DSP GizMo Maker Faire 2014

Digital Audio Signal Processing Fun with FPGAs

Richard Price, Altera

Introduction

I built the DSP GizMo as a way to combines art, science & fun into an entertaining yet educational 'tinker toy'. It produces cool and interesting sound effects based on user control. The user sets the GizMo operating mode by simply setting some switches, then changing the resulting sound effect by adjusting the slider controls. The following picture shows a high level block diagram of the GizMo board. On the left hand side, we have the input, or "analog control surface" . Simply put, the knobs and switches that are used to control the GizMo. Aside from the DIL switches, the signals on this board are analog. Next, we have the FPGA board that does all of the heavy-duty DSP processing. This particular board is the TerASIC DE0-Nano, which conveniently contains an analog-to-digital converter. The signals from the input board feed into the ADC inputs on the DE0-Nano FPGA board. The next board is the output board. This takes digital outputs from the FPGA & converts them back into analog signals through a digital-to-analog converter. The output board also contains a small audio amplifier with a 3.5mm headphone socket. There is also a probe point on this board to hook up an oscilloscope as a means to visualize the audio signals you are hearing.

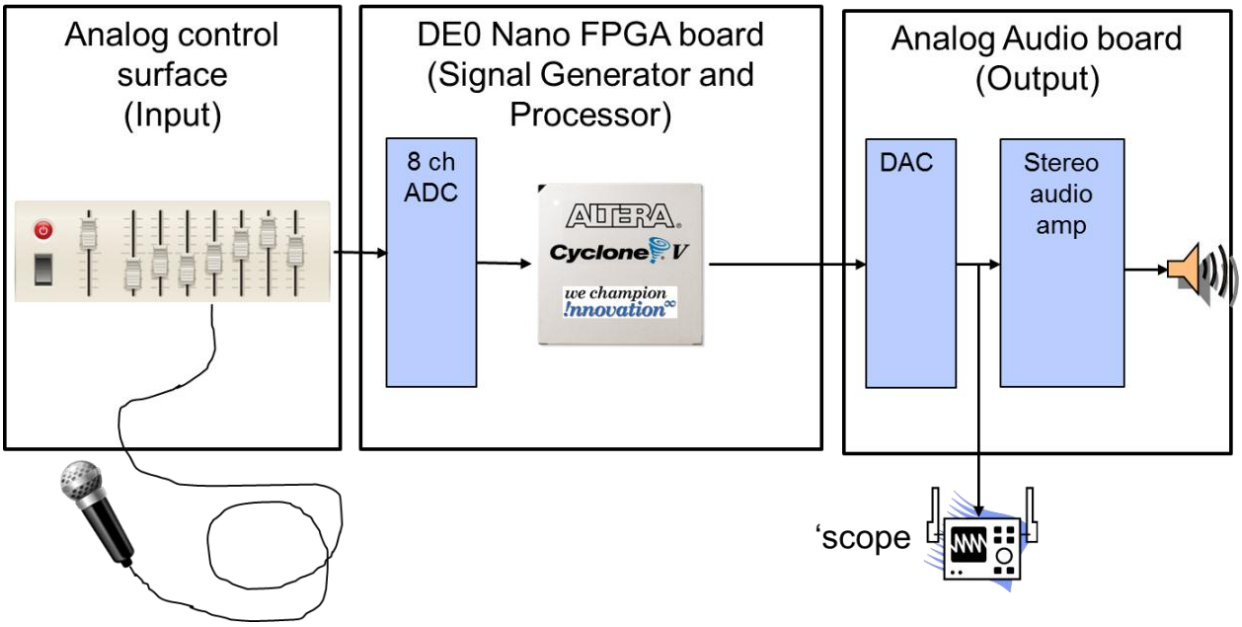


Figure 1. GizMo block diagram

Alas, in this version of the Gizmo, the microphone input is not enabled, but there is no reason why it could not be added later. The picture below shows the actual DSP Audio GizMo board implementation.

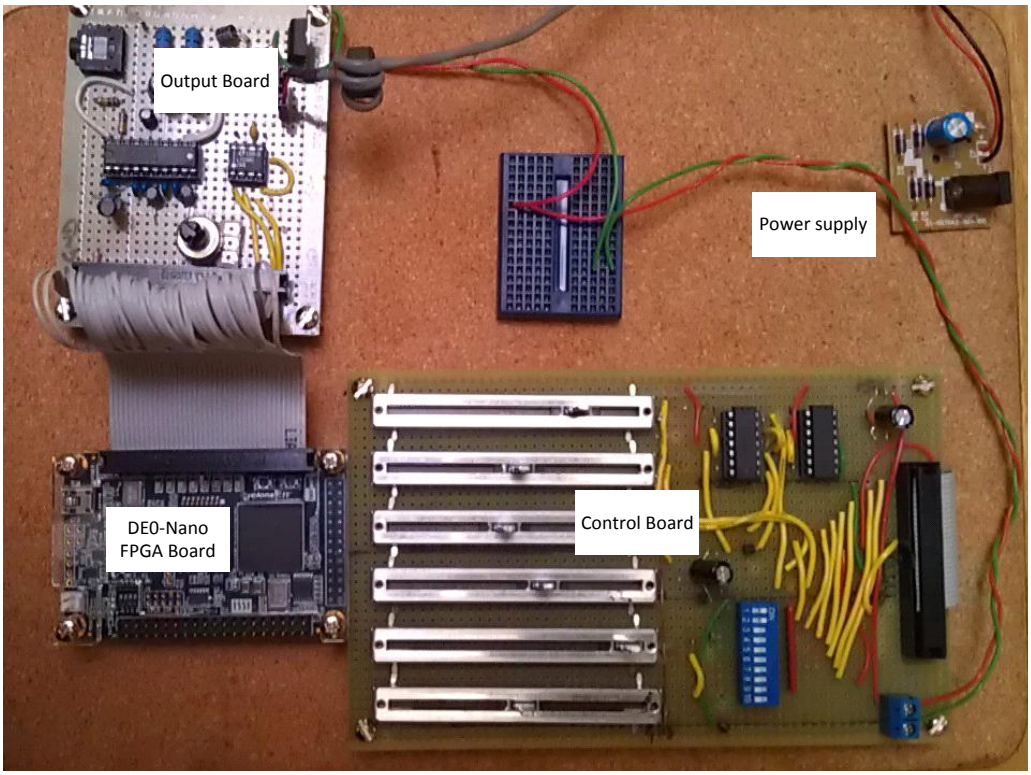


Figure 2. (above) GizMo physical implementation

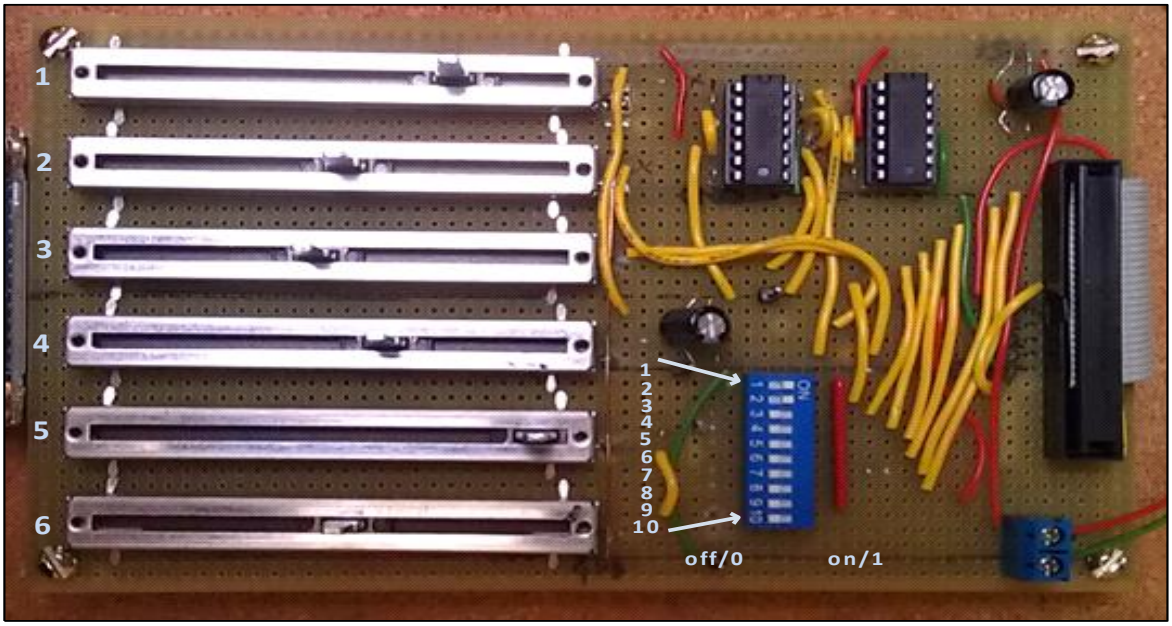


Figure 3. (above) Control board. Note the labeling of the sliders 1 through 6. '1' is at the top and '6' is at the bottom. Slider 6 is currently unused. Also note the orientation and numbering of the DIL switches. Switch '1' is at the top of the bank & '10' is at the bottom. The switches are off (ie logic 0) when moved to the left and are 'on' (ie logic 1) when moved to the right. Switches 9 & 10 are currently unused.

Experiment 8 "Whale or Haunted House?"

DIL Switch setting		1111_1111_xx (switches 1-8 = 1, switches 9-10 = don't care)
Slider	Function	
1	Frequency (F_1)	
2	-	
3	-	
4	-	
5	Volume (V_1)	
6	-	

$$F_{out} = (F_1 * V_1)$$

Discussion

Observe how the volume control affects the height of the waveform on the oscilloscope
Observe how the frequency slider affects the length of the waveform on the oscilloscope
Choose a low frequency, then measure its wavelength, Next, ask the user to move the slider until they think the frequency has doubled. Check by reading the frequency of the scope.

Experiment 7 "Haunted Whale"

DIL Switch setting		1111_1110_xx (switches 1-7 = 1, 8=0, 9-10 = don't care)
Slider	Function	
1	Frequency (F_1)	
2	Frequency (F_2)	
3	-	
4	Volume (V_2)	
5	Volume (V_1)	
6	-	

$$F_{out} = (F_1 * V_1) + (F_2 * V_2)$$

Discussion

Similar to the experiment 8, except a second tone is heard. If you set the two frequencies close to each other, you will hear the beat frequency, which is the difference between the two frequencies. Observe on the scope how the signals cancel each other out periodically. This is the underlying principle of how noise cancelling headphones work.

Experiment 6 "Sandpaper Shuffle"

DIL Switch setting		1111_1100_xx (switches 1-6 = 1, 7-8=0, 9-10 = don't care)
Slider	Function	
1	-	
2		
3	-	
4	Frequency (F_{lfo})	
5	Volume (V_{master})	
6	-	

$$F_{out} = (F_{lfo} * Noise) * V_{master}$$

Discussion

In this experiment, an LFSR circuit in the FPGA is used to generate a white noise source. A tone generator, also in the FPGA, but this time running a frequency of a few Hertz (lfo = low frequency oscillator) is multiplied with the noise signal. The result is amplitude modulation of the noise signal. If you get the frequency just right, it sounds like rubbing sandpaper on wood. Slow the LFO frequency right down, and it sounds like a steam train. On the oscilloscope you will see the magnitude of the noise signal change is a sinusoidal fashion, hence the term AM or amplitude modulation. Refer to experiment 1 to learn more about noise generation.

Experiment 5 "Stegosaurus AMMO-dulation"

DIL Switch setting		1111_1000_xx (switches 1-5 = 1, 6-8=0, 9-10 = don't care)
Slider	Function	
1	Frequency (F_1)	
2		
3	-	
4	Frequency (F_{lfo})	
5	Volume (V_{master})	
6	-	

$$F_{out} = (F_1 * F_{lfo}) * V_{master}$$

Discussion

This experiment is similar to experiment #6 except that the noise source has been replaced by a tone generator (aka Frequency generator F_1). This configuration is an example of amplitude modulation or "AM", and is the exact principle behind AM radio. In the case of radio, F_1 termed the carrier signal, is in the hundreds-of-kHz range , and F_{lfo} is the signal to be carried is in the few-kHz range, which is the sound we hear through the loudspeaker when we tune in. If you set the frequencies and volume controls just right, you can create what looks like a Stegosauru s dinosaur on the oscilloscope.

Experiment 4 "Dentist's Drill"

DIL Switch setting		1111_0000_xx (switches 1-4 = 1, 5-8=0, 9-10 = don't care)
Slider	Function	
1	Frequency (F_1)	
2	Frequency (F_2)	
3	Frequency (F_3)	
4	-	
5	Volume (V_{master})	
6	-	

$$F_{out} = (F_1+F_2+F_3) * V_{master}$$

Discussion

In this experiment, three tones are heard, and the frequency of each is controlled by an individual slider. One slider controls the master volume. If you adjust the three frequencies to make a pleasant harmonious sound you have probably created a musical chord. If you hear an unpleasant sound, you have probably created discordant frequencies. Remind you of a dentist's drill?

Experiment 3 "Distorted Phaser"

Caution: Before selecting this mode, turn the amplifier volume dow, and move all the sliders to the right. It will be loud!

DIL Switch setting		1110_0000_xx (switches 1-3 = 1, 4-8=0, 9-10 = don't care)
Slider	Function	
1	Frequency (F_1)	
2	-	
3	-	
4	-	
5	Volume (V_1)	
6	-	

$$F_{out} = F_1 * V_1 \quad \text{where} \quad (V_1 > 1)$$

Discussion

This experiment came about as a result of a mistake in the Verilog RTL code used to program the FPGA. The mistake was that the number of bits required to represent the output signal F_{out} was incorrectly specified in the code. As a result, when the signal reached its maximum positive value, it wrapped around to the maximum negative value. The result is actually a pretty interesting phaser-type sound effect. When the volume is set to quite a low level, you will hear and see a regular sine wave. As soon as the volume switch hits a point where the gain is greater than 1, distortion occurs and you can see the magnitude of the signal "wrap around" on the oscilloscope. Move the volume and frequency sliders around simultaneously to get some interesting sound effects.

Experiment 2 "PacMan Police Chase"

DIL Switch setting		1100_0000_xx (switches 1-2 = 1, 3-8=0, 9-10 = don't care)
Slider	Function	
1	Frequency (F_1)	
2	Frequency (F_2)	
3	Frequency (F_{lfo2})	
4	Frequency (F_{lof1})	
5	Volume (V_{master})	
6	-	

$$F_{out} = ((F_1 * F_{lfo1}) + (F_2 * F_{lfo2})) * V_{master}$$

Discussion

This is one of my favorites. The math looks complicated, but it's not really. In this example two audio oscillators are modulated (multiplied) by low frequency oscillators. The frequency of each audio oscillator & each low frequency oscillator (LFO) is controlled by a separate slider. If you set the sliders at the right position you can get a Police siren effect. At another position you will hear a PacMan type effect.

Experiment1 "Noise Generator"

DIL Switch setting		1000_0000_xx (switch 1= 1, 2-8=0, 9-10 = don't care)
Slider	Function	
1	-	
2	Volume (V_{master})	
3	-	
4	-	
5	-	
6	-	

$$F_{out} = Noise * V_{master}$$

Discussion

There is no frequency control in this experiment as white noise contains all frequencies simultaneously. The display on the oscilloscope shows what this random noise looks like. But in fact, the noise is not random. It is pseud-random, meaning this seemingly reason sound repeats itself after a while. But it's undetectable to the human ear. The pseudo-random sequence is defined by the following equation:

lfsr[39:1] <= {lfsr[38:1], (lfsr[18] ^ lfsr[5] ^ lfsr[1] ^ lfsr[0])};

This equation consists of a 40 bit number (lfsr[39:0] and is updated a million times a second. This means the pattern will repeat itself after 2 ^40 * 1/1,000,000 = 1099511.6 seconds or approximately 12.7 days.

Experiment 0 "Spaceship Landing"

DIL Switch setting		0000_0000_xx (switches 1-8=0, 9-10 = don't care)
Slider	Function	
1	Frequency (F_1)	
2	Frequency (F_2)	
3	Frequency (F_{mod2})	
4	Frequency (F_{mod1})	
5	Volume (V_{master})	
6	-	

$$F_{out} = ((F_1 * F_{mod1}) + (F_2 * F_{mod2})) * V_{master}$$

Discussion

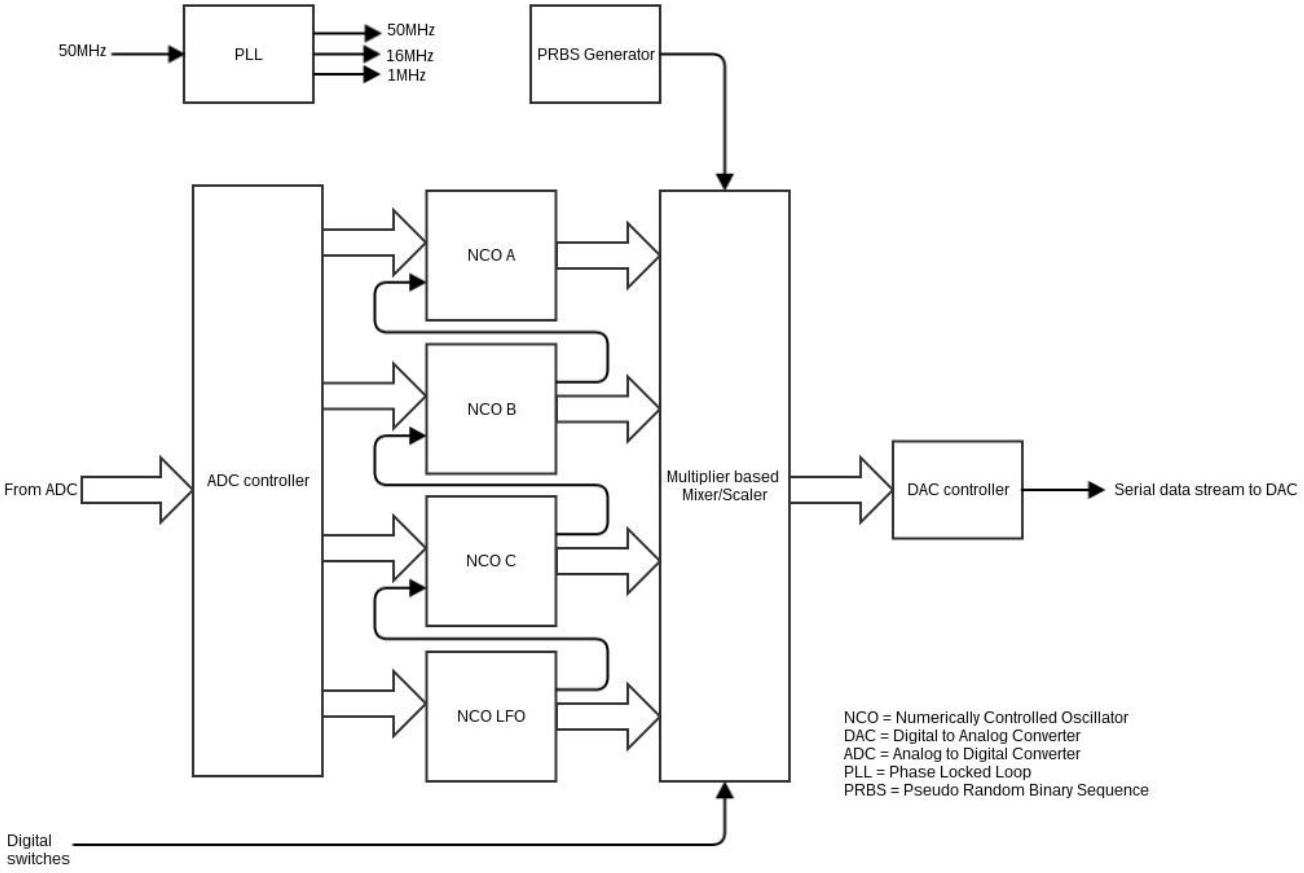
This one is similar to experiment 2, expect the low frequency oscillators are replaced with oscillators running at audio frequencies. The result is an effect that sounds like spaceships landing and taking off. That is, if you manage to move the sliders the right way.

The Flashing LEDs

Ok, I admit it, engineers love to make LEDs flash. You may have noticed that the LEDs on the DE0-Nano board flash differently depending on the mode you select. In fact they flash 'x' number of times where 'x' is the mode you select. If you happen to select an invalid mode, the LEDs will not blink. This is what the code looks like to make the LEDs flash.

```
// dsw = digital switch settings read from the control board, not the shoe store.
always @ (dsw[7:0])
  begin
    case (dsw[7:0])
      8'b0000_0000 : led6 = 1'b0;
      8'b1000_0000 : led6 = count[20] & (count[16] & count[19] & count[18] & count[17]);
      8'b1100_0000 : led6 = count[20] & (count[16] & count[19] & count[18]);
      8'b1110_0000 : led6 = count[20] & (count[16] & count[19] & (count[18] | count[17]));
      8'b1111_0000 : led6 = count[20] & (count[16] & count[19]);
      8'b1111_1000 : led6 = count[20] & ((count[16] & count[19]) | (count[16] & count[18]…
      8'b1111_1100 : led6 = count[20] & (count[16] & (count[19] | count[18]));
      8'b1111_1110 : led6 = count[20] & (count[16] & (count[19] | count[18] | count[17]));
      8'b1111_1111 : led6 = count[20] & (count[16]);
      default : led6 = 1'b1;
    endcase
  end
assign LED[5] = led6;
assign LED[4] = !led6; //just for fun, have LED[4] do the exact opposite of LED[5]
// 1MHz clock signal divided by 2^20 to get frequency in the 1-10Hz range.
always @ (posedge clk_1p0)
   count[20:0] <= count[20:0] + 1'b1;
```

FPGA DSP Processor Block Diagram



Control board schematic diagram