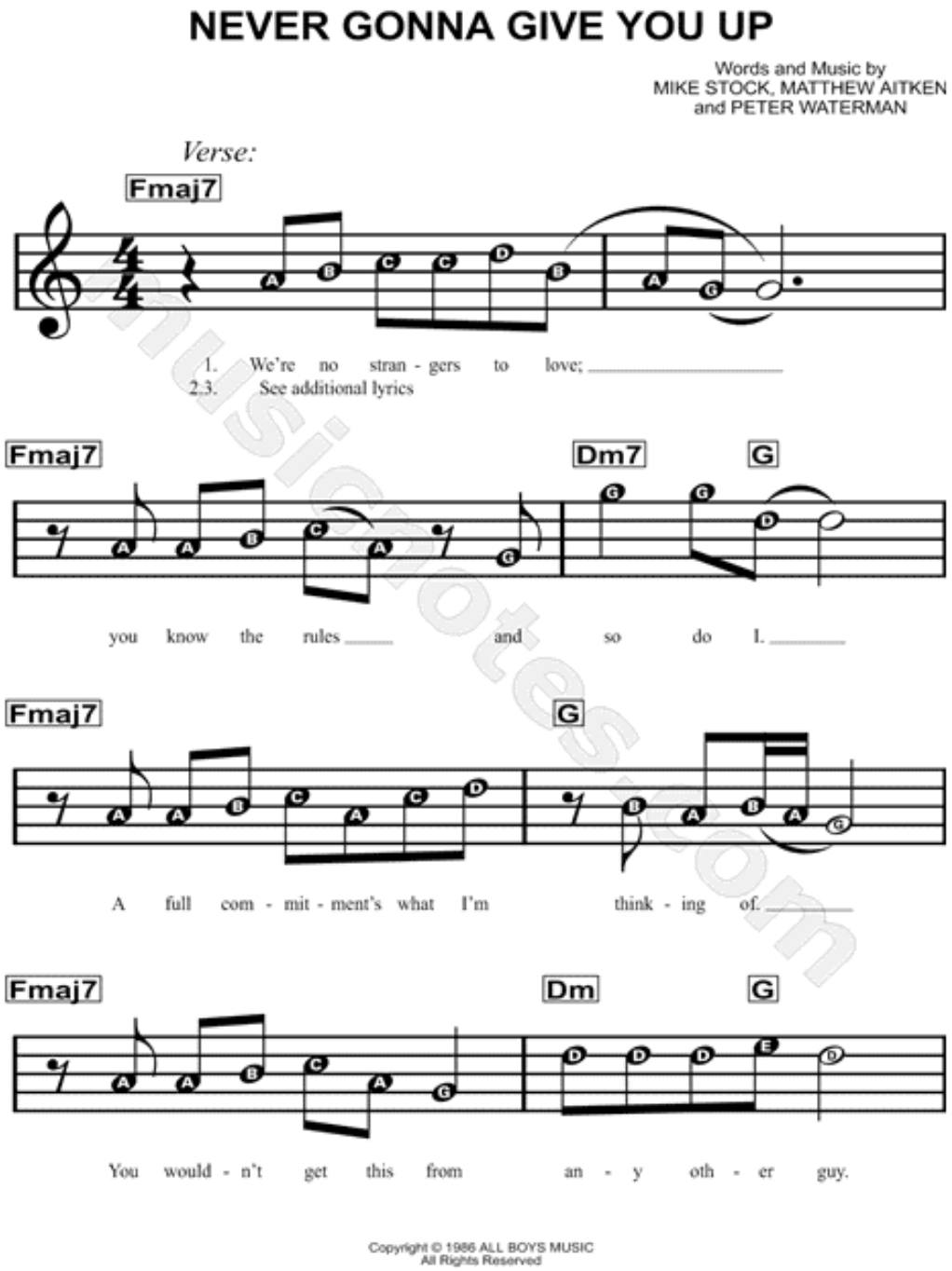
Lab 6: Digital Piano FPGA Part II
EE3082: Digital Systems Lab
Kevin Qiu (kzq2000), Aarya Agarwal (asa2276)

The name of the song was “Never Gonna Give You Up” by Rick Astley.
The attached sheet music with notes used is directly below:



NEVER GONNA GIVE YOU UP

Words and Music by
MIKE STOCK, MATTHEW AITKEN
and PETER WATERMAN

Verse:

Fmaj7

1. We're no stran - gers to love;
2.3. See additional lyrics

Fmaj7 Dm7 G

you know the rules and so do I.

Fmaj7 G

A full com - mit - ment's what I'm think - ing of.

Fmaj7 Dm G

You would - n't get this from an - y oth - er guy.

Code Explanation:

The mechanism that starts the song (such as flipping a switch).

```
elsif (switch = "00000001") then -- start playing the song!
```

We have this line in our code which stipulates that the song will play as long as the first switch is in the on position. Once it is switched to the off position, the song will hold the last note until the PB is hit to reset the whole piano, at which point the piano will wait until the switch is flipped again to start the music.

How the notes are generated/modified(octave, flat, sharp). Feel free to generate whatever notes you desire.

The notes are generated based on the 5 bits fed into the NOTE_NEXT variable. These 5 bits are refreshed according to the timing we have set below. The bits are iterative, based on how the keys are laid out on an actual piano. The higher this number, the higher the note. In the given default code, it used two push buttons to decode sharps and flats. This code is still in our code. However, we did not need to utilize it.

How the timing of notes are done.

What you'll notice is this following section of code:

```
if (CLK'event and CLK = '1') then
                if (count = "00000010111110101111000010000000") then -- play a
note!
```

What is happening here is we are using the divided clock (1Mhz) and stipulating that after a certain number of clock cycles, equivalent to 120 bpm, a note will be played. All of the timing is done relative to eighth notes because our chosen piece of music had a lot of eighth notes. It is also easy to just break a quarter note down into two eighth notes.

How the notes are displayed on the seven-segment display.

In the *piano.vhd* file, we display the note being played on the seven-segment display by hooking it up to the functionality provided by the *seven_seg.vhd* file using the following segment of code:

```
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
                 );
```

Because the notes are mapped in the same way in the seven segment display's code, the seven segment display reads in the current note being played into the NOTE_IN parameter. Then, the note is decoded into individual signals that describe to the seven segment displays which specific LEDs should be lit up. At every rising edge on the clock, the seven segment display checks whether to update (if there is a new value for NOTE_IN) or simply continues in its previous state. See the *seven_seg.vhd* file provided by the lab for additional reference.

The Code For Our Music:

```
Unset
--
-- piano.vhd - FPGA Piano
--
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 note_sel    : std_logic_vector(3 downto 0);   -- Encoding of
switches.
```

```
   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

   signal count       : std_logic_vector(31 downto 0);  -- Signal to count
up to 50000000 clock cycles (approx. 1/8th note at 120 bpm)
   signal note        : std_logic_vector(7 downto 0);   -- Tracks which
note is being played in switch statement below

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",  -- 50
                    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", -- 50
                   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
```

```
                    );

    -- User Interface
    note_in <= note_next;
    process (CLK,RST) begin

        if (RST = '1') then -- reset player if button is pressed
            count <= "00000000000000000000000000000000";
            note <= "00000000";
            note_next <= "00000";
        elsif (switch = "00000001") then -- start playing the song!
            if (CLK'event and CLK = '1') then
                if (count = "00000010111110101111000010000000") then --
play a note!
                    case note is
                        -- rest
                        -- rest
                        when "00000010" => note_next <= "01010"; -- A3
                        when "00000011" => note_next <= "01100"; -- B3
                        when "00000100" => note_next <= "10001"; -- C4
                        when "00000101" => note_next <= "10001"; -- C4
                        when "00000111" => note_next <= "10011"; -- D4
                        when "00001000" => note_next <= "01100"; -- B3
                        when "00001001" => note_next <= "01010"; -- A3
                        when "00001010" => note_next <= "01000"; -- G3
                        when "00001011" => note_next <= "01000"; -- G3
                        when "00001100" => note_next <= "01000"; -- G3
                        when "00001101" => note_next <= "01000"; -- G3
                        when "00001110" => note_next <= "01000"; -- G3
                        when "00001111" => note_next <= "01000"; -- G3
                        when "00010000" => note_next <= "01000"; -- G3
                        -- rest
                        when "00010010" => note_next <= "01010"; -- A3
                        when "00010011" => note_next <= "01010"; -- A3
                        when "00010100" => note_next <= "01100"; -- B3
                        when "00010101" => note_next <= "10001"; -- C4
                        when "00010110" => note_next <= "01010"; -- A3
                        -- rest
                        when "00011000" => note_next <= "01000"; -- G3
                        when "00011001" => note_next <= "11000"; -- G4
                        when "00011010" => note_next <= "11000"; -- G4
                        when "00011011" => note_next <= "10011"; -- D4
```

```
                        when "00011100" => note_next <= "10011"; -- D4
                        when "00011101" => note_next <= "10011"; -- D4
                        when "00011110" => note_next <= "10011"; -- D4
                        when "00011111" => note_next <= "10011"; -- D4
                        -- rest
                        when "00100001" => note_next <= "01010"; -- A3
                        when "00100010" => note_next <= "01010"; -- A3
                        when "00100011" => note_next <= "01100"; -- B3
                        when "00100100" => note_next <= "10001"; -- C4
                        when "00100101" => note_next <= "01010"; -- A3
                        when "00100110" => note_next <= "10001"; -- C4
                        when "00100111" => note_next <= "10011"; -- D4
                        -- rest
                        when "00101001" => note_next <= "01100"; -- B3
                        when "00101010" => note_next <= "01010"; -- A3
                        when "00101011" => note_next <= "01010"; -- A3
                        when "00101100" => note_next <= "01000"; -- G3
                        when "00101101" => note_next <= "01000"; -- G3
                        when "00101110" => note_next <= "01000"; -- G3
                        when "00101111" => note_next <= "01000"; -- G3
                        -- rest
                        when "00110001" => note_next <= "01010"; -- A3
                        when "00110010" => note_next <= "01010"; -- A3
                        when "00110011" => note_next <= "01100"; -- B3
                        when "00110100" => note_next <= "10001"; -- C4
                        when "00110101" => note_next <= "01010"; -- A3
                        when "00110110" => note_next <= "01000"; -- G3
                        when "00110111" => note_next <= "01000"; -- G3
                        when "00111110" => note_next <= "10011"; -- D4
                        when "00111111" => note_next <= "10011"; -- D4
                        when "01000000" => note_next <= "10011"; -- D4
                        when "01000001" => note_next <= "10101"; -- E4
                        when "01000010" => note_next <= "10011"; -- D4
                        when "01000011" => note_next <= "10011"; -- D4
                        when "01000100" => note_next <= "10011"; -- D4
                        when "01000101" => note_next <= "10011"; -- D4
                        when others     => note_next <= "00000"; -- Handle
rests!
                    end case;

                    note <= note + 1; -- go to next note
```

```
                    count <= "00000000000000000000000000000000"; -- reset
note count
                    if (note = "01001101") then -- at end, wait a full
measure before resetting everything automatically!
                        note <= "00000000";
                    end if;
                else
                    count <= count + 1;
                end if;
            end if;
        end if;
    end process;

end Behavioral;
```