -12'sd6;
        8'd182:  coeff = -12'sd5;
        8'd183:  coeff = -12'sd5;
        8'd184:  coeff = -12'sd4;
        8'd185:  coeff = -12'sd3;
        8'd186:  coeff = -12'sd3;
        8'd187:  coeff = -12'sd2;
        8'd188:  coeff = -12'sd1;
        8'd189:  coeff = -12'sd1;
        8'd190:  coeff = 12'sd0;
        8'd191:  coeff = 12'sd0;
        8'd192:  coeff = 12'sd1;
        8'd193:  coeff = 12'sd1;
        8'd194:  coeff = 12'sd1;
        8'd195:  coeff = 12'sd2;
        8'd196:  coeff = 12'sd2;
        8'd197:  coeff = 12'sd2;
        8'd198:  coeff = 12'sd2;
        8'd199:  coeff = 12'sd3;
        8'd200:  coeff = 12'sd3;
        8'd201:  coeff = 12'sd3;
        8'd202:  coeff = 12'sd3;
        8'd203:  coeff = 12'sd3;
        8'd204:  coeff = 12'sd3;
        8'd205:  coeff = 12'sd3;
        8'd206:  coeff = 12'sd3;
        8'd207:  coeff = 12'sd3;
        8'd208:  coeff = 12'sd2;
        8'd209:  coeff = 12'sd2;
        8'd210:  coeff = 12'sd2;
        8'd211:  coeff = 12'sd2;
        8'd212:  coeff = 12'sd2;
        8'd213:  coeff = 12'sd2;
        8'd214:  coeff = 12'sd1;
        8'd215:  coeff = 12'sd1;
        8'd216:  coeff = 12'sd1;
        8'd217:  coeff = 12'sd1;
        8'd218:  coeff = 12'sd1;
        8'd219:  coeff = 12'sd1;
        8'd220:  coeff = 12'sd0;
        8'd221:  coeff = 12'sd0;
        8'd222:  coeff = 12'sd0;
        8'd223:  coeff = 12'sd0;
        8'd224:  coeff = 12'sd0;
        8'd225:  coeff = 12'sd0;
        8'd226:  coeff = 12'sd0;
        8'd227:  coeff = 12'sd0;
        8'd228:  coeff = 12'sd0;
        8'd229:  coeff = 12'sd0;
        8'd230:  coeff = 12'sd0;
        8'd231:  coeff = -12'sd1;
        8'd232:  coeff = -12'sd1;
        8'd233:  coeff = -12'sd1;
        8'd234:  coeff = -12'sd1;
        8'd235:  coeff = -12'sd1;
        8'd236:  coeff = -12'sd1;
        8'd237:  coeff = -12'sd1;
        8'd238:  coeff = -12'sd1;
        8'd239:  coeff = -12'sd1;
        8'd240:  coeff = -12'sd1;
        8'd241:  coeff = -12'sd1;
        8'd242:  coeff = 12'sd0;
        8'd243:  coeff = 12'sd0;
        8'd244:  coeff = 12'sd0;
        8'd245:  coeff = 12'sd0;
        8'd246:  coeff = 12'sd0;
        8'd247:  coeff = 12'sd0;
        8'd248:  coeff = 12'sd0;
        8'd249:  coeff = 12'sd0;
        8'd250:  coeff = 12'sd0;
      default: coeff = 12'hXXX;
    endcase
endmodule

module firlow840(
  input wire clock,reset,ready,
  input wire signed [17:0] x,
  output reg signed [17:0] y
);
```

```
        reg signed [17:0]big_array[250:0]; //change big_array[x:0] x to coeff #
        reg [7:0]index = 0;
        reg [7:0]offset = 0;
        wire signed [11:0]coeff;
        reg signed [29:0]accumulator = 0;
        reg go = 0;

  always @(posedge clock) begin
                if (reset) begin //reinitialize everything
                        offset <= 0;
                        index <= 0;
                        accumulator <= 0;
                        go <= 0;end
                else if (ready) begin //on ready insert x into big_array and reset index
                        big_array[offset] <= x;
                        index <= 0;
                        go <= 1;end
                else if (go == 1) begin //while accumulator is accumulating
                        if (index == 8'd250) begin //change 8'dx x to coeff #
                                index <= 0;
                                go <= 0;
                                offset <= offset + 1;
                                y <= accumulator[29:12]<<<1;
                                accumulator <= 0;
                                end
                        else begin //accumulate procedure
                                index <= index + 1;
                                accumulator <= accumulator + coeff*big_array[offset - index];end
                        end
                else;
  end
  firlow840coeffs coeffs(index,coeff);
endmodule

module firlow840coeffs(
  input wire [7:0] index,
  output reg signed [11:0] coeff
);
  // tools will turn this into a 31x10 ROM
  always @(index)
    case (index)
        8'd0:  coeff = 12'sd0;
        8'd1:  coeff = 12'sd0;
        8'd2:  coeff = 12'sd0;
        8'd3:  coeff = 12'sd0;
        8'd4:  coeff = 12'sd0;
        8'd5:  coeff = 12'sd0;
        8'd6:  coeff = 12'sd0;
        8'd7:  coeff = 12'sd0;
        8'd8:  coeff = 12'sd0;
        8'd9:  coeff = 12'sd0;
        8'd10:  coeff = 12'sd0;
        8'd11:  coeff = 12'sd0;
        8'd12:  coeff = 12'sd0;
        8'd13:  coeff = 12'sd0;
        8'd14:  coeff = 12'sd0;
        8'd15:  coeff = 12'sd0;
        8'd16:  coeff = 12'sd0;
        8'd17:  coeff = 12'sd0;
        8'd18:  coeff = -12'sd1;
        8'd19:  coeff = -12'sd1;
        8'd20:  coeff = -12'sd1;
        8'd21:  coeff = -12'sd1;
        8'd22:  coeff = -12'sd1;
        8'd23:  coeff = -12'sd1;
        8'd24:  coeff = -12'sd1;
        8'd25:  coeff = -12'sd1;
        8'd26:  coeff = -12'sd1;
        8'd27:  coeff = -12'sd1;
        8'd28:  coeff = -12'sd1;
        8'd29:  coeff = -12'sd1;
        8'd30:  coeff = -12'sd1;
        8'd31:  coeff = -12'sd1;
        8'd32:  coeff = -12'sd1;
        8'd33:  coeff = -12'sd1;
        8'd34:  coeff = -12'sd1;
        8'd35:  coeff = -12'sd1;
        8'd36:  coeff = -12'sd1;
        8'd37:  coeff = 12'sd0;
        8'd38:  coeff = 12'sd0;
        8'd39:  coeff = 12'sd0;
        8'd40:  coeff = 12'sd0;
        8'd41:  coeff = 12'sd0;
        8'd42:  coeff = 12'sd1;
        8'd43:  coeff = 12'sd1;
        8'd44:  coeff = 12'sd1;
        8'd45:  coeff = 12'sd2;
        8'd46:  coeff = 12'sd2;
        8'd47:  coeff = 12'sd2;
        8'd48:  coeff = 12'sd3;
        8'd49:  coeff = 12'sd3;
        8'd50:  coeff = 12'sd3;
```

```
8'd51:  coeff = 12'sd3;
8'd52:  coeff = 12'sd4;
8'd53:  coeff = 12'sd4;
8'd54:  coeff = 12'sd4;
8'd55:  coeff = 12'sd4;
8'd56:  coeff = 12'sd4;
8'd57:  coeff = 12'sd4;
8'd58:  coeff = 12'sd4;
8'd59:  coeff = 12'sd4;
8'd60:  coeff = 12'sd4;
8'd61:  coeff = 12'sd4;
8'd62:  coeff = 12'sd3;
8'd63:  coeff = 12'sd3;
8'd64:  coeff = 12'sd2;
8'd65:  coeff = 12'sd2;
8'd66:  coeff = 12'sd1;
8'd67:  coeff = 12'sd1;
8'd68:  coeff = 12'sd0;
8'd69:  coeff = -12'sd1;
8'd70:  coeff = -12'sd2;
8'd71:  coeff = -12'sd3;
8'd72:  coeff = -12'sd4;
8'd73:  coeff = -12'sd4;
8'd74:  coeff = -12'sd5;
8'd75:  coeff = -12'sd6;
8'd76:  coeff = -12'sd7;
8'd77:  coeff = -12'sd8;
8'd78:  coeff = -12'sd9;
8'd79:  coeff = -12'sd10;
8'd80:  coeff = -12'sd10;
8'd81:  coeff = -12'sd11;
8'd82:  coeff = -12'sd11;
8'd83:  coeff = -12'sd12;
8'd84:  coeff = -12'sd12;
8'd85:  coeff = -12'sd12;
8'd86:  coeff = -12'sd12;
8'd87:  coeff = -12'sd12;
8'd88:  coeff = -12'sd11;
8'd89:  coeff = -12'sd11;
8'd90:  coeff = -12'sd10;
8'd91:  coeff = -12'sd9;
8'd92:  coeff = -12'sd8;
8'd93:  coeff = -12'sd6;
8'd94:  coeff = -12'sd5;
8'd95:  coeff = -12'sd3;
8'd96:  coeff = -12'sd1;
8'd97:  coeff = 12'sd1;
8'd98:  coeff = 12'sd4;
8'd99:  coeff = 12'sd6;
8'd100:  coeff = 12'sd9;
8'd101:  coeff = 12'sd12;
8'd102:  coeff = 12'sd15;
8'd103:  coeff = 12'sd18;
8'd104:  coeff = 12'sd22;
8'd105:  coeff = 12'sd25;
8'd106:  coeff = 12'sd28;
8'd107:  coeff = 12'sd32;
8'd108:  coeff = 12'sd35;
8'd109:  coeff = 12'sd39;
8'd110:  coeff = 12'sd42;
8'd111:  coeff = 12'sd45;
8'd112:  coeff = 12'sd48;
8'd113:  coeff = 12'sd52;
8'd114:  coeff = 12'sd55;
8'd115:  coeff = 12'sd57;
8'd116:  coeff = 12'sd60;
8'd117:  coeff = 12'sd62;
8'd118:  coeff = 12'sd64;
8'd119:  coeff = 12'sd66;
8'd120:  coeff = 12'sd68;
8'd121:  coeff = 12'sd69;
8'd122:  coeff = 12'sd70;
8'd123:  coeff = 12'sd71;
8'd124:  coeff = 12'sd72;
8'd125:  coeff = 12'sd72;
8'd126:  coeff = 12'sd72;
8'd127:  coeff = 12'sd71;
8'd128:  coeff = 12'sd70;
8'd129:  coeff = 12'sd69;
8'd130:  coeff = 12'sd68;
8'd131:  coeff = 12'sd66;
8'd132:  coeff = 12'sd64;
8'd133:  coeff = 12'sd62;
8'd134:  coeff = 12'sd60;
8'd135:  coeff = 12'sd57;
8'd136:  coeff = 12'sd55;
8'd137:  coeff = 12'sd52;
8'd138:  coeff = 12'sd48;
8'd139:  coeff = 12'sd45;
8'd140:  coeff = 12'sd42;
8'd141:  coeff = 12'sd39;
8'd142:  coeff = 12'sd35;
```

```
8'd143:  coeff = 12'sd32;
8'd144:  coeff = 12'sd28;
8'd145:  coeff = 12'sd25;
8'd146:  coeff = 12'sd22;
8'd147:  coeff = 12'sd18;
8'd148:  coeff = 12'sd15;
8'd149:  coeff = 12'sd12;
8'd150:  coeff = 12'sd9;
8'd151:  coeff = 12'sd6;
8'd152:  coeff = 12'sd4;
8'd153:  coeff = 12'sd1;
8'd154:  coeff = -12'sd1;
8'd155:  coeff = -12'sd3;
8'd156:  coeff = -12'sd5;
8'd157:  coeff = -12'sd6;
8'd158:  coeff = -12'sd8;
8'd159:  coeff = -12'sd9;
8'd160:  coeff = -12'sd10;
8'd161:  coeff = -12'sd11;
8'd162:  coeff = -12'sd11;
8'd163:  coeff = -12'sd12;
8'd164:  coeff = -12'sd12;
8'd165:  coeff = -12'sd12;
8'd166:  coeff = -12'sd12;
8'd167:  coeff = -12'sd12;
8'd168:  coeff = -12'sd11;
8'd169:  coeff = -12'sd11;
8'd170:  coeff = -12'sd10;
8'd171:  coeff = -12'sd10;
8'd172:  coeff = -12'sd9;
8'd173:  coeff = -12'sd8;
8'd174:  coeff = -12'sd7;
8'd175:  coeff = -12'sd6;
8'd176:  coeff = -12'sd5;
8'd177:  coeff = -12'sd4;
8'd178:  coeff = -12'sd4;
8'd179:  coeff = -12'sd3;
8'd180:  coeff = -12'sd2;
8'd181:  coeff = -12'sd1;
8'd182:  coeff = 12'sd0;
8'd183:  coeff = 12'sd1;
8'd184:  coeff = 12'sd1;
8'd185:  coeff = 12'sd2;
8'd186:  coeff = 12'sd2;
8'd187:  coeff = 12'sd3;
8'd188:  coeff = 12'sd3;
8'd189:  coeff = 12'sd4;
8'd190:  coeff = 12'sd4;
8'd191:  coeff = 12'sd4;
8'd192:  coeff = 12'sd4;
8'd193:  coeff = 12'sd4;
8'd194:  coeff = 12'sd4;
8'd195:  coeff = 12'sd4;
8'd196:  coeff = 12'sd4;
8'd197:  coeff = 12'sd4;
8'd198:  coeff = 12'sd4;
8'd199:  coeff = 12'sd3;
8'd200:  coeff = 12'sd3;
8'd201:  coeff = 12'sd3;
8'd202:  coeff = 12'sd3;
8'd203:  coeff = 12'sd2;
8'd204:  coeff = 12'sd2;
8'd205:  coeff = 12'sd2;
8'd206:  coeff = 12'sd1;
8'd207:  coeff = 12'sd1;
8'd208:  coeff = 12'sd1;
8'd209:  coeff = 12'sd0;
8'd210:  coeff = 12'sd0;
8'd211:  coeff = 12'sd0;
8'd212:  coeff = 12'sd0;
8'd213:  coeff = 12'sd0;
8'd214:  coeff = -12'sd1;
8'd215:  coeff = -12'sd1;
8'd216:  coeff = -12'sd1;
8'd217:  coeff = -12'sd1;
8'd218:  coeff = -12'sd1;
8'd219:  coeff = -12'sd1;
8'd220:  coeff = -12'sd1;
8'd221:  coeff = -12'sd1;
8'd222:  coeff = -12'sd1;
8'd223:  coeff = -12'sd1;
8'd224:  coeff = -12'sd1;
8'd225:  coeff = -12'sd1;
8'd226:  coeff = -12'sd1;
8'd227:  coeff = -12'sd1;
8'd228:  coeff = -12'sd1;
8'd229:  coeff = -12'sd1;
8'd230:  coeff = -12'sd1;
8'd231:  coeff = -12'sd1;
8'd232:  coeff = -12'sd1;
8'd233:  coeff = 12'sd0;
8'd234:  coeff = 12'sd0;
```

```
        8'd235:  coeff = 12'sd0;
        8'd236:  coeff = 12'sd0;
        8'd237:  coeff = 12'sd0;
        8'd238:  coeff = 12'sd0;
        8'd239:  coeff = 12'sd0;
        8'd240:  coeff = 12'sd0;
        8'd241:  coeff = 12'sd0;
        8'd242:  coeff = 12'sd0;
        8'd243:  coeff = 12'sd0;
        8'd244:  coeff = 12'sd0;
        8'd245:  coeff = 12'sd0;
        8'd246:  coeff = 12'sd0;
        8'd247:  coeff = 12'sd0;
        8'd248:  coeff = 12'sd0;
        8'd249:  coeff = 12'sd0;
        8'd250:  coeff = 12'sd0;
      default: coeff = 12'hXXX;
    endcase
endmodule

module firlow960(
  input wire clock,reset,ready,
  input wire signed [17:0] x,
  output reg signed [17:0] y
);
        reg signed [17:0]big_array[250:0]; //change big_array[x:0] x to coeff #
        reg [7:0]index = 0;
        reg [7:0]offset = 0;
        wire signed [11:0]coeff;
        reg signed [29:0]accumulator = 0;
        reg go = 0;

  always @(posedge clock) begin
                if (reset) begin //reinitialize everything
                        offset <= 0;
                        index <= 0;
                        accumulator <= 0;
                        go <= 0;end
                else if (ready) begin //on ready insert x into big_array and reset index
                        big_array[offset] <= x;
                        index <= 0;
                        go <= 1;end
                else if (go == 1) begin //while accumulator is accumulating
                        if (index == 8'd250) begin //change 8'dx x to coeff #
                                index <= 0;
                                go <= 0;
                                offset <= offset + 1;
                                y <= accumulator[29:12]<<<1;
                                accumulator <= 0;
                                end
                        else begin //accumulate procedure
                                index <= index + 1;
                                accumulator <= accumulator + coeff*big_array[offset - index];end
                        end
                else;
  end
  firlow960coeffs coeffs(index,coeff);
endmodule

module firlow960coeffs(
  input wire [7:0] index,
  output reg signed [11:0] coeff
);
  // tools will turn this into a 31x10 ROM
  always @(index)
    case (index)
        8'd0:  coeff = 12'sd0;
        8'd1:  coeff = 12'sd0;
        8'd2:  coeff = 12'sd0;
        8'd3:  coeff = 12'sd0;
        8'd4:  coeff = 12'sd0;
        8'd5:  coeff = 12'sd0;
        8'd6:  coeff = 12'sd0;
        8'd7:  coeff = 12'sd0;
        8'd8:  coeff = 12'sd0;
        8'd9:  coeff = 12'sd0;
        8'd10:  coeff = 12'sd1;
        8'd11:  coeff = 12'sd1;
        8'd12:  coeff = 12'sd1;
        8'd13:  coeff = 12'sd1;
        8'd14:  coeff = 12'sd1;
        8'd15:  coeff = 12'sd1;
        8'd16:  coeff = 12'sd1;
        8'd17:  coeff = 12'sd1;
        8'd18:  coeff = 12'sd1;
        8'd19:  coeff = 12'sd1;
        8'd20:  coeff = 12'sd0;
        8'd21:  coeff = 12'sd0;
        8'd22:  coeff = 12'sd0;
        8'd23:  coeff = 12'sd0;
        8'd24:  coeff = 12'sd0;
        8'd25:  coeff = 12'sd0;
```

```
8'd26:  coeff = 12'sd0;
8'd27:  coeff = 12'sd0;
8'd28:  coeff = 12'sd0;
8'd29:  coeff = -12'sd1;
8'd30:  coeff = -12'sd1;
8'd31:  coeff = -12'sd1;
8'd32:  coeff = -12'sd1;
8'd33:  coeff = -12'sd1;
8'd34:  coeff = -12'sd2;
8'd35:  coeff = -12'sd2;
8'd36:  coeff = -12'sd2;
8'd37:  coeff = -12'sd2;
8'd38:  coeff = -12'sd2;
8'd39:  coeff = -12'sd2;
8'd40:  coeff = -12'sd2;
8'd41:  coeff = -12'sd2;
8'd42:  coeff = -12'sd2;
8'd43:  coeff = -12'sd2;
8'd44:  coeff = -12'sd2;
8'd45:  coeff = -12'sd2;
8'd46:  coeff = -12'sd1;
8'd47:  coeff = -12'sd1;
8'd48:  coeff = -12'sd1;
8'd49:  coeff = 12'sd0;
8'd50:  coeff = 12'sd0;
8'd51:  coeff = 12'sd0;
8'd52:  coeff = 12'sd1;
8'd53:  coeff = 12'sd1;
8'd54:  coeff = 12'sd2;
8'd55:  coeff = 12'sd2;
8'd56:  coeff = 12'sd3;
8'd57:  coeff = 12'sd4;
8'd58:  coeff = 12'sd4;
8'd59:  coeff = 12'sd4;
8'd60:  coeff = 12'sd5;
8'd61:  coeff = 12'sd5;
8'd62:  coeff = 12'sd6;
8'd63:  coeff = 12'sd6;
8'd64:  coeff = 12'sd6;
8'd65:  coeff = 12'sd6;
8'd66:  coeff = 12'sd6;
8'd67:  coeff = 12'sd6;
8'd68:  coeff = 12'sd5;
8'd69:  coeff = 12'sd5;
8'd70:  coeff = 12'sd4;
8'd71:  coeff = 12'sd4;
8'd72:  coeff = 12'sd3;
8'd73:  coeff = 12'sd2;
8'd74:  coeff = 12'sd1;
8'd75:  coeff = 12'sd0;
8'd76:  coeff = -12'sd1;
8'd77:  coeff = -12'sd2;
8'd78:  coeff = -12'sd4;
8'd79:  coeff = -12'sd5;
8'd80:  coeff = -12'sd6;
8'd81:  coeff = -12'sd8;
8'd82:  coeff = -12'sd9;
8'd83:  coeff = -12'sd10;
8'd84:  coeff = -12'sd11;
8'd85:  coeff = -12'sd12;
8'd86:  coeff = -12'sd13;
8'd87:  coeff = -12'sd14;
8'd88:  coeff = -12'sd14;
8'd89:  coeff = -12'sd15;
8'd90:  coeff = -12'sd15;
8'd91:  coeff = -12'sd15;
8'd92:  coeff = -12'sd14;
8'd93:  coeff = -12'sd13;
8'd94:  coeff = -12'sd12;
8'd95:  coeff = -12'sd11;
8'd96:  coeff = -12'sd10;
8'd97:  coeff = -12'sd8;
8'd98:  coeff = -12'sd5;
8'd99:  coeff = -12'sd3;
8'd100:  coeff = 12'sd0;
8'd101:  coeff = 12'sd3;
8'd102:  coeff = 12'sd7;
8'd103:  coeff = 12'sd10;
8'd104:  coeff = 12'sd14;
8'd105:  coeff = 12'sd18;
8'd106:  coeff = 12'sd22;
8'd107:  coeff = 12'sd27;
8'd108:  coeff = 12'sd31;
8'd109:  coeff = 12'sd35;
8'd110:  coeff = 12'sd40;
8'd111:  coeff = 12'sd44;
8'd112:  coeff = 12'sd49;
8'd113:  coeff = 12'sd53;
8'd114:  coeff = 12'sd57;
8'd115:  coeff = 12'sd61;
8'd116:  coeff = 12'sd65;
8'd117:  coeff = 12'sd68;
```

```
8'd118:  coeff = 12'sd71;
8'd119:  coeff = 12'sd74;
8'd120:  coeff = 12'sd76;
8'd121:  coeff = 12'sd78;
8'd122:  coeff = 12'sd80;
8'd123:  coeff = 12'sd81;
8'd124:  coeff = 12'sd81;
8'd125:  coeff = 12'sd82;
8'd126:  coeff = 12'sd81;
8'd127:  coeff = 12'sd81;
8'd128:  coeff = 12'sd80;
8'd129:  coeff = 12'sd78;
8'd130:  coeff = 12'sd76;
8'd131:  coeff = 12'sd74;
8'd132:  coeff = 12'sd71;
8'd133:  coeff = 12'sd68;
8'd134:  coeff = 12'sd65;
8'd135:  coeff = 12'sd61;
8'd136:  coeff = 12'sd57;
8'd137:  coeff = 12'sd53;
8'd138:  coeff = 12'sd49;
8'd139:  coeff = 12'sd44;
8'd140:  coeff = 12'sd40;
8'd141:  coeff = 12'sd35;
8'd142:  coeff = 12'sd31;
8'd143:  coeff = 12'sd27;
8'd144:  coeff = 12'sd22;
8'd145:  coeff = 12'sd18;
8'd146:  coeff = 12'sd14;
8'd147:  coeff = 12'sd10;
8'd148:  coeff = 12'sd7;
8'd149:  coeff = 12'sd3;
8'd150:  coeff = 12'sd0;
8'd151:  coeff = -12'sd3;
8'd152:  coeff = -12'sd5;
8'd153:  coeff = -12'sd8;
8'd154:  coeff = -12'sd10;
8'd155:  coeff = -12'sd11;
8'd156:  coeff = -12'sd12;
8'd157:  coeff = -12'sd13;
8'd158:  coeff = -12'sd14;
8'd159:  coeff = -12'sd15;
8'd160:  coeff = -12'sd15;
8'd161:  coeff = -12'sd15;
8'd162:  coeff = -12'sd14;
8'd163:  coeff = -12'sd14;
8'd164:  coeff = -12'sd13;
8'd165:  coeff = -12'sd12;
8'd166:  coeff = -12'sd11;
8'd167:  coeff = -12'sd10;
8'd168:  coeff = -12'sd9;
8'd169:  coeff = -12'sd8;
8'd170:  coeff = -12'sd6;
8'd171:  coeff = -12'sd5;
8'd172:  coeff = -12'sd4;
8'd173:  coeff = -12'sd2;
8'd174:  coeff = -12'sd1;
8'd175:  coeff = 12'sd0;
8'd176:  coeff = 12'sd1;
8'd177:  coeff = 12'sd2;
8'd178:  coeff = 12'sd3;
8'd179:  coeff = 12'sd4;
8'd180:  coeff = 12'sd4;
8'd181:  coeff = 12'sd5;
8'd182:  coeff = 12'sd5;
8'd183:  coeff = 12'sd6;
8'd184:  coeff = 12'sd6;
8'd185:  coeff = 12'sd6;
8'd186:  coeff = 12'sd6;
8'd187:  coeff = 12'sd6;
8'd188:  coeff = 12'sd6;
8'd189:  coeff = 12'sd5;
8'd190:  coeff = 12'sd5;
8'd191:  coeff = 12'sd4;
8'd192:  coeff = 12'sd4;
8'd193:  coeff = 12'sd4;
8'd194:  coeff = 12'sd3;
8'd195:  coeff = 12'sd2;
8'd196:  coeff = 12'sd2;
8'd197:  coeff = 12'sd1;
8'd198:  coeff = 12'sd1;
8'd199:  coeff = 12'sd0;
8'd200:  coeff = 12'sd0;
8'd201:  coeff = 12'sd0;
8'd202:  coeff = -12'sd1;
8'd203:  coeff = -12'sd1;
8'd204:  coeff = -12'sd1;
8'd205:  coeff = -12'sd2;
8'd206:  coeff = -12'sd2;
8'd207:  coeff = -12'sd2;
8'd208:  coeff = -12'sd2;
8'd209:  coeff = -12'sd2;
```

```
        8'd210:  coeff = -12'sd2;
        8'd211:  coeff = -12'sd2;
        8'd212:  coeff = -12'sd2;
        8'd213:  coeff = -12'sd2;
        8'd214:  coeff = -12'sd2;
        8'd215:  coeff = -12'sd2;
        8'd216:  coeff = -12'sd2;
        8'd217:  coeff = -12'sd1;
        8'd218:  coeff = -12'sd1;
        8'd219:  coeff = -12'sd1;
        8'd220:  coeff = -12'sd1;
        8'd221:  coeff = -12'sd1;
        8'd222:  coeff = 12'sd0;
        8'd223:  coeff = 12'sd0;
        8'd224:  coeff = 12'sd0;
        8'd225:  coeff = 12'sd0;
        8'd226:  coeff = 12'sd0;
        8'd227:  coeff = 12'sd0;
        8'd228:  coeff = 12'sd0;
        8'd229:  coeff = 12'sd0;
        8'd230:  coeff = 12'sd0;
        8'd231:  coeff = 12'sd1;
        8'd232:  coeff = 12'sd1;
        8'd233:  coeff = 12'sd1;
        8'd234:  coeff = 12'sd1;
        8'd235:  coeff = 12'sd1;
        8'd236:  coeff = 12'sd1;
        8'd237:  coeff = 12'sd1;
        8'd238:  coeff = 12'sd1;
        8'd239:  coeff = 12'sd1;
        8'd240:  coeff = 12'sd1;
        8'd241:  coeff = 12'sd0;
        8'd242:  coeff = 12'sd0;
        8'd243:  coeff = 12'sd0;
        8'd244:  coeff = 12'sd0;
        8'd245:  coeff = 12'sd0;
        8'd246:  coeff = 12'sd0;
        8'd247:  coeff = 12'sd0;
        8'd248:  coeff = 12'sd0;
        8'd249:  coeff = 12'sd0;
        8'd250:  coeff = 12'sd0;
      default: coeff = 12'hXXX;
    endcase
endmodule

module firlow1100(
  input wire clock,reset,ready,
  input wire signed [17:0] x,
  output reg signed [17:0] y
);
        reg signed [17:0]big_array[250:0]; //change big_array[x:0] x to coeff #
        reg [7:0]index = 0;
        reg [7:0]offset = 0;
        wire signed [11:0]coeff;
        reg signed [29:0]accumulator = 0;
        reg go = 0;

  always @(posedge clock) begin
                if (reset) begin //reinitialize everything
                        offset <= 0;
                        index <= 0;
                        accumulator <= 0;
                        go <= 0;end
                else if (ready) begin //on ready insert x into big_array and reset index
                        big_array[offset] <= x;
                        index <= 0;
                        go <= 1;end
                else if (go == 1) begin //while accumulator is accumulating
                        if (index == 8'd250) begin //change 8'dx x to coeff #
                                index <= 0;
                                go <= 0;
                                offset <= offset + 1;
                                y <= accumulator[29:12]<<<1;
                                accumulator <= 0;
                                end
                        else begin //accumulate procedure
                                index <= index + 1;
                                accumulator <= accumulator + coeff*big_array[offset - index];end
                        end
                else;
  end
  firlow1100coeffs coeffs(index,coeff);
endmodule

module firlow1100coeffs(
  input wire [7:0] index,
  output reg signed [11:0] coeff
);
  // tools will turn this into a 31x10 ROM
  always @(index)
    case (index)
        8'd0:  coeff = 12'sd0;
```

```
8'd1:  coeff = 12'sd0;
8'd2:  coeff = 12'sd0;
8'd3:  coeff = 12'sd0;
8'd4:  coeff = 12'sd0;
8'd5:  coeff = 12'sd0;
8'd6:  coeff = 12'sd0;
8'd7:  coeff = 12'sd0;
8'd8:  coeff = 12'sd0;
8'd9:  coeff = 12'sd0;
8'd10:  coeff = 12'sd0;
8'd11:  coeff = 12'sd0;
8'd12:  coeff = 12'sd0;
8'd13:  coeff = 12'sd0;
8'd14:  coeff = 12'sd0;
8'd15:  coeff = 12'sd0;
8'd16:  coeff = 12'sd0;
8'd17:  coeff = 12'sd0;
8'd18:  coeff = 12'sd0;
8'd19:  coeff = 12'sd0;
8'd20:  coeff = 12'sd0;
8'd21:  coeff = 12'sd1;
8'd22:  coeff = 12'sd1;
8'd23:  coeff = 12'sd1;
8'd24:  coeff = 12'sd1;
8'd25:  coeff = 12'sd1;
8'd26:  coeff = 12'sd1;
8'd27:  coeff = 12'sd1;
8'd28:  coeff = 12'sd1;
8'd29:  coeff = 12'sd1;
8'd30:  coeff = 12'sd1;
8'd31:  coeff = 12'sd1;
8'd32:  coeff = 12'sd1;
8'd33:  coeff = 12'sd1;
8'd34:  coeff = 12'sd1;
8'd35:  coeff = 12'sd1;
8'd36:  coeff = 12'sd0;
8'd37:  coeff = 12'sd0;
8'd38:  coeff = 12'sd0;
8'd39:  coeff = 12'sd0;
8'd40:  coeff = -12'sd1;
8'd41:  coeff = -12'sd1;
8'd42:  coeff = -12'sd1;
8'd43:  coeff = -12'sd2;
8'd44:  coeff = -12'sd2;
8'd45:  coeff = -12'sd2;
8'd46:  coeff = -12'sd3;
8'd47:  coeff = -12'sd3;
8'd48:  coeff = -12'sd3;
8'd49:  coeff = -12'sd3;
8'd50:  coeff = -12'sd3;
8'd51:  coeff = -12'sd3;
8'd52:  coeff = -12'sd3;
8'd53:  coeff = -12'sd3;
8'd54:  coeff = -12'sd3;
8'd55:  coeff = -12'sd3;
8'd56:  coeff = -12'sd2;
8'd57:  coeff = -12'sd2;
8'd58:  coeff = -12'sd1;
8'd59:  coeff = 12'sd0;
8'd60:  coeff = 12'sd0;
8'd61:  coeff = 12'sd1;
8'd62:  coeff = 12'sd2;
8'd63:  coeff = 12'sd3;
8'd64:  coeff = 12'sd4;
8'd65:  coeff = 12'sd4;
8'd66:  coeff = 12'sd5;
8'd67:  coeff = 12'sd6;
8'd68:  coeff = 12'sd6;
8'd69:  coeff = 12'sd7;
8'd70:  coeff = 12'sd7;
8'd71:  coeff = 12'sd8;
8'd72:  coeff = 12'sd8;
8'd73:  coeff = 12'sd8;
8'd74:  coeff = 12'sd7;
8'd75:  coeff = 12'sd7;
8'd76:  coeff = 12'sd6;
8'd77:  coeff = 12'sd6;
8'd78:  coeff = 12'sd5;
8'd79:  coeff = 12'sd3;
8'd80:  coeff = 12'sd2;
8'd81:  coeff = 12'sd1;
8'd82:  coeff = -12'sd1;
8'd83:  coeff = -12'sd3;
8'd84:  coeff = -12'sd5;
8'd85:  coeff = -12'sd6;
8'd86:  coeff = -12'sd8;
8'd87:  coeff = -12'sd10;
8'd88:  coeff = -12'sd12;
8'd89:  coeff = -12'sd13;
8'd90:  coeff = -12'sd15;
8'd91:  coeff = -12'sd16;
8'd92:  coeff = -12'sd17;
```

```
8'd93:  coeff = -12'sd17;
8'd94:  coeff = -12'sd18;
8'd95:  coeff = -12'sd18;
8'd96:  coeff = -12'sd17;
8'd97:  coeff = -12'sd16;
8'd98:  coeff = -12'sd15;
8'd99:  coeff = -12'sd13;
8'd100:  coeff = -12'sd11;
8'd101:  coeff = -12'sd8;
8'd102:  coeff = -12'sd4;
8'd103:  coeff = -12'sd1;
8'd104:  coeff = 12'sd3;
8'd105:  coeff = 12'sd8;
8'd106:  coeff = 12'sd13;
8'd107:  coeff = 12'sd18;
8'd108:  coeff = 12'sd24;
8'd109:  coeff = 12'sd29;
8'd110:  coeff = 12'sd35;
8'd111:  coeff = 12'sd41;
8'd112:  coeff = 12'sd47;
8'd113:  coeff = 12'sd53;
8'd114:  coeff = 12'sd58;
8'd115:  coeff = 12'sd64;
8'd116:  coeff = 12'sd69;
8'd117:  coeff = 12'sd74;
8'd118:  coeff = 12'sd78;
8'd119:  coeff = 12'sd82;
8'd120:  coeff = 12'sd86;
8'd121:  coeff = 12'sd89;
8'd122:  coeff = 12'sd91;
8'd123:  coeff = 12'sd93;
8'd124:  coeff = 12'sd94;
8'd125:  coeff = 12'sd94;
8'd126:  coeff = 12'sd94;
8'd127:  coeff = 12'sd93;
8'd128:  coeff = 12'sd91;
8'd129:  coeff = 12'sd89;
8'd130:  coeff = 12'sd86;
8'd131:  coeff = 12'sd82;
8'd132:  coeff = 12'sd78;
8'd133:  coeff = 12'sd74;
8'd134:  coeff = 12'sd69;
8'd135:  coeff = 12'sd64;
8'd136:  coeff = 12'sd58;
8'd137:  coeff = 12'sd53;
8'd138:  coeff = 12'sd47;
8'd139:  coeff = 12'sd41;
8'd140:  coeff = 12'sd35;
8'd141:  coeff = 12'sd29;
8'd142:  coeff = 12'sd24;
8'd143:  coeff = 12'sd18;
8'd144:  coeff = 12'sd13;
8'd145:  coeff = 12'sd8;
8'd146:  coeff = 12'sd3;
8'd147:  coeff = -12'sd1;
8'd148:  coeff = -12'sd4;
8'd149:  coeff = -12'sd8;
8'd150:  coeff = -12'sd11;
8'd151:  coeff = -12'sd13;
8'd152:  coeff = -12'sd15;
8'd153:  coeff = -12'sd16;
8'd154:  coeff = -12'sd17;
8'd155:  coeff = -12'sd18;
8'd156:  coeff = -12'sd18;
8'd157:  coeff = -12'sd17;
8'd158:  coeff = -12'sd17;
8'd159:  coeff = -12'sd16;
8'd160:  coeff = -12'sd15;
8'd161:  coeff = -12'sd13;
8'd162:  coeff = -12'sd12;
8'd163:  coeff = -12'sd10;
8'd164:  coeff = -12'sd8;
8'd165:  coeff = -12'sd6;
8'd166:  coeff = -12'sd5;
8'd167:  coeff = -12'sd3;
8'd168:  coeff = -12'sd1;
8'd169:  coeff = 12'sd1;
8'd170:  coeff = 12'sd2;
8'd171:  coeff = 12'sd3;
8'd172:  coeff = 12'sd5;
8'd173:  coeff = 12'sd6;
8'd174:  coeff = 12'sd6;
8'd175:  coeff = 12'sd7;
8'd176:  coeff = 12'sd7;
8'd177:  coeff = 12'sd8;
8'd178:  coeff = 12'sd8;
8'd179:  coeff = 12'sd8;
8'd180:  coeff = 12'sd7;
8'd181:  coeff = 12'sd7;
8'd182:  coeff = 12'sd6;
8'd183:  coeff = 12'sd6;
8'd184:  coeff = 12'sd5;
```

```
        8'd185:  coeff = 12'sd4;
        8'd186:  coeff = 12'sd4;
        8'd187:  coeff = 12'sd3;
        8'd188:  coeff = 12'sd2;
        8'd189:  coeff = 12'sd1;
        8'd190:  coeff = 12'sd0;
        8'd191:  coeff = 12'sd0;
        8'd192:  coeff = -12'sd1;
        8'd193:  coeff = -12'sd2;
        8'd194:  coeff = -12'sd2;
        8'd195:  coeff = -12'sd3;
        8'd196:  coeff = -12'sd3;
        8'd197:  coeff = -12'sd3;
        8'd198:  coeff = -12'sd3;
        8'd199:  coeff = -12'sd3;
        8'd200:  coeff = -12'sd3;
        8'd201:  coeff = -12'sd3;
        8'd202:  coeff = -12'sd3;
        8'd203:  coeff = -12'sd3;
        8'd204:  coeff = -12'sd3;
        8'd205:  coeff = -12'sd2;
        8'd206:  coeff = -12'sd2;
        8'd207:  coeff = -12'sd2;
        8'd208:  coeff = -12'sd1;
        8'd209:  coeff = -12'sd1;
        8'd210:  coeff = -12'sd1;
        8'd211:  coeff = 12'sd0;
        8'd212:  coeff = 12'sd0;
        8'd213:  coeff = 12'sd0;
        8'd214:  coeff = 12'sd0;
        8'd215:  coeff = 12'sd1;
        8'd216:  coeff = 12'sd1;
        8'd217:  coeff = 12'sd1;
        8'd218:  coeff = 12'sd1;
        8'd219:  coeff = 12'sd1;
        8'd220:  coeff = 12'sd1;
        8'd221:  coeff = 12'sd1;
        8'd222:  coeff = 12'sd1;
        8'd223:  coeff = 12'sd1;
        8'd224:  coeff = 12'sd1;
        8'd225:  coeff = 12'sd1;
        8'd226:  coeff = 12'sd1;
        8'd227:  coeff = 12'sd1;
        8'd228:  coeff = 12'sd1;
        8'd229:  coeff = 12'sd1;
        8'd230:  coeff = 12'sd0;
        8'd231:  coeff = 12'sd0;
        8'd232:  coeff = 12'sd0;
        8'd233:  coeff = 12'sd0;
        8'd234:  coeff = 12'sd0;
        8'd235:  coeff = 12'sd0;
        8'd236:  coeff = 12'sd0;
        8'd237:  coeff = 12'sd0;
        8'd238:  coeff = 12'sd0;
        8'd239:  coeff = 12'sd0;
        8'd240:  coeff = 12'sd0;
        8'd241:  coeff = 12'sd0;
        8'd242:  coeff = 12'sd0;
        8'd243:  coeff = 12'sd0;
        8'd244:  coeff = 12'sd0;
        8'd245:  coeff = 12'sd0;
        8'd246:  coeff = 12'sd0;
        8'd247:  coeff = 12'sd0;
        8'd248:  coeff = 12'sd0;
        8'd249:  coeff = 12'sd0;
        8'd250:  coeff = 12'sd0;
      default: coeff = 12'hXXX;
    endcase
endmodule

module firlow1300(
  input wire clock,reset,ready,
  input wire signed [17:0] x,
  output reg signed [17:0] y
);
        reg signed [17:0]big_array[250:0]; //change big_array[x:0] x to coeff #
        reg [7:0]index = 0;
        reg [7:0]offset = 0;
        wire signed [11:0]coeff;
        reg signed [29:0]accumulator = 0;
        reg go = 0;

  always @(posedge clock) begin
                if (reset) begin //reinitialize everything
                        offset <= 0;
                        index <= 0;
                        accumulator <= 0;
                        go <= 0;end
                else if (ready) begin //on ready insert x into big_array and reset index
                        big_array[offset] <= x;
                        index <= 0;
                        go <= 1;end
```

```
                else if (go == 1) begin //while accumulator is accumulating
                        if (index == 8'd250) begin //change 8'dx x to coeff #
                                index <= 0;
                                go <= 0;
                                offset <= offset + 1;
                                y <= accumulator[29:12]<<<1;
                                accumulator <= 0;
                                end
                        else begin //accumulate procedure
                                index <= index + 1;
                                accumulator <= accumulator + coeff*big_array[offset - index];end
                        end
                else;
  end
  firlow1300coeffs coeffs(index,coeff);
endmodule

module firlow1300coeffs(
  input wire [7:0] index,
  output reg signed [11:0] coeff
);
  // tools will turn this into a 31x10 ROM
  always @(index)
    case (index)
         8'd0:  coeff = 12'sd0;
         8'd1:  coeff = 12'sd0;
         8'd2:  coeff = 12'sd0;
         8'd3:  coeff = 12'sd0;
         8'd4:  coeff = 12'sd0;
         8'd5:  coeff = 12'sd0;
         8'd6:  coeff = 12'sd0;
         8'd7:  coeff = 12'sd0;
         8'd8:  coeff = 12'sd0;
         8'd9:  coeff = 12'sd0;
         8'd10:  coeff = 12'sd0;
         8'd11:  coeff = 12'sd0;
         8'd12:  coeff = 12'sd0;
         8'd13:  coeff = 12'sd0;
         8'd14:  coeff = 12'sd0;
         8'd15:  coeff = 12'sd0;
         8'd16:  coeff = 12'sd0;
         8'd17:  coeff = 12'sd0;
         8'd18:  coeff = 12'sd0;
         8'd19:  coeff = -12'sd1;
         8'd20:  coeff = -12'sd1;
         8'd21:  coeff = -12'sd1;
         8'd22:  coeff = -12'sd1;
         8'd23:  coeff = -12'sd1;
         8'd24:  coeff = -12'sd1;
         8'd25:  coeff = -12'sd1;
         8'd26:  coeff = -12'sd1;
         8'd27:  coeff = -12'sd1;
         8'd28:  coeff = -12'sd1;
         8'd29:  coeff = -12'sd1;
         8'd30:  coeff = -12'sd1;
         8'd31:  coeff = 12'sd0;
         8'd32:  coeff = 12'sd0;
         8'd33:  coeff = 12'sd0;
         8'd34:  coeff = 12'sd0;
         8'd35:  coeff = 12'sd1;
         8'd36:  coeff = 12'sd1;
         8'd37:  coeff = 12'sd1;
         8'd38:  coeff = 12'sd2;
         8'd39:  coeff = 12'sd2;
         8'd40:  coeff = 12'sd2;
         8'd41:  coeff = 12'sd2;
         8'd42:  coeff = 12'sd2;
         8'd43:  coeff = 12'sd3;
         8'd44:  coeff = 12'sd3;
         8'd45:  coeff = 12'sd2;
         8'd46:  coeff = 12'sd2;
         8'd47:  coeff = 12'sd2;
         8'd48:  coeff = 12'sd2;
         8'd49:  coeff = 12'sd1;
         8'd50:  coeff = 12'sd1;
         8'd51:  coeff = 12'sd0;
         8'd52:  coeff = -12'sd1;
         8'd53:  coeff = -12'sd1;
         8'd54:  coeff = -12'sd2;
         8'd55:  coeff = -12'sd3;
         8'd56:  coeff = -12'sd3;
         8'd57:  coeff = -12'sd4;
         8'd58:  coeff = -12'sd4;
         8'd59:  coeff = -12'sd5;
         8'd60:  coeff = -12'sd5;
         8'd61:  coeff = -12'sd5;
         8'd62:  coeff = -12'sd5;
         8'd63:  coeff = -12'sd5;
         8'd64:  coeff = -12'sd5;
         8'd65:  coeff = -12'sd4;
         8'd66:  coeff = -12'sd4;
         8'd67:  coeff = -12'sd3;
```

```
8'd68:  coeff = -12'sd2;
8'd69:  coeff = -12'sd1;
8'd70:  coeff = 12'sd0;
8'd71:  coeff = 12'sd2;
8'd72:  coeff = 12'sd3;
8'd73:  coeff = 12'sd4;
8'd74:  coeff = 12'sd6;
8'd75:  coeff = 12'sd7;
8'd76:  coeff = 12'sd8;
8'd77:  coeff = 12'sd9;
8'd78:  coeff = 12'sd10;
8'd79:  coeff = 12'sd10;
8'd80:  coeff = 12'sd10;
8'd81:  coeff = 12'sd10;
8'd82:  coeff = 12'sd10;
8'd83:  coeff = 12'sd9;
8'd84:  coeff = 12'sd8;
8'd85:  coeff = 12'sd6;
8'd86:  coeff = 12'sd5;
8'd87:  coeff = 12'sd3;
8'd88:  coeff = 12'sd0;
8'd89:  coeff = -12'sd2;
8'd90:  coeff = -12'sd5;
8'd91:  coeff = -12'sd8;
8'd92:  coeff = -12'sd10;
8'd93:  coeff = -12'sd13;
8'd94:  coeff = -12'sd15;
8'd95:  coeff = -12'sd18;
8'd96:  coeff = -12'sd19;
8'd97:  coeff = -12'sd21;
8'd98:  coeff = -12'sd21;
8'd99:  coeff = -12'sd22;
8'd100:  coeff = -12'sd21;
8'd101:  coeff = -12'sd20;
8'd102:  coeff = -12'sd18;
8'd103:  coeff = -12'sd16;
8'd104:  coeff = -12'sd12;
8'd105:  coeff = -12'sd8;
8'd106:  coeff = -12'sd3;
8'd107:  coeff = 12'sd3;
8'd108:  coeff = 12'sd9;
8'd109:  coeff = 12'sd16;
8'd110:  coeff = 12'sd23;
8'd111:  coeff = 12'sd31;
8'd112:  coeff = 12'sd39;
8'd113:  coeff = 12'sd47;
8'd114:  coeff = 12'sd56;
8'd115:  coeff = 12'sd64;
8'd116:  coeff = 12'sd71;
8'd117:  coeff = 12'sd79;
8'd118:  coeff = 12'sd86;
8'd119:  coeff = 12'sd92;
8'd120:  coeff = 12'sd98;
8'd121:  coeff = 12'sd102;
8'd122:  coeff = 12'sd106;
8'd123:  coeff = 12'sd109;
8'd124:  coeff = 12'sd110;
8'd125:  coeff = 12'sd111;
8'd126:  coeff = 12'sd110;
8'd127:  coeff = 12'sd109;
8'd128:  coeff = 12'sd106;
8'd129:  coeff = 12'sd102;
8'd130:  coeff = 12'sd98;
8'd131:  coeff = 12'sd92;
8'd132:  coeff = 12'sd86;
8'd133:  coeff = 12'sd79;
8'd134:  coeff = 12'sd71;
8'd135:  coeff = 12'sd64;
8'd136:  coeff = 12'sd56;
8'd137:  coeff = 12'sd47;
8'd138:  coeff = 12'sd39;
8'd139:  coeff = 12'sd31;
8'd140:  coeff = 12'sd23;
8'd141:  coeff = 12'sd16;
8'd142:  coeff = 12'sd9;
8'd143:  coeff = 12'sd3;
8'd144:  coeff = -12'sd3;
8'd145:  coeff = -12'sd8;
8'd146:  coeff = -12'sd12;
8'd147:  coeff = -12'sd16;
8'd148:  coeff = -12'sd18;
8'd149:  coeff = -12'sd20;
8'd150:  coeff = -12'sd21;
8'd151:  coeff = -12'sd22;
8'd152:  coeff = -12'sd21;
8'd153:  coeff = -12'sd21;
8'd154:  coeff = -12'sd19;
8'd155:  coeff = -12'sd18;
8'd156:  coeff = -12'sd15;
8'd157:  coeff = -12'sd13;
8'd158:  coeff = -12'sd10;
8'd159:  coeff = -12'sd8;
```

```
   8'd160:  coeff = -12'sd5;
   8'd161:  coeff = -12'sd2;
   8'd162:  coeff = 12'sd0;
   8'd163:  coeff = 12'sd3;
   8'd164:  coeff = 12'sd5;
   8'd165:  coeff = 12'sd6;
   8'd166:  coeff = 12'sd8;
   8'd167:  coeff = 12'sd9;
   8'd168:  coeff = 12'sd10;
   8'd169:  coeff = 12'sd10;
   8'd170:  coeff = 12'sd10;
   8'd171:  coeff = 12'sd10;
   8'd172:  coeff = 12'sd10;
   8'd173:  coeff = 12'sd9;
   8'd174:  coeff = 12'sd8;
   8'd175:  coeff = 12'sd7;
   8'd176:  coeff = 12'sd6;
   8'd177:  coeff = 12'sd4;
   8'd178:  coeff = 12'sd3;
   8'd179:  coeff = 12'sd2;
   8'd180:  coeff = 12'sd0;
   8'd181:  coeff = -12'sd1;
   8'd182:  coeff = -12'sd2;
   8'd183:  coeff = -12'sd3;
   8'd184:  coeff = -12'sd4;
   8'd185:  coeff = -12'sd4;
   8'd186:  coeff = -12'sd5;
   8'd187:  coeff = -12'sd5;
   8'd188:  coeff = -12'sd5;
   8'd189:  coeff = -12'sd5;
   8'd190:  coeff = -12'sd5;
   8'd191:  coeff = -12'sd5;
   8'd192:  coeff = -12'sd4;
   8'd193:  coeff = -12'sd4;
   8'd194:  coeff = -12'sd3;
   8'd195:  coeff = -12'sd3;
   8'd196:  coeff = -12'sd2;
   8'd197:  coeff = -12'sd1;
   8'd198:  coeff = -12'sd1;
   8'd199:  coeff = 12'sd0;
   8'd200:  coeff = 12'sd1;
   8'd201:  coeff = 12'sd1;
   8'd202:  coeff = 12'sd2;
   8'd203:  coeff = 12'sd2;
   8'd204:  coeff = 12'sd2;
   8'd205:  coeff = 12'sd2;
   8'd206:  coeff = 12'sd3;
   8'd207:  coeff = 12'sd3;
   8'd208:  coeff = 12'sd2;
   8'd209:  coeff = 12'sd2;
   8'd210:  coeff = 12'sd2;
   8'd211:  coeff = 12'sd2;
   8'd212:  coeff = 12'sd2;
   8'd213:  coeff = 12'sd1;
   8'd214:  coeff = 12'sd1;
   8'd215:  coeff = 12'sd1;
   8'd216:  coeff = 12'sd0;
   8'd217:  coeff = 12'sd0;
   8'd218:  coeff = 12'sd0;
   8'd219:  coeff = 12'sd0;
   8'd220:  coeff = -12'sd1;
   8'd221:  coeff = -12'sd1;
   8'd222:  coeff = -12'sd1;
   8'd223:  coeff = -12'sd1;
   8'd224:  coeff = -12'sd1;
   8'd225:  coeff = -12'sd1;
   8'd226:  coeff = -12'sd1;
   8'd227:  coeff = -12'sd1;
   8'd228:  coeff = -12'sd1;
   8'd229:  coeff = -12'sd1;
   8'd230:  coeff = -12'sd1;
   8'd231:  coeff = -12'sd1;
   8'd232:  coeff = 12'sd0;
   8'd233:  coeff = 12'sd0;
   8'd234:  coeff = 12'sd0;
   8'd235:  coeff = 12'sd0;
   8'd236:  coeff = 12'sd0;
   8'd237:  coeff = 12'sd0;
   8'd238:  coeff = 12'sd0;
   8'd239:  coeff = 12'sd0;
   8'd240:  coeff = 12'sd0;
   8'd241:  coeff = 12'sd0;
   8'd242:  coeff = 12'sd0;
   8'd243:  coeff = 12'sd0;
   8'd244:  coeff = 12'sd0;
   8'd245:  coeff = 12'sd0;
   8'd246:  coeff = 12'sd0;
   8'd247:  coeff = 12'sd0;
   8'd248:  coeff = 12'sd0;
   8'd249:  coeff = 12'sd0;
   8'd250:  coeff = 12'sd0;
default: coeff = 12'hXXX;
```

```
    endcase
endmodule

module firlow1600(
  input wire clock,reset,ready,
  input wire signed [17:0] x,
  output reg signed [17:0] y
);
        reg signed [17:0]big_array[250:0]; //change big_array[x:0] x to coeff #
        reg [7:0]index = 0;
        reg [7:0]offset = 0;
        wire signed [11:0]coeff;
        reg signed [29:0]accumulator = 0;
        reg go = 0;

  always @(posedge clock) begin
                if (reset) begin //reinitialize everything
                        offset <= 0;
                        index <= 0;
                        accumulator <= 0;
                        go <= 0;end
                else if (ready) begin //on ready insert x into big_array and reset index
                        big_array[offset] <= x;
                        index <= 0;
                        go <= 1;end
                else if (go == 1) begin //while accumulator is accumulating
                        if (index == 8'd250) begin //change 8'dx x to coeff #
                                index <= 0;
                                go <= 0;
                                offset <= offset + 1;
                                y <= accumulator[29:12]<<<1;
                                accumulator <= 0;
                                end
                        else begin //accumulate procedure
                                index <= index + 1;
                                accumulator <= accumulator + coeff*big_array[offset - index];end
                        end
                else;
  end
  firlow1600coeffs coeffs(index,coeff);
endmodule

module firlow1600coeffs(
  input wire [7:0] index,
  output reg signed [11:0] coeff
);
  // tools will turn this into a 31x10 ROM
  always @(index)
    case (index)
        8'd0:  coeff = 12'sd0;
        8'd1:  coeff = 12'sd0;
        8'd2:  coeff = 12'sd0;
        8'd3:  coeff = 12'sd0;
        8'd4:  coeff = 12'sd0;
        8'd5:  coeff = 12'sd0;
        8'd6:  coeff = 12'sd0;
        8'd7:  coeff = 12'sd0;
        8'd8:  coeff = 12'sd0;
        8'd9:  coeff = 12'sd0;
        8'd10:  coeff = 12'sd0;
        8'd11:  coeff = -12'sd1;
        8'd12:  coeff = -12'sd1;
        8'd13:  coeff = -12'sd1;
        8'd14:  coeff = -12'sd1;
        8'd15:  coeff = -12'sd1;
        8'd16:  coeff = -12'sd1;
        8'd17:  coeff = 12'sd0;
        8'd18:  coeff = 12'sd0;
        8'd19:  coeff = 12'sd0;
        8'd20:  coeff = 12'sd0;
        8'd21:  coeff = 12'sd0;
        8'd22:  coeff = 12'sd0;
        8'd23:  coeff = 12'sd1;
        8'd24:  coeff = 12'sd1;
        8'd25:  coeff = 12'sd1;
        8'd26:  coeff = 12'sd1;
        8'd27:  coeff = 12'sd1;
        8'd28:  coeff = 12'sd1;
        8'd29:  coeff = 12'sd1;
        8'd30:  coeff = 12'sd1;
        8'd31:  coeff = 12'sd1;
        8'd32:  coeff = 12'sd1;
        8'd33:  coeff = 12'sd1;
        8'd34:  coeff = 12'sd0;
        8'd35:  coeff = 12'sd0;
        8'd36:  coeff = 12'sd0;
        8'd37:  coeff = -12'sd1;
        8'd38:  coeff = -12'sd1;
        8'd39:  coeff = -12'sd2;
        8'd40:  coeff = -12'sd2;
        8'd41:  coeff = -12'sd2;
        8'd42:  coeff = -12'sd2;
```

```
8'd43:  coeff = -12'sd3;
8'd44:  coeff = -12'sd3;
8'd45:  coeff = -12'sd2;
8'd46:  coeff = -12'sd2;
8'd47:  coeff = -12'sd2;
8'd48:  coeff = -12'sd1;
8'd49:  coeff = -12'sd1;
8'd50:  coeff = 12'sd0;
8'd51:  coeff = 12'sd1;
8'd52:  coeff = 12'sd2;
8'd53:  coeff = 12'sd2;
8'd54:  coeff = 12'sd3;
8'd55:  coeff = 12'sd4;
8'd56:  coeff = 12'sd4;
8'd57:  coeff = 12'sd5;
8'd58:  coeff = 12'sd5;
8'd59:  coeff = 12'sd5;
8'd60:  coeff = 12'sd4;
8'd61:  coeff = 12'sd4;
8'd62:  coeff = 12'sd3;
8'd63:  coeff = 12'sd2;
8'd64:  coeff = 12'sd1;
8'd65:  coeff = 12'sd0;
8'd66:  coeff = -12'sd1;
8'd67:  coeff = -12'sd3;
8'd68:  coeff = -12'sd4;
8'd69:  coeff = -12'sd5;
8'd70:  coeff = -12'sd6;
8'd71:  coeff = -12'sd7;
8'd72:  coeff = -12'sd8;
8'd73:  coeff = -12'sd8;
8'd74:  coeff = -12'sd8;
8'd75:  coeff = -12'sd8;
8'd76:  coeff = -12'sd7;
8'd77:  coeff = -12'sd6;
8'd78:  coeff = -12'sd4;
8'd79:  coeff = -12'sd2;
8'd80:  coeff = 12'sd0;
8'd81:  coeff = 12'sd2;
8'd82:  coeff = 12'sd5;
8'd83:  coeff = 12'sd7;
8'd84:  coeff = 12'sd9;
8'd85:  coeff = 12'sd11;
8'd86:  coeff = 12'sd13;
8'd87:  coeff = 12'sd14;
8'd88:  coeff = 12'sd14;
8'd89:  coeff = 12'sd14;
8'd90:  coeff = 12'sd13;
8'd91:  coeff = 12'sd12;
8'd92:  coeff = 12'sd10;
8'd93:  coeff = 12'sd7;
8'd94:  coeff = 12'sd4;
8'd95:  coeff = 12'sd0;
8'd96:  coeff = -12'sd4;
8'd97:  coeff = -12'sd8;
8'd98:  coeff = -12'sd13;
8'd99:  coeff = -12'sd17;
8'd100:  coeff = -12'sd21;
8'd101:  coeff = -12'sd24;
8'd102:  coeff = -12'sd26;
8'd103:  coeff = -12'sd27;
8'd104:  coeff = -12'sd28;
8'd105:  coeff = -12'sd27;
8'd106:  coeff = -12'sd24;
8'd107:  coeff = -12'sd20;
8'd108:  coeff = -12'sd15;
8'd109:  coeff = -12'sd8;
8'd110:  coeff = 12'sd0;
8'd111:  coeff = 12'sd9;
8'd112:  coeff = 12'sd20;
8'd113:  coeff = 12'sd31;
8'd114:  coeff = 12'sd43;
8'd115:  coeff = 12'sd56;
8'd116:  coeff = 12'sd68;
8'd117:  coeff = 12'sd80;
8'd118:  coeff = 12'sd92;
8'd119:  coeff = 12'sd103;
8'd120:  coeff = 12'sd113;
8'd121:  coeff = 12'sd121;
8'd122:  coeff = 12'sd128;
8'd123:  coeff = 12'sd133;
8'd124:  coeff = 12'sd136;
8'd125:  coeff = 12'sd137;
8'd126:  coeff = 12'sd136;
8'd127:  coeff = 12'sd133;
8'd128:  coeff = 12'sd128;
8'd129:  coeff = 12'sd121;
8'd130:  coeff = 12'sd113;
8'd131:  coeff = 12'sd103;
8'd132:  coeff = 12'sd92;
8'd133:  coeff = 12'sd80;
8'd134:  coeff = 12'sd68;
```

```
8'd135:  coeff = 12'sd56;
8'd136:  coeff = 12'sd43;
8'd137:  coeff = 12'sd31;
8'd138:  coeff = 12'sd20;
8'd139:  coeff = 12'sd9;
8'd140:  coeff = 12'sd0;
8'd141:  coeff = -12'sd8;
8'd142:  coeff = -12'sd15;
8'd143:  coeff = -12'sd20;
8'd144:  coeff = -12'sd24;
8'd145:  coeff = -12'sd27;
8'd146:  coeff = -12'sd28;
8'd147:  coeff = -12'sd27;
8'd148:  coeff = -12'sd26;
8'd149:  coeff = -12'sd24;
8'd150:  coeff = -12'sd21;
8'd151:  coeff = -12'sd17;
8'd152:  coeff = -12'sd13;
8'd153:  coeff = -12'sd8;
8'd154:  coeff = -12'sd4;
8'd155:  coeff = 12'sd0;
8'd156:  coeff = 12'sd4;
8'd157:  coeff = 12'sd7;
8'd158:  coeff = 12'sd10;
8'd159:  coeff = 12'sd12;
8'd160:  coeff = 12'sd13;
8'd161:  coeff = 12'sd14;
8'd162:  coeff = 12'sd14;
8'd163:  coeff = 12'sd14;
8'd164:  coeff = 12'sd13;
8'd165:  coeff = 12'sd11;
8'd166:  coeff = 12'sd9;
8'd167:  coeff = 12'sd7;
8'd168:  coeff = 12'sd5;
8'd169:  coeff = 12'sd2;
8'd170:  coeff = 12'sd0;
8'd171:  coeff = -12'sd2;
8'd172:  coeff = -12'sd4;
8'd173:  coeff = -12'sd6;
8'd174:  coeff = -12'sd7;
8'd175:  coeff = -12'sd8;
8'd176:  coeff = -12'sd8;
8'd177:  coeff = -12'sd8;
8'd178:  coeff = -12'sd8;
8'd179:  coeff = -12'sd7;
8'd180:  coeff = -12'sd6;
8'd181:  coeff = -12'sd5;
8'd182:  coeff = -12'sd4;
8'd183:  coeff = -12'sd3;
8'd184:  coeff = -12'sd1;
8'd185:  coeff = 12'sd0;
8'd186:  coeff = 12'sd1;
8'd187:  coeff = 12'sd2;
8'd188:  coeff = 12'sd3;
8'd189:  coeff = 12'sd4;
8'd190:  coeff = 12'sd4;
8'd191:  coeff = 12'sd5;
8'd192:  coeff = 12'sd5;
8'd193:  coeff = 12'sd5;
8'd194:  coeff = 12'sd4;
8'd195:  coeff = 12'sd4;
8'd196:  coeff = 12'sd3;
8'd197:  coeff = 12'sd2;
8'd198:  coeff = 12'sd2;
8'd199:  coeff = 12'sd1;
8'd200:  coeff = 12'sd0;
8'd201:  coeff = -12'sd1;
8'd202:  coeff = -12'sd1;
8'd203:  coeff = -12'sd2;
8'd204:  coeff = -12'sd2;
8'd205:  coeff = -12'sd2;
8'd206:  coeff = -12'sd3;
8'd207:  coeff = -12'sd3;
8'd208:  coeff = -12'sd2;
8'd209:  coeff = -12'sd2;
8'd210:  coeff = -12'sd2;
8'd211:  coeff = -12'sd2;
8'd212:  coeff = -12'sd1;
8'd213:  coeff = -12'sd1;
8'd214:  coeff = 12'sd0;
8'd215:  coeff = 12'sd0;
8'd216:  coeff = 12'sd0;
8'd217:  coeff = 12'sd1;
8'd218:  coeff = 12'sd1;
8'd219:  coeff = 12'sd1;
8'd220:  coeff = 12'sd1;
8'd221:  coeff = 12'sd1;
8'd222:  coeff = 12'sd1;
8'd223:  coeff = 12'sd1;
8'd224:  coeff = 12'sd1;
8'd225:  coeff = 12'sd1;
8'd226:  coeff = 12'sd1;
```

```
        8'd227:  coeff = 12'sd1;
        8'd228:  coeff = 12'sd0;
        8'd229:  coeff = 12'sd0;
        8'd230:  coeff = 12'sd0;
        8'd231:  coeff = 12'sd0;
        8'd232:  coeff = 12'sd0;
        8'd233:  coeff = 12'sd0;
        8'd234:  coeff = -12'sd1;
        8'd235:  coeff = -12'sd1;
        8'd236:  coeff = -12'sd1;
        8'd237:  coeff = -12'sd1;
        8'd238:  coeff = -12'sd1;
        8'd239:  coeff = -12'sd1;
        8'd240:  coeff = 12'sd0;
        8'd241:  coeff = 12'sd0;
        8'd242:  coeff = 12'sd0;
        8'd243:  coeff = 12'sd0;
        8'd244:  coeff = 12'sd0;
        8'd245:  coeff = 12'sd0;
        8'd246:  coeff = 12'sd0;
        8'd247:  coeff = 12'sd0;
        8'd248:  coeff = 12'sd0;
        8'd249:  coeff = 12'sd0;
        8'd250:  coeff = 12'sd0;
      default: coeff = 12'hXXX;
    endcase
endmodule

module firlow7000(
  input wire clock,reset,ready,
  input wire signed [17:0] x,
  output reg signed [17:0] y
);
        reg signed [17:0]big_array[250:0]; //change big_array[x:0] x to coeff #
        reg [7:0]index = 0;
        reg [7:0]offset = 0;
        wire signed [11:0]coeff;
        reg signed [29:0]accumulator = 0;
        reg go = 0;

  always @(posedge clock) begin
                if (reset) begin //reinitialize everything
                        offset <= 0;
                        index <= 0;
                        accumulator <= 0;
                        go <= 0;end
                else if (ready) begin //on ready insert x into big_array and reset index
                        big_array[offset] <= x;
                        index <= 0;
                        go <= 1;end
                else if (go == 1) begin //while accumulator is accumulating
                        if (index == 8'd250) begin //change 8'dx x to coeff #
                                index <= 0;
                                go <= 0;
                                offset <= offset + 1;
                                y <= accumulator[29:12]<<<1;
                                accumulator <= 0;
                                end
                        else begin //accumulate procedure
                                index <= index + 1;
                                accumulator <= accumulator + coeff*big_array[offset - index];end
                        end
                else;
  end
  firlow7000coeffs coeffs(index,coeff);
endmodule

module firlow7000coeffs(
  input wire [7:0] index,
  output reg signed [11:0] coeff
);
  // tools will turn this into a 31x10 ROM
  always @(index)
    case (index)
        8'd0:  coeff = 12'sd0;
        8'd1:  coeff = 12'sd0;
        8'd2:  coeff = 12'sd0;
        8'd3:  coeff = 12'sd0;
        8'd4:  coeff = 12'sd0;
        8'd5:  coeff = 12'sd0;
        8'd6:  coeff = 12'sd0;
        8'd7:  coeff = 12'sd0;
        8'd8:  coeff = 12'sd0;
        8'd9:  coeff = 12'sd0;
        8'd10:  coeff = -12'sd1;
        8'd11:  coeff = 12'sd0;
        8'd12:  coeff = 12'sd0;
        8'd13:  coeff = 12'sd1;
        8'd14:  coeff = 12'sd1;
        8'd15:  coeff = 12'sd0;
        8'd16:  coeff = 12'sd0;
        8'd17:  coeff = -12'sd1;
```

```
8'd18:  coeff = 12'sd0;
8'd19:  coeff = 12'sd0;
8'd20:  coeff = 12'sd1;
8'd21:  coeff = 12'sd1;
8'd22:  coeff = 12'sd0;
8'd23:  coeff = -12'sd1;
8'd24:  coeff = -12'sd1;
8'd25:  coeff = -12'sd1;
8'd26:  coeff = 12'sd0;
8'd27:  coeff = 12'sd1;
8'd28:  coeff = 12'sd1;
8'd29:  coeff = 12'sd0;
8'd30:  coeff = -12'sd1;
8'd31:  coeff = -12'sd1;
8'd32:  coeff = -12'sd1;
8'd33:  coeff = 12'sd1;
8'd34:  coeff = 12'sd2;
8'd35:  coeff = 12'sd1;
8'd36:  coeff = 12'sd0;
8'd37:  coeff = -12'sd2;
8'd38:  coeff = -12'sd2;
8'd39:  coeff = -12'sd1;
8'd40:  coeff = 12'sd1;
8'd41:  coeff = 12'sd2;
8'd42:  coeff = 12'sd1;
8'd43:  coeff = -12'sd1;
8'd44:  coeff = -12'sd2;
8'd45:  coeff = -12'sd2;
8'd46:  coeff = 12'sd0;
8'd47:  coeff = 12'sd2;
8'd48:  coeff = 12'sd3;
8'd49:  coeff = 12'sd2;
8'd50:  coeff = -12'sd1;
8'd51:  coeff = -12'sd3;
8'd52:  coeff = -12'sd3;
8'd53:  coeff = 12'sd0;
8'd54:  coeff = 12'sd3;
8'd55:  coeff = 12'sd4;
8'd56:  coeff = 12'sd2;
8'd57:  coeff = -12'sd2;
8'd58:  coeff = -12'sd5;
8'd59:  coeff = -12'sd3;
8'd60:  coeff = 12'sd1;
8'd61:  coeff = 12'sd5;
8'd62:  coeff = 12'sd5;
8'd63:  coeff = 12'sd1;
8'd64:  coeff = -12'sd4;
8'd65:  coeff = -12'sd6;
8'd66:  coeff = -12'sd4;
8'd67:  coeff = 12'sd2;
8'd68:  coeff = 12'sd6;
8'd69:  coeff = 12'sd6;
8'd70:  coeff = 12'sd1;
8'd71:  coeff = -12'sd5;
8'd72:  coeff = -12'sd8;
8'd73:  coeff = -12'sd4;
8'd74:  coeff = 12'sd3;
8'd75:  coeff = 12'sd9;
8'd76:  coeff = 12'sd7;
8'd77:  coeff = 12'sd0;
8'd78:  coeff = -12'sd8;
8'd79:  coeff = -12'sd10;
8'd80:  coeff = -12'sd4;
8'd81:  coeff = 12'sd6;
8'd82:  coeff = 12'sd11;
8'd83:  coeff = 12'sd8;
8'd84:  coeff = -12'sd2;
8'd85:  coeff = -12'sd11;
8'd86:  coeff = -12'sd12;
8'd87:  coeff = -12'sd4;
8'd88:  coeff = 12'sd9;
8'd89:  coeff = 12'sd15;
8'd90:  coeff = 12'sd9;
8'd91:  coeff = -12'sd4;
8'd92:  coeff = -12'sd16;
8'd93:  coeff = -12'sd15;
8'd94:  coeff = -12'sd2;
8'd95:  coeff = 12'sd13;
8'd96:  coeff = 12'sd20;
8'd97:  coeff = 12'sd10;
8'd98:  coeff = -12'sd8;
8'd99:  coeff = -12'sd22;
8'd100:  coeff = -12'sd19;
8'd101:  coeff = 12'sd0;
8'd102:  coeff = 12'sd21;
8'd103:  coeff = 12'sd27;
8'd104:  coeff = 12'sd11;
8'd105:  coeff = -12'sd15;
8'd106:  coeff = -12'sd32;
8'd107:  coeff = -12'sd24;
8'd108:  coeff = 12'sd5;
8'd109:  coeff = 12'sd34;
```

```
8'd110:  coeff = 12'sd39;
8'd111:  coeff = 12'sd12;
8'd112:  coeff = -12'sd30;
8'd113:  coeff = -12'sd53;
8'd114:  coeff = -12'sd35;
8'd115:  coeff = 12'sd17;
8'd116:  coeff = 12'sd66;
8'd117:  coeff = 12'sd70;
8'd118:  coeff = 12'sd12;
8'd119:  coeff = -12'sd76;
8'd120:  coeff = -12'sd129;
8'd121:  coeff = -12'sd81;
8'd122:  coeff = 12'sd83;
8'd123:  coeff = 12'sd315;
8'd124:  coeff = 12'sd517;
8'd125:  coeff = 12'sd597;
8'd126:  coeff = 12'sd517;
8'd127:  coeff = 12'sd315;
8'd128:  coeff = 12'sd83;
8'd129:  coeff = -12'sd81;
8'd130:  coeff = -12'sd129;
8'd131:  coeff = -12'sd76;
8'd132:  coeff = 12'sd12;
8'd133:  coeff = 12'sd70;
8'd134:  coeff = 12'sd66;
8'd135:  coeff = 12'sd17;
8'd136:  coeff = -12'sd35;
8'd137:  coeff = -12'sd53;
8'd138:  coeff = -12'sd30;
8'd139:  coeff = 12'sd12;
8'd140:  coeff = 12'sd39;
8'd141:  coeff = 12'sd34;
8'd142:  coeff = 12'sd5;
8'd143:  coeff = -12'sd24;
8'd144:  coeff = -12'sd32;
8'd145:  coeff = -12'sd15;
8'd146:  coeff = 12'sd11;
8'd147:  coeff = 12'sd27;
8'd148:  coeff = 12'sd21;
8'd149:  coeff = 12'sd0;
8'd150:  coeff = -12'sd19;
8'd151:  coeff = -12'sd22;
8'd152:  coeff = -12'sd8;
8'd153:  coeff = 12'sd10;
8'd154:  coeff = 12'sd20;
8'd155:  coeff = 12'sd13;
8'd156:  coeff = -12'sd2;
8'd157:  coeff = -12'sd15;
8'd158:  coeff = -12'sd16;
8'd159:  coeff = -12'sd4;
8'd160:  coeff = 12'sd9;
8'd161:  coeff = 12'sd15;
8'd162:  coeff = 12'sd9;
8'd163:  coeff = -12'sd4;
8'd164:  coeff = -12'sd12;
8'd165:  coeff = -12'sd11;
8'd166:  coeff = -12'sd2;
8'd167:  coeff = 12'sd8;
8'd168:  coeff = 12'sd11;
8'd169:  coeff = 12'sd6;
8'd170:  coeff = -12'sd4;
8'd171:  coeff = -12'sd10;
8'd172:  coeff = -12'sd8;
8'd173:  coeff = 12'sd0;
8'd174:  coeff = 12'sd7;
8'd175:  coeff = 12'sd9;
8'd176:  coeff = 12'sd3;
8'd177:  coeff = -12'sd4;
8'd178:  coeff = -12'sd8;
8'd179:  coeff = -12'sd5;
8'd180:  coeff = 12'sd1;
8'd181:  coeff = 12'sd6;
8'd182:  coeff = 12'sd6;
8'd183:  coeff = 12'sd2;
8'd184:  coeff = -12'sd4;
8'd185:  coeff = -12'sd6;
8'd186:  coeff = -12'sd4;
8'd187:  coeff = 12'sd1;
8'd188:  coeff = 12'sd5;
8'd189:  coeff = 12'sd5;
8'd190:  coeff = 12'sd1;
8'd191:  coeff = -12'sd3;
8'd192:  coeff = -12'sd5;
8'd193:  coeff = -12'sd2;
8'd194:  coeff = 12'sd2;
8'd195:  coeff = 12'sd4;
8'd196:  coeff = 12'sd3;
8'd197:  coeff = 12'sd0;
8'd198:  coeff = -12'sd3;
8'd199:  coeff = -12'sd3;
8'd200:  coeff = -12'sd1;
8'd201:  coeff = 12'sd2;
```

```
         8'd202:  coeff = 12'sd3;
         8'd203:  coeff = 12'sd2;
         8'd204:  coeff = 12'sd0;
         8'd205:  coeff = -12'sd2;
         8'd206:  coeff = -12'sd2;
         8'd207:  coeff = -12'sd1;
         8'd208:  coeff = 12'sd1;
         8'd209:  coeff = 12'sd2;
         8'd210:  coeff = 12'sd1;
         8'd211:  coeff = -12'sd1;
         8'd212:  coeff = -12'sd2;
         8'd213:  coeff = -12'sd2;
         8'd214:  coeff = 12'sd0;
         8'd215:  coeff = 12'sd1;
         8'd216:  coeff = 12'sd2;
         8'd217:  coeff = 12'sd1;
         8'd218:  coeff = -12'sd1;
         8'd219:  coeff = -12'sd1;
         8'd220:  coeff = -12'sd1;
         8'd221:  coeff = 12'sd0;
         8'd222:  coeff = 12'sd1;
         8'd223:  coeff = 12'sd1;
         8'd224:  coeff = 12'sd0;
         8'd225:  coeff = -12'sd1;
         8'd226:  coeff = -12'sd1;
         8'd227:  coeff = -12'sd1;
         8'd228:  coeff = 12'sd0;
         8'd229:  coeff = 12'sd1;
         8'd230:  coeff = 12'sd1;
         8'd231:  coeff = 12'sd0;
         8'd232:  coeff = 12'sd0;
         8'd233:  coeff = -12'sd1;
         8'd234:  coeff = 12'sd0;
         8'd235:  coeff = 12'sd0;
         8'd236:  coeff = 12'sd1;
         8'd237:  coeff = 12'sd1;
         8'd238:  coeff = 12'sd0;
         8'd239:  coeff = 12'sd0;
         8'd240:  coeff = -12'sd1;
         8'd241:  coeff = 12'sd0;
         8'd242:  coeff = 12'sd0;
         8'd243:  coeff = 12'sd0;
         8'd244:  coeff = 12'sd0;
         8'd245:  coeff = 12'sd0;
         8'd246:  coeff = 12'sd0;
         8'd247:  coeff = 12'sd0;
         8'd248:  coeff = 12'sd0;
         8'd249:  coeff = 12'sd0;
         8'd250:  coeff = 12'sd0;
       default: coeff = 12'hXXX;
     endcase
endmodule // firlow7000coeffs

module mybram_saw(input clock,
   input wire [7:0] index,
   output reg signed [8:0] y);

   always @(index)
      case (index)
         8'd0:  y = -9'sd255;
         8'd1:  y = -9'sd253;
         8'd2:  y = -9'sd251;
         8'd3:  y = -9'sd249;
         8'd4:  y = -9'sd247;
         8'd5:  y = -9'sd245;
         8'd6:  y = -9'sd243;
         8'd7:  y = -9'sd241;
         8'd8:  y = -9'sd239;
         8'd9:  y = -9'sd237;
         8'd10:  y = -9'sd235;
         8'd11:  y = -9'sd233;
         8'd12:  y = -9'sd231;
         8'd13:  y = -9'sd229;
         8'd14:  y = -9'sd227;
         8'd15:  y = -9'sd225;
         8'd16:  y = -9'sd223;
         8'd17:  y = -9'sd221;
         8'd18:  y = -9'sd219;
         8'd19:  y = -9'sd217;
         8'd20:  y = -9'sd215;
         8'd21:  y = -9'sd213;
         8'd22:  y = -9'sd211;
         8'd23:  y = -9'sd209;
         8'd24:  y = -9'sd207;
         8'd25:  y = -9'sd205;
         8'd26:  y = -9'sd203;
         8'd27:  y = -9'sd201;
         8'd28:  y = -9'sd199;
         8'd29:  y = -9'sd197;
         8'd30:  y = -9'sd195;
         8'd31:  y = -9'sd193;
         8'd32:  y = -9'sd191;
```

```
8'd33:  y = -9'sd189;
8'd34:  y = -9'sd187;
8'd35:  y = -9'sd185;
8'd36:  y = -9'sd183;
8'd37:  y = -9'sd181;
8'd38:  y = -9'sd179;
8'd39:  y = -9'sd177;
8'd40:  y = -9'sd175;
8'd41:  y = -9'sd173;
8'd42:  y = -9'sd171;
8'd43:  y = -9'sd169;
8'd44:  y = -9'sd167;
8'd45:  y = -9'sd165;
8'd46:  y = -9'sd163;
8'd47:  y = -9'sd161;
8'd48:  y = -9'sd159;
8'd49:  y = -9'sd157;
8'd50:  y = -9'sd155;
8'd51:  y = -9'sd153;
8'd52:  y = -9'sd151;
8'd53:  y = -9'sd149;
8'd54:  y = -9'sd147;
8'd55:  y = -9'sd145;
8'd56:  y = -9'sd143;
8'd57:  y = -9'sd141;
8'd58:  y = -9'sd139;
8'd59:  y = -9'sd137;
8'd60:  y = -9'sd135;
8'd61:  y = -9'sd133;
8'd62:  y = -9'sd131;
8'd63:  y = -9'sd129;
8'd64:  y = -9'sd127;
8'd65:  y = -9'sd125;
8'd66:  y = -9'sd123;
8'd67:  y = -9'sd121;
8'd68:  y = -9'sd119;
8'd69:  y = -9'sd117;
8'd70:  y = -9'sd115;
8'd71:  y = -9'sd113;
8'd72:  y = -9'sd111;
8'd73:  y = -9'sd109;
8'd74:  y = -9'sd107;
8'd75:  y = -9'sd105;
8'd76:  y = -9'sd103;
8'd77:  y = -9'sd101;
8'd78:  y = -9'sd99;
8'd79:  y = -9'sd97;
8'd80:  y = -9'sd95;
8'd81:  y = -9'sd93;
8'd82:  y = -9'sd91;
8'd83:  y = -9'sd89;
8'd84:  y = -9'sd87;
8'd85:  y = -9'sd85;
8'd86:  y = -9'sd83;
8'd87:  y = -9'sd81;
8'd88:  y = -9'sd79;
8'd89:  y = -9'sd77;
8'd90:  y = -9'sd75;
8'd91:  y = -9'sd73;
8'd92:  y = -9'sd71;
8'd93:  y = -9'sd69;
8'd94:  y = -9'sd67;
8'd95:  y = -9'sd65;
8'd96:  y = -9'sd63;
8'd97:  y = -9'sd61;
8'd98:  y = -9'sd59;
8'd99:  y = -9'sd57;
8'd100:  y = -9'sd55;
8'd101:  y = -9'sd53;
8'd102:  y = -9'sd51;
8'd103:  y = -9'sd49;
8'd104:  y = -9'sd47;
8'd105:  y = -9'sd45;
8'd106:  y = -9'sd43;
8'd107:  y = -9'sd41;
8'd108:  y = -9'sd39;
8'd109:  y = -9'sd37;
8'd110:  y = -9'sd35;
8'd111:  y = -9'sd33;
8'd112:  y = -9'sd31;
8'd113:  y = -9'sd29;
8'd114:  y = -9'sd27;
8'd115:  y = -9'sd25;
8'd116:  y = -9'sd23;
8'd117:  y = -9'sd21;
8'd118:  y = -9'sd19;
8'd119:  y = -9'sd17;
8'd120:  y = -9'sd15;
8'd121:  y = -9'sd13;
8'd122:  y = -9'sd11;
8'd123:  y = -9'sd9;
8'd124:  y = -9'sd7;
```

```
8'd125:  y = -9'sd5;
8'd126:  y = -9'sd3;
8'd127:  y = -9'sd1;
8'd128:  y = 9'sd1;
8'd129:  y = 9'sd3;
8'd130:  y = 9'sd5;
8'd131:  y = 9'sd7;
8'd132:  y = 9'sd9;
8'd133:  y = 9'sd11;
8'd134:  y = 9'sd13;
8'd135:  y = 9'sd15;
8'd136:  y = 9'sd17;
8'd137:  y = 9'sd19;
8'd138:  y = 9'sd21;
8'd139:  y = 9'sd23;
8'd140:  y = 9'sd25;
8'd141:  y = 9'sd27;
8'd142:  y = 9'sd29;
8'd143:  y = 9'sd31;
8'd144:  y = 9'sd33;
8'd145:  y = 9'sd35;
8'd146:  y = 9'sd37;
8'd147:  y = 9'sd39;
8'd148:  y = 9'sd41;
8'd149:  y = 9'sd43;
8'd150:  y = 9'sd45;
8'd151:  y = 9'sd47;
8'd152:  y = 9'sd49;
8'd153:  y = 9'sd51;
8'd154:  y = 9'sd53;
8'd155:  y = 9'sd55;
8'd156:  y = 9'sd57;
8'd157:  y = 9'sd59;
8'd158:  y = 9'sd61;
8'd159:  y = 9'sd63;
8'd160:  y = 9'sd65;
8'd161:  y = 9'sd67;
8'd162:  y = 9'sd69;
8'd163:  y = 9'sd71;
8'd164:  y = 9'sd73;
8'd165:  y = 9'sd75;
8'd166:  y = 9'sd77;
8'd167:  y = 9'sd79;
8'd168:  y = 9'sd81;
8'd169:  y = 9'sd83;
8'd170:  y = 9'sd85;
8'd171:  y = 9'sd87;
8'd172:  y = 9'sd89;
8'd173:  y = 9'sd91;
8'd174:  y = 9'sd93;
8'd175:  y = 9'sd95;
8'd176:  y = 9'sd97;
8'd177:  y = 9'sd99;
8'd178:  y = 9'sd101;
8'd179:  y = 9'sd103;
8'd180:  y = 9'sd105;
8'd181:  y = 9'sd107;
8'd182:  y = 9'sd109;
8'd183:  y = 9'sd111;
8'd184:  y = 9'sd113;
8'd185:  y = 9'sd115;
8'd186:  y = 9'sd117;
8'd187:  y = 9'sd119;
8'd188:  y = 9'sd121;
8'd189:  y = 9'sd123;
8'd190:  y = 9'sd125;
8'd191:  y = 9'sd127;
8'd192:  y = 9'sd129;
8'd193:  y = 9'sd131;
8'd194:  y = 9'sd133;
8'd195:  y = 9'sd135;
8'd196:  y = 9'sd137;
8'd197:  y = 9'sd139;
8'd198:  y = 9'sd141;
8'd199:  y = 9'sd143;
8'd200:  y = 9'sd145;
8'd201:  y = 9'sd147;
8'd202:  y = 9'sd149;
8'd203:  y = 9'sd151;
8'd204:  y = 9'sd153;
8'd205:  y = 9'sd155;
8'd206:  y = 9'sd157;
8'd207:  y = 9'sd159;
8'd208:  y = 9'sd161;
8'd209:  y = 9'sd163;
8'd210:  y = 9'sd165;
8'd211:  y = 9'sd167;
8'd212:  y = 9'sd169;
8'd213:  y = 9'sd171;
8'd214:  y = 9'sd173;
8'd215:  y = 9'sd175;
8'd216:  y = 9'sd177;
```

```
         8'd217:  y = 9'sd179;
         8'd218:  y = 9'sd181;
         8'd219:  y = 9'sd183;
         8'd220:  y = 9'sd185;
         8'd221:  y = 9'sd187;
         8'd222:  y = 9'sd189;
         8'd223:  y = 9'sd191;
         8'd224:  y = 9'sd193;
         8'd225:  y = 9'sd195;
         8'd226:  y = 9'sd197;
         8'd227:  y = 9'sd199;
         8'd228:  y = 9'sd201;
         8'd229:  y = 9'sd203;
         8'd230:  y = 9'sd205;
         8'd231:  y = 9'sd207;
         8'd232:  y = 9'sd209;
         8'd233:  y = 9'sd211;
         8'd234:  y = 9'sd213;
         8'd235:  y = 9'sd215;
         8'd236:  y = 9'sd217;
         8'd237:  y = 9'sd219;
         8'd238:  y = 9'sd221;
         8'd239:  y = 9'sd223;
         8'd240:  y = 9'sd225;
         8'd241:  y = 9'sd227;
         8'd242:  y = 9'sd229;
         8'd243:  y = 9'sd231;
         8'd244:  y = 9'sd233;
         8'd245:  y = 9'sd235;
         8'd246:  y = 9'sd237;
         8'd247:  y = 9'sd239;
         8'd248:  y = 9'sd241;
         8'd249:  y = 9'sd243;
         8'd250:  y = 9'sd245;
         8'd251:  y = 9'sd247;
         8'd252:  y = 9'sd249;
         8'd253:  y = 9'sd251;
         8'd254:  y = 9'sd253;
         8'd255:  y = 9'sd255;
      default: y = 9'sbXXXXXXXXX;
   endcase
endmodule
module mybram_sin(input clock,
                  input wire [7:0] index,
                  output reg signed [8:0] y);

   always @(index)
     case (index)
       8'd0:  y = 9'sd0;
       8'd1:  y = 9'sd6;
       8'd2:  y = 9'sd12;
       8'd3:  y = 9'sd18;
       8'd4:  y = 9'sd24;
       8'd5:  y = 9'sd31;
       8'd6:  y = 9'sd37;
       8'd7:  y = 9'sd43;
       8'd8:  y = 9'sd49;
       8'd9:  y = 9'sd55;
       8'd10:  y = 9'sd61;
       8'd11:  y = 9'sd68;
       8'd12:  y = 9'sd74;
       8'd13:  y = 9'sd79;
       8'd14:  y = 9'sd85;
       8'd15:  y = 9'sd91;
       8'd16:  y = 9'sd97;
       8'd17:  y = 9'sd103;
       8'd18:  y = 9'sd109;
       8'd19:  y = 9'sd114;
       8'd20:  y = 9'sd120;
       8'd21:  y = 9'sd125;
       8'd22:  y = 9'sd131;
       8'd23:  y = 9'sd136;
       8'd24:  y = 9'sd141;
       8'd25:  y = 9'sd146;
       8'd26:  y = 9'sd151;
       8'd27:  y = 9'sd156;
       8'd28:  y = 9'sd161;
       8'd29:  y = 9'sd166;
       8'd30:  y = 9'sd171;
       8'd31:  y = 9'sd175;
       8'd32:  y = 9'sd180;
       8'd33:  y = 9'sd184;
       8'd34:  y = 9'sd188;
       8'd35:  y = 9'sd193;
       8'd36:  y = 9'sd197;
       8'd37:  y = 9'sd201;
       8'd38:  y = 9'sd204;
       8'd39:  y = 9'sd208;
       8'd40:  y = 9'sd212;
       8'd41:  y = 9'sd215;
       8'd42:  y = 9'sd218;
       8'd43:  y = 9'sd221;
```

```
8'd44:  y = 9'sd224;
8'd45:  y = 9'sd227;
8'd46:  y = 9'sd230;
8'd47:  y = 9'sd233;
8'd48:  y = 9'sd235;
8'd49:  y = 9'sd237;
8'd50:  y = 9'sd240;
8'd51:  y = 9'sd242;
8'd52:  y = 9'sd244;
8'd53:  y = 9'sd245;
8'd54:  y = 9'sd247;
8'd55:  y = 9'sd248;
8'd56:  y = 9'sd250;
8'd57:  y = 9'sd251;
8'd58:  y = 9'sd252;
8'd59:  y = 9'sd253;
8'd60:  y = 9'sd253;
8'd61:  y = 9'sd254;
8'd62:  y = 9'sd254;
8'd63:  y = 9'sd254;
8'd64:  y = 9'sd255;
8'd65:  y = 9'sd254;
8'd66:  y = 9'sd254;
8'd67:  y = 9'sd254;
8'd68:  y = 9'sd253;
8'd69:  y = 9'sd253;
8'd70:  y = 9'sd252;
8'd71:  y = 9'sd251;
8'd72:  y = 9'sd250;
8'd73:  y = 9'sd248;
8'd74:  y = 9'sd247;
8'd75:  y = 9'sd245;
8'd76:  y = 9'sd244;
8'd77:  y = 9'sd242;
8'd78:  y = 9'sd240;
8'd79:  y = 9'sd237;
8'd80:  y = 9'sd235;
8'd81:  y = 9'sd233;
8'd82:  y = 9'sd230;
8'd83:  y = 9'sd227;
8'd84:  y = 9'sd224;
8'd85:  y = 9'sd221;
8'd86:  y = 9'sd218;
8'd87:  y = 9'sd215;
8'd88:  y = 9'sd212;
8'd89:  y = 9'sd208;
8'd90:  y = 9'sd204;
8'd91:  y = 9'sd201;
8'd92:  y = 9'sd197;
8'd93:  y = 9'sd193;
8'd94:  y = 9'sd188;
8'd95:  y = 9'sd184;
8'd96:  y = 9'sd180;
8'd97:  y = 9'sd175;
8'd98:  y = 9'sd171;
8'd99:  y = 9'sd166;
8'd100:  y = 9'sd161;
8'd101:  y = 9'sd156;
8'd102:  y = 9'sd151;
8'd103:  y = 9'sd146;
8'd104:  y = 9'sd141;
8'd105:  y = 9'sd136;
8'd106:  y = 9'sd131;
8'd107:  y = 9'sd125;
8'd108:  y = 9'sd120;
8'd109:  y = 9'sd114;
8'd110:  y = 9'sd109;
8'd111:  y = 9'sd103;
8'd112:  y = 9'sd97;
8'd113:  y = 9'sd91;
8'd114:  y = 9'sd85;
8'd115:  y = 9'sd79;
8'd116:  y = 9'sd74;
8'd117:  y = 9'sd68;
8'd118:  y = 9'sd61;
8'd119:  y = 9'sd55;
8'd120:  y = 9'sd49;
8'd121:  y = 9'sd43;
8'd122:  y = 9'sd37;
8'd123:  y = 9'sd31;
8'd124:  y = 9'sd24;
8'd125:  y = 9'sd18;
8'd126:  y = 9'sd12;
8'd127:  y = 9'sd6;
8'd128:  y = 9'sd0;
8'd129:  y = -9'sd6;
8'd130:  y = -9'sd12;
8'd131:  y = -9'sd18;
8'd132:  y = -9'sd24;
8'd133:  y = -9'sd31;
8'd134:  y = -9'sd37;
8'd135:  y = -9'sd43;
```

```
8'd136:  y = -9'sd49;
8'd137:  y = -9'sd55;
8'd138:  y = -9'sd61;
8'd139:  y = -9'sd68;
8'd140:  y = -9'sd74;
8'd141:  y = -9'sd79;
8'd142:  y = -9'sd85;
8'd143:  y = -9'sd91;
8'd144:  y = -9'sd97;
8'd145:  y = -9'sd103;
8'd146:  y = -9'sd109;
8'd147:  y = -9'sd114;
8'd148:  y = -9'sd120;
8'd149:  y = -9'sd125;
8'd150:  y = -9'sd131;
8'd151:  y = -9'sd136;
8'd152:  y = -9'sd141;
8'd153:  y = -9'sd146;
8'd154:  y = -9'sd151;
8'd155:  y = -9'sd156;
8'd156:  y = -9'sd161;
8'd157:  y = -9'sd166;
8'd158:  y = -9'sd171;
8'd159:  y = -9'sd175;
8'd160:  y = -9'sd180;
8'd161:  y = -9'sd184;
8'd162:  y = -9'sd188;
8'd163:  y = -9'sd193;
8'd164:  y = -9'sd197;
8'd165:  y = -9'sd201;
8'd166:  y = -9'sd204;
8'd167:  y = -9'sd208;
8'd168:  y = -9'sd212;
8'd169:  y = -9'sd215;
8'd170:  y = -9'sd218;
8'd171:  y = -9'sd221;
8'd172:  y = -9'sd224;
8'd173:  y = -9'sd227;
8'd174:  y = -9'sd230;
8'd175:  y = -9'sd233;
8'd176:  y = -9'sd235;
8'd177:  y = -9'sd237;
8'd178:  y = -9'sd240;
8'd179:  y = -9'sd242;
8'd180:  y = -9'sd244;
8'd181:  y = -9'sd245;
8'd182:  y = -9'sd247;
8'd183:  y = -9'sd248;
8'd184:  y = -9'sd250;
8'd185:  y = -9'sd251;
8'd186:  y = -9'sd252;
8'd187:  y = -9'sd253;
8'd188:  y = -9'sd253;
8'd189:  y = -9'sd254;
8'd190:  y = -9'sd254;
8'd191:  y = -9'sd254;
8'd192:  y = -9'sd255;
8'd193:  y = -9'sd254;
8'd194:  y = -9'sd254;
8'd195:  y = -9'sd254;
8'd196:  y = -9'sd253;
8'd197:  y = -9'sd253;
8'd198:  y = -9'sd252;
8'd199:  y = -9'sd251;
8'd200:  y = -9'sd250;
8'd201:  y = -9'sd248;
8'd202:  y = -9'sd247;
8'd203:  y = -9'sd245;
8'd204:  y = -9'sd244;
8'd205:  y = -9'sd242;
8'd206:  y = -9'sd240;
8'd207:  y = -9'sd237;
8'd208:  y = -9'sd235;
8'd209:  y = -9'sd233;
8'd210:  y = -9'sd230;
8'd211:  y = -9'sd227;
8'd212:  y = -9'sd224;
8'd213:  y = -9'sd221;
8'd214:  y = -9'sd218;
8'd215:  y = -9'sd215;
8'd216:  y = -9'sd212;
8'd217:  y = -9'sd208;
8'd218:  y = -9'sd204;
8'd219:  y = -9'sd201;
8'd220:  y = -9'sd197;
8'd221:  y = -9'sd193;
8'd222:  y = -9'sd188;
8'd223:  y = -9'sd184;
8'd224:  y = -9'sd180;
8'd225:  y = -9'sd175;
8'd226:  y = -9'sd171;
8'd227:  y = -9'sd166;
```

```
        8'd228:  y = -9'sd161;
        8'd229:  y = -9'sd156;
        8'd230:  y = -9'sd151;
        8'd231:  y = -9'sd146;
        8'd232:  y = -9'sd141;
        8'd233:  y = -9'sd136;
        8'd234:  y = -9'sd131;
        8'd235:  y = -9'sd125;
        8'd236:  y = -9'sd120;
        8'd237:  y = -9'sd114;
        8'd238:  y = -9'sd109;
        8'd239:  y = -9'sd103;
        8'd240:  y = -9'sd97;
        8'd241:  y = -9'sd91;
        8'd242:  y = -9'sd85;
        8'd243:  y = -9'sd79;
        8'd244:  y = -9'sd74;
        8'd245:  y = -9'sd68;
        8'd246:  y = -9'sd61;
        8'd247:  y = -9'sd55;
        8'd248:  y = -9'sd49;
        8'd249:  y = -9'sd43;
        8'd250:  y = -9'sd37;
        8'd251:  y = -9'sd31;
        8'd252:  y = -9'sd24;
        8'd253:  y = -9'sd18;
        8'd254:  y = -9'sd12;
        8'd255:  y = -9'sd6;
        default: y = 9'sbXXXXXXXXX;
      endcase
endmodule
module mybram_triangle(input clock,
   input wire [7:0] index,
   output reg signed [8:0] y);

   always @(index)
      case (index)
         8'd0:  y = -9'sd254;
         8'd1:  y = -9'sd250;
         8'd2:  y = -9'sd246;
         8'd3:  y = -9'sd242;
         8'd4:  y = -9'sd238;
         8'd5:  y = -9'sd234;
         8'd6:  y = -9'sd230;
         8'd7:  y = -9'sd226;
         8'd8:  y = -9'sd222;
         8'd9:  y = -9'sd218;
         8'd10:  y = -9'sd214;
         8'd11:  y = -9'sd210;
         8'd12:  y = -9'sd206;
         8'd13:  y = -9'sd202;
         8'd14:  y = -9'sd198;
         8'd15:  y = -9'sd194;
         8'd16:  y = -9'sd190;
         8'd17:  y = -9'sd186;
         8'd18:  y = -9'sd182;
         8'd19:  y = -9'sd178;
         8'd20:  y = -9'sd174;
         8'd21:  y = -9'sd170;
         8'd22:  y = -9'sd166;
         8'd23:  y = -9'sd162;
         8'd24:  y = -9'sd158;
         8'd25:  y = -9'sd154;
         8'd26:  y = -9'sd150;
         8'd27:  y = -9'sd146;
         8'd28:  y = -9'sd142;
         8'd29:  y = -9'sd138;
         8'd30:  y = -9'sd134;
         8'd31:  y = -9'sd130;
         8'd32:  y = -9'sd126;
         8'd33:  y = -9'sd122;
         8'd34:  y = -9'sd118;
         8'd35:  y = -9'sd114;
         8'd36:  y = -9'sd110;
         8'd37:  y = -9'sd106;
         8'd38:  y = -9'sd102;
         8'd39:  y = -9'sd98;
         8'd40:  y = -9'sd94;
         8'd41:  y = -9'sd90;
         8'd42:  y = -9'sd86;
         8'd43:  y = -9'sd82;
         8'd44:  y = -9'sd78;
         8'd45:  y = -9'sd74;
         8'd46:  y = -9'sd70;
         8'd47:  y = -9'sd66;
         8'd48:  y = -9'sd62;
         8'd49:  y = -9'sd58;
         8'd50:  y = -9'sd54;
         8'd51:  y = -9'sd50;
         8'd52:  y = -9'sd46;
         8'd53:  y = -9'sd42;
         8'd54:  y = -9'sd38;
```

```
8'd55:  y = -9'sd34;
8'd56:  y = -9'sd30;
8'd57:  y = -9'sd26;
8'd58:  y = -9'sd22;
8'd59:  y = -9'sd18;
8'd60:  y = -9'sd14;
8'd61:  y = -9'sd10;
8'd62:  y = -9'sd6;
8'd63:  y = -9'sd2;
8'd64:  y = 9'sd2;
8'd65:  y = 9'sd6;
8'd66:  y = 9'sd10;
8'd67:  y = 9'sd14;
8'd68:  y = 9'sd18;
8'd69:  y = 9'sd22;
8'd70:  y = 9'sd26;
8'd71:  y = 9'sd30;
8'd72:  y = 9'sd34;
8'd73:  y = 9'sd38;
8'd74:  y = 9'sd42;
8'd75:  y = 9'sd46;
8'd76:  y = 9'sd50;
8'd77:  y = 9'sd54;
8'd78:  y = 9'sd58;
8'd79:  y = 9'sd62;
8'd80:  y = 9'sd66;
8'd81:  y = 9'sd70;
8'd82:  y = 9'sd74;
8'd83:  y = 9'sd78;
8'd84:  y = 9'sd82;
8'd85:  y = 9'sd86;
8'd86:  y = 9'sd90;
8'd87:  y = 9'sd94;
8'd88:  y = 9'sd98;
8'd89:  y = 9'sd102;
8'd90:  y = 9'sd106;
8'd91:  y = 9'sd110;
8'd92:  y = 9'sd114;
8'd93:  y = 9'sd118;
8'd94:  y = 9'sd122;
8'd95:  y = 9'sd126;
8'd96:  y = 9'sd130;
8'd97:  y = 9'sd134;
8'd98:  y = 9'sd138;
8'd99:  y = 9'sd142;
8'd100:  y = 9'sd146;
8'd101:  y = 9'sd150;
8'd102:  y = 9'sd154;
8'd103:  y = 9'sd158;
8'd104:  y = 9'sd162;
8'd105:  y = 9'sd166;
8'd106:  y = 9'sd170;
8'd107:  y = 9'sd174;
8'd108:  y = 9'sd178;
8'd109:  y = 9'sd182;
8'd110:  y = 9'sd186;
8'd111:  y = 9'sd190;
8'd112:  y = 9'sd194;
8'd113:  y = 9'sd198;
8'd114:  y = 9'sd202;
8'd115:  y = 9'sd206;
8'd116:  y = 9'sd210;
8'd117:  y = 9'sd214;
8'd118:  y = 9'sd218;
8'd119:  y = 9'sd222;
8'd120:  y = 9'sd226;
8'd121:  y = 9'sd230;
8'd122:  y = 9'sd234;
8'd123:  y = 9'sd238;
8'd124:  y = 9'sd242;
8'd125:  y = 9'sd246;
8'd126:  y = 9'sd250;
8'd127:  y = 9'sd254;
8'd128:  y = 9'sd254;
8'd129:  y = 9'sd250;
8'd130:  y = 9'sd246;
8'd131:  y = 9'sd242;
8'd132:  y = 9'sd238;
8'd133:  y = 9'sd234;
8'd134:  y = 9'sd230;
8'd135:  y = 9'sd226;
8'd136:  y = 9'sd222;
8'd137:  y = 9'sd218;
8'd138:  y = 9'sd214;
8'd139:  y = 9'sd210;
8'd140:  y = 9'sd206;
8'd141:  y = 9'sd202;
8'd142:  y = 9'sd198;
8'd143:  y = 9'sd194;
8'd144:  y = 9'sd190;
8'd145:  y = 9'sd186;
8'd146:  y = 9'sd182;
```

```
8'd147:  y = 9'sd178;
8'd148:  y = 9'sd174;
8'd149:  y = 9'sd170;
8'd150:  y = 9'sd166;
8'd151:  y = 9'sd162;
8'd152:  y = 9'sd158;
8'd153:  y = 9'sd154;
8'd154:  y = 9'sd150;
8'd155:  y = 9'sd146;
8'd156:  y = 9'sd142;
8'd157:  y = 9'sd138;
8'd158:  y = 9'sd134;
8'd159:  y = 9'sd130;
8'd160:  y = 9'sd126;
8'd161:  y = 9'sd122;
8'd162:  y = 9'sd118;
8'd163:  y = 9'sd114;
8'd164:  y = 9'sd110;
8'd165:  y = 9'sd106;
8'd166:  y = 9'sd102;
8'd167:  y = 9'sd98;
8'd168:  y = 9'sd94;
8'd169:  y = 9'sd90;
8'd170:  y = 9'sd86;
8'd171:  y = 9'sd82;
8'd172:  y = 9'sd78;
8'd173:  y = 9'sd74;
8'd174:  y = 9'sd70;
8'd175:  y = 9'sd66;
8'd176:  y = 9'sd62;
8'd177:  y = 9'sd58;
8'd178:  y = 9'sd54;
8'd179:  y = 9'sd50;
8'd180:  y = 9'sd46;
8'd181:  y = 9'sd42;
8'd182:  y = 9'sd38;
8'd183:  y = 9'sd34;
8'd184:  y = 9'sd30;
8'd185:  y = 9'sd26;
8'd186:  y = 9'sd22;
8'd187:  y = 9'sd18;
8'd188:  y = 9'sd14;
8'd189:  y = 9'sd10;
8'd190:  y = 9'sd6;
8'd191:  y = 9'sd2;
8'd192:  y = -9'sd2;
8'd193:  y = -9'sd6;
8'd194:  y = -9'sd10;
8'd195:  y = -9'sd14;
8'd196:  y = -9'sd18;
8'd197:  y = -9'sd22;
8'd198:  y = -9'sd26;
8'd199:  y = -9'sd30;
8'd200:  y = -9'sd34;
8'd201:  y = -9'sd38;
8'd202:  y = -9'sd42;
8'd203:  y = -9'sd46;
8'd204:  y = -9'sd50;
8'd205:  y = -9'sd54;
8'd206:  y = -9'sd58;
8'd207:  y = -9'sd62;
8'd208:  y = -9'sd66;
8'd209:  y = -9'sd70;
8'd210:  y = -9'sd74;
8'd211:  y = -9'sd78;
8'd212:  y = -9'sd82;
8'd213:  y = -9'sd86;
8'd214:  y = -9'sd90;
8'd215:  y = -9'sd94;
8'd216:  y = -9'sd98;
8'd217:  y = -9'sd102;
8'd218:  y = -9'sd106;
8'd219:  y = -9'sd110;
8'd220:  y = -9'sd114;
8'd221:  y = -9'sd118;
8'd222:  y = -9'sd122;
8'd223:  y = -9'sd126;
8'd224:  y = -9'sd130;
8'd225:  y = -9'sd134;
8'd226:  y = -9'sd138;
8'd227:  y = -9'sd142;
8'd228:  y = -9'sd146;
8'd229:  y = -9'sd150;
8'd230:  y = -9'sd154;
8'd231:  y = -9'sd158;
8'd232:  y = -9'sd162;
8'd233:  y = -9'sd166;
8'd234:  y = -9'sd170;
8'd235:  y = -9'sd174;
8'd236:  y = -9'sd178;
8'd237:  y = -9'sd182;
8'd238:  y = -9'sd186;
```

```
         8'd239:  y = -9'sd190;
         8'd240:  y = -9'sd194;
         8'd241:  y = -9'sd198;
         8'd242:  y = -9'sd202;
         8'd243:  y = -9'sd206;
         8'd244:  y = -9'sd210;
         8'd245:  y = -9'sd214;
         8'd246:  y = -9'sd218;
         8'd247:  y = -9'sd222;
         8'd248:  y = -9'sd226;
         8'd249:  y = -9'sd230;
         8'd250:  y = -9'sd234;
         8'd251:  y = -9'sd238;
         8'd252:  y = -9'sd242;
         8'd253:  y = -9'sd246;
         8'd254:  y = -9'sd250;
         8'd255:  y = -9'sd254;
      default: y = 9'sbXXXXXXXXX;
   endcase
endmodule

//`default_nettype none

////////////////////////////////////////////////////////////////////////////////
//
// Switch Debounce Module
//
////////////////////////////////////////////////////////////////////////////////

module debounce (
                  input wire reset, clock, noisy,
                  output reg clean
                  );
   reg [18:0]                count;
   reg                       new;

   always @(posedge clock)
     if (reset) begin
         count <= 0;
         new <= noisy;
         clean <= noisy;
     end
     else if (noisy != new) begin
         // noisy input changed, restart the .01 sec clock
         new <= noisy;
         count <= 0;
     end
     else if (count == 270000)
       // noisy input stable for .01 secs, pass it along!
       clean <= new;
     else
       // waiting for .01 sec to pass
       count <= count+1;

endmodule // debounce

////////////////////////////////////////////////////////////////////////////////
//
// bi-directional monaural interface to AC97
//
////////////////////////////////////////////////////////////////////////////////

module lab5audio (
                   input wire clock_27mhz,
                   input wire reset,
                   input wire [4:0] volume,
                   output wire signed [17:0] audio_in_data, // CHANGED FROM [7:0] and Signed
                   input wire [35:0] audio_out_data, // CHANGED FROM [7:0], 35 is for left and right
                   output wire ready,
                   output reg audio_reset_b,   // ac97 interface signals
                   output wire ac97_sdata_out,
                   input wire ac97_sdata_in,
                   output wire ac97_synch,
                   input wire ac97_bit_clock
                   );

   wire [7:0]                 command_address;
   wire [15:0]                command_data;
   wire                  command_valid;
   wire signed [19:0]         left_in_data, right_in_data; //signed
   wire signed [19:0]         left_out_data, right_out_data; //signed

   // wait a little before enabling the AC97 codec
   reg [9:0]                  reset_count;
   always @(posedge clock_27mhz) begin
      if (reset) begin
          audio_reset_b = 1'b0;
          reset_count = 0;
      end else if (reset_count == 1023)
        audio_reset_b = 1'b1;
      else
        reset_count = reset_count+1;
```

```
   end

   wire ac97_ready;
   ac97 ac97(.ready(ac97_ready),
             .command_address(command_address),
             .command_data(command_data),
             .command_valid(command_valid),
             .left_data(left_out_data), .left_valid(1'b1),
             .right_data(right_out_data), .right_valid(1'b1),
             .left_in_data(left_in_data), .right_in_data(right_in_data),
             .ac97_sdata_out(ac97_sdata_out),
             .ac97_sdata_in(ac97_sdata_in),
             .ac97_synch(ac97_synch),
             .ac97_bit_clock(ac97_bit_clock));

   // ready: one cycle pulse synchronous with clock_27mhz
   reg [2:0] ready_sync;
   always @ (posedge clock_27mhz) ready_sync <= {ready_sync[1:0], ac97_ready};
   assign ready = ready_sync[1] & ~ready_sync[2];

   reg [35:0] out_data; // CHANGED FROM [7:0], 35 is for left and right
   always @ (posedge clock_27mhz)
     if (ready) out_data <= audio_out_data;
   assign audio_in_data = left_in_data[19:2]; //CHANGED FROM [19:12]
   assign left_out_data = {out_data[35:18], 2'b00}; //CHANGED FROM 12'b
   assign right_out_data = {out_data[17:0], 2'b00}; //CHANGED FROM = left_out_data

   // generate repeating sequence of read/writes to AC97 registers
   ac97commands cmds(.clock(clock_27mhz), .ready(ready),
                     .command_address(command_address),
                     .command_data(command_data),
                     .command_valid(command_valid),
                     .volume(volume),
                     .source(3'b000));     // mic
endmodule

// assemble/disassemble AC97 serial frames
module ac97 (
             output reg ready,
             input wire [7:0] command_address,
             input wire [15:0] command_data,
             input wire command_valid,
             input wire signed [19:0] left_data, //signed
             input wire left_valid,
             input wire signed [19:0] right_data, //signed
             input wire right_valid,
             output reg signed [19:0] left_in_data, right_in_data, //signed
             output reg ac97_sdata_out,
             input wire ac97_sdata_in,
             output reg ac97_synch,
             input wire ac97_bit_clock
             );
   reg [7:0]             bit_count;

   reg [19:0]            l_cmd_addr;
   reg [19:0]            l_cmd_data;
   reg [19:0]            l_left_data, l_right_data;
   reg                   l_cmd_v, l_left_v, l_right_v;

   initial begin
      ready <= 1'b0;
      // synthesis attribute init of ready is "0";
      ac97_sdata_out <= 1'b0;
      // synthesis attribute init of ac97_sdata_out is "0";
      ac97_synch <= 1'b0;
      // synthesis attribute init of ac97_synch is "0";

      bit_count <= 8'h00;
      // synthesis attribute init of bit_count is "0000";
      l_cmd_v <= 1'b0;
      // synthesis attribute init of l_cmd_v is "0";
      l_left_v <= 1'b0;
      // synthesis attribute init of l_left_v is "0";
      l_right_v <= 1'b0;
      // synthesis attribute init of l_right_v is "0";

      left_in_data <= 20'h00000;
      // synthesis attribute init of left_in_data is "00000";
      right_in_data <= 20'h00000;
      // synthesis attribute init of right_in_data is "00000";
   end

   always @(posedge ac97_bit_clock) begin
      // Generate the sync signal
      if (bit_count == 255)
        ac97_synch <= 1'b1;
      if (bit_count == 15)
        ac97_synch <= 1'b0;

      // Generate the ready signal
      if (bit_count == 128)
        ready <= 1'b1;
```

```
      if (bit_count == 2)
        ready <= 1'b0;

      // Latch user data at the end of each frame. This ensures that the
      // first frame after reset will be empty.
      if (bit_count == 255) begin
          l_cmd_addr <= {command_address, 12'h000};
          l_cmd_data <= {command_data, 4'h0};
          l_cmd_v <= command_valid;
          l_left_data <= left_data;
          l_left_v <= left_valid;
          l_right_data <= right_data;
          l_right_v <= right_valid;
      end

      if ((bit_count >= 0) && (bit_count <= 15))
        // Slot 0: Tags
        case (bit_count[3:0])
          4'h0: ac97_sdata_out <= 1'b1;      // Frame valid
          4'h1: ac97_sdata_out <= l_cmd_v;   // Command address valid
          4'h2: ac97_sdata_out <= l_cmd_v;   // Command data valid
          4'h3: ac97_sdata_out <= l_left_v;  // Left data valid
          4'h4: ac97_sdata_out <= l_right_v; // Right data valid
          default: ac97_sdata_out <= 1'b0;
        endcase
      else if ((bit_count >= 16) && (bit_count <= 35))
        // Slot 1: Command address (8-bits, left justified)
        ac97_sdata_out <= l_cmd_v ? l_cmd_addr[35-bit_count] : 1'b0;
      else if ((bit_count >= 36) && (bit_count <= 55))
        // Slot 2: Command data (16-bits, left justified)
        ac97_sdata_out <= l_cmd_v ? l_cmd_data[55-bit_count] : 1'b0;
      else if ((bit_count >= 56) && (bit_count <= 75)) begin
         // Slot 3: Left channel
         ac97_sdata_out <= l_left_v ? l_left_data[19] : 1'b0;
         l_left_data <= { l_left_data[18:0], l_left_data[19] };
      end
      else if ((bit_count >= 76) && (bit_count <= 95))
        // Slot 4: Right channel
        ac97_sdata_out <= l_right_v ? l_right_data[95-bit_count] : 1'b0;
      else
        ac97_sdata_out <= 1'b0;

      bit_count <= bit_count+1;
   end // always @ (posedge ac97_bit_clock)

   always @(negedge ac97_bit_clock) begin
      if ((bit_count >= 57) && (bit_count <= 76))
        // Slot 3: Left channel
        left_in_data <= { left_in_data[18:0], ac97_sdata_in };
      else if ((bit_count >= 77) && (bit_count <= 96))
        // Slot 4: Right channel
        right_in_data <= { right_in_data[18:0], ac97_sdata_in };
   end
endmodule

// issue initialization commands to AC97
module ac97commands (
                     input wire clock,
                     input wire ready,
                     output wire [7:0] command_address,
                     output wire [15:0] command_data,
                     output reg command_valid,
                     input wire [4:0] volume,
                     input wire [2:0] source
                     );
   reg [23:0]                        command;

   reg [3:0]                         state;
   initial begin
      command <= 4'h0;
      // synthesis attribute init of command is "0";
      command_valid <= 1'b0;
      // synthesis attribute init of command_valid is "0";
      state <= 16'h0000;
      // synthesis attribute init of state is "0000";
   end

   assign command_address = command[23:16];
   assign command_data = command[15:0];

   wire [4:0] vol;
   assign vol = 31-volume;  // convert to attenuation

   always @(posedge clock) begin
      if (ready) state <= state+1;

      case (state)
        4'h0: // Read ID
          begin
             command <= 24'h80_0000;
             command_valid <= 1'b1;
          end
```

```
        4'h1: // Read ID
          command <= 24'h80_0000;
        4'h3: // headphone volume
          command <= { 8'h04, 3'b000, vol, 3'b000, vol };
        4'h5: // PCM volume
          command <= 24'h18_0808;
        4'h6: // Record source select
          command <= { 8'h1A, 5'b00000, source, 5'b00000, source};
        4'h7: // Record gain = max
          command <= 24'h1C_0F0F;
        4'h9: // set +20db mic gain
          command <= 24'h0E_8048;
        4'hA: // Set beep volume
          command <= 24'h0A_0000;
        4'hB: // PCM out bypass mix1
          command <= 24'h20_8000;
        default:
          command <= 24'h80_0000;
      endcase // case(state)
   end // always @ (posedge clock)
endmodule // ac97commands


//////////////////////////////////////////////////////////////////////////////////
////
//// 6.111 FPGA Labkit -- Template Toplevel Module
////
//// For Labkit Revision 004
//// Created: October 31, 2004, from revision 003 file
//// Author: Nathan Ickes, 6.111 staff
////
//////////////////////////////////////////////////////////////////////////////////

module lab5   (beep, audio_reset_b, ac97_sdata_out, ac97_sdata_in, ac97_synch,
               ac97_bit_clock,

               vga_out_red, vga_out_green, vga_out_blue, vga_out_sync_b,
               vga_out_blank_b, vga_out_pixel_clock, vga_out_hsync,
               vga_out_vsync,

               tv_out_ycrcb, tv_out_reset_b, tv_out_clock, tv_out_i2c_clock,
               tv_out_i2c_data, tv_out_pal_ntsc, tv_out_hsync_b,
               tv_out_vsync_b, tv_out_blank_b, tv_out_subcar_reset,

               tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1,
               tv_in_line_clock2, tv_in_aef, tv_in_hff, tv_in_aff,
               tv_in_i2c_clock, tv_in_i2c_data, tv_in_fifo_read,
               tv_in_fifo_clock, tv_in_iso, tv_in_reset_b, tv_in_clock,

               ram0_data, ram0_address, ram0_adv_ld, ram0_clk, ram0_cen_b,
               ram0_ce_b, ram0_oe_b, ram0_we_b, ram0_bwe_b,

               ram1_data, ram1_address, ram1_adv_ld, ram1_clk, ram1_cen_b,
               ram1_ce_b, ram1_oe_b, ram1_we_b, ram1_bwe_b,

               clock_feedback_out, clock_feedback_in,

               flash_data, flash_address, flash_ce_b, flash_oe_b, flash_we_b,
               flash_reset_b, flash_sts, flash_byte_b,

               rs232_txd, rs232_rxd, rs232_rts, rs232_cts,

               mouse_clock, mouse_data, keyboard_clock, keyboard_data,

               clock_27mhz, clock1, clock2,

               disp_blank, disp_data_out, disp_clock, disp_rs, disp_ce_b,
               disp_reset_b, disp_data_in,

               button0, button1, button2, button3, button_enter, button_right,
               button_left, button_down, button_up,

               switch,

               led,

               user1, user2, user3, user4,

               daughtercard,

               systemace_data, systemace_address, systemace_ce_b,
               systemace_we_b, systemace_oe_b, systemace_irq, systemace_mpbrdy,

               analyzer1_data, analyzer1_clock,
               analyzer2_data, analyzer2_clock,
               analyzer3_data, analyzer3_clock,
               analyzer4_data, analyzer4_clock);

   output beep, audio_reset_b, ac97_synch, ac97_sdata_out;
   input  ac97_bit_clock, ac97_sdata_in;

   output [7:0] vga_out_red, vga_out_green, vga_out_blue;
```

```
output        vga_out_sync_b, vga_out_blank_b, vga_out_pixel_clock,
              vga_out_hsync, vga_out_vsync;

output [9:0] tv_out_ycrcb;
output        tv_out_reset_b, tv_out_clock, tv_out_i2c_clock, tv_out_i2c_data,
              tv_out_pal_ntsc, tv_out_hsync_b, tv_out_vsync_b, tv_out_blank_b,
              tv_out_subcar_reset;

input [19:0] tv_in_ycrcb;
input         tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2, tv_in_aef,
              tv_in_hff, tv_in_aff;
output        tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock, tv_in_iso,
              tv_in_reset_b, tv_in_clock;
inout         tv_in_i2c_data;

inout [35:0] ram0_data;
output [18:0] ram0_address;
output         ram0_adv_ld, ram0_clk, ram0_cen_b, ram0_ce_b, ram0_oe_b, ram0_we_b;
output [3:0]  ram0_bwe_b;

inout [35:0]  ram1_data;
output [18:0] ram1_address;
output         ram1_adv_ld, ram1_clk, ram1_cen_b, ram1_ce_b, ram1_oe_b, ram1_we_b;
output [3:0]  ram1_bwe_b;

input          clock_feedback_in;
output         clock_feedback_out;

inout [15:0]  flash_data;
output [23:0] flash_address;
output         flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b, flash_byte_b;
input          flash_sts;

output         rs232_txd, rs232_rts;
input          rs232_rxd, rs232_cts;

input          mouse_clock, mouse_data, keyboard_clock, keyboard_data;

input          clock_27mhz, clock1, clock2;

output         disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b;
input          disp_data_in;
output         disp_data_out;

input          button0, button1, button2, button3, button_enter, button_right,
               button_left, button_down, button_up;
input [7:0]    switch;
output [7:0]  led;

inout [31:0]  user1, user2, user3, user4;

inout [43:0]  daughtercard;

inout [15:0]  systemace_data;
output [6:0]  systemace_address;
output         systemace_ce_b, systemace_we_b, systemace_oe_b;
input          systemace_irq, systemace_mpbrdy;

output [15:0] analyzer1_data, analyzer2_data, analyzer3_data,
               analyzer4_data;
output         analyzer1_clock, analyzer2_clock, analyzer3_clock, analyzer4_clock;

////////////////////////////////////////////////////////////////////////////////
//
// I/O Assignments
//
////////////////////////////////////////////////////////////////////////////////


// Audio Input and Output
assign beep= 1'b0;
//lab5 assign audio_reset_b = 1'b0;
//lab5 assign ac97_synch = 1'b0;
//lab5 assign ac97_sdata_out = 1'b0;
// ac97_sdata_in is an input

// VGA Output
assign vga_out_red = 10'h0;
assign vga_out_green = 10'h0;
assign vga_out_blue = 10'h0;
assign vga_out_sync_b = 1'b1;
assign vga_out_blank_b = 1'b1;
assign vga_out_pixel_clock = 1'b0;
assign vga_out_hsync = 1'b0;
assign vga_out_vsync = 1'b0;

// Video Output
assign tv_out_ycrcb = 10'h0;
assign tv_out_reset_b = 1'b0;
assign tv_out_clock = 1'b0;
assign tv_out_i2c_clock = 1'b0;
assign tv_out_i2c_data = 1'b0;
```

```
assign tv_out_pal_ntsc = 1'b0;
assign tv_out_hsync_b = 1'b1;
assign tv_out_vsync_b = 1'b1;
assign tv_out_blank_b = 1'b1;
assign tv_out_subcar_reset = 1'b0;

// Video Input
assign tv_in_i2c_clock = 1'b0;
assign tv_in_fifo_read = 1'b0;
assign tv_in_fifo_clock = 1'b0;
assign tv_in_iso = 1'b0;
assign tv_in_reset_b = 1'b0;
assign tv_in_clock = 1'b0;
assign tv_in_i2c_data = 1'bZ;
// tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2,
// tv_in_aef, tv_in_hff, and tv_in_aff are inputs

// SRAMs
// COMMENTED OUT FOR ZBT RECORDER MODULE
// assign ram0_data = 36'hZ;
// assign ram0_address = 19'h0;
// assign ram0_adv_ld = 1'b0;
// assign ram0_clk = 1'b0;
// assign ram0_cen_b = 1'b1;
// assign ram0_ce_b = 1'b1;
// assign ram0_oe_b = 1'b1;
// assign ram0_we_b = 1'b1;
// assign ram0_bwe_b = 4'hF;
// assign ram1_data = 36'hZ;
// assign ram1_address = 19'h0;
// assign ram1_adv_ld = 1'b0;
// assign ram1_clk = 1'b0;
// assign ram1_cen_b = 1'b1;
// assign ram1_ce_b = 1'b1;
// assign ram1_oe_b = 1'b1;
// assign ram1_we_b = 1'b1;
// assign ram1_bwe_b = 4'hF;
// assign clock_feedback_out = 1'b0;
// clock_feedback_in is an input

// new ram0/ram1 values for ZBT recorder
assign ram0_ce_b = 1'b0;
assign ram0_oe_b = 1'b0;
assign ram0_adv_ld = 1'b0;
assign ram0_bwe_b = 4'h0;

assign ram1_ce_b = 1'b0;
assign ram1_oe_b = 1'b0;
assign ram1_adv_ld = 1'b0;
assign ram1_bwe_b = 4'h0;

// Flash ROM
assign flash_data = 16'hZ;
assign flash_address = 24'h0;
assign flash_ce_b = 1'b1;
assign flash_oe_b = 1'b1;
assign flash_we_b = 1'b1;
assign flash_reset_b = 1'b0;
assign flash_byte_b = 1'b1;
// flash_sts is an input

// RS-232 Interface
assign rs232_txd = 1'b1;
assign rs232_rts = 1'b1;
// rs232_rxd and rs232_cts are inputs

// PS/2 Ports
// mouse_clock, mouse_data, keyboard_clock, and keyboard_data are inputs

// LED Displays
//assign disp_blank = 1'b1;
//assign disp_clock = 1'b0;
//assign disp_rs = 1'b0;
//assign disp_ce_b = 1'b1;
//assign disp_reset_b = 1'b0;
//assign disp_data_out = 1'b0;
// disp_data_in is an input

// Buttons, Switches, and Individual LEDs
assign led[6:4] = 3'b111;
// button0, button1, button2, button3, button_enter, button_right,
// button_left, button_down, button_up, and switches are inputs

// User I/Os
//assign user1 = 32'hZ;
assign user1[31:4] = 28'hZ;
assign user2 = 32'hZ;
//assign user3 = 32'hZ;
assign user3[31:8] = 24'hZ;
assign user4 = 32'hZ;

// Daughtercard Connectors
```

```
   assign daughtercard = 44'hZ;

   // SystemACE Microprocessor Port
   assign systemace_data = 16'hZ;
   assign systemace_address = 7'h0;
   assign systemace_ce_b = 1'b1;
   assign systemace_we_b = 1'b1;
   assign systemace_oe_b = 1'b1;
   // systemace_irq and systemace_mpbrdy are inputs

   // Logic Analyzer
   //lab5 assign analyzer1_data = 16'h0;
   //lab5 assign analyzer1_clock = 1'b1;
   //assign analyzer2_data = 16'h0;
   //assign analyzer2_clock = 1'b1;
   //lab5 assign analyzer3_data = 16'h0;
   //lab5 assign analyzer3_clock = 1'b1;
   //assign analyzer4_data = 16'h0;
   //assign analyzer4_clock = 1'b1;

   //   wire [7:0] from_ac97_data, to_ac97_data;
   //   wire ready;

   ////////////////////////////////////////////////////////////////////////
   //
   // Reset Generation
   //
   // A shift register primitive is used to generate an active-high reset
   // signal that remains high for 16 clock cycles after configuration finishes
   // and the FPGA's internal clocks begin toggling.
   //
   ////////////////////////////////////////////////////////////////////////
   // used by ZBT module
   wire  locked;
   wire  clock;
   ramclock rc(.ref_clock(clock_27mhz), .fpga_clock(clock),
               .ram0_clock(ram0_clk),
               .ram1_clock(ram1_clk),   //uncomment if ram1 is used
               .clock_feedback_in(clock_feedback_in),
               .clock_feedback_out(clock_feedback_out), .locked(locked));

   wire  reset;
   SRL16 #(.INIT(16'hFFFF)) reset_sr(.D(1'b0), .CLK(clock), .Q(reset),
                                     .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));

   wire signed [17:0] from_ac97_data; // CHANGED FROM [7:0]
   wire signed [35:0] to_ac97_data; // CHANGED FROM [7:0], 35 for left and right, signed
   wire       ready;

   // the zbt controllers
   // zbt chip 0
   wire [18:0]         ram0_addr;
   wire       ram0_we;
   wire [35:0]         ram0_write_data;
   wire [35:0]         ram0_read_data;
   wire       ram0_clk_not_used;
   zbt_6111 zbt0(clock, 1'b1, ram0_we, ram0_addr,
                 ram0_write_data, ram0_read_data,
                 ram0_clk_not_used,
                 ram0_we_b, ram0_address, ram0_data, ram0_cen_b);
   // zbt chip 1
   wire [18:0]         ram1_addr;
   wire       ram1_we;
   wire [35:0]         ram1_write_data;
   wire [35:0]         ram1_read_data;
   wire       ram1_clk_not_used;
   zbt_6111 zbt1(clock, 1'b1, ram1_we, ram1_addr,
                 ram1_write_data, ram1_read_data,
                 ram1_clk_not_used,
                 ram1_we_b, ram1_address, ram1_data, ram1_cen_b);

   //debounce bup(.reset(reset),.clock(clock_27mhz),.noisy(~button_up),.clean(vup));   STILL NEED VOLUME UP AND VOLUME DOWN
BUT NOT USING UP AND DOWN BUTTONS
   //debounce bdown(.reset(reset),.clock(clock_27mhz),.noisy(~button_down),.clean(vdown));
   wire [4:0]          volume;
   // AC97 driver -- use clock from ramclock above??
   lab5audio a(.clock_27mhz(clock), .reset(reset), .volume(volume), .audio_in_data(from_ac97_data),
.audio_out_data(to_ac97_data), .ready(ready),.audio_reset_b(audio_reset_b), .ac97_sdata_out(ac97_sdata_out),
.ac97_sdata_in(ac97_sdata_in), .ac97_synch(ac97_synch), .ac97_bit_clock(ac97_bit_clock));

   //WIRE ALL THE THINGS!!
   wire       scro_up,scro_down,scro_left,scro_right,rec_0ch,rec_1ch,rec_2ch,rec_3ch,chan_on,
                       disto,wah,moog,rev_cho,trem,long_del,equalizer,pan;
   wire [15:0]         chan_vol_out = 0;

   //LED inverter logic
   wire [3:0]          pre_ch_enable_led;
   wire       pre_action_led;
   assign led[3:0] = ~pre_ch_enable_led;
   assign led[7] = ~pre_action_led;

   //DEBOUNCE ALL THE THINGS!!
```

```
   debounce bup(.reset(reset),.clock(clock),.noisy(~button_up),.clean(scro_up));
   debounce bdown(.reset(reset),.clock(clock),.noisy(~button_down),.clean(scro_down));
   debounce bleft(.reset(reset),.clock(clock),.noisy(~button_left),.clean(scro_left));
   debounce bright(.reset(reset),.clock(clock),.noisy(~button_right),.clean(scro_right));
   debounce b1rec(.reset(reset),.clock(clock),.noisy(~button0),.clean(rec_0ch));
   debounce b2rec(.reset(reset),.clock(clock),.noisy(~button1),.clean(rec_1ch));
   debounce b3rec(.reset(reset),.clock(clock),.noisy(~button2),.clean(rec_2ch));
   debounce b4rec(.reset(reset),.clock(clock),.noisy(~button3),.clean(rec_3ch));
   debounce bchon(.reset(reset),.clock(clock),.noisy(~button_enter),.clean(chan_on));
   debounce disto_switch(.reset(reset),.clock(clock),.noisy(switch[0]),
                         .clean(disto));
   debounce wah_switch(.reset(reset),.clock(clock),.noisy(switch[1]),
                       .clean(wah));
   debounce moog_switch(.reset(reset),.clock(clock),.noisy(switch[2]),
                        .clean(moog));
   debounce rev_cho_switch(.reset(reset),.clock(clock),.noisy(switch[3]),
                           .clean(rev_cho));
   debounce trem_switch(.reset(reset),.clock(clock),.noisy(switch[4]),
                        .clean(trem));
   debounce long_del_switch(.reset(reset),.clock(clock),.noisy(switch[5]),
                            .clean(long_del));
   debounce equalizer_switch(.reset(reset),.clock(clock),.noisy(switch[6]),
                             .clean(equalizer));
   debounce pan_switch(.reset(reset),.clock(clock),.noisy(switch[7]),
                       .clean(pan));

   // for debugging
   wire [4:0]          state_out;
   wire [17:0]         addr_current_0_out;
   wire [17:0]         addr_current_1_out;
   wire [17:0]         addr_current_2_out;
   wire [17:0]         addr_current_3_out;
   wire [17:0]         addr_current_bak_0_out;
   wire [17:0]         addr_current_bak_1_out;
   wire [17:0]         addr_current_bak_2_out;
   wire [17:0]         addr_current_bak_3_out;
   wire [17:0]         addr_end_0_out;
   wire [17:0]         addr_end_1_out;
   wire [17:0]         addr_end_2_out;
   wire [17:0]         addr_end_3_out;
   wire [2:0]          mix_state;

   // output useful things to the logic analyzer connectors
   assign analyzer1_clock = clock;
   assign analyzer2_clock = clock;
   assign analyzer3_clock = clock;
   assign analyzer4_clock = clock;
   assign analyzer1_data = {8'd0, state_out,mix_state};
   assign analyzer2_data = {addr_current_3_out[3:0],
                            addr_current_2_out[3:0],
                            addr_current_1_out[3:0],
                            addr_current_0_out[3:0]};
   assign analyzer3_data = {addr_current_bak_3_out[3:0],
                            addr_current_bak_2_out[3:0],
                            addr_current_bak_1_out[3:0],
                            addr_current_bak_0_out[3:0]};
   assign analyzer4_data = {addr_end_3_out[3:0],
                            addr_end_2_out[3:0],
                            addr_end_1_out[3:0],
                            addr_end_0_out[3:0]};

   // record module -- use clock from ramclock above
   FXController FX(.clock(clock),.reset(reset),.ready(ready),.intr(user1[3]),.db(user3[7:0]),
                   .scro_up(scro_up),.scro_down(scro_down),.scro_left(scro_left),.scro_right(scro_right),
                   .chan_on(chan_on),.rec_0ch(rec_0ch),.rec_1ch(rec_1ch),.rec_2ch(rec_2ch),
                   .rec_3ch(rec_3ch),.disto(disto),.wah(wah),.moog(moog),.rev_cho(rev_cho),
                   .trem(trem),.long_del(long_del),.equalizer(equalizer),.pan(pan),
                   .from_ac97_data(from_ac97_data),.ram0_read_data(ram0_read_data), .ram1_read_data(ram1_read_data),
                   .ram0_write_data(ram0_write_data), .ram1_write_data(ram1_write_data),.ram0_we(ram0_we),
                   .ram1_we(ram1_we),.ram0_addr(ram0_addr), .ram1_addr(ram1_addr),.four_chan_out(chan_vol_out),
                   .disp_blank(disp_blank),.disp_clock(disp_clock),.disp_data_out(disp_data_out),.disp_rs(disp_rs),
                   .disp_ce_b(disp_ce_b),.disp_reset_b(disp_reset_b),.volume(volume),.effects_out(to_ac97_data),

.cs(user1[0]),.rd(user1[1]),.wr(user1[2]),.looper_ch_enable(pre_ch_enable_led),.action_led_out(pre_action_led),
                   // for debugging
                   .state_out(state_out),
                   .addr_end_0_out(addr_end_0_out),
                   .addr_end_1_out(addr_end_1_out),
                   .addr_end_2_out(addr_end_2_out),
                   .addr_end_3_out(addr_end_3_out)
                   );
endmodule

//////////////////////////////////////////////////////////////////////////////////
     //
// FX CONTROLLER
//
//////////////////////////////////////////////////////////////////////////////////

module FXController(
                    input wire clock,               // 27mhz system clock
                    input wire reset,                // 1 to reset to initial state
```

```
                    input wire ready,                // 1 when AC97 data is available
                    input wire intr,
                    input wire [7:0]db,
                    input wire scro_up,scro_down,scro_left,scro_right,chan_on,rec_0ch,rec_1ch,rec_2ch,rec_3ch,
                    input wire disto, wah, moog, rev_cho, trem, long_del, equalizer, pan, //switch inputs turn effects
on/off. 1 on, 0 off
                    input wire signed [17:0] from_ac97_data, // 18-bit PCM data from mic
                    // ZBT-related inputs/outputs
                    input wire [35:0] ram0_read_data,
                    input wire [35:0] ram1_read_data,
                    output wire [35:0] ram0_write_data,
                    output wire [35:0] ram1_write_data,
                    output wire ram0_we,
                    output wire ram1_we,
                    output wire [18:0] ram0_addr,
                    output wire [18:0] ram1_addr,
                    output reg [15:0] four_chan_out, //volume for the four channels, 4 bits each
                    output disp_blank, disp_clock, disp_data_out, disp_rs, disp_ce_b, disp_reset_b,
                    output [4:0] volume,
                    output reg signed [35:0] effects_out,   // 36-bit PCM data to headphone
                    output wire cs,rd,wr,
                    output wire [3:0] looper_ch_enable,
                    output wire action_led_out,
                    // debugging info
                    output wire [4:0] state_out,
                    output wire [17:0] addr_end_0_out,
                    output wire [17:0] addr_end_1_out,
                    output wire [17:0] addr_end_2_out,
                    output wire [17:0] addr_end_3_out
                    );

   wire [16*8-1:0]                      string_data;
   wire signed [17:0]                   rev_chor_out; //combined output of reverb and chorus
   reg [39:0]                           effect_name = "Disto";
   reg [39:0]                           param_name = "_____";
   reg [23:0]                           param_val = "___";
   reg [3:0]                            effect_menu_name = 4'd0;
   reg [4:0]                            param_menu_name = 4'd22;
   wire                           up_up,down_down,left_left,right_right; //for button scrolling ALMOST KONAMI CODE!!! <- ROFL
LOL :D
   wire [7:0]                           db_out;
   //MENU PARAMETERS
   parameter BLANK = 5'd22;
   parameter DISTORTION = 4'd0;
   parameter THRES = 5'd0;
   parameter OVERD = 5'd1;
   parameter GAIN = 5'd2;
   parameter WAH = 4'd1;
   parameter AUTO = 5'd3;
   parameter WAH_P = 5'd4;
   parameter WIDTH = 5'd5;
   parameter PERIO = 5'd6;
   //parameter THRES = 5'd0;
   parameter MOOG = 4'd2;
   parameter WAVE = 5'd7;
   //parameter PERIO = 5'd6;
   parameter MOOG_AMOUNT = 5'd29;
   parameter REVERB = 4'd3;
   parameter WAVON = 5'd25;
   //parameter WAVE = 5'd7;
   parameter DELAY = 5'd8;
   parameter DECAY = 5'd9;
   parameter WVFRQ = 5'd26;
   parameter CHORUS = 4'd4;
   parameter CHOON = 5'd23;
   //parameter WAVE = 5'd7;
   //parameter DELAY = 5'd8;
   parameter DEPTH = 5'd10;
   parameter WOBBL = 5'd24;
   //parameter DECAY = 5'd9;
   parameter TREMOLO = 4'd5;
   //parameter DECAY = 5'd9;
   //parameter PERIO = 5'd6;
   parameter LONG_DELAY = 4'd6;
   //parameter DELAY = 5'd8;
   //parameter DECAY = 5'd9;
   parameter FIVEBAND_EQ = 4'd7;
   parameter LOWV = 5'd11;
   parameter LOMDV = 5'd12;
   parameter MIDV = 5'd13;
   parameter HIMDV = 5'd14;
   parameter HIV = 5'd15;
   parameter PANNING = 4'd8;
   parameter FADE = 5'd16;
   //parameter PERIO = 5'd6;
   parameter LOOPER = 4'd9;
   // effect on/off
   parameter LOOPER_ONOFF = 5'd27;
   // for selecting loop parameter
   parameter LOOPER_LOOP = 5'd28;
   // channel volumes
   parameter C0VOL = 5'd17;
```

```
parameter C1VOL = 5'd18;
parameter C2VOL = 5'd19;
parameter C3VOL = 5'd20;
// for selecting each channel
parameter CH0 = 2'd0;
parameter CH1 = 2'd1;
parameter CH2 = 2'd2;
parameter CH3 = 2'd3;

parameter VOLUME = 4'd13;
parameter VOLUM = 5'd21;

//DISTORTION IN/OUT
wire signed [17:0]                  dist_in;
reg signed [7:0]                    threshold_in = 8'b00001111;
reg signed [6:0]                    overdrive_scale = 7'sd0;
reg signed [7:0]                    gain_in = 8'b01111111;
wire signed [17:0]                  dist_out;

//WAH IN/OUT
wire signed [17:0]                  wah_in;
reg                                 autowah_on = 0;
reg [1:0]                           width_in = 2'd2;
reg [5:0]                           wah_period_in = 6'b000011;
reg signed [7:0]                    wah_threshold_in = 8'b00000111;
wire signed [17:0]                  wah_out;

//RING MODULATOR IN/OUT
wire signed [17:0]                  moog_in;
reg [1:0]                           moog_wavesel = 2'b01;
reg [9:0]                           moog_period = 10'b0000001111;
reg [5:0]                           moog_amount = 6'd50;
wire signed [17:0]                  moog_out;

//REVERB IN/OUT
wire signed [17:0]                  rev_in;
reg                                 rev_waveon = 0;
reg [1:0]                           rev_wavesel = 2'd0;
reg [9:0]                           rev_delay_in = 10'b0011111111;
reg [9:0]                           rev_decay_in = 10'b0111111111;
reg [10:0]                          rev_wave_freq = 11'b00011111111;
wire signed [17:0]                  rev_out;

//CHORUS IN/OUT
wire signed [17:0]                  chor_in;
reg                                 choon = 0;
reg [1:0]                           chor_wavesel = 2'b01;
reg [9:0]                           chor_delay_in = 10'b0011111111;
reg [7:0]                           chor_depth = 8'b00000011;
reg [8:0]                           chor_rate = 9'b000000111;
reg signed [10:0]                   chor_decay_in = 11'b00111111111;
wire signed [17:0]                  chor_out;

//TREMOLO IN/OUT
wire signed [17:0]                  trem_in;
reg signed [9:0]                    trem_decayval = 10'b0001111111;
reg [9:0]                           trem_halfperiod = 10'b0001111111;
wire signed [17:0]                  trem_out;

//LONG DELAY IN/OUT
wire signed [17:0]                  long_del_in;
reg [9:0]                           long_delay_in = 9'b011111111;
reg [9:0]                           long_decay_in = 9'b011111111;
wire signed [17:0]                  long_del_out;

//FIVE BAND EQ IN/OUT
wire signed [17:0]                  eq_in;
reg signed [5:0]                    lowvol = 6'b001111;
reg signed [5:0]                    lowmidvol = 6'b001111;
reg signed [5:0]                    midvol = 6'b011111;
reg signed [5:0]                    highmidvol = 6'b001111;
reg signed [5:0]                    highvol = 6'b001111;
wire signed [17:0]                  eq_out;

//PANNING IN/OUT
wire signed [17:0]                  pan_in;
reg                                 fade_on = 0;
reg signed [8:0]                    pan_speed_in = 9'b000001111;
wire signed [35:0]                  pan_out;

//LOOPER IN
reg [3:0]                           level_0ch = 4'b0111;
reg [3:0]                           level_1ch = 4'b0111;
reg [3:0]                           level_2ch = 4'b0111;
reg [3:0]                           level_3ch = 4'b0111;
wire [1:0]                          looper_chsel; // which channel to record to
wire [1:0]                          looper_action; // OFF/REC/PLAY
reg                                 looper_on = 1'b0; // enable/disable the effect
reg                                 looper_loop = 1'b1; // loop in playback? default=yes
wire signed [35:0]                  looper_in;
wire signed [35:0]                  looper_out;
reg                                 action_led = 0;
```

```
reg [22:0]                          action_led_counter = 0;
wire                        loop_rec_0ch;
wire                        loop_rec_1ch;
wire                        loop_rec_2ch;
wire                        loop_rec_3ch;
wire                        loop_chan_on;

//VOLUME IN
reg [4:0]                           volume_out = 5'b10011;

//MENU STATE MACHINE
always @(posedge clock) begin
   {threshold_in[7],wah_threshold_in[7],overdrive_scale[6],
    gain_in[7],chor_decay_in[10],lowvol[5],lowmidvol[5],midvol[5],
    highmidvol[5],highvol[5],pan_speed_in[8]} <= 0;
   case(effect_menu_name)
     DISTORTION:begin
        case(param_menu_name)
          BLANK:begin
             effect_name = "Disto";
             param_name = "_____";
             param_val = "___";
             if (up_up) effect_menu_name <= VOLUME;
             else if (down_down) effect_menu_name <= WAH;
             else if (right_right) param_menu_name <= THRES;
             else if (left_left) param_menu_name <= GAIN;
          end
          THRES:begin
             param_name = "Thres";
             param_val = {13'd0,threshold_in[6:4],4'h0,threshold_in[3:0]}; //work out signed
             if (up_up) threshold_in <= threshold_in + 1;
             else if (down_down) threshold_in <= threshold_in - 1;
             else if (right_right) param_menu_name <= OVERD;
             else if (left_left) param_menu_name <= BLANK;
          end
          OVERD:begin
             param_name = "OverD";
             param_val = {14'd0,overdrive_scale[5:4],4'h0,overdrive_scale[3:0]}; //work out signed
             if (up_up) overdrive_scale <= overdrive_scale + 1;
             else if (down_down) overdrive_scale <= overdrive_scale - 1;
             else if (right_right) param_menu_name <= GAIN;
             else if (left_left) param_menu_name <= THRES;
          end
          GAIN:begin
             param_name = "Gain ";
             param_val = {13'd0,gain_in[6:4],4'h0,gain_in[3:0]}; //work out signed
             if (up_up) gain_in <= gain_in + 1;
             else if (down_down) gain_in <= gain_in - 1;
             else if (right_right) param_menu_name <= BLANK;
             else if (left_left) param_menu_name <= OVERD;
          end
          default: param_menu_name <= BLANK;
        endcase
     end
     WAH:begin
        case(param_menu_name)
          BLANK:begin
             effect_name = "Wah  ";
             param_name = "_____";
             param_val = "___";
             if (up_up) effect_menu_name <= DISTORTION;
             else if (down_down) effect_menu_name <= MOOG;
             else if (right_right) param_menu_name <= AUTO;
             else if (left_left) param_menu_name <= THRES;
          end
          AUTO:begin
             param_name = "Auto ";
             param_val = autowah_on ? " On" : "Off";
             if (up_up) autowah_on <= autowah_on + 1;
             else if (down_down) autowah_on <= autowah_on - 1;
             else if (right_right) param_menu_name <= WAH_P;
             else if (left_left) param_menu_name <= BLANK;
          end
          WAH_P:begin
             param_name = "Wah_P";
             param_val = {15'd0,db_out[7],4'h0,db_out[6:3]};
             //if (up_up) wah_position_in <= wah_position_in + 1;
             //else if (down_down) wah_position_in <= wah_position_in - 1;
             if (right_right) param_menu_name <= WIDTH;
             else if (left_left) param_menu_name <= AUTO;
          end
          WIDTH:begin
             param_name = "Width";
             param_val = {22'b0,width_in};
             if (up_up) width_in <= width_in + 1;
             else if (down_down) width_in <= width_in - 1;
             else if (right_right) param_menu_name <= PERIO;
             else if (left_left) param_menu_name <= WAH_P;
          end
          PERIO:begin
             param_name = "Perio";
             param_val = {14'b0,wah_period_in[5:4],4'h0,wah_period_in[3:0]};
```

```
         if (up_up) wah_period_in <= wah_period_in + 1;
         else if (down_down) wah_period_in <= wah_period_in - 1;
         else if (right_right) param_menu_name <= THRES;
         else if (left_left) param_menu_name <= WIDTH;
      end
      THRES:begin
         param_name = "Thres";
         param_val = {13'd0,wah_threshold_in[6:4],4'h0,wah_threshold_in[3:0]}; //work out signed
         if (up_up) wah_threshold_in <= wah_threshold_in + 1;
         else if (down_down) wah_threshold_in <= wah_threshold_in - 1;
         else if (right_right) param_menu_name <= BLANK;
         else if (left_left) param_menu_name <= PERIO;
      end
      default: param_menu_name <= BLANK;
    endcase
end
MOOG:begin
    case(param_menu_name)
      BLANK:begin
         effect_name = "Moog ";
         param_name = "_____";
         param_val = "___";
         if (up_up) effect_menu_name <= WAH;
         else if (down_down) effect_menu_name <= REVERB;
         else if (right_right) param_menu_name <= WAVE;
         else if (left_left) param_menu_name <= PERIO;
      end
      WAVE:begin
         param_name = "Wave ";
         param_val = {22'd0,moog_wavesel};
         if (up_up) moog_wavesel <= moog_wavesel + 1;
         else if (down_down) moog_wavesel <= moog_wavesel - 1;
         else if (right_right) param_menu_name <= MOOG_AMOUNT;
         else if (left_left) param_menu_name <= BLANK;
      end
      MOOG_AMOUNT:begin
         param_name = "Amnt ";
         param_val = {14'd0,moog_amount[5:4],4'h0,moog_amount[3:0]};
         if (up_up) moog_amount <= moog_amount + 1;
         else if (down_down) moog_amount <= moog_amount - 1;
         else if (right_right) param_menu_name <= PERIO;
         else if (left_left) param_menu_name <= WAVE;
      end
      PERIO:begin
         param_name = "Speed";
         param_val = {6'd0,moog_period[9:8],4'h0,moog_period[7:4],4'h0,moog_period[3:0]};
         if (up_up) moog_period <= moog_period + 1;
         else if (down_down) moog_period <= moog_period - 1;
         else if (right_right) param_menu_name <= BLANK;
         else if (left_left) param_menu_name <= MOOG_AMOUNT;
      end
      default: param_menu_name <= BLANK;
    endcase
end
REVERB:begin
    case(param_menu_name)
      BLANK:begin
         effect_name = "Revrb";
         param_name = "_____";
         param_val = "___";
         if (up_up) effect_menu_name <= MOOG;
         else if (down_down) effect_menu_name <= CHORUS;
         else if (right_right) param_menu_name <= WAVON;
         else if (left_left) param_menu_name <= WVFRQ;
      end
      WAVON:begin
         param_name = "WavOn";
         param_val = rev_waveon ? " On" : "Off";
         if (up_up) rev_waveon <= rev_waveon + 1;
         else if (down_down) rev_waveon <= rev_waveon - 1;
         else if (right_right) param_menu_name <= WAVE;
         else if (left_left) param_menu_name <= BLANK;
      end
      WAVE:begin
         param_name = "Wave ";
         param_val = {22'd0,rev_wavesel};
         if (up_up) rev_wavesel <= rev_wavesel + 1;
         else if (down_down) rev_wavesel <= rev_wavesel - 1;
         else if (right_right) param_menu_name <= DELAY;
         else if (left_left) param_menu_name <= WAVON;
      end
      DELAY:begin
         param_name = "Delay";
         param_val = {6'd0,rev_delay_in[9:8],4'h0,rev_delay_in[7:4],4'h0,rev_delay_in[3:0]};
         if (up_up) rev_delay_in <= rev_delay_in + 1;
         else if (down_down) rev_delay_in <= rev_delay_in - 1;
         else if (right_right) param_menu_name <= DECAY;
         else if (left_left) param_menu_name <= WAVE;
      end
      DECAY:begin
         param_name = "Decay";
         param_val = {6'd0,rev_decay_in[9:8],4'h0,rev_decay_in[7:4],4'h0,rev_decay_in[3:0]};
```

```
            if (up_up) rev_decay_in <= rev_decay_in + 1;
            else if (down_down) rev_decay_in <= rev_decay_in - 1;
            else if (right_right) param_menu_name <= WVFRQ;
            else if (left_left) param_menu_name <= DELAY;
        end
        WVFRQ:begin
            param_name = "WvFrq";
            param_val = {5'd0,rev_wave_freq[10:8],4'h0,rev_wave_freq[7:4],4'h0,rev_wave_freq[3:0]};
            if (up_up) rev_wave_freq <= rev_wave_freq + 1;
            else if (down_down) rev_wave_freq <= rev_wave_freq - 1;
            else if (right_right) param_menu_name <= BLANK;
            else if (left_left) param_menu_name <= DECAY;
        end
        default: param_menu_name <= BLANK;
      endcase
    end
    CHORUS:begin
      case(param_menu_name)
        BLANK:begin
            effect_name = "Chor ";
            param_name = "_____";
            param_val = "___";
            if (up_up) effect_menu_name <= REVERB;
            else if (down_down) effect_menu_name <= TREMOLO;
            else if (right_right) param_menu_name <= CHOON;
            else if (left_left) param_menu_name <= DECAY;
        end
        CHOON:begin
            param_name = "ChoOn";
            param_val = choon ? " On" : "Off";
            if (up_up) choon <= choon + 1;
            else if (down_down) choon <= choon - 1;
            else if (right_right) param_menu_name <= WAVE;
            else if (left_left) param_menu_name <= BLANK;
        end
        WAVE:begin
            param_name = "Wave ";
            param_val = {22'd0,chor_wavesel};
            if (up_up) chor_wavesel <= chor_wavesel + 1;
            else if (down_down) chor_wavesel <= chor_wavesel - 1;
            else if (right_right) param_menu_name <= DELAY;
            else if (left_left) param_menu_name <= CHOON;
        end
        DELAY:begin
            param_name = "Delay";
            param_val = {6'd0,chor_delay_in[9:8],4'h0,chor_delay_in[7:4],4'h0,chor_delay_in[3:0]};
            if (up_up) chor_delay_in <= chor_delay_in + 1;
            else if (down_down) chor_delay_in <= chor_delay_in - 1;
            else if (right_right) param_menu_name <= DEPTH;
            else if (left_left) param_menu_name <= WAVE;
        end
        DEPTH:begin
            param_name = "Depth";
            param_val = {12'd0,chor_depth[7:4],4'h0,chor_depth[3:0]};
            if (up_up) chor_depth <= chor_depth + 1;
            else if (down_down) chor_depth <= chor_depth - 1;
            else if (right_right) param_menu_name <= WOBBL;
            else if (left_left) param_menu_name <= DELAY;
        end
        WOBBL:begin
            param_name = "Wobbl";
            param_val = {7'd0,chor_rate[8],4'd0,chor_rate[7:4],4'd0,chor_rate[3:0]};
            if (up_up) chor_rate <= chor_rate + 1;
            else if (down_down) chor_rate <= chor_rate - 1;
            else if (right_right) param_menu_name <= DECAY;
            else if (left_left) param_menu_name <= DEPTH;
        end
        DECAY:begin
            param_name = "Decay";
            param_val = {6'd0,chor_decay_in[9:8],4'h0,chor_decay_in[7:4],4'h0,chor_decay_in[3:0]};
            if (up_up) chor_decay_in <= chor_decay_in + 1;
            else if (down_down) chor_decay_in <= chor_decay_in - 1;
            else if (right_right) param_menu_name <= BLANK;
            else if (left_left) param_menu_name <= WOBBL;
        end
        default: param_menu_name <= BLANK;
      endcase
    end
    TREMOLO:begin
      case(param_menu_name)
        BLANK:begin
            effect_name = "Tremo";
            param_name = "_____";
            param_val = "___";
            if (up_up) effect_menu_name <= CHORUS;
            else if (down_down) effect_menu_name <= LONG_DELAY;
            else if (right_right) param_menu_name <= DECAY;
            else if (left_left) param_menu_name <= PERIO;
        end
        DECAY:begin
            param_name = "Decay";
            param_val = {7'd0,trem_decayval[8],4'h0,trem_decayval[7:4],4'h0,trem_decayval[3:0]}; //FIX SIGNED
```

```
         if (up_up) trem_decayval <= trem_decayval + 1;
         else if (down_down) trem_decayval <= trem_decayval - 1;
         else if (right_right) param_menu_name <= PERIO;
         else if (left_left) param_menu_name <= BLANK;
      end
      PERIO:begin
         param_name = "Perio";
         param_val = {6'd0,trem_halfperiod[9:8],4'h0,trem_halfperiod[7:4],4'h0,trem_halfperiod[3:0]};
         if (up_up) trem_halfperiod <= trem_halfperiod + 1;
         else if (down_down) trem_halfperiod <= trem_halfperiod - 1;
         else if (right_right) param_menu_name <= BLANK;
         else if (left_left) param_menu_name <= DECAY;
      end
      default: param_menu_name <= BLANK;
    endcase
  end
  LONG_DELAY:begin
    case(param_menu_name)
      BLANK:begin
         effect_name = "Delay";
         param_name = "_____";
         param_val = "___";
         if (up_up) effect_menu_name <= TREMOLO;
         else if (down_down) effect_menu_name <= FIVEBAND_EQ;
         else if (right_right) param_menu_name <= DELAY;
         else if (left_left) param_menu_name <= DECAY;
      end
      DELAY:begin
         param_name = "Delay";
         param_val = {6'd0,long_delay_in[9:8],4'h0,long_delay_in[7:4],4'h0,long_delay_in[3:0]};
         if (up_up) long_delay_in <= long_delay_in + 1;
         else if (down_down) long_delay_in <= long_delay_in - 1;
         else if (right_right) param_menu_name <= DECAY;
         else if (left_left) param_menu_name <= BLANK;
      end
      DECAY:begin
         param_name = "Decay";
         param_val = {6'd0,long_decay_in[9:8],4'h0,long_decay_in[7:4],4'h0,long_decay_in[3:0]};
         if (up_up) long_decay_in <= long_decay_in + 1;
         else if (down_down) long_decay_in <= long_decay_in - 1;
         else if (right_right) param_menu_name <= BLANK;
         else if (left_left) param_menu_name <= DELAY;
      end
      default: param_menu_name <= BLANK;
    endcase
  end
  FIVEBAND_EQ:begin
    case(param_menu_name)
      BLANK:begin
         effect_name = "5BdEQ";
         param_name = "_____";
         param_val = "___";
         if (up_up) effect_menu_name <= LONG_DELAY;
         else if (down_down) effect_menu_name <= PANNING;
         else if (right_right) param_menu_name <= LOWV;
         else if (left_left) param_menu_name <= HIV;
      end
      LOWV:begin
         param_name = "LowV ";
         param_val = {15'b0,lowvol[4],4'h0,lowvol[3:0]}; //CORRECT FOR SIGN
         if (up_up) lowvol <= lowvol + 1;
         else if (down_down) lowvol <= lowvol - 1;
         else if (right_right) param_menu_name <= LOMDV;
         else if (left_left) param_menu_name <= BLANK;
      end
      LOMDV:begin
         param_name = "LoMdV";
         param_val = {15'b0,lowmidvol[4],4'h0,lowmidvol[3:0]}; //CORRECT FOR SIGN
         if (up_up) lowmidvol <= lowmidvol + 1;
         else if (down_down) lowmidvol <= lowmidvol - 1;
         else if (right_right) param_menu_name <= MIDV;
         else if (left_left) param_menu_name <= LOWV;
      end
      MIDV:begin
         param_name = "MidV ";
         param_val = {15'b0,midvol[4],4'h0,midvol[3:0]}; //CORRECT FOR SIGN
         if (up_up) midvol <= midvol + 1;
         else if (down_down) midvol <= midvol - 1;
         else if (right_right) param_menu_name <= HIMDV;
         else if (left_left) param_menu_name <= LOMDV;
      end
      HIMDV:begin
         param_name = "HiMdV";
         param_val = {15'b0,highmidvol[4],4'h0,highmidvol[3:0]}; //CORRECT FOR SIGN
         if (up_up) highmidvol <= highmidvol + 1;
         else if (down_down) highmidvol <= highmidvol - 1;
         else if (right_right) param_menu_name <= HIV;
         else if (left_left) param_menu_name <= MIDV;
      end
      HIV:begin
         param_name = "HighV";
         param_val = {15'b0,highvol[4],4'h0,highvol[3:0]}; //CORRECT FOR SIGN
```

```
            if (up_up) highvol <= highvol + 1;
            else if (down_down) highvol <= highvol - 1;
            else if (right_right) param_menu_name <= BLANK;
            else if (left_left) param_menu_name <= HIMDV;
         end
         default: param_menu_name <= BLANK;
      endcase
   end
   PANNING:begin
      case(param_menu_name)
         BLANK:begin
            effect_name = "Pan  ";
            param_name = "_____";
            param_val = "___";
            if (up_up) effect_menu_name <= FIVEBAND_EQ;
            else if (down_down) effect_menu_name <= LOOPER;
            else if (right_right) param_menu_name <= FADE;
            else if (left_left) param_menu_name <= PERIO;
         end
         FADE:begin
            param_name = "Fade ";
            param_val = fade_on ? " On" : "Off";
            if (up_up) fade_on <= fade_on + 1;
            else if (down_down) fade_on <= fade_on - 1;
            else if (right_right) param_menu_name <= PERIO;
            else if (left_left) param_menu_name <= BLANK;
         end
         PERIO:begin
            param_name = fade_on ? "Speed" : "Perio";
            param_val = {12'd0,pan_speed_in[7:4],4'h0,pan_speed_in[3:0]}; //FIX SIGNED
            if (up_up) pan_speed_in <= pan_speed_in + 1;
            else if (down_down) pan_speed_in <= pan_speed_in - 1;
            else if (right_right) param_menu_name <= BLANK;
            else if (left_left) param_menu_name <= FADE;
         end
         default: param_menu_name <= BLANK;
      endcase
   end
   LOOPER:begin
      case(param_menu_name)
         BLANK:begin
            effect_name = "Loop ";
            param_name = "_____";
            param_val = "___";
            if (up_up) effect_menu_name <= PANNING;
            else if (down_down) effect_menu_name <= VOLUME;
            else if (right_right) param_menu_name <= LOOPER_ONOFF;
            else if (left_left) param_menu_name <= C3VOL;
         end
         LOOPER_ONOFF:begin
            param_name = "OnOff";
            param_val = (looper_on) ? "On " : "Off";
            if (up_up || down_down) looper_on <= ~looper_on;
            else if (right_right) param_menu_name <= LOOPER_LOOP;
            else if (left_left) param_menu_name <= BLANK;
         end
         LOOPER_LOOP:begin
            param_name = "Loop ";
            param_val = (looper_loop) ? "On " : "Off";
            if (up_up || down_down) looper_loop <= ~looper_loop;
            else if (right_right) param_menu_name <= C0VOL;
            else if (left_left) param_menu_name <= LOOPER_ONOFF;
         end
         C0VOL:begin
            param_name = "C0Vol";
            param_val = {20'd0,level_0ch[3:0]};
            if (up_up) level_0ch <= level_0ch + 1;
            else if (down_down) level_0ch <= level_0ch - 1;
            else if (right_right) param_menu_name <= C1VOL;
            else if (left_left) param_menu_name <= LOOPER_LOOP;
         end
         C1VOL:begin
            param_name = "C1Vol";
            param_val = {20'd0,level_1ch[3:0]};
            if (up_up) level_1ch <= level_1ch + 1;
            else if (down_down) level_1ch <= level_1ch - 1;
            else if (right_right) param_menu_name <= C2VOL;
            else if (left_left) param_menu_name <= C0VOL;
         end
         C2VOL:begin
            param_name = "C2Vol";
            param_val = {20'd0,level_2ch[3:0]};
            if (up_up) level_2ch <= level_2ch + 1;
            else if (down_down) level_2ch <= level_2ch - 1;
            else if (right_right) param_menu_name <= C3VOL;
            else if (left_left) param_menu_name <= C1VOL;
         end
         C3VOL:begin
            param_name = "C3Vol";
            param_val = {20'd0,level_3ch[3:0]};
            if (up_up) level_3ch <= level_3ch + 1;
            else if (down_down) level_3ch <= level_3ch - 1;
```

```
                else if (right_right) param_menu_name <= BLANK;
                else if (left_left) param_menu_name <= C2VOL;
             end
             default: param_menu_name <= BLANK;
           endcase
        end
        VOLUME:begin
           case(param_menu_name)
             BLANK:begin
                effect_name = "MstrV";
                param_name = "_____";
                param_val = "___";
                if (up_up) effect_menu_name <= LOOPER;
                else if (down_down) effect_menu_name <= DISTORTION;
                else if (right_right) param_menu_name <= VOLUM;
                else if (left_left) param_menu_name <= VOLUM;
             end
             VOLUM:begin
                param_name = "Volum";
                param_val = {15'b0,volume_out[4],4'h0,volume_out[3:0]};
                if (up_up) volume_out <= volume_out + 1;
                else if (down_down) volume_out <= volume_out - 1;
                else if (right_right) param_menu_name <= BLANK;
                else if (left_left) param_menu_name <= BLANK;
             end
             default: param_menu_name <= BLANK;
           endcase
        end
        default: effect_menu_name <= DISTORTION;
     endcase // case (effect_menu_name)
     // effects_out <= pan ? pan_out : {pan_in,pan_in};
     effects_out <= ((looper_on) ? looper_out : looper_in);

     //LED logic
     if ((looper_on & (looper_action == 2'b00)) | ~looper_on) //if looper on and playback off
       action_led <= 0;
     else if (looper_on & (looper_action == 2'b01))
       action_led <= 1;
     else if (looper_on & (looper_action == 2'b10)) begin
         action_led_counter <= action_led_counter + 1;
         if (action_led_counter == 23'd0)
           action_led <= ~action_led;
     end
   end // always @ (posedge clock)


   //EFFECT STREAM LOGIC
   assign dist_in = from_ac97_data >>> 1; //scales the input to half so that there is room for delay effects
   assign wah_in = disto ? dist_out : dist_in;
   assign moog_in = wah ? wah_out : wah_in;
   assign rev_in = moog ? moog_out : moog_in;
   assign chor_in = moog ? moog_out : moog_in;
   assign rev_chor_out = choon ? chor_out : rev_out;
   assign trem_in = rev_cho ? rev_chor_out : rev_in;
   assign long_del_in = trem ? trem_out : trem_in;
   assign eq_in = long_del ? long_del_out : long_del_in;
   assign pan_in = equalizer ? eq_out : eq_in;
   assign looper_in = pan ? pan_out : {pan_in,pan_in};

   assign volume = volume_out;

   // LED logic
   assign action_led_out = action_led;

   // Looper Input Logic
   assign loop_rec_0ch = looper_on ? rec_0ch : 1'b0;
   assign loop_rec_1ch = looper_on ? rec_1ch : 1'b0;
   assign loop_rec_2ch = looper_on ? rec_2ch : 1'b0;
   assign loop_rec_3ch = looper_on ? rec_3ch : 1'b0;
   assign loop_chan_on = looper_on ? chan_on : 1'b0;

   //ALPHANUMERIC DISPLAY
   assign string_data = {effect_name," ",param_name,"  ",param_val};

   //MODULES
   distortion dist_mod(clock,ready,threshold_in,overdrive_scale,gain_in,dist_in,dist_out);
   man_auto_wah wah_mod(clock,reset,ready,autowah_on,db_out[7:3],width_in,wah_period_in,wah_threshold_in,wah_in,wah_out);
   moogerfooger moog_mod(clock,ready,moog_wavesel,moog_period,moog_amount,moog_in,moog_out);
   reverb rev_mod(clock,reset,ready,rev_waveon,rev_wavesel,rev_delay_in,rev_decay_in,rev_wave_freq,rev_in,rev_out);
   chorus chor_mod(clock,reset,ready,chor_wavesel,chor_delay_in,chor_depth,chor_rate,chor_decay_in,chor_in,chor_out);
   tremolo trem_mod(clock,ready,trem_decayval,trem_halfperiod,trem_in,trem_out);
   longdel l_del_mod(clock,reset,ready,long_delay_in,long_decay_in,long_del_in,long_del_out);
   fivebandEQ eq_mod(clock,reset,ready,lowvol,lowmidvol,midvol,highmidvol,highvol,eq_in,eq_out);
   master_pan pan_mod(clock,reset,ready,fade_on,pan_speed_in,pan_in,pan_out);

   // the zbt looper
   looper_control
lc1(clock,reset,loop_rec_0ch,loop_rec_1ch,loop_rec_2ch,loop_rec_3ch,loop_chan_on,looper_chsel,looper_action,looper_ch_enable);
   looper
looper1(.clock(clock),.reset(reset),.ready(ready),.loop(looper_loop),.enter(chan_on),.action(looper_action),.channel_in_sel(lo
oper_chsel),.enable(looper_ch_enable),.volumes({level_3ch, level_2ch, level_1ch, level_0ch}),.data_in(looper_in),
```

```
.ram0_read_data(ram0_read_data),.ram1_read_data(ram1_read_data),.ram0_write_data(ram0_write_data),.ram1_write_data(ram1_write_
data),.ram0_we(ram0_we),.ram1_we(ram1_we),.ram0_addr(ram0_addr),.ram1_addr(ram1_addr),
                  .y(looper_out),
                  // for debugging
                  .state_out(state_out),
                  .addr_end_0_out(addr_end_0_out),
                  .addr_end_1_out(addr_end_1_out),
                  .addr_end_2_out(addr_end_2_out),
                  .addr_end_3_out(addr_end_3_out)
                  );

   button_scroll up(clock,scro_up,up_up);
   button_scroll down(clock,scro_down,down_down);
   button_scroll right(clock,scro_right,right_right);
   button_scroll left(clock,scro_left,left_left);

   adc wah_ped(clock,intr,db[7:0],db_out[7:0],cs,rd,wr);

   display_string ds(reset,clock, string_data, // CHANGE string_data TO ACTUAL
                     disp_blank, disp_clock, disp_rs, disp_ce_b,
                     disp_reset_b, disp_data_out);

endmodule // FXController

module adc(
           input wire clock,intr,
           input wire [7:0]db,
           output reg [7:0]db_out,
           output reg cs,rd,wr);

   reg [2:0]           counter = 3'd0;
   reg [1:0]           state = 2'd0;

   //STATES
   parameter WRITE = 2'd0;
   parameter READ = 2'd1;
   parameter RESET = 2'd2;

   always @(posedge clock) begin
      cs <= 0;
      case(state)
        WRITE:begin
           if (counter == 3'd6)begin
              wr <= 1;
              state <= READ;
              counter <= 0;
           end
           else begin
              wr <= 0;
              rd <= 1;
              counter <= counter + 1;
           end
        end
        READ:begin
           if (intr == 0) begin
              db_out[7:0] <= db[7:0];
              state <= RESET;
           end
        end
        RESET:begin
           rd <= 0;
           if (intr == 1)
             state <= WRITE;
        end
        default: state <= WRITE;
      endcase
   end
endmodule

//MODULE FOR SCROLLING
module button_scroll(
        input clock, button_in,
        output button_out);

        reg [24:0]counter = 0;
        reg old_val = 0;
        reg [1:0]state = 0;
        reg but_out = 0;
        reg [2:0]slow_count = 0;

        parameter BUT_OFF = 2'd0;
        parameter BUT_ON = 2'd1;
        parameter BUT_ON_SLOW = 2'd2;
        parameter BUT_ON_FAST = 2'd3;

        always @(posedge clock)begin
                case(state)
                BUT_OFF: begin
                        but_out <= 0;
                        slow_count <= 0;
                        if (button_in) begin
                                state <= BUT_ON;
```

```
                                counter <= counter + 1;
                                end
                        else counter <= 0;
                        end
                BUT_ON: begin
                        if (~button_in) begin
                                state <= BUT_OFF;
                                but_out <= 1;
                                end
                        else if (counter >= 25'd9000000)
                                state <= BUT_ON_SLOW;
                        else counter <= counter + 1;
                        end
                BUT_ON_SLOW: begin
                        if (slow_count >= 3'd5)
                                state <= BUT_ON_FAST;
                        else if (counter >= 25'd9000000) begin
                                but_out <= 1;
                                slow_count <= slow_count + 1;
                                counter <= 0;
                                end
                        else if (~button_in) begin
                                state <= BUT_OFF;
                                but_out <= 0;
                                end
                        else begin
                                counter <= counter + 1;
                                but_out <= 0;
                                end
                        end
                BUT_ON_FAST: begin
                        if (counter >= 25'd270000) begin
                                but_out <= 1;
                                counter <= 0;
                                end
                        else if (~button_in) begin
                                state <= BUT_OFF;
                                but_out <= 0;
                                end
                        else begin
                                counter <= counter + 1;
                                but_out <= 0;
                                end
                        end
                endcase
        end

        assign button_out = but_out;
endmodule

//////////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- 16 characer ASCII string display
//
//
// File:   display_string.v
// Date:   24-Sep-05
// Author: I. Chuang <ichuang@mit.edu>
//
// Based on Nathan Ickes' hex display code
//
// 28-Nov-2006 CJT: fixed race condition between CE and RS
//
// This module drives the labkit hex displays and shows the value of
// 8 ascii bytes as characters on the displays.
//
// Uses the Jae's ascii2dots module
//
// Inputs:
//
//   reset       - active high
//   clock_27mhz - the synchronous clock
//   string_data - 128 bits; each 8 bits gives an ASCII coded character
//
// Outputs:
//
//    disp_*     - display lines used in the 6.111 labkit (rev 003 & 004)
//
//////////////////////////////////////////////////////////////////////////////////

module display_string (reset, clock_27mhz, string_data,
                disp_blank, disp_clock, disp_rs, disp_ce_b,
                disp_reset_b, disp_data_out);

   input reset, clock_27mhz;      // clock and reset (active high reset)
   input [16*8-1:0] string_data;  // 8 ascii bytes to display

   output disp_blank, disp_clock, disp_data_out, disp_rs, disp_ce_b,
          disp_reset_b;

   reg disp_data_out, disp_rs, disp_ce_b, disp_reset_b;
```

```
//////////////////////////////////////////////////////////////////////////////
//
// Display Clock
//
// Generate a 500kHz clock for driving the displays.
//
//////////////////////////////////////////////////////////////////////////////

reg [4:0] count;
reg [7:0] reset_count;
reg clock;
wire dreset;

always @(posedge clock_27mhz)
  begin
      if (reset)
        begin
           count = 0;
           clock = 0;
        end
      else if (count == 26)
        begin
           clock = ~clock;
           count = 5'h00;
        end
      else
        count = count+1;
  end

always @(posedge clock_27mhz)
  if (reset)
    reset_count <= 100;
  else
    reset_count <= (reset_count==0) ? 0 : reset_count-1;

assign dreset = (reset_count != 0);

assign disp_clock = ~clock;

//////////////////////////////////////////////////////////////////////////////
//
// Display State Machine
//
//////////////////////////////////////////////////////////////////////////////

reg [7:0] state;               // FSM state
reg [9:0] dot_index;           // index to current dot being clocked out
reg [31:0] control;            // control register
reg [3:0] char_index; // index of current character
wire [39:0] dots;              // dots for a single digit
reg [39:0] rdots;              // pipelined dots
reg [7:0] ascii;               // ascii value of current character

assign disp_blank = 1'b0; // low <= not blanked

always @(posedge clock)
  if (dreset)
    begin
       state <= 0;
       dot_index <= 0;
       control <= 32'h7F7F7F7F;
    end
  else
    casex (state)
       8'h00:
         begin
            // Reset displays
            disp_data_out <= 1'b0;
            disp_rs <= 1'b0; // dot register
            disp_ce_b <= 1'b1;
            disp_reset_b <= 1'b0;
            dot_index <= 0;
            state <= state+1;
         end

       8'h01:
         begin
            // End reset
            disp_reset_b <= 1'b1;
            state <= state+1;
         end

       8'h02:
         begin
            // Initialize dot register (set all dots to zero)
            disp_ce_b <= 1'b0;
            disp_data_out <= 1'b0; // dot_index[0];
            if (dot_index == 639)
              state <= state+1;
            else
              dot_index <= dot_index+1;
         end
```

```
         8'h03:
           begin
              // Latch dot data
              disp_ce_b <= 1'b1;
              dot_index <= 31;            // re-purpose to init ctrl reg
              state <= state+1;
              disp_rs <= 1'b1; // Select the control register
           end

         8'h04:
           begin
              // Setup the control register
              disp_ce_b <= 1'b0;
              disp_data_out <= control[31];
              control <= {control[30:0], 1'b0};   // shift left
              if (dot_index == 0)
                state <= state+1;
              else
                dot_index <= dot_index-1;
              char_index <= 15;           // set this up early for pipeline
           end

         8'h05:
           begin
              // Latch the control register data / dot data
              disp_ce_b <= 1'b1;
              dot_index <= 39;            // init for single char
              rdots <= dots;              // store dots of char 15
              char_index <= 14;           // ready for next char
              state <= state+1;
              disp_rs <= 1'b0;                    // Select the dot register
           end

         8'h06:
           begin
              // Load the user's dot data into the dot reg, char by char
              disp_ce_b <= 1'b0;
              disp_data_out <= rdots[dot_index]; // dot data from msb
              if (dot_index == 0)
                if (char_index == 15)
                  state <= 5;                     // all done, latch data
                else
                begin
                  char_index <= char_index - 1;   // goto next char
                  dot_index <= 39;
                  rdots <= dots;          // latch in next char dots
                end
              else
                dot_index <= dot_index-1; // else loop thru all dots
           end

       endcase

   // combinatorial logic to generate dots for current character
   // this mux, and the ascii table lookup, are slow, so note that
   // this is pipelined by one display clock stage in the always
   // loop above.

   always @(string_data or char_index)
     case (char_index)
       4'h0: ascii = string_data[7:0];
       4'h1: ascii = string_data[7+1*8:1*8];
       4'h2: ascii = string_data[7+2*8:2*8];
       4'h3: ascii = string_data[7+3*8:3*8];
       4'h4: ascii = string_data[7+4*8:4*8];
       4'h5: ascii = string_data[7+5*8:5*8];
       4'h6: ascii = string_data[7+6*8:6*8];
       4'h7: ascii = string_data[7+7*8:7*8];
       4'h8: ascii = string_data[7+8*8:8*8];
       4'h9: ascii = string_data[7+9*8:9*8];
       4'hA: ascii = string_data[7+10*8:10*8];
       4'hB: ascii = string_data[7+11*8:11*8];
       4'hC: ascii = string_data[7+12*8:12*8];
       4'hD: ascii = string_data[7+13*8:13*8];
       4'hE: ascii = string_data[7+14*8:14*8];
       4'hF: ascii = string_data[7+15*8:15*8];
     endcase

   ascii2dots a2d(ascii,dots);

endmodule

////////////////////////////////////////////////////////////////////////////
// Display font dots generation from ASCII code

module ascii2dots(ascii_in,char_dots);

input [7:0] ascii_in;
output [39:0] char_dots;

  ////////////////////////////////////////////////////////////////////////
```

```
  // ROM: ASCII-->DOTS conversion
  ////////////////////////////////////////////////////////////////////////
        reg [39:0] char_dots;

        always @(ascii_in)
         case(ascii_in)
           8'h00: char_dots = 40'b00111110_01010001_01001001_01000101_00111110; //0 through F are for HEX display of 8'h0x HEX
input
      8'h01: char_dots = 40'b00000000_01000010_01111111_01000000_00000000;
      8'h02: char_dots = 40'b01100010_01010001_01001001_01001001_01000110;
       8'h03: char_dots = 40'b00100010_01000001_01001001_01001001_00110110;
       8'h04: char_dots = 40'b00011000_00010100_00010010_01111111_00010000;
       8'h05: char_dots = 40'b00100111_01000101_01000101_01000101_00111001;
       8'h06: char_dots = 40'b00111100_01001010_01001001_01001001_00110000;
       8'h07: char_dots = 40'b00000001_01110001_00001001_00000101_00000011;
       8'h08: char_dots = 40'b00110110_01001001_01001001_01001001_00110110;
       8'h09: char_dots = 40'b00000110_01001001_01001001_00101001_00011110;
       8'h0A: char_dots = 40'b01111110_00001001_00001001_00001001_01111110;
       8'h0B: char_dots = 40'b01111111_01001001_01001001_01001001_00110110;
       8'h0C: char_dots = 40'b00111110_01000001_01000001_01000001_00100010;
       8'h0D: char_dots = 40'b01111111_01000001_01000001_01000001_00111110;
       8'h0E: char_dots = 40'b01111111_01001001_01001001_01001001_01000001;
       8'h0F: char_dots = 40'b01111111_00001001_00001001_00001001_00000001;
                 8'h20:  char_dots = 40'b00000000_00000000_00000000_00000000_00000000; //  32        ' '
                 8'h21:  char_dots = 40'b00000000_00000000_00101111_00000000_00000000; //  33         !
                 8'h22:  char_dots = 40'b00000000_00000111_00000000_00000111_00000000; //  34        "
                 8'h23:  char_dots = 40'b00010100_00111110_00010100_00111110_00010100; //  35         #
                 8'h24:  char_dots = 40'b00000100_00101010_00111110_00101010_00010000; //  36         $
                 8'h25:  char_dots = 40'b00010011_00001000_00000100_00110010_00000000; //  37         %
                 8'h26:  char_dots = 40'b00010100_00101010_00010100_00100000_00000000; //  38         &
                 8'h27:  char_dots = 40'b00000000_00000000_00000111_00000000_00000000; //  39        '
                 8'h28:  char_dots = 40'b00000000_00011110_00100001_00000000_00000000;//  40  (
                 8'h29:  char_dots = 40'b00000000_00100001_00011110_00000000_00000000; //  41         )
                 8'h2A:  char_dots = 40'b00000000_00101010_00011100_00101010_00000000; //  42         *
                 8'h2B:  char_dots = 40'b00001000_00001000_00111110_00001000_00001000; //  43          +
                 8'h2C:  char_dots = 40'b00000000_01000000_00110000_00010000_00000000; //  44        ,
                 8'h2D:  char_dots = 40'b00001000_00001000_00001000_00001000_00000000; //  45         -
                 8'h2E:  char_dots = 40'b00000000_00110000_00110000_00000000_00000000; //  46         .
                 8'h2F:  char_dots = 40'b00010000_00001000_00000100_00000010_00000000; //  47         /
                 8'h30:  char_dots = 40'b00000000_00011110_00100001_00011110_00000000; //  48         0                --> 17
                 8'h31:  char_dots = 40'b00000000_00100010_00111111_00100000_00000000; //  49         1
                 8'h32:  char_dots = 40'b00100010_00110001_00101001_00100110_00000000; //  50         2
                 8'h33:  char_dots = 40'b00010001_00100101_00100101_00011011_00000000; //  51         3
                 8'h34:  char_dots = 40'b00001100_00001010_00111111_00001000_00000000; //  52         4
                 8'h35:  char_dots = 40'b00010111_00100101_00100101_00011001_00000000; //  53         5
                 8'h36:  char_dots = 40'b00011110_00100101_00100101_00011000_00000000; //  54         6
                 8'h37:  char_dots = 40'b00000001_00110001_00001101_00000011_00000000; //  55         7
                 8'h38:  char_dots = 40'b00011010_00100101_00100101_00011010_00000000; //  56         8
                 8'h39:  char_dots = 40'b00000110_00101001_00101001_00011110_00000000; //  57         9
                 8'h3A:  char_dots = 40'b00000000_00110110_00110110_00000000_00000000; //  58         :                --> 27
                 8'h3B:  char_dots = 40'b01000000_00110110_00010110_00000000_00000000; //  59         ;
                 8'h3C:  char_dots = 40'b00000000_00001000_00010100_00100010_00000000; //  60         <
                 8'h3D:  char_dots = 40'b00010100_00010100_00010100_00010100_00000000; //  61         =
                 8'h3E:  char_dots = 40'b00000000_00100010_00010100_00001000_00000000; //  62         >
                 8'h3F:  char_dots = 40'b00000000_00000010_00101001_00000110_00000000; //  63         ?
                 8'h40:  char_dots = 40'b00011110_00100001_00101101_00001110_00000000; //  64         @
                 8'h41:  char_dots = 40'b00111110_00001001_00001001_00111110_00000000; //  65         A                --> 34
                 8'h42:  char_dots = 40'b00111111_00100101_00100101_00011010_00000000; //  66         B
                 8'h43:  char_dots = 40'b00011110_00100001_00100001_00010010_00000000; //  67         C
                 8'h44:  char_dots = 40'b00111111_00100001_00100001_00011110_00000000; //  68         D
                 8'h45:  char_dots = 40'b00111111_00100101_00100101_00100001_00000000; //  69         E
                 8'h46:  char_dots = 40'b00111111_00000101_00000101_00000001_00000000; //  70         F
                 8'h47:  char_dots = 40'b00011110_00100001_00101001_00111010_00000000; //  71         G
                 8'h48:  char_dots = 40'b00111111_00000100_00000100_00111111_00000000; //  72         H
                 8'h49:  char_dots = 40'b00000000_00100001_00111111_00100001_00000000; //  73         I
                 8'h4A:  char_dots = 40'b00010000_00100000_00100000_00011111_00000000; //  74         J
                 8'h4B:  char_dots = 40'b00111111_00001100_00010010_00100001_00000000; //  75         K
                 8'h4C:  char_dots = 40'b00111111_00100000_00100000_00100000_00000000; //  76         L
                 8'h4D:  char_dots = 40'b00111111_00000110_00000110_00111111_00000000; //  77         M
                 8'h4E:  char_dots = 40'b00111111_00000110_00011000_00111111_00000000; //  78         N
                 8'h4F:  char_dots = 40'b00011110_00100001_00100001_00011110_00000000; //  79         O
                 8'h50:  char_dots = 40'b00111111_00001001_00001001_00000110_00000000; //  80         P
                 8'h51:  char_dots = 40'b00011110_00110001_00100001_01011110_00000000; //  81         Q
                 8'h52:  char_dots = 40'b00111111_00001001_00011001_00100110_00000000; //  82         R
                 8'h53:  char_dots = 40'b00010010_00100101_00101001_00010010_00000000; //  83         S
                 8'h54:  char_dots = 40'b00000000_00000001_00111111_00000001_00000000; //  84         T
                 8'h55:  char_dots = 40'b00011111_00100000_00100000_00011111_00000000; //  85         U
                 8'h56:  char_dots = 40'b00001111_00110000_00110000_00001111_00000000; //  86         V
                 8'h57:  char_dots = 40'b00111111_00011000_00011000_00111111_00000000; //  87         W
                 8'h58:  char_dots = 40'b00110011_00001100_00001100_00110011_00000000; //  88         X
                 8'h59:  char_dots = 40'b00000000_00000111_00111000_00000111_00000000; //  89         Y
                 8'h5A:  char_dots = 40'b00110001_00101001_00100101_00100011_00000000; //  90         Z                --> 59
                 8'h5B:  char_dots = 40'b00000000_00111111_00100001_00100001_00000000; //  91         [
                 8'h5C:  char_dots = 40'b00000010_00000100_00001000_00010000_00000000; //  92         \
                 8'h5D:  char_dots = 40'b00000000_00100001_00100001_00111111_00000000; //  93         ]
                 8'h5E:  char_dots = 40'b00000000_00000010_00000001_00000010_00000000; //  94         ^
                 8'h5F:  char_dots = 40'b00100000_00100000_00100000_00100000_00000000; //  95         _
                 8'h60:  char_dots = 40'b00000000_00000001_00000010_00000000_00000000; //  96         '
                 8'h61:  char_dots = 40'b00011000_00100100_00010100_00111100_00000000; //  97         a                --> 66
                 8'h62:  char_dots = 40'b00111111_00100100_00100100_00011000_00000000; //  98         b
                 8'h63:  char_dots = 40'b00011000_00100100_00100100_00000000_00000000; //  99         c
                 8'h64:  char_dots = 40'b00011000_00100100_00100100_00111111_00000000; // 100         d
```

```
                8'h65:  char_dots = 40'b00011000_00110100_00101100_00001000_00000000; // 101         e
                8'h66:  char_dots = 40'b00001000_00111110_00001001_00000010_00000000; // 102         f
                8'h67:  char_dots = 40'b00101000_01010100_01010100_01001100_00000000; // 103         g
                8'h68:  char_dots = 40'b00111111_00000100_00000100_00111000_00000000; // 104         h
                8'h69:  char_dots = 40'b00000000_00100100_00111101_00100000_00000000; // 105         i
                8'h6A:  char_dots = 40'b00000000_00100000_01000000_00111101_00000000; // 106         j
                8'h6B:  char_dots = 40'b00111111_00001000_00010100_00100000_00000000; // 107         k
                8'h6C:  char_dots = 40'b00000000_00100001_00111111_00100000_00000000; // 108         l
                8'h6D:  char_dots = 40'b00111100_00001000_00001100_00111000_00000000; // 109         m
                8'h6E:  char_dots = 40'b00111100_00000100_00000100_00111000_00000000; // 110         n
                8'h6F:  char_dots = 40'b00011000_00100100_00100100_00011000_00000000; // 111         o
                8'h70:  char_dots = 40'b01111100_00100100_00100100_00011000_00000000; // 112         p
                8'h71:  char_dots = 40'b00011000_00100100_00100100_01111100_00000000; // 113         q
                8'h72:  char_dots = 40'b00111100_00000100_00000100_00001000_00000000; // 114         r
                8'h73:  char_dots = 40'b00101000_00101100_00110100_00010100_00000000; // 115         s
                8'h74:  char_dots = 40'b00000100_00011111_00100100_00100000_00000000; // 116         t
                8'h75:  char_dots = 40'b00011100_00100000_00100000_00111100_00000000; // 117         u
                8'h76:  char_dots = 40'b00000000_00011100_00100000_00011100_00000000; // 118         v
                8'h77:  char_dots = 40'b00111100_00110000_00110000_00111100_00000000; // 119         w
                8'h78:  char_dots = 40'b00100100_00011000_00011000_00100100_00000000; // 120         x
                8'h79:  char_dots = 40'b00001100_01010000_00100000_00011100_00000000; // 121         y
                8'h7A:  char_dots = 40'b00100100_00110100_00101100_00100100_00000000; // 122         z               --> 91
                8'h7B:  char_dots = 40'b00000000_00000100_00011110_00100001_00000000; // 123         {
                8'h7C:  char_dots = 40'b00000000_00000000_00111111_00000000_00000000; // 124         |
                8'h7D:  char_dots = 40'b00000000_00100001_00011110_00000100_00000000; // 125         }
                8'h7E:  char_dots = 40'b00000010_00000001_00000010_00000001_00000000; // 126         ~               --> 95
                default:char_dots = 40'b01000001_01000001_01000001_01000001_01000001;
    endcase

endmodule

module chorus (input wire clock, reset, ready,
               input wire [1:0] wavesel,
               input wire [9:0] delay_in, // delay time, 0 to (2^10)/44100 = 22 ms
               input wire [7:0] depth, // LFO amplitude, 0 to 2^8
               input wire [8:0] rate, // LFO knob val (period control), 0 to (2^9)/48000 = 11 ms
               input wire signed [10:0] decay_in, // damping of previous signal value CHANGED TO SIGNED
               input wire signed [17:0] x,
               output reg signed [17:0] y);
   // wavesel vals
   parameter SQUARE=2'b00;
   parameter SIN=2'b01;
   parameter TRIANGLE=2'b10;
   parameter SAW=2'b11;
   // states
   parameter IDLE=3'd0;
   parameter READ=3'd1;
   parameter WRITE=3'd2;
   parameter PRE_READ = 3'd3;
   parameter PRE1_READ = 3'd4;
   parameter PRE_WRITE = 3'd5;

   reg signed [28:0]                    decayed_mem_out = 0;
   reg [13:0]                           delay_actual = 0;
   reg [2:0]                            state = IDLE;
   reg [13:0]                           address = 0;
   reg [13:0]                           old_address = 0;
   reg signed [17:0]                    mem_in = 0;
   wire signed [17:0]                   mem_out;
   reg                                  we;
   reg [9:0]                            delay = 0;
   reg signed [10:0]                    decay = 0;

   // TODO add depth to lfo, check signed convention
   wire signed [8:0]                    lfo_signed;
   wire [7:0]                           lfo_temp;
   reg [7:0]                            lfo;

   signalgen signalgen_reverb(.clock(clock), .ready(ready), .pos(1'b1), .wavesel(wavesel), .knobval({2'd0,rate}),
.y(lfo_signed));

   // convert to unsigned (we can only do this because pos is shorted to 1 above)
   assign lfo_temp = lfo_signed[7:0];


   // TODO deal with maxing out lfo_actual when it exceeds depth
   reg [15:0]                           mult;
   reg [9:0]                            lfo_actual;

   //TODO make sure the blocking assignments below are the correct method of attack.
   // Will using blocking assignments as below result in memory read/write issues?
   always @(posedge clock) begin
      // handle square wave amplitude (==1)
      delay <= delay_in;
      decay <= decay_in;
      lfo <= ((wavesel == SQUARE) ? ({lfo_temp[7], 7'd0}) : lfo_temp);
      mult <= lfo * depth;
      lfo_actual <= ((delay_actual < mult[15:6]) ? delay_actual[9:0] : mult[15:6]); // take the top 10 bits, clipping if
necessary
      delay_actual <= {delay,4'b0000}; //{delay,0,0,0,0};

      case(state)
```

```
        IDLE: begin
           we <= 0;
           if (ready) begin
              state <= PRE_READ;
              mem_in <= x;
              old_address <= address;
              address <= address - delay_actual + lfo_actual;
           end
        end
        PRE_READ:
          state <= PRE1_READ;
        PRE1_READ:begin
           decayed_mem_out <= (decay * mem_out) <<< 1;
           state <= READ; end
        READ: begin
           state <= PRE_WRITE;
           address <= old_address + 1; // origval + 1
           if (reset) begin // for resetting the memory to 0 at startup -- testing only
              y <= x;
              mem_in <= 0;
           end
           else begin
              y <= mem_in + decayed_mem_out[28:11]; // x is in mem_in here
              mem_in <= mem_in + decayed_mem_out[28:11];
           end
        end
        PRE_WRITE:
          state <= WRITE;
        WRITE: begin
           state <= IDLE;
           we <= 1;
        end
        default: state <= IDLE;
      endcase
   end

   // make a 1024 x 18 memory
   mybram #(.LOGSIZE(14),.WIDTH(18)) bram_chorus(.addr(address),.clk(clock),.din(mem_in),.dout(mem_out),.we(we));
endmodule
module distortion(
                  input wire clock, ready,
                  input wire signed [7:0]threshold_in,
                  input wire signed [6:0]overdrive_scale,
                  input wire signed [7:0]gain_in,
                  input wire signed [17:0]x,
                  output reg signed [17:0]y);

   reg signed [17:0]                       pos_threshold;
   reg signed [17:0]                       neg_threshold;
   reg signed [24:0]                       compressed;
   reg signed [17:0]                       post_comp;
   reg signed [25:0]                       preshift_out;
   reg [1:0]                               counter = 0;
   reg signed [17:0]                       old_x = 0;

   always @(*)begin //makes positive and negative threshold
      pos_threshold = {threshold_in, 10'sd0} >>> 1;
      neg_threshold = pos_threshold * -1;
   end

   always @(posedge clock) begin
      //compresses input by squaring signals greater than threshold
      old_x <= x;
      if (x >= pos_threshold)
        compressed <= (x-pos_threshold)*overdrive_scale;
      else if (x <= neg_threshold)
        compressed <= (x-neg_threshold)*overdrive_scale;
      else
        compressed <= {x,7'd0};

      if (old_x >= pos_threshold)
        post_comp <= compressed[24:7] + pos_threshold;
      else if (old_x <= neg_threshold)
        post_comp <= compressed[24:7] + neg_threshold;
      else
        post_comp <= compressed[24:7];
      //applies gain to compressed signal
      preshift_out <= (gain_in * post_comp) <<< 3;

      y <= preshift_out[25:8];
   end
endmodule
module fivebandEQ(
                  input wire clock, reset, ready,
                  input wire signed [5:0]lowvol,
                  input wire signed [5:0]lowmidvol,
                  input wire signed [5:0]midvol,
                  input wire signed [5:0]highmidvol,
                  input wire signed [5:0]highvol,
                  input wire signed [17:0]x,
                  output reg signed [17:0]y);
```

```
   reg signed [23:0]                        low_sc;
   reg signed [23:0]                        lowmid_sc;
   reg signed [23:0]                        mid_sc;
   reg signed [23:0]                        highmid_sc;
   reg signed [23:0]                        high_sc;
   reg signed [17:0]                        low_o;
   reg signed [17:0]                        lowmid_o;
   reg signed [17:0]                        mid_o;
   reg signed [17:0]                        highmid_o;
   reg signed [17:0]                        high_o;
   reg [1:0]                                counter = 0;
   wire signed [17:0]                       low;
   wire signed [17:0]                       lowmid;
   wire signed [17:0]                       mid;
   wire signed [17:0]                       highmid;
   wire signed [17:0]                       high;

   always @(posedge clock) begin
      if (ready) begin
         //scales bands by user input volumes
         low_sc <= lowvol*low;
         lowmid_sc <= lowmidvol*(lowmid-low);
         mid_sc <= midvol*(mid-lowmid);
         highmid_sc <= highmidvol*(highmid-mid);
         high_sc <= highvol*(high-highmid);
         counter <= 2'b01;
      end
      else if (counter <= 2'b01) begin
         //caps band outputs and converts to 18-bits from 22-bits
         low_o <= {low_sc[23],low_sc[21:5]};
         lowmid_o <= {lowmid_sc[23],lowmid_sc[21:5]};
         mid_o <= {mid_sc[23],mid_sc[21:5]};
         highmid_o <= {highmid_sc[23],highmid_sc[21:5]};
         high_o <= {high_sc[23],high_sc[21:5]};
         counter <= 2'b10;
      end
      else if (counter == 2'b10)begin
         y <= low_o + lowmid_o + mid_o + highmid_o + high_o; //outputs the combination of all bands
         counter <= 0;
      end
   end

   firlow125 lowfilter(clock, reset, ready, x, low);
   firlow275 lowmidfilter(clock, reset, ready, x, lowmid);
   firlow530 midfilter(clock, reset, ready, x, mid);
   firlow1100 highmidfilter(clock, reset, ready, x, highmid);
   firlow7000 highfilter(clock, reset, ready, x, high);
endmodule

module fivebandEQ(
                  input wire clock, reset, ready,
                  input wire signed [5:0]lowvol,
                  input wire signed [5:0]lowmidvol,
                  input wire signed [5:0]midvol,
                  input wire signed [5:0]highmidvol,
                  input wire signed [5:0]highvol,
                  input wire signed [17:0]x,
                  output reg signed [17:0]y);

   reg signed [23:0]                        low_sc;
   reg signed [23:0]                        lowmid_sc;
   reg signed [23:0]                        mid_sc;
   reg signed [23:0]                        highmid_sc;
   reg signed [23:0]                        high_sc;
   reg signed [17:0]                        low_o;
   reg signed [17:0]                        lowmid_o;
   reg signed [17:0]                        mid_o;
   reg signed [17:0]                        highmid_o;
   reg signed [17:0]                        high_o;
   reg [1:0]                                counter = 0;
   wire signed [17:0]                       low;
   wire signed [17:0]                       lowmid;
   wire signed [17:0]                       mid;
   wire signed [17:0]                       highmid;
   wire signed [17:0]                       high;

   always @(posedge clock) begin
      if (ready) begin
         //scales bands by user input volumes
         low_sc <= lowvol*low;
         lowmid_sc <= lowmidvol*(lowmid-low);
         mid_sc <= midvol*(mid-lowmid);
         highmid_sc <= highmidvol*(highmid-mid);
         high_sc <= highvol*(high-highmid);
         counter <= 2'b01;
      end
      else if (counter <= 2'b01) begin
         //caps band outputs and converts to 18-bits from 22-bits
         low_o <= {low_sc[23],low_sc[21:5]};
         lowmid_o <= {lowmid_sc[23],lowmid_sc[21:5]};
         mid_o <= {mid_sc[23],mid_sc[21:5]};
         highmid_o <= {highmid_sc[23],highmid_sc[21:5]};
```

```
            high_o <= {high_sc[23],high_sc[21:5]};
            counter <= 2'b10;
        end
        else if (counter == 2'b10)begin
            y <= low_o + lowmid_o + mid_o + highmid_o + high_o; //outputs the combination of all bands
            counter <= 0;
        end
    end

    firlow125 lowfilter(clock, reset, ready, x, low);
    firlow275 lowmidfilter(clock, reset, ready, x, lowmid);
    firlow530 midfilter(clock, reset, ready, x, mid);
    firlow1100 highmidfilter(clock, reset, ready, x, highmid);
    firlow7000 highfilter(clock, reset, ready, x, high);
endmodule `default_nettype none

//////////////////////////////////////////////////////////////////////////////
//
// Switch Debounce Module
//
//////////////////////////////////////////////////////////////////////////////

module debounce (
  input wire reset, clock, noisy,
  output reg clean
);
  reg [18:0] count;
  reg new;

  always @(posedge clock)
    if (reset) begin
      count <= 0;
      new <= noisy;
      clean <= noisy;
    end
    else if (noisy != new) begin
      // noisy input changed, restart the .01 sec clock
      new <= noisy;
      count <= 0;
    end
    else if (count == 270000)
      // noisy input stable for .01 secs, pass it along!
      clean <= new;
    else
      // waiting for .01 sec to pass
      count <= count+1;

endmodule

//////////////////////////////////////////////////////////////////////////////
//
// bi-directional monaural interface to AC97
//
//////////////////////////////////////////////////////////////////////////////

module lab5audio (
  input wire clock_27mhz,
  input wire reset,
  input wire [4:0] volume,
  output wire [7:0] audio_in_data,
  input wire [7:0] audio_out_data,
  output wire ready,
  output reg audio_reset_b,   // ac97 interface signals
  output wire ac97_sdata_out,
  input wire ac97_sdata_in,
  output wire ac97_synch,
  input wire ac97_bit_clock
);

  wire [7:0] command_address;
  wire [15:0] command_data;
  wire command_valid;
  wire [19:0] left_in_data, right_in_data;
  wire [19:0] left_out_data, right_out_data;

  // wait a little before enabling the AC97 codec
  reg [9:0] reset_count;
  always @(posedge clock_27mhz) begin
    if (reset) begin
      audio_reset_b = 1'b0;
      reset_count = 0;
    end else if (reset_count == 1023)
      audio_reset_b = 1'b1;
    else
      reset_count = reset_count+1;
  end

  wire ac97_ready;
  ac97 ac97(.ready(ac97_ready),
            .command_address(command_address),
            .command_data(command_data),
            .command_valid(command_valid),
```

```
            .left_data(left_out_data), .left_valid(1'b1),
            .right_data(right_out_data), .right_valid(1'b1),
            .left_in_data(left_in_data), .right_in_data(right_in_data),
            .ac97_sdata_out(ac97_sdata_out),
            .ac97_sdata_in(ac97_sdata_in),
            .ac97_synch(ac97_synch),
            .ac97_bit_clock(ac97_bit_clock));

  // ready: one cycle pulse synchronous with clock_27mhz
  reg [2:0] ready_sync;
  always @ (posedge clock_27mhz) ready_sync <= {ready_sync[1:0], ac97_ready};
  assign ready = ready_sync[1] & ~ready_sync[2];

  reg [7:0] out_data;
  always @ (posedge clock_27mhz)
    if (ready) out_data <= audio_out_data;
  assign audio_in_data = left_in_data[19:12];
  assign left_out_data = {out_data, 12'b000000000000};
  assign right_out_data = left_out_data;

  // generate repeating sequence of read/writes to AC97 registers
  ac97commands cmds(.clock(clock_27mhz), .ready(ready),
                    .command_address(command_address),
                    .command_data(command_data),
                    .command_valid(command_valid),
                    .volume(volume),
                    .source(3'b000));     // mic
endmodule

// assemble/disassemble AC97 serial frames
module ac97 (
  output reg ready,
  input wire [7:0] command_address,
  input wire [15:0] command_data,
  input wire command_valid,
  input wire [19:0] left_data,
  input wire left_valid,
  input wire [19:0] right_data,
  input wire right_valid,
  output reg [19:0] left_in_data, right_in_data,
  output reg ac97_sdata_out,
  input wire ac97_sdata_in,
  output reg ac97_synch,
  input wire ac97_bit_clock
);
  reg [7:0] bit_count;

  reg [19:0] l_cmd_addr;
  reg [19:0] l_cmd_data;
  reg [19:0] l_left_data, l_right_data;
  reg l_cmd_v, l_left_v, l_right_v;

  initial begin
    ready <= 1'b0;
    // synthesis attribute init of ready is "0";
    ac97_sdata_out <= 1'b0;
    // synthesis attribute init of ac97_sdata_out is "0";
    ac97_synch <= 1'b0;
    // synthesis attribute init of ac97_synch is "0";

    bit_count <= 8'h00;
    // synthesis attribute init of bit_count is "0000";
    l_cmd_v <= 1'b0;
    // synthesis attribute init of l_cmd_v is "0";
    l_left_v <= 1'b0;
    // synthesis attribute init of l_left_v is "0";
    l_right_v <= 1'b0;
    // synthesis attribute init of l_right_v is "0";

    left_in_data <= 20'h00000;
    // synthesis attribute init of left_in_data is "00000";
    right_in_data <= 20'h00000;
    // synthesis attribute init of right_in_data is "00000";
  end

  always @(posedge ac97_bit_clock) begin
    // Generate the sync signal
    if (bit_count == 255)
      ac97_synch <= 1'b1;
    if (bit_count == 15)
      ac97_synch <= 1'b0;

    // Generate the ready signal
    if (bit_count == 128)
      ready <= 1'b1;
    if (bit_count == 2)
      ready <= 1'b0;

    // Latch user data at the end of each frame. This ensures that the
    // first frame after reset will be empty.
    if (bit_count == 255) begin
      l_cmd_addr <= {command_address, 12'h000};
```

```
      l_cmd_data <= {command_data, 4'h0};
      l_cmd_v <= command_valid;
      l_left_data <= left_data;
      l_left_v <= left_valid;
      l_right_data <= right_data;
      l_right_v <= right_valid;
    end

    if ((bit_count >= 0) && (bit_count <= 15))
      // Slot 0: Tags
      case (bit_count[3:0])
        4'h0: ac97_sdata_out <= 1'b1;      // Frame valid
        4'h1: ac97_sdata_out <= l_cmd_v;   // Command address valid
        4'h2: ac97_sdata_out <= l_cmd_v;   // Command data valid
        4'h3: ac97_sdata_out <= l_left_v;  // Left data valid
        4'h4: ac97_sdata_out <= l_right_v; // Right data valid
        default: ac97_sdata_out <= 1'b0;
      endcase
    else if ((bit_count >= 16) && (bit_count <= 35))
      // Slot 1: Command address (8-bits, left justified)
      ac97_sdata_out <= l_cmd_v ? l_cmd_addr[35-bit_count] : 1'b0;
    else if ((bit_count >= 36) && (bit_count <= 55))
      // Slot 2: Command data (16-bits, left justified)
      ac97_sdata_out <= l_cmd_v ? l_cmd_data[55-bit_count] : 1'b0;
    else if ((bit_count >= 56) && (bit_count <= 75)) begin
      // Slot 3: Left channel
      ac97_sdata_out <= l_left_v ? l_left_data[19] : 1'b0;
      l_left_data <= { l_left_data[18:0], l_left_data[19] };
    end
    else if ((bit_count >= 76) && (bit_count <= 95))
      // Slot 4: Right channel
      ac97_sdata_out <= l_right_v ? l_right_data[95-bit_count] : 1'b0;
    else
      ac97_sdata_out <= 1'b0;

    bit_count <= bit_count+1;
  end // always @ (posedge ac97_bit_clock)

  always @(negedge ac97_bit_clock) begin
    if ((bit_count >= 57) && (bit_count <= 76))
      // Slot 3: Left channel
      left_in_data <= { left_in_data[18:0], ac97_sdata_in };
    else if ((bit_count >= 77) && (bit_count <= 96))
      // Slot 4: Right channel
      right_in_data <= { right_in_data[18:0], ac97_sdata_in };
  end
endmodule

// issue initialization commands to AC97
module ac97commands (
  input wire clock,
  input wire ready,
  output wire [7:0] command_address,
  output wire [15:0] command_data,
  output reg command_valid,
  input wire [4:0] volume,
  input wire [2:0] source
);
  reg [23:0] command;

  reg [3:0] state;
  initial begin
    command <= 4'h0;
    // synthesis attribute init of command is "0";
    command_valid <= 1'b0;
    // synthesis attribute init of command_valid is "0";
    state <= 16'h0000;
    // synthesis attribute init of state is "0000";
  end

  assign command_address = command[23:16];
  assign command_data = command[15:0];

  wire [4:0] vol;
  assign vol = 31-volume;  // convert to attenuation

  always @(posedge clock) begin
    if (ready) state <= state+1;

    case (state)
      4'h0: // Read ID
        begin
          command <= 24'h80_0000;
          command_valid <= 1'b1;
        end
      4'h1: // Read ID
        command <= 24'h80_0000;
      4'h3: // headphone volume
        command <= { 8'h04, 3'b000, vol, 3'b000, vol };
      4'h5: // PCM volume
        command <= 24'h18_0808;
      4'h6: // Record source select
```

```
        command <= { 8'h1A, 5'b00000, source, 5'b00000, source};
      4'h7: // Record gain = max
        command <= 24'h1C_0F0F;
      4'h9: // set +20db mic gain
        command <= 24'h0E_8048;
      4'hA: // Set beep volume
        command <= 24'h0A_0000;
      4'hB: // PCM out bypass mix1
        command <= 24'h20_8000;
      default:
        command <= 24'h80_0000;
    endcase // case(state)
  end // always @ (posedge clock)
endmodule // ac97commands

////////////////////////////////////////////////////////////////////////////////
//
// generate PCM data for 750hz sine wave (assuming f(ready) = 48khz)
//
////////////////////////////////////////////////////////////////////////////////

module tone750hz (
  input wire clock,
  input wire ready,
  output reg [19:0] pcm_data
);
   reg [8:0] index;

   initial begin
      index <= 8'h00;
      // synthesis attribute init of index is "00";
      pcm_data <= 20'h00000;
      // synthesis attribute init of pcm_data is "00000";
   end

   always @(posedge clock) begin
      if (ready) index <= index+1;
   end

   // one cycle of a sinewave in 64 20-bit samples
   always @(index) begin
      case (index[5:0])
        6'h00: pcm_data <= 20'h00000;
        6'h01: pcm_data <= 20'h0C8BD;
        6'h02: pcm_data <= 20'h18F8B;
        6'h03: pcm_data <= 20'h25280;
        6'h04: pcm_data <= 20'h30FBC;
        6'h05: pcm_data <= 20'h3C56B;
        6'h06: pcm_data <= 20'h471CE;
        6'h07: pcm_data <= 20'h5133C;
        6'h08: pcm_data <= 20'h5A827;
        6'h09: pcm_data <= 20'h62F20;
        6'h0A: pcm_data <= 20'h6A6D9;
        6'h0B: pcm_data <= 20'h70E2C;
        6'h0C: pcm_data <= 20'h7641A;
        6'h0D: pcm_data <= 20'h7A7D0;
        6'h0E: pcm_data <= 20'h7D8A5;
        6'h0F: pcm_data <= 20'h7F623;
        6'h10: pcm_data <= 20'h7FFFF;
        6'h11: pcm_data <= 20'h7F623;
        6'h12: pcm_data <= 20'h7D8A5;
        6'h13: pcm_data <= 20'h7A7D0;
        6'h14: pcm_data <= 20'h7641A;
        6'h15: pcm_data <= 20'h70E2C;
        6'h16: pcm_data <= 20'h6A6D9;
        6'h17: pcm_data <= 20'h62F20;
        6'h18: pcm_data <= 20'h5A827;
        6'h19: pcm_data <= 20'h5133C;
        6'h1A: pcm_data <= 20'h471CE;
        6'h1B: pcm_data <= 20'h3C56B;
        6'h1C: pcm_data <= 20'h30FBC;
        6'h1D: pcm_data <= 20'h25280;
        6'h1E: pcm_data <= 20'h18F8B;
        6'h1F: pcm_data <= 20'h0C8BD;
        6'h20: pcm_data <= 20'h00000;
        6'h21: pcm_data <= 20'hF3743;
        6'h22: pcm_data <= 20'hE7075;
        6'h23: pcm_data <= 20'hDAD80;
        6'h24: pcm_data <= 20'hCF044;
        6'h25: pcm_data <= 20'hC3A95;
        6'h26: pcm_data <= 20'hB8E32;
        6'h27: pcm_data <= 20'hAECC4;
        6'h28: pcm_data <= 20'hA57D9;
        6'h29: pcm_data <= 20'h9D0E0;
        6'h2A: pcm_data <= 20'h95927;
        6'h2B: pcm_data <= 20'h8F1D4;
        6'h2C: pcm_data <= 20'h89BE6;
        6'h2D: pcm_data <= 20'h85830;
        6'h2E: pcm_data <= 20'h8275B;
        6'h2F: pcm_data <= 20'h809DD;
        6'h30: pcm_data <= 20'h80000;
        6'h31: pcm_data <= 20'h809DD;
```

```
        6'h32: pcm_data <= 20'h8275B;
        6'h33: pcm_data <= 20'h85830;
        6'h34: pcm_data <= 20'h89BE6;
        6'h35: pcm_data <= 20'h8F1D4;
        6'h36: pcm_data <= 20'h95927;
        6'h37: pcm_data <= 20'h9D0E0;
        6'h38: pcm_data <= 20'hA57D9;
        6'h39: pcm_data <= 20'hAECC4;
        6'h3A: pcm_data <= 20'hB8E32;
        6'h3B: pcm_data <= 20'hC3A95;
        6'h3C: pcm_data <= 20'hCF044;
        6'h3D: pcm_data <= 20'hDAD80;
        6'h3E: pcm_data <= 20'hE7075;
        6'h3F: pcm_data <= 20'hF3743;
      endcase // case(index[5:0])
   end // always @ (index)
endmodule

//////////////////////////////////////////////////////////////////////////////////
////
//// 6.111 FPGA Labkit -- Template Toplevel Module
////
//// For Labkit Revision 004
//// Created: October 31, 2004, from revision 003 file
//// Author: Nathan Ickes, 6.111 staff
////
//////////////////////////////////////////////////////////////////////////////////

module lab5   (beep, audio_reset_b, ac97_sdata_out, ac97_sdata_in, ac97_synch,
               ac97_bit_clock,

               vga_out_red, vga_out_green, vga_out_blue, vga_out_sync_b,
               vga_out_blank_b, vga_out_pixel_clock, vga_out_hsync,
               vga_out_vsync,

               tv_out_ycrcb, tv_out_reset_b, tv_out_clock, tv_out_i2c_clock,
               tv_out_i2c_data, tv_out_pal_ntsc, tv_out_hsync_b,
               tv_out_vsync_b, tv_out_blank_b, tv_out_subcar_reset,

               tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1,
               tv_in_line_clock2, tv_in_aef, tv_in_hff, tv_in_aff,
               tv_in_i2c_clock, tv_in_i2c_data, tv_in_fifo_read,
               tv_in_fifo_clock, tv_in_iso, tv_in_reset_b, tv_in_clock,

               ram0_data, ram0_address, ram0_adv_ld, ram0_clk, ram0_cen_b,
               ram0_ce_b, ram0_oe_b, ram0_we_b, ram0_bwe_b,

               ram1_data, ram1_address, ram1_adv_ld, ram1_clk, ram1_cen_b,
               ram1_ce_b, ram1_oe_b, ram1_we_b, ram1_bwe_b,

               clock_feedback_out, clock_feedback_in,

               flash_data, flash_address, flash_ce_b, flash_oe_b, flash_we_b,
               flash_reset_b, flash_sts, flash_byte_b,

               rs232_txd, rs232_rxd, rs232_rts, rs232_cts,

               mouse_clock, mouse_data, keyboard_clock, keyboard_data,

               clock_27mhz, clock1, clock2,

               disp_blank, disp_data_out, disp_clock, disp_rs, disp_ce_b,
               disp_reset_b, disp_data_in,

               button0, button1, button2, button3, button_enter, button_right,
               button_left, button_down, button_up,

               switch,

               led,

               user1, user2, user3, user4,

               daughtercard,

               systemace_data, systemace_address, systemace_ce_b,
               systemace_we_b, systemace_oe_b, systemace_irq, systemace_mpbrdy,

               analyzer1_data, analyzer1_clock,
               analyzer2_data, analyzer2_clock,
               analyzer3_data, analyzer3_clock,
               analyzer4_data, analyzer4_clock);

   output beep, audio_reset_b, ac97_synch, ac97_sdata_out;
   input  ac97_bit_clock, ac97_sdata_in;

   output [7:0] vga_out_red, vga_out_green, vga_out_blue;
   output vga_out_sync_b, vga_out_blank_b, vga_out_pixel_clock,
          vga_out_hsync, vga_out_vsync;

   output [9:0] tv_out_ycrcb;
   output tv_out_reset_b, tv_out_clock, tv_out_i2c_clock, tv_out_i2c_data,
```

```
       tv_out_pal_ntsc, tv_out_hsync_b, tv_out_vsync_b, tv_out_blank_b,
       tv_out_subcar_reset;

input  [19:0] tv_in_ycrcb;
input  tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2, tv_in_aef,
       tv_in_hff, tv_in_aff;
output tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock, tv_in_iso,
       tv_in_reset_b, tv_in_clock;
inout  tv_in_i2c_data;

inout  [35:0] ram0_data;
output [18:0] ram0_address;
output ram0_adv_ld, ram0_clk, ram0_cen_b, ram0_ce_b, ram0_oe_b, ram0_we_b;
output [3:0] ram0_bwe_b;

inout  [35:0] ram1_data;
output [18:0] ram1_address;
output ram1_adv_ld, ram1_clk, ram1_cen_b, ram1_ce_b, ram1_oe_b, ram1_we_b;
output [3:0] ram1_bwe_b;

input  clock_feedback_in;
output clock_feedback_out;

inout  [15:0] flash_data;
output [23:0] flash_address;
output flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b, flash_byte_b;
input  flash_sts;

output rs232_txd, rs232_rts;
input  rs232_rxd, rs232_cts;

input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;

input  clock_27mhz, clock1, clock2;

output disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b;
input  disp_data_in;
output  disp_data_out;

input  button0, button1, button2, button3, button_enter, button_right,
       button_left, button_down, button_up;
input  [7:0] switch;
output [7:0] led;

inout [31:0] user1, user2, user3, user4;

inout [43:0] daughtercard;

inout  [15:0] systemace_data;
output [6:0]  systemace_address;
output systemace_ce_b, systemace_we_b, systemace_oe_b;
input  systemace_irq, systemace_mpbrdy;

output [15:0] analyzer1_data, analyzer2_data, analyzer3_data,
              analyzer4_data;
output analyzer1_clock, analyzer2_clock, analyzer3_clock, analyzer4_clock;

//////////////////////////////////////////////////////////////////////////
//
// I/O Assignments
//
//////////////////////////////////////////////////////////////////////////


// Audio Input and Output
assign beep= 1'b0;
//lab5 assign audio_reset_b = 1'b0;
//lab5 assign ac97_synch = 1'b0;
//lab5 assign ac97_sdata_out = 1'b0;
// ac97_sdata_in is an input

// VGA Output
assign vga_out_red = 10'h0;
assign vga_out_green = 10'h0;
assign vga_out_blue = 10'h0;
assign vga_out_sync_b = 1'b1;
assign vga_out_blank_b = 1'b1;
assign vga_out_pixel_clock = 1'b0;
assign vga_out_hsync = 1'b0;
assign vga_out_vsync = 1'b0;

// Video Output
assign tv_out_ycrcb = 10'h0;
assign tv_out_reset_b = 1'b0;
assign tv_out_clock = 1'b0;
assign tv_out_i2c_clock = 1'b0;
assign tv_out_i2c_data = 1'b0;
assign tv_out_pal_ntsc = 1'b0;
assign tv_out_hsync_b = 1'b1;
assign tv_out_vsync_b = 1'b1;
assign tv_out_blank_b = 1'b1;
assign tv_out_subcar_reset = 1'b0;
```

```
// Video Input
assign tv_in_i2c_clock = 1'b0;
assign tv_in_fifo_read = 1'b0;
assign tv_in_fifo_clock = 1'b0;
assign tv_in_iso = 1'b0;
assign tv_in_reset_b = 1'b0;
assign tv_in_clock = 1'b0;
assign tv_in_i2c_data = 1'bZ;
// tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2,
// tv_in_aef, tv_in_hff, and tv_in_aff are inputs

// SRAMs
assign ram0_data = 36'hZ;
assign ram0_address = 19'h0;
assign ram0_adv_ld = 1'b0;
assign ram0_clk = 1'b0;
assign ram0_cen_b = 1'b1;
assign ram0_ce_b = 1'b1;
assign ram0_oe_b = 1'b1;
assign ram0_we_b = 1'b1;
assign ram0_bwe_b = 4'hF;
assign ram1_data = 36'hZ;
assign ram1_address = 19'h0;
assign ram1_adv_ld = 1'b0;
assign ram1_clk = 1'b0;
assign ram1_cen_b = 1'b1;
assign ram1_ce_b = 1'b1;
assign ram1_oe_b = 1'b1;
assign ram1_we_b = 1'b1;
assign ram1_bwe_b = 4'hF;
assign clock_feedback_out = 1'b0;
// clock_feedback_in is an input

// Flash ROM
assign flash_data = 16'hZ;
assign flash_address = 24'h0;
assign flash_ce_b = 1'b1;
assign flash_oe_b = 1'b1;
assign flash_we_b = 1'b1;
assign flash_reset_b = 1'b0;
assign flash_byte_b = 1'b1;
// flash_sts is an input

// RS-232 Interface
assign rs232_txd = 1'b1;
assign rs232_rts = 1'b1;
// rs232_rxd and rs232_cts are inputs

// PS/2 Ports
// mouse_clock, mouse_data, keyboard_clock, and keyboard_data are inputs

// LED Displays
assign disp_blank = 1'b1;
assign disp_clock = 1'b0;
assign disp_rs = 1'b0;
assign disp_ce_b = 1'b1;
assign disp_reset_b = 1'b0;
assign disp_data_out = 1'b0;
// disp_data_in is an input

// Buttons, Switches, and Individual LEDs
//lab5 assign led = 8'hFF;
// button0, button1, button2, button3, button_enter, button_right,
// button_left, button_down, button_up, and switches are inputs

// User I/Os
//assign user1 = 32'hZ;
assign user2 = 32'hZ;
assign user3 = 32'hZ;
assign user4 = 32'hZ;

// Daughtercard Connectors
assign daughtercard = 44'hZ;

// SystemACE Microprocessor Port
assign systemace_data = 16'hZ;
assign systemace_address = 7'h0;
assign systemace_ce_b = 1'b1;
assign systemace_we_b = 1'b1;
assign systemace_oe_b = 1'b1;
// systemace_irq and systemace_mpbrdy are inputs

// Logic Analyzer
//lab5 assign analyzer1_data = 16'h0;
//lab5 assign analyzer1_clock = 1'b1;
assign analyzer2_data = 16'h0;
assign analyzer2_clock = 1'b1;
//lab5 assign analyzer3_data = 16'h0;
//lab5 assign analyzer3_clock = 1'b1;
assign analyzer4_data = 16'h0;
assign analyzer4_clock = 1'b1;
```

```
//   wire [7:0] from_ac97_data, to_ac97_data;
//   wire ready;

   //////////////////////////////////////////////////////////////////////
   //
   // Reset Generation
   //
   // A shift register primitive is used to generate an active-high reset
   // signal that remains high for 16 clock cycles after configuration finishes
   // and the FPGA's internal clocks begin toggling.
   //
   //////////////////////////////////////////////////////////////////////
   wire reset;
   SRL16 #(.INIT(16'hFFFF)) reset_sr(.D(1'b0), .CLK(clock_27mhz), .Q(reset),
                                     .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));

   wire [7:0] from_ac97_data, to_ac97_data;
   wire ready;

   // allow user to adjust volume
   wire vup,vdown;
   reg old_vup,old_vdown;
   debounce bup(.reset(reset),.clock(clock_27mhz),.noisy(~button_up),.clean(vup));
   debounce bdown(.reset(reset),.clock(clock_27mhz),.noisy(~button_down),.clean(vdown));
   reg [4:0] volume;
   always @ (posedge clock_27mhz) begin
     if (reset) volume <= 5'd8;
     else begin
        if (vup & ~old_vup & volume != 5'd31) volume <= volume+1;
        if (vdown & ~old_vdown & volume != 5'd0) volume <= volume-1;
     end
     old_vup <= vup;
     old_vdown <= vdown;
   end

   // AC97 driver
   lab5audio a(clock_27mhz, reset, volume, from_ac97_data, to_ac97_data, ready,
               audio_reset_b, ac97_sdata_out, ac97_sdata_in,
               ac97_synch, ac97_bit_clock);

   // ************************************
   // FYI: call to recorder was here. duh.
   // ************************************

   // output useful things to the logic analyzer connectors
   assign analyzer1_clock = ac97_bit_clock;
   assign analyzer1_data[0] = audio_reset_b;
   assign analyzer1_data[1] = ac97_sdata_out;
   assign analyzer1_data[2] = ac97_sdata_in;
   assign analyzer1_data[3] = ac97_synch;
   assign analyzer1_data[15:4] = 0;

   assign analyzer3_clock = ready;
   assign analyzer3_data = {from_ac97_data, to_ac97_data};
endmodule
module longdel #(parameter [4:0] LOGSIZE=5'd16 // "max delay"--size of allocated bram (used in reverb)
                 )(input wire clock, reset, ready,
                   input wire [9:0] delay_in,
                   input wire [9:0] decay_in,
                   input wire signed [17:0] x,
                   output reg signed [17:0] y);
   parameter [2:0] IDLE = 3'b000; // default state
   parameter [2:0] GETDELAY1 = 3'b010; // fetch the delay sample from bram
   parameter [2:0] GETDELAY2 = 3'b100; // fetch the delay sample from bram, stage 2
   parameter [2:0] WRITE_PRE1 = 3'b101; // add delay sample to current, output
   parameter [2:0] WRITE_PRE2 = 3'b110; // add delay sample to current, output
   parameter [2:0] WRITE = 3'b111; // add delay sample to current, output

   reg [LOGSIZE-1:0]                    address = 0;
   reg [9:0]                            delay_val = 0;
   wire [LOGSIZE-1:0]                   delay_actual;

   assign delay_actual[LOGSIZE-(5'd11):0] = 0;
   assign delay_actual[LOGSIZE-(5'd1):LOGSIZE-(5'd10)] = delay_val;

   reg [9:0]                            decay_val = 0;
   wire signed [10:0]                   decay_signed;
   assign decay_signed = {1'b0, decay_val};
   wire                                 we;
   reg signed [17:0]                    mem_in = 0;
   wire signed [17:0]                   mem_out;
   reg [3:0]                            state = 4'b0000; // last bit is for downsampling

   reg signed [27:0]                    temp = 0;

   // only write in WRITE phase
   assign we = (state == {WRITE, 1'b0});

   // make a LOGSIZE x 18 memory
   mybram #(.LOGSIZE(LOGSIZE),.WIDTH(18))
   bram_longdel(.addr(address),.clk(clock),.we(we),.din(mem_in),.dout(mem_out));
```

```
   always @(posedge clock) begin
      case(state[3:1])
        IDLE: begin // on ready, place new value in buffer (every other time)
           if (ready) begin
              // grab the delay val (must stay constant due to impl.)
              delay_val <= delay_in;
              decay_val <= decay_in;
              // downsampling
              state[0] <= ~state[0]; // flip for next ready pulse
              if (state[0]) begin
                 // read delay value on the next cycle
                 address <= address + 1 - delay_actual;
                 // go to GETDELAY, but preserve downsampling value
                 state[3:1] <= GETDELAY1;
              end // else, stay in IDLE until the next ready pulse
           end
        end // case: IDLE
        GETDELAY1: begin // grab the delayed sample
           // wait
           state[3:1] <= GETDELAY2;
        end // case: GETDELAY1
        GETDELAY2: begin // once we have the delayed sample, multiply
           // write the value
           state[3:1] <= WRITE_PRE1;
           temp <= (mem_out * decay_signed);
        end // case: GETDELAY2
        WRITE_PRE1: begin // perform downshifting
           state[3:1] <= WRITE_PRE2;
           address <= address + delay_actual; // reset address
           mem_in <= temp >>> 4'd10;
        end // case: WRITE_PRE1
        WRITE_PRE2: begin // output the correct value
           // write the value
           state[3:1] <= WRITE;
           if (reset) begin // just for testing, since mem_out = XXXX initially
              y <= x;
              mem_in <= x;
           end else begin
              y <= x + mem_in;
              mem_in <= x + mem_in;
              //y <= x + (mem_out >>> 1);
              //mem_in <= x + (mem_out >>> 1);
           end
        end // case: GETDELAY2
        WRITE: begin // writing the new output, go to idle
           state[3:1] <= IDLE;
        end // case: GETDELAY
      endcase
   end

endmodule
module moogerfooger(input clock, ready,
                    input [1:0] wavesel,
                    input [9:0] period,
                    input [5:0] amount, // how much moogerfooger? max = all moogerfooger, min = none
                    input wire signed [17:0] x,
                    output reg signed [17:0] y);

   parameter [1:0] SQUARE=2'b00;
   parameter [1:0] SIN=2'b01;
   parameter [1:0] TRIANGLE=2'b10;
   parameter [1:0] SAW=2'b11;

   wire signed [8:0]                        sig;
   wire signed [1:0]                        squaresig;
   assign squaresig = {sig[8], sig[0]};

   // stage 1
   reg signed [19:0]                        ypre_sq;
   reg signed [26:0]                        ypre;
   // stage 2
   reg signed [17:0]                        y3;
   // stage 3
   reg signed [23:0]                        yscaled_raw;
   // x pipelining
   reg signed [17:0]                        x2;
   reg signed [17:0]                        x3;

   // the signal generator
   signalgen signalgen_moog (.clock(clock), .ready(ready), .pos(1'b0), .wavesel(wavesel), .knobval({period, 1'b0}), .y(sig));

   // note: could be one assign statement, but this is more readable
   always @(posedge clock) begin
      if (ready) begin
         // pipeline x
         x2 <= x;
         x3 <= x2;
         case(wavesel)
           SQUARE: begin// downshift by 1,
              // stage 1 -- moogerfoock it
              ypre_sq <= (x * squaresig);
```

```
            // stage 2 -- shift to get the output value
            y3 <= ypre_sq[19:2]<<<1;
         end
         default: begin
            // stage 1 -- moogerfoock it
            ypre <= (x * sig);
            // stage 2 -- shift to get the output value
            y3 <= ypre[26:9]<<<1;
         end
       endcase
       // stage 3 -- multiply by value and add
       yscaled_raw <= (y3 * amount) + (x3 * (6'b111111 - amount));
       // stage 4 -- scale
       y <= yscaled_raw[23:6];
    end
  end
endmodule // moogerfooger
// Fade Pan
module fade_pan(
                input wire clock, reset, ready,
                input wire signed [8:0]speed_in,
                input wire signed [17:0]x,
                output reg signed [35:0]y);

  reg signed [29:0]                     counter = 0;
  reg signed [29:0]                     anti_counter = 0;
  reg signed [47:0]                     pre_scale_left = 0;
  reg signed [47:0]                     pre_scale_right = 0;
  reg                                   direction = 0;
  reg signed [29:0]                     constant = 30'sb00111111111111111111111111111111;

  always @(posedge clock) begin
     anti_counter <= constant - counter;
     pre_scale_left <= (counter*x) <<< 2;
     pre_scale_right <= (anti_counter*x) <<< 2;
     if (counter >= constant)//if counter exceeds value change direction
       direction <= 1'b1;
     else if (counter <= 0)
       direction <= 1'b0;
     case(direction)
       1'b0:
         counter <= counter + speed_in;
       1'b1:
         counter <= counter - speed_in;
     endcase
     y <= {pre_scale_left[47:30],pre_scale_right[47:30]};
  end
endmodule

// Ping Pong Pan
module pong_pan(
                input wire clock, reset, ready,
                input wire signed [8:0]speed_in,
                input wire signed [17:0]x,
                output reg signed [35:0]y);

  reg [25:0]                            counter = 0;
  reg signed [17:0]                     pre_scale_left = 0;
  reg signed [17:0]                     pre_scale_right = 0;
  reg                                   direction = 0;
  reg [24:0]                            speed;

  always @(posedge clock) begin
     speed <= {speed_in[7:0],17'd0};
     if (counter >= speed) begin //if counter exceeds value change direction
         counter <= 0;
         direction <= ~direction; end
     else begin
         case(direction)
           1'b0:begin
              pre_scale_left <= x;
              pre_scale_right <= 0;
              counter <= counter + 1; end
           1'b1:begin
              pre_scale_right <= x;
              pre_scale_left <= 0;
              counter <= counter + 1; end
         endcase
     end
     y <= {pre_scale_left[17:0],pre_scale_right[17:0]};
  end


endmodule

// Combined Pan
module master_pan(
                  input wire clock, reset, ready, fade_on,
                  input wire signed [8:0]speed_in,
                  input wire signed [17:0]x,
                  output reg signed [35:0]y);
```

```
   wire signed [35:0]                      fade_out;
   wire signed [35:0]                      pong_out;
   always @(posedge clock)begin
      y <= (fade_on) ? fade_out : pong_out;
   end

   pong_pan pong(clock,reset,ready,speed_in,x,pong_out);
   fade_pan fade(clock,reset,ready,speed_in,x,fade_out);
endmodule
module reverb #(parameter [4:0] LOGSIZE = 5'd13)(input wire clock, reset, ready, waveon,
                                                 input wire [1:0] wavesel,
                                                 input wire [9:0] delay_in,
                                                 input wire [9:0] decay_in,
                                                 input wire [10:0] decay_knob_val,
                                                 input wire signed [17:0] x,
                                                 output wire signed [17:0] y);

   parameter [1:0] SQUARE=2'b00;
   parameter [1:0] SIN=2'b01;
   parameter [1:0] TRIANGLE=2'b10;
   parameter [1:0] SAW=2'b11;

   reg [9:0]                                                                  decay_value;
   wire signed [8:0]                                                          decay_value_wave;
   reg signed [8:0]                                                           decay_value_wave_actual;
   reg [17:0]                                                                 mult;
   reg [9:0]                                                                  shifted;

   // the wave value to use (only positive)
   signalgen signalgen_reverb(.clock(clock), .ready(ready), .pos(1'b1), .wavesel(wavesel), .knobval(decay_knob_val),
.y(decay_value_wave));

   // TODO support raising wave to a nonzero bottom value, to prevent decay value dropping to 0
   always @(posedge clock)begin
      decay_value_wave_actual <= ((wavesel == SQUARE) ? decay_value_wave : (9'sd255 - decay_value_wave));
      mult <= decay_value_wave_actual[7:0] * decay_in;
      shifted <= mult[17:8];
      decay_value <= (waveon ? shifted : decay_in);
   end

   // the delay instance
   // note: 2 * (2**13) / 44100.0 = 371.5 ms ("max delay" of instantiated longdel)
   longdel #(.LOGSIZE(LOGSIZE)) longdel_reverb1 (.clock(clock),.reset(reset),.ready(ready),
                                                 .delay_in(delay_in),
                                                 .decay_in(decay_value),
                                                 .x(x), .y(y));
endmodule // reverb
//////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- ZBT RAM clock generation
//
//
// Created: April 27, 2004
// Author: Nathan Ickes
//
//////////////////////////////////////////////////////////////////////////////
//
// This module generates deskewed clocks for driving the ZBT SRAMs and FPGA
// registers. A special feedback trace on the labkit PCB (which is length
// matched to the RAM traces) is used to adjust the RAM clock phase so that
// rising clock edges reach the RAMs at exactly the same time as rising clock
// edges reach the registers in the FPGA.
//
// The RAM clock signals are driven by DDR output buffers, which further
// ensures that the clock-to-pad delay is the same for the RAM clocks as it is
// for any other registered RAM signal.
//
// When the FPGA is configured, the DCMs are enabled before the chip-level I/O
// drivers are released from tristate. It is therefore necessary to
// artificially hold the DCMs in reset for a few cycles after configuration.
// This is done using a 16-bit shift register. When the DCMs have locked, the
// <lock> output of this mnodule will go high. Until the DCMs are locked, the
// ouput clock timings are not guaranteed, so any logic driven by the
// <fpga_clock> should probably be held inreset until <locked> is high.
//
//////////////////////////////////////////////////////////////////////////////

module ramclock(ref_clock, fpga_clock, ram0_clock, ram1_clock,
                clock_feedback_in, clock_feedback_out, locked);

   input ref_clock;                // Reference clock input
   output fpga_clock;              // Output clock to drive FPGA logic
   output ram0_clock, ram1_clock;  // Output clocks for each RAM chip
   input  clock_feedback_in;       // Output to feedback trace
   output clock_feedback_out;      // Input from feedback trace
   output locked;                  // Indicates that clock outputs are stable

   wire   ref_clk, fpga_clk, ram_clk, fb_clk, lock1, lock2, dcm_reset;

   //////////////////////////////////////////////////////////////////////////

   //IBUFG ref_buf (.O(ref_clk), .I(ref_clock));
```

```
   assign ref_clk = ref_clock;
   BUFG int_buf (.O(fpga_clock), .I(fpga_clk));

   DCM int_dcm (.CLKFB(fpga_clock),
                .CLKIN(ref_clk),
                .RST(dcm_reset),
                .CLK0(fpga_clk),
                .LOCKED(lock1));
   // synthesis attribute DLL_FREQUENCY_MODE of int_dcm is "LOW"
   // synthesis attribute DUTY_CYCLE_CORRECTION of int_dcm is "TRUE"
   // synthesis attribute STARTUP_WAIT of int_dcm is "FALSE"
   // synthesis attribute DFS_FREQUENCY_MODE of int_dcm is "LOW"
   // synthesis attribute CLK_FEEDBACK of int_dcm  is "1X"
   // synthesis attribute CLKOUT_PHASE_SHIFT of int_dcm is "NONE"
   // synthesis attribute PHASE_SHIFT of int_dcm is 0

   BUFG ext_buf (.O(ram_clock), .I(ram_clk));

   IBUFG fb_buf (.O(fb_clk), .I(clock_feedback_in));

   DCM ext_dcm (.CLKFB(fb_clk),
                .CLKIN(ref_clk),
                .RST(dcm_reset),
                .CLK0(ram_clk),
                .LOCKED(lock2));
   // synthesis attribute DLL_FREQUENCY_MODE of ext_dcm is "LOW"
   // synthesis attribute DUTY_CYCLE_CORRECTION of ext_dcm is "TRUE"
   // synthesis attribute STARTUP_WAIT of ext_dcm is "FALSE"
   // synthesis attribute DFS_FREQUENCY_MODE of ext_dcm is "LOW"
   // synthesis attribute CLK_FEEDBACK of ext_dcm  is "1X"
   // synthesis attribute CLKOUT_PHASE_SHIFT of ext_dcm is "NONE"
   // synthesis attribute PHASE_SHIFT of ext_dcm is 0

   SRL16 dcm_rst_sr (.D(1'b0), .CLK(ref_clk), .Q(dcm_reset),
                     .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
   // synthesis attribute init of dcm_rst_sr is "000F";


   OFDDRRSE ddr_reg0 (.Q(ram0_clock), .C0(ram_clock), .C1(~ram_clock),
                      .CE (1'b1), .D0(1'b1), .D1(1'b0), .R(1'b0), .S(1'b0));
   OFDDRRSE ddr_reg1 (.Q(ram1_clock), .C0(ram_clock), .C1(~ram_clock),
                      .CE (1'b1), .D0(1'b1), .D1(1'b0), .R(1'b0), .S(1'b0));
   OFDDRRSE ddr_reg2 (.Q(clock_feedback_out), .C0(ram_clock), .C1(~ram_clock),
                      .CE (1'b1), .D0(1'b1), .D1(1'b0), .R(1'b0), .S(1'b0));

   assign locked = lock1 && lock2;

endmodule
// Some modules for generating signals

module square #(parameter [5:0] IN_W=6'd18
                )(input wire clock, ready,
                  // period goes from 1 to ((2^18) - 1), or from (1/44.1k) sec to almost 6 sec
                  input wire [IN_W-1:0] knobval,
                  output reg out);
   reg [IN_W-1:0]             counter = 0;

   initial begin
      out = 0;
   end

   always @(posedge clock) begin
      if (ready) begin
          if (counter == knobval) begin
             counter <= 0;
             out <= ~out;
          end
          else
            counter <= counter + 1;
      end
   end
endmodule // square

module sin #(parameter [7:0] MOD = 8'd6
             )(
               input wire clock, ready,
               input wire [10:0] knobval,
               output wire signed [8:0] out);
   reg [MOD-1:0]                          counter = 0;
   reg [7:0]                              counternew = 0;
   reg [7:0]                              address = 0;

   // sin bram implemented elsewhere
   mybram_sin mybram_sin1(.clock(clock), .index(address), .y(out));

   always @(posedge clock) begin
      if (ready) begin
          counternew <= (counter + knobval) >> MOD; // how many times did we overflow?
          counter <= counter + knobval;
          // increment address (handle the stopped case too)
          address = (knobval == 0) ? 0 : address + counternew;
      end
```

```
   end
endmodule // sin

module triangle #(parameter [7:0] MOD = 8'd6
                  )(
                    input wire clock, ready,
                    input wire [10:0] knobval,
                    output wire signed [8:0] out);
   reg [MOD-1:0]                                counter = 0;
   reg [7:0]                                    counternew = 0;
   reg [7:0]                                    address = 0;

   // sin bram implemented elsewhere
   mybram_triangle mybram_triangle1(.clock(clock), .index(address), .y(out));

   always @(posedge clock) begin
      if (ready) begin
         counternew <= (counter + knobval) >> MOD; // how many times did we overflow?
         counter <= counter + knobval;
         // increment address (handle the stopped case too)
         address = (knobval == 0) ? 0 : address + counternew;
      end
   end
endmodule // triangle

module saw #(parameter [7:0] MOD = 8'd6
             )(
               input wire clock, ready,
               input wire [10:0] knobval,
               output wire signed [8:0] out);
   reg [MOD-1:0]                            counter = 0;
   reg [7:0]                                counternew = 0;
   reg [7:0]                                address = 0;

   // saw bram implemented elsewhere
   mybram_saw mybram_saw1(.clock(clock), .index(address), .y(out));

   always @(posedge clock) begin
      if (ready) begin
         counternew <= (counter + knobval) >> MOD; // how many times did we overflow?
         counter <= counter + knobval;
         // increment address (handle the stopped case too)
         address = (knobval == 0) ? 0 : address + counternew;
      end
   end
endmodule // sawtooth

// a wrapper for the 4 base waveforms
module signalgen (input wire clock, ready, pos,// if 1, drive all signals only in positive domain
                  input wire [1:0] wavesel,
                  input wire [10:0] knobval,
                  output wire signed [8:0] y);

   parameter [1:0] SQUARE=2'b00;
   parameter [1:0] SIN=2'b01;
   parameter [1:0] TRIANGLE=2'b10;
   parameter [1:0] SAW=2'b11;

   wire signed [8:0]                        value_square;
   assign value_square[8:1] = 0;
   wire signed [8:0]                        value_sin;
   wire signed [8:0]                        value_sin_uncorrected;
   wire signed [8:0]                        value_sin_corrected;
   wire signed [8:0]                        value_triangle;
   wire signed [8:0]                        value_triangle_uncorrected;
   wire signed [8:0]                        value_triangle_corrected;
   wire signed [8:0]                        value_saw;
   wire signed [8:0]                        value_saw_uncorrected;
   wire signed [8:0]                        value_saw_corrected;

   // a square gen
   square #(.IN_W(6'd11)) square_signalgen(.clock(clock),.ready(ready),.knobval(knobval),.out(value_square[0]));
   // a sin gen
   sin sin_signalgen(.clock(clock),.ready(ready),.knobval(knobval),.out(value_sin_uncorrected));
   // a triangle gen
   triangle triange_signalgen(.clock(clock),.ready(ready),.knobval(knobval),.out(value_triangle_uncorrected));
   // a sawtooth gen
   saw saw_signalgen(.clock(clock),.ready(ready),.knobval(knobval),.out(value_saw_uncorrected));

   // if POS, shift sin up and halve, to raise center from 0 to width/2
   assign value_sin_corrected = (value_sin_uncorrected >>> 1) + 9'sd128;
   assign value_sin = ((pos) ? value_sin_corrected : value_sin_uncorrected);

   // if POS, shift triangle up and halve, to raise center from 0 to width/2
   assign value_triangle_corrected = (value_triangle_uncorrected >>> 1) + 9'sd128;
   assign value_triangle = ((pos) ? value_triangle_corrected : value_triangle_uncorrected);

   // if POS, shift saw up and halve, to raise center from 0 to width/2
   assign value_saw_corrected = (value_saw_uncorrected >>> 1) + 9'sd128;
   assign value_saw = ((pos) ? value_saw_corrected : value_saw_uncorrected);

   // the wave value to use
```

```
   assign y = (wavesel[1:0] == SQUARE ? value_square :
               (wavesel[1:0] == SIN ? value_sin :
                (wavesel[1:0] == TRIANGLE ? value_triangle: value_saw)));

endmodule // signalgen
//tremolo module
module tremolo(
               input wire clock, ready,
               input wire signed [9:0] decayval,
               input wire [9:0] halfperiod,
               input wire signed [17:0] x,
               output reg signed [17:0] y);
   parameter ON = 1'b1;
   parameter OFF = 1'b0;

   reg [15:0]                          counter = 0;
   reg                                 on = 0;
   reg signed [27:0]                   pre_y = 0;

   // note: could be one assign statement, but this is more readable
   always @(posedge clock) begin
      if (ready) begin

         if (counter >= (halfperiod <<< 6)) begin
            counter <= 0;
            on <= ~on;
         end
         else counter <= counter + 1;
      end
      case(on)
        ON: y <= x;
        OFF: begin
           pre_y <= (x * decayval)<<<1;
           y <= pre_y[27:10];end
      endcase
   end
endmodule
//full wah effect
module man_auto_wah(
                    input wire clock, reset, ready, auto_on,
                    input wire [4:0]user_in,
                    input wire [1:0]width_in,
                    input wire [5:0]speed_in,
                    input wire signed [7:0]threshold_in,
                    input wire signed [17:0]x,
                    output reg signed [17:0]y);

   reg [4:0]                           wah_in = 0;
   reg [4:0]                           auto_wah_in = 0;
   reg [21:0]                          speed_actual = 0;
   reg [21:0]                          speed_counter = 0;
   reg                                 direction = 0; //0: frequency band moving higher, 1: frequency band moving lower
   reg signed [17:0]                   pos_threshold;
   reg signed [17:0]                   neg_threshold;
   wire signed [17:0]                  pre_y;

   always @(*)begin //makes positive and negative threshold
      pos_threshold = {threshold_in,10'd0};
      neg_threshold = pos_threshold * -1;
   end

   always @(posedge clock)begin
      //converts user speed to actual clock speed
      speed_actual <= {speed_in,15'd0};
      if(~auto_on)
        //if auto_on is off, use manual input
        wah_in <= user_in;
      else begin
          //auto_on is on, move frequency band automatically
          wah_in <= auto_wah_in;
          if ((x >= pos_threshold) || (x <= neg_threshold)) begin
             //if speed_counter is the specified speed value, increase/decrease the frequency band
             if ((speed_counter == speed_actual) && ~direction)begin
                auto_wah_in <= auto_wah_in + 1;
                speed_counter <= 0; end
             else if ((speed_counter == speed_actual) && direction)begin
                auto_wah_in <= auto_wah_in - 1;
                speed_counter <= 0; end
             else speed_counter <= speed_counter + 1;
             //if the frequency bands have reached their max or min, reverse direction
             if (auto_wah_in == 5'd31)
               direction <= 1'b1;
             else if (auto_wah_in == 5'b0)
               direction <=0;
          end
      end
      y <= pre_y + (pre_y >>> 1);
   end
   base_wah wah(clock, reset, ready, wah_in, width_in, x, pre_y);
endmodule

//actual wah effect. To be used inside of parent wah module that incorporates auto and manual wah
```

```
module base_wah(
                input wire clock, reset, ready,
                input wire [4:0]user_in,
                input wire [1:0]width_in,
                input wire signed [17:0]x,
                output reg signed [17:0]y);

   wire signed [17:0]                    fo1;
   wire signed [17:0]                    fo2;
   wire signed [17:0]                    fo3;
   wire signed [17:0]                    fo4;
   wire signed [17:0]                    fo5;
   wire signed [17:0]                    fo6;
   wire signed [17:0]                    fo7;
   wire signed [17:0]                    fo8;
   wire signed [17:0]                    fo9;
   wire signed [17:0]                    fo10;
   wire signed [17:0]                    fo11;

   always @(posedge clock)begin
      case(width_in)
        2'b00:begin
           //width is 0
           case(user_in)
             5'd0: y <= fo1;
             5'd1: y <= fo1;
             5'd2: y <= fo2 - fo1;
             5'd3: y <= fo2 - fo1;
             5'd4: y <= fo2 - fo1;
             5'd5: y <= fo3 - fo2;
             5'd6: y <= fo3 - fo2;
             5'd7: y <= fo3 - fo2;
             5'd8: y <= fo4 - fo3;
             5'd9: y <= fo4 - fo3;
             5'd10: y <= fo4 - fo3;
             5'd11: y <= fo4 - fo3;
             5'd12: y <= fo5 - fo4;
             5'd13: y <= fo5 - fo4;
             5'd14: y <= fo5 - fo4;
             5'd15: y <= fo6 - fo5;
             5'd16: y <= fo6 - fo5;
             5'd17: y <= fo6 - fo5;
             5'd18: y <= fo7 - fo6;
             5'd19: y <= fo7 - fo6;
             5'd20: y <= fo7 - fo6;
             5'd21: y <= fo8 - fo7;
             5'd22: y <= fo8 - fo7;
             5'd23: y <= fo8 - fo7;
             5'd24: y <= fo9 - fo8;
             5'd25: y <= fo9 - fo8;
             5'd26: y <= fo9 - fo8;
             5'd27: y <= fo9 - fo8;
             5'd28: y <= fo10 - fo9;
             5'd29: y <= fo10 - fo9;
             5'd30: y <= fo10 - fo9;
             5'd31: y <= fo11 - fo10;
           endcase
        end
        2'b01:begin
           //width is 1
           case(user_in)
             5'd0: y <= fo2;
             5'd1: y <= fo2;
             5'd2: y <= fo3 - fo1;
             5'd3: y <= fo3 - fo1;
             5'd4: y <= fo3 - fo1;
             5'd5: y <= fo3 - fo1;
             5'd6: y <= fo4 - fo2;
             5'd7: y <= fo4 - fo2;
             5'd8: y <= fo4 - fo2;
             5'd9: y <= fo5 - fo3;
             5'd10: y <= fo5 - fo3;
             5'd11: y <= fo5 - fo3;
             5'd12: y <= fo5 - fo3;
             5'd13: y <= fo6 - fo4;
             5'd14: y <= fo6 - fo4;
             5'd15: y <= fo6 - fo4;
             5'd16: y <= fo7 - fo5;
             5'd17: y <= fo7 - fo5;
             5'd18: y <= fo7 - fo5;
             5'd19: y <= fo7 - fo5;
             5'd20: y <= fo8 - fo6;
             5'd21: y <= fo8 - fo6;
             5'd22: y <= fo8 - fo6;
             5'd23: y <= fo8 - fo6;
             5'd24: y <= fo9 - fo7;
             5'd25: y <= fo9 - fo7;
             5'd26: y <= fo9 - fo7;
             5'd27: y <= fo10 - fo8;
             5'd28: y <= fo10 - fo8;
             5'd29: y <= fo10 - fo8;
             5'd30: y <= fo10 - fo8;
```

```
            5'd31: y <= fo11 - fo9;
          endcase
        end
        2'b10:begin
          //width is 2
          case(user_in)
            5'd0: y <= fo3;
            5'd1: y <= fo3;
            5'd2: y <= fo4 - fo1;
            5'd3: y <= fo4 - fo1;
            5'd4: y <= fo4 - fo1;
            5'd5: y <= fo4 - fo1;
            5'd6: y <= fo5 - fo2;
            5'd7: y <= fo5 - fo2;
            5'd8: y <= fo5 - fo2;
            5'd9: y <= fo5 - fo2;
            5'd10: y <= fo6 - fo3;
            5'd11: y <= fo6 - fo3;
            5'd12: y <= fo6 - fo3;
            5'd13: y <= fo6 - fo3;
            5'd14: y <= fo7 - fo4;
            5'd15: y <= fo7 - fo4;
            5'd16: y <= fo7 - fo4;
            5'd17: y <= fo7 - fo4;
            5'd18: y <= fo8 - fo5;
            5'd19: y <= fo8 - fo5;
            5'd20: y <= fo8 - fo5;
            5'd21: y <= fo8 - fo5;
            5'd22: y <= fo9 - fo6;
            5'd23: y <= fo9 - fo6;
            5'd24: y <= fo9 - fo6;
            5'd25: y <= fo9 - fo6;
            5'd26: y <= fo10 - fo7;
            5'd27: y <= fo10 - fo7;
            5'd28: y <= fo10 - fo7;
            5'd29: y <= fo10 - fo7;
            5'd30: y <= fo11 - fo8;
            5'd31: y <= fo11 - fo8;
          endcase
        end
        2'b11:begin
          //width is 3
          case(user_in)
            5'd0: y <= fo4;
            5'd1: y <= fo4;
            5'd2: y <= fo4;
            5'd3: y <= fo5 - fo1;
            5'd4: y <= fo5 - fo1;
            5'd5: y <= fo5 - fo1;
            5'd6: y <= fo5 - fo1;
            5'd7: y <= fo6 - fo2;
            5'd8: y <= fo6 - fo2;
            5'd9: y <= fo6 - fo2;
            5'd10: y <= fo6 - fo2;
            5'd11: y <= fo6 - fo2;
            5'd12: y <= fo7 - fo3;
            5'd13: y <= fo7 - fo3;
            5'd14: y <= fo7 - fo3;
            5'd15: y <= fo7 - fo3;
            5'd16: y <= fo8 - fo4;
            5'd17: y <= fo8 - fo4;
            5'd18: y <= fo8 - fo4;
            5'd19: y <= fo8 - fo4;
            5'd20: y <= fo8 - fo4;
            5'd21: y <= fo9 - fo5;
            5'd22: y <= fo9 - fo5;
            5'd23: y <= fo9 - fo5;
            5'd24: y <= fo9 - fo5;
            5'd25: y <= fo9 - fo5;
            5'd26: y <= fo10 - fo6;
            5'd27: y <= fo10 - fo6;
            5'd28: y <= fo10 - fo6;
            5'd29: y <= fo10 - fo6;
            5'd30: y <= fo11 - fo7;
            5'd31: y <= fo11 - fo7;
          endcase
        end
      endcase
  end

  firlow385 f1(clock, reset, ready, x, fo1);
  firlow450 f2(clock, reset, ready, x, fo2);
  firlow530 f3(clock, reset, ready, x, fo3);
  firlow620 f4(clock, reset, ready, x, fo4);
  firlow730 f5(clock, reset, ready, x, fo5);
  firlow840 f6(clock, reset, ready, x, fo6);
  firlow960 f7(clock, reset, ready, x, fo7);
  firlow1100 f8(clock, reset, ready, x, fo8);
  firlow1300 f9(clock, reset, ready, x, fo9);
  firlow1600 f10(clock, reset, ready, x, fo10);
  firlow7000 f11(clock, reset, ready, x, fo11);
endmodule
```

```
// the component of the looper that interfaces with the ZBT chips
module zbt_interface (input clock, reset, ready, we, re, loop,
                      // from labkit
                      input [1:0] channel_in_sel,
                      input [15:0] volumes,
                      input wire signed [35:0] data_in,
                      input wire signed [35:0] ram0_read_data,
                      input wire signed [35:0] ram1_read_data,
                      output wire signed [35:0] ram0_write_data,
                      output wire signed [35:0] ram1_write_data,
                      output reg signed [35:0] inter_out,
                      output wire ram0_we,
                      output wire ram1_we,
                      output reg [18:0] ram0_addr,
                      output reg [18:0] ram1_addr,
                      // debugging info
                      output wire [4:0] state_out,
                      output wire [17:0] addr_end_0_out,
                      output wire [17:0] addr_end_1_out,
                      output wire [17:0] addr_end_2_out,
                      output wire [17:0] addr_end_3_out
                      );
   // 4 base states
   parameter IDLE = 4'd0;
   parameter READ02 = 4'd1;
   parameter READ13 = 4'd2;
   parameter WRITE = 4'd3;
   parameter PRE_WRITE1 = 4'd4;
   parameter PRE_WRITE2 = 4'd5;
   parameter PRE_READ02_1 = 4'd6;
   parameter PRE_READ02_2 = 4'd7;
   parameter PRE_READ13_1 = 4'd8;
   parameter PRE_READ13_2 = 4'd9;
   parameter POST_READ = 4'd10;

   //Input Lefts and Rights
   wire signed [17:0]                       ch01L;
   wire signed [17:0]                       ch01R;
   wire signed [17:0]                       ch23L;
   wire signed [17:0]                       ch23R;
   assign ch01L = ram0_read_data[35:18];
   assign ch01R = ram0_read_data[17:0];
   assign ch23L = ram1_read_data[35:18];
   assign ch23R = ram1_read_data[17:0];

   // used to store multiply results
   reg signed [22:0]                        mult01L = 0;
   reg signed [22:0]                        mult01R = 0;
   reg signed [22:0]                        mult23L = 0;
   reg signed [22:0]                        mult23R = 0;

   // volumes
   wire signed [4:0]                        vol0;
   wire signed [4:0]                        vol1;
   wire signed [4:0]                        vol2;
   wire signed [4:0]                        vol3;
   assign vol0 = {1'b0, volumes[3:0]};
   assign vol1 = {1'b0, volumes[7:4]};
   assign vol2 = {1'b0, volumes[11:8]};
   assign vol3 = {1'b0, volumes[15:12]};

   // channel IDs
   parameter [1:0] CH0 = 2'd0;
   parameter [1:0] CH1 = 2'd1;
   parameter [1:0] CH2 = 2'd2;
   parameter [1:0] CH3 = 2'd3;
   // were we previously writing to this channel?
   reg [1:0]                                channel_write = 0;
   // were we previously in write state?
   reg                                      old_we = 0;
   // were we previously in read state?
   reg                                      old_re = 0;

   // last bit of state is used for downsampling
   reg [4:0]                                state = {IDLE, 1'b0};
   reg                                      master_we = 0;
   assign ram0_we = (master_we & (channel_in_sel[1] == 1'b0));
   assign ram1_we = (master_we & (channel_in_sel[1] == 1'b1));
   // we'll always write the current data
   assign ram0_write_data = data_in;
   assign ram1_write_data = data_in;

   // modular ("abstract") addresses -- an easy way to loop. These are juggled by the FSM below
   reg [17:0]                               addr_end_mod [3:0];
   reg [17:0]                               addr_play_mod [3:0];
   // for debugging
   assign state_out = state;
   assign addr_end_0_out = addr_end_mod[CH0];
   assign addr_end_1_out = addr_end_mod[CH1];
   assign addr_end_2_out = addr_end_mod[CH2];
   assign addr_end_3_out = addr_end_mod[CH3];
```

```
   wire                            all_at_end;
   assign all_at_end = ((addr_play_mod[CH0] == addr_end_mod[CH0]) & (addr_play_mod[CH1] == addr_end_mod[CH1]) &
                        (addr_play_mod[CH2] == addr_end_mod[CH2]) & (addr_play_mod[CH3] == addr_end_mod[CH3]));

   // the FSM
   always @(posedge clock) begin
      if (reset) begin
         inter_out <= 0;
         master_we <= 0;
         state <= 0;
         // reset all adresses
         ram0_addr <= 0;
         ram1_addr <= 0;
         addr_end_mod[CH0] <= 18'd1;
         addr_end_mod[CH1] <= 18'd1;
         addr_end_mod[CH2] <= 18'd1;
         addr_end_mod[CH3] <= 18'd1;
         addr_play_mod[CH0] <= 18'd1;
         addr_play_mod[CH1] <= 18'd1;
         addr_play_mod[CH2] <= 18'd1;
         addr_play_mod[CH3] <= 18'd1;
      end else begin
         //Write/Read Logic
         case(state[4:1])
           IDLE: begin
              // then, on each ready pulse
              if (ready) begin
                 state[0] <= ~state[0]; // 2x downsampling
                 if (state[0]) begin // should read/write sample
                    old_we <= we;
                    old_re <= re;

                    if ((~re) & (~we)) begin // if we just switched from write to something else
                       // backup the current address for looping
                       addr_play_mod[0] <= 18'd1;
                       addr_play_mod[1] <= 18'd1;
                       addr_play_mod[2] <= 18'd1;
                       addr_play_mod[3] <= 18'd1;
                       inter_out <= 0;
                    end

                    else if (we & ~old_we) begin
                       addr_end_mod[channel_in_sel] <= 18'd1;
                    end

                    else if (we) begin // write new input sample -- alter end, and current if necessary
                       state[4:1] <= PRE_WRITE1;
                       // increment the appropriate channel's end address
                       if (channel_write != channel_in_sel) begin // if the channel_in_sel just changed, and we're writing to
the new channel:
                          // overwrite the channel_write value with the new channel
                          channel_write <= channel_in_sel;
                       end
                       // update the actual chip addresses
                       else if (addr_play_mod[channel_in_sel] != 18'd0) begin
                          addr_play_mod[channel_in_sel] <= addr_play_mod[channel_in_sel] + 1'b1;
                          addr_end_mod[channel_in_sel] <= addr_end_mod[channel_in_sel] + 1'b1;
                          case(channel_write)
                            //Now using all of channel_in_sel to split ZBT SRAMs in half: Devon
                            CH0: ram0_addr <= {1'b0,addr_play_mod[0] + 1'b1};
                            CH1: ram0_addr <= {1'b1,addr_play_mod[1] + 1'b1};
                            CH2: ram1_addr <= {1'b0,addr_play_mod[2] + 1'b1};
                            CH3: ram1_addr <= {1'b1,addr_play_mod[3] + 1'b1};
                            default: channel_write <= CH1;
                          endcase // case (channel_in_sel[1])
                          // drive master_we high
                          master_we <= 1'b1;
                       end
                       // the data is assigned to the zbt input ports, so no need to include that here

                    end // if (we)

                    else if (re) begin // drive sample to output
                       state[4:1] <= PRE_READ02_1;
                       // channel 0
                       if (addr_play_mod[0] != addr_end_mod[0]) begin // default value
                          // advance the current addresses for channel 0
                          addr_play_mod[0] <= addr_play_mod[0] + 1'b1;
                          // change the ram0 address to the new channel 0 address
                          ram0_addr <= {1'b0, addr_play_mod[0] + 1'b1};
                       end // !if(addr_play...

                       // channel 2
                       if (addr_play_mod[2] != addr_end_mod[2]) begin // default value
                          // advance the current addresses for channel 0
                          addr_play_mod[2] <= addr_play_mod[2] + 1'b1;
                          // change the ram0 address to the new channel 0 address
                          ram1_addr <= {1'b0, addr_play_mod[2] + 1'b1};
                       end // !if(addr_play...

                    end // if (re)
                 end // if (state[0])
```

```
   end // if (ready)
end // case: IDLE

PRE_READ02_1:
  state[4:1] <= PRE_READ02_2;

PRE_READ02_2:
  state[4:1] <= READ02;

READ02: begin // on next cycle, output data from channels 1 and 3
   state[4:1] <= PRE_READ13_1;
   // channel 0
   if (addr_play_mod[0] != addr_end_mod[0]) begin // default value
      // drive channel 0 output
      mult01L <= ch01L * vol0;
      mult01R <= ch01R * vol0;
   end
   else begin // have reached end normally
      mult01L <= 0;
      mult01R <= 0;
      if (loop) begin
         if (all_at_end) begin // if this is the 4th increment, and thus the longest channel
            // reset all current addresses to backup value
            addr_play_mod[0] <= 18'd1;
            addr_play_mod[1] <= 18'd1;
         end
      end // if (loop)
   end // else: !if(~( (addr_current_mod[CH0] == addr_end_mod[CH0]) ||...

   // channel 2
   if (addr_play_mod[2] != addr_end_mod[2]) begin // default value
      // drive channel 2 output
      mult23L <= ch23L*vol2;
      mult23R <= ch23R*vol2;
   end
   else begin // have reached end normally
      mult23L <= 0;
      mult23R <= 0;
      if (loop) begin
         if (all_at_end) begin // if this is the 4th increment, and thus the longest channel
            // reset all current addresses to backup value
            addr_play_mod[2] <= 18'd1;
            addr_play_mod[3] <= 18'd1;
         end
      end // if (loop)
   end // else: !if(~( (addr_current_mod[CH0] == addr_end_mod[CH0]) ||...

   // now advance the chip adresses for channel 1/3
   // channel 1
   if (addr_play_mod[1] != addr_end_mod[1]) begin // default value
      // advance the current addresses for channel 0
      addr_play_mod[1] <= addr_play_mod[1] + 1'b1;
      // change the ram0 address to the new channel 0 address
      ram0_addr <= {1'b1, addr_play_mod[1] + 1'b1};
   end // !if(addr_play...

   // channel 3
   if (addr_play_mod[3] != addr_end_mod[3]) begin // default value
      // advance the current addresses for channel 0
      addr_play_mod[3] <= addr_play_mod[3] + 1'b1;
      // change the ram0 address to the new channel 0 address
      ram1_addr <= {1'b1, addr_play_mod[3] + 1'b1};
   end // !if(addr_play...

end // case: READ02

PRE_READ13_1: begin
   state[4:1] <= PRE_READ13_2;
end
PRE_READ13_2: state[4:1] <= READ13;

READ13: begin // on next cycle, output data from channels 2 and 4
   state[4:1] <= POST_READ;
   // drive the values fetched in the previous cycle to outputs

   // channel 1
   if (addr_play_mod[1] != addr_end_mod[1]) begin // default value
      // drive channel 1 output
      mult01L <= ch01L*vol1 + mult01L;
      mult01R <= ch01R*vol1 + mult01R;
   end
   else begin // have reached end normally
      if (loop) begin
         if (all_at_end) begin // if this is the 4th increment, and thus the longest channel
            // reset all current addresses to backup value
            addr_play_mod[0] <= 18'd1;
            addr_play_mod[1] <= 18'd1;
         end
      end // if (loop)
   end // else: !if(~( (addr_current_mod[CH0] == addr_end_mod[CH0]) ||...

   // channel 3
```

```
            if (addr_play_mod[3] != addr_end_mod[3]) begin // default value
               // drive channel 3 output
               mult23L <= ch23L*vol3 + mult23L;
               mult23R <= ch23R*vol3 + mult23R;
            end
            else begin // have reached end normally
               if (loop) begin
                  if (all_at_end) begin // if this is the 4th increment, and thus the longest channel
                     // reset all current addresses to backup value
                     addr_play_mod[2] <= 18'd1;
                     addr_play_mod[3] <= 18'd1;
                  end
               end // if (loop)
            end // else: !if(~( (addr_current_mod[CH0] == addr_end_mod[CH0]) ||...

         end // case: READ13

         POST_READ: begin
            state <= IDLE;
            inter_out <= {mult01L[22:5] + mult23L[22:5], mult01R[22:5] + mult23R[22:5]};
         end

         PRE_WRITE1:
           state[4:1] <= PRE_WRITE2;

         PRE_WRITE2: state[4:1] <= WRITE;

         WRITE: begin
            state[4:1] <= IDLE;
            // here, the write has completed
            // drive master_we low
            master_we <= 1'b0;
         end
         default: state[4:1] <= IDLE;
       endcase // case (state[4:1])
     end // else: !if(reset)
  end // always @ (posedge clock)
endmodule

// The main component
module looper (input clock, reset, ready, loop, enter,
               input [1:0] action,
               input [1:0] channel_in_sel,
               input [15:0] volumes,
               input [3:0] enable,
               input wire signed [35:0] data_in,
               // from labkit, fed to zbt_interface
               input wire [35:0] ram0_read_data,
               input wire [35:0] ram1_read_data,
               output wire [35:0] ram0_write_data,
               output wire [35:0] ram1_write_data,
               output wire ram0_we,
               output wire ram1_we,
               output wire [18:0] ram0_addr,
               output wire [18:0] ram1_addr,
               output reg signed [35:0] y,
               // debugging info
               output wire [4:0] state_out,
               output wire [17:0] addr_end_0_out,
               output wire [17:0] addr_end_1_out,
               output wire [17:0] addr_end_2_out,
               output wire [17:0] addr_end_3_out
               );

  parameter OFF=2'b00;
  parameter PLAY=2'b01;
  parameter REC=2'b10;
  // rec for 11 as well
  reg                              we, re;
  wire signed [35:0]               inter_out;
  wire signed [35:0]               ch01;
  wire signed [35:0]               ch23;
  reg [1:0]                        state = OFF;
  wire [15:0]                      volumes_muxed;
  assign volumes_muxed = {((enable[3]) ? volumes[15:12] : 4'd0),((enable[2]) ? volumes[11:8] : 4'd0),
                          ((enable[1]) ? volumes[7:4] : 4'd0),((enable[0]) ?volumes[3:0] : 4'd0)};

  always @(posedge clock) begin
     we <= state[1];
     re <= state[0]; // low for 2'b11, for convenience
     state <= action;
     y <= data_in + inter_out;
  end

  zbt_interface zbt_interface1 (.clock(clock), .reset(reset), .ready(ready), .we(we), .re(re), .loop(loop),
                                .channel_in_sel(channel_in_sel),
                                .data_in(data_in),
                                .volumes(volumes_muxed),
                                .ram0_read_data(ram0_read_data),
                                .ram1_read_data(ram1_read_data),
                                .ram0_write_data(ram0_write_data),
                                .ram1_write_data(ram1_write_data),
```

```
                                    .ram0_we(ram0_we),
                                    .ram1_we(ram1_we),
                                    .ram0_addr(ram0_addr),
                                    .ram1_addr(ram1_addr),
                                    .inter_out(inter_out),
                                    // for debugging
                                    .state_out(state_out),
                                    .addr_end_0_out(addr_end_0_out),
                                    .addr_end_1_out(addr_end_1_out),
                                    .addr_end_2_out(addr_end_2_out),
                                    .addr_end_3_out(addr_end_3_out)
                                    );
endmodule

// the looper control module
module looper_control (input wire clock, reset, ch0, ch1, ch2, ch3, enter,
                       output reg [1:0] chsel,
                       output reg [1:0] action,
                       output reg [3:0] ch_enable);
   // channels
   parameter CH0 = 2'd0;
   parameter CH1 = 2'd1;
   parameter CH2 = 2'd2;
   parameter CH3 = 2'd3;
   // looper actions
   parameter LOOPER_OFF = 2'b00;
   parameter LOOPER_PLAY = 2'b01;
   parameter LOOPER_REC = 2'b10;
   // internal states
   parameter IDLE=2'b00;
   parameter ENTER=2'b01;
   parameter REC=2'b10;
   parameter BOTH=2'b11;

   reg [2:0]                              state = {1'b1, IDLE};

   initial begin
      chsel = CH0;
      action = LOOPER_OFF;
      ch_enable = 4'b1111;
   end

   wire all;
   wire some;
   assign some = (ch0 || ch1 || ch2 || ch3);
   // note: this logic assumes "depressed" means driven to 1, and "released" means driven to 0
   always @(posedge clock) begin
      case(state[1:0])
        IDLE: begin
           if (some) begin // a channel button was depressed
              chsel <= ((ch0) ? CH0 : ((ch1) ? CH1 : ((ch2) ? CH2 : CH3)));
              action <= LOOPER_REC;
              state[1:0] <= REC;
           end else if (enter) begin // enter was depressed
              // action <= LOOPER_OFF;
              state[1:0] <= ENTER;
           end
        end
        ENTER: begin
           if (some) begin // a channel button was depressed
              state[1:0] <= BOTH;
              chsel <= ((ch0) ? CH0 : ((ch1) ? CH1 : ((ch2) ? CH2 : CH3))); // store the channel for use in BOTH
           end
           else if (~enter) begin // enter was released
              state <= {~state[2], IDLE};
              action <= (state[2] ? LOOPER_PLAY : LOOPER_OFF); // toggle play/off
           end
        end
        BOTH: begin
           case(chsel)
             CH0: begin
                if (~ch0 & ~enter) begin // the previously selected channel button was released
                   ch_enable[0] <= ~ch_enable[0];
                   state <= {state[2], IDLE}; // state[2] so that it won't change anything when enter is next pressed
                   //action <= LOOPER_OFF;
                end
             end
             CH1: begin
                if (~ch1 & ~enter) begin // the previously selected channel button was released
                   ch_enable[1] <= ~ch_enable[1];
                   state <= {state[2], IDLE}; // state[2] so that it won't change anything when enter is next pressed
                   //action <= LOOPER_OFF;
                end
             end
             CH2: begin
                if (~ch2 & ~enter) begin // the previously selected channel button was released
                   ch_enable[2] <= ~ch_enable[2];
                   state <= {state[2], IDLE}; // state[2] so that it won't change anything when enter is next pressed
                   //action <= LOOPER_OFF;
                end
             end
             CH3: begin
```

```
                if (~ch3 & ~enter) begin // the previously selected channel button was released
                   ch_enable[3] <= ~ch_enable[3];
                   state <= {state[2], IDLE}; // state[2] so that it won't change anything when enter is next pressed
                   //action <= LOOPER_OFF;
                end
             end
             default: chsel <= CH0;
           endcase // case (chsel)
        end // case: BOTH
        REC: begin
           case(chsel)
             CH0: begin
                if (~ch0) begin // the previously selected channel button was released
                   state <= {1'b1, IDLE}; // reset toggle to ensure PLAY will happen the next time enter is pressed
                   action <= LOOPER_OFF;
                end
             end
             CH1: begin
                if (~ch1) begin // the previously selected channel button was released
                   state <= {1'b1, IDLE}; // reset toggle to ensure PLAY will happen the next time enter is pressed
                   action <= LOOPER_OFF;
                end
             end
             CH2: begin
                if (~ch2) begin // the previously selected channel button was released
                   state <= {1'b1, IDLE}; // reset toggle to ensure PLAY will happen the next time enter is pressed
                   action <= LOOPER_OFF;
                end
             end
             CH3: begin
                if (~ch3) begin // the previously selected channel button was released
                   state <= {1'b1, IDLE}; // reset toggle to ensure PLAY will happen the next time enter is pressed
                   action <= LOOPER_OFF;
                end
             end
             default: chsel <= CH0;
           endcase // case (chsel)
        end
        default: state[1:0] <= IDLE;
      endcase
   end // always @ (posedge clock)
endmodule

//
// File:   zbt_6111.v
// Date:   27-Nov-05
// Author: I. Chuang <ichuang@mit.edu>
//
// Simple ZBT driver for the MIT 6.111 labkit, which does not hide the
// pipeline delays of the ZBT from the user.  The ZBT memories have
// two cycle latencies on read and write, and also need extra-long data hold
// times around the clock positive edge to work reliably.
//

////////////////////////////////////////////////////////////////////////////////
// Ike's simple ZBT RAM driver for the MIT 6.111 labkit
//
// Data for writes can be presented and clocked in immediately; the actual
// writing to RAM will happen two cycles later.
//
// Read requests are processed immediately, but the read data is not available
// until two cycles after the intial request.
//
// A clock enable signal is provided; it enables the RAM clock when high.

module zbt_6111(clk, cen, we, addr, write_data, read_data,
                  ram_clk, ram_we_b, ram_address, ram_data, ram_cen_b);

   input clk;                    // system clock
   input cen;                    // clock enable for gating ZBT cycles
   input we;                     // write enable (active HIGH)
   input [18:0] addr;            // memory address
   input [35:0] write_data;      // data to write
   output [35:0] read_data;      // data read from memory
   output         ram_clk;       // physical line to ram clock
   output         ram_we_b;      // physical line to ram we_b
   output [18:0] ram_address;    // physical line to ram address
   inout [35:0]  ram_data;       // physical line to ram data
   output         ram_cen_b;     // physical line to ram clock enable

   // clock enable (should be synchronous and one cycle high at a time)
   wire   ram_cen_b = ~cen;

   // create delayed ram_we signal: note the delay is by two cycles!
   // ie we present the data to be written two cycles after we is raised
   // this means the bus is tri-stated two cycles after we is raised.

   reg [1:0]   we_delay;

   always @(posedge clk)
     we_delay <= cen ? {we_delay[0],we} : we_delay;
```

```
   // create two-stage pipeline for write data

   reg [35:0]  write_data_old1;
   reg [35:0]  write_data_old2;
   always @(posedge clk)
     if (cen)
       {write_data_old2, write_data_old1} <= {write_data_old1, write_data};

   // wire to ZBT RAM signals

   assign      ram_we_b = ~we;
   assign      ram_clk = 1'b0;  // gph 2011-Nov-10
                                // set to zero as place holder

//   assign      ram_clk = ~clk;     // RAM is not happy with our data hold
                                   // times if its clk edges equal FPGA's
                                   // so we clock it on the falling edges
                                   // and thus let data stabilize longer
   assign      ram_address = addr;

   assign      ram_data = we_delay[1] ? write_data_old2 : {36{1'bZ}};
   assign      read_data = ram_data;

endmodule // zbt_6111

////////////////////////////////////////////////////////////////////////////////
//
// Verilog equivalent to a BRAM, tools will infer the right thing!
// number of locations = 1<<LOGSIZE, width in bits = WIDTH.
// default is a 16K x 1 memory.
//
////////////////////////////////////////////////////////////////////////////////

module mybram #(parameter LOGSIZE=14, WIDTH=1)
               (input wire [LOGSIZE-1:0] addr,
                input wire clk,
                input wire [WIDTH-1:0] din,
                output reg [WIDTH-1:0] dout,
                input wire we);
   // let the tools infer the right number of BRAMs
   (* ram_style = "block" *)
   reg [WIDTH-1:0] mem[(1<<LOGSIZE)-1:0];
   always @(posedge clk) begin
     if (we) mem[addr] <= din;
     dout <= mem[addr];
   end
endmodule
```

Appendix C: Testbench Code

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   03:07:25 11/22/2011
// Design Name:   chorus
// Module Name:   /afs/athena.mit.edu/user/d/s/dsherry/6.111/final/verilog//chorus_test.v
// Project Name:  finalproj
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: chorus
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module chorus_test;

   parameter [1:0] SQUARE=2'b00;
   parameter [1:0] SIN=2'b01;
   parameter [1:0] TRIANGLE=2'b10;
   parameter [1:0] SAW=2'b11;

   // Inputs
   reg clock;
   reg ready;
   reg reset;
   reg [1:0] wavesel;
   reg [9:0] delay;
   reg [7:0] depth;
   reg [8:0] rate;
   reg [9:0] decay;
   //reg [17:0] x;

   // Outputs
   wire [17:0] y;

   // input signal generation params
   reg [1:0]   wavesel_x;
   reg [10:0]   rate_x;
   wire signed [8:0] x_raw;
   wire signed [17:0] x;

   // Instantiate the Unit Under Test (UUT)
   chorus #(.LOGSIZE(4'd3)) // small, for testing
   uut (.clock(clock),
        .reset(reset),
        .ready(ready),
        .wavesel(wavesel),
        .delay(delay),
        .depth(depth),
        .rate(rate),
        .decay(decay),
        .x(x),
        .y(y));

   // the generator for x (input waveform)
   signalgen signalgen_reverb(.clock(clock), .ready(ready), .pos(1'b1), .wavesel(wavesel_x), .knobval(rate_x), .y(x_raw));

   // handle square wave amplitude (==1)
   assign x = ((wavesel_x == SQUARE) ? {1'b0, x_raw[0], 16'd0} : x_raw);

   initial begin // 27mhz
      // let's say 1 picosecond timestep is now 19 picoseconds
      forever #1 clock <= ~clock;
   end

   always begin // 48khz
      ready=1;
      #2;
      ready=0;
      #1124;
   end

   initial begin
      // Initialize Inputs
      wavesel_x = 2'd1;
      rate_x = 10'd1;

      clock = 1;
```

```
      reset = 1;
      ready = 0;
      // mess with these parameters for testing
      wavesel = 2'd1;
      delay = 10'd800;
      decay = 10'd512; // scale by half (at most)
      depth = 8'd255; // max value for now
      rate = 11'd5;
      //decay_in = 10'd1023;
      //x = 0;

      // Wait 100 ns for global reset to finish
      #100;
      //reset = 0;

      // Note: to convert from real time to internal time, multiply by 19.
      // To calculate real times from internal times, divide.
      // Add stimulus here
      #18000;
      reset = 0;
      #1000;
      //x = 0;
      //x = 18'sd31071; // high impulse
      #1500;
      //x = 0;
      // memory should be 0 now.


   end
endmodule

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   12:40:02 11/22/2011
// Design Name:   moogerfooger
// Module Name:   /afs/athena.mit.edu/user/d/s/dsherry/6.111/final/verilog//moogerfooger_test.v
// Project Name:  finalproj
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: moogerfooger
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module distortion_test;

   // Inputs
   reg clock;
   reg ready;
   reg signed [7:0]threshold_in;
         reg signed [6:0]overdrive_scale;
         reg signed [7:0]gain_in;
   // reg [17:0] x;

   // Outputs
   wire signed [17:0] y;

   // input signal generation params
   reg [1:0]   wavesel_x;
   reg [10:0]   rate_x;
   wire signed [8:0] x_raw;
   wire signed [17:0] x;

   // Instantiate the Unit Under Test (UUT)
   distortion dist_mod(clock,ready,threshold_in,overdrive_scale,gain_in,x,y);


   // the generator for x (input waveform)
   signalgen signalgen_dist(.clock(clock), .ready(ready), .pos(1'b0), .wavesel(wavesel_x), .knobval(rate_x), .y(x_raw));

   // handle square wave amplitude (==1)
   // assign x = ((wavesel_x == SQUARE) ? {1'b0, x_raw[0], 16'd0} : x_raw);
   assign x = x_raw <<< 9;

   initial begin // 27mhz
      // let's say 1 picosecond timestep is now 19 picoseconds
      forever #1 clock <= ~clock;
   end

   always begin // 48khz
      ready=1;
```

```
      #2;
      ready=0;
      #1124;
   end

   initial begin
      // Initialize Inputs
      clock = 0;
      ready = 0;
      threshold_in = 8'sd10;
      overdrive_scale = 7'sd0;
          gain_in = 8'sb00011111;

          wavesel_x = 2'd1;
      rate_x = 10'd700;

      //x = 0;

      // Wait 100 ns for global reset to finish
      #100;
      #100000;
      overdrive_scale = 7'sd10;
      #100000;
      overdrive_scale = 7'sd20;
      #100000;

      // Add stimulus here

   end

endmodule // moogerfooger_test
// A fake zbt module used for testing -- spits out "random" values for data_out when requested
module data_generator (input clock, reset, send_next_channel,
                  output reg signed [35:0] data_out);

   reg signed [35:0]                        buffer = {18'd1, 18'd1};

   always @(posedge clock) begin
      if (reset) begin
          buffer <= {18'sd1, 18'sd1};
      end
      else if (send_next_channel) begin
          buffer <= {buffer[35:18] + 18'sd1, buffer[17:0] + 18'sd1};
          data_out <= buffer;
      end
   end
endmodule // data_generator

// A simple bram wrapper.
module fake_zbt (input clock, reset, ready, fake_we, fake_address, fake_write_data,
                  output reg signed [35:0] fake_read_data);

   parameter [1:0] IDLE = 2'b00;
   parameter [1:0] WRITE = 2'b10;
   parameter [1:0] READ = 2'b01;

   wire          fake_we;
   assign fake_we = ((state == WRITE) ? 1'b1 : 0);

   reg signed [35:0]     fake_bram_write_data = 0;
   wire signed [35:0]    fake_bram_read_data;

   // make a (2**3) x 36 memory
   // why 2**3? easier to simulate than 2**19. use only high-order bits for address.
   mybram #(.LOGSIZE(3),.WIDTH(36))
   bram_fake(.addr(fake_address[18:16]),.clk(clock),.we(fake_we),.din(fake_bram_write_data),.dout(fake_bram_read_data));

   always @(posedge clock) begin
      case(state)
        IDLE: begin
           if (ready) begin
              if (we) begin // WRITE
                  state <= WRITE;
                  fake_bram_write_data <= fake_write_data;
              end else begin // READ
                  state <= READ;
              end
           end
        end
        READ: begin
           state <= IDLE;
           fake_read_data <= fake_bram_read_data;
        end
        WRITE: begin
           state <= IDLE;
        end
        default: state <= IDLE;
      endcase
   end

endmodule`timescale 1ns / 1ps
```

```
////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   22:39:03 11/21/2011
// Design Name:   longdel
// Module Name:   /afs/athena.mit.edu/user/d/s/dsherry/6.111/final/verilog//longdel_test.v
// Project Name:  finalproj
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: longdel
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module longdel_test;

   // Inputs
   reg clock;
   reg reset;
   reg ready;
   reg [9:0] delay_in;
   reg [9:0] decay_in;
   reg [17:0] x;

   // Outputs
   wire [17:0] y;

   // Instantiate the Unit Under Test (UUT)
   // longdel  #(.LOGSIZE(5'd3))
   longdel uut (.clock(clock),
         .reset(reset),
         .ready(ready),
         .delay_in(delay_in),
         .decay_in(decay_in),
         .x(x),
         .y(y));

   initial begin // 27mhz
      // let's say 1 picosecond timestep is now 19 picoseconds
      forever #1 clock <= ~clock;
   end

   always begin // 48khz
      ready=1;
      #2;
      ready=0;
      #1124;
   end

   initial begin
      // Initialize Inputs
      clock = 1;
      reset = 1;
      ready = 0;
      delay_in = 10'd2;
      decay_in = 10'd512; // scale by half
      //decay_in = 10'd1023;
      x = 0;

      // Wait 100 ns for global reset to finish
      #100;
      //reset = 0;

      // Note: to convert from real time to internal time, multiply by 19.
      // To calculate real times from internal times, divide.
      // Add stimulus here
      #18000;
      reset = 0;
      #1000;
      //x = 0;
      x = 18'sd31071; // high impulse
      #1500;
      x = 0;
      // memory should be 0 now.
   end
endmodule
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   23:03:58 12/05/2011
// Design Name:   looper_control
```

```
// Module Name:   /afs/athena.mit.edu/user/d/s/dsherry/6.111/final/verilog//looper_control_test.v
// Project Name:  finalproj
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: looper_control
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module looper_control_test;

   // Inputs
   reg clock;
   reg reset;
   reg enter;
   reg ch0;
   reg ch1;
   reg ch2;
   reg ch3;

   // Outputs
   wire [1:0] chsel;
   wire [1:0] action;
   wire [3:0] ch_enable;


   // Instantiate the Unit Under Test (UUT)
   looper_control uut (
                       .clock(clock),
                       .reset(reset),
                       .enter(enter),
                       .ch0(ch0),
                       .ch1(ch1),
                       .ch2(ch2),
                       .ch3(ch3),
                       .chsel(chsel),
                       .action(action),
                       .ch_enable(ch_enable)
                       );
   initial begin
      forever #1 clock <= ~clock;
   end

   initial begin
      // Initialize Inputs
      clock = 1'b1;
      reset = 0;
      enter = 0;
      ch0 = 0;
      ch1 = 0;
      ch2 = 0;
      ch3 = 0;
      // Wait 100 ns for global reset to finish
      #100;
      // First, assert/deassert enter a few times to test the playback toggle
      enter = 1'b1;
      #4;
      enter = 0;
      #4;
      enter = 1'b1;
      #4;
      enter = 0;
      #4;
      enter = 1'b1;
      #4;
      enter = 0;
      #4;
      enter = 1'b1;
      #4;
      enter = 0;
      #4;
      // Next, try pressing a channel button
      ch2 = 1'b1;
      #4;
      ch2 = 0;
      #4;
      ch1 = 1'b1;
      #4;
      ch1 = 0;
      #4;
      ch3 = 1'b1;
      #4;
      ch3 = 0;
      #4;
      ch0 = 1'b1;
```

```
      #4;
      ch0 = 0;
      #4;
      // Finally, try both to toggle enable/disable for the channel
      enter = 1'b1;
      #4;
      ch2 = 1'b1;
      #4;
      ch2 = 0;
      #4;
      enter = 0;
      #4;
      enter = 1'b1;
      #4;
      ch1 = 1'b1;
      #4;
      ch1 = 0;
      #4;
      enter = 0;
      #4;
      enter = 1'b1;
      #4;
      ch3 = 1'b1;
      #4;
      enter = 0;
      #4;
      ch3 = 0;
      #4;

   end

endmodule

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   02:08:05 12/09/2011
// Design Name:   looper
// Module Name:   /afs/athena.mit.edu/user/d/s/dsherry/6.111/final/verilog//looper_test.v
// Project Name:  final
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: looper
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module looper_test;

   // Inputs
   reg clock;
   reg reset;
   reg ready;
   reg loop;
   reg enter;
   reg [1:0] action;
   reg [1:0] channel_in_sel;
   reg [15:0] volumes;
   reg [3:0]  enable;
   reg [35:0] data_in;
   reg [35:0] ram0_read_data;
   reg [35:0] ram1_read_data;

   // Outputs
   wire [35:0] ram0_write_data;
   wire [35:0] ram1_write_data;
   wire        ram0_we;
   wire        ram1_we;
   wire [18:0] ram0_addr;
   wire [18:0] ram1_addr;
   wire [35:0] y;
   wire [3:0]  state_out;
   wire [17:0] addr_current_0_out;
   wire [17:0] addr_current_1_out;
   wire [17:0] addr_current_2_out;
   wire [17:0] addr_current_3_out;
   wire [17:0] addr_current_bak_0_out;
   wire [17:0] addr_current_bak_1_out;
   wire [17:0] addr_current_bak_2_out;
   wire [17:0] addr_current_bak_3_out;
   wire [17:0] addr_end_0_out;
   wire [17:0] addr_end_1_out;
```

```
   wire [17:0] addr_end_2_out;
   wire [17:0] addr_end_3_out;

   // Instantiate the Unit Under Test (UUT)
   looper uut (
               .clock(clock),
               .reset(reset),
               .ready(ready),
               .loop(loop),
               .enter(enter),
               .action(action),
               .channel_in_sel(channel_in_sel),
               .volumes(volumes),
               .enable(enable),
               .data_in(data_in),
               .ram0_read_data(ram0_read_data),
               .ram1_read_data(ram1_read_data),
               .ram0_write_data(ram0_write_data),
               .ram1_write_data(ram1_write_data),
               .ram0_we(ram0_we),
               .ram1_we(ram1_we),
               .ram0_addr(ram0_addr),
               .ram1_addr(ram1_addr),
               .y(y),
               .state_out(state_out),
               .addr_current_0_out(addr_current_0_out),
               .addr_current_1_out(addr_current_1_out),
               .addr_current_2_out(addr_current_2_out),
               .addr_current_3_out(addr_current_3_out),
               .addr_current_bak_0_out(addr_current_bak_0_out),
               .addr_current_bak_1_out(addr_current_bak_1_out),
               .addr_current_bak_2_out(addr_current_bak_2_out),
               .addr_current_bak_3_out(addr_current_bak_3_out),
               .addr_end_0_out(addr_end_0_out),
               .addr_end_1_out(addr_end_1_out),
               .addr_end_2_out(addr_end_2_out),
               .addr_end_3_out(addr_end_3_out)
              );


   initial begin
      forever #1 clock <= ~clock;
   end

   always begin // once every 20 clock cycles
      ready=1;
      #2;
      ready=0;
      #18;
   end

   initial begin
      // Initialize Inputs
      clock = 1'b1;
      reset = 0;
      ready = 0;
      loop = 0;
      enter = 0;
      action = 0;
      channel_in_sel = 0;
      volumes = 0;
      enable = 0;
      data_in = 0;
      ram0_read_data = 0;
      ram1_read_data = 0;

      // Wait 100 ns for global reset to finish
      #100;

      // Add stimulus here

   end

endmodule

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   23:05:29 12/05/2011
// Design Name:   mixer
// Module Name:   /afs/athena.mit.edu/user/d/s/dsherry/6.111/final/verilog//mixer_test.v
// Project Name:  finalproj
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: mixer
//
// Dependencies:
```

```
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module mixer_test;

   // Inputs
   reg clock;
   reg reset;
   reg ready;
   reg we;
   reg re;
   reg zbt_go;
   reg [15:0] volumes;
   reg [35:0] master_in;
   reg [35:0] ch01in;
   reg [35:0] ch23in;

   // Outputs
   wire [35:0] y;

   // Instantiate the Unit Under Test (UUT)
   mixer uut (
              .clock(clock),
              .reset(reset),
              .ready(ready),
              .we(we),
              .re(re),
              .zbt_go(zbt_go),
              .volumes(volumes),
              .master_in(master_in),
              .ch01in(ch01in),
              .ch23in(ch23in),
              .y(y)
              );

   initial begin
      forever #1 clock <= ~clock;
   end

   always begin // once every 20 clock cycles
      ready=1;
      #2;
      ready=0;
      #18;
   end

   initial begin
      // Initialize Inputs
      clock = 1'b1;
      reset = 0;
      ready = 0;
      we = 0;
      re = 0;
      zbt_go = 0;
      volumes = 0;
      master_in = 0;
      ch01in = 0;
      ch23in = 0;

      // Wait 100 ns for global reset to finish
      #100;

      // Add stimulus here

   end

endmodule

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   12:40:02 11/22/2011
// Design Name:   moogerfooger
// Module Name:   /afs/athena.mit.edu/user/d/s/dsherry/6.111/final/verilog//moogerfooger_test.v
// Project Name:  finalproj
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: moogerfooger
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
```

```
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module moogerfooger_test;

   parameter [1:0] SQUARE=2'b00;
   parameter [1:0] SIN=2'b01;
   parameter [1:0] TRIANGLE=2'b10;
   parameter [1:0] SAW=2'b11;

   // Inputs
   reg clock;
   reg ready;
   reg [1:0] wavesel;
   reg [9:0] period;
   // reg [17:0] x;

   // Outputs
   wire [17:0] y;

   // input signal generation params
   reg [1:0]   wavesel_x;
   reg [10:0]   rate_x;
   wire signed [8:0] x_raw;
   wire signed [17:0] x;

   // Instantiate the Unit Under Test (UUT)
   moogerfooger uut (.clock(clock),
                     .ready(ready),
                     .wavesel(wavesel),
                     .period(period),
                     .x(x),
                     .y(y));

   // the generator for x (input waveform)
   signalgen signalgen_moogerfoogertest(.clock(clock), .ready(ready), .pos(1'b1), .wavesel(wavesel_x), .knobval(rate_x),
.y(x_raw));

   // handle square wave amplitude (==1)
   // assign x = ((wavesel_x == SQUARE) ? {1'b0, x_raw[0], 16'd0} : x_raw);
   assign x = ((wavesel_x == SQUARE) ? {2'b0, x_raw[0], 15'd0} : x_raw);

   initial begin // 27mhz
      // let's say 1 picosecond timestep is now 19 picoseconds
      forever #1 clock <= ~clock;
   end

   always begin // 48khz
      ready=1;
      #2;
      ready=0;
      #1124;
   end

   initial begin
      // Initialize Inputs
      clock = 0;
      ready = 0;
      wavesel = 2'd2;
      period = 10'd200;

      wavesel_x = 2'd0;
      rate_x = 10'd371;

      //x = 0;

      // Wait 100 ns for global reset to finish
      #100;
      wavesel=2'd0;
      #10000;
      wavesel=2'd1;
      #10000;

      // Add stimulus here

   end

endmodule // moogerfooger_test
module moogerfooger(input clock, ready,
                    input [1:0] wavesel,
                    input [9:0] period,
                    input [5:0] amount, // how much moogerfooger? max = all moogerfooger, min = none
                    input wire signed [17:0] x,
                    output reg signed [17:0] y);

   parameter [1:0] SQUARE=2'b00;
   parameter [1:0] SIN=2'b01;
   parameter [1:0] TRIANGLE=2'b10;
   parameter [1:0] SAW=2'b11;

   wire signed [8:0]                        sig;
```

```
   wire signed [1:0]                          squaresig;
   assign squaresig = {sig[8], sig[0]};

   // stage 1
   reg signed [19:0]                          ypre_sq;
   reg signed [26:0]                          ypre;
   // stage 2
   reg signed [17:0]                          y3;
   // stage 3
   reg signed [23:0]                          yscaled_raw;
   // x pipelining
   reg signed [17:0]                          x2;
   reg signed [17:0]                          x3;

   // the signal generator
   signalgen signalgen_moog (.clock(clock), .ready(ready), .pos(1'b0), .wavesel(wavesel), .knobval({period, 1'b0}), .y(sig));

   // note: could be one assign statement, but this is more readable
   always @(posedge clock) begin
      if (ready) begin
          // pipeline x
          x2 <= x;
          x3 <= x2;
          case(wavesel)
            SQUARE: begin// downshift by 1,
               // stage 1 -- moogerfoock it
               ypre_sq <= (x * squaresig);
               // stage 2 -- shift to get the output value
               y3 <= ypre_sq[19:2]<<<1;
            end
            default: begin
               // stage 1 -- moogerfoock it
               ypre <= (x * sig);
               // stage 2 -- shift to get the output value
               y3 <= ypre[26:9]<<<1;
            end
          endcase
          // stage 3 -- multiply by value and add
          yscaled_raw <= (y3 * amount) + (x3 * (6'b111111 - amount));
          // stage 4 -- scale
          y <= yscaled_raw[23:6];
      end
   end
endmodule // moogerfooger
`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   12:40:02 11/22/2011
// Design Name:   moogerfooger
// Module Name:   /afs/athena.mit.edu/user/d/s/dsherry/6.111/final/verilog//moogerfooger_test.v
// Project Name:  finalproj
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: moogerfooger
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module panning_test;

   // Inputs
   reg clock;
   reg reset;
   reg ready;
   reg fade_on;
   reg signed [8:0]speed_in;
   // reg [17:0] x;

   // Outputs
   wire signed [17:0] y_right;
   wire signed [17:0] y_left;

   // input signal generation params
   reg [1:0]   wavesel_x;
   reg [10:0]   rate_x;
   wire signed [8:0] x_raw;
   wire signed [17:0] x;

   // Instantiate the Unit Under Test (UUT)
   master_pan pan_mod(clock,reset,ready,fade_on,speed_in,x,{y_right,y_left});

   // the generator for x (input waveform)
```

```
   signalgen signalgen_dist(.clock(clock), .ready(ready), .pos(1'b0), .wavesel(wavesel_x), .knobval(rate_x), .y(x_raw));

   // handle square wave amplitude (==1)
   assign x = x_raw <<< 9;

   initial begin // 27mhz
      // let's say 1 picosecond timestep is now 19 picoseconds
      forever #1 clock <= ~clock;
   end

   always begin // 48khz
      ready=1;
      #2;
      ready=0;
      #1124;
   end

   initial begin
      // Initialize Inputs
      clock = 0;
      reset = 0;
      ready = 0;
      fade_on = 0;
      speed_in = 9'sd1;
      wavesel_x = 2'd1;
      rate_x = 10'd700;

      //x = 0;

      // Wait 100 ns for global reset to finish
      #100;
      #2000000;
      fade_on = 1;
      speed_in = 9'sd250;

      // Add stimulus here

   end

endmodule // moogerfooger_test
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   02:05:53 11/22/2011
// Design Name:   reverb
// Module Name:   /afs/athena.mit.edu/user/d/s/dsherry/6.111/final/verilog//reverb_test.v
// Project Name:  finalproj
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: reverb
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module reverb_test;

   // Inputs
   reg clock;
   reg reset;
   reg ready;
   reg waveon;
   reg [1:0] wavesel;
   reg [9:0] delay_in;
   reg [9:0] decay_in;
   reg [10:0] decay_knob_val;
   reg [17:0] x;

   // Outputs
   wire [17:0] y;

   // Instantiate the Unit Under Test (UUT)
   // reverb #(.LOGSIZE(5'd4)) uut (.clock(clock),
   reverb uut (.clock(clock),
               .reset(reset),
               .ready(ready),
               .waveon(waveon),
               .wavesel(wavesel),
               .delay_in(delay_in),
               .decay_in(decay_in),
               .decay_knob_val(decay_knob_val),
               .x(x),
               .y(y));
```

```
   initial begin // 27mhz
      // let's say 1 picosecond timestep is now 19 picoseconds
      forever #1 clock <= ~clock;
   end

   always begin // 48khz
      ready=1;
      #2;
      ready=0;
      #1124;
   end

   initial begin
      // Initialize Inputs
      clock = 1;
      reset = 1;
      ready = 0;
      // mess with these parameters for testing
      waveon = 1'b1;
      wavesel = 2'd1;

      delay_in = 10'd3;
      decay_in = 10'd512; // scale by half (at most)
      decay_knob_val = 11'd4;
      //decay_in = 10'd1023;
      x = 0;

      // Wait 100 ns for global reset to finish
      #100;
      //reset = 0;

      // Note: to convert from real time to internal time, multiply by 19.
      // To calculate real times from internal times, divide.
      // Add stimulus here
      #18000;
      #18000;
      reset = 0;
      #1000;
      //x = 0;
      x = 18'sd31071; // high impulse
      #1500;
      x = 0;
      // memory should be 0 now.


   end

endmodule

//`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   17:42:59 11/21/2011
// Design Name:   saw
// Module Name:   /afs/athena.mit.edu/user/d/s/dsherry/6.111/final/verilog//signalgen_saw_test.v
// Project Name:  finalproj
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: saw
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module signalgen_saw_test;

   // Inputs
   reg clock;
   reg ready;
   reg [10:0] knobval;

   // Outputs
   wire signed [8:0] out;

   // Instantiate the Unit Under Test (UUT)
   saw uut (
            .clock(clock),
            .ready(ready),
            .knobval(knobval),
            .out(out)
            );
```

```
   initial begin // 27mhz
      // 19 picoseconds
      forever #19 clock <= ~clock;
   end

   always begin // 48khz
      ready=1;
      #38;
      ready=0;
      #20769;
   end

   initial begin
      // Initialize Inputs
      clock = 0;
      ready = 0;
      knobval = 0;

      // Wait 100 ns for global reset to finish
      #100;

      // Add stimulus here
      knobval = 18'd3;
      #120000000;  // ~120 ms, the period of the wave
      knobval = 18'd64;
      #10000000; // ~5 ms, the new period
   end

endmodule

//`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   17:42:23 11/21/2011
// Design Name:   sin
// Module Name:   /afs/athena.mit.edu/user/d/s/dsherry/6.111/final/verilog//signalgen_sin_test.v
// Project Name:  finalproj
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: sin
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module signalgen_sin_test;

   // Inputs
   reg clock;
   reg ready;
   reg [10:0] knobval;

   // Outputs
   wire signed [8:0] out;

   // Instantiate the Unit Under Test (UUT)
   sin uut (
            .clock(clock),
            .ready(ready),
            .knobval(knobval),
            .out(out)
            );

   initial begin // 27mhz
      // 19 picoseconds
      forever #19 clock <= ~clock;
   end

   always begin // 48khz
      ready=1;
      #38;
      ready=0;
      #20769;
   end

   initial begin
      // Initialize Inputs
      clock = 0;
      ready = 0;
      knobval = 0;

      // Wait 100 ns for global reset to finish
      #100;
```

```
      // Add stimulus here
      knobval = 18'd3;
      #10000;
      // knobval = 18'd2;
      // #10000;
      // knobval = 18'd3;
      // #10000;
      // knobval = 18'd4;
      // #10000;
      // knobval = 18'd5;
      // #10000;
      // knobval = 18'd10;
      // #3000;
      // knobval = 18'd53;
      // #1000;
      // knobval = 18'd97;
      // #1000;
      // knobval = 18'd125;
      // #800;
      // knobval = 18'd293;
      // #700;
      // knobval = 18'd437;
      // #600;
      // knobval = 18'd647;
      // #500;
      // knobval = 18'd893;
      // #400;
      // knobval = 18'd1000;
      // #300;
      // knobval = 18'd1023;
      // #300;
      // knobval = 18'd2023;
      // #300;
      // knobval = 18'd4023;
      // #300;
      // knobval = 18'd8023;
      // #300;
      // knobval = 18'd10000;
      // #100;
      // knobval = 18'd10014;
      // #100;

   end

endmodule

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   17:42:02 11/21/2011
// Design Name:   square
// Module Name:   /afs/athena.mit.edu/user/d/s/dsherry/6.111/final/verilog//signalgen_square_test.v
// Project Name:  finalproj
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: square
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module signalgen_square_test;

   // Inputs
   reg clock;
   reg ready;
   reg [17:0] knobval;

   // Outputs
   wire       out;

   // Instantiate the Unit Under Test (UUT)
   square uut (
               .clock(clock),
               .ready(ready),
               .knobval(knobval),
               .out(out)
               );
   initial begin
      forever #1 clock <= ~clock;
   end
```

```
   always begin // every 6 steps (3 clock cycles)
      ready=1;
      #2;
      ready=0;
      #4;
   end

   initial begin
      // Initialize Inputs
      clock = 1;
      ready = 0;
      knobval = 18'd5;

      // Wait 100 ns for global reset to finish
      #100;

      // Add stimulus here
      // wait a few cycles
      #500;
      // change the knob val
      knobval = 18'd3;
      // wait a few more cycles
      #600;
      // change the knob val again
      knobval = 18'd10;
      // wait a few more cycles
      #600;

   end

endmodule

//`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   17:41:26 11/21/2011
// Design Name:   signalgen
// Module Name:   /afs/athena.mit.edu/user/d/s/dsherry/6.111/final/verilog//signalgen_test.v
// Project Name:  finalproj
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: signalgen
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module signalgen_test;

   // Inputs
   reg clock;
   reg ready;
   reg [1:0] wavesel;
   reg [10:0] knobval;
   reg        pos;

   // Outputs
   wire signed [8:0] y;

   // Instantiate the Unit Under Test (UUT)
   signalgen uut (.clock(clock),
         .ready(ready),
         .pos(pos),
         .wavesel(wavesel),
         .knobval(knobval),
         .y(y)
         );

   initial begin // 27mhz
      // 19 picoseconds
      forever #19 clock <= ~clock;
   end

   always begin // 48khz
      ready=1;
      #38;
      ready=0;
      #20769;
   end

   initial begin
      // Initialize Inputs
      clock = 0;
```

```
      ready = 0;
      knobval = 0;
      wavesel = 0;
      pos = 0;

      // Wait 100 ns for global reset to finish
      #100;

      // Add stimulus here
      knobval = 11'd64;
      #10000000; // ~5 ms, the new period
      wavesel=2'd1;
      #10000000;
      wavesel=2'd2;
      #10000000;
      wavesel=2'd3;
      #10000000;

      pos=1; // now try only positive output
      knobval=11'd64;
      wavesel=2'd0;
      #10000000; // ~5 ms, the new period
      wavesel=2'd1;
      #10000000;
      wavesel=2'd2;
      #10000000;
      wavesel=2'd3;
      #10000000;
   end
endmodule

//`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   17:42:40 11/21/2011
// Design Name:   triangle
// Module Name:   /afs/athena.mit.edu/user/d/s/dsherry/6.111/final/verilog//signalgen_triangle_test.v
// Project Name:  finalproj
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: triangle
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module signalgen_triangle_test;

   // Inputs
   reg clock;
   reg ready;
   reg [10:0] knobval;

   // Outputs
   wire signed [8:0] out;

   // Instantiate the Unit Under Test (UUT)
   triangle uut (
                 .clock(clock),
                 .ready(ready),
                 .knobval(knobval),
                 .out(out)
                 );
   initial begin // 27mhz
      // 19 picoseconds
      forever #19 clock <= ~clock;
   end

   always begin // 48khz
      ready=1;
      #38;
      ready=0;
      #20769;
   end

   initial begin
      // Initialize Inputs
      clock = 0;
      ready = 0;
      knobval = 0;

      // Wait 100 ns for global reset to finish
      #100;
```

```
      // Add stimulus here
      knobval = 18'd3;
      #120000000;  // ~120 ms, the period of the wave
      knobval = 18'd64;
      #10000000; // ~5 ms, the new period
   end

endmodule

// A test to establish what happens when a signed number is multiplied with
// 1) an unsigned number with MSB=0
// 1) an unsigned number with MSB=1

`timescale 1ns / 1ps

module m (input clock, input signed [17:0] sval, input [9:0] uval, output [27:0] result);

   assign result = sval * uval;

endmodule

module signed_unsigned_test;

   reg clock;
   reg signed [17:0] sval;
   reg [9:0]         uval;
   wire [27:0]       result;
   wire signed [17:0] sig;

   m m1(.clock(clock), .sval(sval), .uval(uval), .result(result));

   initial
     forever #1 clock <= ~clock;


   initial begin
      clock = 0;
      sval = 0;
      uval = 0;
      // Wait 100 ns for global reset to finish
      #100;

      sval = 18'sd10;
      // first, MSB=0
      uval = 10'd10;
      #10;

      // next, MSB=1
      uval = 10'd600;
      #10;

                // try a negative sval
      sval = -18'sd10;
      // next, MSB=0
      uval = 10'd10;
      #10;

      // next, MSB=1
      uval = 10'd600;
      #10;

      // Add stimulus here
   end
endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   12:40:02 11/22/2011
// Design Name:   moogerfooger
// Module Name:   /afs/athena.mit.edu/user/d/s/dsherry/6.111/final/verilog//moogerfooger_test.v
// Project Name:  finalproj
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: moogerfooger
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module trem_test;
```

```
   // Inputs
   reg clock;
   reg ready;
   reg signed [9:0] decayval;
         reg [9:0] halfperiod;
   // reg [17:0] x;

   // Outputs
   wire signed [17:0] y;

   // input signal generation params
   reg [1:0]   wavesel_x;
   reg [10:0]   rate_x;
   wire signed [8:0] x_raw;
   wire signed [17:0] x;

   // Instantiate the Unit Under Test (UUT)
   tremolo trem_mod(clock,ready,decayval,halfperiod,x,y);

   // the generator for x (input waveform)
   signalgen signalgen_dist(.clock(clock), .ready(ready), .pos(1'b0), .wavesel(wavesel_x), .knobval(rate_x), .y(x_raw));

   // handle square wave amplitude (==1)
   assign x = x_raw <<< 9;

   initial begin // 27mhz
      // let's say 1 picosecond timestep is now 19 picoseconds
      forever #1 clock <= ~clock;
   end

   always begin // 48khz
      ready=1;
      #2;
      ready=0;
      #1124;
   end

   initial begin
      // Initialize Inputs
      clock = 0;
      ready = 0;
      decayval = 10'sd0;
      halfperiod = 10'd1;
      wavesel_x = 2'd1;
      rate_x = 10'd700;

      //x = 0;

      // Wait 100 ns for global reset to finish
      #100;
      #500000;
      decayval = 10'sd50;
      #500000;
      decayval = 10'sd150;
      #500000;
      decayval = 10'sd300;
      #500000;
      decayval = 10'sd500;

      // Add stimulus here

   end

endmodule // moogerfooger_test
`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   02:04:51 12/09/2011
// Design Name:   zbt_interface
// Module Name:   /afs/athena.mit.edu/user/d/s/dsherry/6.111/final/verilog//zbt_interface_test.v
// Project Name:  final
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: zbt_interface
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module zbt_interface_test_nofake;

   // Inputs
   reg clock;
```

```
reg reset;
reg ready;
reg we;
reg re;
reg loop;
reg [1:0] channel_in_sel;
reg [35:0] data_in;
reg [35:0] ram0_read_data;
reg [35:0] ram1_read_data;

// Outputs
wire [35:0] ram0_write_data;
wire [35:0] ram1_write_data;
wire        ram0_we;
wire        ram1_we;
wire [18:0] ram0_addr;
wire [18:0] ram1_addr;
wire        zbt_go;
wire        data_flag;
wire [35:0] ch01out;
wire [35:0] ch23out;
wire [4:0]  state_out;
wire [17:0] addr_current_0_out;
wire [17:0] addr_current_1_out;
wire [17:0] addr_current_2_out;
wire [17:0] addr_current_3_out;
wire [17:0] addr_current_bak_0_out;
wire [17:0] addr_current_bak_1_out;
wire [17:0] addr_current_bak_2_out;
wire [17:0] addr_current_bak_3_out;
wire [17:0] addr_end_0_out;
wire [17:0] addr_end_1_out;
wire [17:0] addr_end_2_out;
wire [17:0] addr_end_3_out;

// Instantiate the Unit Under Test (UUT)
zbt_interface uut (
                   .clock(clock),
                   .reset(reset),
                   .ready(ready),
                   .we(we),
                   .re(re),
                   .loop(loop),
                   .channel_in_sel(channel_in_sel),
                   .data_in(data_in),
                   .ram0_read_data(ram0_read_data),
                   .ram1_read_data(ram1_read_data),
                   .ram0_write_data(ram0_write_data),
                   .ram1_write_data(ram1_write_data),
                   .ram0_we(ram0_we),
                   .ram1_we(ram1_we),
                   .ram0_addr(ram0_addr),
                   .ram1_addr(ram1_addr),
                   .zbt_go(zbt_go),
                   .data_flag(data_flag),
                   .ch01out(ch01out),
                   .ch23out(ch23out),
                   .state_out(state_out),
                   .addr_current_0_out(addr_current_0_out),
                   .addr_current_1_out(addr_current_1_out),
                   .addr_current_2_out(addr_current_2_out),
                   .addr_current_3_out(addr_current_3_out),
                   .addr_current_bak_0_out(addr_current_bak_0_out),
                   .addr_current_bak_1_out(addr_current_bak_1_out),
                   .addr_current_bak_2_out(addr_current_bak_2_out),
                   .addr_current_bak_3_out(addr_current_bak_3_out),
                   .addr_end_0_out(addr_end_0_out),
                   .addr_end_1_out(addr_end_1_out),
                   .addr_end_2_out(addr_end_2_out),
                   .addr_end_3_out(addr_end_3_out)
                   );

initial begin
   forever #1 clock <= ~clock;
end

always begin // once every 20 clock cycles
   #1;
   ready=1;
   #2;
   ready=0;
   #17;
end

initial begin
   // Initialize Inputs
   clock = 1'b1;
   reset = 1'b1;
   ready = 0;
   we = 0;
   re = 0;
   loop = 0;
```

```
channel_in_sel = 0;
data_in = 0;
ram0_read_data = 0;
ram1_read_data = 0;

// Wait 100 ns for global reset to finish
#100;
reset = 0;
// Add stimulus here
#100;
#7;
// write some data to channel 0
we = 1'b1;
data_in = {18'sd2, 18'sd2};
ram0_read_data = {18'sd3, 18'sd3};
#13;
// wait for that data to be written
// watch the address and output data
#40;
data_in = {18'sd4, 18'sd4};
ram0_read_data = {18'sd5, 18'sd5};
#40;
data_in = {18'sd6, 18'sd6};
ram0_read_data = {18'sd7, 18'sd7};
#40;
data_in = {18'sd8, 18'sd8};
ram0_read_data = {18'sd9, 18'sd9};
#40;
data_in = {18'sd40, 18'sd40};
ram0_read_data = {18'sd41, 18'sd41};
#40;
data_in = {18'sd42, 18'sd42};
ram0_read_data = {18'sd43, 18'sd43};
#40;
// now try reading
we = 0;
re = 1'b1;
#25;
data_in = {18'sd10, 18'sd10};
ram0_read_data = {18'sd11, 18'sd11};
#4;
ram0_read_data = {18'sd101, 18'sd101};
#36;
data_in = {18'sd12, 18'sd12};
ram0_read_data = {18'sd13, 18'sd13};
#4;
ram0_read_data = {18'sd101, 18'sd101};
#36;
data_in = {18'sd14, 18'sd14};
ram0_read_data = {18'sd15, 18'sd15};
#4;
ram0_read_data = {18'sd101, 18'sd101};
#36;
data_in = {18'sd16, 18'sd16};
ram0_read_data = {18'sd17, 18'sd17};
#4;
ram0_read_data = {18'sd101, 18'sd101};
#36;
#160;
// now "write" to channel 1
// #7;
// write some data to channel 0
we = 1'b1;
re = 0;
channel_in_sel = 2'd1;
data_in = {18'sd2, 18'sd2};
ram1_read_data = {18'sd3, 18'sd3};
#13;
// wait for that data to be written
// watch the address and output data
#40;
data_in = {18'sd4, 18'sd4};
ram1_read_data = {18'sd5, 18'sd5};
#40;
data_in = {18'sd6, 18'sd6};
ram1_read_data = {18'sd7, 18'sd7};
#40;
data_in = {18'sd8, 18'sd8};
ram1_read_data = {18'sd9, 18'sd9};
#40;
// now try reading again
we = 0;
re = 1'b1;
#20;
data_in = {18'sd10, 18'sd10};
ram1_read_data = {18'sd11, 18'sd11};
#4;
ram0_read_data = {18'sd101, 18'sd101};
#36;
data_in = {18'sd12, 18'sd12};
ram1_read_data = {18'sd13, 18'sd13};
#4;
```

```
ram0_read_data = {18'sd101, 18'sd101};
#36;
data_in = {18'sd14, 18'sd14};
ram1_read_data = {18'sd15, 18'sd15};
#4;
ram0_read_data = {18'sd101, 18'sd101};
#36;
data_in = {18'sd16, 18'sd16};
ram1_read_data = {18'sd17, 18'sd17};
#4;
ram0_read_data = {18'sd101, 18'sd101};
#36;
data_in = {18'sd56, 18'sd56};
ram1_read_data = {18'sd57, 18'sd57};
#4;
ram0_read_data = {18'sd101, 18'sd101};
#36;
data_in = {18'sd58, 18'sd58};
ram1_read_data = {18'sd59, 18'sd59};
#4;
ram0_read_data = {18'sd101, 18'sd101};
#36;
data_in = {18'sd60, 18'sd60};
ram1_read_data = {18'sd61, 18'sd61};
#4;
ram0_read_data = {18'sd101, 18'sd101};
#36;
// turn off the looper for a while
re = 0;
#240;
// ************************************************************
// run the same test as before, except for channels 2/3 (copypasta)
#7;
// write some data to channel 0
we = 1'b1;
data_in = {18'sd2, 18'sd2};
ram1_read_data = {18'sd3, 18'sd3};
channel_in_sel = 2'd2;
#13;
// wait for that data to be written
// watch the address and output data
#40;
data_in = {18'sd4, 18'sd4};
ram1_read_data = {18'sd5, 18'sd5};
#40;
data_in = {18'sd6, 18'sd6};
ram1_read_data = {18'sd7, 18'sd7};
#40;
data_in = {18'sd8, 18'sd8};
ram1_read_data = {18'sd9, 18'sd9};
#40;
data_in = {18'sd40, 18'sd40};
ram1_read_data = {18'sd41, 18'sd41};
#40;
data_in = {18'sd42, 18'sd42};
ram1_read_data = {18'sd43, 18'sd43};
#40;
// now try reading
we = 0;
re = 1'b1;
data_in = {18'sd10, 18'sd10};
ram1_read_data = {18'sd11, 18'sd11};
#40;
data_in = {18'sd12, 18'sd12};
ram1_read_data = {18'sd13, 18'sd13};
#40;
data_in = {18'sd14, 18'sd14};
ram1_read_data = {18'sd15, 18'sd15};
#40;
data_in = {18'sd16, 18'sd16};
ram1_read_data = {18'sd17, 18'sd17};
#40;
#160;
// now "write" to channel 1
#7;
// write some data to channel 0
we = 1'b1;
re = 0;
channel_in_sel = 2'd3;
data_in = {18'sd2, 18'sd2};
ram1_read_data = {18'sd3, 18'sd3};
#13;
// wait for that data to be written
// watch the address and output data
#40;
data_in = {18'sd4, 18'sd4};
ram1_read_data = {18'sd5, 18'sd5};
#40;
data_in = {18'sd6, 18'sd6};
ram1_read_data = {18'sd7, 18'sd7};
#40;
data_in = {18'sd8, 18'sd8};
```

```
      ram1_read_data = {18'sd9, 18'sd9};
      #40;
      // now try reading again
      we = 0;
      re = 1'b1;
      data_in = {18'sd10, 18'sd10};
      ram1_read_data = {18'sd11, 18'sd11};
      #40;
      data_in = {18'sd12, 18'sd12};
      ram1_read_data = {18'sd13, 18'sd13};
      #40;
      data_in = {18'sd14, 18'sd14};
      ram1_read_data = {18'sd15, 18'sd15};
      #40;
      data_in = {18'sd16, 18'sd16};
      ram1_read_data = {18'sd17, 18'sd17};
      #40;
      data_in = {18'sd56, 18'sd56};
      ram1_read_data = {18'sd57, 18'sd57};
      #40;
      data_in = {18'sd58, 18'sd58};
      ram1_read_data = {18'sd59, 18'sd59};
      #40;
      data_in = {18'sd60, 18'sd60};
      ram1_read_data = {18'sd61, 18'sd61};
      #40;
      // turn off the looper for a while
      re = 0;


   end

endmodule

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   02:04:51 12/09/2011
// Design Name:   zbt_interface
// Module Name:   /afs/athena.mit.edu/user/d/s/dsherry/6.111/final/verilog//zbt_interface_test.v
// Project Name:  final
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: zbt_interface
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module zbt_interface_test;

   // Inputs
   reg clock;
   reg reset;
   reg ready;
   reg we;
   reg re;
   reg loop;
   reg [1:0] channel_in_sel;
   reg [35:0] data_in;
   reg [35:0] ram0_read_data;
   reg [35:0] ram1_read_data;

   // Outputs
   wire [35:0] ram0_write_data;
   wire [35:0] ram1_write_data;
   wire        ram0_we;
   wire        ram1_we;
   wire [18:0] ram0_addr;
   wire [18:0] ram1_addr;
   wire        zbt_go;
   wire        data_flag;
   wire [35:0] ch01out;
   wire [35:0] ch23out;
   wire [4:0]  state_out;
   wire [17:0] addr_current_0_out;
   wire [17:0] addr_current_1_out;
   wire [17:0] addr_current_2_out;
   wire [17:0] addr_current_3_out;
   wire [17:0] addr_current_bak_0_out;
   wire [17:0] addr_current_bak_1_out;
   wire [17:0] addr_current_bak_2_out;
   wire [17:0] addr_current_bak_3_out;
```

```
wire [17:0] addr_end_0_out;
wire [17:0] addr_end_1_out;
wire [17:0] addr_end_2_out;
wire [17:0] addr_end_3_out;

// Instantiate the Unit Under Test (UUT)
zbt_interface uut (
                    .clock(clock),
                    .reset(reset),
                    .ready(ready),
                    .we(we),
                    .re(re),
                    .loop(loop),
                    .channel_in_sel(channel_in_sel),
                    .data_in(data_in),
                    .ram0_read_data(ram0_read_data),
                    .ram1_read_data(ram1_read_data),
                    .ram0_write_data(ram0_write_data),
                    .ram1_write_data(ram1_write_data),
                    .ram0_we(ram0_we),
                    .ram1_we(ram1_we),
                    .ram0_addr(ram0_addr),
                    .ram1_addr(ram1_addr),
                    .zbt_go(zbt_go),
                    .data_flag(data_flag),
                    .ch01out(ch01out),
                    .ch23out(ch23out),
                    .state_out(state_out),
                    .addr_current_0_out(addr_current_0_out),
                    .addr_current_1_out(addr_current_1_out),
                    .addr_current_2_out(addr_current_2_out),
                    .addr_current_3_out(addr_current_3_out),
                    .addr_current_bak_0_out(addr_current_bak_0_out),
                    .addr_current_bak_1_out(addr_current_bak_1_out),
                    .addr_current_bak_2_out(addr_current_bak_2_out),
                    .addr_current_bak_3_out(addr_current_bak_3_out),
                    .addr_end_0_out(addr_end_0_out),
                    .addr_end_1_out(addr_end_1_out),
                    .addr_end_2_out(addr_end_2_out),
                    .addr_end_3_out(addr_end_3_out)
                    );

wire signed [35:0] data_in_read; // the output from the fake_zbt
wire signed [35:0] data_in_master; // the one that is fed to zbt_interface
// this assignment allows values to be manaually driven during writes in the actual test below
assign data_in_master = ((state_out[4:3] == 2'b11) // in write mode, or not
                          ? data_in : data_in_read);

// instantiate the random output generator
fake_zbt data_generator (.clock(clock),
                     .reset(reset),
                     .send_next_channel(zbt_go),
                     .data_out(data_in_read)
                     );

// instantiate the fake zbt module
fake_zbt fake_zbt1 (.clock(clock),
                     .reset(reset),
                     .ready(ready),

                     .send_next_channel(zbt_go),
                     .data_out(data_in_read)
                     );

initial begin
   forever #1 clock <= ~clock;
end

always begin // once every 20 clock cycles
   #1;
   ready=1;
   #2;
   ready=0;
   #17;
end

initial begin
   // Initialize Inputs
   clock = 1'b1;
   reset = 1'b1;
   ready = 0;
   we = 0;
   re = 0;
   loop = 0;
   channel_in_sel = 0;
   data_in = 0;
   ram0_read_data = 0;
   ram1_read_data = 0;

   // Wait 100 ns for global reset to finish
   #100;
   reset = 0;
```

```
// Add stimulus here
#100;
#7;
// write some data to channel 0
we = 1'b1;
data_in = {18'sd2, 18'sd2};
ram0_read_data = {18'sd3, 18'sd3};
#13;
// wait for that data to be written
// watch the address and output data
#40;
data_in = {18'sd4, 18'sd4};
ram0_read_data = {18'sd5, 18'sd5};
#40;
data_in = {18'sd6, 18'sd6};
ram0_read_data = {18'sd7, 18'sd7};
#40;
data_in = {18'sd8, 18'sd8};
ram0_read_data = {18'sd9, 18'sd9};
#40;
data_in = {18'sd40, 18'sd40};
ram0_read_data = {18'sd41, 18'sd41};
#40;
data_in = {18'sd42, 18'sd42};
ram0_read_data = {18'sd43, 18'sd43};
#40;
// now try reading
we = 0;
re = 1'b1;
#25;
data_in = {18'sd10, 18'sd10};
ram0_read_data = {18'sd11, 18'sd11};
#4;
ram0_read_data = {18'sd101, 18'sd101};
#36;
data_in = {18'sd12, 18'sd12};
ram0_read_data = {18'sd13, 18'sd13};
#4;
ram0_read_data = {18'sd101, 18'sd101};
#36;
data_in = {18'sd14, 18'sd14};
ram0_read_data = {18'sd15, 18'sd15};
#4;
ram0_read_data = {18'sd101, 18'sd101};
#36;
data_in = {18'sd16, 18'sd16};
ram0_read_data = {18'sd17, 18'sd17};
#4;
ram0_read_data = {18'sd101, 18'sd101};
#36;
#160;
// now "write" to channel 1
// #7;
// write some data to channel 0
we = 1'b1;
re = 0;
channel_in_sel = 2'd1;
data_in = {18'sd2, 18'sd2};
ram1_read_data = {18'sd3, 18'sd3};
#13;
// wait for that data to be written
// watch the address and output data
#40;
data_in = {18'sd4, 18'sd4};
ram1_read_data = {18'sd5, 18'sd5};
#40;
data_in = {18'sd6, 18'sd6};
ram1_read_data = {18'sd7, 18'sd7};
#40;
data_in = {18'sd8, 18'sd8};
ram1_read_data = {18'sd9, 18'sd9};
#40;
// now try reading again
we = 0;
re = 1'b1;
#20;
data_in = {18'sd10, 18'sd10};
ram1_read_data = {18'sd11, 18'sd11};
#4;
ram0_read_data = {18'sd101, 18'sd101};
#36;
data_in = {18'sd12, 18'sd12};
ram1_read_data = {18'sd13, 18'sd13};
#4;
ram0_read_data = {18'sd101, 18'sd101};
#36;
data_in = {18'sd14, 18'sd14};
ram1_read_data = {18'sd15, 18'sd15};
#4;
ram0_read_data = {18'sd101, 18'sd101};
#36;
data_in = {18'sd16, 18'sd16};
```

```
ram1_read_data = {18'sd17, 18'sd17};
#4;
ram0_read_data = {18'sd101, 18'sd101};
#36;
data_in = {18'sd56, 18'sd56};
ram1_read_data = {18'sd57, 18'sd57};
#4;
ram0_read_data = {18'sd101, 18'sd101};
#36;
data_in = {18'sd58, 18'sd58};
ram1_read_data = {18'sd59, 18'sd59};
#4;
ram0_read_data = {18'sd101, 18'sd101};
#36;
data_in = {18'sd60, 18'sd60};
ram1_read_data = {18'sd61, 18'sd61};
#4;
ram0_read_data = {18'sd101, 18'sd101};
#36;
// turn off the looper for a while
re = 0;
#240;
// ************************************************************
// run the same test as before, except for channels 2/3 (copypasta)
#7;
// write some data to channel 0
we = 1'b1;
data_in = {18'sd2, 18'sd2};
ram1_read_data = {18'sd3, 18'sd3};
channel_in_sel = 2'd2;
#13;
// wait for that data to be written
// watch the address and output data
#40;
data_in = {18'sd4, 18'sd4};
ram1_read_data = {18'sd5, 18'sd5};
#40;
data_in = {18'sd6, 18'sd6};
ram1_read_data = {18'sd7, 18'sd7};
#40;
data_in = {18'sd8, 18'sd8};
ram1_read_data = {18'sd9, 18'sd9};
#40;
data_in = {18'sd40, 18'sd40};
ram1_read_data = {18'sd41, 18'sd41};
#40;
data_in = {18'sd42, 18'sd42};
ram1_read_data = {18'sd43, 18'sd43};
#40;
// now try reading
we = 0;
re = 1'b1;
data_in = {18'sd10, 18'sd10};
ram1_read_data = {18'sd11, 18'sd11};
#40;
data_in = {18'sd12, 18'sd12};
ram1_read_data = {18'sd13, 18'sd13};
#40;
data_in = {18'sd14, 18'sd14};
ram1_read_data = {18'sd15, 18'sd15};
#40;
data_in = {18'sd16, 18'sd16};
ram1_read_data = {18'sd17, 18'sd17};
#40;
#160;
// now "write" to channel 1
#7;
// write some data to channel 0
we = 1'b1;
re = 0;
channel_in_sel = 2'd3;
data_in = {18'sd2, 18'sd2};
ram1_read_data = {18'sd3, 18'sd3};
#13;
// wait for that data to be written
// watch the address and output data
#40;
data_in = {18'sd4, 18'sd4};
ram1_read_data = {18'sd5, 18'sd5};
#40;
data_in = {18'sd6, 18'sd6};
ram1_read_data = {18'sd7, 18'sd7};
#40;
data_in = {18'sd8, 18'sd8};
ram1_read_data = {18'sd9, 18'sd9};
#40;
// now try reading again
we = 0;
re = 1'b1;
data_in = {18'sd10, 18'sd10};
ram1_read_data = {18'sd11, 18'sd11};
#40;
```

```
        data_in = {18'sd12, 18'sd12};
        ram1_read_data = {18'sd13, 18'sd13};
        #40;
        data_in = {18'sd14, 18'sd14};
        ram1_read_data = {18'sd15, 18'sd15};
        #40;
        data_in = {18'sd16, 18'sd16};
        ram1_read_data = {18'sd17, 18'sd17};
        #40;
        data_in = {18'sd56, 18'sd56};
        ram1_read_data = {18'sd57, 18'sd57};
        #40;
        data_in = {18'sd58, 18'sd58};
        ram1_read_data = {18'sd59, 18'sd59};
        #40;
        data_in = {18'sd60, 18'sd60};
        ram1_read_data = {18'sd61, 18'sd61};
        #40;
        // turn off the looper for a while
        re = 0;


    end

endmodule
```