Department of Electrical Enginering
Columbia University
EE 3082. Digital Electronics Lab

Lab 5. Electric piano design using Xilinx FPGAs, Part I

1 Prelab

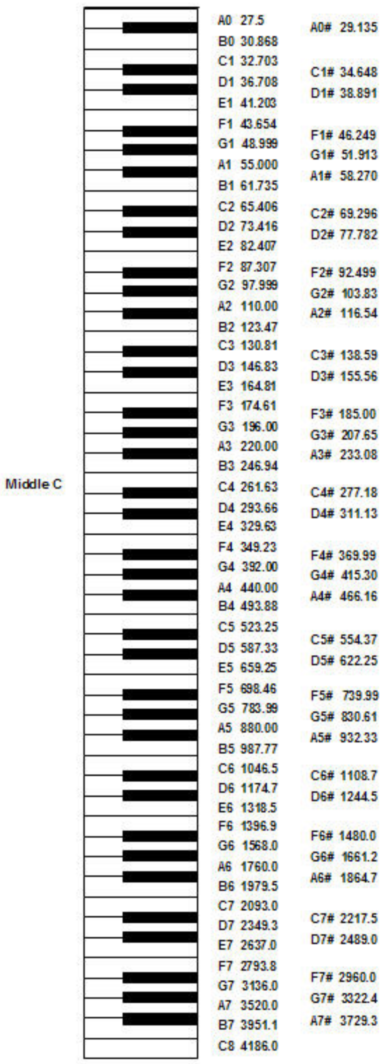
In this design, you will implement an electric piano using field programmable gate arrays (FPGAs). The LSI and MSI gates used to implement the digital logic of the previous labs are largely chips of the past. Digital logic today is implemented in CMOS on VLSI chips – either programmable microprocessors or microcontroller, application-specific integrated circuits (custom VLSI chips with more limited programmability), or FPGAs, which can be customized to perform specific logic functions through a kind of once at start-up programming. FPGAs are often used to "prototype" designs before committing to an expensive ASIC implementation.

Digital logic today is generally designed not as a netlist of gates but as a logic description that is captured in a hardware description language (HDL). The two most popular HDLs are Verilog and VHDL, the latter of which we use extensively at Columbia.

In Labs 5 and 6, you will learn how "real" digital design is done using VHDL, implemented on a fairly state-of-the-art FPGA. The design we have chosen for this exercise is an electric piano. By using a piezoelectric speaker, which has a fairly high impedance, we can drive it directly from the FPGA. This is a similar approach to what is done on the musical greeting cards. The VHDL description is synthesized to logic and mapped to the gates of the FPGA.

If you are interest in some music theory, please check out the following links:

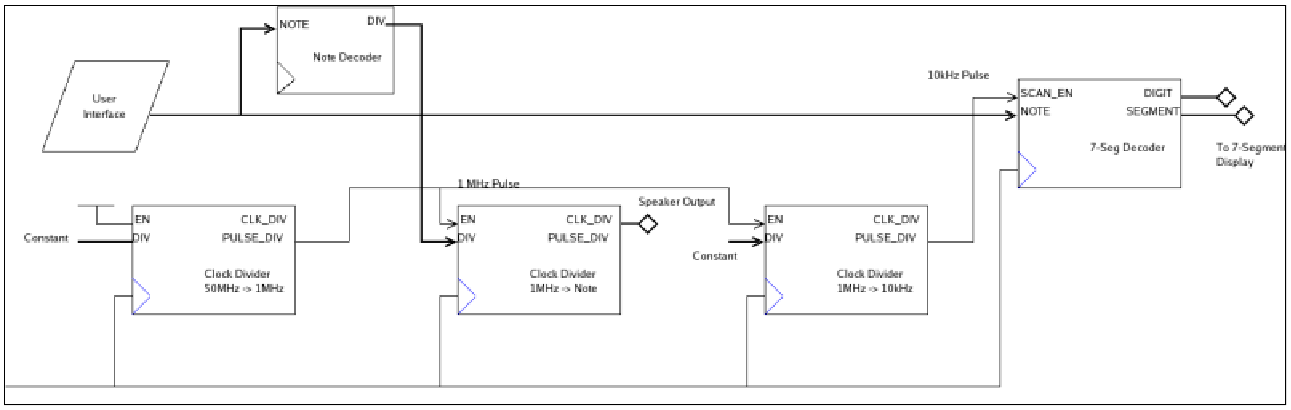
1. Piano Overview
2. Music Theory of a Piano



from vibrationdata.com

2 Design

The schematic of the design is shown below. This is captured in the file piano.vhd, which you will find inside the zipped package uploaded Canvas.



The design consists of the following components:

2.1 User Interface

The user interface (contained in piano.vhd) translates signals from off-chip into a selection to be fed into your note decoder. The current reference design uses the push-buttons and switches on the Xilinx board as ”keys” of the piano.

P How are the notes encoded on the switches?

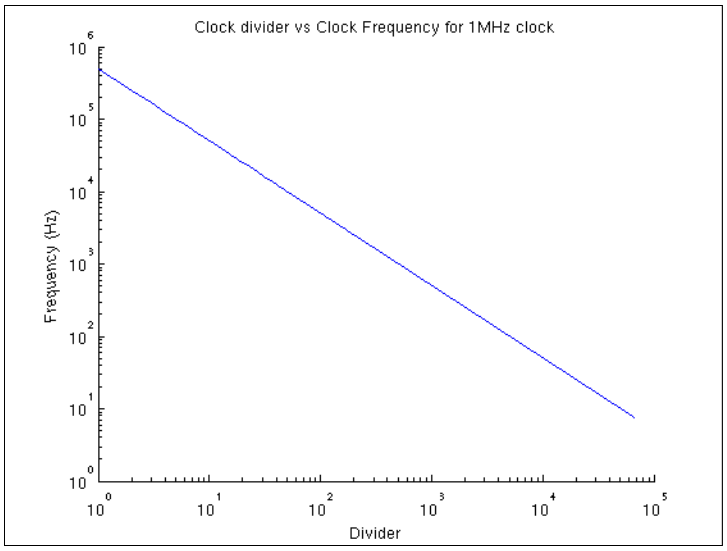
2.2 Clock Divider

The clock divider is implemented as a loadable binary counter. The VHDL source is contained in the file clk_dvd.vhd

P If CLK is a 100 MHz input clock, sketch the waveforms on CLK_OUT and ONE_SHOT for EN = ’1’ and DIV = X”32” (32 in hex, which is equal to 50 in decimal representation). You should also sketch the toggle and trigger waveforms; you can trace these signals one cycle at a time to determine their behavior

2.3 Note Decoder

In this part of the design, we create a decoder that takes as input a binary encoded note, and outputs a binary encoded clock divider to be used with your clock divider ratio. The logic for this part of the design is contained in note_gen.vhd



Here you can see the divider as a function of the output frequency for a 16-bit clock divider with 1MHz clock input. I chose 1MHz as the clock frequency because it gives fidelity in the audio frequency range (20Hz → 20kHz) given the 16-bit granularity.

P For a 1MHz input, calculate the output frequency as a function of divider value. Combining what you calculate with the table at the top of this document, verify the values used for the note decoder to define the notes of the piano.

2.4 7-Segment Display State Machine

In order to save pins the output for each digit of the four-digit 7-segment display is time multiplexed. This means you can only change the value of one of the digits at one time.

By constantly scanning the value of the digits with a low-rate clock you can achieve a persistance of vision similar to a television. Too slow a scan-rate will cause a noticeable flicker, however.

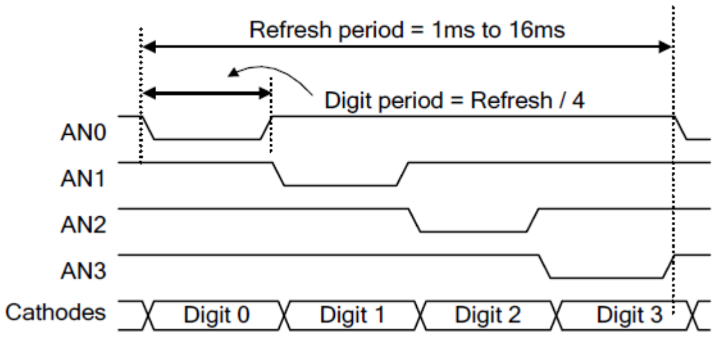
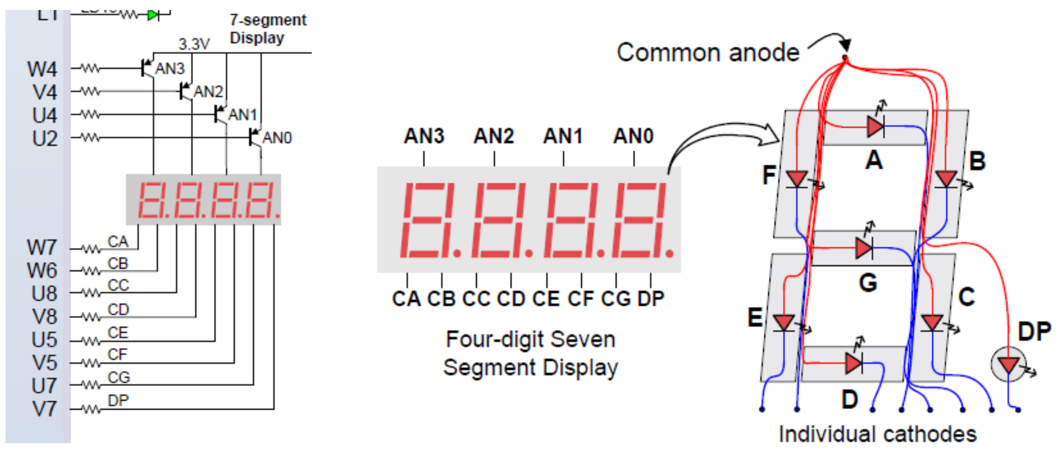


Figure 19. Four digit scanning display controller timing diagram.

This figure from the Basys 3 FPGA Board Reference Manual shows how you will scan each digit in time. Notice that the AN signals are active low.
In this figure you see all the control signals used for the 7-segment display. The design implements a 7-segment decoder that translates a binary-encoded number into the segments needed for that



number.

P Study the implementation in seven_seg.vhd and explain how the design functions.

3 Implementation

We will lean how to synthesize this design to the Xilinx FPGAs to implement the piano.

Reference Manual

There are many documents online for those of you who are interested in the details. To explore beyond what is outlined in the rest of this handout, you may want to look at the files in the links below.

Vivado Documentation
Basys3 Datasheet
Artix-7 Documentation

3.1 Getting started with Vivado

Let's now get started with the Vivado tools.

Create a new project

1. Download piano_src.zip from http://www.bioee.ee.columbia.edu/courses/ee3082/piano_lab.zip and unzip it in the directory you want to work in

2. Open Vivado 2016.4 (not Vivado 2016.4 HLS) from the desktop icon. If there is no desktop icon, execute the following:

```
C:\Xilinx\Vivado\2016.4\bin\vivado.bat
```

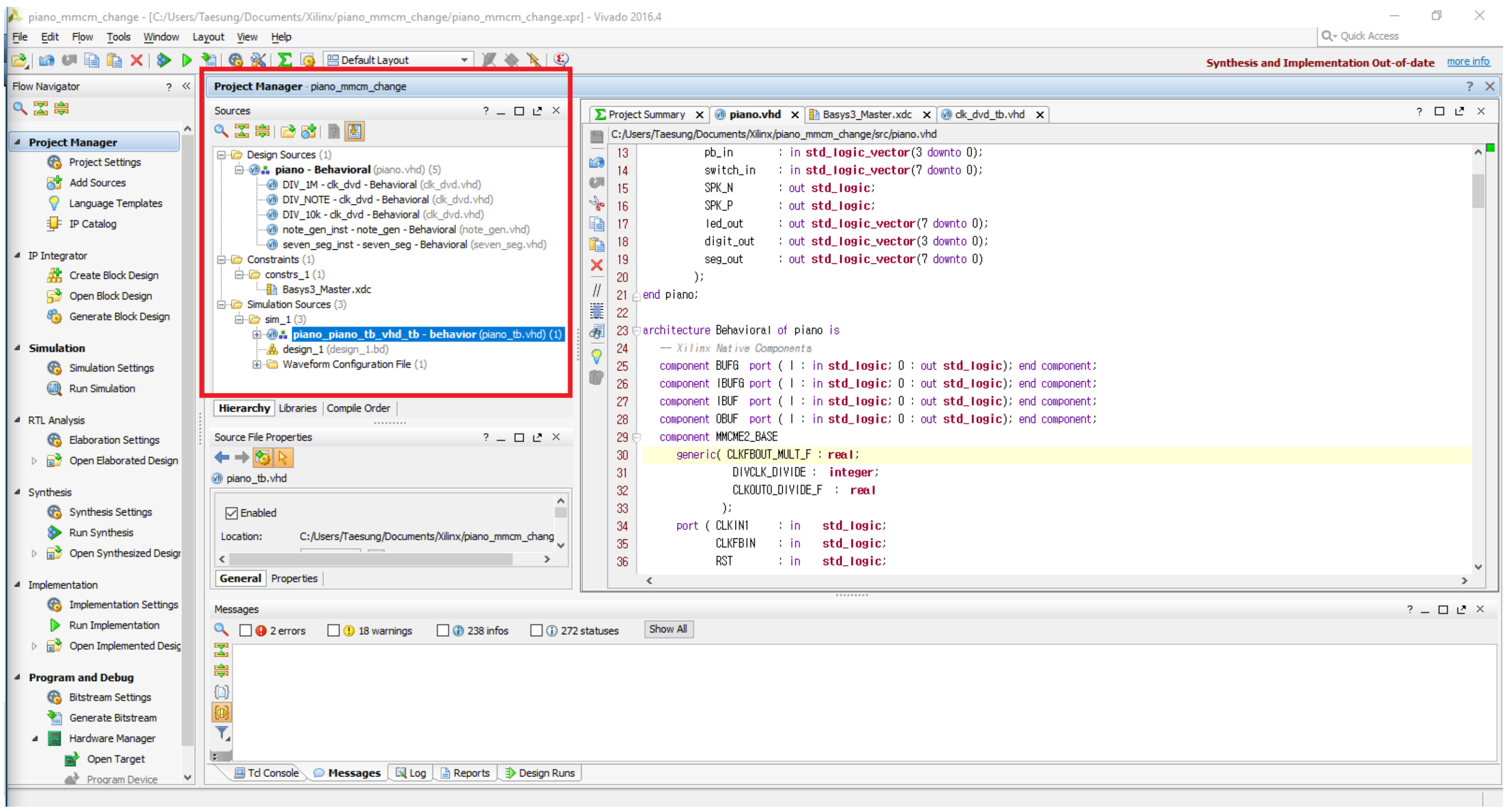
3. Create new project in Quick Start menu. Next

4. Give it a descriptive but short name, no spaces. Next

5. Select RTL project. Next
6. Add all *.vhd files **except for piano_tb.vhd** in the directory from step 0. Select Target language: VHDL. Simulator language: VHDL. Next
7. Skip the IP section. Next
8. Add constraint file: Basys3_Master.xdc. Next
9. Select part xc7a35tcpg236-1. The part number is also laser-etched on top of the ARTIX-7 chip on the board. Next
10. Finish
11. Click Add Sources under Project Manager tab. Select "Add or create simulation sources". Next
12. Add piano_tb.vhd. Finish

Note that the constraint file contains mapping between the ports of your top design module (piano.vhd) and the pins in the FPGA board. For pin names, again refer to Basys 3 FPGA Board Reference Manual.

To navigate through your files, select Project Manager tab in the left panel, and select the file of your interest inside the red-squared region.


piano_mmcm_change - [C:/Users/Taesung/Documents/Xilinx/piano_mmcm_change/piano_mmcm_change.xpr] - Vivado 2016.4

Synthesis and Implementation Out-of-date

Project Manager - piano_mmcm_change

Sources

```
Design Sources (1)
  piano - Behavioral (piano.vhd) (5)
    DIV_1M - clk_dvd - Behavioral (clk_dvd.vhd)
    DIV_NOTE - clk_dvd - Behavioral (clk_dvd.vhd)
    DIV_10k - clk_dvd - Behavioral (clk_dvd.vhd)
    note_gen_inst - note_gen - Behavioral (note_gen.vhd)
    seven_seg_inst - seven_seg - Behavioral (seven_seg.vhd)
Constraints (1)
  constrs_1 (1)
    Basys3_Master.xdc
Simulation Sources (3)
  sim_1 (3)
    piano_piano_tb_vhd_tb - behavior (piano_tb.vhd) (1)
    design_1 (design_1.bd)
    Waveform Configuration File (1)
```

Source File Properties

piano_tb.vhd

Enabled

Location: C:/Users/Taesung/Documents/Xilinx/piano_mmcm_chang

C:/Users/Taesung/Documents/Xilinx/piano_mmcm_change/src/piano.vhd

```
          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;
```

Messages

2 errors | 18 warnings | 238 infos | 272 statuses

3.2 Adding your code

In addition to the clock divider, note decoder, and 7-seg display state machine, the top module makes use of pre-defined Xilinx components such as IBUF, OBUF, MMCM, etc. For details, refer to Artix-7 Clocking Resource.

To briefly explain what they are:
• All offchip inputs and outputs must go to/from buffers. The IBUF and OBUF primitives are used for this purpose.
• All clock signals must be routed through a clock buffer (IBUFG, BUFG).
• The clock is further managed through a Mixed-Mode Clock Manager (MMCM) module.

For this lab, you will only need to modify the test bench file to run simulations. For the next lab, you will need to modify the user interface module to program the FPGA to play a song of your choosing. These require relatively simple fixes to the code, but in case you want to explore further, a good place to look at is Language Templates section of Vivado. To get there, go to **Tools** → **Language Templates** → **VHDL**, where you can find various syntax examples.

3.3 Simulating your design

It is important to debug the design first in a simulator. Place and Route and implementation takes a very long time, and the only way to debug on the board is by setting LEDs and/or using an oscilloscope or logic analyzer. Extensive simulation is really necessary to verify functionality and debug errors.

To simulate your design, you will create a VHDL file called a testbench. This will instantiate the device-under-test (DUT), generate the clocks, and provide other stimulus to your design for testing.
If Vivado could not automatically locate your simulation test bench, you will need to add it manually. To do so, click **Add Sources** under project manager tab → **Add or create simulation sources** → **Add Files** and select the appropriate file → **Finish**

Add test stimulus to your testbench

The following is a basic testbench template.

```
...
-- *** Test Bench - User Defined Section ***

    -- 50MHz system Clock Generation
    clk_gen: PROCESS
    BEGIN
        CLK_IN <= '0';
        wait for 10 ns;
        CLK_IN <= '1';
        wait for 10 ns;
    END PROCESS;
    -- End clock generation addition
    tb : PROCESS
    BEGIN

     -- System Reset - Sets reset to push_button # 0
     pb_in(0) <= '1';
     wait for 100 ns;
     pb_in(0) <= '0';
     -- end system reset addition.

     wait; -- will wait forever

   END PROCESS;
-- *** End Test Bench - User Defined Section ***
```

$\boxed{\textbf{P}}$ Add additional stimulus to test for various values of pb_in and switch_in.. Make sure you save the test bench (Ctrl-S)

Simulate

First, click **Simulation Settings** under Simulation tab. Make sure that target simulator is **Vivado Simulator** and your simulator language is **VHDL**. Click the **Simulation** tab, and set your **runtime** to the desired value.

Finally, click **Run Simulation** under Simulation tab, and click **Run Behavioral Simulation**. When the waveform window appears, adjust the time scale to verify that your design works as intended.

$\boxed{\textbf{P}}$ Save some waveforms from your simulation to demonstrate the functionality of the design