Experiment 6: Music Synthesizer

Rathour Param Jitendrakumar Roll Number: 190070049
EE-214, WEL, IIT Bombay
April 03, 2021

Overview of the experiment:

The purpose of this experiment is to design an automated music synthesizer using FSM and the seven major notes in the Indian classical music named sa, re, ga, ma, pa, dha, ni (lower octave) & Sa (upper octave). Describe it in VHDL using Behavioral Modelling, simulate with the given test-bench using Quartus.

I used the provided skeleton code, designed the circuit accordingly, verified my implementation using Register-transfer level simulation & observed the music by mapping it to the Krypton board interfaced with speaker.

This report contains my approach to the experiment, the circuit's design with the relevant code followed by the RTL netlist view, PIN planning and design confirmation by output waveforms of RTL.

Approach to the experiment:

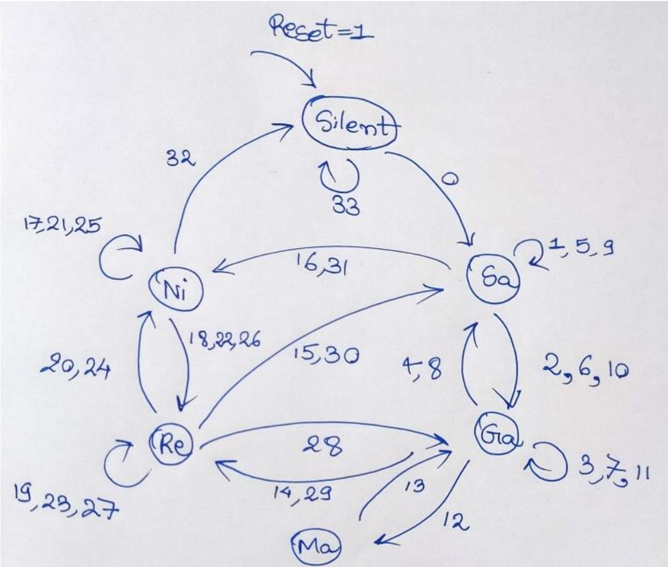
Note	Sa	Ga	Sa	Ga	Sa	Ga	Ma	Ga	Re	Sa
Duration	0.5s	0.5s	0.5s	0.5s	0.5s	0.5s	0.25s	0.25s	0.25s	0.25s
Count	1,2	3,4	5,6	7,8	9,10	11,12	13	14	15	16

Note	Ni	Re	Ni	Re	Ni	Re	Ga	Re	Sa	Ni
Duration	0.5s	0.5s	0.5s	0.5s	0.5s	0.5s	0.25s	0.25s	0.25s	0.25s
Count	17,18	19,20	21,22	23,24	25,26	27,28	29	30	31	32

I used FSM based approach such that note will be played one after another in the given sequence. I generated a 4Hz Clock so that a note is played for a particular duration (in multiples of 0.25 seconds) as shown in table.

This FSM was realised using Behavioral approach (switch statement). With 5 distinct notes to be played and a silent state, we can make FSM with 6 states (Silent, Sa, Re, Ga, Ma, Ni). The corresponding State Transistion Diagram and State Transition Table are shown.

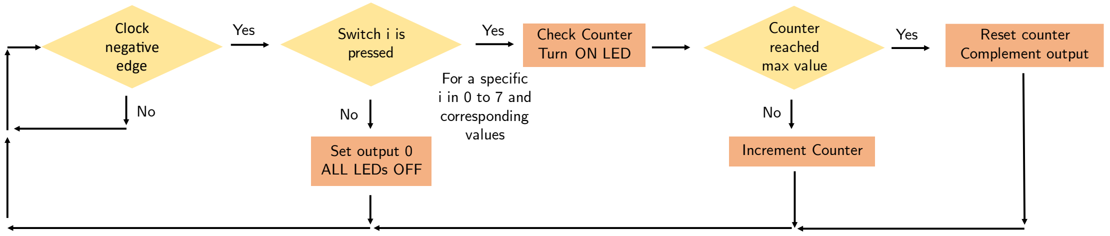
Transitions occur at positive edge of 4Hz clock. In my approach, present state and present value of count is used to go to next state, and then I increment the value of count.
Press Reset to go back to Silent state



State Transistion Diagram

State Transition Table

Current State	Reset	Count	Next State
Silent	1	X	Silent
Silent	0	0	Sa
Silent	0	33	Silent
Sa	1	X	Silent
Sa	0	1	Sa
Sa	0	2	Ga
Sa	0	5	Sa
Sa	0	6	Ga
Sa	0	9	Sa
Sa	0	10	Ga
Sa	0	16	Ni
Sa	0	31	Ni
Re	1	X	Silent
Re	0	15	Sa
Re	0	19	Re
Re	0	20	Ni
Re	0	23	Re
Re	0	24	Ni
Re	0	27	Re
Re	0	28	Ga
Re	0	30	Sa
Ga	1	X	Silent
Ga	0	3	Ga
Ga	0	4	Sa
Ga	0	7	Ga
Ga	0	8	Sa
Ga	0	11	Ga
Ga	0	12	Ma
Ga	0	14	Re
Ga	0	29	Re
Ma	1	X	Silent
Ma	0	13	Ga
Ni	1	X	Silent
Ni	0	17	Ni
Ni	0	18	Re
Ni	0	21	Ni
Ni	0	22	Re
Ni	0	25	Ni
Ni	0	26	Re
Ni	0	32	Silent
Pa	X	X	Silent
Dha	X	X	Silent


Clock negative edge
Yes
No
Switch i is pressed
Yes
No
Check Counter Turn ON LED
For a specific i in 0 to 7 and corresponding values
Set output 0 ALL LEDs OFF
Counter reached max value
Yes
No
Increment Counter
Reset counter Complement output

Tone Generation

Tone Generation appoach as shown in flow chart.
To generate these frequencies, I used clock divider and onboard clock, calculated count for each note using

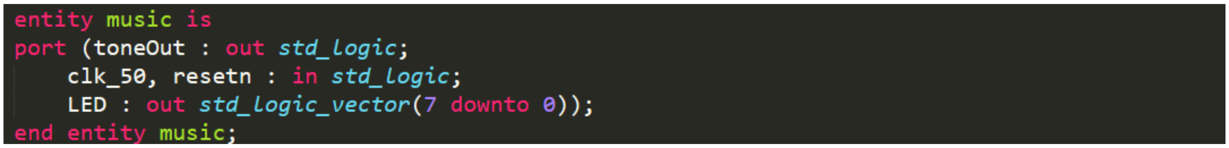
Count = 50 MHz / (2 * f)

Note	Frequency (Hz)	Calculations (Truncated till 2 decimal places)		Count Value (Accurate to nearest integer)
Sa	240	50 * 10^6 / (2 * 240)	104166.66	104167
Re	270	50 * 10^6 / (2 * 270)	92592.59	92593
Ga	300	50 * 10^6 / (2 * 300)	83333.33	83333
Ma	320	50 * 10^6 / (2 * 320)	78125	78125
Pa	360	50 * 10^6 / (2 * 360)	69444.44	69444
Dha	400	50 * 10^6 / (2 * 400)	62500	62500
Ni	225	50 * 10^6 / (2 * 225)	111111.11	111111
Sa (upper octave)	480	50 * 10^6 / (2 * 480)	52083.33	52083
Clock_music	4	50 * 10^6 / (2 * 4)	6250000	62500000

Design document and VHDL code if relevant:

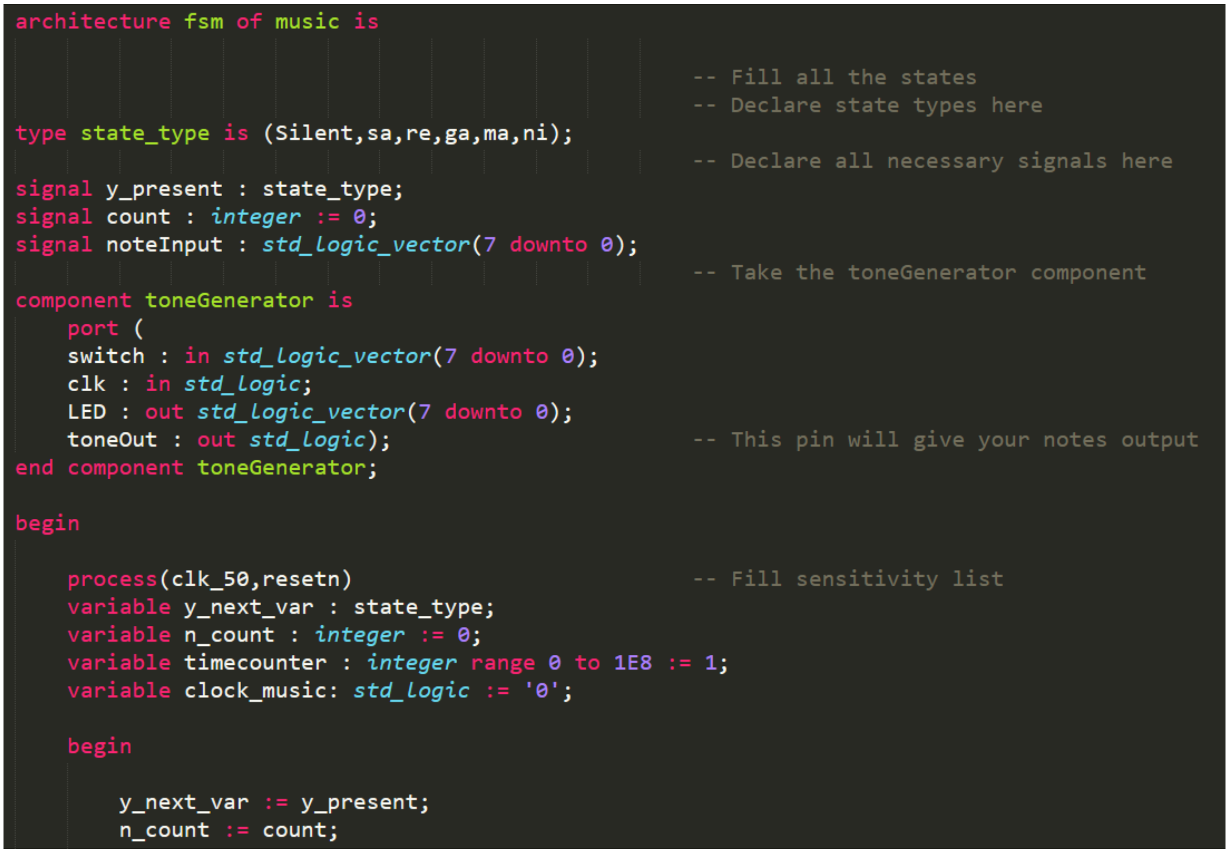
This whole design is part of music with 2 input bit & 9 output (with 8 LED bits to confirm our design).
See music Input/Output Format
I am using Behavioral Modelling for this experiment.
Modified toneGenerator entity from previous experiment is used.
Code can be broken down as shown in approach

Entity declaration



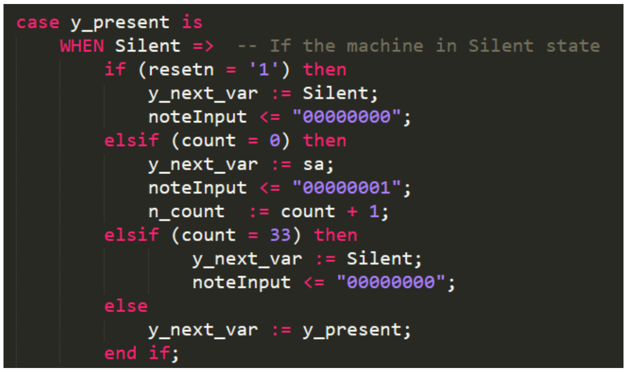
```
entity music is
port (toneOut : out std_logic;
    clk_50, resetn : in std_logic;
    LED : out std_logic_vector(7 downto 0));
end entity music;
```

Architechture of toneGenerator

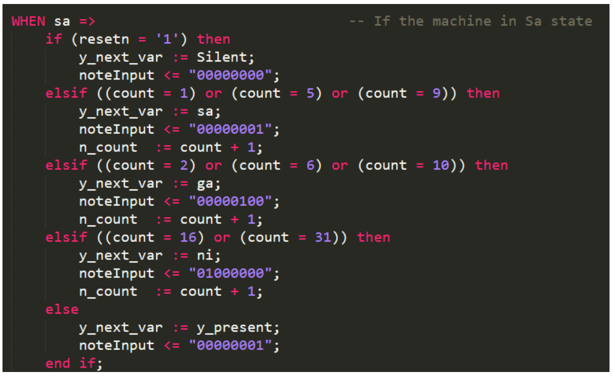


```
architecture fsm of music is

                                                    -- Fill all the states
                                                    -- Declare state types here
type state_type is (Silent,sa,re,ga,ma,ni);
                                                    -- Declare all necessary signals here
signal y_present : state_type;
signal count : integer := 0;
signal noteInput : std_logic_vector(7 downto 0);
                                                    -- Take the toneGenerator component
component toneGenerator is
    port (
    switch : in std_logic_vector(7 downto 0);
    clk : in std_logic;
    LED : out std_logic_vector(7 downto 0);
    toneOut : out std_logic);                       -- This pin will give your notes output
end component toneGenerator;

begin

    process(clk_50,resetn)                          -- Fill sensitivity list
    variable y_next_var : state_type;
    variable n_count : integer := 0;
    variable timecounter : integer range 0 to 1E8 := 1;
    variable clock_music: std_logic := '0';

    begin

        y_next_var := y_present;
        n_count := count;
```

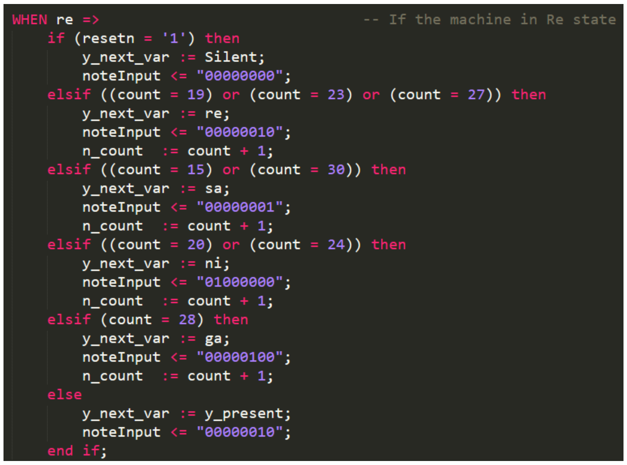
Code for each state



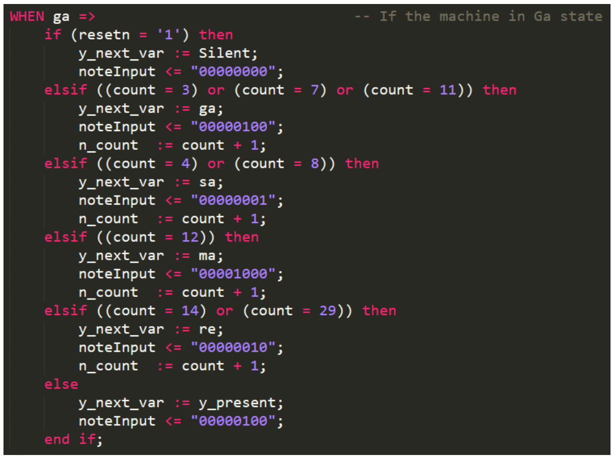
```
case y_present is
    WHEN Silent =>  -- If the machine in Silent state
        if (resetn = '1') then
            y_next_var := Silent;
            noteInput <= "00000000";
        elsif (count = 0) then
            y_next_var := sa;
            noteInput <= "00000001";
            n_count  := count + 1;
        elsif (count = 33) then
                y_next_var := Silent;
                noteInput <= "00000000";
        else
            y_next_var := y_present;
        end if;
```



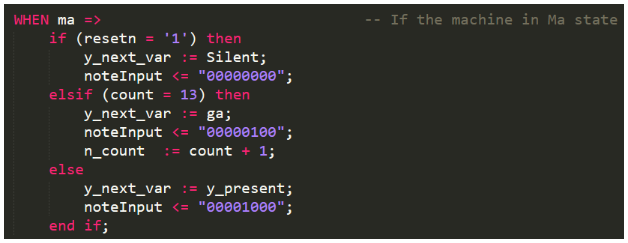
```
WHEN sa =>                                    -- If the machine in Sa state
    if (resetn = '1') then
        y_next_var := Silent;
        noteInput <= "00000000";
    elsif ((count = 1) or (count = 5) or (count = 9)) then
        y_next_var := sa;
        noteInput <= "00000001";
        n_count  := count + 1;
    elsif ((count = 2) or (count = 6) or (count = 10)) then
        y_next_var := ga;
        noteInput <= "00000100";
        n_count  := count + 1;
    elsif ((count = 16) or (count = 31)) then
        y_next_var := ni;
        noteInput <= "01000000";
        n_count  := count + 1;
    else
        y_next_var := y_present;
        noteInput <= "00000001";
    end if;
```



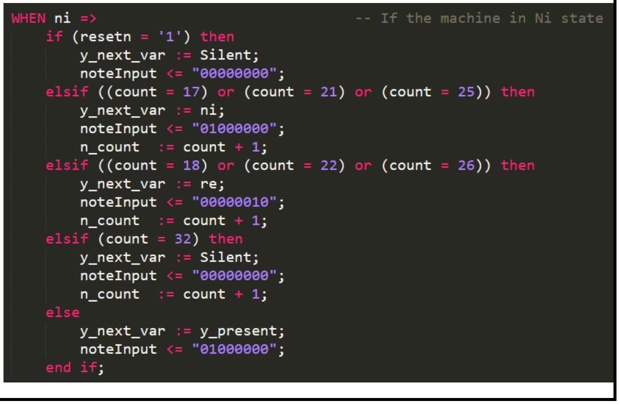
```
WHEN re =>                                    -- If the machine in Re state
    if (resetn = '1') then
        y_next_var := Silent;
        noteInput <= "00000000";
    elsif ((count = 19) or (count = 23) or (count = 27)) then
        y_next_var := re;
        noteInput <= "00000010";
        n_count  := count + 1;
    elsif ((count = 15) or (count = 30)) then
        y_next_var := sa;
        noteInput <= "00000001";
        n_count  := count + 1;
    elsif ((count = 20) or (count = 24)) then
        y_next_var := ni;
        noteInput <= "01000000";
        n_count  := count + 1;
    elsif (count = 28) then
        y_next_var := ga;
        noteInput <= "00000100";
        n_count  := count + 1;
    else
        y_next_var := y_present;
        noteInput <= "00000010";
    end if;
```



```
WHEN ga =>                                    -- If the machine in Ga state
    if (resetn = '1') then
        y_next_var := Silent;
        noteInput <= "00000000";
    elsif ((count = 3) or (count = 7) or (count = 11)) then
        y_next_var := ga;
        noteInput <= "00000100";
        n_count  := count + 1;
    elsif ((count = 4) or (count = 8)) then
        y_next_var := sa;
        noteInput <= "00000001";
        n_count  := count + 1;
    elsif ((count = 12)) then
        y_next_var := ma;
        noteInput <= "00001000";
        n_count  := count + 1;
    elsif ((count = 14) or (count = 29)) then
        y_next_var := re;
        noteInput <= "00000010";
        n_count  := count + 1;
    else
        y_next_var := y_present;
        noteInput <= "00000100";
    end if;
```

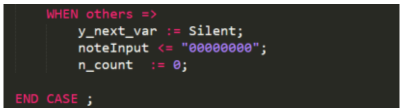


```
WHEN ma =>                                    -- If the machine in Ma state
    if (resetn = '1') then
        y_next_var := Silent;
        noteInput <= "00000000";
    elsif (count = 13) then
        y_next_var := ga;
        noteInput <= "00000100";
        n_count  := count + 1;
    else
        y_next_var := y_present;
        noteInput <= "00001000";
    end if;
```



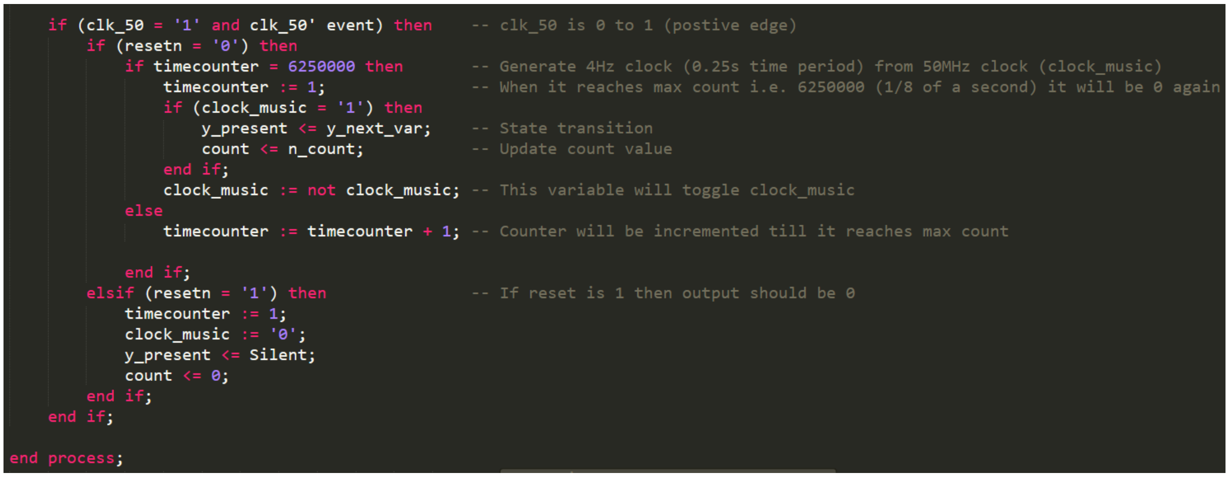
```
WHEN ni =>                                    -- If the machine in Ni state
    if (resetn = '1') then
        y_next_var := Silent;
        noteInput <= "00000000";
    elsif ((count = 17) or (count = 21) or (count = 25)) then
        y_next_var := ni;
        noteInput <= "01000000";
        n_count  := count + 1;
    elsif ((count = 18) or (count = 22) or (count = 26)) then
        y_next_var := re;
        noteInput <= "00000010";
        n_count  := count + 1;
    elsif (count = 32) then
        y_next_var := Silent;
        noteInput <= "00000000";
        n_count  := count + 1;
    else
        y_next_var := y_present;
        noteInput <= "01000000";
    end if;
```

Else Condition



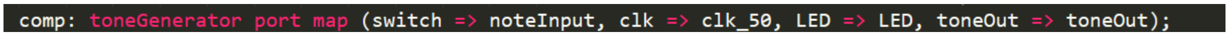
```
    WHEN others =>
        y_next_var := Silent;
        noteInput <= "00000000";
        n_count  := 0;

END CASE ;
```

Clock generation and state transition



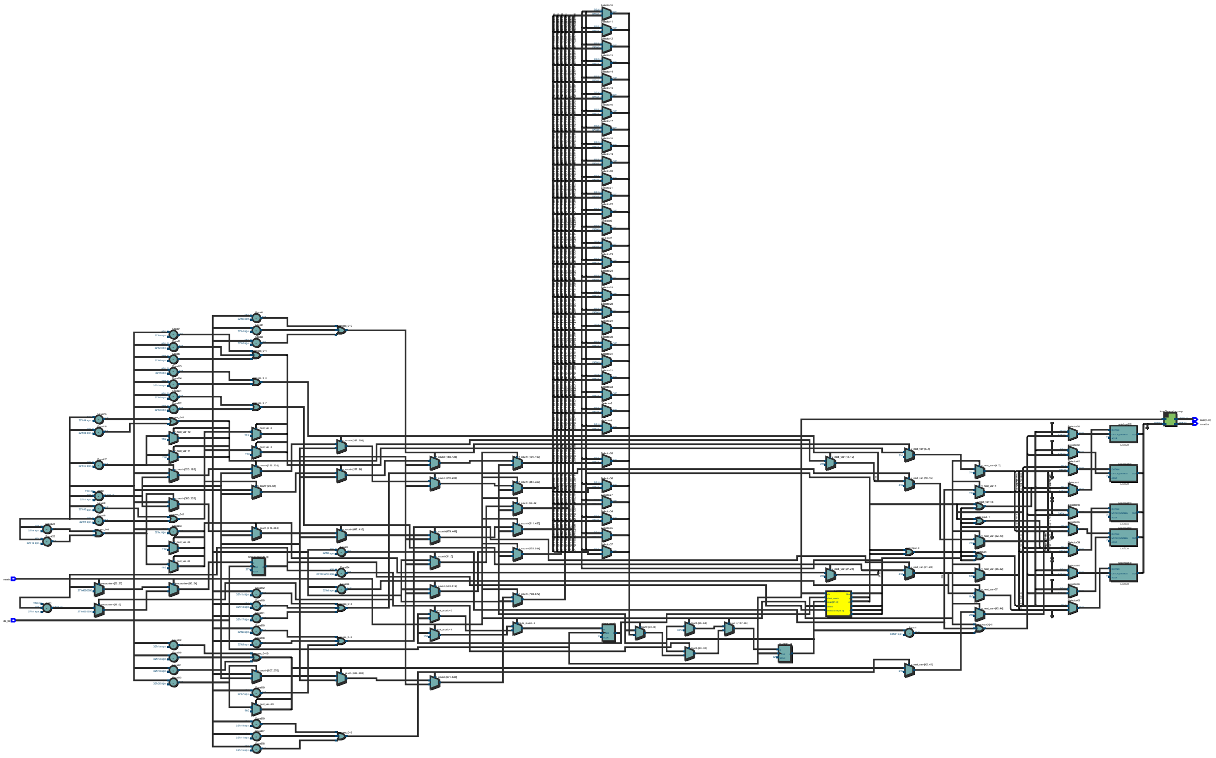
```
    if (clk_50 = '1' and clk_50' event) then    -- clk_50 is 0 to 1 (postive edge)
        if (resetn = '0') then
            if timecounter = 6250000 then       -- Generate 4Hz clock (0.25s time period) from 50MHz clock (clock_music)
                timecounter := 1;               -- When it reaches max count i.e. 6250000 (1/8 of a second) it will be 0 again
                if (clock_music = '1') then
                    y_present <= y_next_var;    -- State transition
                    count <= n_count;           -- Update count value
                end if;
                clock_music := not clock_music; -- This variable will toggle clock_music
            else
                timecounter := timecounter + 1; -- Counter will be incremented till it reaches max count

            end if;
        elsif (resetn = '1') then               -- If reset is 1 then output should be 0
            timecounter := 1;
            clock_music := '0';
            y_present <= Silent;
            count <= 0;
        end if;
    end if;

end process;
```

Instantiate toneGenerator component

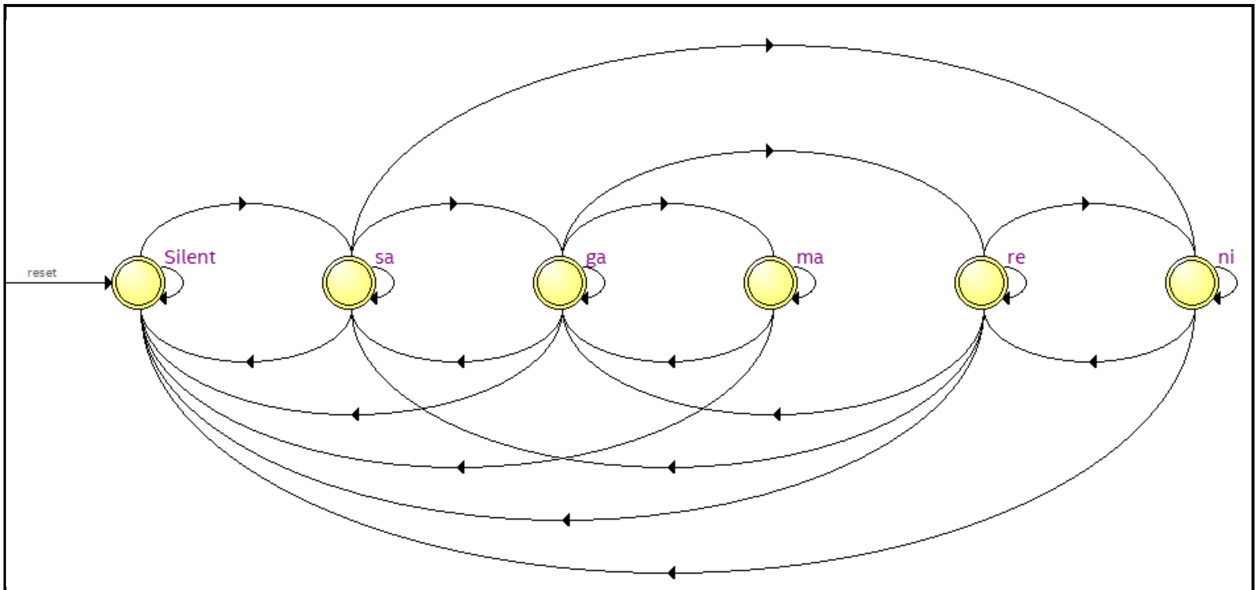


```
comp: toneGenerator port map (switch => noteInput, clk => clk_50, LED => LED, toneOut => toneOut);
```

Netlist View:

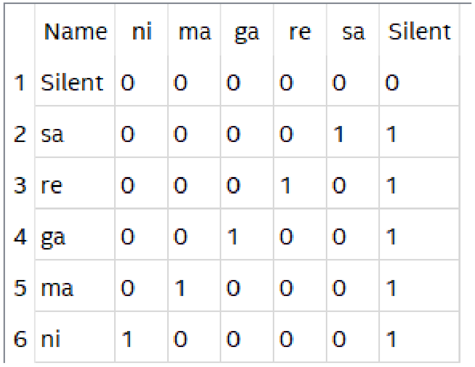


RTL viewer



FSM viewer

Encoding (one hot)

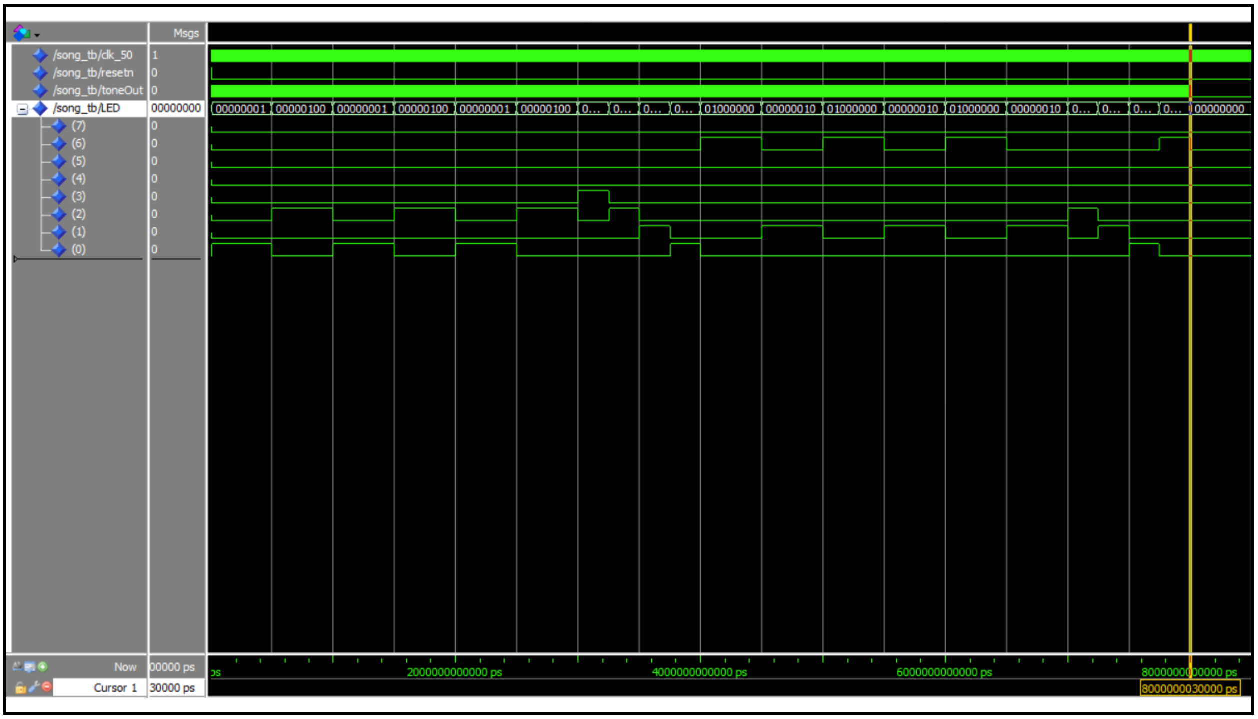


	Name	ni	ma	ga	re	sa	Silent
1	Silent	0	0	0	0	0	0
2	sa	0	0	0	0	1	1
3	re	0	0	0	1	0	1
4	ga	0	0	1	0	0	1
5	ma	0	1	0	0	0	1
6	ni	1	0	0	0	0	1

music Input/Output Format:

Input Format: CS	(C (Onboard Clock) & S (Reset Switch))
Output Format: $OL_8L_7L_6L_5L_4L_3L_2L_1$	(O is output & 8-bit number L (L_8 is MSB L_1 is LSB))
Here, output to speaker oscillates between 0 and 1 according to frequencies of notes and L_i will glow (other LEDs are OFF) when corresponding note is played.	

RTL Simulation:



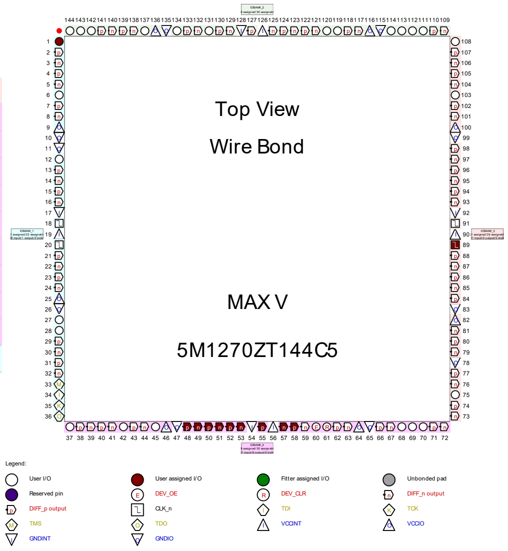
Gate-level Simulation:

Gate-level Simulation takes a lot of time and my computer crashes.

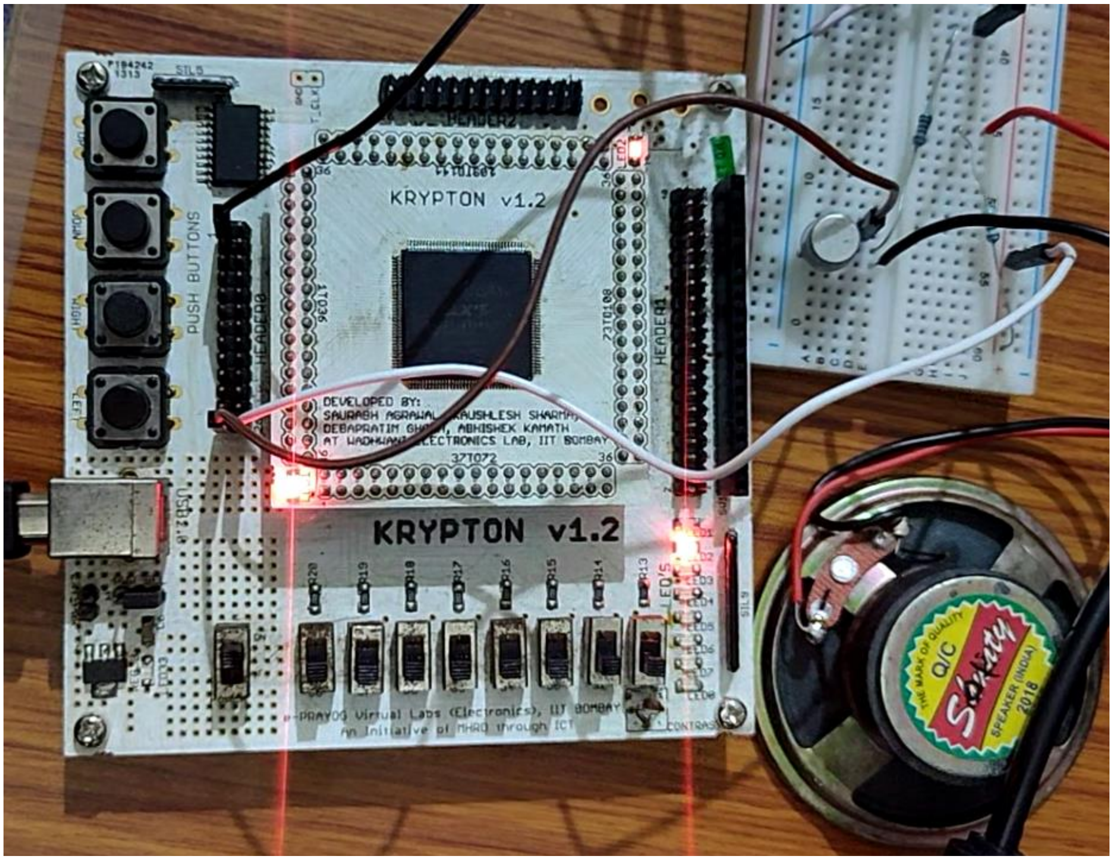
Krypton board:

Pin Planning (Note: VCC at PIN 26 and GND at PIN 25)

Node Name	Direction	Location	I/O Bank	Fitter Location
clk_50	Input	PIN_89	3	PIN_89
LED[7]	Output	PIN_58	4	PIN_58
LED[6]	Output	PIN_57	4	PIN_57
LED[5]	Output	PIN_55	4	PIN_55
LED[4]	Output	PIN_53	4	PIN_53
LED[3]	Output	PIN_52	4	PIN_52
LED[2]	Output	PIN_51	4	PIN_51
LED[1]	Output	PIN_50	4	PIN_50
LED[0]	Output	PIN_49	4	PIN_49
resetn	Input	PIN_48	4	PIN_48
toneOut	Output	PIN_1	1	PIN_1



Board


KRYPTON v1.2
PUSH BUTTONS
UP
DOWN
HIGH
LEFT
HEADER0
HEADER1
HEADER2
DEVELOPED BY:
SAURABH AGRAWAL, KAUSHLESH SHARMA,
DEBAPRATIM GHOSH, ABHISHEK KAMATH
AT WADHWANI ELECTRONICS LAB, IIT BOMBAY
KRYPTON v1.2
USB2.0
LEDS
e-PRAYOG Virtual Labs (Electronics), IIT BOMBAY
An Initiative of MHRD through ICT
CONTRAST
THE MARK OF QUALITY
Q/C
SPEAKER (INDIA)
2018

Observations:

Observed frequencies from RTL simulation are very close to required frequencies
Music starts as soon as board is powered ON. By toggling reset switch (S1) music can be turned OFF when reset is ON and then turned ON again by swithing OFF reset pin.

References:

Tertulien Ndjountche *Digital Electronics 3 Finite State Machines Circuits* Wiley