Victoria Allen (vca2102)

Lab Report 6

Video:

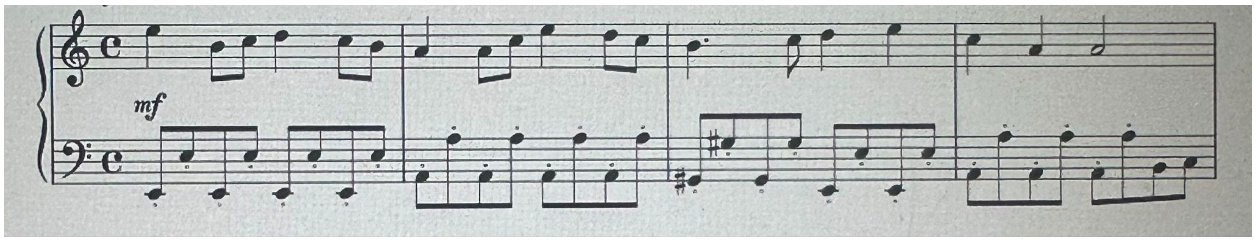
https://drive.google.com/file/d/1cSu8pEL4ozmH9Q9UKPc1RjRmuT5rEstM/view?usp=drive_link

Name of Song:

Tetris Theme (also known as Korobeiniki)

List of Notes:

E, B, C, D, C, B, A, A, C, E, D, C, B, C, D, E, C, A, A
(Shown on treble clef below)



VHDL code:

```
--
-- piano.vhd - FPGA Tetris Theme Player
--
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

library UNISIM;

use UNISIM.VComponents.all;


entity piano is

   port ( CLK_IN        : in std_logic;

          pb_in        : in std_logic_vector(3 downto 0);
```

```
         switch_in    : in std_logic_vector(7 downto 0);
         SPK_N        : out std_logic;
         SPK_P        : out std_logic;
         led_out      : out std_logic_vector(7 downto 0);
         digit_out    : out std_logic_vector(3 downto 0);
         seg_out      : out std_logic_vector(7 downto 0)
        );
end piano;

architecture Behavioral of piano is

  -- Xilinx Native Components
  component BUFG  port ( I : in std_logic; O : out std_logic); end component;
  component IBUFG port ( I : in std_logic; O : out std_logic); end component;
  component IBUF  port ( I : in std_logic; O : out std_logic); end component;
  component OBUF  port ( I : in std_logic; O : out std_logic); end component;
  component MMCME2_BASE
    generic( CLKFBOUT_MULT_F : real;
             DIVCLK_DIVIDE :  integer;
             CLKOUT0_DIVIDE_F  :  real
           );
    port ( CLKIN1     : in    std_logic;
           CLKFBIN    : in    std_logic;
           RST        : in    std_logic;
           PWRDWN     : in    std_logic;
           CLKOUT0    : out   std_logic;
           CLKOUT0B   : out   std_logic;
           CLKOUT1    : out   std_logic;
           CLKOUT1B   : out   std_logic;

           CLKOUT2    : out   std_logic;

           CLKOUT2B   : out   std_logic;

           CLKOUT3    : out   std_logic;

           CLKOUT3B   : out   std_logic;

           CLKOUT4    : out   std_logic;

           CLKOUT5    : out   std_logic;

           CLKOUT6    : out   std_logic;

           CLKFBOUT   : out   std_logic;

           CLKFBOUTB  : out   std_logic;

           LOCKED     : out   std_logic);

  end component;


  -- My Components

  --  Clock Divider
  component clk_dvd

  port (

       CLK     : in std_logic;

       RST     : in std_logic;

       DIV     : in std_logic_vector(15 downto 0);

       EN      : in std_logic;

       CLK_OUT : out std_logic;

       ONE_SHOT: out std_logic

        );

  end component;

  -- Note decoder
  component note_gen

  port (

       CLK       : in  std_logic;
```

```
      RST       : in  std_logic;

      NOTE_IN   : in  std_logic_vector(4 downto 0);

      DIV       : out std_logic_vector(15 downto 0)

     );

end component;

-- 7-Segment Display for Notes
component seven_seg

port (

      CLK      : in std_logic;

      RST      : in std_logic;

      NOTE_IN  : in std_logic_vector(4 downto 0);

      SCAN_EN  : in std_logic;

      DIGIT    : out std_logic_vector(3 downto 0);

      SEG      : out std_logic_vector(7 downto 0)

     );

end component;


-- Signals

signal CLK         : std_logic; -- 50MHz clock after DCM and BUFG

signal CLK0        : std_logic; -- 50MHz clock from pad

signal CLK_BUF     : std_logic; -- 50MHz clock after IBUF

signal GND         : std_logic;

signal RST         : std_logic;

signal PB          : std_logic_vector(3 downto 0); -- Pushbuttons after ibufs

signal digit_l     : std_logic_vector(3 downto 0); -- 7-seg digit MUX before obuf

signal switch      : std_logic_vector(7 downto 0); -- Toggle switches after ibufs

signal led         : std_logic_vector(7 downto 0); -- LEDs after ibufs

signal seg_l       : std_logic_vector(7 downto 0); -- 7-seg segment select before obuf.
```

```
    signal one_mhz      : std_logic;  -- 1MHz Clock

    signal one_mhz_1    : std_logic; -- pulse with f=1 MHz created by divider

    signal clk_10k_1    : std_logic; -- pulse with f=10kHz created by divider

    signal div          : std_logic_vector(15 downto 0); -- variable clock divider for loadable counter

    signal note_in      : std_logic_vector(4 downto 0); -- output of user interface. Current Note

    signal note_next    : std_logic_vector(4 downto 0); -- Buffer holding current Note

    signal div_1        : std_logic; -- 1MHz pulse

    signal sound        : std_logic; -- Output of Loadable Clock Divider. Sent to Speaker if note is playing.

    signal SPK          : std_logic; -- Output for Speaker fed to OBUF


    -- Added Signals
    signal counter      : integer := 0; -- Keeps track of how long the current note or pause has been playing

    signal step    : integer := 0; -- Keeps track of which note in the song we're on


begin


    GND <= '0';

    RST <= PB(0); -- push button one is the reset

    led(1) <= RST; -- This is just to make sure our design is running.


    -- Combinational logic to turn the sound on and off

    process (div, sound)

    begin

        if (div = x"0000") then

            SPK <= GND;

        else

            SPK <= sound;

        end if;

    end process;
```

```
-- Speaker output

SPK_OBUF_INST : OBUF port map (I => SPK, O => SPK_N);

SPK_P <= GND;


-- Input/Output Buffers

loop0 : for i in 0 to 3 generate

    pb_ibuf  : IBUF port map (I => pb_in(i), O => PB(i));

    dig_obuf : OBUF port map (I => digit_l(i), O => digit_out(i));

end generate;



loop1 : for i in 0 to 7 generate

    swt_obuf : IBUF port map (I => switch_in(i), O => switch(i));

    led_obuf : OBUF port map (I => led(i), O => led_out(i));

    seg_obuf : OBUF port map (I => seg_l(i), O => seg_out(i));

end generate;


-- Global Clock Buffers
-- Pad -> DCM
CLKIN_IBUFG_INST : IBUFG port map (I => CLK_IN, O => CLK0);

-- DCM -> CLK
CLK0_BUFG_INST : BUFG port map (I => CLK_BUF, O => CLK);

-- MMCM for Clock deskew and frequency synthesis
MMCM_INST : MMCME2_BASE

  generic map (

    CLKFBOUT_MULT_F => 10.0,

    DIVCLK_DIVIDE => 1,

    CLKOUT0_DIVIDE_F => 10.0

  )

  port map (
        CLKIN1 => CLK0,

        CLKFBIN => CLK,

        RST => RST,

        PWRDWN => GND,

        CLKOUT0 => CLK_BUF,

        CLKOUT0B => open,

        CLKOUT1 => open,

        CLKOUT1B => open,

        CLKOUT2 => open,

        CLKOUT2B => open,

        CLKOUT3 => open,

        CLKOUT3B => open,

        CLKOUT4 => open,

        CLKOUT5 => open,

        CLKOUT6 => open,

        CLKFBOUT => open,

        CLKFBOUTB => open,

        LOCKED => led(0)

      );

   -- Divide 100Mhz to 1Mhz clock
   DIV_1M : clk_dvd port map (CLK => CLK, RST => RST, DIV => x"0032", EN => '1', CLK_OUT => one_mhz,
ONE_SHOT => one_mhz_1);

   -- Divide 1Mhz to Various frequencies for the notes.
   DIV_NOTE : clk_dvd port map (CLK => CLK, RST => RST, DIV => div, EN => one_mhz_1, CLK_OUT => sound,
ONE_SHOT => div_1);

   -- Divide 1Mhz to 10k
   DIV_10k : clk_dvd port map (CLK => CLK, RST => RST, DIV => x"0032", EN => one_mhz_1, CLK_OUT => open,
ONE_SHOT => clk_10k_1);

   -- Translate Encoded Note to clock divider for 1MHz clock.
   note_gen_inst : note_gen port map (CLK => CLK, RST => RST, NOTE_IN => note_in, DIV => div);

   -- Wire up seven-seg controller to display current note.
```

```
    seven_seg_inst : seven_seg port map (CLK => CLK, RST => RST, NOTE_IN => note_in, SCAN_EN =>
clk_10k_1, DIGIT => digit_l, SEG => seg_l);

    note_in <= note_next;

    -- User interface
    process (CLK, RST)
        variable note_length : integer := 42860000; -- BPM of 140 (quarter note)
        variable pause : std_logic := '0'; -- Pause toggle
        variable song_end   : std_logic := '0'; -- Indicates if song has ended
    begin
        if (RST = '1') then   -- Reset logic (clear note, counters, and flags)
            note_next <= (others => '0');
            counter <= 0;
            step <= 0;
            pause := '0';
            song_end := '0';
        elsif rising_edge(CLK) then
            if (counter < note_length) then
                counter <= counter + 1;
            else
                counter <= 0;
                if (pause = '0') then
                    note_next <= (others => '0');
                    if (step = 19) then
                        note_length := 500000000; -- Silence for five seconds (to give a break since it loops)
                        song_end := '1';
                    else
                        note_length := 5000000; -- 0.05 seconds of silence (at 100MHz clock) to break notes up
                        song_end := '0';

                    end if;

                    pause := '1';

                else

                    if (song_end = '1') then

                        step <= 0;

                        song_end := '0';

                    else

                        step <= step + 1;

                    end if;


                    -- Tetris Theme Melody
                    case step is

                        when 0  => note_next <= "10101"; note_length := 42860000; -- E (quarter note)

                        when 1  => note_next <= "01100"; note_length := 21430000; -- B (eighth note)

                        when 2  => note_next <= "10001"; note_length := 21430000; -- C

                        when 3  => note_next <= "10011"; note_length := 42860000; -- D

                        when 4  => note_next <= "10001"; note_length := 21430000; -- C

                        when 5  => note_next <= "01100"; note_length := 21430000; -- B

                        when 6  => note_next <= "01010"; note_length := 42860000; -- A

                        when 7  => note_next <= "01010"; note_length := 21430000; -- A

                        when 8  => note_next <= "10001"; note_length := 21430000; -- C

                        when 9  => note_next <= "10101"; note_length := 42860000; -- E

                        when 10 => note_next <= "10011"; note_length := 21430000; -- D

                        when 11 => note_next <= "10001"; note_length := 21430000; -- C

                        when 12 => note_next <= "01100"; note_length := 64290000; -- B (dotted quarter note)

                        when 13 => note_next <= "10001"; note_length := 21430000; -- C

                        when 14 => note_next <= "10011"; note_length := 42860000; -- D
```

```
                when 15 => note_next <= "10101"; note_length := 42860000; -- E
                when 16 => note_next <= "10001"; note_length := 42860000; -- C
                when 17 => note_next <= "01010"; note_length := 42860000; -- A
                when 18 => note_next <= "01010"; note_length := 85710000; -- A (half note)
                when others => note_next <= (others => '0'); note_length := 42860000;
              end case;

              pause := '0';
            end if;
          end if;
        end if;
    end process;

end Behavioral;
```

Code Walkthrough:

The mechanism that starts the song is it starts automatically (when the reset button/push button 1 is pressed). RST gets set to high (clearing internal counters) and the song is set to start from the first note. No switch flipping is needed. When we ran it, it would play automatically and loop infinitely.

The notes are represented by a binary value of 5 bits (called note_next). The user interface process sets this value based on where you are in the song. This value is sent to note_gen, which figures out how quickly to toggle the speaker to create the correct pitch (these specific frequencies correspond to musical notes). To include different octaves, sharps, or flats, you could do that by adding more note values and updating the note_gen module to handle them (octaves are handled by doubling or halving the frequency, and sharps and flats fall between natural notes, so you'd use divider values that generate intermediate frequencies).

The timing of each note is controlled by a counter that tracks how long the note has been playing. Every clock cycle, the counter goes up, and once it reaches the set note_length, the system either adds a short pause or moves on to the next note. The note_length (which is based on the song’s tempo of 140 beats per minute, so a quarter note lasts about 42.86 million clock cycles on a 100 MHz clock) is customized for each note in the song so it can play different durations like quarter notes, eighth notes, dotted notes, and half notes. There’s also a small pause between notes to make them more distinct, and a longer break at the end before the song loops back to the beginning.

The notes are initially represented as a 5 cells vector. Each different vector corresponds to a different note (ex: 10011 = D). The importance of this encoding was already explained on lab 5. To reiterate: the seven segment display is represented by a 8-cells vector. Each cell of the vector can take in either 1 or 0. Each disposition of 1’s and 0’s creates a different symbol on the seven segment display. As previously explained in lab 5, notice that on the seven_seg file there is a 1-to-1 correspondence between each note encoded in the 5 cells vector and a particular iteration of the 8-cells vector that generates a symbol on the display. The component of the seven segment display is called in the piano.vhd file. Thus, the 344 line of the program:

“seven_seg_inst : seven_seg port map (CLK => CLK, RST => RST, NOTE_IN => note_in, SCAN_EN => clk_10k_1, DIGIT => digit_l, SEG => seg_l);”

takes in the current iteration of note_in (current note) to feed to the seven_seg module which then does the job of creating a correspondence to the display.