CODE

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

entity touchbuzzer is
    Port (
        clk         : in  STD_LOGIC;
        touch_input  : in  STD_LOGIC; -- Subtracts 30 minutes from countdown
        button_add_time : in  STD_LOGIC; -- Adds 1 hour to countdown
        buzzer_out   : out STD_LOGIC;
        led1_out     : out STD_LOGIC;
        led2_out     : out STD_LOGIC;
        led3_out     : out STD_LOGIC;
        seg          : out STD_LOGIC_VECTOR(6 downto 0);
        digit_en     : out STD_LOGIC_VECTOR(3 downto 0) -- For HH:MM display
    );
end touchbuzzer;

architecture Behavioral of touchbuzzer is
    constant DEBOUNCE_TIME : integer := 2500000; -- 50ms at 50MHz
    constant CLK_FREQ      : integer := 50000000;
    constant ONE_SECOND    : integer := CLK_FREQ;

    constant TIME_INCREMENT_SECONDS   : integer := 3600;  -- 1 hour
    constant TIME_DECREMENT_SECONDS   : integer := 1800;  -- 30 minutes
    constant MAX_COUNTDOWN_SECONDS    : integer := (99 * 3600) + (59 * 60); -- 99 hours 59 minutes

    -- Startup delay constants
    constant STARTUP_DELAY_MS        : integer := 60; -- Milliseconds for startup delay
    constant STARTUP_DELAY_CYCLES     : integer := (CLK_FREQ / 1000) * STARTUP_DELAY_MS; -- Approx 3,000,000 cycles for 60ms @ 50MHz

    -- Musical note frequencies
    constant NOTE_C6  : integer := 1047;
    constant NOTE_D6  : integer := 1175;
    constant NOTE_E6  : integer := 1319;
    constant NOTE_F6  : integer := 1420;
    constant NOTE_G6  : integer := 1568;
    constant NOTE_A6  : integer := 1760;
    constant NOTE_B6  : integer := 1976;
    constant NOTE_C7  : integer := 2093;
    constant NOTE_Bb6 : integer := 1865;

    -- Note durations
    constant QUARTER_NOTE : integer := ONE_SECOND / 2;   -- 500ms
    constant HALF_NOTE    : integer := ONE_SECOND;       -- 1s
    constant WHOLE_NOTE   : integer := 2 * ONE_SECOND;   -- 2s

    -- Debounce for touch_input
    signal s_touch_counter      : integer range 0 to DEBOUNCE_TIME := 0;
    signal s_touch_stable_state : STD_LOGIC := '0';
    signal s_touch_last_input   : STD_LOGIC := '0';
    signal s_touch_last_stable  : STD_LOGIC := '0';

    -- Debounce for button_add_time
    signal s_btn_add_time_counter      : integer range 0 to DEBOUNCE_TIME := 0;
    signal s_btn_add_time_stable_state : STD_LOGIC := '0';
    signal s_btn_add_time_last_input   : STD_LOGIC := '0';
    signal s_btn_add_time_last_stable  : STD_LOGIC := '0';

    -- Countdown state (total seconds)
    signal s_countdown_total_seconds : integer range 0 to MAX_COUNTDOWN_SECONDS := 0; -- Initialized to 0
    signal s_countdown_active        : STD_LOGIC := '0';
    signal s_second_tick_counter     : integer range 0 to ONE_SECOND -1 := 0;
    signal s_song_should_play_finish : STD_LOGIC := '0';

    -- Song playing signals
    signal s_song_active    : STD_LOGIC := '0';
    signal s_note_index     : integer range 0 to 28 := 0;
    signal s_note_timer     : integer := 0;

    -- Tone generation signals
    signal s_tone_gen       : STD_LOGIC := '0';
    signal s_tone_counter   : integer := 0;
    signal s_current_period : integer := 0;

    -- 7-segment display digit values (BCD for HH:MM)
    signal s_disp_h1   : unsigned(3 downto 0) := (others => '0'); -- Hours Tens
    signal s_disp_h0   : unsigned(3 downto 0) := (others => '0'); -- Hours Units
    signal s_disp_m1   : unsigned(3 downto 0) := (others => '0'); -- Minutes Tens
    signal s_disp_m0   : unsigned(3 downto 0) := (others => '0'); -- Minutes Units
```

```
        -- Startup delay signals
        signal s_startup_delay_counter : integer range 0 to
STARTUP_DELAY_CYCLES := 0;
        signal s_system_ready          : STD_LOGIC := '0';



        type note_record is record
            freq     : integer;
            duration : integer;
        end record;

        type song_array is array (0 to 28) of note_record;
      Ayanna
      buzzer_out <= s_tone_gen when s_song_active = '1'
else '1';

        led1_out <= '0' when s_countdown_active = '1'
else '1';
        led2_out <= '0' when s_countdown_active = '1' and
s_countdown_total_seconds >= 3600 else '1';
        led3_out <= '0' when s_countdown_active = '1' and
s_countdown_total_seconds > 0 and
s_countdown_total_seconds <= 600 else '1';

        -- 7-segment display multiplexing and segment
decoding (HH:MM)
        process(clk)
            variable v_digit_scan_counter : integer range 0
to 100000 := 0;
            variable v_current_mux_digit  : integer range 0
to 3     := 0;
        begin
            if rising_edge(clk) then
                if v_digit_scan_counter < 50000 then
                    v_digit_scan_counter :=
v_digit_scan_counter + 1;
                else
                    v_digit_scan_counter := 0;
                    v_current_mux_digit  :=
(v_current_mux_digit + 1) mod 4;
                end if;

                case v_current_mux_digit is
                    when 0 =>
                        digit_en <= "1110";
                        case s_disp_m0 is
                            when "0000" => seg <= "1111110";
when "0001" => seg <= "0110000";
                            when "0010" => seg <= "1101101";
when "0011" => seg <= "1111001";
                            when "0100" => seg <= "0110011";
when "0101" => seg <= "1011011";
                            when "0110" => seg <= "1011111";
when "0111" => seg <= "1110000";
                            when "1000" => seg <= "1111111";
when "1001" => seg <= "1111011";
                            when others => seg <= "0000001";
                        end case;
                    when 1 =>
                        digit_en <= "1101";
                        case s_disp_m1 is
                            when "0000" => seg <= "1111110";
when "0001" => seg <= "0110000";
                            when "0010" => seg <= "1101101";
when "0011" => seg <= "1111001";
                            when "0100" => seg <= "0110011";
when "0101" => seg <= "1011011";
                            when others => seg <= "0000000";
                        end case;
                    when 2 =>
                        digit_en <= "1011";
                        case s_disp_h0 is
                            when "0000" => seg <= "1111110";
when "0001" => seg <= "0110000";
                            when "0010" => seg <= "1101101";
when "0011" => seg <= "1111001";
                            when "0100" => seg <= "0110011";
when "0101" => seg <= "1011011";
                            when "0110" => seg <= "1011111";
when "0111" => seg <= "1110000";
                            when "1000" => seg <= "1111111";
when "1001" => seg <= "1111011";
                            when others => seg <= "1111110";
                        end case;
                    when 3 =>
                        digit_en <= "0111";
                        case s_disp_h1 is
                            when "0000" => seg <= "1111110";
when "0001" => seg <= "0110000";
                            when "0010" => seg <= "1101101";
when "0011" => seg <= "1111001";
                            when "0100" => seg <= "0110011";
when "0101" => seg <= "1011011";
                            when "0110" => seg <= "1011111";
when "0111" => seg <= "1110000";
                            when "1000" => seg <= "1111111";
when "1001" => seg <= "1111011";
                            when others => seg <= "0000000";
                        end case;
                end case;
            end if;
        end process;

    end Behavioral;
```