طراحی سیستم های دیجیتال

سیستم ضد سرقت خودرو

استاد: دکتر عبدلی

دانشجویان: محمد میرزایی 9912358039

سید علی امامی 9912358005

مهدی براتی 9912358008

دانشگاه بوعلی سینا همدان - ترم 4022

Contents

مقدمه

موضوع پروژه سیستم ضد سرقت خودرو است . ما با استفاده از زبان برنامه نویسی vhdl آن را پیاده سازی کرده ایم. رویکرد در این پروژه بدین صورت بوده است که اجزاری مختلف به صورت جداگانه پیاده سازی شده و توسط یک ماژول مرکزی (FSM) کنترل میگردد. علاوه بر ماژول های موجود در متن پروژه ، ماژول synchronization به صورت جداگانه افزوده شده است و تغییراتی برای بهبود عملکرد نیز انجام شده است.

توضیحات Schema

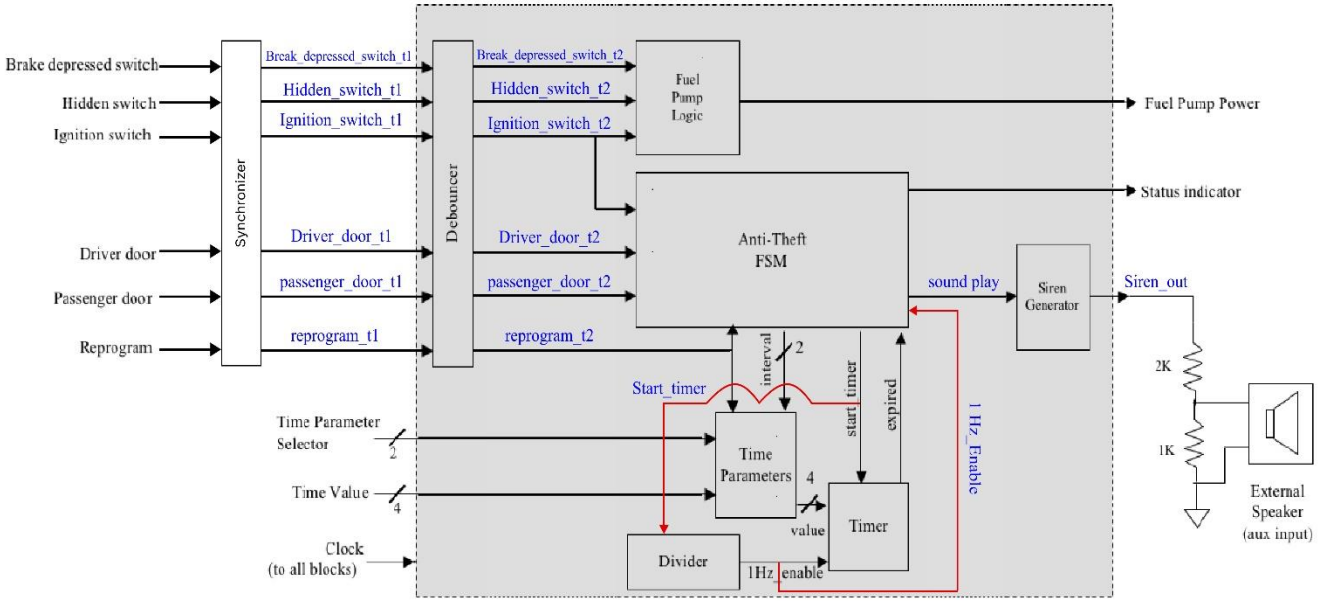
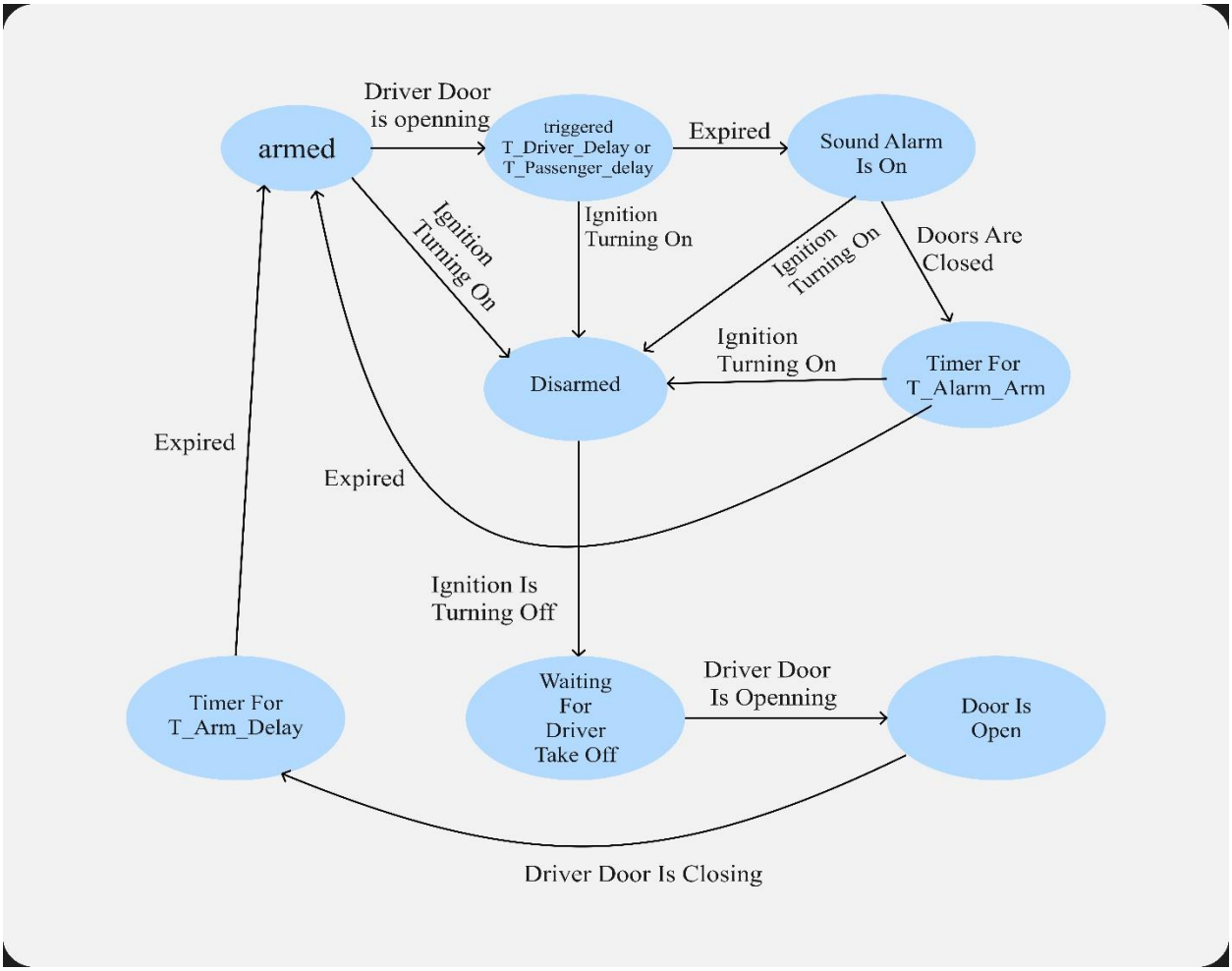


Figure 2: Block Diagram of Anti-Theft System

در شمای بالا ماژول synchronizer را همانطور که اشاره شده بود ، برای ورودی های debouncer اضافه کردیم. برای کارکرد درست ماژول های مختلف از سیگنال های میانی استفاده کردیم. برای مثال : خروجی synchronizer را با پسوند t1_ و خروجی debouncer را با پسوند t2_ نامگذاری کردیم . همچنین از سیگنال های 1 Hz enable در ماژول FSM استفاده شده برای همین یک سیگنال به FSM وارد شده و برای کارکرد صحیح تر divider ، یک ورودی از start timer به آن متصل کرده ایم.

توضیحات Finite State Machine



Armed: در این حالت سیستم ضد سرقت فعال است و چنانچه هریک از درب های خودرو باز شود ، سیستم به حالت triggered T driver delay or T passenger delay میرود و اگر خودرو روشن شود به حالت disarmed میرود . در این حالت آژیر خاموش است و status indicator (چراغ روی داشبورد در بازه ی ۲ ثانیه ای چشمک میزند)

Triggered T driver delay or T passenger delay : در این حالت سیستم در حالت برانگیخته است و با توجه به اینکه کدام یک از درب های خودرو باز شده است (تاخیر برای درب راننده یا تاخیر برای درب مسافر) تایمر در حال شمارش است . چنانچه در بازه زمانی تایمر که در حال شمارش است ، خودرو روشن شود به سیستم حالت disarmed میرود و چنانچه بازه زمانی آن تاخیر سپری شود و سیگنال expired فعال شود ، به حالت sound alarm is on میرود. در این حالت status indicator و آژیر خاموش است.

Sound alarm is on : در این وضعیت چنانچه خودرو روشن شود ، سیستم به حالت disarmed و اگر تمام در ها بسته شود به حالت timer for T alarm arm میرود . در این حالت آژیر روشن است و |status indicator نیز به صورت ممتد روشن است.

Timer for T alarm arm : در این وضعیت تایمر تاخیر t alarm on را محاسبه میکند چنانچه در بازه شمارش معکوس خودرو روشن شود به حالت disarmed میرود ، و چنانچه سیگنال expired فعال شود به حالت armed میرود. در این حالت status indicator و آژیر هردو روشن هستند .

Disarmed : در این حالت چنانچه خودرو خاموش شود ، به وضعیت waiting for driver take off میرود . آژیر و status indicator خاموش هستند.

Waiting for deriver take off : در این حالت سیستم منتظر میماند تا زمانی که درب راننده باز شود و به حالت door is open میرود . در این حالت status indicator و آژیر خاموش هستند.

Door is Open : در این حالت سیستم منتظر میماند تا تمامی درب ها بسته شوند سپس به وضعیت timer for t arm delay میرود . status indicator و آژیر خاموش هستند .

Timer for t arm delay : در این حالت سیستم شمارش معکوس را با مقدار (t arm delay) انجام میدهد. زمانی که شمارش به پایان برسد و سیگنال expired فعال شود ، سیستم به حالت Armed میرود . status indicator و آژیر غیر فعال است.

توضیحات ماژول ها

ماژول synchronizer : ورودی ها قبل از اینکه به debouncer وارد شوند ، ابتدا وارد این ماژول میشود و به علت اینکه ورودی ها Asynchron هستند ، تغییرات آنها در نزدیکی لبه clock مشکلات عدیده ای ایجاد مینماید .لذا با استفاده از این ماژول و process حساس به لبه کلاک ، انتقال ورودی ها سمت debouncer و سایر ماژول ها را همگام میسازد. در این کد از سیگنال در نقش بافر استفاده شده که به صورت پیشفرض دارای یک flop میباشد و با تغییرات clock ورودی ها در این سیگنال ها قرار میگیرند و مقدار قبلی سیگنال ها به خارج از این ماژول انتقال داده میشود.

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

entity synchronize is
    generic (
        NSYNC : integer := 2  -- تعداد فلیپ‌فلاپ‌های همگام‌ساز، باید حداقل 2 باشد
    );
    port (
        clk  : in  std_logic;
        Break_depressed_switch  : in  std_logic; -- input for synchronization
        Hidden_switch           : in  std_logic; -- input for synchronization
        Ignition_switch         : in  std_logic; -- input for synchronization
        Driver_door             : in  std_logic; -- input for synchronization
        Passenger_door          : in  std_logic; -- input for synchronization
        Reprogram               : in  std_logic; -- input for synchronization
        Break_depressed_switch_t1  : out std_logic; -- output after synchronization
        Hidden_switch_t1           : out std_logic; -- output after synchronization
        Ignition_switch_t1         : out std_logic; -- output after synchronization
        Driver_door_t1             : out std_logic; -- output after synchronization
        Passenger_door_t1          : out std_logic; -- output after synchronization
        Reprogram_t1               : out std_logic
    );
end entity synchronize;

architecture Behavioral of synchronize is
    -- signal to buffering value to avoid change near a clock time
    signal sync_b : std_logic_vector(NSYNC-2 downto 0);
    -- signal to buffering value to avoid change near a clock time
    signal sync_h : std_logic_vector(NSYNC-2 downto 0);
    -- signal to buffering value to avoid change near a clock time
    signal sync_i : std_logic_vector(NSYNC-2 downto 0);
    -- signal to buffering value to avoid change near a clock time
    signal sync_d : std_logic_vector(NSYNC-2 downto 0);
    -- signal to buffering value to avoid change near a clock time
    signal sync_p : std_logic_vector(NSYNC-2 downto 0);
    -- signal to buffering value to avoid change near a clock time
    signal sync_r : std_logic_vector(NSYNC-2 downto 0);
begin

    process (clk)
    begin
        -- put value sinchronized by clock
        sync_b <= sync_b(NSYNC-3 downto 0) & Break_depressed_switch;
        -- put value sinchronized by clock
        sync_h <= sync_h(NSYNC-3 downto 0) & Hidden_switch;
        -- put value sinchronized by clock
        sync_i <= sync_i(NSYNC-3 downto 0) & Ignition_switch;
        -- put value sinchronized by clock
        sync_d <= sync_d(NSYNC-3 downto 0) & Driver_door;
        -- put value sinchronized by clock
        sync_p <= sync_p(NSYNC-3 downto 0) & Passenger_door;
        -- put value sinchronized by clock
        sync_r <= sync_r(NSYNC-3 downto 0) & Reprogram;
    end process;

    -- put value to output after sinchronization
    Break_depressed_switch_t1 <= sync_b(NSYNC-2);
    -- put value to output after sinchronization
    Hidden_switch_t1 <= sync_h(NSYNC-2);
    -- put value to output after sinchronization
    Ignition_switch_t1  <= sync_i(NSYNC-2);
    -- put value to output after sinchronization
    Driver_door_t1  <= sync_d(NSYNC-2);
    -- put value to output after sinchronization
    Passenger_door_t1 <= sync_p(NSYNC-2);
    -- put value to output after sinchronization
    Reprogram_t1 <= sync_r(NSYNC-2);

end architecture Behavioral;
```

Debouncer : به دلیل وجود مشکلات ناشی از پرش مکانیکی در سوئیچ ها در زمان اتصال و قطع یک سوئیچ ممکن است برای یک بازه کوتاهی یک سیگنال ثابت نباشد لذا با استفاده از debouncer چک میکنیم که یک سیگنال در مدت ۰٫۱ ثانیه معادل ۶۵۰۰ کلاک ثابت بود ، مقدار آن را به خروجی منتقل کنیم . در کارکرد این ماژول یک شمارنده برای هر ورودی در نظر گرفته شده است و با هر تغییرات clock تمام ورودی ها را چک میکنیم تا با مقدار قبلی خود تفاوتی نداشته باشند . اگر هر سیگنال تغییری نداشته باشد ، ۱ واحد به شمارنده آن افزوده میشود ولی اگر تغییری داشته باشد ، شمارنده آن ۰ میشود و در نهایت اگر یک سیگنال به مدت ۰٫۱ ثانیه ثابت بود مقدار آن را به خروجی منتقل میکنیم .

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

entity Debouncer is
    Port (
        Clk : in std_logic; -- input
        Brake_depressed_switch_t1 : in std_logic; -- input
        Hidden_switch_t1 : in std_logic; -- input
        Ignition_switch_t1 : in std_logic; -- input
        Driver_door_t1 : in std_logic; -- input
        Passenger_door_t1 : in std_logic; -- input
        Reprogram_t1 : in std_logic; -- input
        Brake_depressed_switch_t2 : out std_logic; -- output
        Hidden_Switch_t2 : out std_logic; -- output
        Ignition_switch_t2 : out std_logic; -- output
        Driver_door_t2 : out std_logic; -- output
        Passenger_door_t2 : out std_logic; -- output
        Reprogram_t2 : out std_logic -- output
    );
end Debouncer;

architecture Behavioral of Debouncer is
    -- new type for counting
    type Debounce_Count_Array is array (0 to 5) of natural range 0 to 6500;
    -- signal to count the mounts of clocks that is in each put was stable
    signal Debounce_Count : Debounce_Count_Array := (others => 0);
    -- saving value of inputs in previous clock
    signal Debounce_State_Vector : std_logic_vector(5 downto 0) := (others => '0');

begin
    -- Debounce counter process
    process (Clk)
    begin
            for i in 0 to 5 loop
                if ( -- check each input to do not have any changes
                    (i = 0 and Brake_depressed_switch_t1 /= Debounce_State_Vector(i)) or
                    (i = 1 and Hidden_switch_t1 /= Debounce_State_Vector(i)) or
                    (i = 2 and Ignition_switch_t1 /= Debounce_State_Vector(i)) or
                    (i = 3 and Driver_door_t1 /= Debounce_State_Vector(i)) or
                    (i = 4 and Passenger_door_t1 /= Debounce_State_Vector(i)) or
                    (i = 5 and Reprogram_t1 /= Debounce_State_Vector(i))
                ) then
                    -- State change detected, reset counter
                    Debounce_Count(i) <= 0;
                    -- put new value in debunce state vector
                    case i is
                      when 0 =>
                        Debounce_State_Vector(i) <= Brake_depressed_switch_t1;
                      when 1 =>
                        Debounce_State_Vector(i) <= Hidden_switch_t1;
                      when 2 =>
                        Debounce_State_Vector(i) <= Ignition_switch_t1;
                      when 3 =>
                        Debounce_State_Vector(i) <= Driver_door_t1;
                      when 4 =>
                        Debounce_State_Vector(i) <= Passenger_door_t1;
                      when 5 =>
                        Debounce_State_Vector(i) <= Reprogram_t1;
                      when others =>
                          null;
                  end case;

                elsif Debounce_Count(i) = 6500 then
                    -- Stable state for 6500 cycles , update output
                    -- ( 6500 is equal with 0.1 second)
                    case i is
                        when 0 =>
                            Brake_depressed_switch_t2 <= Brake_depressed_switch_t1;
                        when 1 =>
                            Hidden_Switch_t2 <= Hidden_switch_t1;
                        when 2 =>
                            Ignition_switch_t2 <= Ignition_switch_t1;
                        when 3 =>
                            Driver_door_t2 <= Driver_door_t1;
                        when 4 =>
                            Passenger_door_t2 <= Passenger_door_t1;
                        when 5 =>
                            Reprogram_t2 <= Reprogram_t1;
                        when others =>
                            null;
                    end case;
                else
                    -- Increment counter
                    Debounce_Count(i) <= Debounce_Count(i) + 1;
                end if;
            end loop;
    end process;
end Behavioral;
```

همانطور که در شبیه سازی مشخص است ، تاخیر ۰٫۱ ثانیه ای اعمال شده.

Fuel Pump

در این کد سیگنال fuel pump در صورتی فعال میشود که سوئیچ روشن باشد و کلید مخفی و ترمز همزمان فشرده باشد و برای خاموش شده تنها در صورتی که سوئیچ خاموش شود fuel pump قطع میشود.

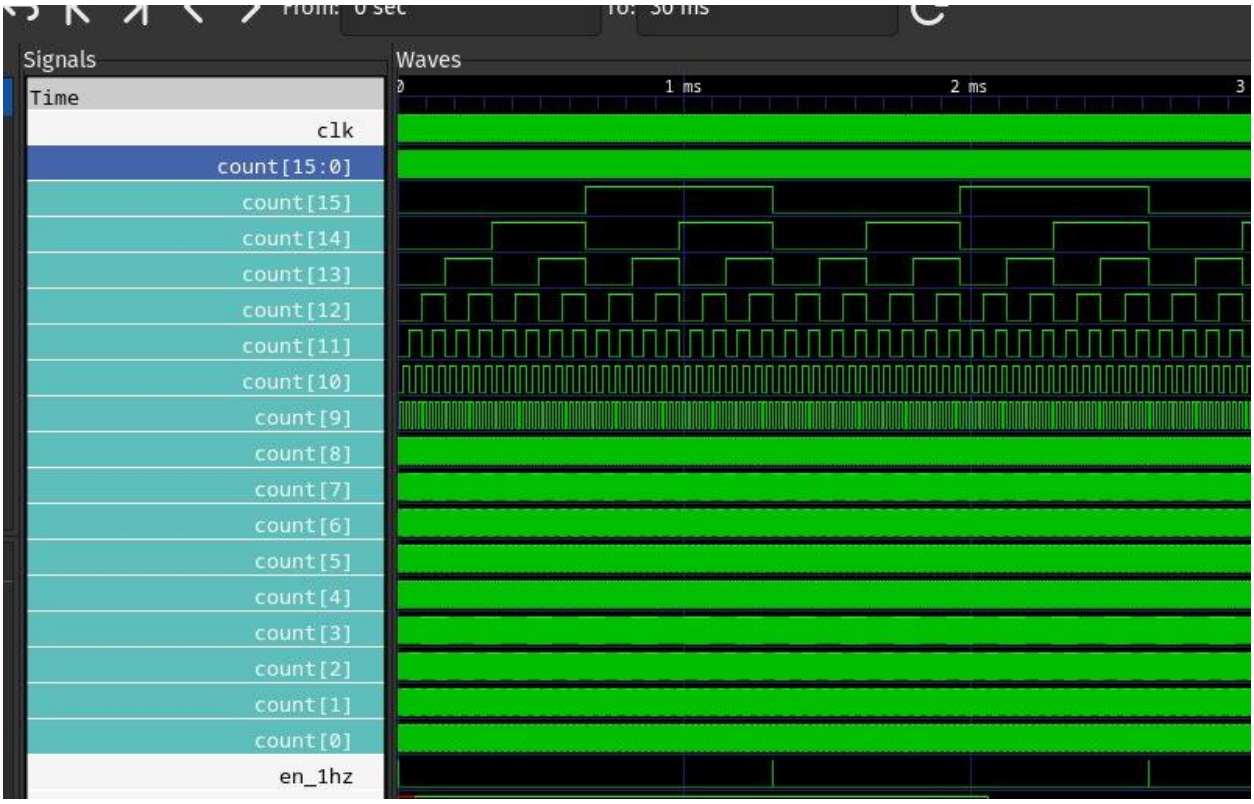
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Fuel_Pump_Logic is
    port (
        Brake_depressed_switch_t2 : in  std_logic; -- input
        Hidden_Switch_t2          : in  std_logic; -- input
        Ignition_switch_t2        : in  std_logic; -- input
        Fuel_Pump_Power       : out std_logic -- output
    );
end entity Fuel_Pump_Logic;

architecture Behavioral of Fuel_Pump_Logic is
begin
    process (Brake_depressed_switch_t2, Hidden_Switch_t2, Ignition_switch_t2)
    begin -- sensetive to inputs
        if Ignition_switch_t2 = '1' then -- check for open switch
            if Brake_depressed_switch_t2 = '1' and Hidden_Switch_t2 = '1' then
                -- check for break depressed switch and hidden switch are pressed
                Fuel_Pump_Power <= '1';  -- Fuel pump switch is turned on
            end if;
        else
            Fuel_Pump_Power <= '0';  -- fuel pump power turn off when ignittion switch is off
        end if;
    end process;
end architecture Behavioral;
```

زمانی که ترمز و سوئیچ مخفی فشار داده شدند fuel pump روشن و زمانی که سوئیچ قطعه خاموشه .

Divider

در این کد از یک سیگنال ۱۶ بیتی برای شمارش استفاده شده است و فرض کرده ایم که ۲ به توان ۱۶ معادل ۱ ثانیه است . برای اینکه شمارش ما دقت کافی را داشته باشد از start timer یک سیگنال ورودی گرفته ایم تا با استفاده از آن به محض ۱ شدن start timer ، شمارش را ۰ کنیم و از ابتدا شروع به شمارش کنیم و در زمانی که به عدد ۲ به توان ۱۶ که معادل ۱۱۱۱۱۱۱۱۱۱۱۱۱۱۱۱ شد ، یک سیگنال با عنوان 1 hz enable در خروجی ایجاد میکنیم تا با استفاده از آن ثانیه را در ماژول تایمر و FSM داشته باشیم و استفاده کنیم .

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

entity divider is

  port(
    clk_in : in STD_LOGIC;
    start_timer : in STD_LOGIC; -- using for reset timer
    en_1hz : out STD_LOGIC0 -- output to show 1 second
  );
end divider;


architecture Behavioral of divider is
  signal count : unsigned(15 downto 0) := (others => '0'); -- clock counter for 1 second
begin
  process (clk_in, start_timer)
  begin

    if start_timer = '1' then -- reset timer
      count <= (others => '0');
      en_1hz <= '0'; -- if time input was zero turn on expired
    end if;

      if rising_edge(clk_in) then -- check for counting
      if count = "1111111111111111" then
        en_1hz <= '1';
        count <= (others => '0');

      else
        count <= count + 1;
        en_1hz <= '0';
      end if;
    end if;
  end process;
end Behavioral;
```



همانطور که در تصویر مشخص است ، زمانی که شمارش به ۲ به توان ۱۶ میرسد ، ۱ سیگنال 1 Hz enable ایجاد میشود .

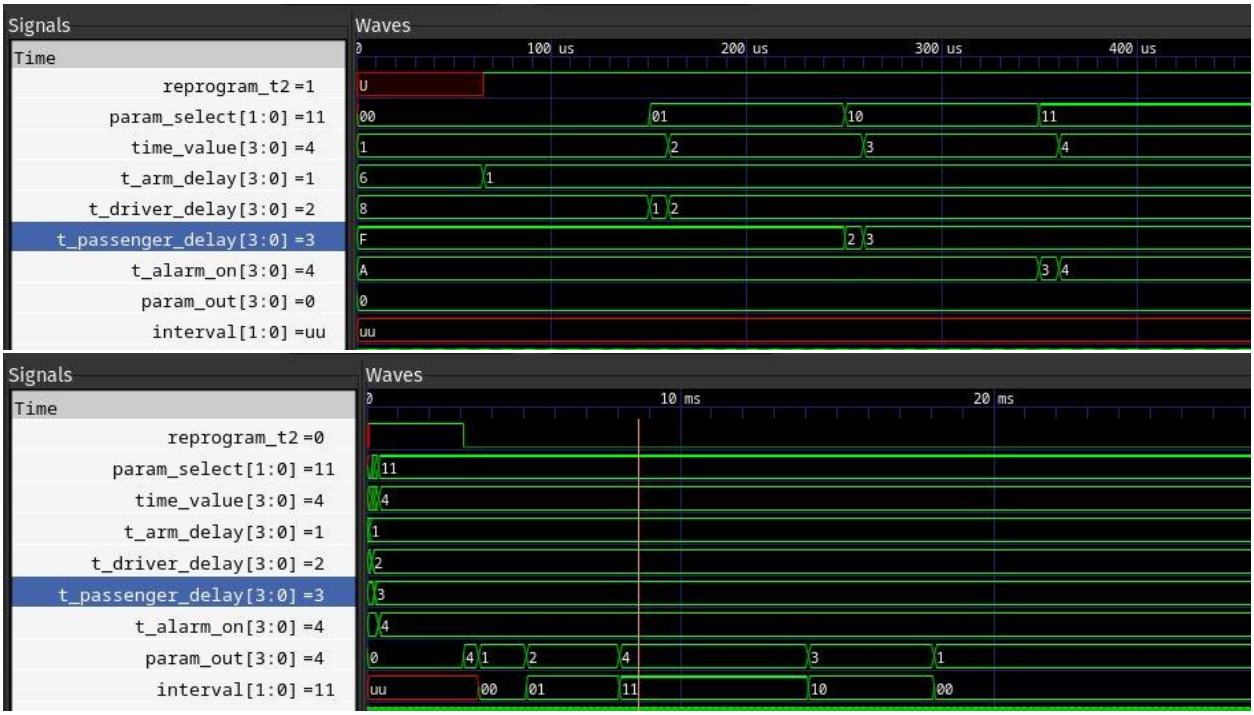
Time parameter

در این ماژول دو کار انجام میشود :

۱- میتوانیم مقادیر پیشفرض را برای تاخیر ها تغییر دهیم .
۲- مقدار تاخیری را که FSM از ما میخواهد در خروجی ایجاد کنیم.

با استفاده از سیگنال reprogram t2 تعیین میکنیم که در وضعیت تغییر مقدار تاخیر هستیم یا در حال انتقال آن . زمانی که این سیگنال ۱ باشد ، param select تاخیر مورد نظر را انتخاب میکند و time value مقدار جدید را برای آن تاخیر set میکند و زمانی که reprogram t2 در حالت ۰ است ، ماژول با توجه به مقدار سیگنال ورودی interval تشخیص میدهد که کدام تاخیر را در خروجی خود (param out) قرار دهد .

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


entity time_parameters is
    Port (
        clk : in STD_LOGIC;
        Reprogram_t2 : in STD_LOGIC; -- signal to decide reprogram
        param_select : in STD_LOGIC_VECTOR (1 downto 0); -- select delay
        time_value : in STD_LOGIC_VECTOR (3 downto 0); -- value for selected delay
        interval : in STD_LOGIC_VECTOR (1 downto 0); -- select delay to put it in param out
        param_out : out STD_LOGIC_VECTOR (3 downto 0) -- as input for timer
    );
end time_parameters;


architecture Behavioral of time_parameters is
    -- defualt valuse for delays
    signal T_ARM_DELAY : STD_LOGIC_VECTOR (3 downto 0) := "0110";
    signal T_DRIVER_DELAY : STD_LOGIC_VECTOR (3 downto 0) := "1000";
    signal T_PASSENGER_DELAY : STD_LOGIC_VECTOR (3 downto 0) := "1111";
    signal T_ALARM_ON : STD_LOGIC_VECTOR (3 downto 0) := "1010";
    signal temp : STD_LOGIC_VECTOR (3 downto 0) := "0000";
begin

    process(clk, Reprogram_t2)
    begin
            if Reprogram_t2 = '1' then -- check flag to change value sums of delay
                case param_select is

                    when "00" =>
                        T_ARM_DELAY <= time_value;

                    when "01" =>
                        T_DRIVER_DELAY <= time_value;

                    when "10" =>
                        T_PASSENGER_DELAY <= time_value;

                    when others =>
                        T_ALARM_ON <= time_value;
                end case;

            elsif(Reprogram_t2 = '0') then -- when FSM select a delay , send it to Timer
                case interval is
                    when "00" =>
                        temp  <= T_ARM_DELAY;

                    when "01" =>
                        temp <= T_DRIVER_DELAY;

                    when "10" =>
                        temp <= T_PASSENGER_DELAY;

                    when others =>
                        temp <= T_ALARM_ON;
                end case;
            end if;
    end process;
    param_out <= temp;
end Behavioral;
```

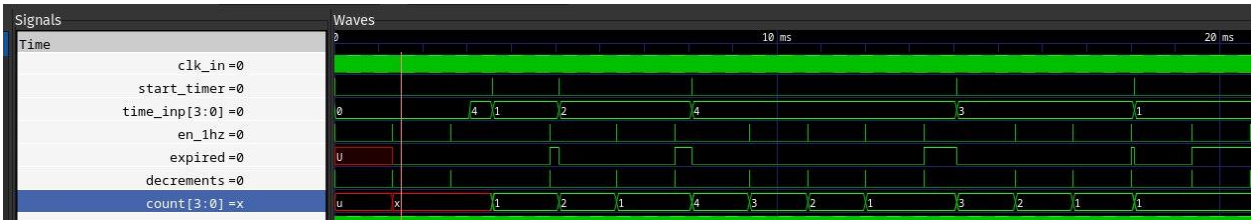

Signals
Time
reprogram_t2 =1
param_select[1:0] =11
time_value[3:0] =4
t_arm_delay[3:0] =1
t_driver_delay[3:0] =2
t_passenger_delay[3:0] =3
t_alarm_on[3:0] =4
param_out[3:0] =0
interval[1:0] =uu
Waves
0
100 us
200 us
300 us
400 us
U
00
01
10
11
1
2
3
4
6
1
8
1
2
F
2
3
A
3
4
0
uu
Signals
Time
reprogram_t2 =0
param_select[1:0] =11
time_value[3:0] =4
t_arm_delay[3:0] =1
t_driver_delay[3:0] =2
t_passenger_delay[3:0] =3
t_alarm_on[3:0] =4
param_out[3:0] =4
interval[1:0] =11
Waves
0
10 ms
20 ms
11
4
1
2
3
4
0
4
1
2
4
3
1
uu
00
01
11
10
00

timer

در این ماژول یک سیگنال با عنوان start timer از سمت FSM که نشانگر شروع شمارش است گرفته میشود و یک ورودی به نام time inp از سمت time parameter گرفته میشود که مقدار زمانی است که باید شمارش معکوس انجام دهیم و یک ورودی en 1 hz از divider گرفته میشود که سیگنال نشان دهنده ی هر ثانیه است . این ماژول یک سیگنال خروجی به نام expired دارد که در زمانی که شمارش به پایان برسد ، این سیگنال فعال میشود.

از سیگنال count به عنوان یک متغیر برای ذخیره ی مقداری که باید شمارش کنیم استفاده میکنیم و سیگنال decrement برای تشخیص اینکه یک ثانیه سپری شده است استفاده میکنیم. کارکرد کلی سیستم به این صورت است که چنانچه start timer فعال شود ، ماژول مقدار ورودی time inp را در متغیر count قرار میدهد و با چک کردن سیگنال en 1 hz تشخیص میدهد در چه زمان هایی count را کاهش دهد. زمانی که شمارش معکوس به پایان برسد ، سیگنال expired فعال میشود .

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

entity timer is
  port(
    clk_in : in STD_LOGIC;
    start_timer : in STD_LOGIC; -- start to counting
    time_inp : in STD_LOGIC_VECTOR(3 downto 0); -- value for counting
    en_1hz : in STD_LOGIC; -- input signal for 1 second
    expired : out STD_LOGIC -- flag for expired time
  );
end timer;

architecture Behavioral of timer is
  signal count : unsigned(3 downto 0); -- temporary variable for save time
  signal decrements : STD_LOGIC; -- to decide decrementing count
begin
  process (clk_in) -- decrement 1 cycle aghab tare  bekhater in process
  begin

    if start_timer = '1' then -- check for start counting
      expired <= '0'; -- turning off expired flag
       if unsigned(time_inp) = 0 then -- checking for zero input avoiding of more counting
         count <= "0001" ;
         expired <= '1';
       else
         count <= unsigned(time_inp); -- set new value for count
       end if ;
      decrements <= '0';

    elsif en_1hz = '1' then -- check 1 second flag
      decrements <= '1';

    else
      decrements <= '0';
    end if;

    if decrements = '1' then
      if count = 1 then -- check for last second of timer
        expired <= '1';

      else
        count <= count - 1;
        expired <= '0';
      end if;

      decrements <= '0';
    end if;
  end process;
end Behavioral;
```



با خوردن start timer با روشن شدن start timer ، en 1 hz ریسیت میشود و از آن نقطه شروع به شمارش میکند و سیگنال expired پس از به پایان رسیدن شمارش فعال میشود و از سیگنال decrement برای تشخیص زمان کاهش شمارشگر (count) استفاده میشود.

Siren Generator

این ماژول در پروژه نقش آژیر را ایفا میکند. کارکرد این ماژول به این صورت است که اگر یک سیگنال ورودی به نام play sound فعال باشد ، یک صدای با فرکانس متغیر بین ۴۰۰ تا ۷۰۰ ایجاد میشود . بدین صورت که از فرکانس ۴۰۰ شروع شده و به صورت پله ای تا ۷۰۰ پیش میرود و در نهایت وقتی به ۷۰۰ رسید به صورت پلکانی تا ۴۰۰ کم میشود و همین روند ادامه دارد.

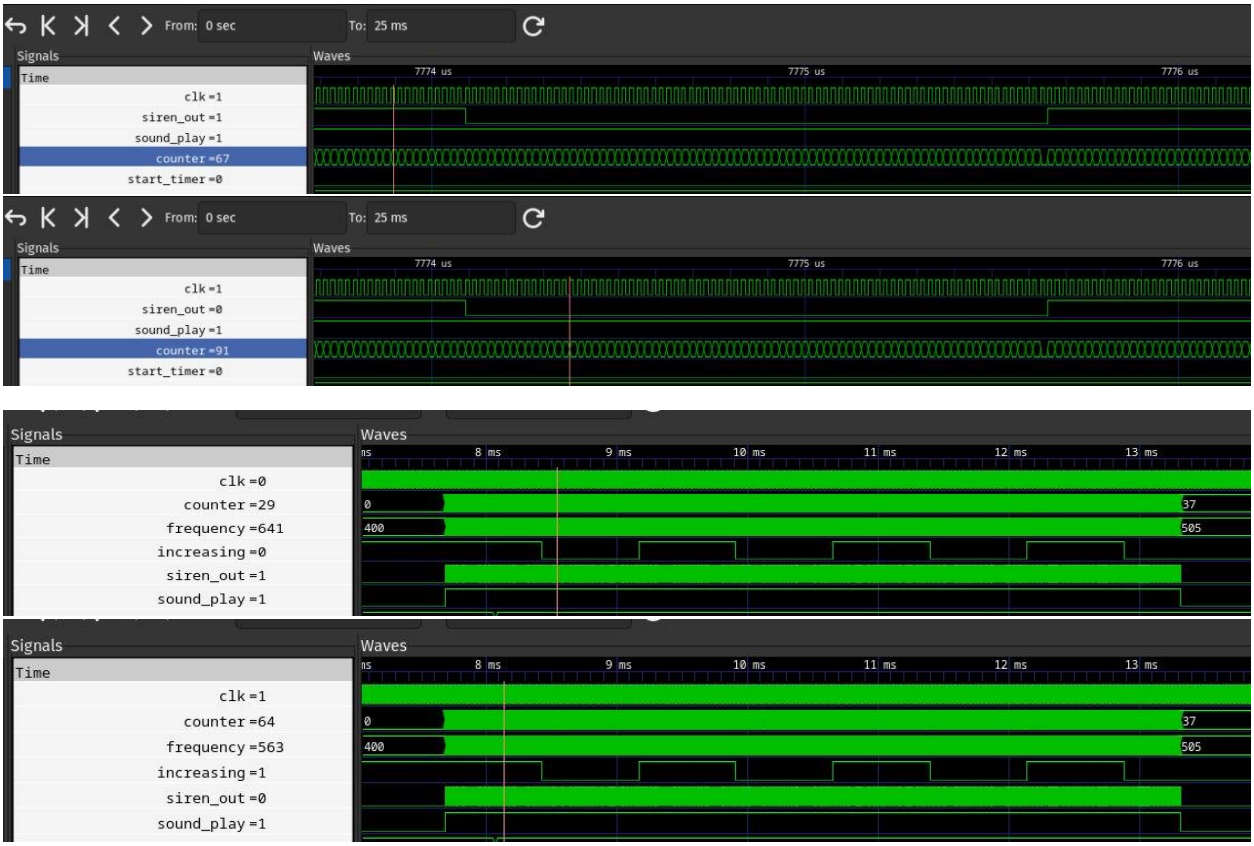
در بازه ی ۲٫۵ ثانیه ای فرکانس یکباره به ۷۰۰ رسیده و کاهش میابد.

سیگنال counter برای شمارش clock میباشد ، تا در زمان معین سیگنال خروجی را ۰ یا ۱ کند.

سیگنال frequency ، سیگنال فرکانس را در خود نگه میدارد که با استفاده از سیگنال increasing تشخیص میدهیم که سیگنال frequency را افزایش یا کاهش دهیم.

در زمانی که sound play غیرفعال باشد ، سیگنال خروجی ۰ است.

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

entity SineWaveGenerator is
    port(
        clk: in std_logic;      -- input
        sound_play : in std_logic ; -- input signal as a switch on or off
        Siren_out: out std_logic    -- output signal to used in speaker
    );
end entity SineWaveGenerator;

architecture Behavioral of SineWaveGenerator is
    signal counter: integer := 0; -- counter to decide output value
    signal frequency: integer := 400;  -- frequency of output signal
    signal increasing: std_logic := '1';  -- flag to determine increasing or decreasing frequency

begin
    process(clk)
    begin
        if rising_edge(clk)  and sound_play = '1' then
            counter <= counter + 1;
            if counter = 65536/frequency then
                if frequency = 700 then
                    increasing <= '0';  -- change flag to decreasing frequency
                elsif frequency = 400 then
                    increasing <= '1';  -- change flag to increasing frequency
                end if;

                if increasing = '1' then
                    frequency <= frequency + 1;  -- increasing frequency
                else
                    frequency <= frequency - 1;  -- decreasing frequency
                end if;

                counter <= 0;
            end if;

            if counter < (65536/frequency)/2 then  -- change to create periodic signal
                Siren_out <= '1';
            else
                Siren_out <= '0';
            end if;

        elsif sound_play = '0' then -- when input is off output is off
            Siren_out <= '0';
        end if;
    end process;
end architecture Behavioral;

```


From: 0 sec
To: 25 ms
Signals
Waves
Time
clk =1
siren_out =1
sound_play =1
counter =67
start_timer =0
7774 us
7775 us
7776 us
From: 0 sec
To: 25 ms
Signals
Waves
Time
clk =1
siren_out =0
sound_play =1
counter =91
start_timer =0
7774 us
7775 us
7776 us
Signals
Waves
Time
clk =0
counter =29
frequency =641
increasing =0
siren_out =1
sound_play =1
ns
8 ms
9 ms
10 ms
11 ms
12 ms
13 ms
0
37
400
505
Signals
Waves
Time
clk =1
counter =64
frequency =563
increasing =1
siren_out =0
sound_play =1
ns
8 ms
9 ms
10 ms
11 ms
12 ms
13 ms
0
37
400
505

FSM

این بخش نقش هسته اصلی برنامه ما را دارد و حالت های مختلف برنامه و وضعیت سیگنال هارا مشخص میکند. در این ماژول status type تعریف شده که شامل حالت های سیستم است و سیگنال current state وضعیت فعلی را در خود نگه میدارد که به صورت پیشفرض armed است.

سیگنال indicator count در حالت armed استفاده شده تا با استفاده از آن status indicator را در بازه های ۲ ثانیه ای روشن و خاموش کنیم .

سیگنال flag start برای این است که در هر state چنانچه نیاز به start timer بود ، با استفاده از آن روشن شدن start را چک کنیم.

سیگنال flag expi برای کنترل این است که سیگنال expi موجود حاصل از همان state باشد و تاثیر state های قبلی در آن نباشد.

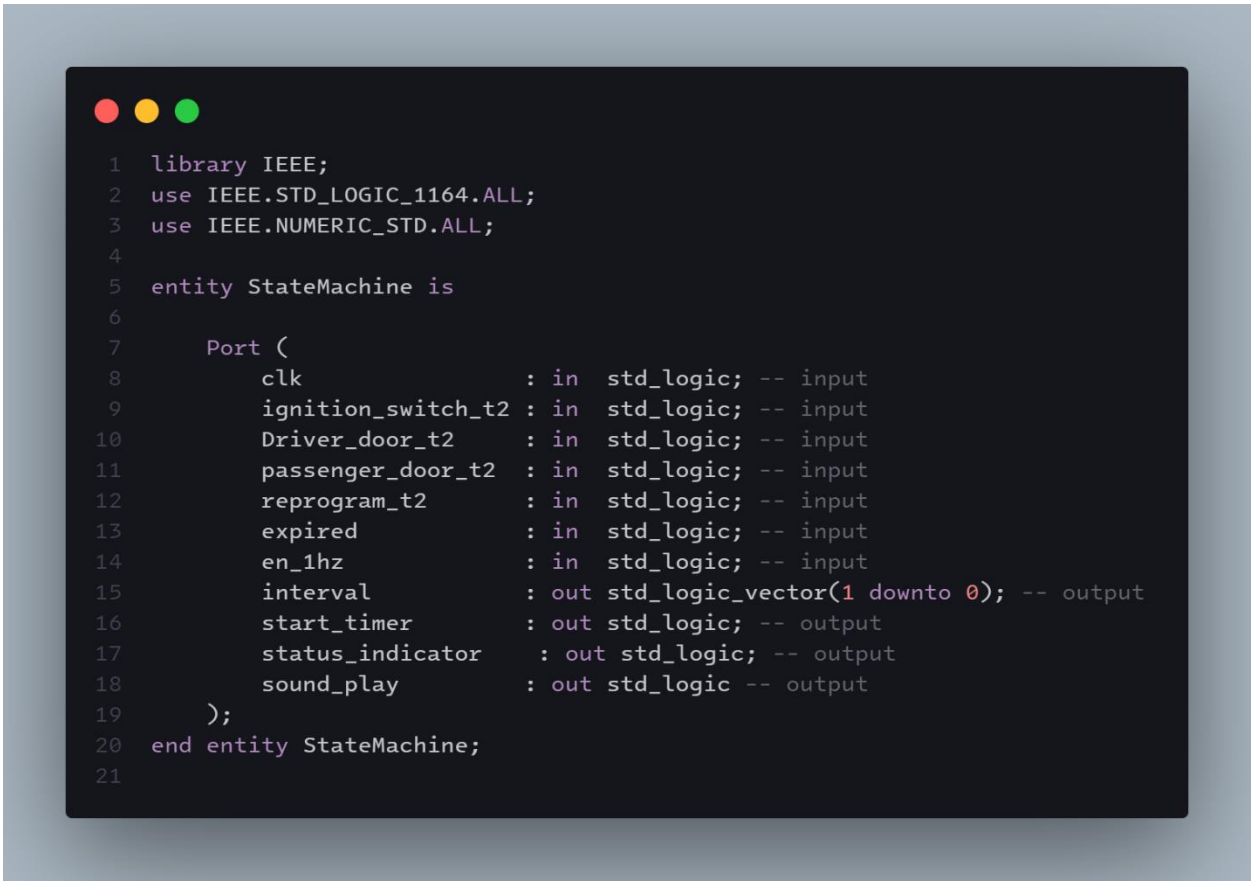
سیگنال internal start timer به عنوان متغیر محلی جایگزین start timer شده تا بتوان از آن در شرط ها استفاده کنیم.

در ادامه در هر state که نیاز به flag start و flag expi بود ، در state قبل از آن این مقادیر ریست شده و مقدار 0 به خود میگیرند.

زمانی که در بعضی از state ها نیاز به تایمر داشتیم ، مقدار internal start timer را برای ماژول تایمر و interval را برای ماژول time parameter و flag start را برای اطمینان از آغاز زمان شمارش و flag expi را برای اطمینان از بدست آمدن flag expired از این state را ست میکنیم.

مثال از کد در خط ۸۱ الی ۹۹ قابل مشاهده میباشد.

روند فعالیت تمام state ها در state machine در دیاگرام قید شده است.



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

entity StateMachine is

    Port (
        clk                : in  std_logic; -- input
        ignition_switch_t2 : in  std_logic; -- input
        Driver_door_t2     : in  std_logic; -- input
        passenger_door_t2  : in  std_logic; -- input
        reprogram_t2       : in  std_logic; -- input
        expired            : in  std_logic; -- input
        en_1hz             : in  std_logic; -- input
        interval           : out std_logic_vector(1 downto 0); -- output
        start_timer        : out std_logic; -- output
        status_indicator    : out std_logic; -- output
        sound_play         : out std_logic -- output
    );
end entity StateMachine;

```

```
architecture Behavioral of StateMachine is
    type state_type is (armed , triggredT_Driver_Delay_orT_Passenger_Delay ,
                        Sound_Alarm_Is_on , Disarmed , Timer_ForT_Alarm_On ,
                        Timer_ForT_Arm_Delay , Waiting_for_Driver_Take_Off ,
                        Door_Is_Open); -- type state
    signal current_state : state_type := armed; -- to save current state
    signal indicator_count : bit := '0'; -- use to counting for changing status_indicators value
    signal flag_start : bit := '0'; -- flag to check that start timer turned on
    signal flag_expi : bit := '0'; -- flag to be sure that expired signal change this state
    signal internal_start_timer : std_logic := '0';  -- Internal signal for start_timer

begin
process(clk)  -- Use an edge-triggered process for state transitions

begin
    if rising_edge(clk) then
        case current_state is -- determine states
            when armed =>
                if en_1hz = '1' then -- counting to change status in indicator each 2 sec
                    indicator_count <= not indicator_count;
                end if;

                if indicator_count = '1' and en_1hz = '1' then
                -- for turnning on status indicator each 2 sec
                    status_indicator <= '1';

                else
                    status_indicator <= '0';
                end if;

                sound_play <= '0'; -- in this case sound is off
                if Driver_door_t2 = '1' or passenger_door_t2 = '1' then
                    -- check for openning door
                    current_state <= triggredT_Driver_Delay_orT_Passenger_Delay;

                elsif ignition_switch_t2 = '1' then -- check for turnning on
                    current_state <= Disarmed;
                end if;

                flag_start <= '0';
                flag_expi <= '0';
                internal_start_timer <= '0';  -- Reset internal timer

            when triggredT_Driver_Delay_orT_Passenger_Delay =>
                status_indicator <= '1'; -- in this case status indicator is always on
                sound_play <= '0'; -- in this case sound is off

                if ignition_switch_t2 = '1' then -- check for turnning on
                    current_state <= Disarmed;
                end if;


                if internal_start_timer = '1' then -- if start timer was 1 turn it off
                    internal_start_timer <= '0';  -- Reset internal timer
                end if;


                -- check expired flag to vector that expired achived in this case and it is not prev value
                if flag_expi = '1' then
                    if expired = '1' then --
                        current_state <= Sound_Alarm_Is_on;
                    end if;
                end if;

```

```
                -- check to selecct delay time
                if passenger_door_t2 = '1' and flag_start = '0' and reprogram_t2 = '0' then
                    interval <= "10";  -- Assuming interval is in unsigned format
                    internal_start_timer <= '1';
                    flag_start <= '1';
                    flag_expi <= '1';
                elsif Driver_door_t2 = '1' and flag_start = '0' and reprogram_t2 = '0' then
                    interval <= "01";  -- Assuming interval is in unsigned format
                    internal_start_timer <= '1';
                    flag_start <= '1';
                    flag_expi <= '1';
                end if;


            when Sound_Alarm_Is_on =>
                status_indicator <= '1';
                sound_play <= '1';
                if Driver_door_t2 = '0' and passenger_door_t2 = '0' then -- check condition to change state
                    current_state <= Timer_ForT_Alarm_On;
                elsif ignition_switch_t2 = '1' then -- check condition to change state
                    current_state <= Disarmed;
                end if;
                flag_start <= '0';
                flag_expi <= '0';

            when Timer_ForT_Alarm_On =>
                status_indicator <= '1';
                sound_play <= '1';

                if ignition_switch_t2 = '1' then -- check condition to change state
                    current_state <= Disarmed;
                end if;



                if internal_start_timer = '1' then
                    internal_start_timer <= '0';  -- Reset internal timer
                end if;


                if flag_expi = '1' then
                    if expired = '1' then
                        current_state <= armed;
                    end if;
                end if;

                -- check expired flag to vector that expired achived in this case and it is not prev value
                if flag_start = '0' and reprogram_t2 = '0' then
                    interval <= "11";  -- Assuming interval is in unsigned format
                    internal_start_timer <= '1';
                    flag_start <= '1';
                    flag_expi <= '1';
                end if;

            when Disarmed =>
                status_indicator <= '0';
                sound_play <= '0';
                if ignition_switch_t2 = '0' then
                    current_state <= Waiting_for_Driver_Take_Off;
                end if;

            when Waiting_for_Driver_Take_Off =>
                status_indicator <= '0';
                sound_play <= '0';

                if Driver_door_t2 = '1' then -- check to openning door then change current state
                    current_state <= Door_Is_Open;
                end if;

            when Door_Is_Open =>
                status_indicator <= '0';
                sound_play <= '0';

                if Driver_door_t2 = '0' and passenger_door_t2 = '0' then -- check to be sure that all doors are closed
                    current_state <= Timer_ForT_Arm_Delay;
                end if;

                flag_start <= '0';
                flag_expi <= '0';

            when Timer_ForT_Arm_Delay =>
                status_indicator <= '0';
                sound_play <= '0';

                if internal_start_timer = '1' then
                    internal_start_timer <= '0';  -- Reset internal timer
                end if;

                if flag_expi = '1' then
                    if expired = '1' then
                        current_state <= armed;
                    end if;
                end if;

                if flag_start = '0' and reprogram_t2 = '0' then
                    interval <= "00";  -- Assuming interval is in unsigned format
                    internal_start_timer <= '1';
                    flag_start <= '1';
                    flag_expi <= '1';
                end if;

            when others =>
                current_state <= armed;  -- Reset on unexpected state
        end case;
    end if;
end process;

start_timer <= internal_start_timer;  -- Drive the output port from the internal signal
end Behavioral;
```


Signals
Waves
Time
clk
current_state
driver_door_t2
en_1hz
expired
flag_expi
flag_start
ignition_switch_t2
indicator_count
internal_start_timer
interval[1:0]
passenger_door_t2
reprogram_t2
sound_play
start_timer
status_indicator
disarmed
waiting_for_driv
door_is_open
timer_fort_arm_delay
armed
triggredt_driver_delay_ort_passenger_delay
sound_a
timer_fort_alarm_on
triggredt_dri
disarmed
waiting_for_driver_take_off