FPGA Guitar Multi-Effects Processor

6.111 Final Project Report

Haris Brkic
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Table of Contents

Overview

First analog guitar pedals appeared in the 1930s. Shortly after being introduced to guitar players, they became an essential part of any guitarist's rig. The components used to make these pedals have changed over the years but the sound effects they produce have remained the same. It is quite expensive to make a versatile pedalboard since we would need a lot of analog pedals to complete it.

The goal of this project was to simulate the behavior of analog pedals using digital signal processing to create the digital equivalents of a large variety of pedals used by guitarists. By doing so, we created a significantly cheaper version of a multi-effects pedalboard that can be controlled from our labkit.

We digitized the analog signal from the guitar's magnetic pickups using the AC97 codec and passed the digitized signal to the Effects Controller module. The AC97 allows us to sample the incoming audio from the guitar's magnetic pickup at 48 kHz and ensures high-quality audio output.

The Effects Controller module is an FSM that cycles through eight states and at each step applies one of the effects if they are being used by passing the current input. The effects are controlled by the labkit switches and the active effects are passed to the logic controlling the graphics found in the top-level module, so they can be displayed on a monitor.

Besides this, the digitized signal is passed to a looper/recorder module found in the top-level module. When active, the looper stores the input signal in the onboard ZBT memory and allows us to play the stored audio so that we can play and store multiple parts of a musical phrase or song.

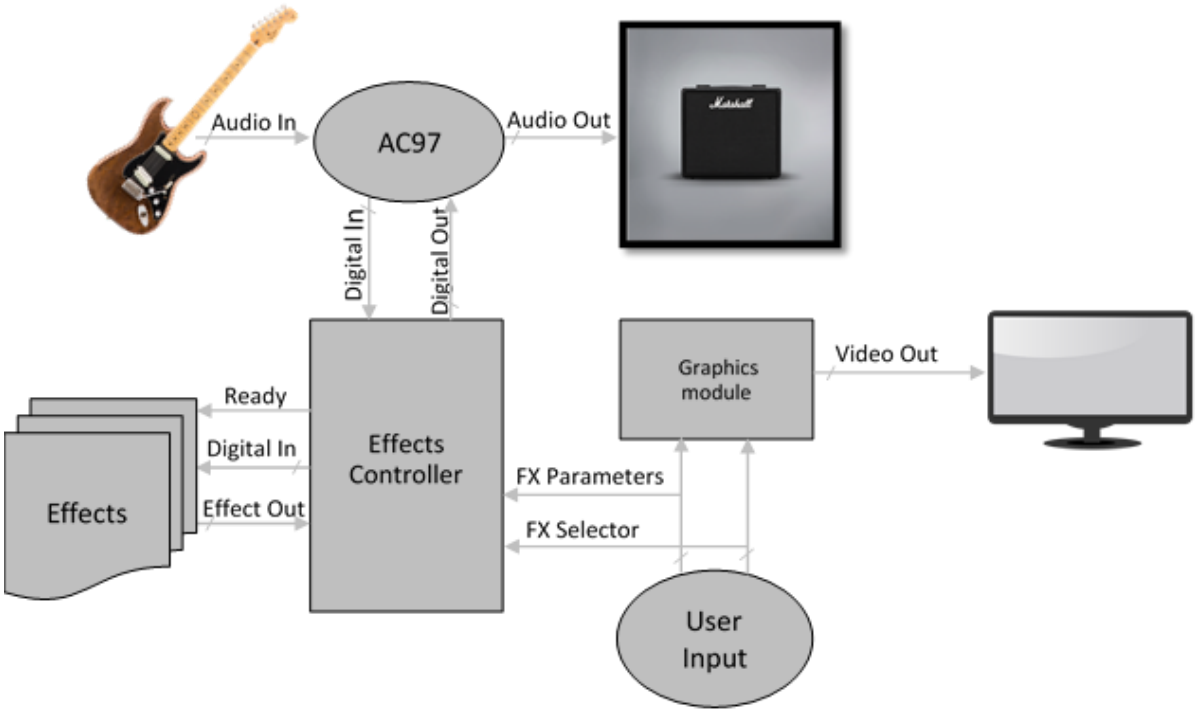

Figure 1: High-level Block Diagram

Design Implementation

FPGA Processor

FPGA Processor is a slightly modified version of the *lab5* module. The main two modifications are the inclusion of the *looper* and graphics logic inside this module. The looper must be included in the top-level module in order to interact with the onboard ZBT memory. Signals used for generating the graphics are also found in the *FPGA Processor* module, so the VGA code was put inside that module.

The other modifications to the *lab5* module were minor. We switched from using 8-bit wide input and output in *lab5audio* to using signed 20-bit wide audio input and output. This is because lower number of bits resulted in unwanted distortion of the input signal.

Lab5 module proved to be a good communication link between the magnetic pickups and the onboard AC97. This setup did not require any form of preprocessing of the input signal. Furthermore, the output from this module was fully compatible with the guitar amplifier. The output from this module was the sum of the outputs of the *looper* and the *Effects Controller*.

The *Effects Controller* is instantiated within the *FPGA Processor* and the debounced values of the labkit switches are passed to it to control the effects.

Looper allows a performer to record and replay a phrase in a loop. In addition, it allows the performer to record multiple phrases or melodies on top of each other.

The *looper* is implemented like the lab5 recorder. Both are implemented as simple FSMs that increment the read or write memory address after each clock cycle and store and/or output the value inside the current memory slot.

The major difference between the two is that the *looper* allows to record over the currently stored values inside the ZBT memory slots. To achieve this, we introduced a new register which would hold the current value inside the ZBT memory slot prior to the overwrite. After three clock cycles, we store the sum of the value stored within the register and the current effects output into the same memory address.

One problem that arose with this implementation was the timing of the write enable signal used by the ZBT memory. The write enable signal must be held longer than 0.5 ns. Otherwise, the labkit does not have enough time to recognize that it needs to store the passed data in the current memory slot. To solve this problem, we introduced three intermediate states and held the write enable as an active low during those states. After exiting the last intermediate state, we reset the write enable signal and increment the memory address.

One thing to keep in mind is overflow due to too much overwriting. We did not encounter this issue due to the signal values being relatively small and the fact that we used 20-bit wide registers and wires.

The graphics was implemented using the code that allows us to display ASCII characters on a monitor. The code and the font files are available on the course website. The inputs are the vertical and horizontal location of the string, and the string, and the output is the RGB value associated with the pixel location.

Using this code, all the effect names were displayed along with their current state. The states were to the right of the effect names and switched between different strings whenever the state of an effect changed. The VGA module used was identical to the one used for lab 3.

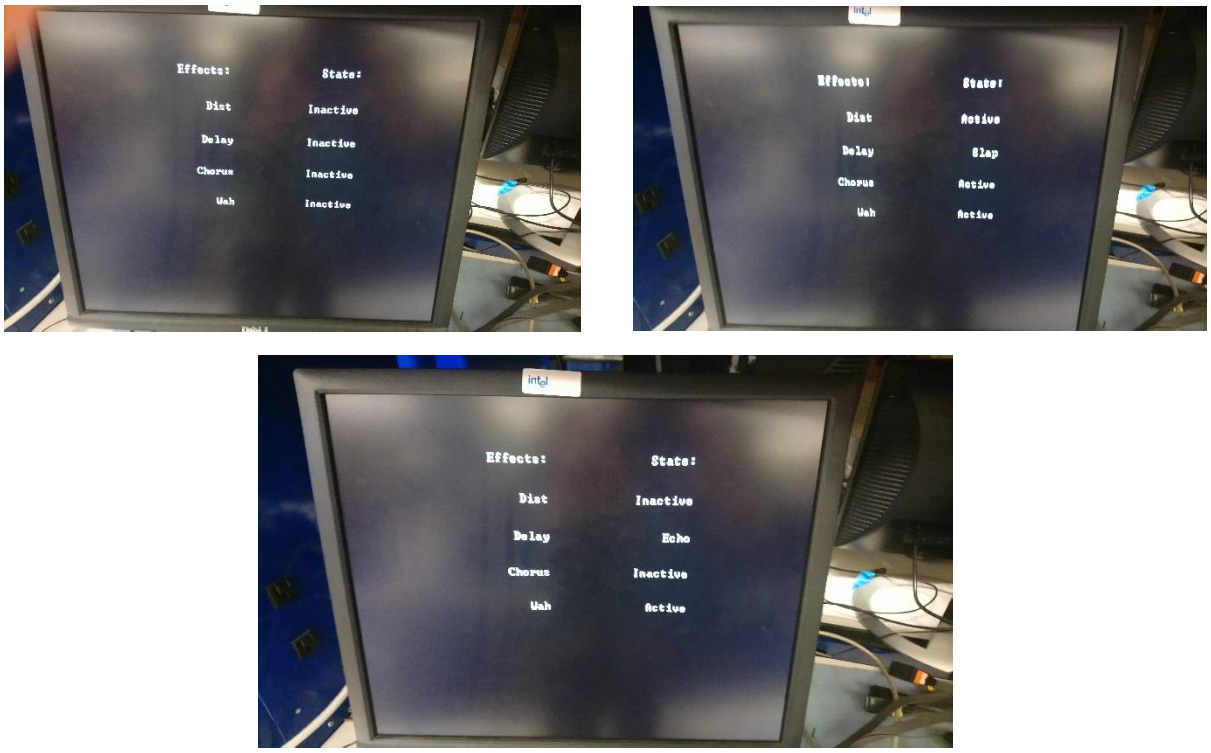

Figure 2: VGA output

Effects Controller

The AC97 digitizes the guitar signal and outputs the digital signal to the *Effects Controller* which passes that signal to the appropriate effects. Our effect modules alter the digital signal and return it back to the *Effects Controller.*

Effects Controller is a simple FSM that initializes a temporary register in which it stores the signal modified by the effects. It has eight states and each one corresponds to a particular effect. The order in which the effects are applied matters. Any form of filtering is supposed to happen before being passed to other effects. Similarly, effects which store the output must be placed at the end of the chain.

The chain of effects that yielded the best results was: *wah, distortion, chorus, delay,* and *looper.* Every effect module takes in the mentioned temporary register as input, performs the necessary calculations and stores the new signal back in the same register at the next positive clock edge.

Each of the effects requires its own ready signal. This is because all the effects share the same input signal and must wait until the previous effect has finished modifying the temporary register. Therefore, we change the ready signal for each of the effects when we transition into the state representing that specific effect.

Once we reach the last state, the temporary register is stored into the register holding the output and passed to the top-level module. After that, *Effects Controller* waits for the next sample and the ready signal from the AC97. Even though we take 8 additional clock cycles to produce the output once we receive the input signal, we maintain the 48 kHz sampling rate because the labkit uses a 27 MHz clock.

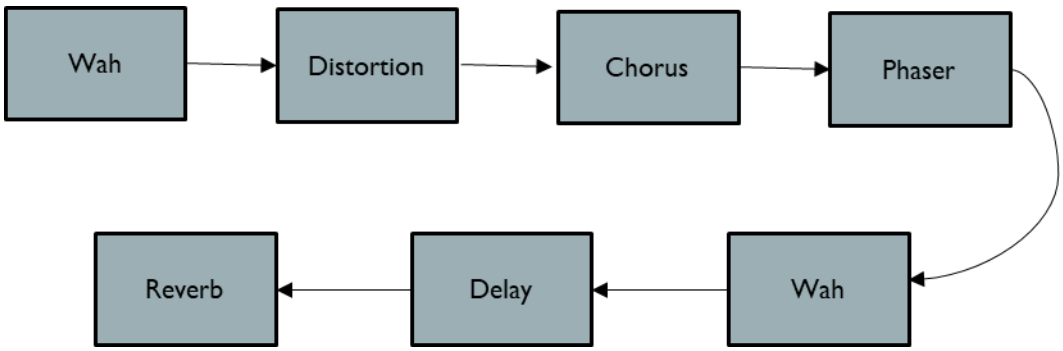

Figure 3: Proper effects chain

Distortion

Distortion results from „hard clipping“ the instrument's audio signal. By clipping the signal, the wave form of the input signal is distorted, and overtones are added. Harmonic overtones produce a „warm“ sound and inharmonic overtones contribute towards a more „gritty“ sound. „Hard clipping“ means that the input audio signal is re-shaped such that it has perfectly flattened peaks.

We implemented this effect by comparing the input value to four different threshold values. These four thresholds consisted of two pairs of with equal absolute values with different signs. Any input value above the highest or lowest threshold results in an output equal to the corresponding threshold. Values in between two thresholds result in an output equal to the threshold with smaller absolute value. All other inputs are directly passed as outputs.

One issue that was encountered was making sure this module only produces output on the AC97 ready signal. Otherwise, the output will be just silence since this module will output the same signal to many times in a row.

This was the simplest effect to implement. It was implemented during the first week. This effect becomes better with an increase in the number of thresholds used. We observed this because the initial implementation of *distortion* used one pair of thresholds and the sound was heavily distorted.

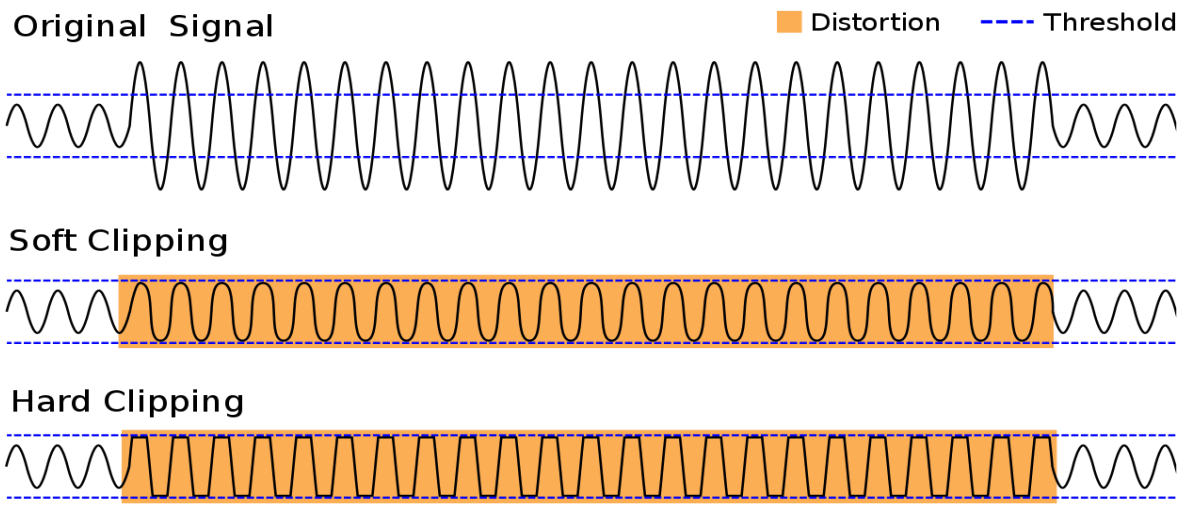

Figure 4: Comparison between a normal and distorted sound wave

Delay

Delay pedals produce an echo effect by reproducing an input audio signal with a slight time-delay. The two types of delay are „slap“ (single repetition) and echo (multiple repetitions).

Both types are implemented using BRAM to store the input signal and later mix them with the input. However, this means that we need a dual-port BRAM so the *bram* module from lab5 is not going to work. To change the lab5 *bram* module into a dual-port BRAM module all we need to do is to add one more input. The input added is a read address that is different from the write address. This way, we can both read and write to the BRAM in the same clock cycle. The output of the dual-port BRAM is the value inside the read address.

At each ready signal, we store the input into the BRAM and output the input signal mixed with an earlier stored signal. The difference between „slap“ and echo is in the data they store to the BRAM. „Slap“ stores only the input data while echo stores the input and the output of the BRAM at half volume (extended right shift). By doing this, we create the feedback loop required to implement the decaying echo delay effect.

To create a consistent and nice sounding delay, the difference between the read and write memory addresses passed as inputs to the BRAM should be constant. Otherwise, the sound we get becomes muddled.

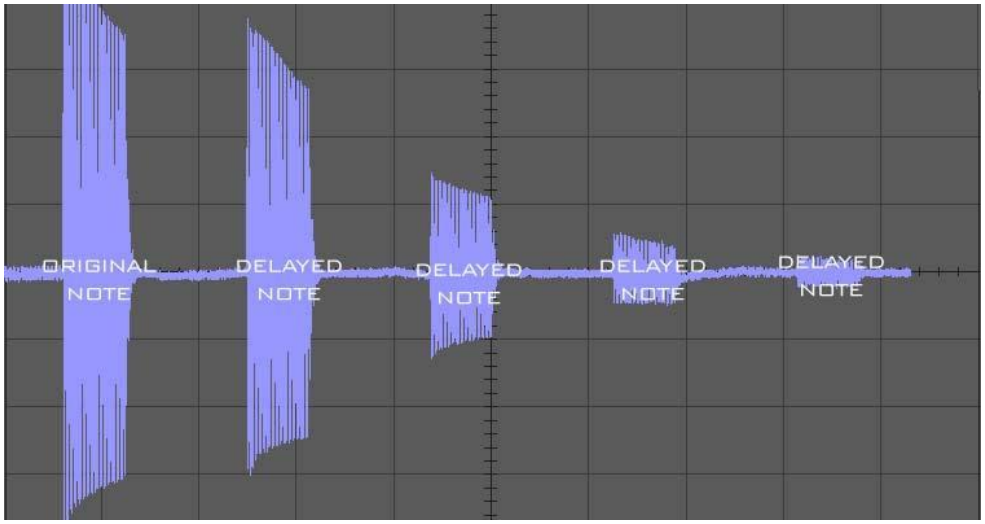

Figure 5: Illustration of the delay effect

Auto-wah

Auto-wah is an effect that sounds similar to a crying baby. This pedal is a moving bandpass filter whose frequency center is periodically changed. It accents a part of the inputs frequency and diminishes the rest of the input. The filters were written in MATLAB and the coefficients were stored in a ROM within the labkit. We used the labkit to change between the filters.

The *auto-wah* module periodically sweeps between the ten bandpass filter outputs and mixes the filter output with the input signal. After mixing them, it outputs the mixed signal. This module has two states: INCREASING and DECREASING. While in the INCREASING state we go to the next bandpass filter whose center frequency is higher. At each step it mixes the input with the output of the current filter. Once we reach the bandpass filter with the highest center frequency we go to the DECREASING state. The DECREASING state performs the same operations except it switches to the bandpass filter with a lower cutoff frequency after each period.

The speed at which the filters switch is input by the user, a good one is a 100 Hz. Higher speeds have a negative effect on the output. If the speed used is too close to the sampling rate, the effect results in a noisy output audio signal. Another problem was encountered when the filter output was right shifted. This caused the output to become distorted. Removing the extended right shift solved this issue.

This effect required the most time to implement. Furthermore, it was the most tedious because all the filter coefficients had to be entered manually.

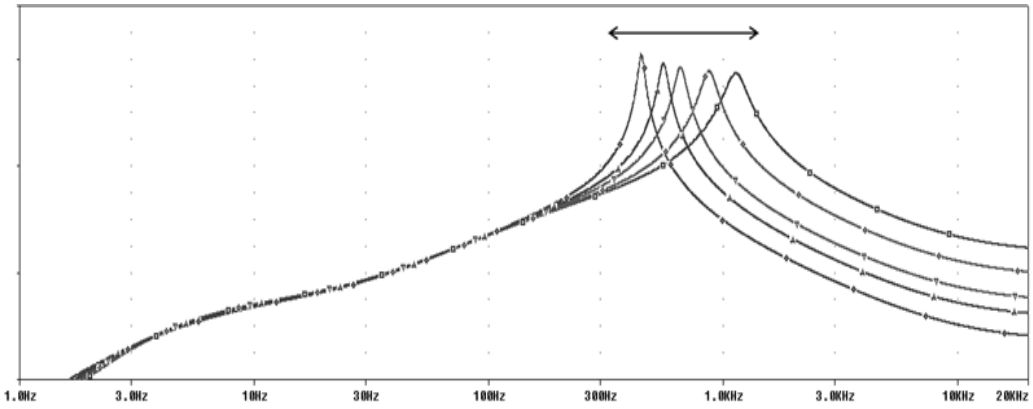

Figure 6: Auto-wah effect using different bandpass filters

Filtering

The *auto-wah* effect requires the output of multiple bandpass filters. First, we used the same method as in lab5 and created 10 lowpass filters with 256 coefficients and multiplied their values by 2^{12}. The coefficients were stored in a manner that would be transformed into ROM by the compiling tools once loaded onto the labkit. The cutoff frequencies of these filters span the frequencies that can be produced by the guitar.

Next, we used the accumulator method described in lab5 to produce the output by applying the filter to the input. Finally, to create the bandpass outputs we just take the outputs of two lowpass filters and output the difference between the outputs of the two filters. This is because the difference between two lowpass filters is a bandpass filter.

The MATLAB command used to obtain the filter coefficients is: round(fir1(255,p)*4096), where p is the decimal number equal to the cutoff frequency divided by the sampling rate. The cutoff frequencies for the lowpass filters used were: 300 Hz, 375 Hz, 450 Hz, 540 Hz, 660 Hz, 800 Hz, 950 Hz, 1100 Hz, 1350 Hz, and 1600 Hz.

One thing we learned is that the effect would sound better if we used bandpass filters instead of subtracting the outputs of two lowpass filters.

Chorus

Chorus pedals produces a wide swelling sound by mixing sounds with small differences in sound quality and pitch. This is achieved by splitting up the audio signal in two, delaying and/or modulating one of the signals, and mixing the signals back together. This produces the sensation of hearing multiple guitars being played even though we are using a single guitar.

This effect was implemented by playing a pitch shifted copy of the input signal with a slight time delay. The delay was implemented the same way as in the “slap” delay effect. We just store the input in a dual-port BRAM and play it back later.

The pitch shift was a harder to introduce. To pitch shift the input we had to resample the input signal. Because of this, we had to change the approach to the chorus effect within the *Effects Controller* since the sampling rate of the *chorus* is controlled by that module. The input to the *chorus* module was sampled at approximately 47 kHz (a bit below the sampling rate). This resulted in a slightly lower tone than the one played.

After producing the time delayed and pitch shifted copy we mix it with the input signal and pass that as the output of the module.

However, resampling the input signal produced an annoying ticking noise on top of the guitar sound. To fix this, we could implement the chorus effect using just the time delay or find a better way to produce the pitch shift. The sampling rate should not be much faster or slower than 48 kHz because it would result in a pitch that is too high or too low.

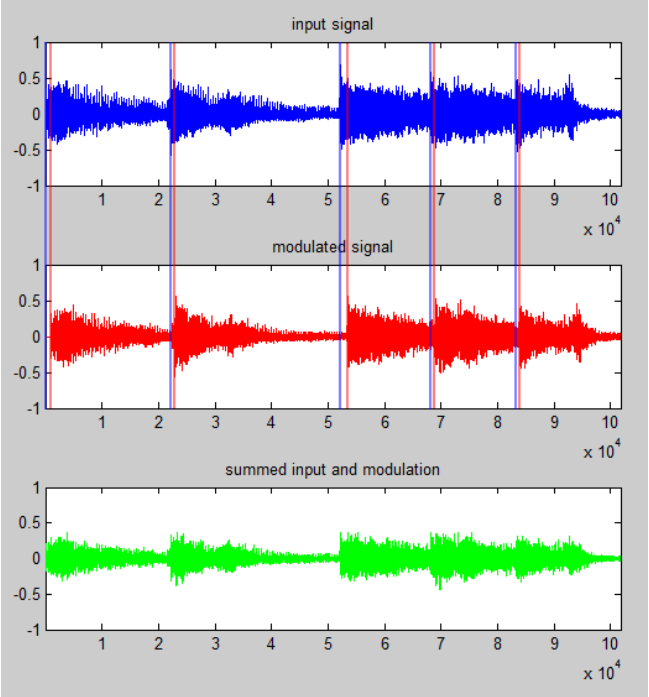

Figure 7: Illustration of the chorus effect

Future Work

One of the ways to improve this project would be to implement some of the more complex guitar effects. The three most popular are *pitch-shifter, reverb,* and *phaser*.

Pitch shifters raise or lower notes played by a set interval by altering the frequency of the audio signal. Simple *pitch shifters* known as *octavers* raise or lower the note played by an octave or two. Complex *pitch shifters* allow a broad range of interval alterations. A *pitch shifter* could be created by resampling the input signal at a different rate.

Reverb pedals simulate the spacious sounds produced in acoustic spaces such as halls and cathedrals. This effect is achieved by creating many gradually decaying echoes. To create this effect using digital logic, digital signal processing could be used to simulate the behavior of multiple feedback delay circuits. This is quite computationally expensive and requires the use of complex digital signal processing algorithms such as GVerb.

A *phaser* splits the audio signal and cyclically alters the phase of one of parts of the signals. After shifting one of the parts' phase, it mixes the two parts of the audio signal back together. The *phaser* pedal can be implemented using a cascaded 2^{nd} order notch filter with a low quality factor.

Furthermore, the existing effects could be refined and more user control could be added. Currently, to change the parameters we have to reprogram the labkit with new parameters. We could give the user more control over all the parameters and connect the *wah* effect to an external pedal so that the user can control the center frequency of the bandpass filter by changing the position of the external pedal.

One thing that we haven’t spent much time on is the graphics module. It performs its function, but it is not visually appealing. It could be improved such that it resembles an actual menu or maybe a guitar pedalboard which allows us to select a pedal and see the change in its parameters as turning knobs.

We could also create a storage module that would allow the user to store his favorite presets and load them whenever he wants to from memory. Additionally, we could change the interface to the AC97 and produce stereophonic sounds.

There are many possible improvements for this specific project. However, most of these improvements are hard to implement while retaining high-quality audio output.

Appendix: Verilog Code

```
module coeffs300(
  input wire [7:0] index,
  output reg signed [11:0] coeff
); // Lowpass filter with cutoff frequency at 300Hz.
  // tools will turn this into a 256x12 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'sd2;
      8'd18: coeff = 12'sd2;
      8'd19: coeff = 12'sd2;
      8'd20: coeff = 12'sd2;
      8'd21: coeff = 12'sd2;
      8'd22: coeff = 12'sd2;
      8'd23: coeff = 12'sd2;
      8'd24: coeff = 12'sd2;
      8'd25: coeff = 12'sd3;
      8'd26: coeff = 12'sd3;
      8'd27: coeff = 12'sd3;
      8'd28: coeff = 12'sd3;
      8'd29: coeff = 12'sd3;
      8'd30: coeff = 12'sd3;
      8'd31: coeff = 12'sd4;
      8'd32:  coeff = 12'sd4;
      8'd33:  coeff = 12'sd4;
      8'd34:  coeff = 12'sd4;
      8'd35:  coeff = 12'sd4;
      8'd36:  coeff = 12'sd5;
      8'd37:  coeff = 12'sd5;
      8'd38:  coeff = 12'sd5;
      8'd39:  coeff = 12'sd5;
      8'd40:  coeff = 12'sd6;
      8'd41:  coeff = 12'sd6;
      8'd42: coeff = 12'sd6;
      8'd43: coeff = 12'sd7;
      8'd44: coeff = 12'sd7;
      8'd45: coeff = 12'sd7;
      8'd46: coeff = 12'sd8;
      8'd47: coeff = 12'sd8;
      8'd48: coeff = 12'sd8;
```

```
8'd49: coeff = 12'sd9;
8'd50: coeff = 12'sd9;
8'd51: coeff = 12'sd9;
8'd52: coeff = 12'sd10;
8'd53: coeff = 12'sd10;
8'd54: coeff = 12'sd10;
8'd55: coeff = 12'sd11;
8'd56: coeff = 12'sd11;
8'd57: coeff = 12'sd11;
8'd58: coeff = 12'sd12;
8'd59: coeff = 12'sd12;
8'd60: coeff = 12'sd13;
8'd61: coeff = 12'sd13;
8'd62: coeff = 12'sd14;
8'd63: coeff = 12'sd14;
8'd64:  coeff = 12'sd14;
8'd65:  coeff = 12'sd15;
8'd66:  coeff = 12'sd15;
8'd67:  coeff = 12'sd16;
8'd68:  coeff = 12'sd16;
8'd69:  coeff = 12'sd17;
8'd70:  coeff = 12'sd17;
8'd71:  coeff = 12'sd18;
8'd72:  coeff = 12'sd18;
8'd73:  coeff = 12'sd18;
8'd74: coeff = 12'sd19;
8'd75: coeff = 12'sd19;
8'd76: coeff = 12'sd20;
8'd77: coeff = 12'sd20;
8'd78: coeff = 12'sd21;
8'd79: coeff = 12'sd21;
8'd80: coeff = 12'sd22;
8'd81: coeff = 12'sd22;
8'd82: coeff = 12'sd23;
8'd83: coeff = 12'sd23;
8'd84: coeff = 12'sd23;
8'd85: coeff = 12'sd24;
8'd86: coeff = 12'sd24;
8'd87: coeff = 12'sd25;
8'd88: coeff = 12'sd25;
8'd89: coeff = 12'sd26;
8'd90: coeff = 12'sd26;
8'd91: coeff = 12'sd26;
8'd92: coeff = 12'sd27;
8'd93: coeff = 12'sd27;
8'd94: coeff = 12'sd28;
8'd95: coeff = 12'sd28;
8'd96:  coeff = 12'sd28;
8'd97:  coeff = 12'sd29;
8'd98:  coeff = 12'sd29;
8'd99:  coeff = 12'sd30;
8'd100:  coeff = 12'sd30;
8'd101:  coeff = 12'sd30;
8'd102:  coeff = 12'sd31;
8'd103:  coeff = 12'sd31;
8'd104:  coeff = 12'sd31;
8'd105:  coeff = 12'sd31;
8'd106: coeff = 12'sd32;
8'd107: coeff = 12'sd32;
8'd108: coeff = 12'sd32;
```

```
8'd109: coeff = 12'sd33;
8'd110: coeff = 12'sd33;
8'd111: coeff = 12'sd33;
8'd112: coeff = 12'sd33;
8'd113: coeff = 12'sd33;
8'd114: coeff = 12'sd34;
8'd115: coeff = 12'sd34;
8'd116: coeff = 12'sd34;
8'd117: coeff = 12'sd34;
8'd118: coeff = 12'sd34;
8'd119: coeff = 12'sd34;
8'd120: coeff = 12'sd35;
8'd121: coeff = 12'sd35;
8'd122: coeff = 12'sd35;
8'd123: coeff = 12'sd35;
8'd124: coeff = 12'sd35;
8'd125: coeff = 12'sd35;
8'd126: coeff = 12'sd35;
8'd127: coeff = 12'sd35;
8'd128:  coeff = 12'sd35;
8'd129:  coeff = 12'sd35;
8'd130:  coeff = 12'sd35;
8'd131: coeff = 12'sd35;
8'd132:  coeff = 12'sd35;
8'd133:  coeff = 12'sd35;
8'd134:  coeff = 12'sd35;
8'd135:  coeff = 12'sd35;
8'd136:  coeff = 12'sd34;
8'd137:  coeff = 12'sd34;
8'd138:  coeff = 12'sd34;
8'd139:  coeff = 12'sd34;
8'd140:  coeff = 12'sd34;
8'd141:  coeff = 12'sd34;
8'd142: coeff = 12'sd33;
8'd143: coeff = 12'sd33;
8'd144: coeff = 12'sd33;
8'd145: coeff = 12'sd33;
8'd146: coeff = 12'sd33;
8'd147: coeff = 12'sd32;
8'd148: coeff = 12'sd32;
8'd149: coeff = 12'sd32;
8'd150: coeff = 12'sd31;
8'd151: coeff = 12'sd31;
8'd152: coeff = 12'sd31;
8'd153: coeff = 12'sd31;
8'd154: coeff = 12'sd30;
8'd155: coeff = 12'sd30;
8'd156: coeff = 12'sd30;
8'd157: coeff = 12'sd29;
8'd158: coeff = 12'sd29;
8'd159: coeff = 12'sd28;
8'd160: coeff = 12'sd28;
8'd161: coeff = 12'sd28;
8'd162: coeff = 12'sd27;
8'd163: coeff = 12'sd27;
8'd164:  coeff = 12'sd26;
8'd165:  coeff = 12'sd26;
8'd166:  coeff = 12'sd26;
8'd167:  coeff = 12'sd25;
8'd168:  coeff = 12'sd25;
```

```
8'd169:  coeff = 12'sd24;
8'd170:  coeff = 12'sd24;
8'd171:  coeff = 12'sd23;
8'd172:  coeff = 12'sd23;
8'd173:  coeff = 12'sd23;
8'd174: coeff = 12'sd22;
8'd175: coeff = 12'sd22;
8'd176: coeff = 12'sd21;
8'd177: coeff = 12'sd21;
8'd178: coeff = 12'sd20;
8'd179: coeff = 12'sd20;
8'd180: coeff = 12'sd19;
8'd181: coeff = 12'sd19;
8'd182: coeff = 12'sd18;
8'd183: coeff = 12'sd18;
8'd184: coeff = 12'sd18;
8'd185: coeff = 12'sd17;
8'd186: coeff = 12'sd17;
8'd187: coeff = 12'sd16;
8'd188: coeff = 12'sd16;
8'd189: coeff = 12'sd15;
8'd190: coeff = 12'sd15;
8'd191: coeff = 12'sd14;
8'd192: coeff = 12'sd14;
8'd193: coeff = 12'sd14;
8'd194: coeff = 12'sd13;
8'd195: coeff = 12'sd13;
8'd196:  coeff = 12'sd12;
8'd197:  coeff = 12'sd12;
8'd198:  coeff = 12'sd11;
8'd199:  coeff = 12'sd11;
8'd200:  coeff = 12'sd11;
8'd201:  coeff = 12'sd10;
8'd202:  coeff = 12'sd10;
8'd203:  coeff = 12'sd10;
8'd204:  coeff = 12'sd9;
8'd205:  coeff = 12'sd9;
8'd206: coeff = 12'sd9;
8'd207: coeff = 12'sd8;
8'd208: coeff = 12'sd8;
8'd209: coeff = 12'sd8;
8'd210: coeff = 12'sd7;
8'd211: coeff = 12'sd7;
8'd212: coeff = 12'sd7;
8'd213: coeff = 12'sd6;
8'd214: coeff = 12'sd6;
8'd215: coeff = 12'sd6;
8'd216: coeff = 12'sd5;
8'd217: coeff = 12'sd5;
8'd218: coeff = 12'sd5;
8'd219: coeff = 12'sd5;
8'd220: coeff = 12'sd4;
8'd221: coeff = 12'sd4;
8'd222: coeff = 12'sd4;
8'd223: coeff = 12'sd4;
8'd224: coeff = 12'sd4;
8'd225: coeff = 12'sd3;
8'd226: coeff = 12'sd3;
8'd227: coeff = 12'sd3;
8'd228:  coeff = 12'sd3;
```

```
      8'd229:  coeff = 12'sd3;
      8'd230:  coeff = 12'sd3;
      8'd231: coeff = 12'sd2;
      8'd232:  coeff = 12'sd2;
      8'd233:  coeff = 12'sd2;
      8'd234:  coeff = 12'sd2;
      8'd235:  coeff = 12'sd2;
      8'd236:  coeff = 12'sd2;
      8'd237:  coeff = 12'sd2;
      8'd238:  coeff = 12'sd2;
      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;
      8'd251: coeff = 12'sd1;
      8'd252: coeff = 12'sd1;
      8'd253: coeff = 12'sd1;
      8'd254: coeff = 12'sd1;
      8'd255: coeff = 12'sd1;
      default: coeff = 12'hXXX;
    endcase
endmodule

module coeffs375(
  input wire [7:0] index,
  output reg signed [11:0] coeff
); // Lowpass filter with cutoff frequency at 375Hz.
  // tools will turn this into a 256x12 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'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'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'sd3;
8'd33:  coeff = 12'sd3;
8'd34:  coeff = 12'sd3;
8'd35:  coeff = 12'sd3;
8'd36:  coeff = 12'sd3;
8'd37:  coeff = 12'sd4;
8'd38:  coeff = 12'sd4;
8'd39:  coeff = 12'sd4;
8'd40:  coeff = 12'sd4;
8'd41:  coeff = 12'sd4;
8'd42: coeff = 12'sd5;
8'd43: coeff = 12'sd5;
8'd44: coeff = 12'sd5;
8'd45: coeff = 12'sd6;
8'd46: coeff = 12'sd6;
8'd47: coeff = 12'sd6;
8'd48: coeff = 12'sd7;
8'd49: coeff = 12'sd7;
8'd50: coeff = 12'sd7;
8'd51: coeff = 12'sd8;
8'd52: coeff = 12'sd8;
8'd53: coeff = 12'sd8;
8'd54: coeff = 12'sd9;
8'd55: coeff = 12'sd9;
8'd56: coeff = 12'sd10;
8'd57: coeff = 12'sd10;
8'd58: coeff = 12'sd10;
8'd59: coeff = 12'sd11;
8'd60: coeff = 12'sd11;
8'd61: coeff = 12'sd12;
8'd62: coeff = 12'sd12;
8'd63: coeff = 12'sd13;
8'd64:  coeff = 12'sd13;
8'd65:  coeff = 12'sd14;
8'd66:  coeff = 12'sd14;
8'd67:  coeff = 12'sd15;
8'd68:  coeff = 12'sd15;
8'd69:  coeff = 12'sd16;
8'd70:  coeff = 12'sd16;
8'd71:  coeff = 12'sd17;
8'd72:  coeff = 12'sd17;
8'd73:  coeff = 12'sd18;
8'd74: coeff = 12'sd18;
8'd75: coeff = 12'sd19;
8'd76: coeff = 12'sd19;
8'd77: coeff = 12'sd20;
8'd78: coeff = 12'sd20;
8'd79: coeff = 12'sd21;
8'd80: coeff = 12'sd22;
8'd81: coeff = 12'sd22;
```

```
8'd82: coeff = 12'sd23;
8'd83: coeff = 12'sd23;
8'd84: coeff = 12'sd24;
8'd85: coeff = 12'sd24;
8'd86: coeff = 12'sd25;
8'd87: coeff = 12'sd25;
8'd88: coeff = 12'sd26;
8'd89: coeff = 12'sd26;
8'd90: coeff = 12'sd27;
8'd91: coeff = 12'sd27;
8'd92: coeff = 12'sd28;
8'd93: coeff = 12'sd28;
8'd94: coeff = 12'sd29;
8'd95: coeff = 12'sd29;
8'd96:  coeff = 12'sd30;
8'd97:  coeff = 12'sd30;
8'd98:  coeff = 12'sd31;
8'd99:  coeff = 12'sd31;
8'd100:  coeff = 12'sd32;
8'd101:  coeff = 12'sd32;
8'd102:  coeff = 12'sd32;
8'd103:  coeff = 12'sd33;
8'd104:  coeff = 12'sd33;
8'd105:  coeff = 12'sd34;
8'd106: coeff = 12'sd34;
8'd107: coeff = 12'sd34;
8'd108: coeff = 12'sd35;
8'd109: coeff = 12'sd35;
8'd110: coeff = 12'sd35;
8'd111: coeff = 12'sd36;
8'd112: coeff = 12'sd36;
8'd113: coeff = 12'sd36;
8'd114: coeff = 12'sd36;
8'd115: coeff = 12'sd37;
8'd116: coeff = 12'sd37;
8'd117: coeff = 12'sd37;
8'd118: coeff = 12'sd37;
8'd119: coeff = 12'sd37;
8'd120: coeff = 12'sd37;
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'sd38;
8'd132:  coeff = 12'sd38;
8'd133:  coeff = 12'sd38;
8'd134:  coeff = 12'sd38;
8'd135:  coeff = 12'sd37;
8'd136:  coeff = 12'sd37;
8'd137:  coeff = 12'sd37;
8'd138:  coeff = 12'sd37;
8'd139:  coeff = 12'sd37;
8'd140:  coeff = 12'sd37;
8'd141:  coeff = 12'sd36;
```

```
8'd142: coeff = 12'sd36;
8'd143: coeff = 12'sd36;
8'd144: coeff = 12'sd36;
8'd145: coeff = 12'sd35;
8'd146: coeff = 12'sd35;
8'd147: coeff = 12'sd35;
8'd148: coeff = 12'sd34;
8'd149: coeff = 12'sd34;
8'd150: coeff = 12'sd34;
8'd151: coeff = 12'sd33;
8'd152: coeff = 12'sd33;
8'd153: coeff = 12'sd32;
8'd154: coeff = 12'sd32;
8'd155: coeff = 12'sd32;
8'd156: coeff = 12'sd31;
8'd157: coeff = 12'sd31;
8'd158: coeff = 12'sd30;
8'd159: coeff = 12'sd30;
8'd160: coeff = 12'sd29;
8'd161: coeff = 12'sd29;
8'd162: coeff = 12'sd28;
8'd163: coeff = 12'sd28;
8'd164:  coeff = 12'sd27;
8'd165:  coeff = 12'sd27;
8'd166:  coeff = 12'sd26;
8'd167:  coeff = 12'sd26;
8'd168:  coeff = 12'sd25;
8'd169:  coeff = 12'sd25;
8'd170:  coeff = 12'sd24;
8'd171:  coeff = 12'sd24;
8'd172:  coeff = 12'sd23;
8'd173:  coeff = 12'sd23;
8'd174: coeff = 12'sd22;
8'd175: coeff = 12'sd22;
8'd176: coeff = 12'sd21;
8'd177: coeff = 12'sd20;
8'd178: coeff = 12'sd20;
8'd179: coeff = 12'sd19;
8'd180: coeff = 12'sd19;
8'd181: coeff = 12'sd18;
8'd182: coeff = 12'sd18;
8'd183: coeff = 12'sd17;
8'd184: coeff = 12'sd17;
8'd185: coeff = 12'sd16;
8'd186: coeff = 12'sd16;
8'd187: coeff = 12'sd15;
8'd188: coeff = 12'sd15;
8'd189: coeff = 12'sd14;
8'd190: coeff = 12'sd14;
8'd191: coeff = 12'sd13;
8'd192: coeff = 12'sd13;
8'd193: coeff = 12'sd12;
8'd194: coeff = 12'sd12;
8'd195: coeff = 12'sd11;
8'd196:  coeff = 12'sd11;
8'd197:  coeff = 12'sd10;
8'd198:  coeff = 12'sd10;
8'd199:  coeff = 12'sd10;
8'd200:  coeff = 12'sd9;
8'd201:  coeff = 12'sd9;
```

```
      8'd202:  coeff = 12'sd8;
      8'd203:  coeff = 12'sd8;
      8'd204:  coeff = 12'sd8;
      8'd205:  coeff = 12'sd7;
      8'd206: coeff = 12'sd7;
      8'd207: coeff = 12'sd7;
      8'd208: coeff = 12'sd6;
      8'd209: coeff = 12'sd6;
      8'd210: coeff = 12'sd6;
      8'd211: coeff = 12'sd5;
      8'd212: coeff = 12'sd5;
      8'd213: coeff = 12'sd5;
      8'd214: coeff = 12'sd4;
      8'd215: coeff = 12'sd4;
      8'd216: coeff = 12'sd4;
      8'd217: coeff = 12'sd4;
      8'd218: coeff = 12'sd4;
      8'd219: coeff = 12'sd3;
      8'd220: coeff = 12'sd3;
      8'd221: coeff = 12'sd3;
      8'd222: coeff = 12'sd3;
      8'd223: coeff = 12'sd3;
      8'd224: coeff = 12'sd2;
      8'd225: coeff = 12'sd2;
      8'd226: coeff = 12'sd2;
      8'd227: coeff = 12'sd2;
      8'd228:  coeff = 12'sd2;
      8'd229:  coeff = 12'sd2;
      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;
      8'd251: coeff = 12'sd0;
      8'd252: coeff = 12'sd0;
      8'd253: coeff = 12'sd0;
      8'd254: coeff = 12'sd0;
      8'd255: coeff = 12'sd0;
      default: coeff = 12'hXXX;
    endcase
endmodule
```

```
module coeffs450(
  input wire [7:0] index,
  output reg signed [11:0] coeff
); // Lowpass filter with cutoff frequency at 450Hz.
  // tools will turn this into a 256x12 ROM
  always @(index)
    case (index)
      8'd0:  coeff = -12'sd1;
      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'sd0;
      8'd27: coeff = 12'sd0;
      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'sd3;
      8'd42: coeff = 12'sd3;
      8'd43: coeff = 12'sd3;
      8'd44: coeff = 12'sd3;
      8'd45: coeff = 12'sd4;
      8'd46: coeff = 12'sd4;
      8'd47: coeff = 12'sd4;
      8'd48: coeff = 12'sd5;
      8'd49: coeff = 12'sd5;
      8'd50: coeff = 12'sd5;
      8'd51: coeff = 12'sd6;
      8'd52: coeff = 12'sd6;
```

```
8'd53: coeff = 12'sd6;
8'd54: coeff = 12'sd7;
8'd55: coeff = 12'sd7;
8'd56: coeff = 12'sd8;
8'd57: coeff = 12'sd8;
8'd58: coeff = 12'sd9;
8'd59: coeff = 12'sd9;
8'd60: coeff = 12'sd10;
8'd61: coeff = 12'sd10;
8'd62: coeff = 12'sd11;
8'd63: coeff = 12'sd11;
8'd64:  coeff = 12'sd12;
8'd65:  coeff = 12'sd12;
8'd66:  coeff = 12'sd13;
8'd67:  coeff = 12'sd13;
8'd68:  coeff = 12'sd14;
8'd69:  coeff = 12'sd14;
8'd70:  coeff = 12'sd15;
8'd71:  coeff = 12'sd16;
8'd72:  coeff = 12'sd16;
8'd73:  coeff = 12'sd17;
8'd74: coeff = 12'sd17;
8'd75: coeff = 12'sd18;
8'd76: coeff = 12'sd19;
8'd77: coeff = 12'sd19;
8'd78: coeff = 12'sd20;
8'd79: coeff = 12'sd21;
8'd80: coeff = 12'sd21;
8'd81: coeff = 12'sd22;
8'd82: coeff = 12'sd22;
8'd83: coeff = 12'sd23;
8'd84: coeff = 12'sd24;
8'd85: coeff = 12'sd24;
8'd86: coeff = 12'sd25;
8'd87: coeff = 12'sd26;
8'd88: coeff = 12'sd26;
8'd89: coeff = 12'sd27;
8'd90: coeff = 12'sd28;
8'd91: coeff = 12'sd28;
8'd92: coeff = 12'sd29;
8'd93: coeff = 12'sd30;
8'd94: coeff = 12'sd30;
8'd95: coeff = 12'sd31;
8'd96:  coeff = 12'sd31;
8'd97:  coeff = 12'sd32;
8'd98:  coeff = 12'sd32;
8'd99:  coeff = 12'sd33;
8'd100:  coeff = 12'sd34;
8'd101:  coeff = 12'sd34;
8'd102:  coeff = 12'sd35;
8'd103:  coeff = 12'sd35;
8'd104:  coeff = 12'sd36;
8'd105:  coeff = 12'sd36;
8'd106: coeff = 12'sd37;
8'd107: coeff = 12'sd37;
8'd108: coeff = 12'sd37;
8'd109: coeff = 12'sd38;
8'd110: coeff = 12'sd38;
8'd111: coeff = 12'sd39;
8'd112: coeff = 12'sd39;
```

```
8'd113: coeff = 12'sd39;
8'd114: coeff = 12'sd40;
8'd115: coeff = 12'sd40;
8'd116: coeff = 12'sd40;
8'd117: coeff = 12'sd41;
8'd118: coeff = 12'sd41;
8'd119: coeff = 12'sd41;
8'd120: coeff = 12'sd41;
8'd121: coeff = 12'sd41;
8'd122: coeff = 12'sd41;
8'd123: coeff = 12'sd42;
8'd124: coeff = 12'sd42;
8'd125: coeff = 12'sd42;
8'd126: coeff = 12'sd42;
8'd127: coeff = 12'sd42;
8'd128:  coeff = 12'sd42;
8'd129:  coeff = 12'sd42;
8'd130:  coeff = 12'sd42;
8'd131: coeff = 12'sd42;
8'd132:  coeff = 12'sd42;
8'd133:  coeff = 12'sd41;
8'd134:  coeff = 12'sd41;
8'd135:  coeff = 12'sd41;
8'd136:  coeff = 12'sd41;
8'd137:  coeff = 12'sd41;
8'd138:  coeff = 12'sd41;
8'd139:  coeff = 12'sd40;
8'd140:  coeff = 12'sd40;
8'd141:  coeff = 12'sd40;
8'd142: coeff = 12'sd39;
8'd143: coeff = 12'sd39;
8'd144: coeff = 12'sd39;
8'd145: coeff = 12'sd38;
8'd146: coeff = 12'sd38;
8'd147: coeff = 12'sd37;
8'd148: coeff = 12'sd37;
8'd149: coeff = 12'sd37;
8'd150: coeff = 12'sd36;
8'd151: coeff = 12'sd36;
8'd152: coeff = 12'sd35;
8'd153: coeff = 12'sd35;
8'd154: coeff = 12'sd34;
8'd155: coeff = 12'sd34;
8'd156: coeff = 12'sd33;
8'd157: coeff = 12'sd32;
8'd158: coeff = 12'sd32;
8'd159: coeff = 12'sd31;
8'd160: coeff = 12'sd31;
8'd161: coeff = 12'sd30;
8'd162: coeff = 12'sd30;
8'd163: coeff = 12'sd29;
8'd164:  coeff = 12'sd28;
8'd165:  coeff = 12'sd28;
8'd166:  coeff = 12'sd27;
8'd167:  coeff = 12'sd26;
8'd168:  coeff = 12'sd26;
8'd169:  coeff = 12'sd25;
8'd170:  coeff = 12'sd24;
8'd171:  coeff = 12'sd24;
8'd172:  coeff = 12'sd23;
```

```
8'd173:  coeff = 12'sd22;
8'd174: coeff = 12'sd22;
8'd175: coeff = 12'sd21;
8'd176: coeff = 12'sd21;
8'd177: coeff = 12'sd20;
8'd178: coeff = 12'sd19;
8'd179: coeff = 12'sd19;
8'd180: coeff = 12'sd18;
8'd181: coeff = 12'sd17;
8'd182: coeff = 12'sd17;
8'd183: coeff = 12'sd16;
8'd184: coeff = 12'sd16;
8'd185: coeff = 12'sd15;
8'd186: coeff = 12'sd14;
8'd187: coeff = 12'sd14;
8'd188: coeff = 12'sd13;
8'd189: coeff = 12'sd13;
8'd190: coeff = 12'sd12;
8'd191: coeff = 12'sd12;
8'd192: coeff = 12'sd11;
8'd193: coeff = 12'sd11;
8'd194: coeff = 12'sd10;
8'd195: coeff = 12'sd10;
8'd196:  coeff = 12'sd9;
8'd197:  coeff = 12'sd9;
8'd198:  coeff = 12'sd8;
8'd199:  coeff = 12'sd8;
8'd200:  coeff = 12'sd7;
8'd201:  coeff = 12'sd7;
8'd202:  coeff = 12'sd6;
8'd203:  coeff = 12'sd6;
8'd204:  coeff = 12'sd6;
8'd205:  coeff = 12'sd5;
8'd206: coeff = 12'sd5;
8'd207: coeff = 12'sd5;
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'sd2;
8'd216: coeff = 12'sd2;
8'd217: coeff = 12'sd2;
8'd218: coeff = 12'sd2;
8'd219: coeff = 12'sd2;
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'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;
      8'd251: coeff = 12'sd0;
      8'd252: coeff = 12'sd0;
      8'd253: coeff = 12'sd0;
      8'd254: coeff = 12'sd0;
      8'd255: coeff = -12'sd1;
      default: coeff = 12'hXXX;
    endcase
endmodule


module coeffs540(
  input wire [7:0] index,
  output reg signed [11:0] coeff
); // Lowpass filter with cutoff frequency at 540Hz.
  // tools will turn this into a 256x12 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'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'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'sd1;
8'd43: coeff = 12'sd1;
8'd44: coeff = 12'sd1;
8'd45: coeff = 12'sd1;
8'd46: coeff = 12'sd1;
8'd47: coeff = 12'sd2;
8'd48: coeff = 12'sd2;
8'd49: coeff = 12'sd2;
8'd50: coeff = 12'sd3;
8'd51: coeff = 12'sd3;
8'd52: coeff = 12'sd3;
8'd53: coeff = 12'sd4;
8'd54: coeff = 12'sd4;
8'd55: coeff = 12'sd4;
8'd56: coeff = 12'sd5;
8'd57: coeff = 12'sd5;
8'd58: coeff = 12'sd6;
8'd59: coeff = 12'sd6;
8'd60: coeff = 12'sd7;
8'd61: coeff = 12'sd7;
8'd62: coeff = 12'sd8;
8'd63: coeff = 12'sd8;
8'd64:  coeff = 12'sd9;
8'd65:  coeff = 12'sd10;
8'd66:  coeff = 12'sd10;
8'd67:  coeff = 12'sd11;
8'd68:  coeff = 12'sd11;
8'd69:  coeff = 12'sd12;
8'd70:  coeff = 12'sd13;
8'd71:  coeff = 12'sd13;
8'd72:  coeff = 12'sd14;
8'd73:  coeff = 12'sd15;
8'd74: coeff = 12'sd16;
8'd75: coeff = 12'sd16;
8'd76: coeff = 12'sd17;
8'd77: coeff = 12'sd18;
8'd78: coeff = 12'sd19;
8'd79: coeff = 12'sd19;
8'd80: coeff = 12'sd20;
8'd81: coeff = 12'sd21;
8'd82: coeff = 12'sd22;
8'd83: coeff = 12'sd23;
8'd84: coeff = 12'sd23;
```

```
8'd85: coeff = 12'sd24;
8'd86: coeff = 12'sd25;
8'd87: coeff = 12'sd26;
8'd88: coeff = 12'sd27;
8'd89: coeff = 12'sd28;
8'd90: coeff = 12'sd28;
8'd91: coeff = 12'sd29;
8'd92: coeff = 12'sd30;
8'd93: coeff = 12'sd31;
8'd94: coeff = 12'sd32;
8'd95: coeff = 12'sd32;
8'd96:  coeff = 12'sd33;
8'd97:  coeff = 12'sd34;
8'd98:  coeff = 12'sd35;
8'd99:  coeff = 12'sd35;
8'd100:  coeff = 12'sd36;
8'd101:  coeff = 12'sd37;
8'd102:  coeff = 12'sd38;
8'd103:  coeff = 12'sd38;
8'd104:  coeff = 12'sd39;
8'd105:  coeff = 12'sd40;
8'd106: coeff = 12'sd40;
8'd107: coeff = 12'sd41;
8'd108: coeff = 12'sd41;
8'd109: coeff = 12'sd42;
8'd110: coeff = 12'sd43;
8'd111: coeff = 12'sd43;
8'd112: coeff = 12'sd44;
8'd113: coeff = 12'sd44;
8'd114: coeff = 12'sd44;
8'd115: coeff = 12'sd45;
8'd116: coeff = 12'sd45;
8'd117: coeff = 12'sd46;
8'd118: coeff = 12'sd46;
8'd119: coeff = 12'sd46;
8'd120: coeff = 12'sd46;
8'd121: coeff = 12'sd47;
8'd122: coeff = 12'sd47;
8'd123: coeff = 12'sd47;
8'd124: coeff = 12'sd47;
8'd125: coeff = 12'sd47;
8'd126: coeff = 12'sd47;
8'd127: coeff = 12'sd47;
8'd128:  coeff = 12'sd47;
8'd129:  coeff = 12'sd47;
8'd130:  coeff = 12'sd47;
8'd131: coeff = 12'sd47;
8'd132:  coeff = 12'sd47;
8'd133:  coeff = 12'sd47;
8'd134:  coeff = 12'sd47;
8'd135:  coeff = 12'sd46;
8'd136:  coeff = 12'sd46;
8'd137:  coeff = 12'sd46;
8'd138:  coeff = 12'sd46;
8'd139:  coeff = 12'sd45;
8'd140:  coeff = 12'sd45;
8'd141:  coeff = 12'sd44;
8'd142: coeff = 12'sd44;
8'd143: coeff = 12'sd44;
8'd144: coeff = 12'sd43;
```

```
8'd145: coeff = 12'sd43;
8'd146: coeff = 12'sd42;
8'd147: coeff = 12'sd41;
8'd148: coeff = 12'sd41;
8'd149: coeff = 12'sd40;
8'd150: coeff = 12'sd40;
8'd151: coeff = 12'sd39;
8'd152: coeff = 12'sd38;
8'd153: coeff = 12'sd38;
8'd154: coeff = 12'sd37;
8'd155: coeff = 12'sd36;
8'd156: coeff = 12'sd35;
8'd157: coeff = 12'sd35;
8'd158: coeff = 12'sd34;
8'd159: coeff = 12'sd33;
8'd160: coeff = 12'sd32;
8'd161: coeff = 12'sd32;
8'd162: coeff = 12'sd31;
8'd163: coeff = 12'sd30;
8'd164:  coeff = 12'sd29;
8'd165:  coeff = 12'sd28;
8'd166:  coeff = 12'sd28;
8'd167:  coeff = 12'sd27;
8'd168:  coeff = 12'sd26;
8'd169:  coeff = 12'sd25;
8'd170:  coeff = 12'sd24;
8'd171:  coeff = 12'sd23;
8'd172:  coeff = 12'sd23;
8'd173:  coeff = 12'sd22;
8'd174: coeff = 12'sd21;
8'd175: coeff = 12'sd20;
8'd176: coeff = 12'sd19;
8'd177: coeff = 12'sd19;
8'd178: coeff = 12'sd18;
8'd179: coeff = 12'sd17;
8'd180: coeff = 12'sd16;
8'd181: coeff = 12'sd16;
8'd182: coeff = 12'sd15;
8'd183: coeff = 12'sd14;
8'd184: coeff = 12'sd13;
8'd185: coeff = 12'sd13;
8'd186: coeff = 12'sd12;
8'd187: coeff = 12'sd11;
8'd188: coeff = 12'sd11;
8'd189: coeff = 12'sd10;
8'd190: coeff = 12'sd10;
8'd191: coeff = 12'sd9;
8'd192: coeff = 12'sd8;
8'd193: coeff = 12'sd8;
8'd194: coeff = 12'sd7;
8'd195: coeff = 12'sd7;
8'd196:  coeff = 12'sd6;
8'd197:  coeff = 12'sd6;
8'd198:  coeff = 12'sd5;
8'd199:  coeff = 12'sd5;
8'd200:  coeff = 12'sd4;
8'd201:  coeff = 12'sd4;
8'd202:  coeff = 12'sd4;
8'd203:  coeff = 12'sd3;
8'd204:  coeff = 12'sd3;
```

```
      8'd205:  coeff = 12'sd3;
      8'd206: coeff = 12'sd2;
      8'd207: coeff = 12'sd2;
      8'd208: coeff = 12'sd2;
      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'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'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;
      8'd251: coeff = -12'sd1;
      8'd252: coeff = -12'sd1;
      8'd253: coeff = -12'sd1;
      8'd254: coeff = -12'sd1;
      8'd255: coeff = -12'sd1;
      default: coeff = 12'hXXX;
    endcase
endmodule


module coeffs660(
  input wire [7:0] index,
  output reg signed [11:0] coeff
); // Lowpass filter with cutoff frequency at 660Hz.
```

```
// tools will turn this into a 256x12 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'sd2;
    8'd20: coeff = -12'sd2;
    8'd21: coeff = -12'sd2;
    8'd22: coeff = -12'sd2;
    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'sd2;
    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'sd3;
    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'sd1;
    8'd51: coeff = -12'sd1;
    8'd52: coeff = -12'sd1;
    8'd53: coeff = -12'sd1;
    8'd54: coeff = 12'sd0;
    8'd55: coeff = 12'sd0;
    8'd56: coeff = 12'sd0;
```

```
8'd57: coeff = 12'sd1;
8'd58: coeff = 12'sd1;
8'd59: coeff = 12'sd2;
8'd60: coeff = 12'sd2;
8'd61: coeff = 12'sd3;
8'd62: coeff = 12'sd3;
8'd63: coeff = 12'sd4;
8'd64:  coeff = 12'sd4;
8'd65:  coeff = 12'sd5;
8'd66:  coeff = 12'sd6;
8'd67:  coeff = 12'sd6;
8'd68:  coeff = 12'sd7;
8'd69:  coeff = 12'sd8;
8'd70:  coeff = 12'sd8;
8'd71:  coeff = 12'sd9;
8'd72:  coeff = 12'sd10;
8'd73:  coeff = 12'sd11;
8'd74: coeff = 12'sd12;
8'd75: coeff = 12'sd13;
8'd76: coeff = 12'sd14;
8'd77: coeff = 12'sd15;
8'd78: coeff = 12'sd16;
8'd79: coeff = 12'sd16;
8'd80: coeff = 12'sd18;
8'd81: coeff = 12'sd19;
8'd82: coeff = 12'sd20;
8'd83: coeff = 12'sd21;
8'd84: coeff = 12'sd22;
8'd85: coeff = 12'sd23;
8'd86: coeff = 12'sd24;
8'd87: coeff = 12'sd25;
8'd88: coeff = 12'sd26;
8'd89: coeff = 12'sd27;
8'd90: coeff = 12'sd28;
8'd91: coeff = 12'sd30;
8'd92: coeff = 12'sd31;
8'd93: coeff = 12'sd32;
8'd94: coeff = 12'sd33;
8'd95: coeff = 12'sd34;
8'd96:  coeff = 12'sd35;
8'd97:  coeff = 12'sd36;
8'd98:  coeff = 12'sd37;
8'd99:  coeff = 12'sd38;
8'd100:  coeff = 12'sd40;
8'd101:  coeff = 12'sd41;
8'd102:  coeff = 12'sd42;
8'd103:  coeff = 12'sd43;
8'd104:  coeff = 12'sd44;
8'd105:  coeff = 12'sd45;
8'd106: coeff = 12'sd45;
8'd107: coeff = 12'sd46;
8'd108: coeff = 12'sd47;
8'd109: coeff = 12'sd48;
8'd110: coeff = 12'sd49;
8'd111: coeff = 12'sd50;
8'd112: coeff = 12'sd50;
8'd113: coeff = 12'sd51;
8'd114: coeff = 12'sd52;
8'd115: coeff = 12'sd52;
8'd116: coeff = 12'sd53;
```

```
8'd117: coeff = 12'sd54;
8'd118: coeff = 12'sd54;
8'd119: coeff = 12'sd55;
8'd120: coeff = 12'sd55;
8'd121: coeff = 12'sd55;
8'd122: coeff = 12'sd56;
8'd123: coeff = 12'sd56;
8'd124: coeff = 12'sd56;
8'd125: coeff = 12'sd56;
8'd126: coeff = 12'sd56;
8'd127: coeff = 12'sd56;
8'd128:  coeff = 12'sd56;
8'd129:  coeff = 12'sd56;
8'd130:  coeff = 12'sd56;
8'd131: coeff = 12'sd56;
8'd132:  coeff = 12'sd56;
8'd133:  coeff = 12'sd56;
8'd134:  coeff = 12'sd55;
8'd135:  coeff = 12'sd55;
8'd136:  coeff = 12'sd55;
8'd137:  coeff = 12'sd54;
8'd138:  coeff = 12'sd54;
8'd139:  coeff = 12'sd53;
8'd140:  coeff = 12'sd52;
8'd141:  coeff = 12'sd52;
8'd142: coeff = 12'sd51;
8'd143: coeff = 12'sd50;
8'd144: coeff = 12'sd50;
8'd145: coeff = 12'sd49;
8'd146: coeff = 12'sd48;
8'd147: coeff = 12'sd47;
8'd148: coeff = 12'sd46;
8'd149: coeff = 12'sd45;
8'd150: coeff = 12'sd45;
8'd151: coeff = 12'sd44;
8'd152: coeff = 12'sd43;
8'd153: coeff = 12'sd42;
8'd154: coeff = 12'sd41;
8'd155: coeff = 12'sd40;
8'd156: coeff = 12'sd38;
8'd157: coeff = 12'sd37;
8'd158: coeff = 12'sd36;
8'd159: coeff = 12'sd35;
8'd160: coeff = 12'sd34;
8'd161: coeff = 12'sd33;
8'd162: coeff = 12'sd32;
8'd163: coeff = 12'sd31;
8'd164:  coeff = 12'sd30;
8'd165:  coeff = 12'sd28;
8'd166:  coeff = 12'sd27;
8'd167:  coeff = 12'sd26;
8'd168:  coeff = 12'sd25;
8'd169:  coeff = 12'sd24;
8'd170:  coeff = 12'sd23;
8'd171:  coeff = 12'sd22;
8'd172:  coeff = 12'sd21;
8'd173:  coeff = 12'sd20;
8'd174: coeff = 12'sd19;
8'd175: coeff = 12'sd18;
8'd176: coeff = 12'sd16;
```

```
8'd177: coeff = 12'sd16;
8'd178: coeff = 12'sd15;
8'd179: coeff = 12'sd14;
8'd180: coeff = 12'sd13;
8'd181: coeff = 12'sd12;
8'd182: coeff = 12'sd11;
8'd183: coeff = 12'sd10;
8'd184: coeff = 12'sd9;
8'd185: coeff = 12'sd8;
8'd186: coeff = 12'sd8;
8'd187: coeff = 12'sd7;
8'd188: coeff = 12'sd6;
8'd189: coeff = 12'sd6;
8'd190: coeff = 12'sd5;
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'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'sd1;
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'sd3;
8'd216: coeff = -12'sd3;
8'd217: coeff = -12'sd3;
8'd218: coeff = -12'sd3;
8'd219: coeff = -12'sd3;
8'd220: coeff = -12'sd3;
8'd221: coeff = -12'sd3;
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'sd2;
8'd229:  coeff = -12'sd2;
8'd230:  coeff = -12'sd2;
8'd231: coeff = -12'sd2;
8'd232:  coeff = -12'sd2;
8'd233:  coeff = -12'sd2;
8'd234:  coeff = -12'sd2;
8'd235:  coeff = -12'sd2;
8'd236:  coeff = -12'sd2;
```

```
      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;
      8'd251: coeff = -12'sd1;
      8'd252: coeff = -12'sd1;
      8'd253: coeff = -12'sd1;
      8'd254: coeff = -12'sd1;
      8'd255: coeff = -12'sd1;
      default: coeff = 12'hXXX;
    endcase
endmodule


module coeffs800(
  input wire [7:0] index,
  output reg signed [11:0] coeff
); // Lowpass filter with cutoff frequency at 800Hz.
  // tools will turn this into a 256x12 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'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'sd3;
8'd32:  coeff = -12'sd3;
8'd33:  coeff = -12'sd3;
8'd34:  coeff = -12'sd3;
8'd35:  coeff = -12'sd3;
8'd36:  coeff = -12'sd4;
8'd37:  coeff = -12'sd4;
8'd38:  coeff = -12'sd4;
8'd39:  coeff = -12'sd4;
8'd40:  coeff = -12'sd4;
8'd41:  coeff = -12'sd4;
8'd42: coeff = -12'sd5;
8'd43: coeff = -12'sd5;
8'd44: coeff = -12'sd5;
8'd45: coeff = -12'sd5;
8'd46: coeff = -12'sd5;
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'sd5;
8'd53: coeff = -12'sd5;
8'd54: coeff = -12'sd5;
8'd55: coeff = -12'sd5;
8'd56: coeff = -12'sd5;
8'd57: coeff = -12'sd4;
8'd58: coeff = -12'sd4;
8'd59: coeff = -12'sd4;
8'd60: coeff = -12'sd4;
8'd61: coeff = -12'sd3;
8'd62: coeff = -12'sd3;
8'd63: coeff = -12'sd2;
8'd64:  coeff = -12'sd2;
8'd65:  coeff = -12'sd2;
8'd66:  coeff = -12'sd1;
8'd67:  coeff = 12'sd0;
8'd68:  coeff = 12'sd0;
8'd69:  coeff = 12'sd1;
8'd70:  coeff = 12'sd2;
8'd71:  coeff = 12'sd3;
8'd72:  coeff = 12'sd3;
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'sd10;
8'd79: coeff = 12'sd11;
8'd80: coeff = 12'sd12;
8'd81: coeff = 12'sd13;
8'd82: coeff = 12'sd15;
8'd83: coeff = 12'sd16;
8'd84: coeff = 12'sd17;
8'd85: coeff = 12'sd19;
8'd86: coeff = 12'sd20;
8'd87: coeff = 12'sd22;
8'd88: coeff = 12'sd23;
```

```
8'd89: coeff = 12'sd25;
8'd90: coeff = 12'sd26;
8'd91: coeff = 12'sd28;
8'd92: coeff = 12'sd29;
8'd93: coeff = 12'sd31;
8'd94: coeff = 12'sd32;
8'd95: coeff = 12'sd34;
8'd96:  coeff = 12'sd36;
8'd97:  coeff = 12'sd37;
8'd98:  coeff = 12'sd39;
8'd99:  coeff = 12'sd40;
8'd100:  coeff = 12'sd42;
8'd101:  coeff = 12'sd44;
8'd102:  coeff = 12'sd45;
8'd103:  coeff = 12'sd47;
8'd104:  coeff = 12'sd48;
8'd105:  coeff = 12'sd50;
8'd106: coeff = 12'sd51;
8'd107: coeff = 12'sd52;
8'd108: coeff = 12'sd54;
8'd109: coeff = 12'sd55;
8'd110: coeff = 12'sd56;
8'd111: coeff = 12'sd58;
8'd112: coeff = 12'sd59;
8'd113: coeff = 12'sd60;
8'd114: coeff = 12'sd61;
8'd115: coeff = 12'sd62;
8'd116: coeff = 12'sd63;
8'd117: coeff = 12'sd64;
8'd118: coeff = 12'sd64;
8'd119: coeff = 12'sd65;
8'd120: coeff = 12'sd66;
8'd121: coeff = 12'sd66;
8'd122: coeff = 12'sd67;
8'd123: coeff = 12'sd67;
8'd124: coeff = 12'sd68;
8'd125: coeff = 12'sd68;
8'd126: coeff = 12'sd68;
8'd127: coeff = 12'sd68;
8'd128:  coeff = 12'sd68;
8'd129:  coeff = 12'sd68;
8'd130:  coeff = 12'sd68;
8'd131: coeff = 12'sd68;
8'd132:  coeff = 12'sd67;
8'd133:  coeff = 12'sd67;
8'd134:  coeff = 12'sd66;
8'd135:  coeff = 12'sd66;
8'd136:  coeff = 12'sd65;
8'd137:  coeff = 12'sd64;
8'd138:  coeff = 12'sd64;
8'd139:  coeff = 12'sd63;
8'd140:  coeff = 12'sd62;
8'd141:  coeff = 12'sd61;
8'd142: coeff = 12'sd60;
8'd143: coeff = 12'sd59;
8'd144: coeff = 12'sd58;
8'd145: coeff = 12'sd56;
8'd146: coeff = 12'sd55;
8'd147: coeff = 12'sd54;
8'd148: coeff = 12'sd52;
```

```
8'd149: coeff = 12'sd51;
8'd150: coeff = 12'sd50;
8'd151: coeff = 12'sd48;
8'd152: coeff = 12'sd47;
8'd153: coeff = 12'sd45;
8'd154: coeff = 12'sd44;
8'd155: coeff = 12'sd42;
8'd156: coeff = 12'sd40;
8'd157: coeff = 12'sd39;
8'd158: coeff = 12'sd37;
8'd159: coeff = 12'sd36;
8'd160: coeff = 12'sd34;
8'd161: coeff = 12'sd32;
8'd162: coeff = 12'sd31;
8'd163: coeff = 12'sd29;
8'd164:  coeff = 12'sd28;
8'd165:  coeff = 12'sd26;
8'd166:  coeff = 12'sd25;
8'd167:  coeff = 12'sd23;
8'd168:  coeff = 12'sd22;
8'd169:  coeff = 12'sd20;
8'd170:  coeff = 12'sd19;
8'd171:  coeff = 12'sd17;
8'd172:  coeff = 12'sd16;
8'd173:  coeff = 12'sd15;
8'd174: coeff = 12'sd13;
8'd175: coeff = 12'sd12;
8'd176: coeff = 12'sd11;
8'd177: coeff = 12'sd10;
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'sd3;
8'd185: coeff = 12'sd2;
8'd186: coeff = 12'sd1;
8'd187: coeff = 12'sd0;
8'd188: coeff = 12'sd0;
8'd189: coeff = -12'sd1;
8'd190: coeff = -12'sd2;
8'd191: coeff = -12'sd2;
8'd192: coeff = -12'sd2;
8'd193: coeff = -12'sd3;
8'd194: coeff = -12'sd3;
8'd195: coeff = -12'sd4;
8'd196:  coeff = -12'sd4;
8'd197:  coeff = -12'sd4;
8'd198:  coeff = -12'sd4;
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'sd5;
8'd205:  coeff = -12'sd5;
8'd206: coeff = -12'sd5;
8'd207: coeff = -12'sd5;
8'd208: coeff = -12'sd5;
```

```
      8'd209: coeff = -12'sd5;
      8'd210: coeff = -12'sd5;
      8'd211: coeff = -12'sd5;
      8'd212: coeff = -12'sd5;
      8'd213: coeff = -12'sd5;
      8'd214: coeff = -12'sd4;
      8'd215: coeff = -12'sd4;
      8'd216: coeff = -12'sd4;
      8'd217: coeff = -12'sd4;
      8'd218: coeff = -12'sd4;
      8'd219: coeff = -12'sd4;
      8'd220: coeff = -12'sd3;
      8'd221: coeff = -12'sd3;
      8'd222: coeff = -12'sd3;
      8'd223: coeff = -12'sd3;
      8'd224: coeff = -12'sd3;
      8'd225: coeff = -12'sd2;
      8'd226: coeff = -12'sd2;
      8'd227: coeff = -12'sd2;
      8'd228:  coeff = -12'sd2;
      8'd229:  coeff = -12'sd2;
      8'd230:  coeff = -12'sd2;
      8'd231: coeff = -12'sd2;
      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'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;
      8'd251: coeff = 12'sd0;
      8'd252: coeff = 12'sd0;
      8'd253: coeff = 12'sd0;
      8'd254: coeff = 12'sd0;
      8'd255: coeff = 12'sd0;
      default: coeff = 12'hXXX;
    endcase
endmodule


module coeffs950(
  input wire [7:0] index,
  output reg signed [11:0] coeff
); // Lowpass filter with cutoff frequency at 950Hz.
  // tools will turn this into a 256x12 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'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'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'sd2;
8'd36:  coeff = -12'sd2;
8'd37:  coeff = -12'sd2;
8'd38:  coeff = -12'sd3;
8'd39:  coeff = -12'sd3;
8'd40:  coeff = -12'sd3;
8'd41:  coeff = -12'sd3;
8'd42: coeff = -12'sd4;
8'd43: coeff = -12'sd4;
8'd44: coeff = -12'sd4;
8'd45: coeff = -12'sd5;
8'd46: coeff = -12'sd5;
8'd47: coeff = -12'sd6;
8'd48: coeff = -12'sd6;
8'd49: coeff = -12'sd6;
8'd50: coeff = -12'sd6;
8'd51: coeff = -12'sd7;
8'd52: coeff = -12'sd7;
8'd53: coeff = -12'sd7;
8'd54: coeff = -12'sd8;
8'd55: coeff = -12'sd8;
8'd56: coeff = -12'sd8;
8'd57: coeff = -12'sd8;
8'd58: coeff = -12'sd8;
8'd59: coeff = -12'sd8;
8'd60: coeff = -12'sd8;
```

```
8'd61: coeff = -12'sd8;
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'sd7;
8'd68:  coeff = -12'sd7;
8'd69:  coeff = -12'sd6;
8'd70:  coeff = -12'sd6;
8'd71:  coeff = -12'sd5;
8'd72:  coeff = -12'sd5;
8'd73:  coeff = -12'sd4;
8'd74: coeff = -12'sd3;
8'd75: coeff = -12'sd2;
8'd76: coeff = -12'sd1;
8'd77: coeff = 12'sd0;
8'd78: coeff = 12'sd1;
8'd79: coeff = 12'sd2;
8'd80: coeff = 12'sd4;
8'd81: coeff = 12'sd5;
8'd82: coeff = 12'sd7;
8'd83: coeff = 12'sd8;
8'd84: coeff = 12'sd10;
8'd85: coeff = 12'sd11;
8'd86: coeff = 12'sd13;
8'd87: coeff = 12'sd15;
8'd88: coeff = 12'sd17;
8'd89: coeff = 12'sd19;
8'd90: coeff = 12'sd21;
8'd91: coeff = 12'sd23;
8'd92: coeff = 12'sd25;
8'd93: coeff = 12'sd27;
8'd94: coeff = 12'sd29;
8'd95: coeff = 12'sd31;
8'd96:  coeff = 12'sd33;
8'd97:  coeff = 12'sd35;
8'd98:  coeff = 12'sd38;
8'd99:  coeff = 12'sd40;
8'd100:  coeff = 12'sd42;
8'd101:  coeff = 12'sd44;
8'd102:  coeff = 12'sd47;
8'd103:  coeff = 12'sd49;
8'd104:  coeff = 12'sd51;
8'd105:  coeff = 12'sd53;
8'd106: coeff = 12'sd54;
8'd107: coeff = 12'sd57;
8'd108: coeff = 12'sd59;
8'd109: coeff = 12'sd61;
8'd110: coeff = 12'sd63;
8'd111: coeff = 12'sd65;
8'd112: coeff = 12'sd67;
8'd113: coeff = 12'sd68;
8'd114: coeff = 12'sd70;
8'd115: coeff = 12'sd71;
8'd116: coeff = 12'sd73;
8'd117: coeff = 12'sd74;
8'd118: coeff = 12'sd75;
8'd119: coeff = 12'sd76;
8'd120: coeff = 12'sd77;
```

```
8'd121: coeff = 12'sd78;
8'd122: coeff = 12'sd79;
8'd123: coeff = 12'sd80;
8'd124: coeff = 12'sd80;
8'd125: coeff = 12'sd81;
8'd126: coeff = 12'sd81;
8'd127: coeff = 12'sd81;
8'd128:  coeff = 12'sd81;
8'd129:  coeff = 12'sd81;
8'd130:  coeff = 12'sd81;
8'd131: coeff = 12'sd80;
8'd132:  coeff = 12'sd80;
8'd133:  coeff = 12'sd79;
8'd134:  coeff = 12'sd78;
8'd135:  coeff = 12'sd77;
8'd136:  coeff = 12'sd76;
8'd137:  coeff = 12'sd75;
8'd138:  coeff = 12'sd74;
8'd139:  coeff = 12'sd73;
8'd140:  coeff = 12'sd71;
8'd141:  coeff = 12'sd70;
8'd142: coeff = 12'sd68;
8'd143: coeff = 12'sd67;
8'd144: coeff = 12'sd65;
8'd145: coeff = 12'sd63;
8'd146: coeff = 12'sd61;
8'd147: coeff = 12'sd59;
8'd148: coeff = 12'sd57;
8'd149: coeff = 12'sd55;
8'd150: coeff = 12'sd53;
8'd151: coeff = 12'sd51;
8'd152: coeff = 12'sd49;
8'd153: coeff = 12'sd47;
8'd154: coeff = 12'sd44;
8'd155: coeff = 12'sd42;
8'd156: coeff = 12'sd40;
8'd157: coeff = 12'sd38;
8'd158: coeff = 12'sd35;
8'd159: coeff = 12'sd33;
8'd160: coeff = 12'sd31;
8'd161: coeff = 12'sd29;
8'd162: coeff = 12'sd27;
8'd163: coeff = 12'sd25;
8'd164:  coeff = 12'sd23;
8'd165:  coeff = 12'sd21;
8'd166:  coeff = 12'sd19;
8'd167:  coeff = 12'sd17;
8'd168:  coeff = 12'sd15;
8'd169:  coeff = 12'sd13;
8'd170:  coeff = 12'sd11;
8'd171:  coeff = 12'sd10;
8'd172:  coeff = 12'sd8;
8'd173:  coeff = 12'sd7;
8'd174: coeff = 12'sd5;
8'd175: coeff = 12'sd4;
8'd176: coeff = 12'sd2;
8'd177: coeff = 12'sd1;
8'd178: coeff = 12'sd0;
8'd179: coeff = -12'sd1;
8'd180: coeff = -12'sd2;
```

```
8'd181: coeff = -12'sd3;
8'd182: coeff = -12'sd4;
8'd183: coeff = -12'sd5;
8'd184: coeff = -12'sd5;
8'd185: coeff = -12'sd6;
8'd186: coeff = -12'sd6;
8'd187: coeff = -12'sd7;
8'd188: coeff = -12'sd7;
8'd189: coeff = -12'sd8;
8'd190: coeff = -12'sd8;
8'd191: coeff = -12'sd8;
8'd192: coeff = -12'sd8;
8'd193: coeff = -12'sd8;
8'd194: coeff = -12'sd8;
8'd195: coeff = -12'sd8;
8'd196:  coeff = -12'sd8;
8'd197:  coeff = -12'sd8;
8'd198:  coeff = -12'sd8;
8'd199:  coeff = -12'sd8;
8'd200:  coeff = -12'sd8;
8'd201:  coeff = -12'sd8;
8'd202:  coeff = -12'sd7;
8'd203:  coeff = -12'sd7;
8'd204:  coeff = -12'sd7;
8'd205:  coeff = -12'sd6;
8'd206: coeff = -12'sd6;
8'd207: coeff = -12'sd6;
8'd208: coeff = -12'sd6;
8'd209: coeff = -12'sd5;
8'd210: coeff = -12'sd5;
8'd211: coeff = -12'sd4;
8'd212: coeff = -12'sd4;
8'd213: coeff = -12'sd4;
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'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'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'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;
      8'd251: coeff = 12'sd1;
      8'd252: coeff = 12'sd1;
      8'd253: coeff = 12'sd1;
      8'd254: coeff = 12'sd1;
      8'd255: coeff = 12'sd1;
      default: coeff = 12'hXXX;
    endcase
endmodule


module coeffs1100(
  input wire [7:0] index,
  output reg signed [11:0] coeff
); // Lowpass filter with cutoff frequency at 1100Hz.
  // tools will turn this into a 256x12 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'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'sd2;
      8'd20: coeff = 12'sd2;
      8'd21: coeff = 12'sd2;
      8'd22: coeff = 12'sd2;
      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'sd2;
8'd34:  coeff = 12'sd1;
8'd35:  coeff = 12'sd1;
8'd36:  coeff = 12'sd1;
8'd37:  coeff = 12'sd1;
8'd38:  coeff = 12'sd1;
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'sd3;
8'd48: coeff = -12'sd3;
8'd49: coeff = -12'sd4;
8'd50: coeff = -12'sd4;
8'd51: coeff = -12'sd5;
8'd52: coeff = -12'sd5;
8'd53: coeff = -12'sd6;
8'd54: coeff = -12'sd6;
8'd55: coeff = -12'sd7;
8'd56: coeff = -12'sd7;
8'd57: coeff = -12'sd8;
8'd58: coeff = -12'sd9;
8'd59: coeff = -12'sd9;
8'd60: coeff = -12'sd9;
8'd61: coeff = -12'sd10;
8'd62: coeff = -12'sd10;
8'd63: coeff = -12'sd11;
8'd64:  coeff = -12'sd11;
8'd65:  coeff = -12'sd11;
8'd66:  coeff = -12'sd11;
8'd67:  coeff = -12'sd12;
8'd68:  coeff = -12'sd12;
8'd69:  coeff = -12'sd12;
8'd70:  coeff = -12'sd12;
8'd71:  coeff = -12'sd11;
8'd72:  coeff = -12'sd11;
8'd73:  coeff = -12'sd11;
8'd74: coeff = -12'sd10;
8'd75: coeff = -12'sd10;
8'd76: coeff = -12'sd9;
8'd77: coeff = -12'sd8;
8'd78: coeff = -12'sd8;
8'd79: coeff = -12'sd7;
8'd80: coeff = -12'sd5;
8'd81: coeff = -12'sd4;
8'd82: coeff = -12'sd3;
8'd83: coeff = -12'sd1;
8'd84: coeff = 12'sd0;
8'd85: coeff = 12'sd2;
8'd86: coeff = 12'sd4;
8'd87: coeff = 12'sd6;
8'd88: coeff = 12'sd8;
8'd89: coeff = 12'sd10;
8'd90: coeff = 12'sd12;
8'd91: coeff = 12'sd14;
8'd92: coeff = 12'sd17;
```

```
8'd93: coeff = 12'sd19;
8'd94: coeff = 12'sd22;
8'd95: coeff = 12'sd25;
8'd96:  coeff = 12'sd28;
8'd97:  coeff = 12'sd30;
8'd98:  coeff = 12'sd33;
8'd99:  coeff = 12'sd36;
8'd100:  coeff = 12'sd39;
8'd101:  coeff = 12'sd42;
8'd102:  coeff = 12'sd45;
8'd103:  coeff = 12'sd48;
8'd104:  coeff = 12'sd51;
8'd105:  coeff = 12'sd54;
8'd106: coeff = 12'sd57;
8'd107: coeff = 12'sd60;
8'd108: coeff = 12'sd62;
8'd109: coeff = 12'sd65;
8'd110: coeff = 12'sd68;
8'd111: coeff = 12'sd70;
8'd112: coeff = 12'sd73;
8'd113: coeff = 12'sd75;
8'd114: coeff = 12'sd78;
8'd115: coeff = 12'sd80;
8'd116: coeff = 12'sd82;
8'd117: coeff = 12'sd84;
8'd118: coeff = 12'sd85;
8'd119: coeff = 12'sd87;
8'd120: coeff = 12'sd89;
8'd121: coeff = 12'sd90;
8'd122: coeff = 12'sd91;
8'd123: coeff = 12'sd92;
8'd124: coeff = 12'sd93;
8'd125: coeff = 12'sd93;
8'd126: coeff = 12'sd93;
8'd127: coeff = 12'sd94;
8'd128:  coeff = 12'sd94;
8'd129:  coeff = 12'sd93;
8'd130:  coeff = 12'sd93;
8'd131: coeff = 12'sd93;
8'd132:  coeff = 12'sd92;
8'd133:  coeff = 12'sd91;
8'd134:  coeff = 12'sd90;
8'd135:  coeff = 12'sd89;
8'd136:  coeff = 12'sd87;
8'd137:  coeff = 12'sd85;
8'd138:  coeff = 12'sd84;
8'd139:  coeff = 12'sd82;
8'd140:  coeff = 12'sd80;
8'd141:  coeff = 12'sd78;
8'd142: coeff = 12'sd75;
8'd143: coeff = 12'sd73;
8'd144: coeff = 12'sd70;
8'd145: coeff = 12'sd68;
8'd146: coeff = 12'sd65;
8'd147: coeff = 12'sd62;
8'd148: coeff = 12'sd60;
8'd149: coeff = 12'sd57;
8'd150: coeff = 12'sd54;
8'd151: coeff = 12'sd51;
8'd152: coeff = 12'sd48;
```

```
8'd153: coeff = 12'sd45;
8'd154: coeff = 12'sd42;
8'd155: coeff = 12'sd39;
8'd156: coeff = 12'sd36;
8'd157: coeff = 12'sd33;
8'd158: coeff = 12'sd30;
8'd159: coeff = 12'sd28;
8'd160: coeff = 12'sd25;
8'd161: coeff = 12'sd22;
8'd162: coeff = 12'sd19;
8'd163: coeff = 12'sd17;
8'd164:  coeff = 12'sd14;
8'd165:  coeff = 12'sd12;
8'd166:  coeff = 12'sd10;
8'd167:  coeff = 12'sd8;
8'd168:  coeff = 12'sd6;
8'd169:  coeff = 12'sd4;
8'd170:  coeff = 12'sd2;
8'd171:  coeff = 12'sd0;
8'd172:  coeff = -12'sd1;
8'd173:  coeff = -12'sd3;
8'd174: coeff = -12'sd4;
8'd175: coeff = -12'sd5;
8'd176: coeff = -12'sd7;
8'd177: coeff = -12'sd8;
8'd178: coeff = -12'sd8;
8'd179: coeff = -12'sd9;
8'd180: coeff = -12'sd10;
8'd181: coeff = -12'sd10;
8'd182: coeff = -12'sd11;
8'd183: coeff = -12'sd11;
8'd184: coeff = -12'sd11;
8'd185: coeff = -12'sd12;
8'd186: coeff = -12'sd12;
8'd187: coeff = -12'sd12;
8'd188: coeff = -12'sd12;
8'd189: coeff = -12'sd11;
8'd190: coeff = -12'sd11;
8'd191: coeff = -12'sd11;
8'd192: coeff = -12'sd11;
8'd193: coeff = -12'sd10;
8'd194: coeff = -12'sd10;
8'd195: coeff = -12'sd9;
8'd196:  coeff = -12'sd9;
8'd197:  coeff = -12'sd9;
8'd198:  coeff = -12'sd8;
8'd199:  coeff = -12'sd7;
8'd200:  coeff = -12'sd7;
8'd201:  coeff = -12'sd6;
8'd202:  coeff = -12'sd6;
8'd203:  coeff = -12'sd5;
8'd204:  coeff = -12'sd5;
8'd205:  coeff = -12'sd4;
8'd206: coeff = -12'sd4;
8'd207: coeff = -12'sd3;
8'd208: coeff = -12'sd3;
8'd209: coeff = -12'sd2;
8'd210: coeff = -12'sd2;
8'd211: coeff = -12'sd1;
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'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'sd2;
      8'd229:  coeff = 12'sd2;
      8'd230:  coeff = 12'sd2;
      8'd231: coeff = 12'sd2;
      8'd232:  coeff = 12'sd2;
      8'd233:  coeff = 12'sd2;
      8'd234:  coeff = 12'sd2;
      8'd235:  coeff = 12'sd2;
      8'd236:  coeff = 12'sd2;
      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;
      8'd251: coeff = 12'sd0;
      8'd252: coeff = 12'sd0;
      8'd253: coeff = 12'sd0;
      8'd254: coeff = 12'sd0;
      8'd255: coeff = 12'sd0;
      default: coeff = 12'hXXX;
    endcase
endmodule


module coeffs1350(
  input wire [7:0] index,
  output reg signed [11:0] coeff
); // Lowpass filter with cutoff frequency at 1350Hz.
  // tools will turn this into a 256x12 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'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'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'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'sd4;
8'd38:  coeff = 12'sd4;
8'd39:  coeff = 12'sd4;
8'd40:  coeff = 12'sd4;
8'd41:  coeff = 12'sd4;
8'd42: coeff = 12'sd4;
8'd43: coeff = 12'sd5;
8'd44: coeff = 12'sd5;
8'd45: coeff = 12'sd4;
8'd46: coeff = 12'sd4;
8'd47: coeff = 12'sd4;
8'd48: coeff = 12'sd4;
8'd49: coeff = 12'sd4;
8'd50: coeff = 12'sd3;
8'd51: coeff = 12'sd3;
8'd52: coeff = 12'sd3;
8'd53: coeff = 12'sd2;
8'd54: coeff = 12'sd2;
8'd55: coeff = 12'sd1;
8'd56: coeff = 12'sd0;
8'd57: coeff = 12'sd0;
8'd58: coeff = -12'sd1;
8'd59: coeff = -12'sd2;
8'd60: coeff = -12'sd3;
8'd61: coeff = -12'sd4;
8'd62: coeff = -12'sd5;
8'd63: coeff = -12'sd6;
8'd64:  coeff = -12'sd7;
```

```
8'd65:  coeff = -12'sd8;
8'd66:  coeff = -12'sd9;
8'd67:  coeff = -12'sd10;
8'd68:  coeff = -12'sd11;
8'd69:  coeff = -12'sd12;
8'd70:  coeff = -12'sd13;
8'd71:  coeff = -12'sd14;
8'd72:  coeff = -12'sd15;
8'd73:  coeff = -12'sd15;
8'd74: coeff = -12'sd16;
8'd75: coeff = -12'sd17;
8'd76: coeff = -12'sd17;
8'd77: coeff = -12'sd17;
8'd78: coeff = -12'sd17;
8'd79: coeff = -12'sd17;
8'd80: coeff = -12'sd17;
8'd81: coeff = -12'sd17;
8'd82: coeff = -12'sd16;
8'd83: coeff = -12'sd16;
8'd84: coeff = -12'sd15;
8'd85: coeff = -12'sd14;
8'd86: coeff = -12'sd12;
8'd87: coeff = -12'sd11;
8'd88: coeff = -12'sd9;
8'd89: coeff = -12'sd7;
8'd90: coeff = -12'sd5;
8'd91: coeff = -12'sd2;
8'd92: coeff = 12'sd0;
8'd93: coeff = 12'sd3;
8'd94: coeff = 12'sd6;
8'd95: coeff = 12'sd9;
8'd96:  coeff = 12'sd13;
8'd97:  coeff = 12'sd16;
8'd98:  coeff = 12'sd20;
8'd99:  coeff = 12'sd24;
8'd100:  coeff = 12'sd28;
8'd101:  coeff = 12'sd32;
8'd102:  coeff = 12'sd36;
8'd103:  coeff = 12'sd41;
8'd104:  coeff = 12'sd45;
8'd105:  coeff = 12'sd49;
8'd106: coeff = 12'sd54;
8'd107: coeff = 12'sd58;
8'd108: coeff = 12'sd63;
8'd109: coeff = 12'sd67;
8'd110: coeff = 12'sd71;
8'd111: coeff = 12'sd76;
8'd112: coeff = 12'sd80;
8'd113: coeff = 12'sd84;
8'd114: coeff = 12'sd88;
8'd115: coeff = 12'sd91;
8'd116: coeff = 12'sd95;
8'd117: coeff = 12'sd98;
8'd118: coeff = 12'sd101;
8'd119: coeff = 12'sd104;
8'd120: coeff = 12'sd106;
8'd121: coeff = 12'sd108;
8'd122: coeff = 12'sd110;
8'd123: coeff = 12'sd112;
8'd124: coeff = 12'sd113;
```

```
8'd125: coeff = 12'sd114;
8'd126: coeff = 12'sd115;
8'd127: coeff = 12'sd115;
8'd128:  coeff = 12'sd115;
8'd129:  coeff = 12'sd115;
8'd130:  coeff = 12'sd114;
8'd131: coeff = 12'sd113;
8'd132:  coeff = 12'sd112;
8'd133:  coeff = 12'sd110;
8'd134:  coeff = 12'sd108;
8'd135:  coeff = 12'sd106;
8'd136:  coeff = 12'sd104;
8'd137:  coeff = 12'sd101;
8'd138:  coeff = 12'sd98;
8'd139:  coeff = 12'sd95;
8'd140:  coeff = 12'sd91;
8'd141:  coeff = 12'sd88;
8'd142: coeff = 12'sd84;
8'd143: coeff = 12'sd80;
8'd144: coeff = 12'sd76;
8'd145: coeff = 12'sd71;
8'd146: coeff = 12'sd67;
8'd147: coeff = 12'sd63;
8'd148: coeff = 12'sd58;
8'd149: coeff = 12'sd54;
8'd150: coeff = 12'sd49;
8'd151: coeff = 12'sd45;
8'd152: coeff = 12'sd41;
8'd153: coeff = 12'sd36;
8'd154: coeff = 12'sd32;
8'd155: coeff = 12'sd28;
8'd156: coeff = 12'sd24;
8'd157: coeff = 12'sd20;
8'd158: coeff = 12'sd16;
8'd159: coeff = 12'sd13;
8'd160: coeff = 12'sd9;
8'd161: coeff = 12'sd6;
8'd162: coeff = 12'sd3;
8'd163: coeff = 12'sd0;
8'd164:  coeff = -12'sd2;
8'd165:  coeff = -12'sd5;
8'd166:  coeff = -12'sd7;
8'd167:  coeff = -12'sd9;
8'd168:  coeff = -12'sd11;
8'd169:  coeff = -12'sd12;
8'd170:  coeff = -12'sd14;
8'd171:  coeff = -12'sd15;
8'd172:  coeff = -12'sd16;
8'd173:  coeff = -12'sd16;
8'd174: coeff = -12'sd17;
8'd175: coeff = -12'sd17;
8'd176: coeff = -12'sd17;
8'd177: coeff = -12'sd17;
8'd178: coeff = -12'sd17;
8'd179: coeff = -12'sd17;
8'd180: coeff = -12'sd17;
8'd181: coeff = -12'sd16;
8'd182: coeff = -12'sd15;
8'd183: coeff = -12'sd15;
8'd184: coeff = -12'sd14;
```

```
8'd185: coeff = -12'sd13;
8'd186: coeff = -12'sd12;
8'd187: coeff = -12'sd11;
8'd188: coeff = -12'sd10;
8'd189: coeff = -12'sd9;
8'd190: coeff = -12'sd8;
8'd191: coeff = -12'sd7;
8'd192: coeff = -12'sd6;
8'd193: coeff = -12'sd5;
8'd194: coeff = -12'sd4;
8'd195: coeff = -12'sd3;
8'd196:  coeff = -12'sd2;
8'd197:  coeff = -12'sd1;
8'd198:  coeff = 12'sd0;
8'd199:  coeff = 12'sd0;
8'd200:  coeff = 12'sd1;
8'd201:  coeff = 12'sd2;
8'd202:  coeff = 12'sd2;
8'd203:  coeff = 12'sd3;
8'd204:  coeff = 12'sd3;
8'd205:  coeff = 12'sd3;
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'sd5;
8'd212: coeff = 12'sd5;
8'd213: coeff = 12'sd4;
8'd214: coeff = 12'sd4;
8'd215: coeff = 12'sd4;
8'd216: coeff = 12'sd4;
8'd217: coeff = 12'sd4;
8'd218: coeff = 12'sd4;
8'd219: coeff = 12'sd3;
8'd220: coeff = 12'sd3;
8'd221: coeff = 12'sd3;
8'd222: coeff = 12'sd3;
8'd223: coeff = 12'sd2;
8'd224: coeff = 12'sd2;
8'd225: coeff = 12'sd2;
8'd226: coeff = 12'sd2;
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'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;
      8'd251: coeff = -12'sd1;
      8'd252: coeff = -12'sd1;
      8'd253: coeff = -12'sd1;
      8'd254: coeff = -12'sd1;
      8'd255: coeff = -12'sd1;
      default: coeff = 12'hXXX;
    endcase
endmodule


module coeffs1600(
  input wire [7:0] index,
  output reg signed [11:0] coeff
); // Lowpass filter with cutoff frequency at 1600Hz.
  // tools will turn this into a 256x12 ROM
  always @(index)
    case (index)
      8'd0:  coeff = 12'sd1;
      8'd1:  coeff = 12'sd1;
      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'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'sd2;
      8'd21: coeff = -12'sd2;
      8'd22: coeff = -12'sd2;
      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'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'sd1;
8'd40:  coeff = 12'sd1;
8'd41:  coeff = 12'sd2;
8'd42: coeff = 12'sd2;
8'd43: coeff = 12'sd3;
8'd44: coeff = 12'sd3;
8'd45: coeff = 12'sd4;
8'd46: coeff = 12'sd4;
8'd47: coeff = 12'sd5;
8'd48: coeff = 12'sd5;
8'd49: coeff = 12'sd6;
8'd50: coeff = 12'sd6;
8'd51: coeff = 12'sd7;
8'd52: coeff = 12'sd7;
8'd53: coeff = 12'sd7;
8'd54: coeff = 12'sd8;
8'd55: coeff = 12'sd8;
8'd56: coeff = 12'sd8;
8'd57: coeff = 12'sd8;
8'd58: coeff = 12'sd7;
8'd59: coeff = 12'sd7;
8'd60: coeff = 12'sd7;
8'd61: coeff = 12'sd6;
8'd62: coeff = 12'sd6;
8'd63: coeff = 12'sd5;
8'd64:  coeff = 12'sd4;
8'd65:  coeff = 12'sd3;
8'd66:  coeff = 12'sd2;
8'd67:  coeff = 12'sd1;
8'd68:  coeff = -12'sd1;
8'd69:  coeff = -12'sd2;
8'd70:  coeff = -12'sd4;
8'd71:  coeff = -12'sd5;
8'd72:  coeff = -12'sd7;
8'd73:  coeff = -12'sd8;
8'd74: coeff = -12'sd10;
8'd75: coeff = -12'sd12;
8'd76: coeff = -12'sd13;
8'd77: coeff = -12'sd15;
8'd78: coeff = -12'sd16;
8'd79: coeff = -12'sd18;
8'd80: coeff = -12'sd19;
8'd81: coeff = -12'sd20;
8'd82: coeff = -12'sd21;
8'd83: coeff = -12'sd22;
8'd84: coeff = -12'sd23;
8'd85: coeff = -12'sd23;
8'd86: coeff = -12'sd23;
8'd87: coeff = -12'sd23;
8'd88: coeff = -12'sd22;
8'd89: coeff = -12'sd21;
8'd90: coeff = -12'sd20;
8'd91: coeff = -12'sd19;
8'd92: coeff = -12'sd17;
8'd93: coeff = -12'sd14;
8'd94: coeff = -12'sd12;
8'd95: coeff = -12'sd9;
8'd96:  coeff = -12'sd6;
```

```
8'd97:  coeff = -12'sd2;
8'd98:  coeff = 12'sd2;
8'd99:  coeff = 12'sd6;
8'd100:  coeff = 12'sd11;
8'd101:  coeff = 12'sd16;
8'd102:  coeff = 12'sd21;
8'd103:  coeff = 12'sd27;
8'd104:  coeff = 12'sd32;
8'd105:  coeff = 12'sd38;
8'd106: coeff = 12'sd44;
8'd107: coeff = 12'sd50;
8'd108: coeff = 12'sd57;
8'd109: coeff = 12'sd63;
8'd110: coeff = 12'sd69;
8'd111: coeff = 12'sd75;
8'd112: coeff = 12'sd81;
8'd113: coeff = 12'sd87;
8'd114: coeff = 12'sd93;
8'd115: coeff = 12'sd99;
8'd116: coeff = 12'sd104;
8'd117: coeff = 12'sd109;
8'd118: coeff = 12'sd114;
8'd119: coeff = 12'sd118;
8'd120: coeff = 12'sd122;
8'd121: coeff = 12'sd126;
8'd122: coeff = 12'sd129;
8'd123: coeff = 12'sd131;
8'd124: coeff = 12'sd133;
8'd125: coeff = 12'sd135;
8'd126: coeff = 12'sd136;
8'd127: coeff = 12'sd137;
8'd128:  coeff = 12'sd137;
8'd129:  coeff = 12'sd136;
8'd130:  coeff = 12'sd135;
8'd131: coeff = 12'sd1133;
8'd132:  coeff = 12'sd131;
8'd133:  coeff = 12'sd129;
8'd134:  coeff = 12'sd126;
8'd135:  coeff = 12'sd122;
8'd136:  coeff = 12'sd118;
8'd137:  coeff = 12'sd114;
8'd138:  coeff = 12'sd109;
8'd139:  coeff = 12'sd104;
8'd140:  coeff = 12'sd99;
8'd141:  coeff = 12'sd93;
8'd142: coeff = 12'sd87;
8'd143: coeff = 12'sd81;
8'd144: coeff = 12'sd75;
8'd145: coeff = 12'sd69;
8'd146: coeff = 12'sd63;
8'd147: coeff = 12'sd57;
8'd148: coeff = 12'sd50;
8'd149: coeff = 12'sd44;
8'd150: coeff = 12'sd38;
8'd151: coeff = 12'sd32;
8'd152: coeff = 12'sd27;
8'd153: coeff = 12'sd21;
8'd154: coeff = 12'sd16;
8'd155: coeff = 12'sd11;
8'd156: coeff = 12'sd6;
```

```
8'd157: coeff = 12'sd2;
8'd158: coeff = -12'sd2;
8'd159: coeff = -12'sd6;
8'd160: coeff = -12'sd9;
8'd161: coeff = -12'sd12;
8'd162: coeff = -12'sd14;
8'd163: coeff = -12'sd17;
8'd164:  coeff = -12'sd19;
8'd165:  coeff = -12'sd20;
8'd166:  coeff = -12'sd21;
8'd167:  coeff = -12'sd22;
8'd168:  coeff = -12'sd23;
8'd169:  coeff = -12'sd23;
8'd170:  coeff = -12'sd23;
8'd171:  coeff = -12'sd23;
8'd172:  coeff = -12'sd22;
8'd173:  coeff = -12'sd21;
8'd174: coeff = -12'sd20;
8'd175: coeff = -12'sd19;
8'd176: coeff = -12'sd18;
8'd177: coeff = -12'sd16;
8'd178: coeff = -12'sd15;
8'd179: coeff = -12'sd13;
8'd180: coeff = -12'sd12;
8'd181: coeff = -12'sd10;
8'd182: coeff = -12'sd8;
8'd183: coeff = -12'sd7;
8'd184: coeff = -12'sd5;
8'd185: coeff = -12'sd4;
8'd186: coeff = -12'sd2;
8'd187: coeff = -12'sd1;
8'd188: coeff = 12'sd1;
8'd189: coeff = 12'sd2;
8'd190: coeff = 12'sd3;
8'd191: coeff = 12'sd4;
8'd192: coeff = 12'sd5;
8'd193: coeff = 12'sd6;
8'd194: coeff = 12'sd6;
8'd195: coeff = 12'sd7;
8'd196:  coeff = 12'sd7;
8'd197:  coeff = 12'sd7;
8'd198:  coeff = 12'sd8;
8'd199:  coeff = 12'sd8;
8'd200:  coeff = 12'sd8;
8'd201:  coeff = 12'sd8;
8'd202:  coeff = 12'sd7;
8'd203:  coeff = 12'sd7;
8'd204:  coeff = 12'sd7;
8'd205:  coeff = 12'sd6;
8'd206: coeff = 12'sd6;
8'd207: coeff = 12'sd5;
8'd208: coeff = 12'sd5;
8'd209: coeff = 12'sd4;
8'd210: coeff = 12'sd4;
8'd211: coeff = 12'sd3;
8'd212: coeff = 12'sd3;
8'd213: coeff = 12'sd2;
8'd214: coeff = 12'sd2;
8'd215: coeff = 12'sd1;
8'd216: coeff = 12'sd1;
```

```
      8'd217: coeff = 12'sd0;
      8'd218: coeff = 12'sd0;
      8'd219: coeff = -12'sd1;
      8'd220: coeff = -12'sd1;
      8'd221: coeff = -12'sd1;
      8'd222: coeff = -12'sd1;
      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'sd2;
      8'd229:  coeff = -12'sd2;
      8'd230:  coeff = -12'sd2;
      8'd231: coeff = -12'sd2;
      8'd232:  coeff = -12'sd2;
      8'd233:  coeff = -12'sd2;
      8'd234:  coeff = -12'sd2;
      8'd235:  coeff = -12'sd2;
      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'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;
      8'd251: coeff = 12'sd0;
      8'd252: coeff = 12'sd0;
      8'd253: coeff = 12'sd0;
      8'd254: coeff = 12'sd1;
      8'd255: coeff = 12'sd1;
      default: coeff = 12'hXXX;
    endcase
endmodule


module filter300Hz #(parameter N = 20)(
  input wire clock,reset,ready,
  input wire signed [N-1:0] digital_in,
  output reg signed [N+11:0] digital_out
);

  parameter START = 1'b0; // Start state (first sample )
  parameter CALCULATING = 1'b1; // Calculation state
  wire signed [11:0] coeff;
  reg [7:0] offset, index, prev_offset;
  reg state;
  reg signed [N+11:0] accumulator;
  reg signed [N-1:0] ram_in;
  wire signed [N-1:0] sample;
  reg [7:0] write_mem_address = 0;
  reg [7:0] read_mem_address = 0;
```

```
new_bram #(.LOGSIZE(8), .WIDTH(20)) ram0
          (.addr_write(write_mem_address),
               .addr_read(read_mem_address),
           .clock(clock),
           .din(ram_in),
           .dout(sample),
           .we(ready));


coeffs300 fir300(.index(index), .coeff(coeff)); // Module for returning the right coeficient.

always @(posedge clock) begin

 // When reset is one, initialize all of the registers.
 if (reset) begin
    accumulator <= 0;
    index <= 0;
    offset <= 0;
    state <= START;
 end

 case(state)

    // In the START state, we store the first sample and initialize a new accumulator.
    // After that, we go into the CALCULATING state.
    START: begin

       if (ready) begin
          state <= CALCULATING;
          offset <= offset + 1;
          prev_offset <= offset;
          ram_in <= digital_in;
              write_mem_address <= offset + 1;
          accumulator <= 0;
          index <= 0;
       end
    end

    CALCULATING: begin
           // In the CALCULATING staate, at each clock cycle we update our accumulator
           // using the next sample and coefficient. We then increment the index by one.
       if (index != 255) begin
          index <= index + 1;
              read_mem_address <= prev_offset - index + 1;
          accumulator <= accumulator + coeff*sample;
       end

           // If the index is 31, our y value is ready so we output it.
           // After outputing y, we go back into the START state.
       else if (index == 255) begin
          state <= START;
          index <= 255;
          digital_out <= accumulator;
       end
    end

    // On default, reset all of the registers.
    default: begin
       accumulator <= 0;
```

```
        index <= 0;
        offset <= 0;
        state <= START;
      end
      endcase
   end
endmodule



module filter375Hz #(parameter N = 20)(
  input wire clock,reset,ready,
  input wire signed [N-1:0] digital_in,
  output reg signed [N+11:0] digital_out
);

  parameter START = 1'b0; // Start state (first sample )
  parameter CALCULATING = 1'b1; // Calculation state
  wire signed [11:0] coeff;
  reg [7:0] offset, index, prev_offset;
  reg state;
  reg signed [N+11:0] accumulator;
  reg signed [N-1:0] ram_in;
  wire signed [N-1:0] sample;
  reg [7:0] write_mem_address = 0;
  reg [7:0] read_mem_address = 0;
  reg we = 0;

  new_bram #(.LOGSIZE(8), .WIDTH(20)) ram1
            (.addr_write(write_mem_address),
                .addr_read(read_mem_address),
             .clock(clock),
             .din(ram_in),
             .dout(sample),
             .we(we));


  coeffs375 fir375(.index(index), .coeff(coeff)); // Module for returning the right coeficient.

  always @(posedge clock) begin

   // When reset is one, initialize all of the registers.
   if (reset) begin
      accumulator <= 0;
      index <= 0;
      offset <= 0;
      state <= START;
   end

   case(state)

      // In the START state, we store the first sample and initialize a new accumulator.
      // After that, we go into the CALCULATING state.
      START: begin

         if (ready) begin
            state <= CALCULATING;
            offset <= offset + 1;
            prev_offset <= offset;
            ram_in <= digital_in;
```

```
            we <= 1;
            write_mem_address <= offset;
          accumulator <= 0;
          index <= 0;
        end
      end

      CALCULATING: begin
            // In the CALCULATING staate, at each clock cycle we update our accumulator
            // using the next sample and coefficient. We then increment the index by one.
        we <= 0;

        if (index != 255) begin
          index <= index + 1;
              read_mem_address <= prev_offset - index + 1;
          accumulator <= accumulator + coeff*sample;
        end

            // If the index is 31, our y value is ready so we output it.
            // After outputing y, we go back into the START state.
        else if (index == 255) begin
          state <= START;
          index <= 255;
          digital_out <= accumulator;
        end
      end

      // On default, reset all of the registers.
      default: begin
        accumulator <= 0;
        index <= 0;
        offset <= 0;
        state <= START;
      end
      endcase
    end
endmodule


module filter450Hz #(parameter N = 20)(
  input wire clock,reset,ready,
  input wire signed [N-1:0] digital_in,
  output reg signed [N+11:0] digital_out
);

  parameter START = 1'b0; // Start state (first sample )
  parameter CALCULATING = 1'b1; // Calculation state
  wire signed [11:0] coeff;
  reg [7:0] offset, index, prev_offset;
  reg state;
  reg signed [N+11:0] accumulator;
  reg signed [N-1:0] ram_in;
  wire signed [N-1:0] sample;
  reg [7:0] write_mem_address = 0;
  reg [7:0] read_mem_address = 0;
  reg we = 0;

  new_bram #(.LOGSIZE(8), .WIDTH(20)) ram2
            (.addr_write(write_mem_address),
                .addr_read(read_mem_address),
```

```
        .clock(clock),
        .din(ram_in),
        .dout(sample),
        .we(we));


coeffs450 fir450(.index(index), .coeff(coeff)); // Module for returning the right coeficient.

always @(posedge clock) begin

 // When reset is one, initialize all of the registers.
 if (reset) begin
   accumulator <= 0;
   index <= 0;
   offset <= 0;
   state <= START;
 end

 case(state)

   // In the START state, we store the first sample and initialize a new accumulator.
   // After that, we go into the CALCULATING state.
   START: begin

     if (ready) begin
       state <= CALCULATING;
       offset <= offset + 1;
       prev_offset <= offset;
       ram_in <= digital_in;
           write_mem_address <= offset;
       accumulator <= 0;
       index <= 0;
           we <= 1;
     end
   end

   CALCULATING: begin
         // In the CALCULATING staate, at each clock cycle we update our accumulator
         // using the next sample and coefficient. We then increment the index by one.
         we <= 0;

     if (index != 255) begin
       index <= index + 1;
           read_mem_address <= prev_offset - index + 1;
       accumulator <= accumulator + coeff*sample;
     end

         // If the index is 31, our y value is ready so we output it.
         // After outputing y, we go back into the START state.
     else if (index == 255) begin
       state <= START;
       index <= 255;
       digital_out <= accumulator;
     end
   end

   // On default, reset all of the registers.
   default: begin
     accumulator <= 0;
     index <= 0;
```

```
        offset <= 0;
        state <= START;
      end
      endcase
   end
endmodule



module filter540Hz #(parameter N = 20)(
  input wire clock,reset,ready,
  input wire signed [N-1:0] digital_in,
  output reg signed [N+11:0] digital_out
);

  parameter START = 1'b0; // Start state (first sample )
  parameter CALCULATING = 1'b1; // Calculation state
  wire signed [11:0] coeff;
  reg [7:0] offset, index, prev_offset;
  reg state;
  reg signed [N+11:0] accumulator;
  reg signed [N-1:0] ram_in;
  wire signed [N-1:0] sample;
  reg [7:0] write_mem_address = 0;
  reg [7:0] read_mem_address = 0;
  reg we = 0;

  new_bram #(.LOGSIZE(8), .WIDTH(20)) ram3
            (.addr_write(write_mem_address),
                 .addr_read(read_mem_address),
             .clock(clock),
             .din(ram_in),
             .dout(sample),
             .we(we));


  coeffs540 fir540(.index(index), .coeff(coeff)); // Module for returning the right coeficient.

  always @(posedge clock) begin

   // When reset is one, initialize all of the registers.
   if (reset) begin
      accumulator <= 0;
      index <= 0;
      offset <= 0;
      state <= START;
   end

   case(state)

      // In the START state, we store the first sample and initialize a new accumulator.
      // After that, we go into the CALCULATING state.
      START: begin

         if (ready) begin
            state <= CALCULATING;
            offset <= offset + 1;
            prev_offset <= offset;
            ram_in <= digital_in;
                write_mem_address <= offset;
```

```
        accumulator <= 0;
        index <= 0;
            we <= 1;
      end
    end

    CALCULATING: begin
          // In the CALCULATING staate, at each clock cycle we update our accumulator
          // using the next sample and coefficient. We then increment the index by one.
      we <= 0;

      if (index != 255) begin
        index <= index + 1;
            read_mem_address <= prev_offset - index + 1;
        accumulator <= accumulator + coeff*sample;
      end

          // If the index is 31, our y value is ready so we output it.
          // After outputing y, we go back into the START state.
      else if (index == 255) begin
        state <= START;
        index <= 255;
        digital_out <= accumulator;
      end
    end

    // On default, reset all of the registers.
    default: begin
      accumulator <= 0;
      index <= 0;
      offset <= 0;
      state <= START;
    end
    endcase
  end
endmodule



module filter660Hz #(parameter N = 20)(
  input wire clock,reset,ready,
  input wire signed [N-1:0] digital_in,
  output reg signed [N+11:0] digital_out
);

  parameter START = 1'b0; // Start state (first sample )
  parameter CALCULATING = 1'b1; // Calculation state
  wire signed [11:0] coeff;
  reg [7:0] offset, index, prev_offset;
  reg state;
  reg signed [N+11:0] accumulator;
  reg signed [N-1:0] ram_in;
  wire signed [N-1:0] sample;
  reg [7:0] write_mem_address = 0;
  reg [7:0] read_mem_address = 0;
  reg we = 0;

  new_bram #(.LOGSIZE(8), .WIDTH(20)) ram4
          (.addr_write(write_mem_address),
              .addr_read(read_mem_address),
```

```
        .clock(clock),
        .din(ram_in),
        .dout(sample),
        .we(we));


coeffs660 fir660(.index(index), .coeff(coeff)); // Module for returning the right coeficient.

always @(posedge clock) begin

 // When reset is one, initialize all of the registers.
 if (reset) begin
   accumulator <= 0;
   index <= 0;
   offset <= 0;
   state <= START;
 end

 case(state)

   // In the START state, we store the first sample and initialize a new accumulator.
   // After that, we go into the CALCULATING state.
   START: begin

     if (ready) begin
       state <= CALCULATING;
       offset <= offset + 1;
       prev_offset <= offset;
       ram_in <= digital_in;
           write_mem_address <= offset;
       accumulator <= 0;
       index <= 0;
           we <= 1;
     end
   end

   CALCULATING: begin
         // In the CALCULATING staate, at each clock cycle we update our accumulator
         // using the next sample and coefficient. We then increment the index by one.
     we <= 0;

     if (index != 255) begin
       index <= index + 1;
           read_mem_address <= prev_offset - index + 1;
       accumulator <= accumulator + coeff*sample;
     end

         // If the index is 31, our y value is ready so we output it.
         // After outputing y, we go back into the START state.
     else if (index == 255) begin
       state <= START;
       index <= 255;
       digital_out <= accumulator;
     end
   end

   // On default, reset all of the registers.
   default: begin
     accumulator <= 0;
     index <= 0;
```

```
        offset <= 0;
        state <= START;
      end
   endcase
   end
endmodule


module filter800Hz #(parameter N = 20)(
  input wire clock,reset,ready,
  input wire signed [N-1:0] digital_in,
  output reg signed [N+11:0] digital_out
);

  parameter START = 1'b0; // Start state (first sample )
  parameter CALCULATING = 1'b1; // Calculation state
  wire signed [11:0] coeff;
  reg [7:0] offset, index, prev_offset;
  reg state;
  reg signed [N+11:0] accumulator;
  reg signed [N-1:0] ram_in;
  wire signed [N-1:0] sample;
  reg [7:0] write_mem_address = 0;
  reg [7:0] read_mem_address = 0;
  reg we = 0;

  new_bram #(.LOGSIZE(8), .WIDTH(20)) ram5
            (.addr_write(write_mem_address),
                 .addr_read(read_mem_address),
             .clock(clock),
             .din(ram_in),
             .dout(sample),
             .we(we));


  coeffs800 fir800(.index(index), .coeff(coeff)); // Module for returning the right coeficient.

  always @(posedge clock) begin

   // When reset is one, initialize all of the registers.
   if (reset) begin
      accumulator <= 0;
      index <= 0;
      offset <= 0;
      state <= START;
   end

   case(state)

      // In the START state, we store the first sample and initialize a new accumulator.
      // After that, we go into the CALCULATING state.
      START: begin

         if (ready) begin
            state <= CALCULATING;
            offset <= offset + 1;
            prev_offset <= offset;
            ram_in <= digital_in;
                write_mem_address <= offset;
            accumulator <= 0;
```

```
          index <= 0;
              we <= 1;
        end
      end

      CALCULATING: begin
            // In the CALCULATING staate, at each clock cycle we update our accumulator
            // using the next sample and coefficient. We then increment the index by one.
            we <= 0;

        if (index != 255) begin
          index <= index + 1;
              read_mem_address <= prev_offset - index + 1;
          accumulator <= accumulator + coeff*sample;
        end

            // If the index is 31, our y value is ready so we output it.
            // After outputing y, we go back into the START state.
        else if (index == 255) begin
          state <= START;
          index <= 255;
          digital_out <= accumulator;
        end
      end

      // On default, reset all of the registers.
      default: begin
        accumulator <= 0;
        index <= 0;
        offset <= 0;
        state <= START;
      end
      endcase
    end
endmodule



module filter950Hz #(parameter N = 20)(
  input wire clock,reset,ready,
  input wire signed [N-1:0] digital_in,
  output reg signed [N+11:0] digital_out
);

  parameter START = 1'b0; // Start state (first sample )
  parameter CALCULATING = 1'b1; // Calculation state
  wire signed [11:0] coeff;
  reg [7:0] offset, index, prev_offset;
  reg state;
  reg signed [N+11:0] accumulator;
  reg signed [N-1:0] ram_in;
  wire signed [N-1:0] sample;
  reg [7:0] write_mem_address = 0;
  reg [7:0] read_mem_address = 0;
  reg we = 0;

  new_bram #(.LOGSIZE(8), .WIDTH(20)) ram6
            (.addr_write(write_mem_address),
                .addr_read(read_mem_address),
             .clock(clock),
```

```
            .din(ram_in),
            .dout(sample),
            .we(ready));


coeffs950 fir950(.index(index), .coeff(coeff)); // Module for returning the right coeficient.

always @(posedge clock) begin

 // When reset is one, initialize all of the registers.
 if (reset) begin
    accumulator <= 0;
    index <= 0;
    offset <= 0;
    state <= START;
 end

 case(state)

    // In the START state, we store the first sample and initialize a new accumulator.
    // After that, we go into the CALCULATING state.
    START: begin

       if (ready) begin
          state <= CALCULATING;
          offset <= offset + 1;
          prev_offset <= offset;
          ram_in <= digital_in;
              write_mem_address <= offset;
          accumulator <= 0;
          index <= 0;
              we <= 1;
       end
    end

    CALCULATING: begin
           // In the CALCULATING staate, at each clock cycle we update our accumulator
           // using the next sample and coefficient. We then increment the index by one.
           we <= 0;

       if (index != 255) begin
          index <= index + 1;
              read_mem_address <= prev_offset - index + 1;
          accumulator <= accumulator + coeff*sample;
       end

           // If the index is 31, our y value is ready so we output it.
           // After outputing y, we go back into the START state.
       else if (index == 255) begin
          state <= START;
          index <= 255;
          digital_out <= accumulator;
       end
    end

    // On default, reset all of the registers.
    default: begin
       accumulator <= 0;
       index <= 0;
       offset <= 0;
```

```
        state <= START;
      end
      endcase
   end
endmodule


module filter1100Hz #(parameter N = 20)(
  input wire clock,reset,ready,
  input wire signed [N-1:0] digital_in,
  output reg signed [N+11:0] digital_out
);

  parameter START = 1'b0; // Start state (first sample )
  parameter CALCULATING = 1'b1; // Calculation state
  wire signed [11:0] coeff;
  reg [7:0] offset, index, prev_offset;
  reg state;
  reg signed [N+11:0] accumulator;
  reg signed [N-1:0] ram_in;
  wire signed [N-1:0] sample;
  reg [7:0] write_mem_address = 0;
  reg [7:0] read_mem_address = 0;
  reg we = 0;

  new_bram #(.LOGSIZE(8), .WIDTH(20)) ram7
            (.addr_write(write_mem_address),
                 .addr_read(read_mem_address),
             .clock(clock),
             .din(ram_in),
             .dout(sample),
             .we(we));


  coeffs1100 fir1100(.index(index), .coeff(coeff)); // Module for returning the right coeficient.

  always @(posedge clock) begin

   // When reset is one, initialize all of the registers.
   if (reset) begin
      accumulator <= 0;
      index <= 0;
      offset <= 0;
      state <= START;
   end

   case(state)

      // In the START state, we store the first sample and initialize a new accumulator.
      // After that, we go into the CALCULATING state.
      START: begin

         if (ready) begin
            state <= CALCULATING;
            offset <= offset + 1;
            prev_offset <= offset;
            ram_in <= digital_in;
                write_mem_address <= offset;
            accumulator <= 0;
            index <= 0;
```

```
            we <= 1;
        end
    end

    CALCULATING: begin
            // In the CALCULATING staate, at each clock cycle we update our accumulator
            // using the next sample and coefficient. We then increment the index by one.
            we <= 0;

        if (index != 255) begin
            index <= index + 1;
                read_mem_address <= prev_offset - index + 1;
            accumulator <= accumulator + coeff*sample;
        end

            // If the index is 31, our y value is ready so we output it.
            // After outputing y, we go back into the START state.
        else if (index == 255) begin
            state <= START;
            index <= 255;
            digital_out <= accumulator;
        end
    end

    // On default, reset all of the registers.
    default: begin
        accumulator <= 0;
        index <= 0;
        offset <= 0;
        state <= START;
    end
    endcase
  end
endmodule



module filter1350Hz #(parameter N = 20)(
  input wire clock,reset,ready,
  input wire signed [N-1:0] digital_in,
  output reg signed [N+11:0] digital_out
);

  parameter START = 1'b0; // Start state (first sample )
  parameter CALCULATING = 1'b1; // Calculation state
  wire signed [11:0] coeff;
  reg [7:0] offset, index, prev_offset;
  reg state;
  reg signed [N+11:0] accumulator;
  reg signed [N-1:0] ram_in;
  wire signed [N-1:0] sample;
  reg [7:0] write_mem_address = 0;
  reg [7:0] read_mem_address = 0;
  reg we = 0;

  new_bram #(.LOGSIZE(8), .WIDTH(20)) ram8
            (.addr_write(write_mem_address),
                .addr_read(read_mem_address),
             .clock(clock),
             .din(ram_in),
```

```
            .dout(sample),
            .we(we));


coeffs1350 fir1350(.index(index), .coeff(coeff)); // Module for returning the right coeficient.

always @(posedge clock) begin

 // When reset is one, initialize all of the registers.
 if (reset) begin
    accumulator <= 0;
    index <= 0;
    offset <= 0;
    state <= START;
 end

 case(state)

    // In the START state, we store the first sample and initialize a new accumulator.
    // After that, we go into the CALCULATING state.
    START: begin

       if (ready) begin
          state <= CALCULATING;
          offset <= offset + 1;
          prev_offset <= offset;
          ram_in <= digital_in;
          write_mem_address <= offset;
          accumulator <= 0;
          index <= 0;
              we <= 1;
       end
    end

    CALCULATING: begin
            // In the CALCULATING staate, at each clock cycle we update our accumulator
            // using the next sample and coefficient. We then increment the index by one.
            we <= 0;

       if (index != 255) begin
          index <= index + 1;
              read_mem_address <= prev_offset - index + 1;
          accumulator <= accumulator + coeff*sample;
       end

            // If the index is 31, our y value is ready so we output it.
            // After outputing y, we go back into the START state.
       else if (index == 255) begin
          state <= START;
          index <= 255;
          digital_out <= accumulator;
       end
    end

    // On default, reset all of the registers.
    default: begin
       accumulator <= 0;
       index <= 0;
       offset <= 0;
       state <= START;
```

```
    end
    endcase
  end
endmodule


module filter1600Hz #(parameter N = 20)(
  input wire clock,reset,ready,
  input wire signed [N-1:0] digital_in,
  output reg signed [N+11:0] digital_out
);

  parameter START = 1'b0; // Start state (first sample )
  parameter CALCULATING = 1'b1; // Calculation state
  wire signed [11:0] coeff;
  reg [7:0] offset, index, prev_offset;
  reg state;
  reg signed [N+11:0] accumulator;
  reg signed [N-1:0] ram_in;
  wire signed [N-1:0] sample;
  reg [7:0] write_mem_address = 0;
  reg [7:0] read_mem_address = 0;
  reg we = 0;

  new_bram #(.LOGSIZE(8), .WIDTH(20)) ram9
            (.addr_write(write_mem_address),
                .addr_read(read_mem_address),
             .clock(clock),
             .din(ram_in),
             .dout(sample),
             .we(we));


  coeffs1600 fir1600(.index(index), .coeff(coeff)); // Module for returning the right coeficient.

  always @(posedge clock) begin

   // When reset is one, initialize all of the registers.
   if (reset) begin
     accumulator <= 0;
     index <= 0;
     offset <= 0;
     state <= START;
   end

   case(state)

     // In the START state, we store the first sample and initialize a new accumulator.
     // After that, we go into the CALCULATING state.
     START: begin

       if (ready) begin
         state <= CALCULATING;
         offset <= offset + 1;
         prev_offset <= offset;
         ram_in <= digital_in;
             write_mem_address <= offset + 1;
         accumulator <= 0;
         index <= 0;
             we <= 1;
```

```
      end
    end

    CALCULATING: begin
          // In the CALCULATING staate, at each clock cycle we update our accumulator
          // using the next sample and coefficient. We then increment the index by one.
                        we <= 0;

      if (index != 255) begin
        index <= index + 1;
           read_mem_address <= prev_offset - index + 1;
        accumulator <= accumulator + coeff*sample;
      end

          // If the index is 31, our y value is ready so we output it.
          // After outputing y, we go back into the START state.
      else if (index == 255) begin
        state <= START;
        index <= 255;
        digital_out <= accumulator;
      end
    end

    // On default, reset all of the registers.
    default: begin
      accumulator <= 0;
      index <= 0;
      offset <= 0;
      state <= START;
    end
    endcase
  end
endmodule


module font_rom(
          addr,
          clk,
          dout);


input [10 : 0] addr;
input clk;
output [7 : 0] dout;

// synopsys translate_off

    BLKMEMSP_V6_1 #(
                11,     // c_addr_width
                "0",    // c_default_data
                1536,   // c_depth
                0,      // c_enable_rlocs
                0,      // c_has_default_data
                0,      // c_has_din
                0,      // c_has_en
                0,      // c_has_limit_data_pitch
                0,      // c_has_nd
                0,      // c_has_rdy
                0,      // c_has_rfd
                0,      // c_has_sinit
```

```
                0,          // c_has_we
                18,         // c_limit_data_pitch
                "font_rom.mif",  // c_mem_init_file
                0,          // c_pipe_stages
                0,          // c_reg_inputs
                "0",        // c_sinit_value
                8,          // c_width
                0,          // c_write_mode
                "0",        // c_ybottom_addr
                1,          // c_yclk_is_rising
                1,          // c_yen_is_high
                "hierarchy1",     // c_yhierarchy
                0,          // c_ymake_bmm
                "16kx1",            // c_yprimitive_type
                1,          // c_ysinit_is_high
                "1024", // c_ytop_addr
                0,          // c_yuse_single_primitive
                1,          // c_ywe_is_high
                1)          // c_yydisable_warnings
        inst (
                .ADDR(addr),
                .CLK(clk),
                .DOUT(dout),
                .DIN(),
                .EN(),
                .ND(),
                .RFD(),
                .RDY(),
                .SINIT(),
                .WE());


// synopsys translate_on

// FPGA Express black box declaration
// synopsys attribute fpga_dont_touch "true"
// synthesis attribute fpga_dont_touch of font_rom is "true"

// XST black box declaration
// box_type "black_box"
// synthesis attribute box_type of font_rom is "black_box"

endmodule



module char_string_display (vclock,hcount,vcount,pixel,cstring,cx,cy);

  parameter NCHAR = 8;          // number of 8-bit characters in cstring
  parameter NCHAR_BITS = 3; // number of bits in NCHAR

  input vclock;   // 65MHz clock
  input [10:0] hcount;     // horizontal index of current pixel (0..1023)
  input [9:0]      vcount; // vertical index of current pixel (0..767)
  output [2:0] pixel;       // char display's pixel
  input [NCHAR*8-1:0] cstring;  // character string to display
  input [10:0] cx;
  input [9:0]      cy;

  // 1 line x 8 character display (8 x 12 pixel-sized characters)
```

```
  wire [10:0]     hoff = hcount-1-cx;
  wire [9:0]      voff = vcount-cy;
  wire [NCHAR_BITS-1:0] column = NCHAR-1-hoff[NCHAR_BITS-1+4:4];  // < NCHAR
  wire [2:0]      h = hoff[3:1];          // 0 .. 7
  wire [3:0]      v = voff[4:1];                  // 0 .. 11

  // look up character to display (from character string)
  reg [7:0]  char;
  integer  n;
  always @(*)
    for (n=0 ; n<8 ; n = n+1 )                  // 8 bits per character (ASCII)
      char[n] <= cstring[column*8+n];

  // look up raster row from font rom
  wire reverse = char[7];
  wire [10:0] font_addr = char[6:0]*12 + v;    // 12 bytes per character
  wire [7:0]  font_byte;
  font_rom f(font_addr,vclock,font_byte);

  // generate character pixel if we're in the right h,v area
  wire [2:0] cpixel = (font_byte[7 - h] ^ reverse) ? 7 : 0;
  wire dispflag = ((hcount > cx) & (vcount >= cy) & (hcount <= cx+NCHAR*16)
                       & (vcount < cy + 24));
  wire [2:0] pixel = dispflag ? cpixel : 0;

endmodule


module bram #(parameter LOGSIZE=16, WIDTH=20)
             (input wire [LOGSIZE-1:0] addr,
              input wire clock,
              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] = 0;
  always @(posedge clock) begin
    if (we) mem[addr] <= din;
    dout <= mem[addr];
  end
endmodule

module new_bram #(parameter LOGSIZE=14, WIDTH=20)
             (input wire [LOGSIZE-1:0] addr_write,
                                              input wire [LOGSIZE-1:0] addr_read,
              input wire clock,
              input wire signed [WIDTH-1:0] din,
              output reg signed [WIDTH-1:0] dout,
              input wire we);
  // Modified bram code from lab into a dual-port BRAM.
  // let the tools infer the right number of BRAMs
  (* ram_style = "block" *)
  reg [WIDTH-1:0] mem[(1<<LOGSIZE)-1:0];
  always @(posedge clock) begin
    if (we) mem[addr_write] <= din;
    dout <= mem[addr_read];
  end
endmodule
```

```
module distortion #(parameter N=20, THRESHOLD_VOLTAGE_POS = 70000,
THRESHOLD_VOLTAGE_NEG = -70000, HIGH_VOLTAGE_POS = 90000, HIGH_VOLTAGE_NEG
= -90000) (
    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 options, // Type of distortion
    input wire signed [N-1:0] digital_in, // N-bit PCM data from guitar
    output reg signed [N-1:0] digital_out    // N-bit PCM data to amp
    );

  parameter SOFTDIST = 1'b0;
  parameter HARDDIST = 1'b1;

  always @(posedge clock) begin

    case(options)

      // On ready, set the values above and below a threshold equal to the threshold value.
Otherwise, output the input.
      SOFTDIST: begin

        if (digital_in > THRESHOLD_VOLTAGE_POS && ready) digital_out <=
THRESHOLD_VOLTAGE_POS;

        else if (digital_in < THRESHOLD_VOLTAGE_NEG && ready) digital_out <=
THRESHOLD_VOLTAGE_NEG;

        else if (ready) digital_out <= digital_in;

      end

      // On ready, set the values above and below a threshold equal to the threshold value.
Otherwise, output the input.
      HARDDIST: begin

        if (digital_in > HIGH_VOLTAGE_POS && ready) digital_out <= HIGH_VOLTAGE_POS;

        else if (digital_in > THRESHOLD_VOLTAGE_POS && ready) digital_out <=
THRESHOLD_VOLTAGE_POS;

        else if (digital_in < THRESHOLD_VOLTAGE_NEG && ready) digital_out <=
THRESHOLD_VOLTAGE_NEG;

        else if (digital_in < HIGH_VOLTAGE_NEG && ready) digital_out <= HIGH_VOLTAGE_NEG;

        else if (ready) digital_out <= digital_in;

      end

      default: digital_out <= digital_in;

    endcase

  end

endmodule
```

```
module delay #(parameter N=20, DELAY = 14) ( // ac97 - 48000 256-bit frames per seond and a bit clock 12.88Mhz
    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 options, // Type of distortion
    input wire signed [N-1:0] digital_in, // N-bit PCM data from guitar
    output reg signed [N-1:0] digital_out    // N-bit PCM data to amp
    );

  parameter SLAP = 1'b0; // Single repetition.
  parameter ECHO = 1'b1; // Decaying echo.
  parameter ADDRESS_COUNTER = (1<<DELAY)-1;

  reg signed [N-1:0] delay_in;
  wire signed [N-1:0] delay_out;
  reg [DELAY-1:0] write_mem_address = 0;
  reg [DELAY-1:0] read_mem_address = 1;

  // Instantiate a dual-port BRAM.
  new_bram #(.LOGSIZE(DELAY), .WIDTH(N)) bram_delay(.addr_write(write_mem_address), .addr_read(read_mem_address), .clock(clock), .din(delay_in), .dout(delay_out), .we(ready));

  always @(posedge clock) begin

    // On reset, set all registers to default values.
    if (reset) begin
      digital_out <= digital_in;
      write_mem_address <= 0;
      read_mem_address <= 1;
      delay_in <= digital_in;
    end

    else begin

      // Check the delay we want to use.
      case(options)

        SLAP: begin

          // On ready, store the current input. Output a mix of the input and a previously stored input.
          if (ready) begin

            delay_in <= digital_in;

            if (write_mem_address == ADDRESS_COUNTER) write_mem_address <= 0;
            else write_mem_address <= write_mem_address + 1;

            if (read_mem_address == ADDRESS_COUNTER) read_mem_address <= 0;
            else read_mem_address <= read_mem_address + 1;

            digital_out <= digital_in + (delay_out >>> 1);
              end
        end

        ECHO: begin

          // On ready, store the current input and the delayed signal. Output a mix of the input and a previously stored input.
          if (ready) begin
```

```
            delay_in <= digital_in + (delay_out >>> 1);

            if (write_mem_address == ADDRESS_COUNTER) write_mem_address <= 0;
            else write_mem_address <= write_mem_address + 1;

            if (read_mem_address == ADDRESS_COUNTER) read_mem_address <= 0;
            else read_mem_address <= read_mem_address + 1;

            digital_out <= digital_in + (delay_out >>> 1);

          end
        end

        // Set registers to their default values.
        default: begin
          digital_out <= digital_in;
          write_mem_address <= 0;
          read_mem_address <= 1;
          delay_in <= digital_in;
        end
      endcase
    end
  end
endmodule


module auto_wah #(parameter N = 20, SPEED = 100)(
          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 wah_ready, // 1 when wah data is available
          input wire signed [N-1:0] digital_in, // N-bit PCM data from guitar
          output reg signed [N-1:0] digital_out); // N-bit PCM data to amp

   parameter INCREMENTING = 0; // State in which increase the center frequency of the bandpass
filter.
   parameter DECREMENTING = 1;  // State in which decrease the center frequency of the bandpass
filter.
   reg [3:0] wah_counter = 0; // Chooses the bandpass filter.
   reg [15:0] counter = 0; // Counter used to determine the speed of the effect.
   reg state = 0;
   reg signed [N+11:0] filter_out;
   wire signed [N+11:0] f1, f2, f3, f4, f5, f6, f7, f8, f9, f10; // Signals produced by the filters.

   // Instantiation of 10 different lowpass filters.

   filter300Hz filter1(.clock(clock), .reset(reset), .ready(ready), .digital_in(digital_in), .digital_out(f1));
   filter375Hz filter2(.clock(clock), .reset(reset), .ready(ready), .digital_in(digital_in), .digital_out(f2));
   filter450Hz filter3(.clock(clock), .reset(reset), .ready(ready), .digital_in(digital_in), .digital_out(f3));
   filter540Hz filter4(.clock(clock), .reset(reset), .ready(ready), .digital_in(digital_in), .digital_out(f4));
   filter660Hz filter5(.clock(clock), .reset(reset), .ready(ready), .digital_in(digital_in), .digital_out(f5));
   filter800Hz filter6(.clock(clock), .reset(reset), .ready(ready), .digital_in(digital_in), .digital_out(f6));
   filter950Hz filter7(.clock(clock), .reset(reset), .ready(ready), .digital_in(digital_in), .digital_out(f7));
   filter1100Hz filter8(.clock(clock), .reset(reset), .ready(ready), .digital_in(digital_in), .digital_out(f8));
   filter1350Hz filter9(.clock(clock), .reset(reset), .ready(ready), .digital_in(digital_in), .digital_out(f9));
   filter1600Hz filter10(.clock(clock), .reset(reset), .ready(ready), .digital_in(digital_in),
.digital_out(f10));

   always @(posedge clock) begin
```

```
    // Whenever wah effect is ready, apply the filters.
    if (wah_ready) begin

      case (state)

        INCREMENTING: begin

          // Whenever counter reaches SPEED, go to the next center frequency.
          if (counter == SPEED) begin
            wah_counter <= wah_counter + 1;
            counter <= 0;
          end

          // Increment counter.
          else counter <= counter + 1;

          // If we reach the highest center frequency start going backward.
          if (wah_counter == 8) state <= DECREMENTING;

        end


        DECREMENTING: begin

          // Whenever counter reaches SPEED, go to the next center frequency.
          if (counter == SPEED) begin
            wah_counter <= wah_counter - 1;
            counter <= 0;
          end

          // Increment counter.
          else counter <= counter + 1;

          // If we reach the lowest center frequency start going forward.
          if (wah_counter == 0) state <= INCREMENTING;

        end
      endcase

      // Output the appropriate filter.
      case(wah_counter)

         4'd0: filter_out <= f2 - f1;
         4'd1: filter_out <= f3 - f2;
         4'd2: filter_out <= f4 - f3;
         4'd3: filter_out <= f5 - f4;
         4'd4: filter_out <= f6 - f5;
         4'd5: filter_out <= f7 - f6;
         4'd6: filter_out <= f8 - f7;
         4'd7: filter_out <= f9 - f8;
         4'd8: filter_out <= f10 - f9;
         default: filter_out <= f1;
      endcase

      digital_out <= digital_in + filter_out[N+11:12]; // Mix the input with the output of the bandpass
filter and output the mixed signal.
    end
  end
endmodule
```

```
module chorus #(parameter N=20, CHORUS_DELAY = 12) ( // ac97 - 48000 256-bit frames per
seond and a bit clock 12.88Mhz
    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 signed [N-1:0] digital_in, // N-bit PCM data from guitar
    output reg signed [N-1:0] digital_out    // N-bit PCM data to amp
    );

  parameter ADDRESS_COUNTER = (1<<CHORUS_DELAY) - 1;

  reg signed [N-1:0] chorus_in;
  wire signed [N-1:0] chorus_out;
  reg [CHORUS_DELAY-1:0] write_mem_address = 0;
  reg [CHORUS_DELAY-1:0] read_mem_address = 1;

  // Dual-port ram.
  new_bram #(.LOGSIZE(CHORUS_DELAY), .WIDTH(N))
bram_chorus(.addr_write(write_mem_address), .addr_read(read_mem_address), .clock(clock),
.din(chorus_in), .dout(chorus_out), .we(ready));

  always @(posedge clock) begin

    // On reset, reset all the registers.
    if (reset) begin
      digital_out <= digital_in;
      write_mem_address <= 0;
      read_mem_address <= 1;
      chorus_in <= digital_in;
    end

    // On ready, collect the time delayed signal and output it. Increment the memory addresses.
    else if (ready) begin

      chorus_in <= digital_in;

      if (write_mem_address == ADDRESS_COUNTER) write_mem_address <= 0;
      else write_mem_address <= write_mem_address + 1;

      if (read_mem_address == ADDRESS_COUNTER) read_mem_address <= 0;
      else read_mem_address <= read_mem_address + 1;

      digital_out <= (chorus_out >>> 1);
    end
  end
endmodule


module Effects_Controller #(parameter N = 20) (
    input wire clock, // 27mhz system clock
    input wire reset, // 1 to reset to initial state
    input wire ready, // 1 when AC97 data is available - 48kHz
    input wire distortion_options, // Type of distortion
    input wire delay_options, // Type of delay
    input wire distortion, // Is distortion used
    input wire delay, // Is delay used
    input wire wah, // Is wah used
    input wire chorus, // Is chorus used
    input wire reverb, // Is reverb used
    input wire pitch_shifter, // Is pitch-shifter used
```

```
    input wire phaser, // Is phaser used
    input wire signed [N-1:0] digital_in, // N-bit PCM data from guitar
    output reg signed [N-1:0] digital_out    // N-bit PCM data to amp
    );

    parameter START = 3'b000;
    parameter WAH = 3'b001;
    parameter DISTORTION = 3'b010;
    parameter CHORUS = 3'b011;
    parameter PHASER = 3'b100;
    parameter PITCH_SHIFTER = 3'b101;
    parameter DELAY = 3'b110;
    parameter REVERB = 3'b111;

    reg [2:0] state;
    reg signed [N-1:0] temp;
    wire signed [N-1:0] distortion_out, delay_out, chorus_out, wah_out;
    reg dready, wready, cready, phready, piready, deready, rready; // Ready signals for each of the
effects.
    reg [6:0] counter = 0;

    distortion #(.N(N)) distortion1(.clock(clock), .reset(reset), .ready(dready),
.options(distortion_options), .digital_in(temp), .digital_out(distortion_out));
    delay #(.N(N)) delay1(.clock(clock), .reset(reset), .ready(deready), .options(delay_options),
.digital_in(temp),  .digital_out(delay_out));
    chorus #(.N(N)) chorus1(.clock(clock), .reset(reset), .ready(cready), .digital_in(temp),
.digital_out(chorus_out));
    auto_wah #(.N(N)) wah1(.clock(clock), .reset(reset), .ready(ready), .wah_ready(wready),
.digital_in(temp), .digital_out(wah_out));

    always @(posedge clock) begin

      // Reset if the reset signal is high.
      if (reset) begin

        temp <= digital_in;
        state <= START;
        digital_out <= digital_in;
        counter <= 0;
      end

      // At each state apply one effect using the passed parameters and go to the next state.
      case(state)

        START: begin

          // When the AC97 is ready start applying the active effects.
          if (ready) begin
            state <= WAH;
            if (wah) wready <= 1'b1;
            if (counter == 100) counter <= 0;
            else counter <= counter + 1;
          end

          temp <= digital_in;

          end


        WAH: begin
```

```
            wready <= 1'b0;
            state <= DISTORTION;
            if (distortion) dready <= 1'b1;
            if (wready) temp <= wah_out;

        end

        DISTORTION: begin

            dready <= 1'b0;
            state <= CHORUS;
            if (dready) temp <= distortion_out;
            if (chorus && counter != 100) cready <= 1'b1; // Change sampling rate to introduce a small
pitch change.

        end

        CHORUS: begin

            cready <= 1'b0;
            state <= PHASER;
            if (cready) temp <= temp + chorus_out;
            if (phaser) phready <= 1'b1;

        end

        PHASER:begin

            phready <= 1'b0;
            state <= PITCH_SHIFTER;
            if (pitch_shifter) piready <= 1'b1;

        end

        PITCH_SHIFTER:begin

            piready <= 1'b0;
            state <= DELAY;
            if (delay) deready <= 1'b1;

        end

        DELAY:begin

            deready <= 1'b0;
            state <= REVERB;
            if (deready) temp <= delay_out;
            if (reverb) rready <= 1'b1;

        end

        REVERB:begin

            rready <= 1'b0;
            state <= START;
            digital_out <= temp; // Output the resulting signal.

        end
      endcase
```

```
    end
endmodule


//////////////////////////////////////////////////////////////////////////////////
//
// 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 audio (
       input wire       clock_27mhz,
       input wire       reset,
       input wire [4:0]  volume,
       output wire [19:0] audio_in_data,
       input wire [19:0]  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 [19:0] out_data;
  always @ (posedge clock_27mhz)
    if (ready) out_data <= audio_out_data;
  assign audio_in_data = left_in_data[19:0];
  assign left_out_data = out_data;
  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

//////////////////////////////////////////////////////////////////////////////////
////
//// 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 fpga_processor   (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;
// ac97_sdata_in is an input
```

```
  // 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
  //Ram default changes from the labkit
//Enable RAM0 - the control signals are active low
   assign ram0_ce_b = 1'b0;
   assign ram0_oe_b = 1'b0;
   assign ram0_bwe_b = 4'h0;
   assign ram0_adv_ld = 1'b0;
//  Use the RAM data, address and write enable lines in place of BRAM
//  assign ram0_data = 36'hZ;        // store audio data here
//  assign ram0_address = 19'h0;     // ram address - 19 bits wide
   assign ram0_clk = clock_27mhz;
   assign ram0_cen_b = 1'b0;
//  assign ram0_we_b = 1'b1          // ram write control active low
  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
// 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
//   wire [7:0] from_ac97_data, to_ac97_data;
//   wire ready;

// 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;

//////////////////////////////////////////////////////////////////////
//
// 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.
//
//////////////////////////////////////////////////////////////////////

// use FPGA's digital clock manager to produce a
```

```
  // 65MHz clock (actually 64.8MHz)
  wire clock_65mhz_unbuf,clock_65mhz;
  DCM vclk1(.CLKIN(clock_27mhz),.CLKFX(clock_65mhz_unbuf));
  // synthesis attribute CLKFX_DIVIDE of vclk1 is 10
  // synthesis attribute CLKFX_MULTIPLY of vclk1 is 24
  // synthesis attribute CLK_FEEDBACK of vclk1 is NONE
  // synthesis attribute CLKIN_PERIOD of vclk1 is 37
  BUFG vclk2(.O(clock_65mhz),.I(clock_65mhz_unbuf));

  wire reset, power_on_reset;
  SRL16 reset_sr2 (.D(1'b0), .CLK(clock_65mhz), .Q(power_on_reset),
       .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
  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 [19:0] from_ac97_data, to_ac97_data;
  wire ready;

  // display module for debugging

/*
   wire clock_27mhz_buf;
  BUFG vclk3(.O(clock_27mhz_buf),.I(clock_27mhz));

  wire [63:0] dispdata = {1'b0,cx2,2'b0,cy2,40'b0};
  display_16hex hexdisp1(reset, clock_27mhz_buf, dispdata,
    disp_blank, disp_clock, disp_rs, disp_ce_b,
    disp_reset_b, disp_data_out);
*/

  // allow user to adjust volume and control the looper.
  wire vup,vdown, left_button, right_button;
  reg old_vup,old_vdown,change,loop_reset, old_lb, old_rb;
  debounce bup(.reset(reset),.clock(clock_27mhz),.noisy(~button_up),.clean(vup));
  debounce bdown(.reset(reset),.clock(clock_27mhz),.noisy(~button_down),.clean(vdown));
  debounce lb(.reset(reset),.clock(clock_27mhz),.noisy(~button_left),.clean(left_button));
  debounce rb(.reset(reset),.clock(clock_27mhz),.noisy(~button_right),.clean(right_button));
  reg [4:0] volume;
  always @ (posedge clock_27mhz) begin
    if (reset) begin
      volume <= 5'd8;
      change <= 0;
      loop_reset <= 0;
    end

    else begin
   if (vup & ~old_vup & volume != 5'd31) volume <= volume+1;
   if (vdown & ~old_vdown & volume != 5'd0) volume <= volume-1;
   if (left_button & ~old_lb) loop_reset <= 1;
   else loop_reset <= 0;
   if (right_button & ~old_rb) change <= 1;
   else change <= 0;
    end
    old_vup <= vup;
    old_vdown <= vdown;
    old_lb <= left_button;
    old_rb <= right_button;
  end

  // AC97 driver
```

```
  audio 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);


  // Debounce the used switches.
  wire distortion, delay, pitch_shifter, wah, chorus, phaser, reverb, delay_options;

  debounce sw0(.reset(reset),.clock(clock_27mhz),.noisy(switch[0]),.clean(distortion));
  debounce sw1(.reset(reset),.clock(clock_27mhz),.noisy(switch[1]),.clean(delay));
  debounce sw2(.reset(reset),.clock(clock_27mhz),.noisy(switch[2]),.clean(pitch_shifter));
  debounce sw3(.reset(reset),.clock(clock_27mhz),.noisy(switch[3]),.clean(wah));
  debounce sw4(.reset(reset),.clock(clock_27mhz),.noisy(switch[4]),.clean(chorus));
  debounce sw5(.reset(reset),.clock(clock_27mhz),.noisy(switch[5]),.clean(phaser));
  debounce sw6(.reset(reset),.clock(clock_27mhz),.noisy(switch[6]),.clean(reverb));
  debounce sw7(.reset(reset),.clock(clock_27mhz),.noisy(switch[7]),.clean(delay_options));

  // light up LEDs when recording, show volume during playback.
  // led is active low
  assign led[7:3] = 5'b11111;
  wire signed [19:0] looper_out, effects_out;
  reg [20:0] mem_address, highest_address;
  reg [2:0] state;
  reg signed [19:0] mem_inp;
  reg [20:0] highest_adress = 0;
  reg record;

  // Instantiate the effects controller.
  Effects_Controller #(.N(20)) processor(.clock(clock_27mhz), .reset(reset), .ready(ready),
.distortion(distortion), .pitch_shifter(pitch_shifter), .wah(wah), .chorus(chorus), .phaser(phaser),
.delay(delay), .reverb(reverb), .digital_in(from_ac97_data), .digital_out(effects_out),
.distortion_options(1'b1), .delay_options(delay_options));

  // Looper module
  parameter NOT_ACTIVE = 3'd0;
  parameter RECORD = 3'd1;
  parameter PLAYBACK = 3'd2;
  parameter RECORD_AND_PLAYBACK_RETRIEVE = 3'd3;
  parameter RECORD_INTERMEDIATE_ONE = 3'd4;
  parameter RECORD_INTERMEDIATE_TWO = 3'd5;
  parameter RECORD_INTERMEDIATE_THREE = 3'd6;
  parameter RECORD_AND_PLAYBACK_STORE = 3'd7;
  parameter ADDRESS_COUNTER = (1<<19)-1;
  reg [35:0] ram_data;
  reg [1:0] counter = 0;

  always @(posedge clock_27mhz) begin

    // When reset is high, set all of the registers to their default value.
    if (reset) begin
      mem_address <= 0;
      state <= NOT_ACTIVE;
      highest_adress <= 0;
      ram_data <= 0;
      record <= 1;
      counter <= 0;
    end

    case(state)
```

```
// If looper is not active, do nothing. On change, go to record state.
NOT_ACTIVE: begin

  mem_address <= 0;
  highest_address <= 0;
  ram_data <= 0;
  mem_inp <= 0;
  if (change) state <= RECORD;
end

RECORD: begin

  // On reset, go to NOT_ACTIVE state.
  if (loop_reset) state <= NOT_ACTIVE;

  // On change, go to playback state.
  else if (change) state <= PLAYBACK;

  // Store the current input to the ZBT memory. Allow for three clock cycles to hold the ZBT
  // WE long enoguh. Increment the counters and memory address appropriately.
  else if (ready) begin

      record <= 0;
      counter <= counter + 1;

      if (mem_address == ADDRESS_COUNTER) begin

      state <= PLAYBACK;
      mem_address <= 0;
      mem_inp <= effects_out;

      end

      else begin

      mem_address <= mem_address + 1;
      highest_address <= mem_address + 1;
      mem_inp <= effects_out;
      end
  end

  else if (counter == 3) begin
    record <= 1;
    counter <= 0;
  end

  else if (counter != 0) counter <= counter + 1;
end

PLAYBACK: begin

  // Play the stored signals one by one.
  record <= 1;

  // On reset, go back to not active state. On change, record over the stored data.
  if (loop_reset) state <= NOT_ACTIVE;
  else if (change) state <= RECORD_AND_PLAYBACK_RETRIEVE;

  else if (ready) begin
```

```
            ram_data <= ram0_data;
            if (mem_address == highest_address) mem_address <= 0;
            else mem_address <= mem_address + 1;

          end

        end

        RECORD_AND_PLAYBACK_RETRIEVE: begin

          // On reset, go back to not active state. On change, playback the stored data.
          if (loop_reset) state <= NOT_ACTIVE;
          else if (change) state <= PLAYBACK;

          // On ready, collect the current stored value and go to the next state.
          else if (ready) begin

            ram_data <= ram0_data;
                 state <= RECORD_INTERMEDIATE_ONE;
            mem_inp <= effects_out + ram_data[19:0];
            record <= 0;
          end

        end

        RECORD_INTERMEDIATE_ONE: state <= RECORD_INTERMEDIATE_TWO;

        RECORD_INTERMEDIATE_TWO: state <= RECORD_INTERMEDIATE_THREE;

        RECORD_INTERMEDIATE_THREE: state <= RECORD_AND_PLAYBACK_STORE;

        RECORD_AND_PLAYBACK_STORE: begin

,           // ADD the current input to the ZBT memory. Allow for three clock cycles to hold the ZBT
            // WE long enough (the three intermediate states). Increment the memory address
appropriately.
            record <= 1;
            if (loop_reset) state <= NOT_ACTIVE;
            else if (change) state <= PLAYBACK;
            else state <= RECORD_AND_PLAYBACK_RETRIEVE;

            if (mem_address == highest_address) mem_address <= 0;

            else  mem_address <= mem_address + 1;
          end

        // On default, set all the registers to their default state.
        default: begin

          mem_address <= 0;
          state <= NOT_ACTIVE;
          highest_adress <= 0;
          record <= 1;
        end

      endcase

    end

    assign led[2:0] = 3'b111;
```

```
  assign ram0_data = (~record) ? {16'd0, mem_inp} : 36'hZ; // store audio data here - ZBT memory
    assign ram0_address = mem_address; // ZBT address
  assign ram0_we_b = record; // WE signal for ZBT memory

  assign to_ac97_data = effects_out + ram_data[19:0];

  // generate basic XVGA video signals
    wire [10:0] hcount;
  wire [9:0]  vcount;
    wire hsync,vsync,blank;
    xvga xvga1(clock_65mhz,hcount,vcount,hsync,vsync,blank);

  // ASCII characters for display.
    wire [63:0] cstring1 = "Effects:";
  wire [31:0] cstring2 = "Dist";
  wire [39:0] cstring3 = "Delay";
  wire [47:0] cstring4 = "Chorus";
  wire [23:0] cstring5 = "Wah";
  wire [47:0] cstring9 = "State:";
  wire [47:0] cstring10 = "Active";
  wire [63:0] cstring11 = "Inactive";
  wire [31:0] cstring12 = "Slap";
  wire [31:0] cstring13 = "Echo";

  // Wires holding the values for VGA pixels.
  wire [2:0]  cdpixel1, cdpixel2, cdpixel3, cdpixel4, cdpixel5, cdpixel9, cdpixel10, cdpixel_dist_active,
cdpixel_dist_inactive;
  wire [2:0]  cdpixel_delay_slap, cdpixel_delay_echo, cdpixel_delay_inactive, cdpixel_chorus_active,
cdpixel_chorus_inactive, cdpixel_wah_active, cdpixel_wah_inactive;
  wire [2:0] cdpixel_dist_state, cdpixel_delay_state;
  wire [2:0] cdpixel_chorus_state, cdpixel_wah_state;

  // Modules for displaying all of the used effects and their states.
  char_string_display cd1(clock_65mhz,hcount,vcount,
        cdpixel1,cstring1,11'd320,10'd100);

  char_string_display cd2(clock_65mhz,hcount,vcount,
        cdpixel2,cstring2,11'd320,10'd180);

  char_string_display cd3(clock_65mhz,hcount,vcount,
        cdpixel3,cstring3,11'd320,10'd260);

  char_string_display cd4(clock_65mhz,hcount,vcount,
        cdpixel4,cstring4,11'd320,10'd340);

  char_string_display cd5(clock_65mhz,hcount,vcount,
        cdpixel5,cstring5,11'd320,10'd420);

  char_string_display cd9(clock_65mhz,hcount,vcount,
        cdpixel9,cstring9,11'd640,10'd100);

  char_string_display cd10(clock_65mhz,hcount,vcount,
        cdpixel_dist_active,cstring10,11'd640,10'd180);

  char_string_display cd11(clock_65mhz,hcount,vcount,
        cdpixel_dist_inactive,cstring11,11'd640,10'd180);

  assign cdpixel_dist_state = (distortion) ? cdpixel_dist_active : cdpixel_dist_inactive;

  char_string_display cd12(clock_65mhz,hcount,vcount,
```

```
        cdpixel_delay_slap,cstring12,11'd640,10'd260);

   char_string_display cd13(clock_65mhz,hcount,vcount,
        cdpixel_delay_echo,cstring13,11'd640,10'd260);

   char_string_display cd14(clock_65mhz,hcount,vcount,
        cdpixel_delay_inactive,cstring11,11'd640,10'd260);

   assign cdpixel_delay_state = (~delay) ? cdpixel_delay_inactive : (delay_options) ?
cdpixel_delay_echo : cdpixel_delay_slap;

   char_string_display cd15(clock_65mhz,hcount,vcount,
        cdpixel_wah_active,cstring10,11'd640,10'd340);

   char_string_display cd16(clock_65mhz,hcount,vcount,
        cdpixel_wah_inactive,cstring11,11'd640,10'd340);

   assign cdpixel_wah_state = (wah) ? cdpixel_wah_active : cdpixel_wah_inactive;

   char_string_display cd17(clock_65mhz,hcount,vcount,
        cdpixel_chorus_active,cstring10,11'd640,10'd420);

   char_string_display cd18(clock_65mhz,hcount,vcount,
        cdpixel_chorus_inactive,cstring11,11'd640,10'd420);

   assign cdpixel_chorus_state = (chorus) ? cdpixel_chorus_active : cdpixel_chorus_inactive;

   reg [2:0] rgb;
   reg b,hs,vs;

   // Display the appropraite characters.
   always @(posedge clock_65mhz) begin
        hs <= hsync;
        vs <= vsync;
        b <= blank;
        rgb <= cdpixel1 | cdpixel2 | cdpixel3 | cdpixel4 | cdpixel5 | cdpixel9 | cdpixel_dist_state |
cdpixel_delay_state | cdpixel_wah_state | cdpixel_chorus_state ;
        end

   // VGA Output.  In order to meet the setup and hold times of the
      // AD7125, we send it ~clock_65mhz.
      assign vga_out_red = {8{rgb[2]}};
      assign vga_out_green = {8{rgb[1]}};
      assign vga_out_blue = {8{rgb[0]}};
      assign vga_out_sync_b = 1'b1;    // not used
      assign vga_out_blank_b = ~b;
      assign vga_out_pixel_clock = ~clock_65mhz;
      assign vga_out_hsync = hs;
      assign vga_out_vsync = vs;

   // output useful things to the logic analyzer connectors
   assign analyzer1_clock = ac97_bit_clock;
   assign analyzer1_data = {12'd0, to_ac97_data[3:0]};

   assign analyzer3_clock = ready;
   assign analyzer3_data = {to_ac97_data[19:4]};
endmodule

module xvga(vclock,hcount,vcount,hsync,vsync,blank);
   input vclock;
```

```
  output [10:0] hcount;
  output [9:0] vcount;
  output   vsync;
  output   hsync;
  output   blank;

  reg     hsync,vsync,hblank,vblank,blank;
  reg [10:0]   hcount;    // pixel number on current line
  reg [9:0] vcount;  // line number

  // horizontal: 1344 pixels total
  // display 1024 pixels per line
  wire      hsyncon,hsyncoff,hreset,hblankon;
  assign    hblankon = (hcount == 1023);
  assign    hsyncon = (hcount == 1047);
  assign    hsyncoff = (hcount == 1183);
  assign    hreset = (hcount == 1343);

  // vertical: 806 lines total
  // display 768 lines
  wire      vsyncon,vsyncoff,vreset,vblankon;
  assign    vblankon = hreset & (vcount == 767);
  assign    vsyncon = hreset & (vcount == 776);
  assign    vsyncoff = hreset & (vcount == 782);
  assign    vreset = hreset & (vcount == 805);

  // sync and blanking
  wire      next_hblank,next_vblank;
  assign next_hblank = hreset ? 0 : hblankon ? 1 : hblank;
  assign next_vblank = vreset ? 0 : vblankon ? 1 : vblank;
  always @(posedge vclock) begin
    hcount <= hreset ? 0 : hcount + 1;
    hblank <= next_hblank;
    hsync <= hsyncon ? 0 : hsyncoff ? 1 : hsync;  // active low

    vcount <= hreset ? (vreset ? 0 : vcount + 1) : vcount;
    vblank <= next_vblank;
    vsync <= vsyncon ? 0 : vsyncoff ? 1 : vsync;  // active low

    blank <= next_vblank | (next_hblank & ~hreset);
  end
endmodule
```