525.742 SOC FPGA Design Lab

Laboratory Project 3-b

SDR Signal Source (And Verification)

Late Assignments: Up to 1 week late for 10 point penalty

**Introduction**

In this lab we will continue the development of a software defined radio (SDR) on our evaluation board by creating the most basic signal processing chain – a programmable input signal which will drive the codec from Lab 1/2. The input signal will be a numerically controlled oscillator (NCO), also called a direct digital synthesizer (DDS).

A point of emphasis of this lab is the test/validation of a design, so we'll compare the following three things:

1. A MATLAB model of a DDS (you completed this in Lab3a).
2. A VHDL simulation of a DDS Core (yyou completed this in Lab3a).
3. A recording (via ILA) of data from a DDS running on the FPGA (you’ll build this, and save the data to add one more signal to look at in your model).

***Part I: IP Construction***

***Part II: Gate Level Simulation***

***Part III: Hardware Construction***

Now that you have completed Lab 3a, and fully understand how your DDS should be configured, include the DDS that you made in Lab3a into a full project. Consult the lecture slides for architecture suggestions, but Lab2 is a very good starting point since many things will be the same. The output of the DDS should go directly to the low-level-dac-interface. In other words, your FIFO from lab 2 is not necessarily required, since sound will no longer be coming from the microprocessor, but instead directly from the DDS. Do this in such a way that we will hear the identical sound from the DDS on both left and right channels (in both ears) **Important – as discussed in class the output of the DDS will be changing every clock, not just when latched\_data tells it to. So if your low-level-dac-interface doesn’t grab the data itself and store it, you will want to insert a register between the DDS and DAC\_IF. Alternatively, just use the instructor solution to lab 1, which grabs the data on it’s input into an internal register when it needs a new word.**

Make sure that the output signal of the DDS is observable on an ILA in your design .This is how we will verify that the real thing matches our Matlab model

Using the method of your choice, setup the hardware in such a way that the microprocessor will be able to control the phase-increment (aka frequency) of the DDS. An AXI\_GPIO is a good choice here. Make sure you understand how the frequency is set – your simulation is a good tool here.

Using the method of your choice, setup the hardware in such a way that the microprocessor will be able to reset the phase of the DDS to 0 on command (**without changing the frequency)**. The instructor solution does this by using a GPIO on the microprocessor to pulse the reset line of the DDS. You will hear this as a little “pop” as the DDS briefly stops making samples and then starts again at an angle of 0.

Have the audio from the DDS show up on **both** the left and right channels of the headphone output

Note on clocking : As in lab 2, it is required that your DDS and your DAC interface both be clocked from the board provided 125MHz clock on pin K17. It is also required that the processor infrastructure **not** be running on that clock, but instead be using FCLK\_0 from the PS as in earlier labs. This means that there will be some interaction between the two clocks. You will be required to do this safely. The submission will require that you submit a short description which details all signals that go between your two clocks, and how you made that to be reliable.

***Part IV: Software***

Now you will write some PS software which will demonstrate the capabilities of the DDS.

Requirements:

* Print a menu when the software starts. This menu should include your name and document the user interface. There should be a simple command to reprint the menu.
* The user should be able to set the frequency of the DDS to the nearest Hz by entering a string and hitting return. For example, something like the following example inputs should be valid (you can do something better if you like):
  + f 1 [enter] (this configures the DDS to a frequency of 1Hz)
  + f 10020 [enter] (this configures the DDS to a frequency of 10020 Hz)
* Ability to take a serial input and step up and down in 100 Hz and 1000 Hz steps. The output frequency and configuration word should be printed. For example, you could have the following commands: (Note, these should not require a carriage return, just the keypress)
  + u (this steps up 100 Hz)
  + U (this steps up 1000 Hz)
  + d (this steps down 100 Hz)
  + D (this steps down 1000 Hz)
* Any time the frequency is changed, either via the uUdD keys, or the f command, the software should print out the current frequency in decimal Hz, and the current phase increment in decimal (this is the number that you are writing to the DDS itself)
* The user should be able to reset the phase of the DDS to 0 via a serial command. For example, you could have the following command:
  + z (this sets the phase to zero. **Please note that setting the \_phase\_ to 0 is not setting the frequency to 0.** It is simply restarting the angle at 0 degrees. The instructor design pulses a reset line to the DDS to accomplish this. This should not require a carriage return, just the keypress

Note: doing the computation for phase increment may be difficult to dousing only 32-bit integer variables, since the intermediate products of the math are large (though the final result clearly can fit in a 32 bit integer). To overcome this, you may want to do the computation using floating point variables, or a 64 bit integer (long long)

Test your design and make sure that it is generating tones at the proper frequency. Cycle through various frequencies using the up and down keys, and observe the aliasing effect as your DDS frequency goes above fs/2 (fs, the sampling frequency of the DAC is 48.828kHz).

***Part V : Quantitative Validation***

Once you have qualitatively validated the design with your ears, it is now time to validate it a bit more quantitatively. Our goal will be to capture the output of the DDS in-system, and compare it with the data we captured from our testbench and/or matlab model. To do that, you will need to capture samples from the DDS via the ILA. These will be compared sample-by-sample with the data from your simulation, so an important part of this capture is to make sure that both start at phase = 0. This happens in the simulation for “free”, because you are capturing from time 0; however in the implemented version, your FPGA will be constantly generating the sinusoid and your capture will not ever start at phase 0 except by extreme chance. This is where the reset on the DDS comes in, you can use this to set the DDS phase to 0 and start “from the beginning”. Additionally, you can use this signal to trigger your ILA capture, so that the ILA can grab data that is aligned to a known point in time. (Hint: if you use reset to trigger the ILA, the first data point will come out a fixed number of clocks after the trigger. From your simulation, there might even be a better signal to use as the ILA trigger?)

Note: to save data from the ILA, use the following TCL command:

write\_hw\_ila\_data -force -csv\_file foo.csv

This will write a CSV file which has your signals from the waveform viewer at each clock cycle.

Edit your matlab model from the last lab to compare the modelled data with this new datafile that you just recorded. The instructor provided matlab code example from Lab3A shows an example of how to parse that data recorded from the ILA. You will likely have to tweak it a bit since **the format of the data that is recorded from the ILA will be unique to each person’s setup – it will depend on where in the window you set your trigger, the arrangements of the probes…etc.**

**What to Turn In (all at the top level, please don’t zip items 1 and 2):**

1. A bootable image named BOOT.BIN which I can test on my board. This image should be runnable and contain all hardware and software for the lab.
2. your modified matlab dds\_sim\_16bit.m, and your data file from the ILA (the recorded and simulated data). The instructor should be able to download your matlab and dataset, and run them to produce the 2 comparison plots (same as in lab3a, this time with real ILA data). If there are any differences between the model and ILA data, or any differences at all between simulation and recorded data, please explain with text in the submission or include a word document as explanation. No explanation is required if the matlab produces plots that show ideal agreement.
3. A complete project archive for vivado and vitis. In Vitis – select File->Export to export your entire workspace to a zip file. For vivado you may make a project archive using Vivado, or simply zip the directory up yourself. Though we are grading the executable only, we will consult your project to help explain any issues and provide constructive feedback.
4. A text submission in canvas which describes the clock domain crossings. Every logical signal that starts on one of the two clocks and ends on the other should be listed. For each of those signals you should describe the method you used to make it safe. Some examples of explanations might be things like this:
   1. “this signal doesn’t matter, it can’t cause a failure that any user would care about”
   2. The XYZ signal is created on clk1 and is used by component ZYX on clk2. The clock domain is crossed with (a xilinx CDC macro, dual clock FIFO,….some other vetted clock crossing component)
   3. The abc signal is created on clk1 and used on clk2, I registered this signal with a N flipflop synchronizer chain that I built
5. the text of the results from the CDC Report in Vivado

Note: in this lab we are focusing on actually making the clock crossings safe, not on making sure that the tools recognize them as so. You may still have timing errors reported by the tools in this lab assignment, what is important is that you properly understand all hazards and have corrected them functionally