LAB 3 – Digital Piano

1. Lab Objective:

The aims of this laboratory session are:

1.1 To implement a digital piano
1.2 To perform Static Timing Analysis (STA) to look at the delay values
1.3 To perform simulation using a testbench

2. Learning Outcomes:

At the end of this lab session, you would have learned the following:
1.1 Performing STA and understanding reported delay values
1.2 Looking at critical paths in Technology Viewer in Vivado
1.3 Performing simulation using a testbench in Xilinx Vivado Simulator
1.4 Concepts of clock dividing and pulse width modulation

3. Laboratory Exercises:

In the exercises in Lab3, you will implement a design in Verilog in which you can control a speaker's output frequency on the board using the 10 IO ports (named in IO 0-9) on the FPGA device.

3.1 Piano music notes:

Use FPGA to derive the music note of (**Do, Re, Mi, Fa, So, La, Ti**) in three different frequency ranges (low, high, medium) through the **IO switches 0-6**. The speaker should play a corresponding music note when an IO switch is set to high. Only one of the seven switches should be set to high at one time.

When **IO 7** is set to high (i.e., 1) and any IO switch of 0-6 is set to high, a low-frequency range of music notes should be played. If **IO 8** is set to high (i.e., 1) and any IO switch of 0-6 is set to high, the corresponding **medium** frequencies range of music notes should be played. If **IO 9** is set to high (i.e., 1) and any IO switch of 0-6 is set to high, the corresponding **high** frequencies range of music notes should be played.

At most one of IO 7-9 will be set to high. When IO 7-9 are **all set to low**, the speak will be **disabled**.

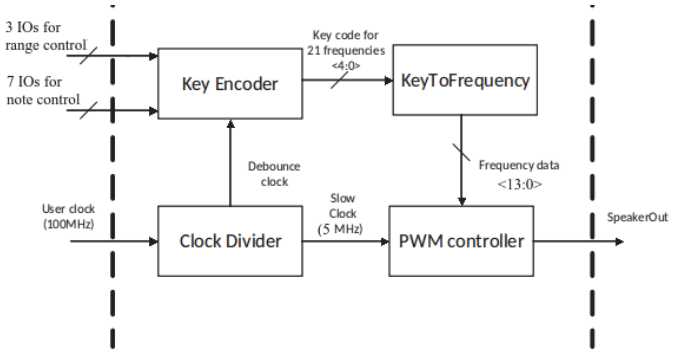


Fig. 2. Piano sub-modules to be implemented.

Fig. 2 shows the design flow and the sub-modules of the digital piano. Here is a simple module specification describing the function of each sub-module and the connections between them.

3.2 Key Encoder:

Take 7 IOs for note control (IO 0-6) and 3 IOs for frequency range control (IO 7-9) as input. Each of the 7 IOs for note control is a piano key note, “Do Ra Me Fa So La Ti” respectively. The output of this module is a key code number from 0 to 21. 1-7 will correspond to 7 music note at low frequency, 8-14 at medium frequency and 15-21 represents high frequency music notes. 3 IOs for frequency range control are used to specify the frequency range, one for low frequency, one for medium frequency and the other one for high frequency. At most one of IO 7-9 will be set to high. When IO 7-9 are all set to low, the speak will be disabled.

3.3 KeyToFrequency:

This will search for the corresponding music note frequency in a hard coded table with the input key code number. The frequency related information will be used to modulate the speaker in PWM controller.

3.4 Clock Divider:

The clock divider will slow down the user clock to 5MHz and feed to the PWM controller.

3.5 **PWM controller**:

This controller will accept frequency related information and generate corresponding pulse-width modulated signal.

The following are the frequencies of 21 music notes

	Low Key(Hz)	Medium Key(Hz)	High Key(Hz)
Do	261	523	1046
Ri	293	587	1174
Mi	329	659	1318
Fa	349	698	1396
So	391	783	1568
La	439	874	1760
Ti	493	987	1974

You should calculate the clock cycle required for a specific key in the 5MHz clock and generate waveform with expected frequency accordingly. Tools like Excel can help you to calculate the values. Check examples in the provided Verilog files. Hint: 5M/noteFrequency.

4. Exercises :

4.1 Steps for Behavioral Verification of Your Design:

4.1.1 Create a Xilinx Vivado project as you did in Lab 1. [**Select device (xc7a35tcpg236-1)**]
4.1.2 When adding all the source and testbench files, select the association of files as shown in Fig. 3.

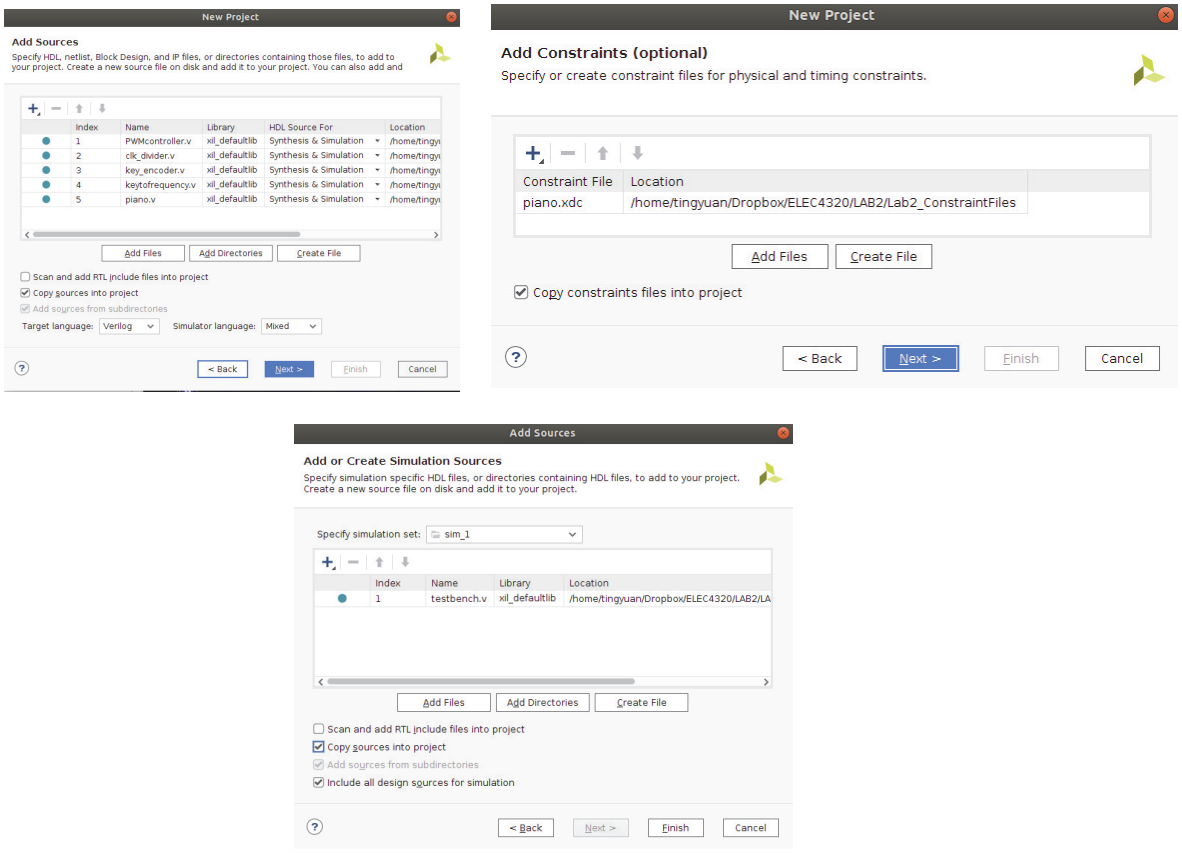
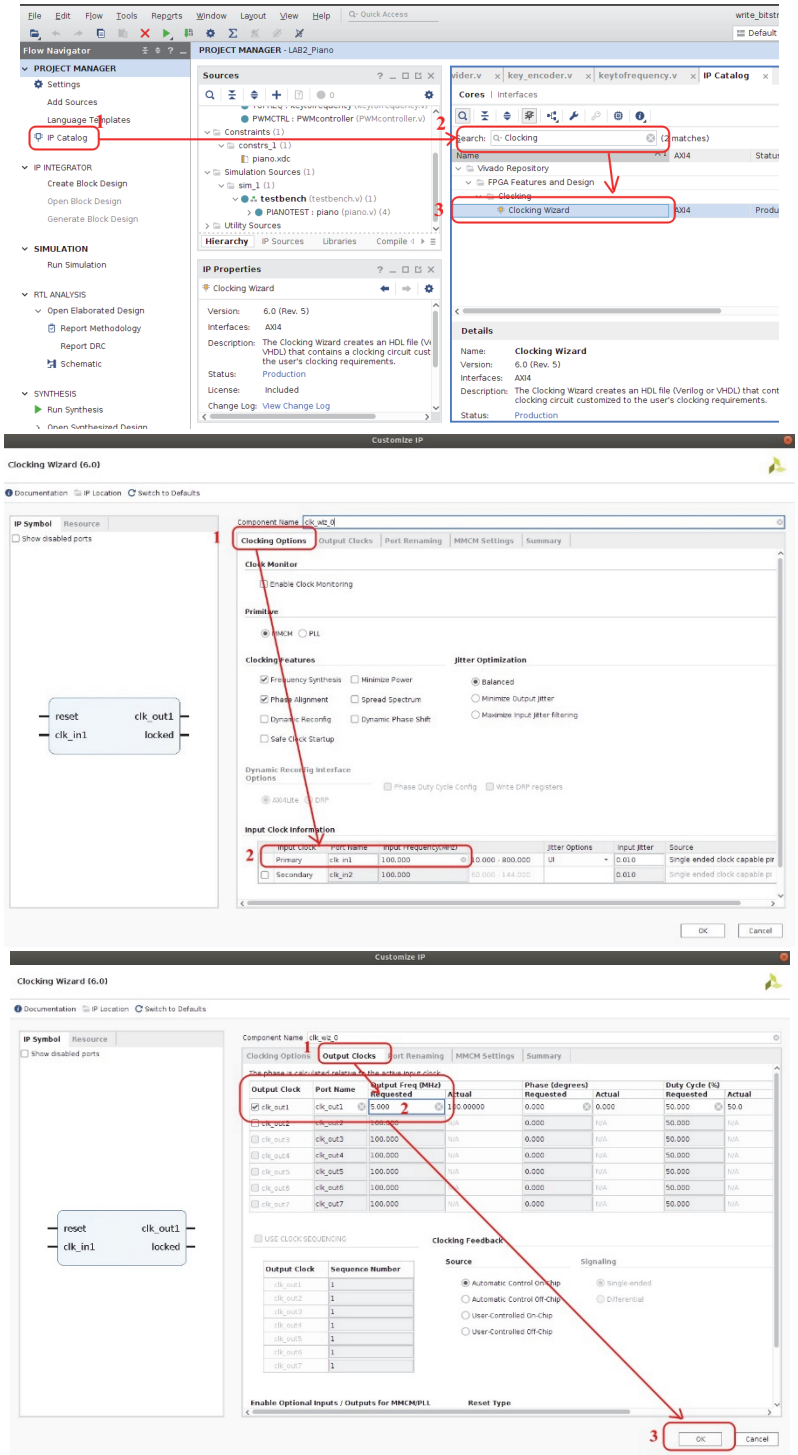


Fig. 3. Add Verilog files, constraint files and testbench files in Vivado

4.1.3 Add a Clocking Wizard IP core (which generates 5MHz clock for PWM from 100MHz input clock). Steps are shown in Fig. 4. After that a defined instance “clk_wiz_0” in clk_divider.v will work properly.



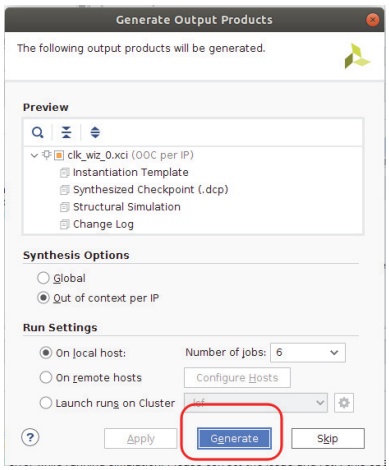


Fig. 4: Add Clocking Wizard IP core to generate 5MHz from 100MHz

4.1.4 Modify the source files (piano.v, key_encoder.v, keytofrequency.v, clk_divider.v, PWMcontroller.v) by the tips in comments.

4.1.5 Repeat the steps for behavioral simulation as you did in Lab 1. (hint: Properly import the testbench to the simulation source)

4.1.6 In the Vivado Simulation window, type **run all** in the terminal or click **run all (F3)** on the Top Toolbar. If no waveform window pops out, click Window->Waveform.

4.1.6.1 You will see the simulation run, print some messages in the console, and then stop after some time.

4.1.6.2 Verify the waveform to see if the **beep** is changing or not as shown in Fig. 5 and the output from Tcl console will display the frequency roughly changing from 261, 293, 523, 587, 1046 to 1174 finally.

4.1.6.3 If the output waveform is not as you expected or Fig. 5, you can debug by checking the output of different signal/register values in modules (add them into the waveform window and re-launch simulation)

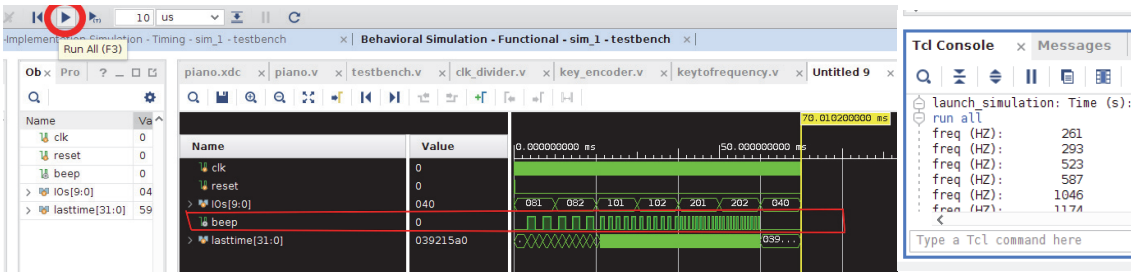


Fig. 5. Observing the change in **beep** signals in Waveform window and the values shown in Tcl Console

4.2 Steps for Checking the Timing Information of Your Design:

4.2.1 Close the simulation and go back to the **implementation** view

4.2.2 **Implement** the design as you did in Lab 1. Click Window->Device to open a view of the FPGA Device.

4.2.3 You will now learn to do Static Timing Analysis (STA) using Timing Analyzer as shown in Fig. 6

4.2.4 Click **OK** on the Timing Report Tips pop-up window.

4.2.4.1 You will see a Timing Report as shown in Fig. 7(b). There are common statistics like Negative Slack and Hold Slack.

4.2.4.2 Click on some links with underline and more details will pop up, such as the critical paths. When you click on a critical path, it will be highlighted in white lines in the device view, as in Fig. 7(a). You can use Ctrl+MouseScroll to zoom in/out in the device view.

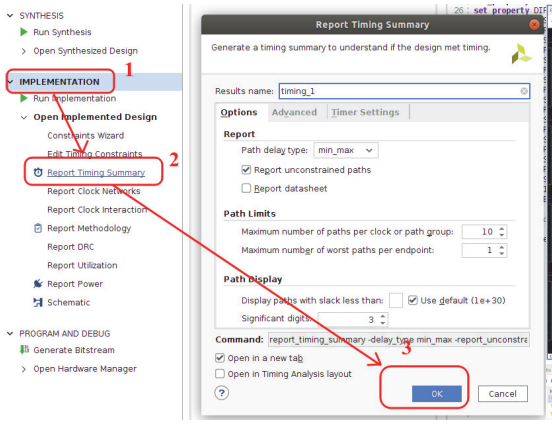


Fig. 6. Performing Static Timing Analysis-1 Fig.

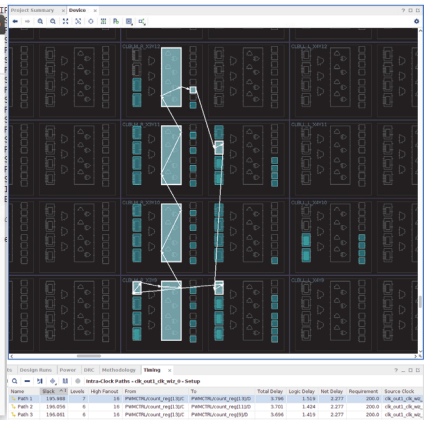


Fig. 7(a). Critical Path in Device View

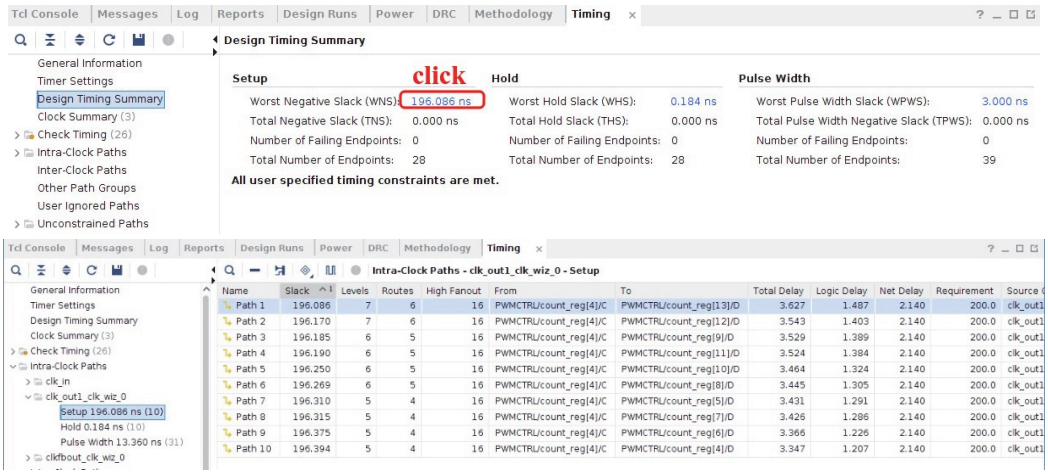


Fig. 7(b). Static Timing Analysis report

4.3 Steps for Verifying Your Design with Post-Implementation Simulation:

4.3.1 After implementation, we can perform post-implementation simulation as shown in Fig. 8. In the Vivado Simulation window, type **run all** in the terminal or click **run all (F3)** on the Top Toolbar.

4.3.1.1 You will see the simulation run, print some messages in the console, and then stop after some time.

4.3.1.2 Verify the simulation as in previous steps for behavioral simulation.

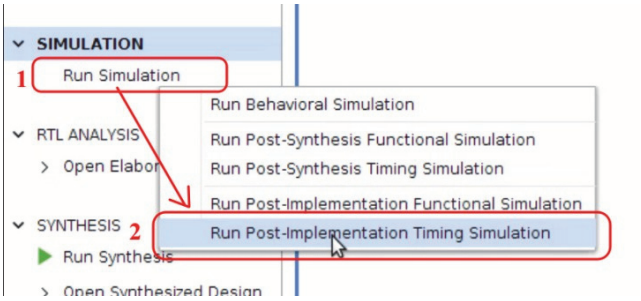


Fig. 8. Post-Implementation Timing Simulation