Final Report

Covert Power Side Channel: Measuring Power in a FPGA

Brandon John

FPGAs have started appearing in datacenters such as AWS, allowing users to rent time on FPGAs to help with their heavy compute tasks. These FPGAs vary, with one variant being built as an SoC (System on Chip) with a processor and FPGA fabric in one package. There can also be scenarios where users can rent a portion of an FPGA, and another user uses another portion of said FPGA. In this scenario presumably the two users would have all signals isolated from each other, but would share voltage supply rails and other ambient conditions. Both of these scenarios share one key detail: the FPGA logic is supplied by a voltage rail that can be dependent on the activity of its neighbors. This shared rail allows for a power side-channel attack to covertly extract data.

In general, power side channels are a useful tool for extracting data from an otherwise secure process. Traditionally they have required physical access to the machine to install a voltage or current probe, such as measuring the voltage across a test resistor or inserting some other mechanism to have a high-accuracy high-granularity reading of the power being drawn. The basic idea is that as a processor works on compute-intensive tasks, it draws more power. This can even be used to decide the difference between a low-power add or move instruction vs a high-power multiply instruction. If a program is data dependent – say it performs a multiply in certain cases and an add in others, a power side-channel can be used to help determine the original conditions that decided between the multiply and add. This has been used to extract secret encryption keys and such from common implementations of various cryptographic programs.

The important question here is if we can design some logic on an FPGA to measure the current power consumption of neighboring silicon, and use this to determine the current activity of said neighbors and extract some secrets. It is known to be possible to covertly measure voltage rails inside an FPGA via the FPGA's fabric, and that certain high-level power usage trends can be observed. But how far can we take it?

In this work I designed and implemented a testbed system for running ring-oscillator power measurement experiments on an FPGA. This system currently runs on a ZYNQ-7000 SoC, a device that contains both an FPGA fabric and a general purpose programmable processor. The processor is used to dynamically configure the fabric and run victim programs. The dynamic configuration can select the number of ring oscillators being sampled, clock frequencies and integration periods, and also set the "power virus" level which can be used for characterization under different loads. Once the processor initiates an experiment, the results are stored in the fpga's BRAM, which can be later read back by the processor for post processing and analysis.

Individual ring oscillators (ROs) are constructed out of a single not gate feeding its own input, and also a chain of TFFs as an asynchronous counter. This RO oscillates at a rate that is a function of the local temperature and voltage. This voltage is a function of the current load in the SoC, as described in Zhao's paper. This counter can then be sampled at a known rate defined by an external clock, allowing for determination of the true oscillation frequency and is thus a proxy for the current load on the SoC. I did make a slight change in the RO layout, specifically I only used 3 layers of T flip flops at the beginning, which I then used the output of as the clock for a synchronous counter. I couldn't use a counter primitive from the beginning as this was too slow to capture all transitions, but using it on the RO/8 frequency worked quite well. The major advantage of using this counter primitive was simply routing space – each TFF required a full logic block to implement, as each block shares a single clock input to

feed the latches and DFFs. I could fit 8 counter bits in a single logic block, where I would otherwise fit only 1 TFF, which made the implementation step much easier and quicker for the toolchain. See Figure 1 for a diagram.

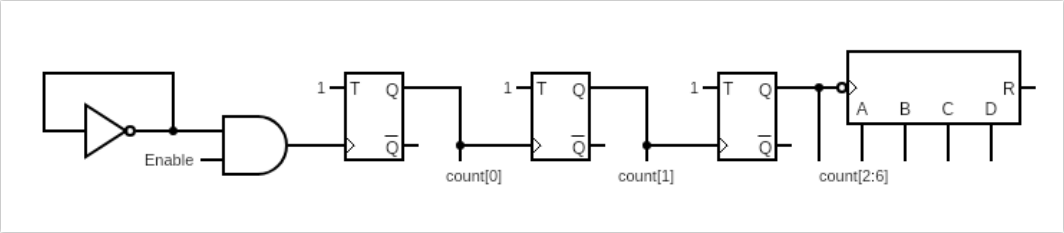


Figure 1: Ring Oscillator with 3 TFFs and 4+ bit counter

With this infrastructure out of the way (though admittedly the majority of my time spent on this project was devoted to building the infrastructure!), my focus shifted to my original goal – I wanted to find a way to characterize the tradeoffs involved in defining these ring oscillators, and how that related to the final result's accuracy and precision. To that end I focused on the tradeoffs between the number of ring oscillators instantiated and the integration period. Intuitively, having more ROs simultaneously taking measurements allows for greater accuracy, and likewise increasing the integration period allows for increased accuracy. However, increasing the integration period also reduces the bandwidth of measurements, meaning short duration events are impossible to measure.

One of the first significant challenges is that individual ROs can operate at vastly different frequencies. As can be seen in Figure 2, individual ROs varied by ~10% comparing their full scale steady state values (i.e. the blue RO varying from 1.8 to 2.0GHz), while there can be over 2x difference in magnitude between separate ROs (purple is always less than ½ of blue in this example). This issue can probably be reduced with very careful constraints for the routing step, though I did not investigate further. Instead of optimizing this away, I made all measurements with a large number of ROs averaged together (generally between 30 and 70), meaning this effect probably only minimally effected my results.

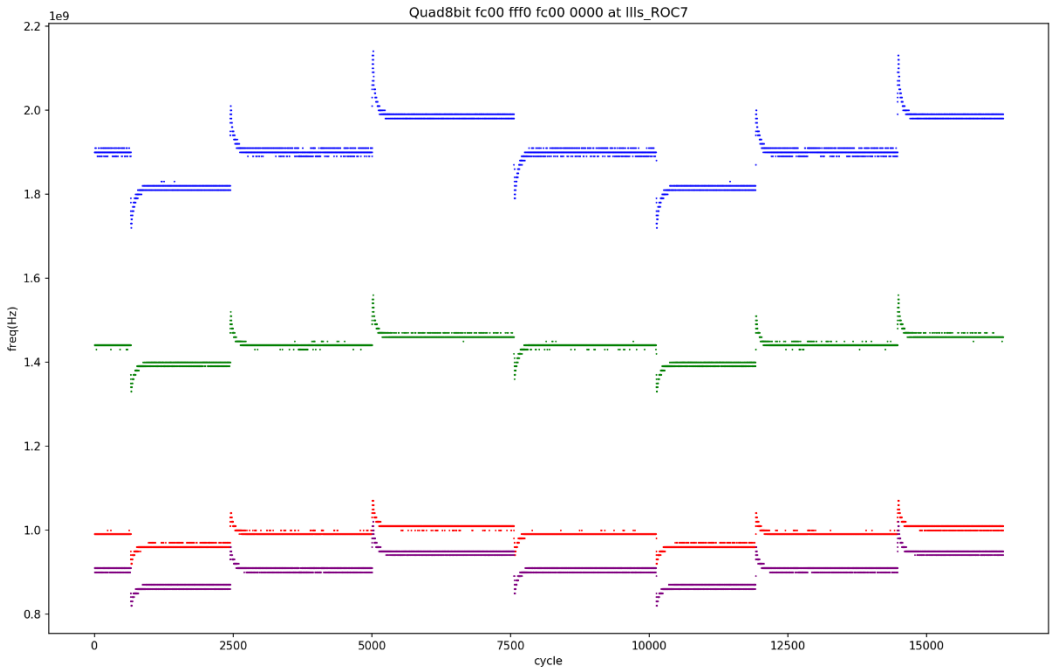


Figure 2: Frequency of 4 separate ROs recorded simultaneously, each "cycle" is 1 recorded sample, sampling at ~12MHz

I also had a plethora of engineering challenges, some of which I had time to meet and others I had to deal with. A big one was that due to the way I pipelined my system, I could only use approximately 70 ROs at a time, as I couldn't add up enough sums quick enough to do more. This can be resolved through more careful pipelining, but I did not have time to deal with this. Likewise, I had a fairly large number of issues with constraining the design. This was a particularly interesting challenge because I was trying to create constraints around combinatorial loops. Whenever I went searching for advice on creating these constraints, the answers online were always "fix your RTL, combinatorial loops are always bad". But this clearly wasn't the case for me, as my RTL was making these loops on purpose. I did succeed in waiving most of the warnings generated, but I did struggle to make progress on specific timing constraints. I ended up reducing the sample clock frequency from the planned 200MHz to only 50MHz, which worked fine for early testing but definitely is not ideal long-term as this is a 4x reduction on its own from the best possible temporal accuracy.

While there are many challenges waiting to be solved and optimizations that can be implemented, the system is already functioning. As shown in Figure 3, we can extract data from the toy example described in Figure 4. In this example I have some code that pulls either a large or small amount of power from the circuit depending on the "secret" bit for about 3ms, and then move on to the next secret bit. This causes the ROs to oscillate slightly faster or slower on average. I then manually added the grid markings based on a visual analysis of the data, and could then determine whether the power was mostly lower or higher for the duration of that sample. While the fluctuations are admittedly quite minor, I sampled 3 family members and they got quite good accuracy on their first attempt, which tells me that this is a reliable way to extract data and that I'm not just imagining the differences with my prior knowledge of the correct answer. However, it did take a few tries to get data that was this clean, some attempts had major noise spikes that blocked out multiple bits (similar to the spikes at 3000us and 9000us in this example). One interesting note is that the section that I designed to be high-power, specifically the loop of exponents, was actually lower power than the "pass" loop. This can be seen in that the exponent loop caused the ROs to oscillate faster, meaning less power was being drawn. The pass loop on the other hand actually used more power, reducing the RO oscillation frequency. This is

certainly a quirk of how Python is dynamically parsing and executing the code, but it also shows that very small deltas in power can be successfully measured.

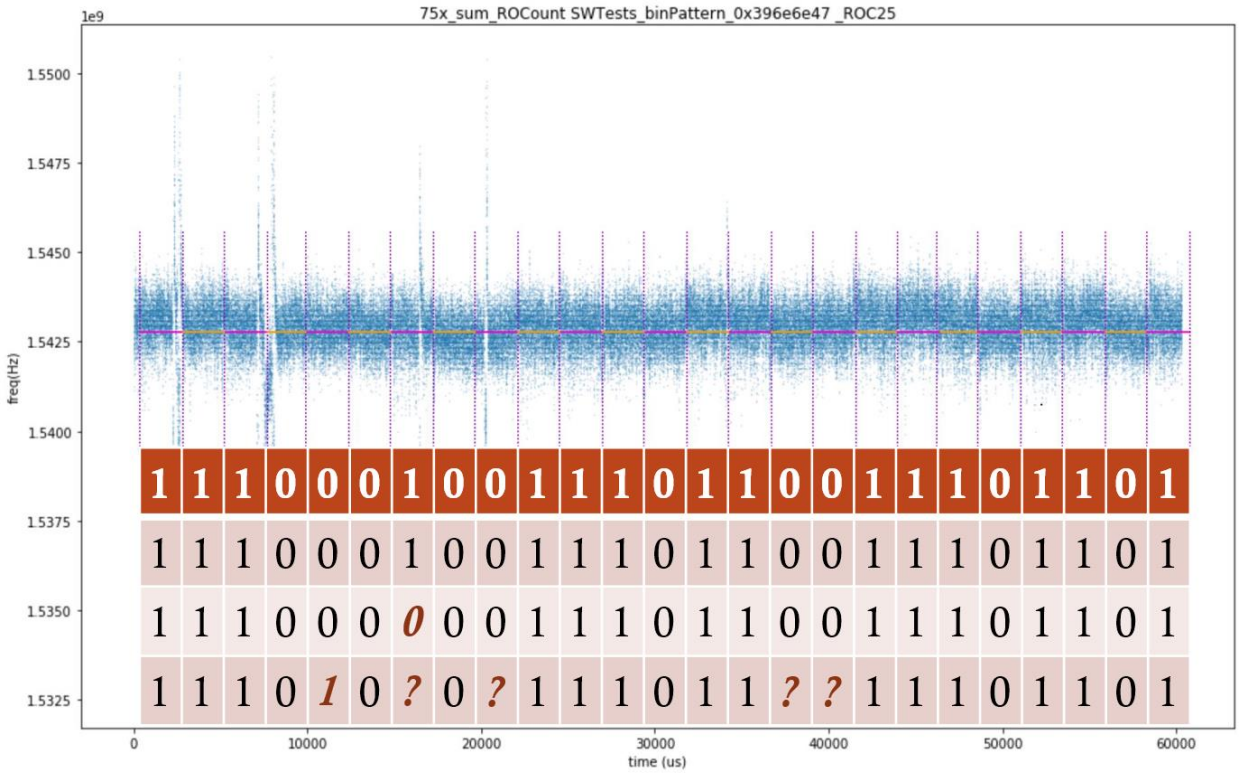


Figure 3: Example 60ms capture, recorded with 75 ROs, a integration period of 500ns, running a software victim. Expected data is shown in row 1 (white on red), extracted values by 3 untrained friends in rows 2-4.

```
def func_hp3ms(): #~3ms duration
    y=7
    for x in range(1,8):
        y=(y**x)

def func_lp3ms(): #~3ms duration
    y=7
    for x in range(1,10000):
        pass

def func_binPattern():
    for a in range(20):
        val = 0x396e6e47 #Secret value, LSB first
        for z in range(32):
            bit = val&1
            val = val>>1
            if (bit):
                func_hp3ms()
            else:
                func_lp3ms()
```

Figure 4: Code used to generate Figure 3

I have many ideas for potential future work, in addition to the challenges that I have already noted above as having left unsolved. The first is a statistical model of the ring oscillator measurement system, which would account for the quantization randomness of the measurements. This may lead to a solution other than just taking the mean average of all enabled ROs for determining the current value.

Whatever analysis needs to be done would have to be done within the FPGA fabric, as there is not enough storage space to simply post-process all the raw data. As it is, I can only store 131,072 samples before running out of BRAM space, dividing this between hundreds of ROs would severely limit the maximum recorded trace length.

The next obvious extension is to look at the actual power drawn via an external oscilloscope verses the measured RO frequency. This can be fairly easily done in a few ways and will allow for a more systematic way of comparing RO precision, but also requires a tool that I unfortunately don't have with me at home. I definitely plan on testing this if I ever get access to campus again, I may even buy an oscilloscope just to satisfy my curiosity here.

Another area for investigation is what it takes to actually capture data from the SoC processor that isn't already designed for being capture. By this I mean a real world program, as opposed to the toy examples I used that purposely went into high and low power modes for some known period of time. I expect that this may involve lots of pre-processing of the data inside a custom pipeline in the FPGA for storage purposes, and perhaps a sets of ring oscillators that are out of phase and some nifty signal processing to get finer time granularity than is otherwise currently possible.

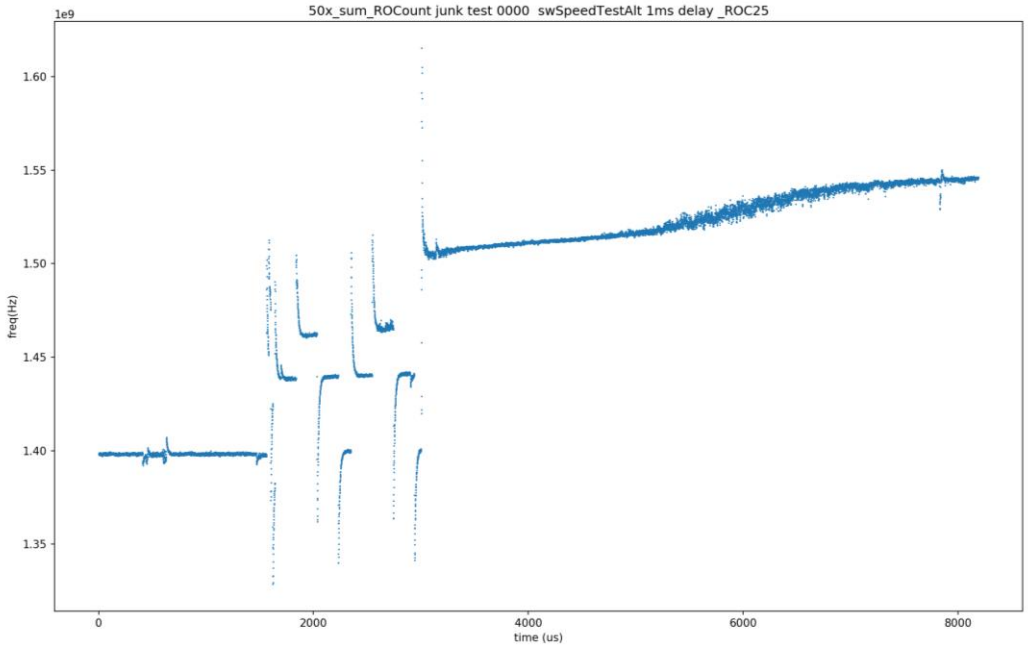
Overall I have clearly proven that ring oscillators can be used in power side channel attacks on FPGAs, and can accurately extract data from a collocated processor running programs with secret data. There is clearly much work left to be done to optimize this project and see just how far it can be pushed, but now the infrastructure is available for whoever wants to build off of it.

Appendix:

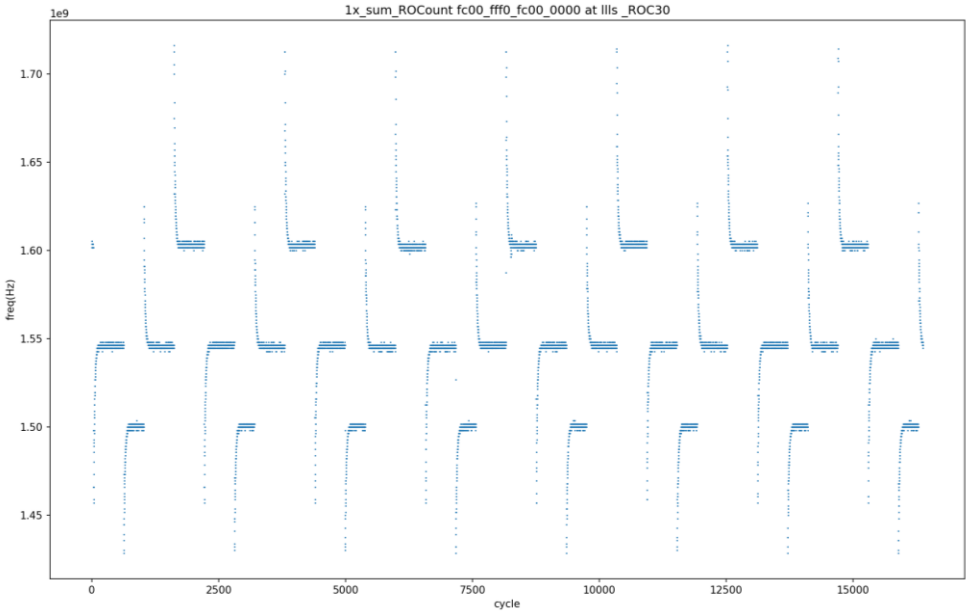
During this work I made a few interesting observations. The first is that when turning on a lot of power viruses at once, I could actually brown out the processor to the point that it just completely crashed, but the FPGA fabric would continue operating after the brownout independently. This could potentially be used either as a denial-of-service attack or perhaps to cause precision corruption of the processor state. I also discovered that by turning power viruses on and off at the appropriate speed, I could make the power supply resonate and create audible pitches with it. This could potentially be used to extract data from an air gapped system, or some other place where an FPGA bitstream could be loaded but the data couldn't be returned.

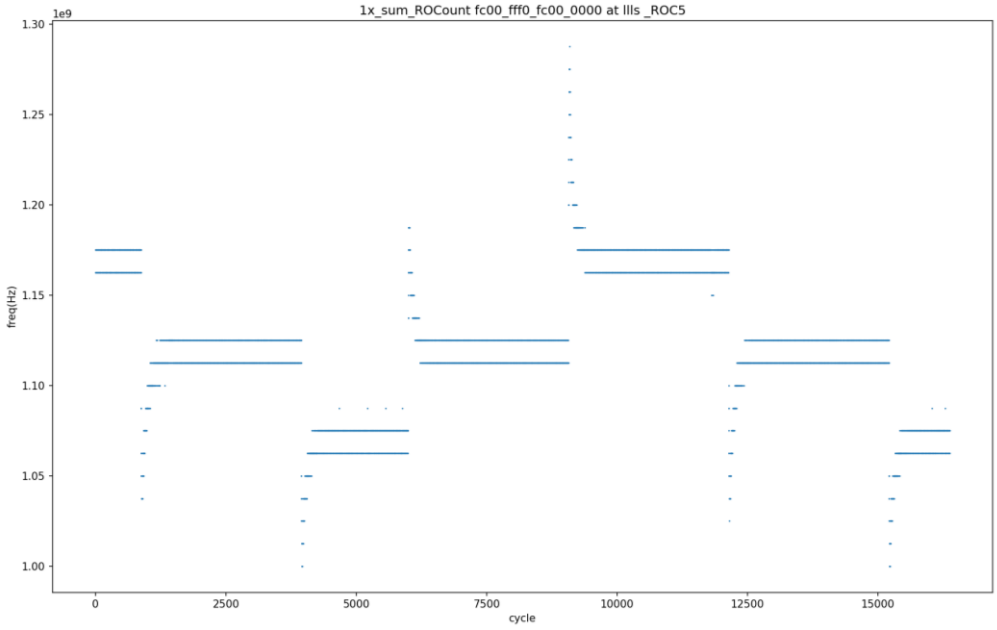
I am also including a few plots below of various experiments that didn't fit in the report above but are still interesting to consider.

First, here is a plot showing the transition from a very high power virus level, to an alternating mode starting at 1.8ms, to completely off at 3.5ms. This shows a few interesting behaviors – the first is that there is significant overshoot in the measured voltage when applying step changes in load levels, which is consistent with the RLC style network transferring power from the DCDC converter to the FPGA, and also hints at the DCDC converter's limited response time (it switches at ~1.5MHz, meaning we could potentially even see individual steps of the power supply with some optimization of the sampling period). After the power viruses all turn off, there is a slow increase in measured period of almost 4% from 3.5ms to the end of the capture, and this is actually showing how the oscillation frequency is also a function of the temperature! When the power viruses all turn off, the chip starts cooling down over the course of a few ms, and this increases the speed of the measuring ring oscillators.

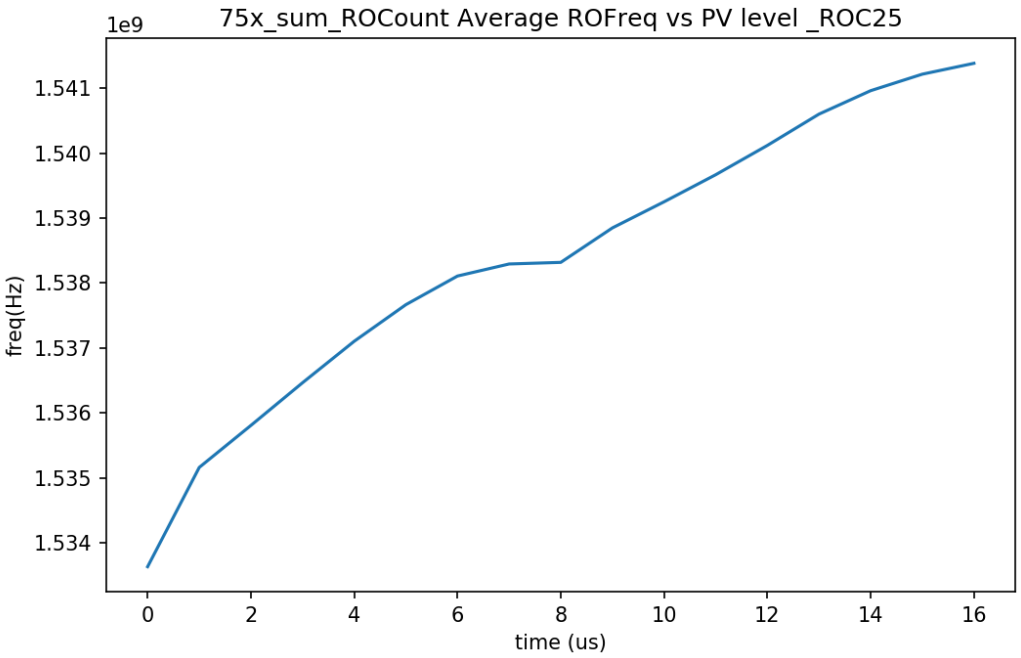


This next pair of plots show the difference in precision available by increasing the integration period. The first graph integrates for 30 cycles, while the second only integrates for 5 cycles. The difference is Both are measuring a single ring oscillator, with the power viruses running a low-med-high-med power pattern. For context, high power runs for slightly shorter duration than medium and low. This again shows the overshoot issue, but more importantly the first graph has much tighter bounds on recorded data, as the quantization issues mean a +/- 2 range over the longer duration is still a smaller frequency range than the +/- 1 over the shorter duration. (Ignore the exact scaling factor on the frequency, that was an unfortunate bug when generating the data for these graphs).

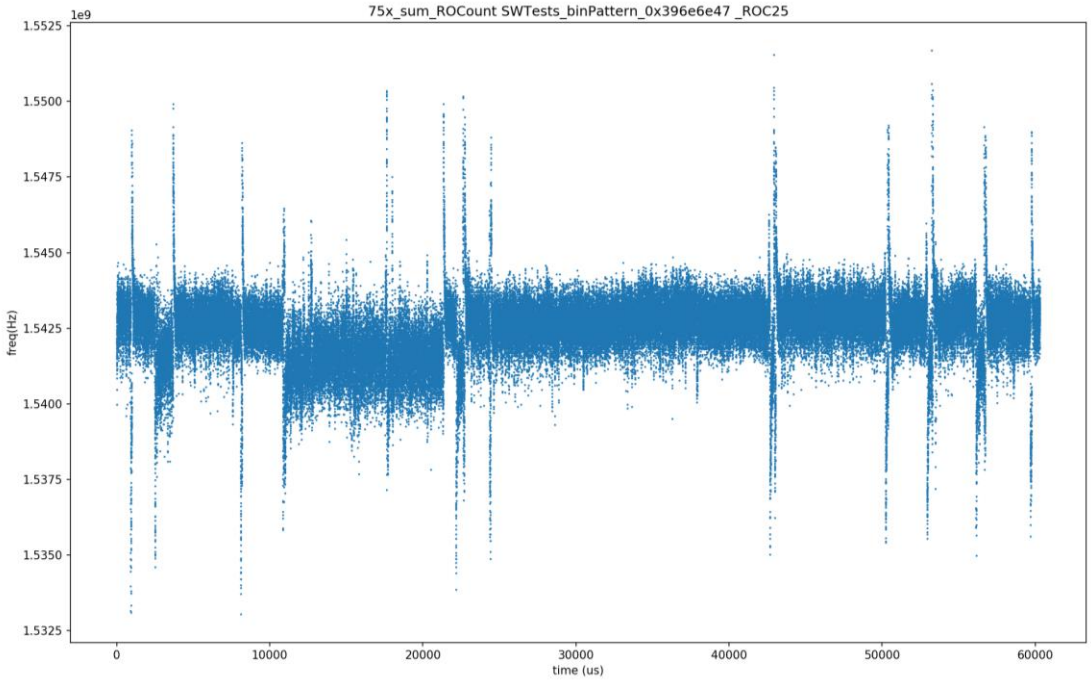




I can also show the average RO count verse the PV level applied, where each step of PV level equates to enabling a bank of 500 ring oscillators that are not counting, just using power. Interestingly bank #8 always has a very small impact on the total power, this is perhaps because that bank was placed far away from the ring oscillators. This was consistent for this bistream but changed after regenerating the bitstream, which makes me strongly believe this is a function of physical placement parameters.



Finally, I had previously mentioned that I would often get significant noise, presumably from some operating system scheduling process. Here is a particularly egregious example for reference.



Source code will probably be published here: https://github.com/wiresboy/fpga-power-sidechannel

References

"TUL PYNQ Z2 Schematic R12," https://d2m32eurp10079.cloudfront.net/Download/TUL_PYNQ_Schematic_R12.pdf , TUL Corporation, accessed: 2020-10-08

"Zynq-7000 SoC: DC and AC Switching Characteristics," https://www.xilinx.com/support/documentation/data_sheets/ds187-XC7Z010-XC7Z020-Data-Sheet.pdf, Xilinx, Inc., accessed: 2020-10-08

M. Zhao, and G. E. Suh, "FPGA-based remote power side-channel attacks" S&P. 2018.

F. Schellenberg, D. R. E. Gnad, A. Moradi, and M. B. Tahoori, "An inside job: Remote power analysis attacks on FPGAs," in 2018 Design, Automation, and Test in Europe Conference and Exhibition (DATE), 2018

"Amazon EC2 F1," https://aws.amazon.com/ec2/instance-types/f1/ , Amazon.com, Inc, accessed: 2020-10-07.