Experiment 5: Tone Synthesizer

Nirmal Shah, Roll Number 190100085
EE-214, WEL, IIT Bombay
January 27, 2021

Overview of the experiment:

The purpose of the experiment is to design a circuit that will generate the seven major notes in the Indian classical music named Sa, Re, Ga, Ma, Pa, Dha, Ni, and Sa (upper octave) on the speaker. This was done by modifying the frequency of the clock(50Mhz) on which Krypton board was running.

The frequency was modulated by using the formula count=$\frac{50Mhz}{2*f}$ where f = desired frequency. The signal would be 1 for count cycles and 0 for other count cycles, of the 50MHz clock frequency.

Approach to the experiment:

Found out the count values for each of the notes, using the above formula. After counting this much value the signal would toggle itself, giving us the desired frequency.

Note	Frequency(Hz)	Count value
Sa	240	104168
Re	270	92593
Ga	300	83333
Ma	320	78125
Pa	360	69444
Dha	400	62500
Ni	450	55555
Sa(Upper octave)	480	52083

Then I defined 8 states of FSM, using one-hot encoding. Where Sa would be 0000_0001,
Re would be 0000_0010, ….., Ni would be 0100_0000, and Sa(Upper octave) - 1000_0000, which is the same mapping for the switches, i.e, when switch 1 is ON, so it would produce the value 0000_0001 which is exactly the same state as that of Sa

I have not developed a priority encoder, but 8-1 MUX.

Sa would be played when we switch 1 is one. Similarly for others too, the same analogy and explanation can be applied

Design document and VHDL code if relevant:

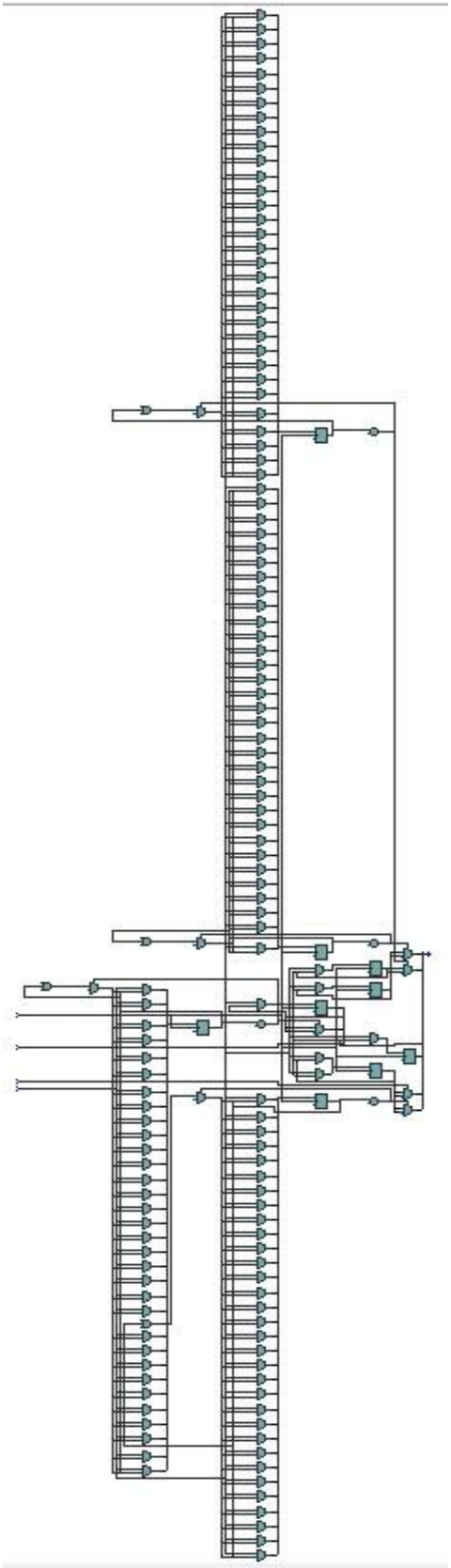
We have defined 8 states, using 1-hot encoding. And then used the help of switch which have the same states as that of our mapped states.

```
library IEEE;
use IEEE.std_logic_1164.all;



entity toneGenerator is
port (toneOut : out std_logic; --this pin will give your notes output
clk : in std_logic;
LED : out std_logic_vector(7 downto 0);
switch : in std_logic_vector(7 downto 0));
end entity toneGenerator;

architecture m of toneGenerator is
constant state_sa1 : std_logic_vector(7 downto 0):=(0=>'1', others=>'0');
constant state_re1 : std_logic_vector(7 downto 0):=(1=>'1', others=>'0');
constant state_ga1 : std_logic_vector(7 downto 0):=(2=>'1', others=>'0');
constant state_ma1 : std_logic_vector(7 downto 0):=(3=>'1', others=>'0');
constant state_pa1 : std_logic_vector(7 downto 0):=(4=>'1', others=>'0');
constant state_dh1 : std_logic_vector(7 downto 0):=(5=>'1', others=>'0');
constant state_ni1 : std_logic_vector(7 downto 0):=(6=>'1', others=>'0');
constant state_sa2 : std_logic_vector(7 downto 0):=(7=>'1', others=>'0');


begin
process(clk,switch)
variable csa1,cre1,cga1,cma1,cpa1,cdh1,cni1,csa2         : integer :=1;
variable sa1,re1,ga1,ma1,pa1,dh1,ni1,sa2          : std_logic:='1';

begin
        if(rising_edge(clk)) then

        case switch is
                when state_sa1=>

                        if (csa1 = 104168) then--240Hz
                        csa1 := 1;
                        sa1 := not sa1;
                        else
                        csa1 := csa1 + 1;
                        end if;
                        toneOut <= sa1;
                        LED <= (0 => '1', others => '0');
```

```
when state_re1=>
        if (cre1 = 92593) then--270Hz
        cre1 := 1;
        re1 := not re1;
        else
        cre1 := cre1 + 1;
        end if;
        toneOut <= re1;
        LED <= (1 => '1', others => '0');

when state_ga1=>
        if (cga1 = 83333) then--300Hz
        cga1 := 1;
        ga1 := not ga1;
        else
        cga1 := cga1 + 1;
        end if;
        toneOut <= ga1;
        LED <= (2 => '1', others => '0');

when state_ma1=>
        if (cma1 = 78125) then--320Hz
        cma1 := 1;
        ma1 := not ma1;
        else
        cma1 := cma1 + 1;
        end if;
        toneOut <= ma1;
        LED <= (3 => '1', others => '0');

when state_pa1=>
        if (cpa1 = 69444) then--360Hz
        cpa1 := 1;
        pa1 := not pa1;
        else
        cpa1 := cpa1 + 1;
        end if;
        toneOut <= pa1;
        LED <= (4 => '1', others => '0');

when state_dh1=>
        if (cdh1 = 62500) then--400Hz
        cdh1 := 1;
        dh1 := not dh1;
        else
        cdh1 := cdh1 + 1;
        end if;
        toneOut <= dh1;
        LED <= (5 => '1', others => '0');

when state_ni1=>
        if (cni1 = 55555) then--450Hz
        cni1 := 1;
        ni1 := not ni1;
```

```
                        else
                        cni1 := cni1 + 1;
                        end if;
                        toneOut <= ni1;
                        LED <= (6 => '1', others => '0');

                when state_sa2=>
                        if (csa2 = 52083) then--480Hz
                        csa2 := 1;
                        sa2 := not sa2;
                        else
                        csa2 := csa2 + 1;
                        end if;
                        toneOut <= sa2;
                        LED <= (7 => '1', others => '0');


                when others=>
                        LED<=(others=>'0');



        end case;
        end if;
end process;
end m;
```

RTL View:



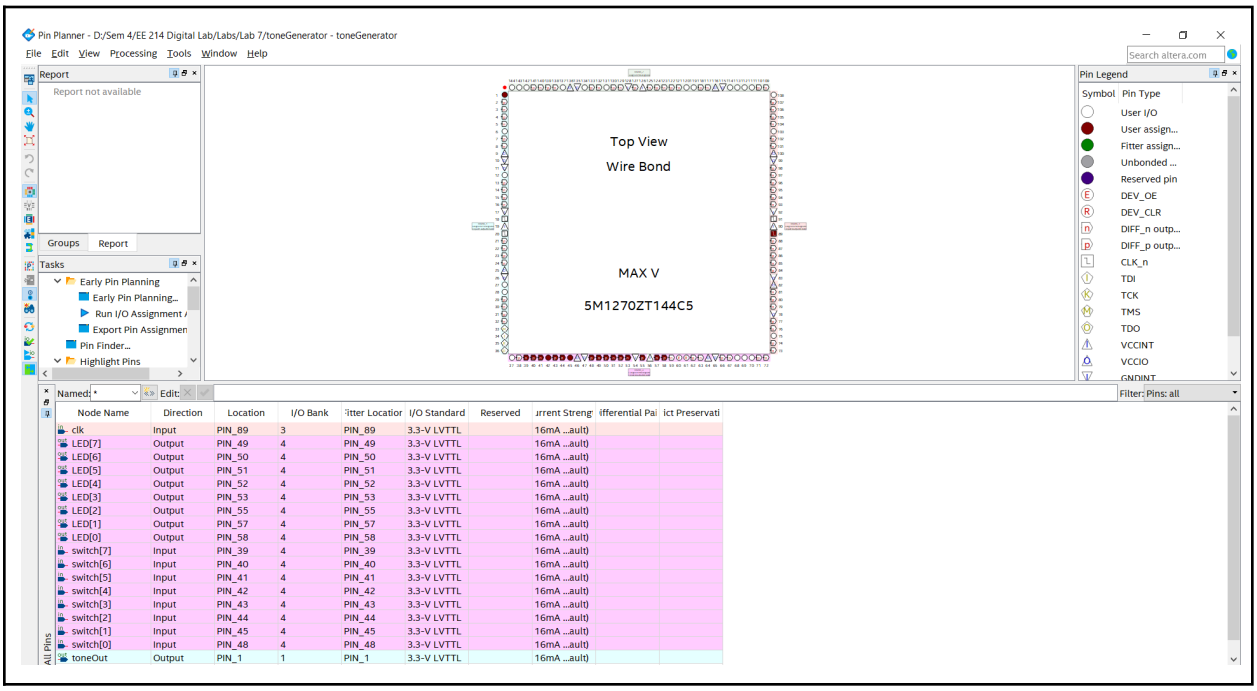
DUT Input/Output Format:

```
toneOut : out std_logic;
clk : in std_logic;
LED : out std_logic_vector(7 downto 0);
switch : in std_logic_vector(7 downto 0));
```

Testbench Used:-

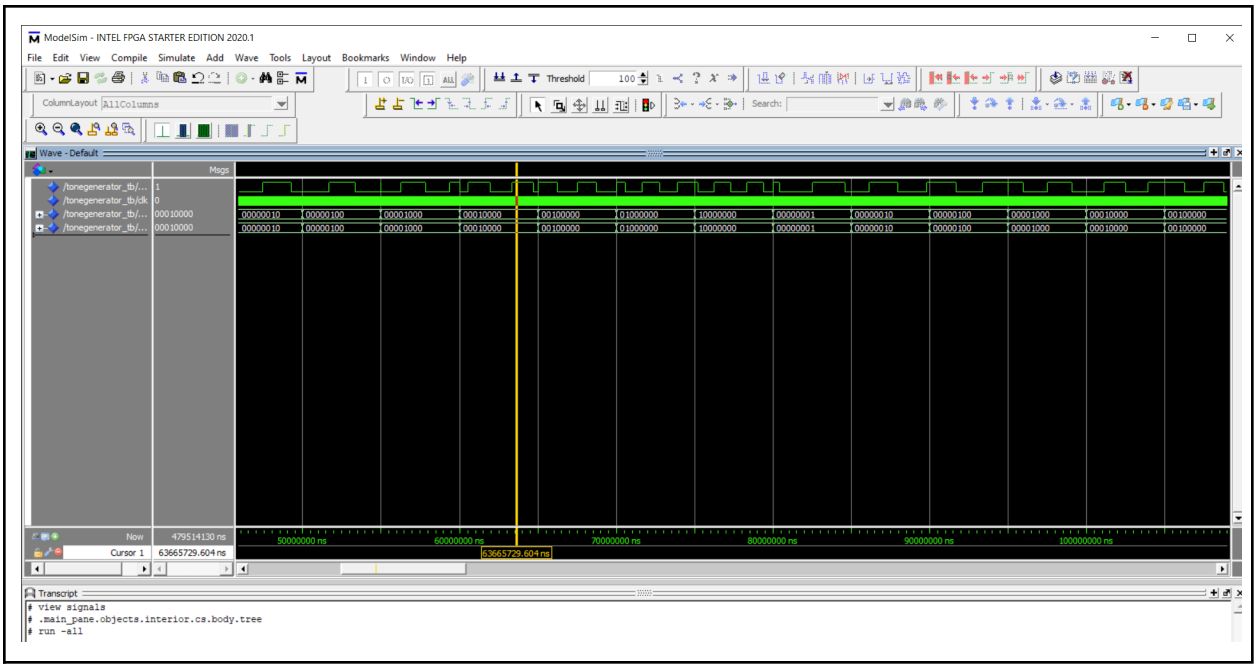
```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
entity toneGenerator_tb is
end entity toneGenerator_tb;

architecture bhv of toneGenerator_tb is
component toneGenerator is
port (toneOut : out std_logic;
clk : in std_logic;
LED : out std_logic_vector(7 downto 0);
switch : in std_logic_vector(7 downto 0));
end component;
signal toneOut : std_logic := '0';
signal clk: std_logic := '0';
signal switch, LED : std_logic_vector(7 downto 0) := (others => '0');
constant clk_period : time := 20 ns;
begin
dut_instance: toneGenerator port map(toneOut, clk, LED, switch);
clk <= not clk after clk_period/2 ;
stim_proc: process
begin
for i in 0 to 7 loop --This loop will run by making only 1 switch on at a time
switch <= (i => '1', others => '0');
wait for 5 ms;
end loop;
end process;
end bhv;
-- Run the testbench for 40 ms
```

Pin Planning:-

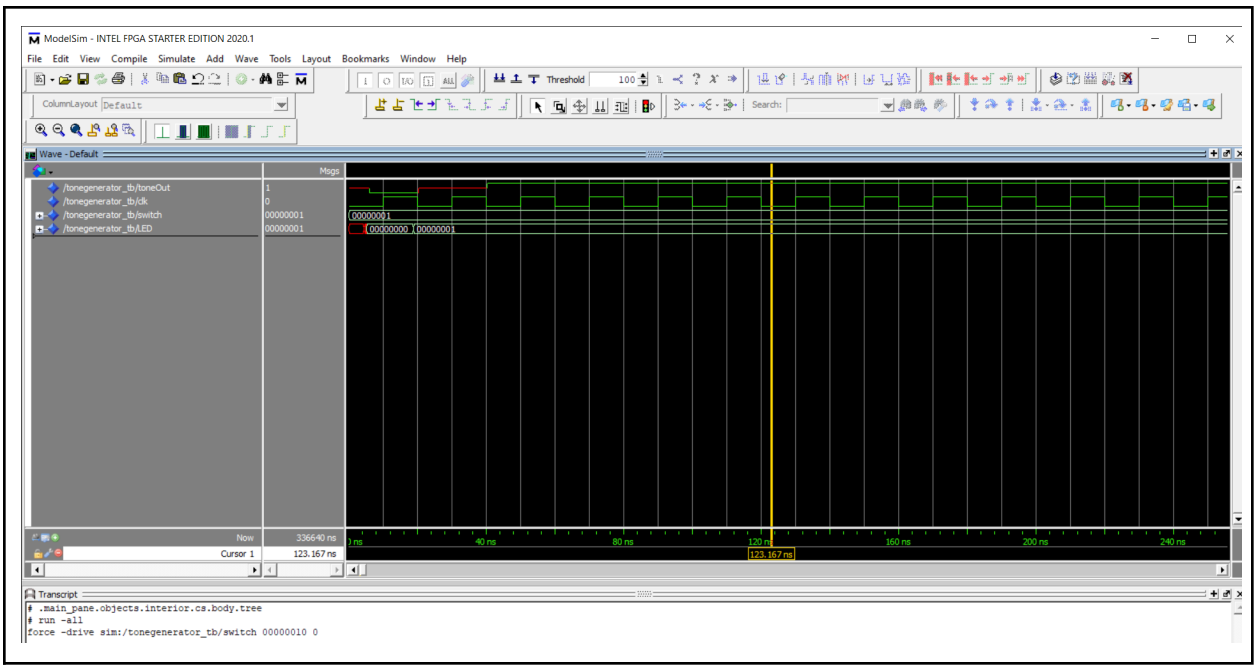


Node Name	Direction	Location	I/O Bank	itter Locatior	I/O Standard	Reserved	urrent Streng	ifferential Pai	ict Preservati
clk	Input	PIN_89	3	PIN_89	3.3-V LVTTL		16mA ...ault)		
LED[7]	Output	PIN_49	4	PIN_49	3.3-V LVTTL		16mA ...ault)		
LED[6]	Output	PIN_50	4	PIN_50	3.3-V LVTTL		16mA ...ault)		
LED[5]	Output	PIN_51	4	PIN_51	3.3-V LVTTL		16mA ...ault)		
LED[4]	Output	PIN_52	4	PIN_52	3.3-V LVTTL		16mA ...ault)		
LED[3]	Output	PIN_53	4	PIN_53	3.3-V LVTTL		16mA ...ault)		
LED[2]	Output	PIN_55	4	PIN_55	3.3-V LVTTL		16mA ...ault)		
LED[1]	Output	PIN_57	4	PIN_57	3.3-V LVTTL		16mA ...ault)		
LED[0]	Output	PIN_58	4	PIN_58	3.3-V LVTTL		16mA ...ault)		
switch[7]	Input	PIN_39	4	PIN_39	3.3-V LVTTL		16mA ...ault)		
switch[6]	Input	PIN_40	4	PIN_40	3.3-V LVTTL		16mA ...ault)		
switch[5]	Input	PIN_41	4	PIN_41	3.3-V LVTTL		16mA ...ault)		
switch[4]	Input	PIN_42	4	PIN_42	3.3-V LVTTL		16mA ...ault)		
switch[3]	Input	PIN_43	4	PIN_43	3.3-V LVTTL		16mA ...ault)		
switch[2]	Input	PIN_44	4	PIN_44	3.3-V LVTTL		16mA ...ault)		
switch[1]	Input	PIN_45	4	PIN_45	3.3-V LVTTL		16mA ...ault)		
switch[0]	Input	PIN_48	4	PIN_48	3.3-V LVTTL		16mA ...ault)		
toneOut	Output	PIN_1	1	PIN_1	3.3-V LVTTL		16mA ...ault)		

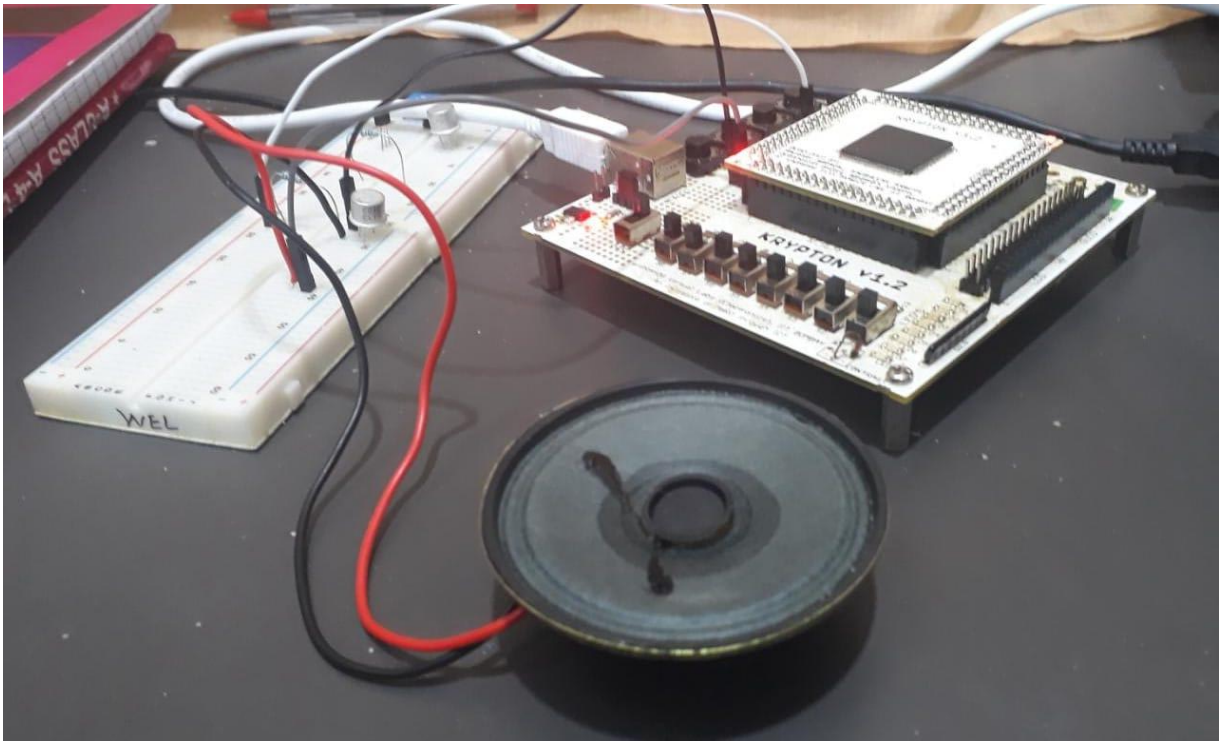
RTL Simulation:



Gate Level Simulation:-



Krypton board:



Observations:

Attaching the link of the video here:-
https://drive.google.com/file/d/1Q-sLgZjrxIUYwh5eucLvNAmjqtZUHmxv/view?usp=sharing

References:

Clock Divider Tutorial by Teaching Assistant Mr. Sandesh Goyal