Audio Effects Project on Basys 3 FPGA

Student 1: Kennard Ng

Student 2: Ryan Teo

In this project, we designed a sound device that had a basic feature of reading sound inputs from a microphone and releasing sound outputs. This device has several modes based on the user's inputs. Below are some special features that we have added onto the basic machine.

Features:

1. Musical Tone Generator (Ryan)
2. Hello Playback (Ryan)
3. Volume Indicator (Kennard)
4. Output Delay (Kennard)
5. Pitch Shift (Kennard)
6. Techno Tone Modifier (Kennard and Ryan)

User guide

Feature	Input	Description	Output
Musical Tone Generator	SW[0] — SW[7]	Select the following switches to play the specified note in the 4th octave: SW[0]: C SW[1]: D SW[2]: E SW[3]: F SW[4]: G SW[5]: A SW[6]: B **Note:** If more than 1 switch is activated, the note corresponding to the lowest activated switch number will be produced. **Example:** Suppose the following switches are activated: SW[1], SW[2], SW[5], SW[6]	Corresponding musical note output

		The D note (corresponding to SW[1]) will be played. Select the following switch the change the output note to the 3rd octave: SW[7]	
Hello Playback	PB_C	Push this button to playback "Hello" continuously. Push it again to stop the playback. **Note: The "Hello" playback takes precedence over all other functionality**, including the microphone pass-through. When it is activated, all other features will be disabled, regardless of their activation state. **Example:** Suppose SW[0] (C note output) is activated. PB_C is now pressed. The output sound will be "Hello" playback.	Spoken "hello" output
Volume Indicator	(DEFAULT)	This feature has a dual volume indicator. Facing the Basys 3, the leftmost volume indicator will be the display the output of the microphone, while the rightmost volume indicator will display the microphone input. **Take note that:** ● The output volume indicator uses LED[15] to LED[9]. ● The input volume indicator uses LED[0] to LED[6] ● LED[7] and LED[8] are unasserted and serve as to separate the two volume indicators. ● LED[15] and LED[0] are always asserted to indicate that both volume indicators are functional. By default the volume indicator will represent the 7 most significant bits of the volume. A value of 011 0111 represents 0110 111x xxxx. The least	LED[15] to LED[9] LED[0] to LED[6] Volume indicator output

		significant bit of the volume indicator. The most significant bit for the output volume indicator is LED[9] while its least significant bit is LED[15]. The most significant bit for the input volume input indicator is LED[6] while its least significant bit is LED[0].	
	SW[15]	When this switch is turned on, the volume indicator changes into a pseudo linear volume indicator. The number of LEDs being lit up increase with the amplitude of the sound. Louder output/input means that more leds will be turn on and be displayed. The LEDs will light up from the leftmost LED to the rightmost LED, LED[9] for the output volume indicator and it will light up from the rightmost LED to the leftmost LED, LED[6] for the input volume indicator.	LED[15] to LED[9] LED[0] to LED[6] Pseudo-linear volume indicator output
Output Delay	SW[14]	When this switch is turned on, the sound output of the device (except the “hello” playback and pitch shift) will be delayed by 0.25 seconds.	Delayed Microphone output
	SW[14] and SW[13]	When these switches are turned on, the sound output of the device (except the “hello” playback and pitch shift) will be delayed by 0.5 seconds. **Note:** You can also playback the sound output in the past 0.5s. 1. Flick SW[13] on. 2. Speak into the mic. 3. Immediately flick SW[14] on.	Delayed Microphone output (variable)
Pitch Shift	SW[12]	When this switch is turned on, the device will increase the pitch of your voice by approximately 2 times and output your pitch-shifted voice	Pitch shifted Sound output
	SW[12] and SW[11]	When these switches are turned on, the device will increase the pitch of your voice by approximately 3 times and output your pitch shifted voice	Pitch shifted Sound output (variable)

Techno Tone Modifier	SW[12]	When SW[12] is activated, it will modify the musical note currently being output into a techno tone. Works in conjunction with the octave switch as well. You can also assert SW[12] with SW[13]. Doing so will increase the number of notes you have. In total, we have 28 different techno notes that you can play. **Example:** Suppose SW[6] (B note output) and SW[12] is activated. The output sound will be a techno-modified sound based on the B note.	Corresponding techno tone output

Implementation

Feature	Description
Filter	We implemented a filter in order to achieve clearer sound being output by the AMP. We applied basic principles of wave filtering, which was to eliminate abnormally high and low signal values. This was done through simple bit shifting. Code snippet below ——————————————————————— *speaker_out <= ((sound_out >> 3) << 1);* ———————————————————————
Musical Tone Generator	The corresponding musical notes are generated using a simple square wave generator with the specified frequency of the note. However, as the generated output is only 1 bit, it is amplified before being passed to the DAC. Code snippet below ——————————————————————— *module CSoundProducer(* *input clock,* *output reg [11:0] sound* *);* *reg count = 0;* *CLK_WITH_F Cslowclock (clock, 26'd191_570, slowclock);* *always @ (posedge slowclock)* *begin* *count <= ~count;* *sound <= count * 12'd4000; //amplification* *end* *endmodule* ——————————————————————— SW[7] (octave selector) is passed into the generator to determine the octave of the note. Code snippet below ——————————————————————— *assign Csound = octave ? Csound2 : Csound1;* *assign Dsound = octave ? Dsound2 : Dsound1;* *assign Esound = octave ? Esound2 : Esound1;* *assign Fsound = octave ? Fsound2 : Fsound1;* *assign Gsound = octave ? Gsound2 : Gsound1;* *assign Asound = octave ? Asound2 : Asound1;* *assign Bsound = octave ? Bsound2 : Bsound1;* ——————————————————————— I have decided to short-circuit the switches such that only 1 note will be

	played at any time. This is due to the fact that superimposing multiple tones tends to create distortion in the output sound, unlike a real musical instrument. Code snippet below _____ *assign sound = switches[0] ? Csound :* *switches[1] ? Dsound :* *switches[2] ? Esound :* *switches[3] ? Fsound :* *switches[4] ? Gsound :* *switches[5] ? Asound :* *switches[6] ? Bsound :* *MIC;* _____ In the code snippet above, it is evident that the MIC_IN signal is being passed through into the musical tone generator as well. This decision was made as it would allow us to combine multiple features to work with modifying the MIC_IN signal **and** the musical tone, such as delay.
Volume Indicator	The volume indicator measures and records the highest amplitude in a span of time, in our case it takes the records the highest amplitude every 0.25 seconds and displays the volume using the LEDs. To record the highest amplitude, the volume indicator records every new sample that the microphone takes in as seen below, where LED_ARRAY_IN represents the microphone input. _____ *always @ (LED_ARRAY_IN) begin* *MAX_SAMPLE <= (COUNT == 26'b0) ? (LED_ARRAY_IN) :* *((LED_ARRAY_IN) > MAX_SAMPLE) ? (LED_ARRAY_IN) : MAX_SAMPLE;* *end* _____ The volume indicator operates at a rate of 4Hz as seen in the code snippet below. This rate is chosen to ensure that the volume indicator minimizes the amount of flashing. Furthermore, most individuals with photosensitive epilepsy will be able to use our board without having relapses. You can refer to the link here to learn about photosensitive epilepsy. Hence, we kept our operation rate for our volume indicator as close to 3Hz as possible to ensure that it remains functional without causing epileptic shocks. As such we chose a frequency of 4Hz _____

always @ (posedge CLK_4) begin

The code snippet below shows the logic for showing the displaying the volume output. If *isLinear* is asserted, then the volume indicator will operate in pseudo linear mode. A base value of *12'd2000* is chosen since the sinusoidal sound wave oscillates at about volume = 12'd2048.

Note here the volume indicator operates in pseudo linear mode, where the user has to raise his voice by a larger margin to assert the next LED, as seen below where the increments are pseudo-linear i.e. *2000 -> 2200 -> 2450 -> 2750 etc.*This feature was implemented to model a similar idea to a logarithm function where there is a smaller rate of increase as the volume increases.

```
    LED_ARRAY[0] <= (isLinear) ? ((MAX_SAMPLE >= 12'd2000) ? 1 : 0) : LED_ARRAY_IN[5];
    LED_ARRAY[1] <= (isLinear) ? ((MAX_SAMPLE >= 12'd2200) ? 1 : 0) : LED_ARRAY_IN[6];
    LED_ARRAY[2] <= (isLinear) ? ((MAX_SAMPLE >= 12'd2450) ? 1 : 0) : LED_ARRAY_IN[7];
    LED_ARRAY[3] <= (isLinear) ? ((MAX_SAMPLE >= 12'd2750) ? 1 : 0) : LED_ARRAY_IN[8];
    LED_ARRAY[4] <= (isLinear) ? ((MAX_SAMPLE >= 12'd3100) ? 1 : 0) : LED_ARRAY_IN[9];
    LED_ARRAY[5] <= (isLinear) ? ((MAX_SAMPLE >= 12'd3500) ? 1 : 0) : LED_ARRAY_IN[10];
    LED_ARRAY[6] <= (isLinear) ? ((MAX_SAMPLE >= 12'd3950) ? 1 : 0) : LED_ARRAY_IN[11];
  end
endmodule
```

The following are the input and output descriptions of the volume indicator module:

```
module VOLUME_INDICATOR(
  input CLK,
  input CLK_4,
  input [11:0] LED_ARRAY_IN,
  input [25:0] VAL,
  input isLinear,
  output reg [6:0] LED_ARRAY
  );
```

This was how the volume indicator was used in the AUDIO_FX_TOP, as seen in the code snippet below. VOL_IN represents the input volume

	indicator while VOL_OUT represents the output volume indicator. --- *VOLUME_INDICATOR VOL_IN (clk_20k, clk_4, MIC_in, 26'd5_000, isLinear, volume_ind_in);* *VOLUME_INDICATOR VOL_OUT (clk_20k, clk_4, volume_ind_out_IN, 26'd5_000, isLinear, volume_ind_out);* --- While designing the volume indicator, we incurred the problem of the output volume indicator interfering with the quality of our sound output when *speaker_out* is assigned directly as an input the output volume indicator (probably the signals being close together resulted in more noise). As a result, I built some flip-flops in order to remove this interference to maintain the quality of sound of our device as seen below: --- *always @(posedge CLK) begin* *volume_ind_out_IN[0] = 1'b0;* *volume_ind_out_IN[1] = 1'b0;* *volume_ind_out_IN[2] = speaker_out[0];* *volume_ind_out_IN[3] = speaker_out[1];* *volume_ind_out_IN[4] = speaker_out[2];* *volume_ind_out_IN[5] = speaker_out[3];* *volume_ind_out_IN[6] = speaker_out[4];* *volume_ind_out_IN[7] = speaker_out[5];* *volume_ind_out_IN[8] = speaker_out[6];* *volume_ind_out_IN[9] = speaker_out[7];* *volume_ind_out_IN[10] = speaker_out[8];* *volume_ind_out_IN[11] = speaker_out[9];* *end* --- *volume_ind_out_IN[0] and volume_ind_out_IN[1]* are assigned zeroes since they have been removed during the filtering of our output.
Volume Delay	The volume delay uses a 2D array of registers to store samples and output the samples at a delayed rate. Firstly, samples are stored into the 2D array. New samples are stored at *memory[0]*. When there is a new sample, the previous samples will be shifted down i.e. *memory[i+1] = memory[i]* . This can be seen in the code snippet below. --- *reg [11:0] memory [0:(numSamples - 1)]; // stores samples of sample.* *reg [13:0] i; // loop counter variable* *reg [3:0] index;*

	always @ (posedge CLK) begin *for(i = 0; i < (numSamples - 1); i=i+1) begin* *memory[i+1] <= memory[i]; // shift stored samples.* *end* *memory[0] <= sample; // store new sample* *index <= ((index == 4'd0) | !isPitch) ? (pitchType ? 4'd12 : 4'd8) : (index - 2'd3);* *end* --- The volume delay feature also uses has a variable delay. If *delayType* is asserted, the delay will be approximately 0.5 seconds. Otherwise, it will be approximately 0.25 seconds. --- *assign delayed_sample = isPitch ? memory[index] : delayType ? memory[(numSamples - 1)] : memory[(numSamples / 2)]; // output delayed* ---
Pitch Shift	The volume delay module also functions as a pitch shift module. As seen in the below code snippet, an extract of the code also used in the volume delay feature, if *isPitch*, then the volume delay module will function as a pitch shift. The pitch shift module works by skipping some samples, thereby reading the samples at a faster rate, albeit losing some samples in the process. The *index* register updates differently if the *isPitch* is asserted. If *pitchType* is asserted, the rate of change is increased, as seen in the code snipet below too. --- *index <= ((index == 4'd0) | !isPitch) ? (pitchType ? 4'd12 : 4'd8) : (index - 2'd3);* *assign delayed_sample = isPitch ? memory[index] : delayType ? memory[(numSamples - 1)] : memory[(numSamples / 2)]; // output delayed* --- The pitch shift does not have a volume delay. The reason for this choice is because we decided to reduce the amount of change we had to do in order to implement the pitch shift. This is due to shortage of time and adding more modules would add to the amount of time needed to synthesis a new design. Furthermore, we implemented our machine as a DJ machine and we high pitch delayed sounds are not very common. Hence, we wanted to reduce the amount of confusion to the user and the number of buttons he had to switch on/off to enter the different modes of our device.

Hello playback	Instead of mapping it to a switch and using a button to control the play/pause state, I have decided to control the playback directly with a button only. This is to avoid confusion for the user as our other features have used almost all the switches. The button is also debounced in order to prevent accidental input. After experimentation, the ideal debouncing frequency was 10Hz for a balance of responsiveness and accidental input filtering. Code snippet of debounced signal in top level module below ——————————————————————————— *single_pulse sp (clk_10, sp_button, sp_value);* ——————————————————————————— Code snippet of debouncer below ——————————————————————————— *module single_pulse(* *input slowclock,* *input switch,* *output pulse* *);* *wire Q1;* *wire Q2;* *DFF dff_1 (slowclock, switch, Q1);* *DFF dff_2 (slowclock, Q1, Q2);* *assign pulse = Q1 && ~Q2;* *endmodule* ——————————————————————————— The data for the hello playback is stored in a ROM, which is driven at 20khz to produce the desired sound. Code snippet below ——————————————————————————— *module hello_playback(* *input CLK_20k,* *output [11:0] sound* *);* *reg[12:0] romIN = 0;* *wire[11:0] romOUT;* *always @ (posedge CLK_20k) begin* *romIN <= romIN + 1;* *end* *hello_playback_ROM rom (romIN, romOUT);* *assign sound = romOUT;* *endmodule*

I have also decided to give the hello playback precedence over other features in order to preserve the clarity of the sound. This was achieved using simple if-else logic.

Code snippet below

```
always @ (posedge CLK) begin
        if (hello_activation) begin
            speaker_out <= ((hello_soundOUT >> 3) << 1);
        end

        else if (isDelay | isPitch) begin
          speaker_out <= ((delayed >> 3) << 1);
        end

        else begin
            speaker_out <= ((sound_out >> 3) << 1);
        end
    end
```

Feedback

What did you like the most/ the least about the project	
Ryan	Kennard
Like: ● The ability to work with an FPGA to implement features and see the fruits of our effort. Dislike: ● Generating synthesis and implementation is an absolute pain. Took me 1+ hour to complete a cycle of synthesis, implementation and bitstream generation for the integrated project file.	Like: ● Working with the FPGA was interesting. It was fun to work with analog and digital conversions and implementing various subsystems incrementally. It also allowed us to apply the theories we had learnt in lectures. Dislike: ● Different classes had different sets of lecture notes. The Tuesday group had more help from their GAs and received additional information on how they could add on such as circular buffers. The Monday group did not receive this assistance. It kind of felt like we were shortchanged on the amount of stuff we could learn.

How would you suggest that the overall project assignment be improved?	
Ryan	Kennard
Allocate 1 set of modules to each student. It would allow easier collaboration as we would be able to work on the project simultaneously without having to meet up.	It would be better this was an individual project. It was difficult to work on the project as a team since it was difficult to separate work such as integration or special feature implementation equally. And a delay in someone's work can delay the other person's work. Also, we had to share one set of modules, which was not a very pleasant experience since it meant that the other person could not test his code if he does not have the necessary modules.

Any constructive feedback/suggestions are welcome.	
Ryan	Kennard
	Make the project more open-ended such that we can build whatever we wanted based on the principles that were taught in class.

As a team, working on this project was a pleasant experience. The feedback we have given above were some improvements we think could be made to the module. In general, the GAs were very friendly and helpful to our needs. We appreciate the practical lab sessions albeit the number of hours we spent synthesizing our code. We have learnt a lot through this project and have developed a deeper appreciation for hardware-level description languages.