Physics 623 - Electronic Aids to Measurement
Fall 2018
FPGA lab # 2 Digital Phase Detection

1 Introduction

Last week you learned the basics of loading programs onto an Altera FPGA and running them. At the end of the lab last week, you were also introduced to your first Verilog program, which was used to create a state machine that cycled through different patterns of the LED's. This week, you will create a Verilog program that acts as a digital phase detector between two input signals. If you recall, phase detectors are a critical component for the creation of Phase Locked Loops which are used in Lock-In amplifiers and many other devices. The phase detector in this lab leverages the built in ADC converter on the DE0-Nano board, which we discussed on class. The basic idea of this lab is to read the incoming signal at two of the input ports of the ADC converter, sum the digitial signals together, and then average them over time. Unlike last week, everything today will be done in Verilog, we wont be using schematic capture.

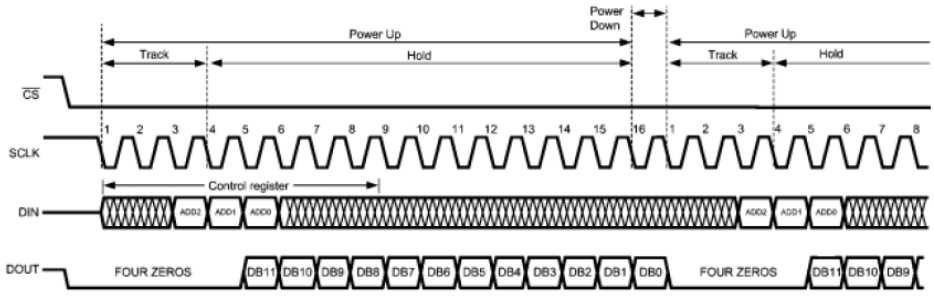

Figure 1: timing diagram for ADC converter on the NanoBoard

2 ADC converter

The DEO-Nano board includes a national instruments ADC128S022 Analogue-to-digital converter, and the documentation for it can be found on all the lab computers. This converters has 8 different inputs, and the output is serial. Basically, over the course of 16 clock cycles, the ADC first reads 3 bits sequentially, which determine which pin to record data from (on DIN) and then it spits out 8 bits sequentially (on DOUT) which is a digital code for the voltage it read. The timing diagram for it is shown below. (Recall that the clock that controls DIN is slightly phase shifted from the clock that controls DOUT, you will not need to worry about this, but you should not be confused when you see iCLK and iCLK_n in the code).

The ADC chip is wired up to the FPGA and to the pins on the DEO-Nano board as shown in Figure 2. The address names you see coming from the ALTERA chip (like ADC_SCLK) are dedicated names, similar to CLOCK_50, and LED[7..0] from the lab last week. The first thing you should do is make sure the ADC converter is working. To do this connect your nanoboard to the computer, and open the DEO-Nano control panel program. Connect to the board and click on the ADC. Attach a wire to the ground pin (pin #26) and connect another wire to one of the input pins. Run the two wires to the tunable DC power supply (60mA) and apply a voltage

between +/- 3V. You should see it measured on the control panel.

Next you are going to load the basic ADC program onto the FPGA. First, copy the "DEO_NANO_ADC" folder (and all of it's contents) to your personal folder on the desktop. (you can find DEO_NANO_ADC in "623/DEO-Nano/Demonstration/"). Go into the folder that you just copied, and open the DEO_NANO.qpf program, which should open the Altera Quantus software. This program contains two sub-programs, DEO_NANO.v and ADC_CTRL.v. The first of these programs (DEO_NANO.v) basically defines all the pins and variable for the FPGA to interact with the ADC, and the second program (ADC_CTRL.v) contains all the real code that defines how the ADC chips is interfaced. For this lab, we are only really going to be modifying the second file, but the first file (DEO_NANO.v) contains the main module from which the program is launched.

The basic ADC readout program should already be ready to go. Double click the "Compile Design" button to compile it, and afterwords double click the "Program Device" field. Select the program and hit "Start". Note that you may need to unplug and replug the board a few times from the USB. Re-connect your variable DC power supply. You need to flip the dip switches on the board to select the right pin that you are connecting your voltage to. Also, you may need to hit the two buttons on the Nano board to reset it.

Question 1: How does the LED output pattern correlate with the applied voltage as you vary the voltage to +/- 3V? Describe the binary to voltage decoding rule.

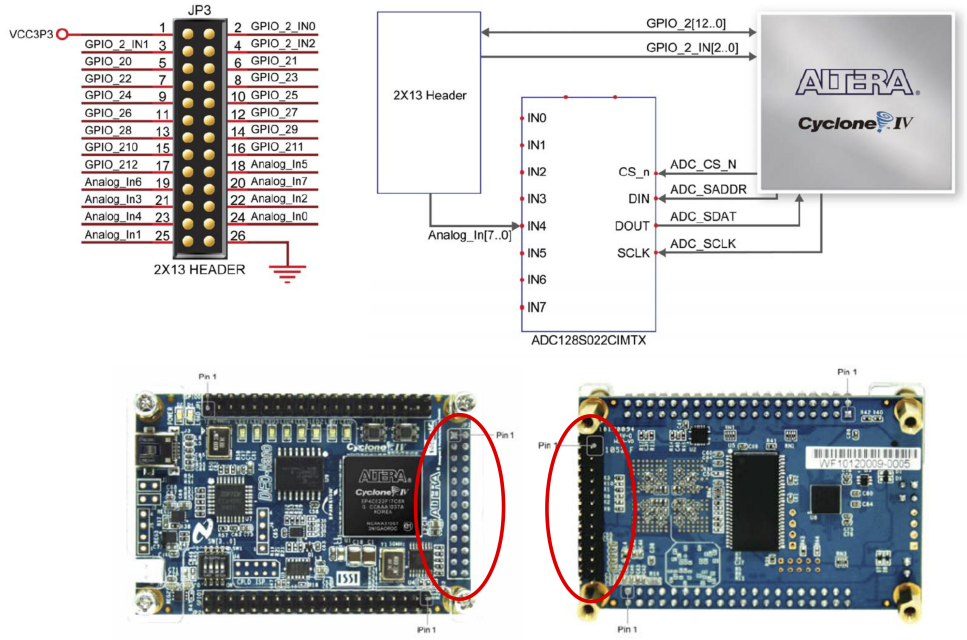

Figure 2: Pinout and connection buses between ADC converter and the Altera FPGA.

3 Averaging

Next, disconnect your variable DC power supply and hook up the ground and input pins to your waveform generator. To begin with, create a square wave at 1Hz. Make sure that the signal

never exceeds +/- 3V. You can use an offset if you want, but the signal should not exceed +/- 3V. **Describe how the LEDs behave now**

Now begin increasing the frequency of the output wave and observe how the LEDs behave **At what frequency does the FPGA begin to output a non-sensical pattern on the LEDs?**.

Now, set the square wave frequency to 1kHz and make the lower limit 0V, and upper limit 2V. **What voltage does the LED pattern appear to indicate**

Verilog Project #1: Change the code of the ADC_CTRL.v file such that the voltage output represents a time average over 40ms. Show that your code works to the instructor to have this section of your lab checked off. Include the relevant section of your code in you lab notebook, and be sure to include comments.

4 Reading Two Channels

Verilog Project #2: Change the code of the ADC_CTRL.v file such that the voltage output represents an average between two of the ADC inputs. To test your program, use two of the variable 60mA DC voltage supplies. Show that your code works to the instructor to have this section of your lab checked off. Include the relevant section of your code in you lab notebook, and be sure to include comments.

5 Sum Two Channels and Average

Verilog Project #3: Finally, you are going to make a digital phase detector by reading two of the ADC inputs, summing the signals together, and averaging the sum over 40ms. Change the code of the ADC_CTRL.v file accordingly. To test your program, use two of the waveform generators to output a 1kHz square wave with a 0V minimum and a 3 volt maximum. Vary the phase of one of the waves to show that your phase detector is working. Show that your code works to the instructor to have this section of your lab checked off. Include the relevant section of your code in you lab notebook, and be sure to include comments.

For this final section, you may find it useful to output the phase on only 1 LED at a time, rather than giving a complicated binary signal. To this end you may want to call a decoder program similar to the one below, which outputs a small number as 00000001 on the LEDs, and a big number as 10000000, with 6 ranges in between.

```
module decoder(a, b);
input [7:0] a;
output [7:0] b;

assign b = (a<8'h20) ? 8'b00000001 :
((a>=8'h20) && (a<8'h40)) ? 8'b00000010 :
((a>=8'h40) && (a<8'h60)) ? 8'b00000100 :
((a>=8'h60) && (a<8'h80)) ? 8'b00001000 :
```

```
((a>=8'h80) && (a<8'ha0)) ? 8'b00010000 :
((a>=8'ha0) && (a<8'hc0)) ? 8'b00100000 :
((a>=8'hc0) && (a<8'he0)) ? 8'b01000000 :
8'b10000000;

endmodule
```