Christopher Spadavecchia
Eli Shtindler
CPE 487

Lab 5

Project siren:

Original Siren

After uploading the original code, headphones are displaying a triangular wave sound. The wailing speed is at a set speed. There is only one instance of this triangular wave sound in both the left and right audio channels.

Sound Level Increasing after Button Press

When pushing the button, the resulting square wave seems to be louder than the triangle wave. This makes sense as the square wave is always at maximum amplitude while the triangle wave approaches the maximum amplitude more slowly. In addition to this, the wailing speed can be seen speeding up as more switches are turned on.

Different Audios in the Left and Right Audio Channel

In the left channel the original wail is playing, while in the right channel a higher pitched wail is playing.

siren.vhd

Original:

```
library IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY siren IS
	PORT (
		clk_50MHz : IN STD_LOGIC; -- system clock (50 MHz)
		dac_MCLK : OUT STD_LOGIC; -- outputs to PMODI2L DAC
		dac_LRCK : OUT STD_LOGIC;
		dac_SCLK : OUT STD_LOGIC;
		dac_SDIN : OUT STD_LOGIC
	);
END siren;

ARCHITECTURE Behavioral OF siren IS
	CONSTANT lo_tone : UNSIGNED (13 DOWNTO 0) := to_unsigned (344, 14); -- lower limit of
siren = 256 Hz
```

```
	CONSTANT hi_tone : UNSIGNED (13 DOWNTO 0) := to_unsigned (687, 14); -- upper limit of
siren = 512 Hz
	CONSTANT wail_speed : UNSIGNED (7 DOWNTO 0) := to_unsigned (8, 8); -- sets wailing
speed
	COMPONENT dac_if IS
		PORT (
			SCLK : IN STD_LOGIC;
			L_start : IN STD_LOGIC;
			R_start : IN STD_LOGIC;
			L_data : IN signed (15 DOWNTO 0);
			R_data : IN signed (15 DOWNTO 0);
			SDATA : OUT STD_LOGIC
		);
	END COMPONENT;
	COMPONENT wail IS
		PORT (
			lo_pitch : IN UNSIGNED (13 DOWNTO 0);
			hi_pitch : IN UNSIGNED (13 DOWNTO 0);
			wspeed : IN UNSIGNED (7 DOWNTO 0);
			wclk : IN STD_LOGIC;
			audio_clk : IN STD_LOGIC;
			audio_data : OUT SIGNED (15 DOWNTO 0)
		);
	END COMPONENT;
	SIGNAL tcount : unsigned (19 DOWNTO 0) := (OTHERS => '0'); -- timing counter
	SIGNAL data_L, data_R : SIGNED (15 DOWNTO 0); -- 16-bit signed audio data
	SIGNAL dac_load_L, dac_load_R : STD_LOGIC; -- timing pulses to load DAC shift reg.
	SIGNAL slo_clk, sclk, audio_CLK : STD_LOGIC;
BEGIN
	-- this process sets up a 20 bit binary counter clocked at 50MHz. This is used
	-- to generate all necessary timing signals. dac_load_L and dac_load_R are pulses
	-- sent to dac_if to load parallel data into shift register for serial clocking
	-- out to DAC
	tim_pr : PROCESS
	BEGIN
		WAIT UNTIL rising_edge(clk_50MHz);
		IF (tcount(9 DOWNTO 0) >= X"00F") AND (tcount(9 DOWNTO 0) < X"02E") THEN
			dac_load_L <= '1';
		ELSE
			dac_load_L <= '0';
		END IF;
		IF (tcount(9 DOWNTO 0) >= X"20F") AND (tcount(9 DOWNTO 0) < X"22E") THEN
			dac_load_R <= '1';
		ELSE dac_load_R <= '0';
		END IF;
		tcount <= tcount + 1;
	END PROCESS;
	dac_MCLK <= NOT tcount(1); -- DAC master clock (12.5 MHz)
	audio_CLK <= tcount(9); -- audio sampling rate (48.8 kHz)
	dac_LRCK <= audio_CLK; -- also sent to DAC as left/right clock
	sclk <= tcount(4); -- serial data clock (1.56 MHz)
	dac_SCLK <= sclk; -- also sent to DAC as SCLK
```

```
	slo_clk <= tcount(19); -- clock to control wailing of tone (47.6 Hz)
	dac : dac_if
	PORT MAP(
		SCLK => sclk, -- instantiate parallel to serial DAC interface
		L_start => dac_load_L,
		R_start => dac_load_R,
		L_data => data_L,
		R_data => data_R,
		SDATA => dac_SDIN
		);
		w1 : wail
		PORT MAP(
			lo_pitch => lo_tone, -- instantiate wailing siren
			hi_pitch => hi_tone,
			wspeed => wail_speed,
			wclk => slo_clk,
			audio_clk => audio_clk,
			audio_data => data_L
		);
		data_R <= data_L; -- duplicate data on right channel
END Behavioral;
```

Modified:

```
library IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY siren IS
	PORT (
		clk_50MHz : IN STD_LOGIC; -- system clock (50 MHz)
		BTNU : IN STD_LOGIC;
		SW : IN UNSIGNED (7 DOWNTO 0);
		dac_MCLK : OUT STD_LOGIC; -- outputs to PMODI2L DAC
		dac_LRCK : OUT STD_LOGIC;
		dac_SCLK : OUT STD_LOGIC;
		dac_SDIN : OUT STD_LOGIC
	);
END siren;

ARCHITECTURE Behavioral OF siren IS
	CONSTANT lo_tone : UNSIGNED (13 DOWNTO 0) := to_unsigned (344, 14); -- lower limit of
siren = 256 Hz
	CONSTANT hi_tone : UNSIGNED (13 DOWNTO 0) := to_unsigned (687, 14); -- upper limit of
siren = 512 Hz
	CONSTANT wail_speed : UNSIGNED (7 DOWNTO 0) := to_unsigned (8, 8); -- sets wailing
speed
	COMPONENT dac_if IS
		PORT (
			SCLK : IN STD_LOGIC;
			L_start : IN STD_LOGIC;
			R_start : IN STD_LOGIC;
```

```
                        L_data : IN signed (15 DOWNTO 0);
                        R_data : IN signed (15 DOWNTO 0);
                        SDATA : OUT STD_LOGIC
                );
        END COMPONENT;
        COMPONENT wail IS
                PORT (
                        lo_pitch : IN UNSIGNED (13 DOWNTO 0);
                        hi_pitch : IN UNSIGNED (13 DOWNTO 0);
                        wspeed : IN UNSIGNED (7 DOWNTO 0);
                        wclk : IN STD_LOGIC;
                        audio_clk : IN STD_LOGIC;
                        button : IN STD_LOGIC;
                        audio_data : OUT SIGNED (15 DOWNTO 0)
                );
        END COMPONENT;
        SIGNAL tcount : unsigned (19 DOWNTO 0) := (OTHERS => '0'); -- timing counter
        SIGNAL data_L, data_R : SIGNED (15 DOWNTO 0); -- 16-bit signed audio data
        SIGNAL dac_load_L, dac_load_R : STD_LOGIC; -- timing pulses to load DAC shift reg.
        SIGNAL slo_clk, sclk, audio_CLK : STD_LOGIC;
BEGIN
        -- this process sets up a 20 bit binary counter clocked at 50MHz. This is used
        -- to generate all necessary timing signals. dac_load_L and dac_load_R are pulses
        -- sent to dac_if to load parallel data into shift register for serial clocking
        -- out to DAC
        tim_pr : PROCESS
        BEGIN
                WAIT UNTIL rising_edge(clk_50MHz);
                IF (tcount(9 DOWNTO 0) >= X"00F") AND (tcount(9 DOWNTO 0) < X"02E") THEN
                        dac_load_L <= '1';
                ELSE
                        dac_load_L <= '0';
                END IF;
                IF (tcount(9 DOWNTO 0) >= X"20F") AND (tcount(9 DOWNTO 0) < X"22E") THEN
                        dac_load_R <= '1';
                ELSE dac_load_R <= '0';
                END IF;
                tcount <= tcount + 1;
        END PROCESS;
        dac_MCLK <= NOT tcount(1); -- DAC master clock (12.5 MHz)
        audio_CLK <= tcount(9); -- audio sampling rate (48.8 kHz)
        dac_LRCK <= audio_CLK; -- also sent to DAC as left/right clock
        sclk <= tcount(4); -- serial data clock (1.56 MHz)
        dac_SCLK <= sclk; -- also sent to DAC as SCLK
        slo_clk <= tcount(19); -- clock to control wailing of tone (47.6 Hz)
        dac : dac_if
        PORT MAP(
                SCLK => sclk, -- instantiate parallel to serial DAC interface
                L_start => dac_load_L,
                R_start => dac_load_R,
                L_data => data_L,
                R_data => data_R,
```

```
                SDATA => dac_SDIN
    );
    w1 : wail
    PORT MAP(
        lo_pitch => lo_tone, -- instantiate wailing siren
        hi_pitch => hi_tone,
        wspeed => SW, -- Set the wail speed to the switch input
        wclk => slo_clk,
        audio_clk => audio_clk,
        button => BTNU,
        audio_data => data_L
    );

    w2 : wail
    PORT MAP(
        lo_pitch => TO_UNSIGNED(1000, 14), -- instantiate wailing siren
        hi_pitch => TO_UNSIGNED(2000, 14),
        wspeed => TO_UNSIGNED(5, 8), -- Set the wail speed to the switch input
        wclk => slo_clk,
        audio_clk => audio_clk,
        button => BTNU,
        audio_data => data_R
    );

END Behavioral;
```

For part A of the lab, the BTNU signal was just passed from siren down to wail and then tone. Most of the logic is implemented in tone.vhd. For part B, the switches were added into the constraint file and as an input to the siren. The value of the first 8 switches was simply read as a binary digit and passed as the wail speed to w1. For part C a new module, w2, was created with a low pitch of 1000, high pitch of 2000, and wail speed of 5. The output was set to the right channel instead of the left.

tone.vhd

Original:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;

-- Generates a 16-bit signed triangle wave sequence at a sampling rate determined
-- by input clk and with a frequency of (clk*pitch)/65,536
ENTITY tone IS
       PORT (
              clk : IN STD_LOGIC; -- 48.8 kHz audio sampling clock
              pitch : IN UNSIGNED (13 DOWNTO 0); -- frequency (in units of 0.745 Hz)
       data : OUT SIGNED (15 DOWNTO 0)); -- signed triangle wave out
END tone;
```

```
ARCHITECTURE Behavioral OF tone IS
	SIGNAL count : unsigned (15 DOWNTO 0); -- represents current phase of waveform
	SIGNAL quad : std_logic_vector (1 DOWNTO 0); -- current quadrant of phase
	SIGNAL index : signed (15 DOWNTO 0); -- index into current quadrant
BEGIN
	-- This process adds "pitch" to the current phase every sampling period. Generates
	-- an unsigned 16-bit sawtooth waveform. Frequency is determined by pitch. For
	-- example when pitch=1, then frequency will be 0.745 Hz. When pitch=16,384,
frequency
	-- will be 12.2 kHz.
	cnt_pr : PROCESS
	BEGIN
		WAIT UNTIL rising_edge(clk);
		count <= count + pitch;
	END PROCESS;
	quad <= std_logic_vector (count (15 DOWNTO 14)); -- splits count range into 4 phases
	index <= signed ("00" & count (13 DOWNTO 0)); -- 14-bit index into the current phase
	-- This select statement converts an unsigned 16-bit sawtooth that ranges from 65,535
	-- into a signed 12-bit triangle wave that ranges from -16,383 to +16,383
	WITH quad SELECT
	data <= index WHEN "00", -- 1st quadrant
		16383 - index WHEN "01", -- 2nd quadrant
		0 - index WHEN "10", -- 3rd quadrant
		index - 16383 WHEN OTHERS; -- 4th quadrant
END Behavioral;
```

Modified:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;

-- Generates a 16-bit signed triangle wave sequence at a sampling rate determined
-- by input clk and with a frequency of (clk*pitch)/65,536
ENTITY tone IS
	PORT (
		clk : IN STD_LOGIC; -- 48.8 kHz audio sampling clock
		button : IN STD_LOGIC;
	    pitch : IN UNSIGNED (13 DOWNTO 0); -- frequency (in units of 0.745 Hz)
	data : OUT SIGNED (15 DOWNTO 0)); -- signed triangle wave out
END tone;

ARCHITECTURE Behavioral OF tone IS
	SIGNAL count : unsigned (15 DOWNTO 0); -- represents current phase of waveform
	SIGNAL quad : std_logic_vector (1 DOWNTO 0); -- current quadrant of phase
	SIGNAL index : signed (15 DOWNTO 0); -- index into current quadrant
BEGIN
	-- This process adds "pitch" to the current phase every sampling period. Generates
	-- an unsigned 16-bit sawtooth waveform. Frequency is determined by pitch. For
	-- example when pitch=1, then frequency will be 0.745 Hz. When pitch=16,384,
```

```
frequency
        -- will be 12.2 kHz.
        cnt_pr : PROCESS
        BEGIN
            WAIT UNTIL rising_edge(clk);
                count <= count + pitch;
        END PROCESS;
        quad <= std_logic_vector (count (15 DOWNTO 14)); -- splits count range into 4 phases
        index <= signed ("00" & count (13 DOWNTO 0)); -- 14-bit index into the current phase
        -- This select statement converts an unsigned 16-bit sawtooth that ranges from 65,535
        -- into a signed 12-bit triangle wave that ranges from -16,383 to +16,383
        WITH (quad & button) SELECT data <=
           index WHEN "000", -- 1st quadrant
           16383 - index WHEN "010", -- 2nd quadrant
           0 - index WHEN "100", -- 3rd quadrant
           index - 16383 WHEN "110", -- 4th quadrant
           to_signed(16383, 16) WHEN ("001" OR "011"),
           to_signed(-16383, 16) WHEN ("101" OR "111"),
           to_signed(0, 16) WHEN OTHERS;

END Behavioral;
```

In order to generate a square wave, the select statement at the end of the file was modified. Instead of just using quad, it uses quad and button. When button = 0, the select statement acts the same as before, outputting a square wave. When button = 1, the select statement outputs 16383 in quadrants 0 and 1, and -16383 in quadrants 2 and 3. This creates a square wave with the same frequency and max amplitudes as the original triangle wave.