```
/*********************************

part1.sv

*********************************/

/*
 * Connor Aksama
 * 02/23/2023
 * CSE 371
 * Lab 5
*/

/**
 * Top-level module for Lab 5

 * Inputs:
 *      CLOCK_50 [1 bit] - clock to use for this system
 *      CLOCK2_50 [1 bit] - clock to use for clock generator
 *      KEY [1 bit] - Onboard KEY0 input signal
 *      SW [10 bit] - Onboard SW input signals
 *      AUD_DACLRCK [1 bit] - Audio CODEC interface signal
 *      AUD_ADCLRCK [1 bit] - Audio CODEC interface signal
 *      AUD_BCLK [1 bit] - Audio CODEC interface signal
 *      AUD_ADCDAT [1 bit] - Audio CODEC interface signal
 * Inouts:
 *      FPGA_I2C_SDAT [1 bit] - Audio CODEC interface signal
 * Outputs:
 *      FPGA_12C_SCLK [1 bit] - Audio CODEC interface signal
 *      AUD_XCK [1 bit] - Audio CODEC interface signal
 *      AUD_DACDAT [1 bit] - Audio CODEC interface signal
*/
module part1 (CLOCK_50, CLOCK2_50, KEY, SW, FPGA_I2C_SCLK, FPGA_I2C_SDAT, AUD_XCK,
                AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK, AUD_ADCDAT, AUD_DACDAT);

    input CLOCK_50, CLOCK2_50;
    input [0:0] KEY;
    input [9:0] SW;
    // I2C Audio/Video config interface
    output FPGA_I2C_SCLK;
    inout FPGA_I2C_SDAT;
    // Audio CODEC
    output AUD_XCK;
    input AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK;
    input AUD_ADCDAT;
    output AUD_DACDAT;

    // Local wires.
    wire read_ready, write_ready, read, write;
    wire [23:0] readdata_left, readdata_right;
    reg [23:0] writedata_left, writedata_right;
    wire reset = ~KEY[0];

    ////////////////////////////////
    // Your code goes here
    ////////////////////////////////

    wire [23:0] tone_data;

    // Tone sample generator
    // tone_data - current tone sample
    // write - signal to get next sample from ROM
    // CLOCK_50 - clk to use for generator
    // reset - reset the generator to the starting sample
    tone_generator gen (
         tone_data
        ,write
        ,CLOCK_50
        ,reset
```

```
70      );
71
72      // Mux the write data to the CODEC based on SW9
73      always @(tone_data, readdata_left, readdata_right, SW[9]) begin
74          if (SW[9]) begin
75              writedata_left = tone_data;
76              writedata_right = tone_data;
77          end else begin
78              writedata_left = readdata_left;
79              writedata_right = readdata_right;
80          end
81      end
82
83      assign read = read_ready;
84      assign write = write_ready & (read_ready | SW[9]);
85
86  /////////////////////////////////////////////////////////////////////////////
87  // Audio CODEC interface.
88  //
89  // The interface consists of the following wires:
90  // read_ready, write_ready - CODEC ready for read/write operation
91  // readdata_left, readdata_right - left and right channel data from the CODEC
92  // read - send data from the CODEC (both channels)
93  // writedata_left, writedata_right - left and right channel data to the CODEC
94  // write - send data to the CODEC (both channels)
95  // AUD_* - should connect to top-level entity I/O of the same name.
96  //         These signals go directly to the Audio CODEC
97  // I2C_* - should connect to top-level entity I/O of the same name.
98  //         These signals go directly to the Audio/Video Config module
99  /////////////////////////////////////////////////////////////////////////////
100     clock_generator my_clock_gen(
101         // inputs
102         CLOCK2_50,
103         reset,
104
105         // outputs
106         AUD_XCK
107     );
108
109     audio_and_video_config cfg(
110         // Inputs
111         CLOCK_50,
112         reset,
113
114         // Bidirectionals
115         FPGA_I2C_SDAT,
116         FPGA_I2C_SCLK
117     );
118
119     audio_codec codec(
120         // Inputs
121         CLOCK_50,
122         reset,
123
124         read,   write,
125         writedata_left, writedata_right,
126
127         AUD_ADCDAT,
128
129         // Bidirectionals
130         AUD_BCLK,
131         AUD_ADCLRCK,
132         AUD_DACLRCK,
133
134         // Outputs
135         read_ready, write_ready,
136         readdata_left, readdata_right,
137         AUD_DACDAT
138     );
```

```
139
140 endmodule
141
142 /**********************************
143
144 tone_generator.sv
145
146 **********************************/
147
148 /*
149  * Connor Aksama
150  * 02/23/2023
151  * CSE 371
152  * Lab 5
153 */
154
155 /**
156  * Module to output sound data to generate a tone.
157
158  * Inputs:
159  *      write [1 bit] - 1 if the next sample should be output on this cycle, 0 o.w.
160  *      clk [1 bit] - the clock to use for this module
161  *      reset [1 bit] - resets the module to the first sample in the cycle if 1
162
163  * Outputs:
164  *      tone_data [24 bit] - the current sample for the tone
165 */
166 module tone_generator (
167     output [23:0] tone_data
168     ,input write, clk, reset
169     );
170
171     localparam num_samples = 520;
172
173     reg [9:0] rom_addr;
174     wire [23:0] rom_data;
175
176     // ROM storing tone samples
177     // rom_addr - address to read from
178     // clk - clock to use
179     // tone_data - output data
180     lab5rom rom (
181          rom_addr
182         ,clk
183         ,tone_data
184     );
185
186     always @(posedge clk) begin
187         if (reset || (rom_addr == num_samples - 1)) begin
188             // Reset to 0 on reset or end of samples reached
189             rom_addr <= 0;
190         end else if (write) begin
191             // Read from next address
192             rom_addr <= rom_addr + 1;
193         end
194     end
195
196 endmodule  // tone_generator
197
198
199 /*
200  * Testbench to test the functionality of the tone_generator module
201  */
202 `timescale 1 ps / 1 ps
203 module tone_generator_testbench();
204
205     logic [23:0] tone_data;
206     logic write, clk, reset;
207
```

```
208         tone_generator dut (.*);
209 
210         parameter CLOCK_PERIOD = 100;
211         initial begin
212             integer i;
213             clk <= 0;
214             forever #(CLOCK_PERIOD / 2) clk <= ~clk;
215         end
216 
217         initial begin
218             integer i;
219             @(posedge clk) reset <= 1'b1;
220             @(posedge clk) reset <= 1'b0; write <= 1'b1;
221             // Get every sample in the ROM and cycle back
222             for (i = 0; i < 530; i++) begin
223                 @(posedge clk);
224             end
225             // Don't read anything
226             @(posedge clk) write <= 1'b0;
227             for (i = 0; i < 10; i++) begin
228                 @(posedge clk);
229             end
230             $stop;
231         end
232 
233     endmodule  //tone_generator_testbench
234 
```