EECS 151/251A FPGA Lab
Lab 2: Introduction to FPGA Development + Creating a Tone Generator

Prof. Jan Rabaey
TAs: George Alexandrov, Ali Moin
Department of Electrical Engineering and Computer Sciences
College of Engineering, University of California, Berkeley

1 Before You Start This Lab

This lab and all future labs will be done in groups of 2.

Make sure that you have gone through and completed the steps involved in Lab 1. Let the TA know if you are not signed up for this class on bCourses and Piazza or if you do not have a class account (eecs151-xxx), so we can get that sorted out. Go through the Verilog Primer Slides that are linked on bCourses; you should feel somewhat comfortable with the basics of Verilog to complete this lab.

To fetch the skeleton files for this lab `cd` to the git repository (`fa18_fpga_labs`) that you had cloned in Lab 1 and execute the command `git pull`.

You can find the documents/datasheets useful for this lab in the `fa18_fpga_labs/docs` folder.

2 A Structural and Behavioral Adder Design + Using fpga_editor and the FPGA schematic

2.1 Build a Structural 14-bit Adder

To help you with this task, please refer to the 'Code Generation with for-generate loops' slide in the Verilog Primer Slides (slide 35).

Navigate to the `lab2/src` directory. Begin by opening `full_adder.v` and fill in the logic to produce the full adder outputs from the inputs. Then open `structural_adder.v` and construct a ripple carry adder using the full adder cells you designed earlier.

Finally, inspect the `ml505top.v` top-level module and see how your structural adder is instantiated and hooked up to the top-level signals. For now, just look at the `user_adder` instance of your structural adder.

Run `make` in the `lab2` directory and let the build process complete. Then run `make impact` to send your design to the FPGA. Test out your design and see that you get the correct results from your adder. You should try entering different binary numbers into your adder with the DIP switches and see that the correct sum is displayed on the GPIO LEDs.

It is highly recommended that you run 'make report', and check under 'Synthesis Messages' to see if there are any warnings that indicate problems with your design. You can expect to see a warning that 'Connection to output port 'sum' does not match port size' and 2 warnings that 'The value init of the FF/Latch error hinder the constant cleaning...'; these warnings can be ignored, but any others could reveal an issue in your design.

2.2 Inspection of Structural Adder Using Schematic and fpga_editor

2.2.1 Schematic

Now let's take a look at how the Verilog you wrote mapped to the primitive components on the FPGA. To do this, we will first look at a high-level schematic of your circuit using Xilinx ISE.

Execute the command `make schematic` in the `lab2` directory. This will launch the ISE project navigator. Once inside ISE, go to `File -> Open` and select `ml505top.ngr`. You will recall that the NGR file is an output of XST (synthesis). Choose `Start with a schematic of the top-level block` in the dialog that pops up.

This will given you a fairly straightforward hierarchical block-level view of your design. You will find your circuit by drilling down in the following modules: `ml505top, user_adder, full_adder`. Check to see that your structural adder module is hooked up properly and looks sane. It's ok if the wires don't appear to be connected, just hover your mouse over the endpoints on the schematic and ensure that the connections are as you expect. Take note of the primitive blocks used in your circuit.

2.2.2 FPGA Editor

Fall 18 Note: Opening fpga_editor might not work this semester; skip this part if that's the case

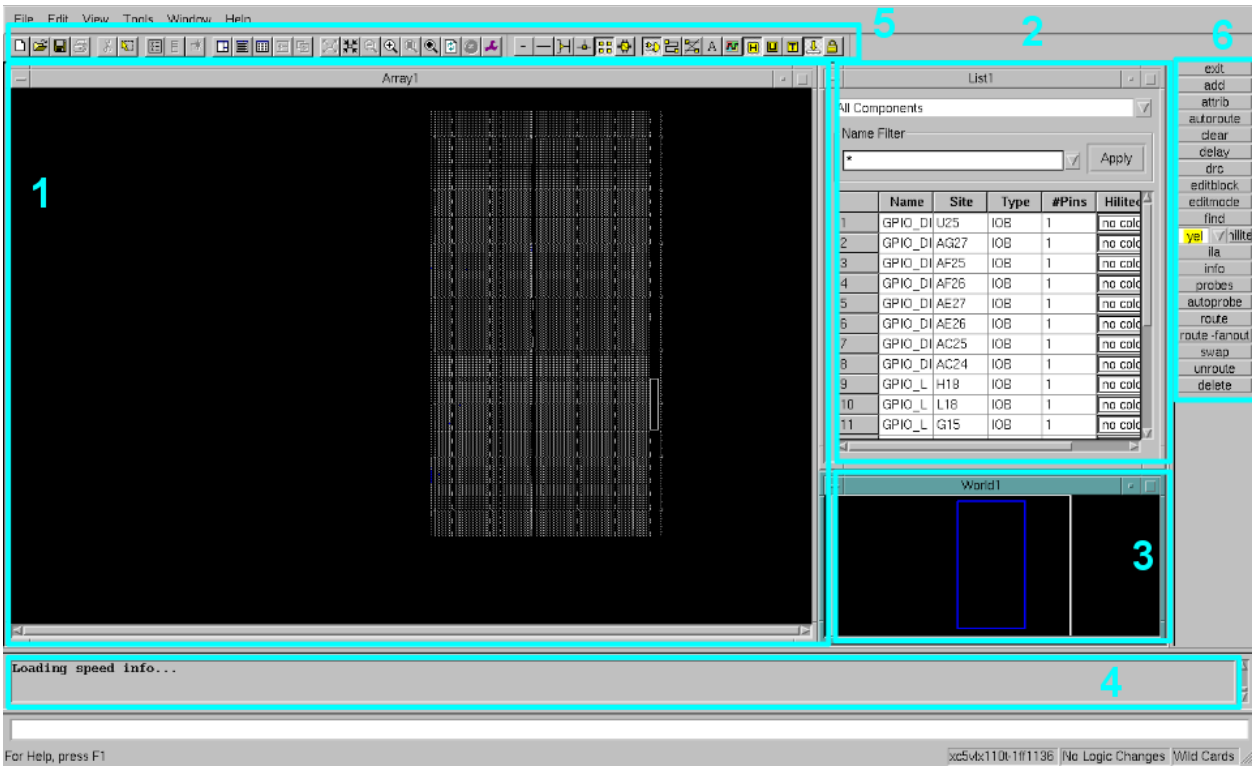
Now let's take a look at how your circuit was placed and laid out on the FPGA with the `fpga_editor`. Recall that the `.ncd` file is the final result of the FPGA toolchain flow. There also exists a `.pcf` file (Physical Constraints File) that contains the information originally present in the UCF. By opening these files in the FPGA editor, you can visualize how your design will actually be mapped to the FPGA. Run these commands:

```
cd lab2/build/ml505top
DISPLAY=:0 fpga_editor ml505top.ncd ml505top.pcf
```

If you get an error that states something like 'Can't open display' when running `fpga_editor`, try running the command again without the `DISPLAY=:0` prefix like this:

```
fpga_editor ml505top.ncd ml505top.pcf
```

The `fpga_editor` might take up to a minute to open.

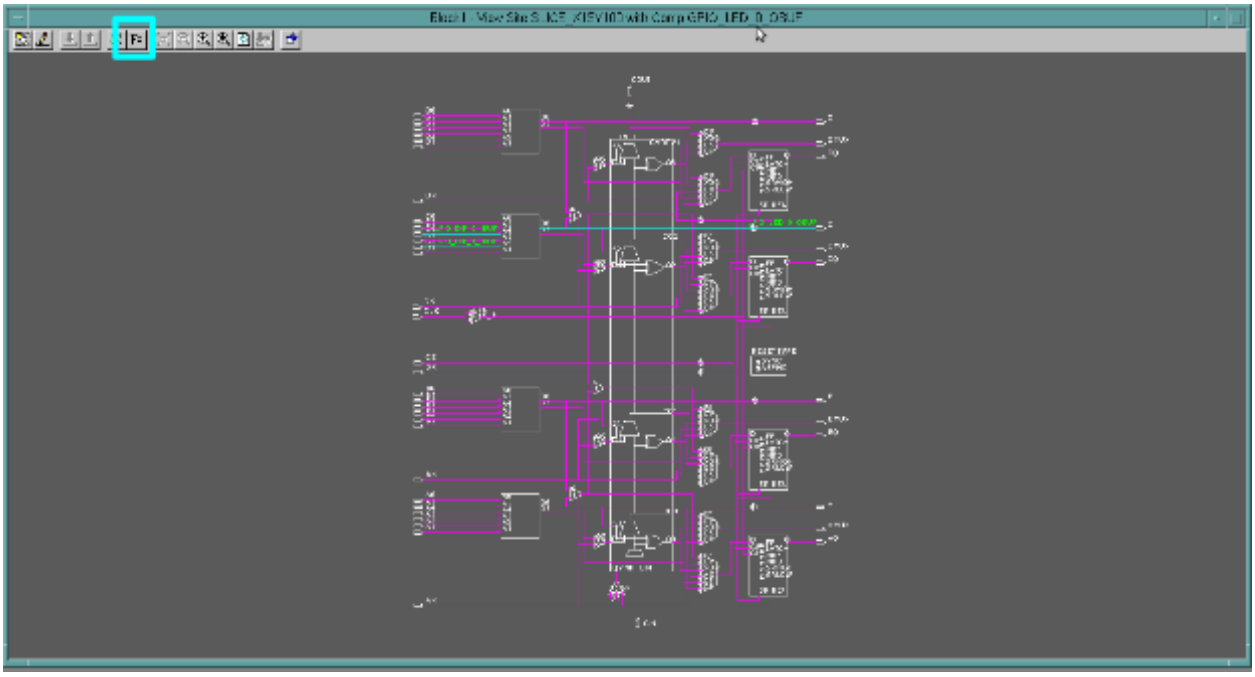


The image above shows the main view for the FPGA editor. It is split up into the following windows:

1. **Array Window** - shows a schematic of the FPGA and highlights the parts of the FPGA that are currently utilized, as well as the connections between these components.
2. **List Window** - lists the components and nets that are used in your design. Double-clicking items here will zoom in on them in the Array Window.
3. **World Window** - this shows you where you are currently focused in the Array Window.
4. **Console Output** - prints messages that often contain useful diagnostic information.
5. **Toolbar** - contains useful buttons for manipulating the windows; mouse over the buttons to reveal what is does.
6. **Button Bar** - contains other useful buttons for modifying the design.

Now you can explore your design and look for the modules that you wrote. If you scroll down in the `List Window` you will find that the type of some items is SLICEL.

Recall that SLICELs contain the look-up tables that actually implement the logic you want. Double clicking on the SLICEL named `structural_out<x>` (where x is some number) will take you to the corresponding SLICEL in your Array Window. Double click on the red rectangle (the SLICEL) to bring up the following Block Window:



Click on the `F=` button to reveal the function of the LUT. Go ahead and explore several SLICELs that implement the structural adder to see how they are connected to each other and the outputs of your circuit.

2.3 Build a Behavioral 14-bit Adder

Check out `behavioral_adder.v`. It has already been filled with the appropriate logic for you. Notice how behavioral Verilog allows you to describe the function of a circuit rather than the topology or implementation.

In `ml505top.v`, you can see that the `structural_adder` and the `behavioral_adder` are both instantiated in the self-test section. A module called `adder_tester` has been written for you that will check that the sums generated by both your adders are equal for all possible input combinations. If both your adders are operating identically, all the compass LEDs will light up. Verify this on your board. The self-test takes around 10 seconds to run.

2.4 Inspection of Behavioral Adder Using Schematic and fpga_editor

Go through the same steps as you did for inspecting the structural adder. First run `make schematic` and then run the FPGA editor. In the FPGA editor, inspect the `behavioral_out<x>` SLICE and the `behavioral_test_adder/Madd...` SLICE. Record and note down any differences you see between both types of adders in the schematic and the FPGA editor. You will be asked for some observations during checkoff.

3 Designing a Tone Generator

Now it's time to try something new. Let's create a tone generator/buzzer on the FPGA.

Please take a look at the `ml505_user_guide.pdf` in the `fa18_fpga_labs/docs` folder. On page 20, you can see a list of oscillators that are present on the development board. These clock signals are generated outside the FPGA by a crystal oscillator or a programmable clock generator IC. These clock signals are then connected to pins on the FPGA so that they can be used in your Verilog design.

Take a look at the `ml505top.v` module and notice the `CLK_33MHZ_FPGA` input. Next take a look at the UCF (scroll to the bottom) `ml505top.ucf` and notice how the LOC for the clock net is set to AH17, just as specified in the ML505 User Guide. There is a 33Mhz clock signal that is generated on a clock generation IC on the ML505 board and is then routed to the FPGA's AH17 pin. We can access the signal from within our Verilog top-level module and can propagate this clock signal to any submodules that may need it. This is a 33 Mhz clock signal, so there are a couple other lines in the UCF that specify the period of the clock so that timing checks can be performed when PAR is being executed.

3.1 Piezoelectronic Buzzers

Take a look at page 14 of the ML505 User Guide to see a block diagram of all the signals that come into our FPGA and can be monitored/driven from our Verilog design. The fourth from the top box on the left is labeled Piezo/Speaker. To get more information about this component go to page 37.

Notice how the table specifies that the piezo signal that is routed to the actual component on the board can be driven by FPGA pin G30. Keep this in mind.

Take a look at this wiki page to see how a piezoelectric speaker functions.

3.2 Finding the Piezo in the Schematic

Let's look into how the piezo is connected to the FPGA and the ML505 board. First take a look at the datasheet for the piezo on this board. You can find the datasheet in `fa18_fpga_labs/docs` with filename `AT-1220-TT-2-R.pdf`. The datasheet has a picture of how the piezo speaker looks; now find it on the ML505 board!

Once you have located the piezo speaker, let's see the schematic of the board to see how it is driven by the FPGA. Open the schematic file in the `fa18_fpga_labs/docs` folder called `ml50x_schematics.pdf`. Go to page 3 of the schematics and take a look at the Bank 15 block. The third pin from the top on the right side is labeled `PIEZO_SPEAKER`. Notice also how `G30` is listed next to the net indicating that is connected to the G30 pin on the FPGA. Bank 15 is an I/O block on the FPGA; general purpose I/O connections on the FPGA are organized in many banks, each with their own voltage supply.

Clicking on the `PIEZO_SPEAKER` label will take you to page 18 of the schematic. Here you can see the piezo net going into an IC (SN74LVC1G126). This IC just acts as a buffer for the signal coming from the FPGA to drive the piezo speaker. From this IC you can see the signal go through a 10μF coupling capacitor before going into the piezo (SP1 in the schematic).

3.3 Adding the Piezo signal to the UCF File

Now let's add the piezo connection to the UCF so that we can bring it in as an output from the Verilog top-level module. To do this, we will use the master UCF file. Find this in the `fa18_fpga_labs/docs` folder with the filename `master_xupv5-lx110t.ucf`. Open the master UCF and search for 'piezo'. You will find on line 368 the net declaration for the `PIEZO_SPEAKER` signal. Copy this line into your `ml505top.ucf` file. Specify the IOSTANDARD for this net in the UCF as `LVCMOS18` since the bank (Bank 15) it is connected to has a Vcco of 1.8V.

Ask a TA if you need help for this part.

3.4 Generating a Square Wave

Let's say we want to play a 220 Hz square wave into the piezo speaker. We want our square wave to have a 50% duty cycle, so for half of the period of one wave the signal should be high and for the other half, the signal should be low. We have a 33 Mhz clock input we can use to time our circuit.

Find the following:

1. The period of our clock signal (frequency = 33 Mhz)?
2. The period of a 220 Hz square wave?
3. How many clock cycles does it take for one period of the square wave?

Knowing how many clock cycles equals one cycle of the square wave, you can design this circuit. First open `tone_generator.v`. Some starter code is included in this file. Begin by sizing your `clock_counter` register to the number of bits it would take to store the clock cycles per square wave period. Design the logic such that a 220 Hz square wave comes out of the `square_wave_out` output.

Add an output to `ml505top.v` with a matching net name to the piezo net you declared in the UCF. Instantiate the `tone_generator` and connect it to the piezo output.

Now `make` and `make report`. Check for any warnings or errors and try to fix them. Ask a TA if you need help here. When everything looks good run `make impact`. You should now hear a buzzing noise at 220Hz. To stop the buzzing, you can erase the design on the FPGA by pressing the leftmost button right above the PCI-e finger (to the right of the DVI connector).

3.5 Switching the Wave On and Off

Now you have a tone, but it can't be toggled on and off without pulling the power to the FPGA board. Let's use the `output_enable` input of the `tone_generator` module to gate the square wave output. When `output_enable` is 0, you should pass 0 to the `square_wave_out` output, but when `output_enable` is 1, you should pass your square wave to `square_wave_out`.

Wire up the `output_enable` signal to the first DIP switch (`GPIO_DIP[0]`) in `ml505top`.

Now `make` and `make report`. Check for any warnings or errors and try to fix them. Ask a TA if you need help here. When everything looks good run `make impact`. You should now hear a buzzing noise at 220Hz that can be turned on or off by toggling the first DIP switch.

You should verify that the tone is indeed 220 Hz by comparing it to a reference tone here: `http://onlinetonegenerator.com/`.

4 Optional: Use Remaining Switches to Control Frequency

In the next lab, we will be making our `tone_generator` more full-featured. If you have time now, we recommend you implement some of these features since the next lab will be longer. If you choose not to implement this now, skip this section and proceed to the checkoff, but keep in mind that you will have to do it next week as part of Lab 3!

Let's extend our `tone_generator` so that it can play different notes. First, you should add a 24-bit input to the `tone_generator` called `tone_switch_period`. Note you will also have to modify your `clock_counter` to be 24 bits wide.

The `tone_switch_period` describes how often the square wave output switches from high to low or low to high. For example a `tone_switch_period` of 75000 tells us to invert the square wave output every 75000 clock cycles, which for a 33 Mhz clock translates to a 220 Hz square wave. Here is the derivation:

$$\frac{33 \times 10^6 \text{ cycles}}{1 \text{ second}} \div \frac{220 \text{ periods}}{1 \text{ second}} = \frac{150000 \text{ cycles}}{1 \text{ period}}$$
$$150000 \text{ cycles/period} \rightarrow 75000 \text{ cycles/half-period}$$

You may have to modify the architecture of your `tone_generator` to accommodate this new input signal. You should reset the internal `clock_counter` every `tone_switch_period` cycles and should also invert the square wave output. Remember to initialize any new registers declared in your `tone_generator` to their desired initial value to prevent unknowns during simulation.

4.1 Try the variable frequency tone_generator on the FPGA

Modify the top-level Verilog module `ml505top.v` to include the new input to the `tone_generator`. You should tie the `tone_switch_period` to the `GPIO_DIP[7:1]` switches left-shifted by 9 bits

(effectively a multiplication by 512). This will allow you to control the `tone_switch_period` from 512 to around 65000. Leave `GPIO_DIP[0]` to control `output_enable`. Here is a code snippet:

```
tone_generator piezo_controller (
    .output_enable(GPIO_DIP[0]),
    .tone_switch_period({17'd0, GPIO_DIP[7:1]} << 9)
);
```

Run the usual `make` process and then `make impact` to put your new `tone_generator` on the FPGA. Verify that toggling the DIP switches changes the frequency of your `tone_generator`.

5 Checkoff

To checkoff for this lab, have these things ready to show the TA:

1. Answers for the questions in part 3.1
2. Be able to explain the differences between the behavioral and structural adder as they are synthesized in both the high-level schematic and low-level SLICE views
3. Show the RTL you used to create your tone generator, and your calculations for obtaining the square wave at 220Hz
4. Demonstrate your tone generator on the FPGA and show that the 0th DIP switch controls the buzzing of the piezo speaker

You are done with this lab. In the next lab, we will simulate our digital designs in software, and extend our `tone_generator` to read a song from a ROM and play it out over the piezo speaker.