Experiment 6: Music Synthesizer

Manideep Vudayagiri, Roll Number 190070074
EE-214, WEL, IIT Bombay
March 31st, 2021

Overview of the experiment:

The purpose of the experiment was to generate and play Musical notes in a particular sequence in order to produce a tune.

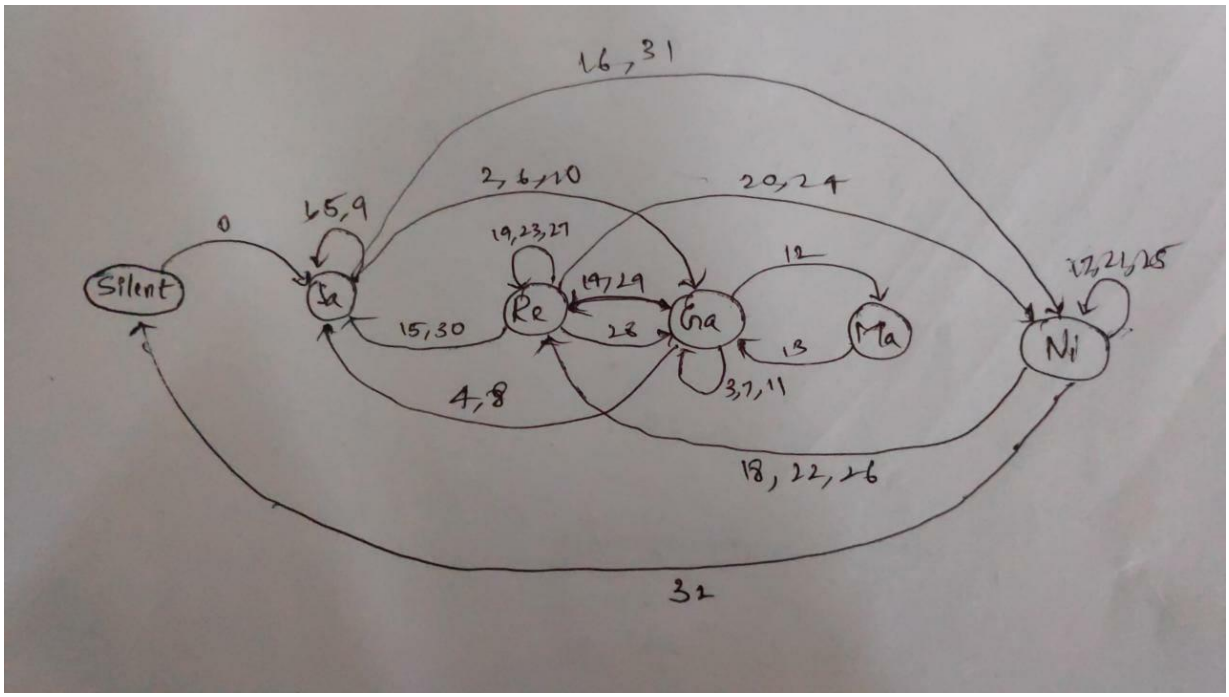
We use an FSM to automate the sequence with notes of different frequencies as states. To form a sequence, count values with 'n' corresponding to the n^{th} 0.25 seconds were given to the notes to be played. The states were the musical octet along with a silent note played at the beginning of the sequence. The state transitions occur at the negative edge of a 4Hz clock which was implemented using a clock divider. This was implemented using a behavioral style design in VHDL and then simulated and, tested on the Krypton board with the musical sequence playing on an output speaker.

I've summarized the approach, VHDL design, simulations and the result of the experiment.

Approach to the experiment:

The count value table given is filled as follows:

Note	**Sa**	**Ga**	**Sa**	**Ga**	**Sa**	**Ga**	**Ma**	**Ga**	**Re**	**Sa**
Duration (in sec)	0.5	0.5	0.5	0.5	0.5	0.5	0.25	0.25	0.25	0.25
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	0.5	0.5	0.5	0.5	0.5	0.5	0.25	0.25	0.25	0.25
Count	17,18	19,20	21,22	23,24	25,26	27,28	29	30	31	32



Design document and VHDL code if relevant:

Song_tb.vhd: Test bench code for the whole design.
ToneGenerator.vhd: Produces waveforms of different given frequencies.
Music.vhd: Main design. Implements the FSM and outputs the corresponding signals for LEDs and input for the ToneGenerator component.

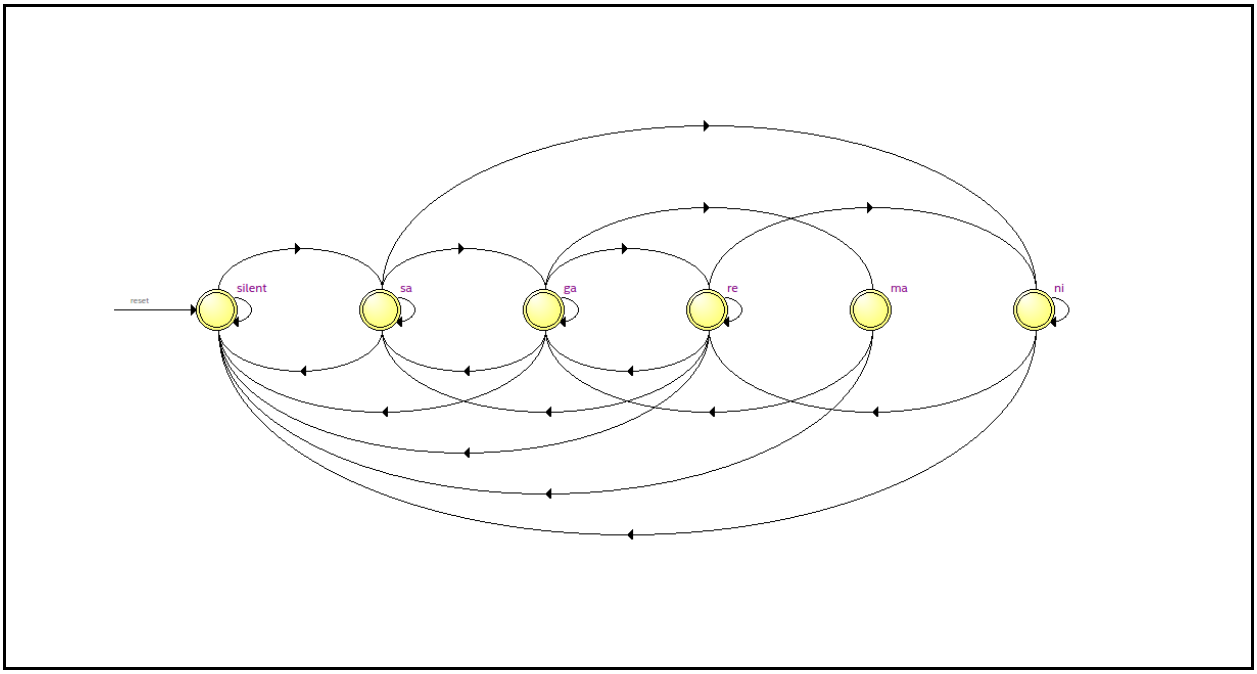
The architecture of Music:

```
architecture fsm of music is

-- 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 in_switch: std_logic_vector(7 downto 0);
signal count: integer := 0;
signal clock_music: std_logic := '1';

-- Take the toneGenerator component
	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;
begin
```

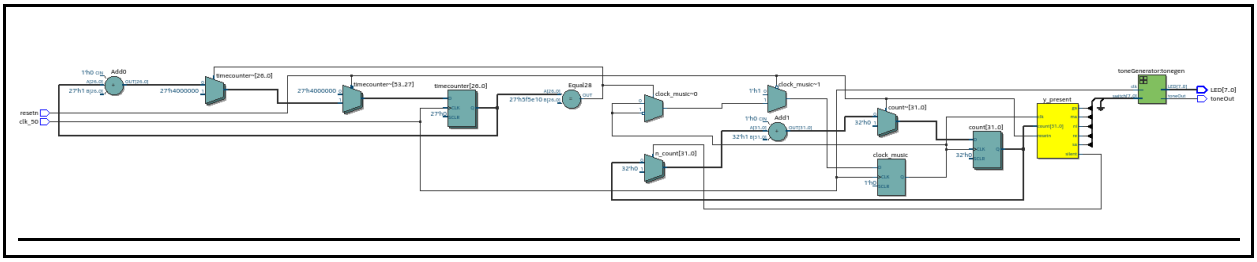
```
process(clk_50,resetn,y_present,clock_music)    -- Fill sensitivity list
variable y_next_var : state_type;
variable n_count, count_music: integer := 0;
variable timecounter : integer range 0 to 1E8 := 0;

begin

        y_next_var := y_present;
        n_count := count;

        case y_present is
                when Silent=>
                        y_next_var := sa;
                        n_count := 0;
                        --assign the signal for switch which will be the input of toneGenerator
                        in_switch <= (others => '0');
                WHEN sa =>     --if the machine in Sa state
                        if((count = 1) or (count = 5) or (count = 9)) then
                                y_next_var := sa;
                        elsif((count = 2) or (count = 6) or (count = 10)) then
                                y_next_var := ga;
                        elsif((count =16) or (count = 31)) then
                                y_next_var := ni;
                        end if;
                        --assign the signal for switch which will be the input of toneGenerator
                        in_switch <= (0 => '1', others => '0');
                WHEN re =>
                        if((count = 19) or (count = 23) or (count = 27)) then
                                y_next_var := re;
                        elsif((count = 15) or (count = 30)) then
                                y_next_var := sa;
                        elsif((count = 20) or (count = 24)) then
                                y_next_var := ni;
                        elsif(count = 28) then
                                y_next_var := ga;
                        end if;
                        in_switch <= (1 => '1', others => '0');
                WHEN ga =>
                        if((count = 3) or (count = 7) or (count = 11)) then
                                y_next_var := ga;
                        elsif((count = 4) or (count = 8)) then
                                y_next_var := sa;
                        elsif(count = 12) then
                                y_next_var := ma;
                        elsif((count = 14) or (count = 29)) then
                                y_next_var := re;
                        end if;
                        in_switch <= (2 => '1', others => '0');
                WHEN ma =>
                        y_next_var := ga;
                        in_switch <= (3 => '1', others => '0');
                WHEN ni =>
                        if((count=17) or (count=21) or (count=25)) then
                                y_next_var := ni;
                        elsif(count=32) then
```

```
                                        y_next_var := silent;
                                else
                                        y_next_var := re;
                                end if;
                                in_switch <= (6 => '1', others => '0');

                        WHEN others =>
                                y_next_var := silent;
                                in_switch <= (others => '0');

                END CASE ;

--              generate 4Hz clock (0.25s time period) from 50MHz clock (clock_music)
                if(clk_50 = '1' and clk_50' event) then
                        if(resetn = '0') then
                                if (timecounter = 6250000) then--4Hz clock
                                        timecounter := 1;
                                        clock_music <= not clock_music;
                                else
                                        timecounter := timecounter + 1;
                                end if;
                        else
                                timecounter := 1;
                                clock_music <= '1';
                        end if;
                end if;
--              state transition
                if (clock_music = '1' and clock_music' event) then
                        if (resetn = '1') then
                                y_present <= silent;
                                count <= 0;
                        else
                                y_present <= y_next_var;
                                count <= n_count + 1;
                        end if;
                end if;
        end process;

-- instantiate the component of toneGenerator
tonegen: toneGenerator
        port map(toneOut, clk_50, LED, in_switch);
end fsm;
```

State diagram view on Quartus:



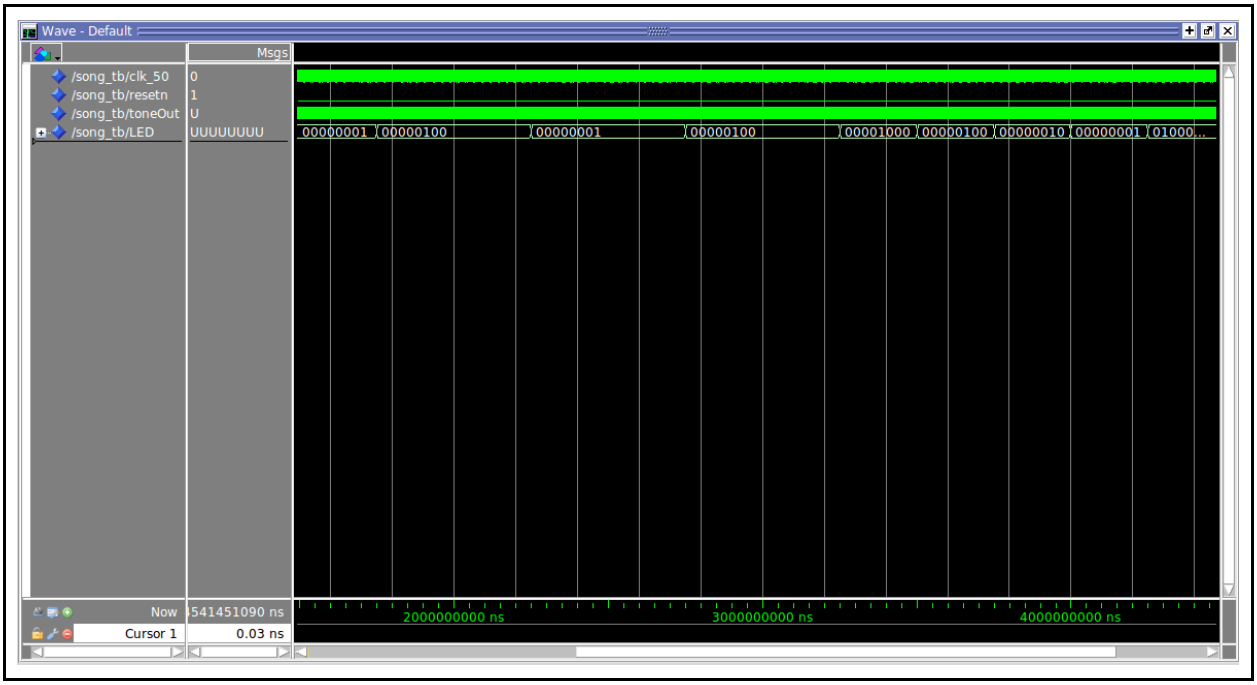
RTL view:



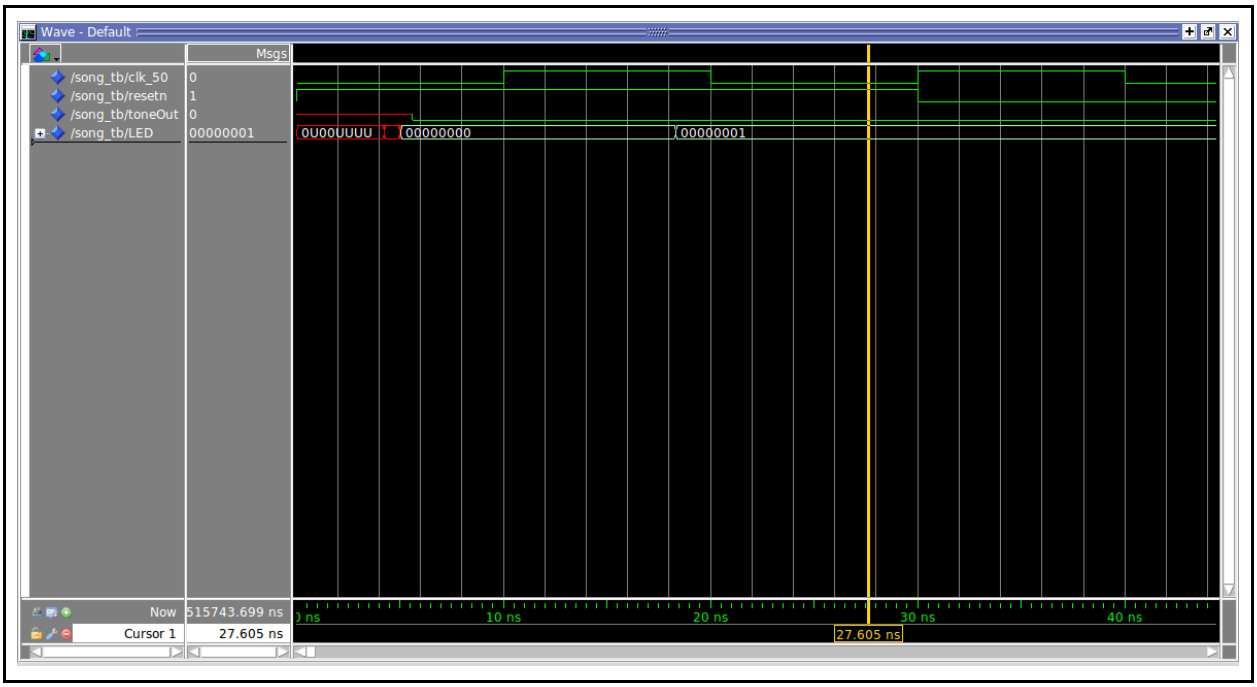
DUT Input/Output Format:

The input is the reset input which is given through the Switch 8 of the Krypton Board and the 50MHz clock which is used in the design.
The output consists of that LED corresponding to that note which is being played according to the sequence given. The final music output is through the PIN 1 of the board.

RTL Simulation:



Gate-level Simulation:

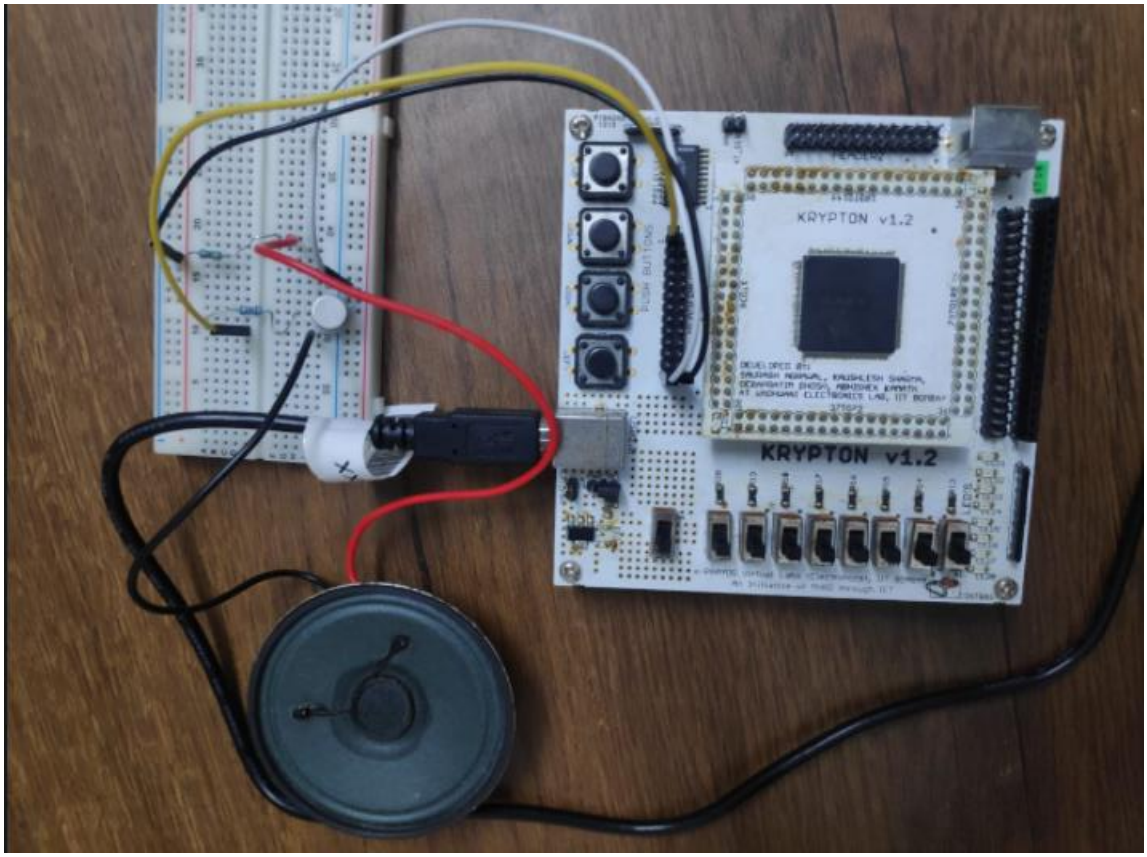


Krypton board:

Pin Planning:

Node Name	Direction	Location	I/O Bank	Fitter Location	I/O Standard	Reserved	Current Strengt	Differential Pair	trict Preservatio
clk_50	Input	PIN_89	3	PIN_89	3.3-V LVTTL		16mA (default)		
LED[7]	Output	PIN_49	4	PIN_49	3.3-V LVTTL		16mA (default)		
LED[6]	Output	PIN_50	4	PIN_50	3.3-V LVTTL		16mA (default)		
LED[5]	Output	PIN_51	4	PIN_51	3.3-V LVTTL		16mA (default)		
LED[4]	Output	PIN_52	4	PIN_52	3.3-V LVTTL		16mA (default)		
LED[3]	Output	PIN_53	4	PIN_53	3.3-V LVTTL		16mA (default)		
LED[2]	Output	PIN_55	4	PIN_55	3.3-V LVTTL		16mA (default)		
LED[1]	Output	PIN_57	4	PIN_57	3.3-V LVTTL		16mA (default)		
LED[0]	Output	PIN_58	4	PIN_58	3.3-V LVTTL		16mA (default)		
resetn	Input	PIN_48	4	PIN_48	3.3-V LVTTL		16mA (default)		
toneOut	Output	PIN_1	1	PIN_1	3.3-V LVTTL		16mA (default)		
<<new node>>									

Connections between Krypton and Speaker components:



Observations:

The sound generated by the speaker and the LEDs glowing correspond to the notes in the sequence.