4-State Sound Visualizer with Alarm System

Elif Duru Akgül

EE102-02

May 16, 2025

1. **Purpose:**

This project aims to implement a 4-state led sound visualizer in VHDL, using Basys 3 FGPA board. It has an FSM system which includes 4 states, NORMAL, ALARM, ADAPTIVE_NORMAL, ADAPTIVE_ALARM. According to these states, the data from a microphone and a lights sensor is correspondingly displayed on a 7-segment display and a 10-segment LED bar graph. There is also an alarm system that rings when a certain sound level is reached.

2. **Methodology:**

The microphone module, MAX4466, has an implemented amplifier module inside and gives analog output. Therefore, to read data from that module an ADC module is required. I used the Basys 3 board's third PMOD's and Xilinx's XADC Wizard to be able to read this analog data. The channels vauxn6, vauxp6, and the sequential mode is used in XADC wizard, which provided a complete ADC module. The data that derives from the microphone, xadc_data_raw, is processed through peak-to-peak calculations afterwards, in top_module.

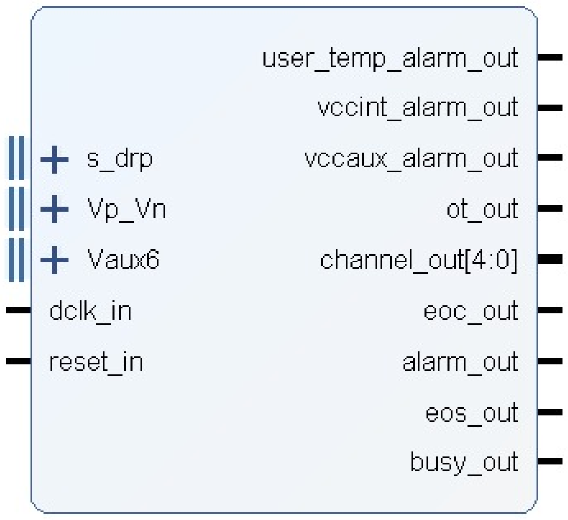

Figure 1: XADC Wizard

XADC interface signals:

```
component xadc_wiz_0
    port (
        daddr_in : in std_logic_vector(6 downto 0);
        dclk_in : in std_logic;
        den_in : in std_logic;
        di_in : in std_logic_vector(15 downto 0);
        dwe_in : in std_logic;
        do_out : out std_logic_vector(15 downto 0);
        drdy_out : out std_logic; vp_in : in std_logic;
        vn_in : in std_logic;
        vauxp6 : in std_logic;
        vauxn6 : in std_logic
     );
end component;
```

An extra FSM controller module is utilized, which manages the overall behaviour of the project. The states changes are determined by the microphone and lights sensor inputs. The sound level is represented via a bar-graph made of LEDs. The LEDs' intensity visualizes the sound level, also the sound level's numerical value or status message ("HIGH" or "LL") are shown on the 7-segment display of Basys 3 board. An audible alert is provided by a buzzer. In the ALARM and ADAPTIVE_ALARM states, this buzzer is triggered.

3. Design Specifications:

Firstly, for sound measurement, the XADC performs continuous sampling. The microphone signal is divided into smaller sampling windows, 50ms, each comprising 5 million clock cycles at 100MHz clock rate. During that each sample windows, peak-to-peak signal is computed, as explained before. This peak_to_peak signal is then, displayed on the 7-segment display (scaled from 0 to 9999) when at NORMAL state.

Secondly, four modes managed by an FSM is operated. In NORMAL mode (light_sensor = "1", sound_int < 9000), system displays current sound level numerically on 7 segment display and demonstrates the intensity on led bar graph, buzzer isn't triggered. When the threshold of 9000 is exceeded, system transitions into the ALARM mode (light_sensor = "1", sound_int > 9000). In ALARM mode, instead of showing the sound level numerically on 7 segment display, the message "HIGH" occurs. The buzzer is triggered which creates a continuous ringing, and led bar graph starts flashing.

When light_sensor = "0", adaptive modes activate. When in ADAPTIVE_NORMAL mode sound level is under 5000, while in ADAPTIVE_ALARM mode this threshold is exceeded. In both of these modes the message "LL" is displayed on 7 segment display. In ADAPTIVE_NORMAL mode, the intensity is shown on led bar graph, no buzzer activated. Whereas in ADAPTIVE_ALARM mode, instead of a continuously ringing alarm, the buzzer pulses as one-shot, and led bar graph flashes.

	NORMAL	ALARM	ADAPTIVE_NORMAL	ADAPTIVE_ALARM
Light sensor	1	1	0	0
Sound level	<9000	>9000	<5000	>5000
7 segment	Numerical value	“HIGH”	“LL”	“LL”
Led bar graph	Intensity of sound level	Flashing	Intensity of sound level	Flashing
Buzzer	-	Continuous buzzing	-	Pulsing

Then, the timing is regulated through the clock operating 100MHz. The display refreshing logic uses a clock divider to 1kHz, which refreshes each digit at approximately 250Hz to avoid flickering. The buzzer’s pulse durations and the sampling window are both defined in clock cycle terms, 10 million for the buzzer pulse (100ms) and 5 million for sampling (50ms).

Lastly, The project is divided into three modules: top_module, fsm_controller, and led_bargraph. The top_module integrates all subsystems. It handles XADC configuration and interface, manages sound level calculations, and coordinates the displays and the alarm system. Additionally, the buzzer system behaviour is defined here. The fsm_controller module governs between the states, and the led_bargraph module visualizes the intensity of sound level in a 10-LED array, which also supports flashing.

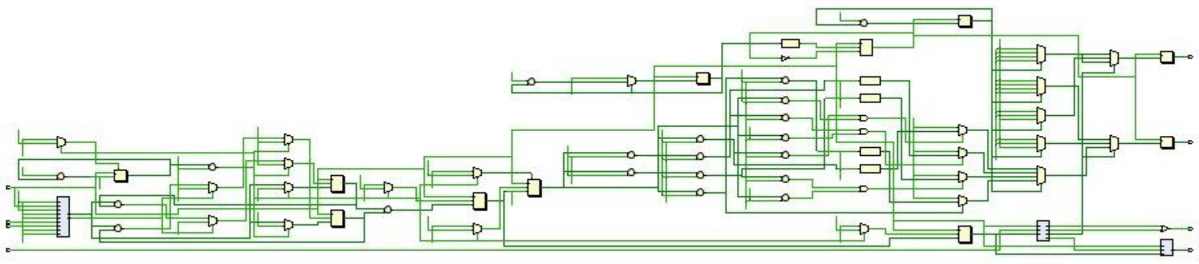

Figure 3: The Elaborated Design Schematic

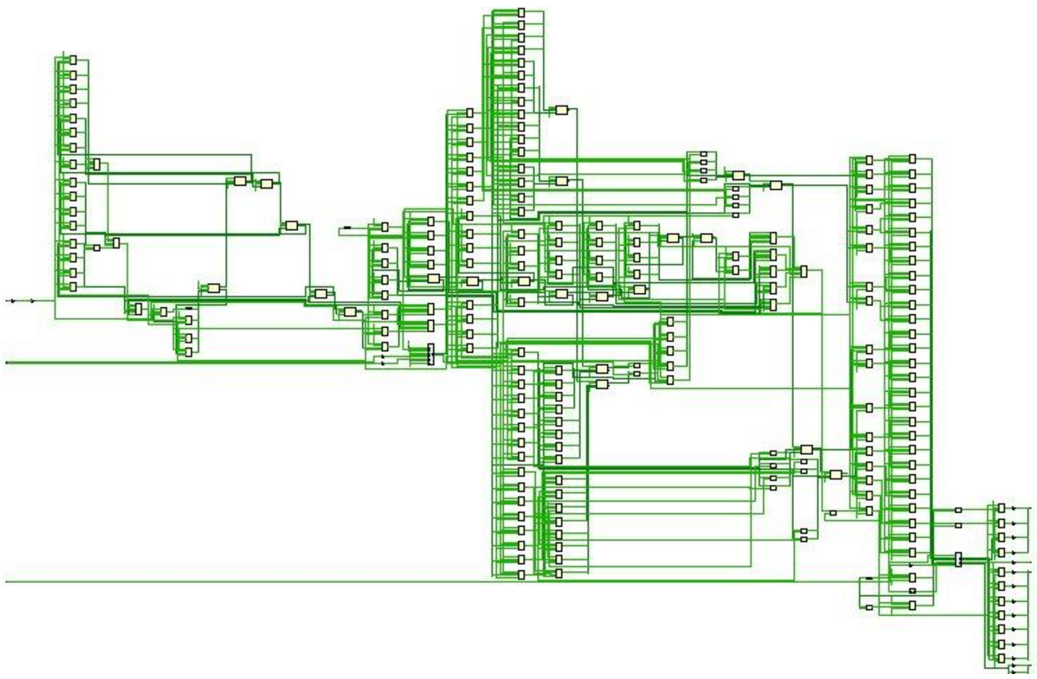

Figure 2: The Synthesised Design Schematic

4. Results:

For every sound level, the results were as expected. Also, according to the light that got caught by the LDR, states changed. Figures show the results.

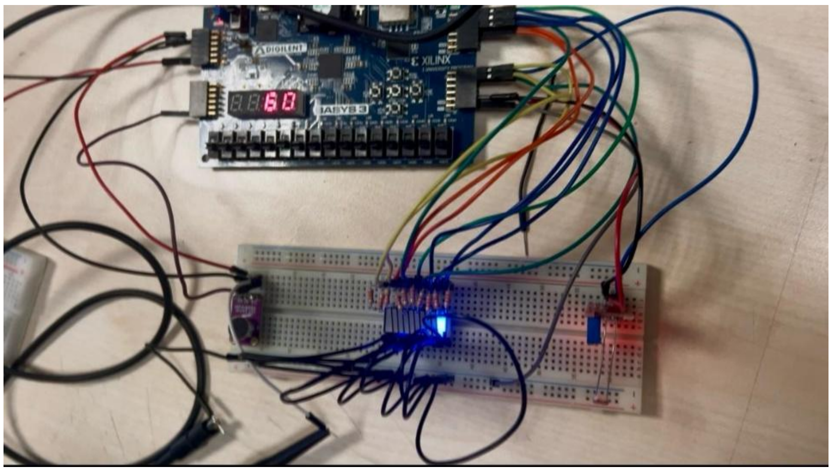

Figure 4: NORMAL STATE

- Microphone value is below 9000.
- Light is on. (light_sensor = “1”)
- Sound level is demonstrated numerically on 7 segment display
- Sound intensity is shown in led bar graph (low at 60, blue only)
- Buzzer is not triggered.

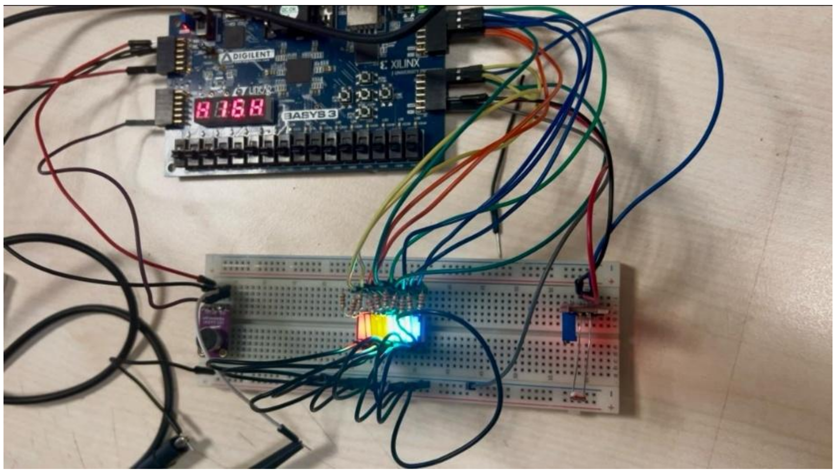

Figure 5: ALARM STATE

- Microphone value is above 9000
- Light is on (light_sensor = “1”)
- The message “HIGH” is demonstrated in 7 segment display.
- Led bar graph flashes.
- Buzzer keeps emitting sound continuously.

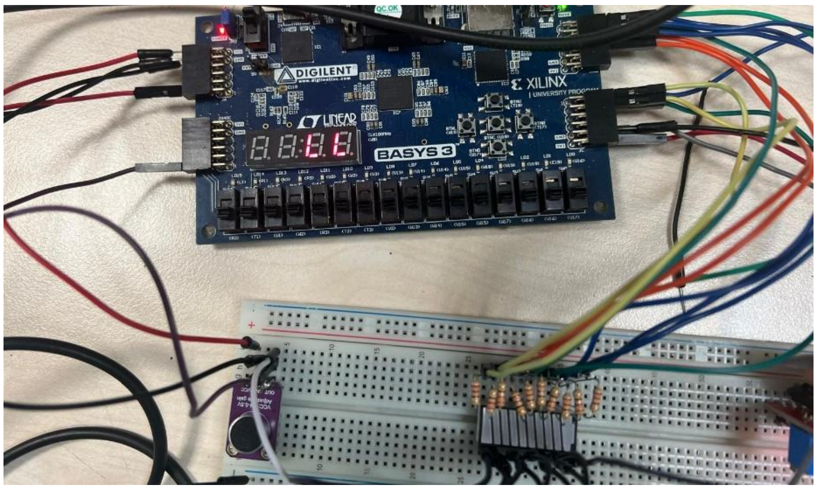

Figure 6: ADAPTIVE_NORMAL STATE

- Microphone value is below 5000
- Light is off (light_sensor = “0”)
- The message “LL” is demonstrated in 7 segment display.
- Led bar graph shows the intensity (in this case no sound).
- Buzzer is not triggered.

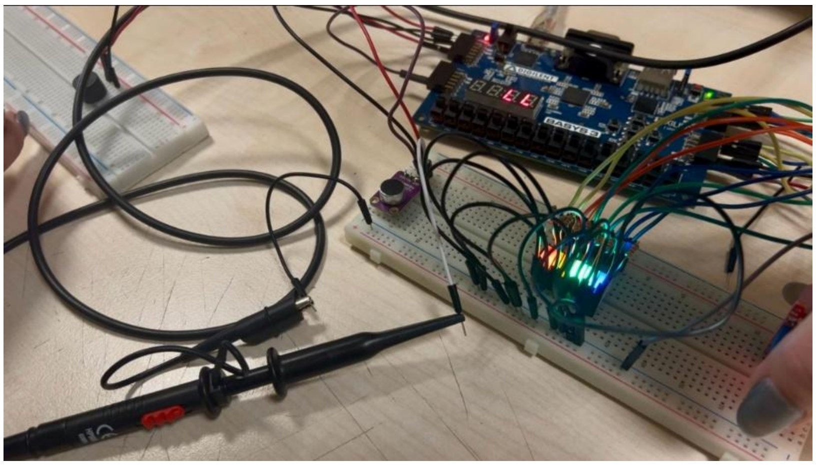

Figure 7: ADAPTIVE_ALARM STATE

- Microphone value is above 5000
- Light is off (light_sensor = “0”)
- The message “LL” is demonstrated in 7 segment display.
- Led bar graph flashes.
- Buzzer emits a one-shot pulse.

5. Conclusion:

The features of this project, such as adaptive sensibility, integrated FSM states, and modifying displays, can serve as a regulatory system across some environments where sound monitoring is critical. For instance, music venues, libraries and even hospitals. In loud settings, it can monitor the peak noise levels and provide visual and audible alerts to ensure safe audio levels. In quieter, low-light settings, it becomes more sensitive and uses discreet alerts to maintain the peace, without causing disruption. Its adaptability makes it ideal for variety of environments. In general, this system is a useful tool for real-time sound monitoring and control since it shows how integrated solutions can react to environmental changes intelligently. The basis for wider applications in public safety, health, and smart building technologies is laid by the possibility of additional improvements such data logging and frequency analysis.

6. **YouTube Link**:

https://youtu.be/-jB-Lq_E15U?si=XMz2qZ-8KqDelKPa

7. References:

• Digilent Inc. *Basys 3 Reference Manual*. Retrieved from https://digilent.com/reference/_media/basys3:basys3_rm.pdf

• AMD Inc. *XADC Wizard v1.3 (PG091)*. Retrieved from https://docs.amd.com/v/u/en-US/pg091-xadc-wiz

• Maxim Integrated. *MAX4466 Low-Cost, Micropower, Rail-to-Rail Output Op Amps* [Data sheet]. Retrieved from https://www.alldatasheet.com/datasheet-pdf/view/73367/MAXIM/MAX4466.html

8. **Appendices**:

Top Module:

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;


entity top_module is

  port (

    CLK100MHZ    : in std_logic;

    SEG          : out std_logic_vector(6 downto 0);

    AN           : out std_logic_vector(3 downto 0);

    buzzer       : out std_logic;

    leds         : out std_logic_vector(9 downto 0);

    vauxp6       : in std_logic;

    vauxn6       : in std_logic;

    light_sensor : in std_logic

  );

end top_module;


architecture Behavioral of top_module is


  function digit_to_segment(d: integer) return std_logic_vector is

  begin

    case d is
```

```
        when 0 => return "1000000";
        when 1 => return "1111001";
        when 2 => return "0100100";
        when 3 => return "0110000";
        when 4 => return "0011001";
        when 5 => return "0010010";
        when 6 => return "0000010";
        when 7 => return "1110000";
        when 8 => return "0000000";
        when 9 => return "0010000";
        when others => return "1111111";
    end case;
end function;

signal xadc_data_raw : std_logic_vector(15 downto 0);
signal drdy          : std_logic;
signal daddr         : std_logic_vector(6 downto 0) := "0010110";
signal den           : std_logic := '1';
signal dwe           : std_logic := '0';
signal dummy_di      : std_logic_vector(15 downto 0) := (others => '0');

signal clk_divider   : natural := 0;
signal segment_clock : std_logic := '0';
signal anode_div     : std_logic_vector(1 downto 0) := "00";
signal value         : natural := 0;
signal signal_max      : unsigned(15 downto 0) := (others => '0');
signal signal_min      : unsigned(15 downto 0) := (others => '1');
signal peak_to_peak    : unsigned(15 downto 0) := (others => '0');
signal sample_counter  : natural := 0;
constant SAMPLE_WINDOW : natural := 5_000_000;

signal sound_level     : std_logic_vector(15 downto 0);

signal buzzer_ctrl     : std_logic;
signal led_flash       : std_logic;
signal buzz_once       : std_logic;
signal display_mode    : std_logic_vector(1 downto 0);

component xadc_wiz_0
   port (
       daddr_in    : in std_logic_vector(6 downto 0);
       dclk_in     : in std_logic;
       den_in      : in std_logic;
       di_in       : in std_logic_vector(15 downto 0);
       dwe_in      : in std_logic;
       do_out      : out std_logic_vector(15 downto 0);
       drdy_out    : out std_logic;
       vp_in       : in std_logic;
       vn_in       : in std_logic;
```

```
        vauxp6      : in std_logic;
        vauxn6      : in std_logic
    );
end component;

component fsm_controller
    Port (
        clk          : in  STD_LOGIC;
        sound_level  : in  STD_LOGIC_VECTOR(15 downto 0);
        light_sensor : in  STD_LOGIC;
        buzzer_ctrl  : out STD_LOGIC;
        led_flash    : out STD_LOGIC;
        buzz_once    : out STD_LOGIC;
        display_mode : out STD_LOGIC_VECTOR(1 downto 0)
    );
end component;

component led_bargraph
    port (
        sound_level : in std_logic_vector(15 downto 0);
        leds        : out std_logic_vector(9 downto 0);
        led_flash   : in std_logic;
        clk         : in std_logic
    );
end component;
```

```
begin

    xadc_inst : xadc_wiz_0
        port map (
            daddr_in => daddr,
            dclk_in  => CLK100MHZ,
            den_in   => den,
            di_in    => dummy_di,
            dwe_in   => dwe,
            do_out   => xadc_data_raw,
            drdy_out => drdy,
            vp_in    => '0',
            vn_in    => '0',
            vauxp6   => vauxp6,
            vauxn6   => vauxn6
        );

    process(CLK100MHZ)
    begin
        if rising_edge(CLK100MHZ) then
            if sample_counter < SAMPLE_WINDOW then
                sample_counter <= sample_counter + 1;
                if drdy = '1' then
                    if unsigned(xadc_data_raw) > signal_max then
                        signal_max <= unsigned(xadc_data_raw);
                    end if;
                    if unsigned(xadc_data_raw) < signal_min then
                        signal_min <= unsigned(xadc_data_raw);
                    end if;
                end if;
            else
                peak_to_peak <= signal_max - signal_min;
                value <= to_integer(peak_to_peak(15 downto 4));
                sound_level <= std_logic_vector(peak_to_peak);
                signal_max <= (others => '0');
                signal_min <= (others => '1');
                sample_counter <= 0;
            end if;
        end if;
    end process;

    fsm_inst : fsm_controller
        port map (
            clk          => CLK100MHZ,
            sound_level  => sound_level,
            light_sensor => light_sensor,
            buzzer_ctrl  => buzzer_ctrl,
            led_flash    => led_flash,
            buzz_once    => buzz_once,
            display_mode => display_mode
        );

    process(CLK100MHZ)
    begin
        if rising_edge(CLK100MHZ) then
            if clk_divider = 99999 then
                segment_clock <= not segment_clock;
                clk_divider <= 0;
            else
                clk_divider <= clk_divider + 1;
            end if;
        end if;
    end process;

    process(segment_clock)
    begin
        if rising_edge(segment_clock) then
            anode_div <= std_logic_vector(unsigned(anode_div) + 1);
            case display_mode is
                when "01" =>  -- Display HIGH
                    case anode_div is
                        when "00" => AN <= "1110"; SEG <= "0001001";
                        when "01" => AN <= "1101"; SEG <= "0000010";
                        when "10" => AN <= "1011"; SEG <= "1111001";
                        when "11" => AN <= "0111"; SEG <= "0001001";
                        when others => AN <= "1111"; SEG <= "1111111";
                    end case;
                when "10" =>  -- Display LL (low light)
                    case anode_div is
                        when "00" => AN <= "1110"; SEG <= "1000111";
                        when "01" => AN <= "1101"; SEG <= "1000111";
                        when others => AN <= "1111"; SEG <= "1111111";
                    end case;
                when others =>  -- Numerical
                    case anode_div is
                        when "00" => AN <= "1110";
                            if value mod 10 = 0 and value < 10 then
                                SEG <= "1111111";
                            else
                                SEG <= digit_to_segment(value mod 10);
                            end if;
                        when "01" => AN <= "1101";
                            if (value/10) mod 10 = 0 and value < 100 then
                                SEG <= "1111111";
                            else
                                SEG <= digit_to_segment((value/10) mod 10);
                            end if;
                        when "10" => AN <= "1011";
                            if (value/100) mod 10 = 0 and value < 1000 then
```

```
                    SEG <= "1111111";
                else
                    SEG <= digit_to_segment((value/100) mod 10);
                end if;
            when "11" => AN <= "0111";
                if (value/1000) = 0 then
                    SEG <= "1111111";
                else
                    SEG <= digit_to_segment((value/1000) mod 10);
                end if;
            when others => AN <= "1111"; SEG <= "1111111";
        end case;
    end case;
  end if;
end process;


buzzer <= buzz_once or buzzer_ctrl;


led_inst : led_bargraph
  port map (
    sound_level => sound_level,
    leds        => leds,
    led_flash   => led_flash,
    clk         => CLK100MHZ
  );
```

```
end Behavioral;
```

<u>FSM Controller:</u>

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

entity fsm_controller is
    Port (
        clk          : in  STD_LOGIC;
        sound_level  : in  STD_LOGIC_VECTOR(15 downto 0);
        light_sensor : in  STD_LOGIC;
        buzzer_ctrl  : out STD_LOGIC;
        led_flash    : out STD_LOGIC;
        buzz_once    : out STD_LOGIC;
        display_mode : out STD_LOGIC_VECTOR(1 downto 0)
    );
end fsm_controller;

architecture Behavioral of fsm_controller is
    type state_type is (NORMAL, ALARM, ADAPTIVE_NORMAL,
ADAPTIVE_ALARM);
    signal current_state, next_state : state_type;
    signal prev_state : state_type;
    signal sound_int : integer;



    signal entering_adaptive_alarm : std_logic;


        signal buzz_pulse : std_logic := '0';
    signal buzz_counter : integer := 0;
    constant BUZZ_DURATION : integer := 10000000;
begin

    process(clk)
    begin
        if rising_edge(clk) then
            current_state <= next_state;
            prev_state <= current_state;
        end if;
    end process;



    entering_adaptive_alarm <= '1' when (current_state = ADAPTIVE_ALARM and
prev_state /= ADAPTIVE_ALARM) else '0';



    process(current_state, sound_level, light_sensor)
    begin
```

```
sound_int <= to_integer(unsigned(sound_level));
next_state <= current_state;

case current_state is
    when NORMAL =>
        if light_sensor = '1' and sound_int >= 9000 then
            next_state <= ALARM;
        elsif light_sensor = '0' and sound_int < 5000 then
            next_state <= ADAPTIVE_NORMAL;
        elsif light_sensor = '0' and sound_int >= 5000 then
            next_state <= ADAPTIVE_ALARM;
        end if;

    when ALARM =>
        if sound_int < 9000 then
            next_state <= NORMAL;
        end if;

    when ADAPTIVE_NORMAL =>
        if light_sensor = '1' then
            next_state <= NORMAL;
        elsif sound_int >= 5000 then
            next_state <= ADAPTIVE_ALARM;
        end if;
        when ADAPTIVE_ALARM =>
            if light_sensor = '1' then
                next_state <= NORMAL;
            elsif sound_int < 5000 then
                next_state <= ADAPTIVE_NORMAL;
            end if;
    end case;
end process;



process(clk)
begin
    if rising_edge(clk) then

        buzz_pulse <= '0';



        if entering_adaptive_alarm = '1' then
            buzz_counter <= 0;
            buzz_pulse <= '1';

        elsif buzz_counter < BUZZ_DURATION then
            buzz_counter <= buzz_counter + 1;
            buzz_pulse <= '1';
        end if;
```

```
  end if;
end process;


process(current_state, buzz_pulse)
begin

  buzzer_ctrl  <= '0';
  led_flash    <= '0';
  buzz_once    <= '0';
  display_mode <= "00";

  case current_state is
    when NORMAL =>
      display_mode <= "00";

    when ALARM =>
      buzzer_ctrl  <= '1';
      led_flash    <= '1';
      display_mode <= "01";

    when ADAPTIVE_NORMAL =>
      display_mode <= "10";

    when ADAPTIVE_ALARM =>
```

```
            buzz_once    <= buzz_pulse;
            led_flash    <= '1';
            display_mode <= "10";
        end case;
    end process;

end Behavioral;
```

<u>LED bar graph:</u>

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity led_bargraph is
    Port (
        sound_level : in STD_LOGIC_VECTOR(15 downto 0);
        leds        : out STD_LOGIC_VECTOR(9 downto 0);
        led_flash   : in STD_LOGIC;
        clk         : in STD_LOGIC
    );
end led_bargraph;
architecture Behavioral of led_bargraph is
    signal flash_clk : std_logic := '0';
    signal counter    : integer := 0;
    constant THRESHOLD : integer := 10_000_000;
    signal base_leds  : std_logic_vector(9 downto 0);
```

```
begin

    process(sound_level)
        variable level : integer;
    begin
        level := to_integer(unsigned(sound_level));
        base_leds <= (others => '0');
        if level > 500  then base_leds(0) <= '1'; end if;
        if level > 1000 then base_leds(1) <= '1'; end if;
        if level > 1500 then base_leds(2) <= '1'; end if;
        if level > 2000 then base_leds(3) <= '1'; end if;
        if level > 2500 then base_leds(4) <= '1'; end if;
        if level > 3000 then base_leds(5) <= '1'; end if;
        if level > 3500 then base_leds(6) <= '1'; end if;
        if level > 4000 then base_leds(7) <= '1'; end if;
        if level > 4500 then base_leds(8) <= '1'; end if;
        if level > 5000 then base_leds(9) <= '1'; end if;
    end process;

    process(clk)
    begin
        if rising_edge(clk) then
            if counter = THRESHOLD then
                flash_clk <= not flash_clk;
                counter <= 0;
            else
                counter <= counter + 1;
            end if;
        end if;
    end process;

    process(flash_clk, led_flash, base_leds)
    begin
        if led_flash = '1' then
            if flash_clk = '1' then
                leds <= (others => '1');
            else
                leds <= (others => '0');
            end if;
        else
            leds <= base_leds;
        end if;
    end process;
end Behavioral;
```