Random Number Generation Instructions:

1. Connect station probes to MTJ
   1. Using the probe station, move the probes one at a time over the MTJ pads, then lower slowly until the probe slides inwards toward the intersection where the MTJ is located. Always hold the ground wire, or wear the ground bracelet when interacting with the probe station.
   2. Optional: connect station probes from MTJ to the keithley signal generator, and run sweep\_clark.py to check for hysteresis loop
   3. Or instead connect probes from MTJ to the FPGA board, PULSESUMOUT connects to the wire that goes around the probe station, OPIN connects to the output, the probe close to the desk. Check for 2 distinct states on the oscilloscope once the MTJ is connected.
   4. Scope probes : using ground spring, probe pulseout located under the PULSESUMOUT cable. Connect 2nd probe to OpOut hole located under the opout cable
2. Program the FPGA
   1. Open Quartus prime, and under recent projects select altera-eth…., usually the top choice
   2. Under files, select C:/git/LL10G/altera\_eth\_top\_time\_limited.cdf, to open the FPGA programmer.
   3. Click start in the top left, if the bar in the top right goes to 100 percent and says complete, the FPGA is programmed, if it instead shows (failed) follow the next steps.
   4. Open process explorer from the toolbar, search jtagserver.exe in the search bar
   5. Select -> kill processP
   6. If access is not allowed, select file -> run as administrator, then open task manager and kill process, or in the administrator terminal enter command “tasklist” find jtagserver.exe, and run command to kill process
   7. Return to FPGA programmer, click hardware setup, check that USB-blaster is selected, this may take a minute or two after killing the jtag server.
   8. Exit the hardware selection page, and click start again, now the process should complete
3. Compiling firmware: takes 15-20 minutes, so be precise about making changes to firmware!
   1. After making changes to firmware, click the play button in the top, or processing -> start compilation, to begin the compilation process.
   2. Close the open core plus status window that is opened on programming, or quartus will close when compilation is complete. This is fine compilation will save either way
   3. If several errors appear from files you have not changes, close quartus and try again
4. Setting output waveform: once firmware is set, can begin to generate bits
   1. In quartus > tools > system debugging tools > system console
   2. Expand TD console in bottom right
   3. Enter commands in order: cd hwtesting//system\_console,  source main.tcl,  SEND 0 10G 1000000000000, (# of 0’s doesn’t matter, just put a lot), if one of these fails close and open a new console
   4. Now pulse commands should work, to set the width of the 4 pulses, the command is PULSES\_pos#dur ##, typically, all can be set to 10. Ex, PULSES\_pos1dur 10 sets the first pulse to 10 clock cycles
   5. To set pulse amplitudes and dac offsets, the command is: dacwrite # ## ####, or dacwrite chip channel value. The 4 pulse amplitudes are on chip 0, channels 0 for writeda, 5 for reset, and 6 (verify and read pulses are both channel 6, as they are identical), ex, to set the write pulse amplitude to 700, dacwrite 0 00 0700, or dacwrite 0 0 700 (leading zeros can be omitted).
   6. Starting values that have worked well are
      1. dacwrite 0 5 1200
      2. dacwrite 0 0 700
      3. dacwrite 0 6 350
   7. The amplifier offset are on chip 1, channel 0 is amplifier offset, channel 1 shifts up and channel 3 shifts down
   8. The strategy I use to generate a good output waveform is to set dacwrite 1 0 0 so there is no offset, often two states are visible with no offset. Then I increase dacwrite 1 0 until one state is high and the other is low, you should not have to increase it higher than about 1500, once close to having two clean states, increasing dacwrite 0 6 can be used to lower the low state, and increasing dacwrite 1 0 raises the high state, adjust until there are two distinct states.
   9. Once the output trace has a high and low state, we are ready to collect data
5. Collecting data:
   1. Open leanlistener\_2.py in notepad++
   2. At the bottom of the file, change the 3 filenames that record bits, write pulse voltage, and time, and set n\_bits\_to\_collect to the amount of bits you wish to record
   3. Open two command prompts, on the first: cd Documents > cd variable-write-pulse-experiment > python [leansender.py](http://leansender.py)
   4. ON the second: cd Documents > cd variable-write-pulse-experiment > python lean\_listener\_2.py
   5. Back in the quartus tcl terminal from step 4, run offset ## to tell the FPGA where to read the bits, I have found offset 35 to be about where the bits should be. Now probabilities should be showing up in the leanlistener\_2 command prompt, check to see non 1 or 0 probability
   6. Adjust offset up and down one or two to make sure you are not reading near and edge, adjust write pulse to get near 0.5 probability, and random bits are being collected.
6. Additional tcl commands:
   1. Turn on and off XOR operation, xor 0 is off xor 1 is on, it is on be default
   2. Feedback: feedback 0 turns on feedback, feedback 1 is off. It is set to off by default, IMPORTANT: dacwrite commands will not work while feedback is on
   3. Other commands can be added in the tcl console, I can make another manual on how to do this
7. Important Firmware files:
   1. St\_gen file is where bits are collected and ethernet packets are assembled
   2. Long\_xor is where the xor is performed, currently performes a 4096x4096 xor
   3. Pulse\_controller is where the pulse cycle is controlled from
   4. Pulse\_adjuster is where feedback is controlled, target probability and other parameters are set
   5. I2c.v  is where i2c commands are sent to dacs from, works well enough for now, though a very bear bones i2c controller

Extras:

DAC channels and corresponding PulseController switch addresses:

\*\*all channels are on chip 0, so to change voltage on a given channel in tcl -

Channel       address          polarity

0         0001\_0000      (-)

1                  0010\_0000      (-)

2                  0100\_0000      (-)

3                  1000\_0000      (-)

4                  0000\_0001      (+) tcl control currently not working, likely a bug in i2c module

5                  0000\_0010      (+)

6                  0000\_0100      (+)

7                  0000\_1000      (+)