Creating a Self Contained Power Monitor Framework for Side Channel Attacks Against FPGA-Based Ciphers

Rahul Jayachandran
Mentored by Dr. Laurent Michel, Professor, Computer Science and Engineering, University of Connecticut

ABSTRACT

The FPGA, or Field Programmable Gate Array, has rapidly become an integral part of cloud based datacenters. FPGAs are commonly used to run cryptological ciphers, such as AES and RSA. This paper shows that these integrated FPGAs are susceptible to power based side channel attacks, even without physical access to the device. We first created a power monitor within the programmable fabric of the FPGA, and defined a relationship between its frequency and the power draw of the FPGA. Then, we constructed a sampling framework using only tools provided in the Vivado SDK (Software Development Kit) and the FPGA's integrated SoC (System On Chip), and demonstrated its function. By creating a self contained power monitor and sampling system, we demonstrated the vulnerability of popular cloud based FPGA + SoC datacenters.

INTRODUCTION

FPGAs are a type of semiconductor device based on a matrix of logic blocks connected by programmable contacts. Each block is configurable, which allows the FPGA hardware to be reprogrammed to fit different tasks. This allows for incredibly quick calculation speeds when compared to a CPU based system, since the hardware itself is designed for the specific task (Field Programmable Gate Array (FPGA), n.d.).

As the world continuously demands more computing power, FPGAs have been rapidly integrated into large scale computing systems. This phenomenon is further demonstrated by cloud based datacenters, with companies like Amazon allowing FPGA rental on their cloud based EC2 service (Amazon EC2, n.d.). FPGA vendors such as AMD's Xilinx boast workload optimization of up to 10x throughput with 1/10 of the power draw of CPU/GPU based systems in the cloud. To this end, Xilinx, along with Intel and other hardware vendors have introduced systems that combine CPUs and FPGA fabrics on one device (Cloud Computing, n.d.).

One popular use of FPGAs is encryption and decryption of data. Many industry standard ciphers, such as AES (Advanced Encryption Standard) and RSA (named after creators Rivest, Shamir, and Adleman) have been implemented on FPGAs to accelerate their efficiency (Rodriguez-Henriquez, 2007). However, there are methods to attack ciphers while running. A side channel attack leverages side channel information, such as the power usage of the cipher, to recover the key of a cipher. For instance, TASCA (a Tolerant Algebraic Side Channel Attack) uses a power trace and a combinatorial solving methodology to recover keys while a text is being

encrypted. In their paper, researchers Lui, Cruz, Johnson and Michel used a TASCA against AES running on an 8-bit microcontroller. Their solver, in conjunction with power trace, was able to recover the key with a high level of precision and efficiency, thus demonstrating that AES is susceptible to power based side channel attacks on 8-bit microcontrollers (Liu, 2017). Zhao and Suh created a ring oscillator based power monitor within the programmable logic of an FPGA. They used this monitor to run a power based attack against RSA, a simpler cipher than AES (Zhao, 2018). Both of these papers, however, use some external software or hardware to offload the power trace data from the monitor. While these methods are viable in a controlled environment, it is preferable to have a self contained system, using only the tools provided by the original board manufacturer. This paper details the creation and efficacy of a power monitor framework built only using the basic hardware and software intrinsic to an FPGA and SoC board.

METHODS

I: The Ring Oscillator

The first step is to build the power trace for the FPGA. This trace, unlike oscilloscopes or other external devices, will be produced by the FPGA logic, taking advantage of the inherent programmability of FPGA boards. The project uses a ring oscillator (RO), which is a simple circuit consisting of a loop with an odd number of not gates (see Figure 1). A not gate takes an input and returns the opposite as an output (1 to 0 or 0 to 1), and is drawn as a triangle with a circle at the end in a circuit schematic. The other block in Figure 1 is an and gate, which takes two inputs and outputs a 1 if and only if both inputs are 1. We use the and gate to enable and disable the RO. This RO oscillates at a frequency inversely proportional to the power used by other processes on the FPGA. This is because all logic gates require power to switch their output, and all logic runs on one shared power rail. Therefore, when more power is used elsewhere, less will go to the RO, and the not gates will switch outputs more slowly, thus changing the frequency. Notably, this frequency is not tied to any set frequency clock, and rather oscillates as fast as the gates are able to update. As the not gates update sequentially, their input oscillates from 1 to 0, thus causing a constant state of oscillation (if there was an even number of not gates, there would be no oscillation as the circuit would reach a stable state where no gates were switching outputs). Our device under test (DUT) is a Zybo Z7-10 integrated FPGA board (see Figure 2), chosen due to its large documentation and usage of the ZYNQ-7000 processor, a standard issue SoC for all Xilinx FPGA boards (Zybo Z7, n.d.). Thus, to build the RO, we used the Vivado development environment, the standard software for Xilinx FPGA boards (Vivado, n.d.).

Figure 1

A schematic of the Ring Oscillator unit with Enable and Output wires labeled

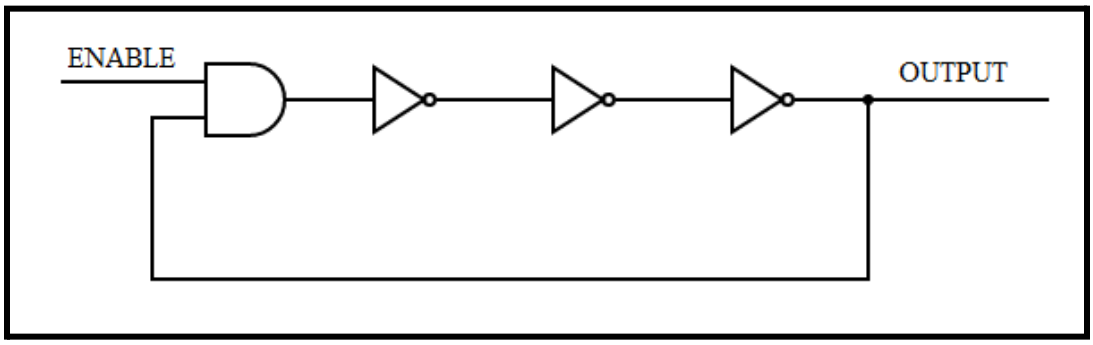
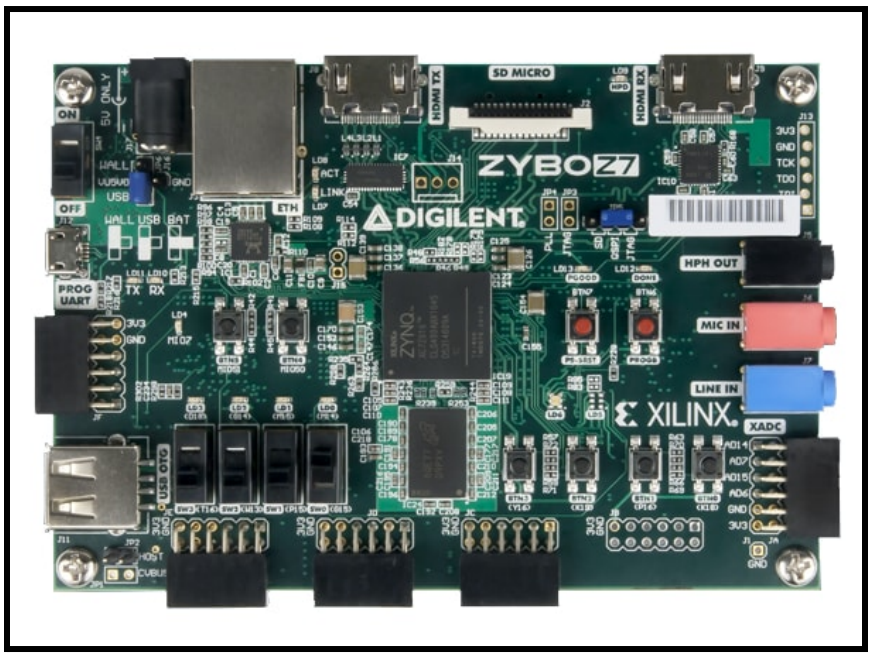


Figure 2

The Digilent Zybo Z7-10 Board featuring the ZYNQ-7000 SoC, the DUT for this project



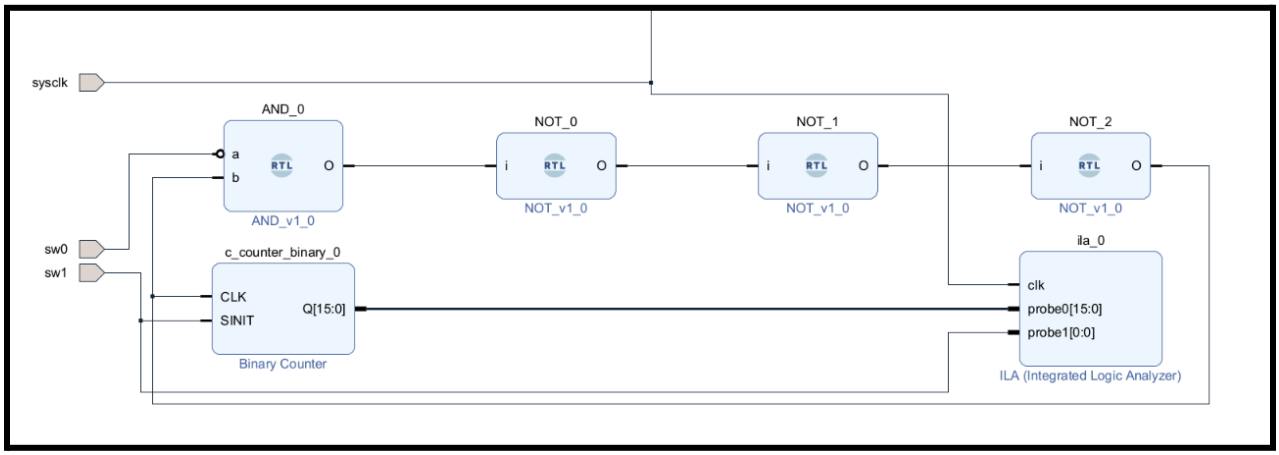
II: The Counter and Sampler Framework

Our next step was to create a sampling framework for the ring oscillator. We first implemented Xilinx's binary counter module into the FPGA fabric, which took the ring oscillator as its clock input. This counter consists of a chain of T-Flip-Flops (TFFs). A TFF takes an input T and an input clock. When T is 1, the TFF inverts its output Q upon every full oscillation of the input clock. By chaining TFF modules together, Xilinx created a binary counter module which increments a 16 bit wide output by 1 upon every oscillation of the input clock. By connecting this counter to the RO, we were able to use its value to find the number of oscillations between samples. Then, we used Xilinx's integrated logic analyzer (ILA) to sample the counter value. We connected the ILA to the integrated 125mHz clock within the PL. We configured the ILA to take

1024 samples of the binary counter, sampling at the same rate as the integrated clock, upon every triggering event. Finally, we tied the ILA's trigger to the binary counter's enable, and drove both with a single signal from the programmable logic's I/O (see Figure 3). Thus, upon resetting the counter, the ILA collects and stores 1024 samples of its value at a rate of 125mHz. Finally, we tied the output of the ILA to the same virtual serial line (built on a USB connection) used to program the FPGA. By using only IP provided by Xilinx themselves in making our counter framework, we ensured that this setup would be replicable for any ZYNQ based device, and by using the same wire to program our IP and receive the data, we ensure the system requires nothing but the ability to program the FPGA in order to function.

Figure 3

A diagram of the Ring Oscillator and Sampler setup



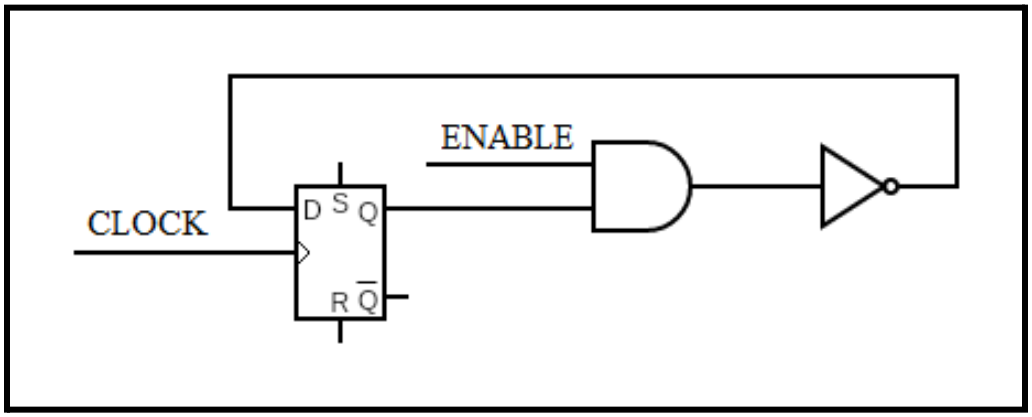
Note: Here, the AND, and NOT gates make up the Ring Oscillator framework. The input sysclk is the 125mHz clock that comes as a standard input on this line of FPGAs, sw0 is a physical switch on the FPGA board that enables the RO when flipped, and sw1 is another physical switch enables the counter and triggers the ILA to begin data collection when flipped.

III: The Power Virus

The next step was creating a power virus to test the framework on. Our power virus simply consisted of a D-Flip-Flop (DFF), an AND gate, and a NOT gate (see Figure 4). A DFF module takes a binary input (D) and a clock signal (here, we again use the 125mHz system clock), and changes the output (Q) to match the input at every positive edge of the input clock (note, a TFF is a simplified version of a DFF, which simply switches the output on every oscillation instead of matching it to an input). When enabled, the not gate reverses this output and sends the inverse value back to the input of the DFF. Thus, when enabled, this power virus continuously oscillates between 0 and 1 on each cycle of the system clock. We used this design because it was easily replicable, and simple to implement.

Figure 4

A schematic of the Power Virus unit with Enable and Clock wires labeled



IV: The Experimental Setup

The last step was running the power trace system in conjunction with different amounts of active power viruses and recording the dips in frequency of the ring oscillator. We enabled power viruses in multiples of 2000, and recorded the counter value after 1024 oscillations of the 125mHz clock. From here, we were able to find the frequency (in Hz) by dividing the clock value by 1024 and then multiplying by 125,000,000. We then plotted these frequencies against the number of active viruses to define a relationship between the two.

RESULTS

I: Frequency Calculation Method

All frequency measurements are found with the following equation:

$$F = C * 125,000,000/1024$$

II: Frequency VS # of Idle Power Viruses

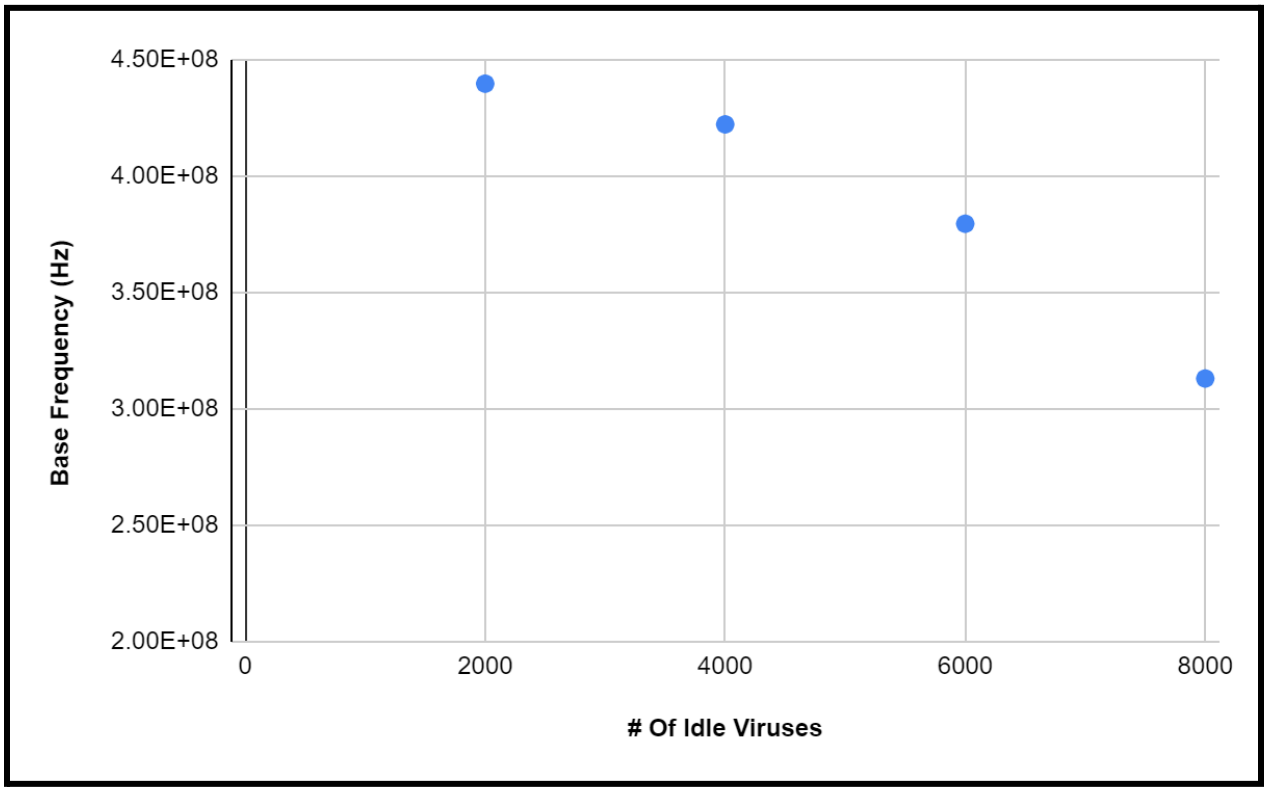
Below are the average base frequencies of the ring oscillator with different amounts of idle (not running) power viruses present within the FPGA fabric. Despite not being on, there is a relationship between simply placing more logic on the FPGA and the frequency of the RO. Figure 4 displays the average frequency values and Figure 5 plots them against the # of idle viruses present:

Figure 5

# Of Idle Power Viruses Present	AVG Base Frequency of Ring Oscillator (Hz)
2000	4.40E+08
4000	4.22E+08
6000	3.80E+08
8000	3.13E+08

Figure 6

Base Ring Oscillator Frequency VS # of Idle Power Viruses Present



III: Frequency VS # Of Active Power Viruses

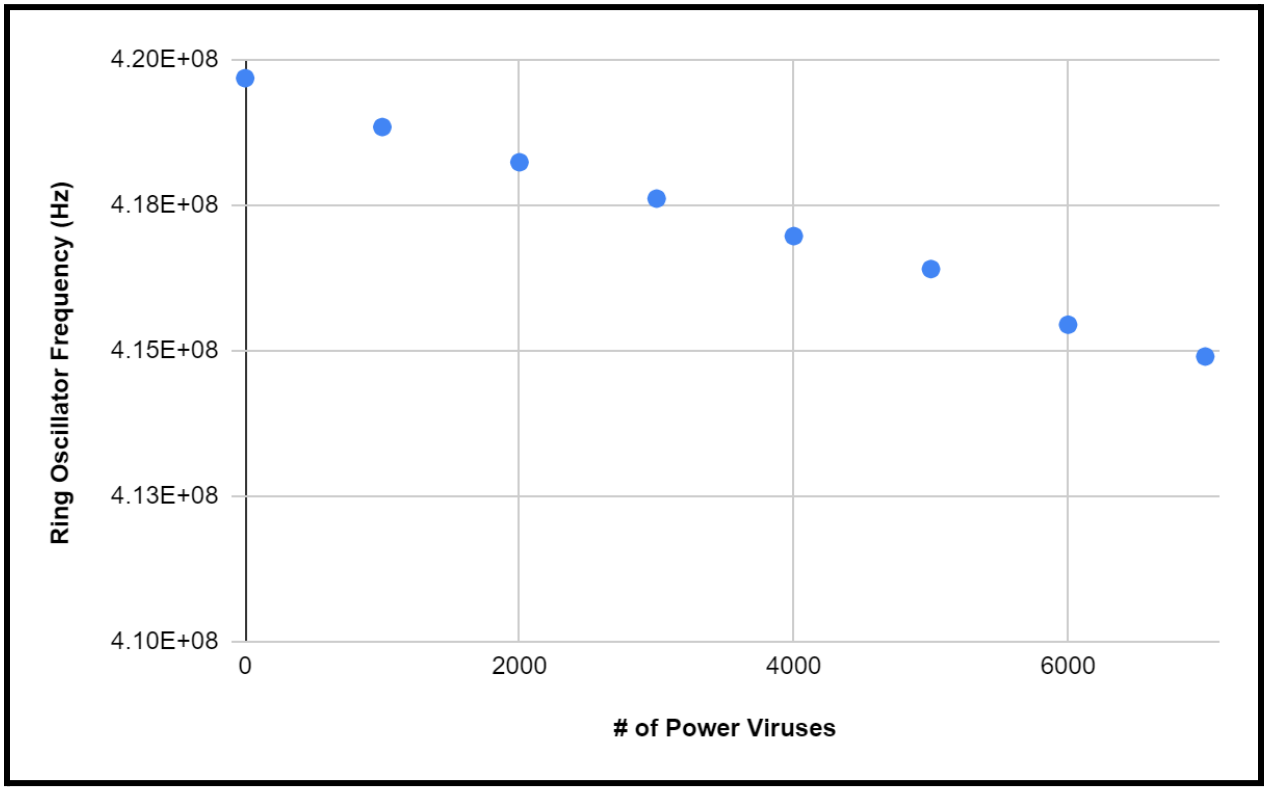
Below are the average frequencies of the ring oscillator with different amounts of active power viruses in the FPGA fabric. All 8000 instances are present and are activated in groups of 2000. Figure 6 displays the average frequency values and Figure 7 plots them against the # of active viruses:

Figure 7

# Of Active Power Viruses	Ring Oscillator Frequency (Hz)
0	4.1969E+08
1000	4.1885E+08
2000	4.1824E+08
3000	4.1762E+08
4000	4.1697E+08
5000	4.1641E+08
6000	4.1545E+08
7000	4.1490E+08

Figure 8

Ring Oscillator Frequency VS # of Active Power Viruses



DISCUSSION

We had hypothesized that we would find a relationship between the frequency of the ring oscillator and the magnitude of the synthetic load, as this had been proven in the past on other boards. We were surprised at the precision of the measurements, and that adding more idle viruses also decreased the frequency. This is not a problem for RO based side channel attacks, however, because a baseline frequency can be found by examining the values before and after the cipher completes its encryption process.

Over large sample sizes, a relationship can be established between average frequency and power load on the FPGA. However, ciphers do not steadily consume the same amount of power over a long period of time. In order to be used in a TASCA, this framework must be able to consistently provide reliable and precise frequency measures. The results of this study are promising, as they show a clear relationship between the two, and because the measurements are precise, and there is not much deviation from sample to sample under a given load.

Our next test would be to take shorter and shorter sampling periods, to more closely emulate the sampling speed of a framework used for TASCA. We will be able to find the point at which the resolution is no longer high enough for a difference to be discerned between different loads.

ACKNOWLEDGEMENTS

I would like to thank my mentor, Professor Laurent Michel, for making this project possible. The impact of the expertise he provided cannot be quantified, and I would not have been this successful without his generous time and dedication.

I would also like to thank Ms. Pintavalle, and Mr. Falcigno for their support through the Glastonbury High School ARM program. They taught me how to present my work and make a compelling argument for its impact, a skill just as important as the research itself.

REFERENCES

Amazon EC2. (n.d.). Amazon. Retrieved April 23, 2023, from https://aws.amazon.com/pm/ec2/?trk=9cd376cd-1c18-46f2-9f75-0e1cdbca94c5&sc_channel=ps&ef_id=CjwKCAjwrpOiBhBVEiwA_473dDdGVB9RHmgQQe4TLH0xWNSt85EcBV78so6sxmNG7QcQiK2iChGasBoC9I4QAvD_BwE:G:s&s_kwcid=AL!4422!3!651751059342!p!!g!!amazon%20ec2%20virtual%20machine!19852662176!145019190617

Cloud Computing. (n.d.). AMD Xilinx. Retrieved April 23, 2023, from https://www.xilinx.com/applications/megatrends/cloud-computing.html

Field Programmable Gate Array (FPGA). (n.d.). AMD Xilinx. Retrieved April 23, 2023, from https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html

Liu, F., Cruz, W., Ma, C., Johnson, G., & Michel, L. (2017). A Tolerant Algebraic Side-Channel Attack on AES Using CP. *International Conference on Principles and Practice of Constraint Programming*, 189-205. https://doi.org/10.1007/978-3-319-66158-2_13

Rodriguez-Henriquez, F., Saqib, N. A., Perez, A. D., & Koc, C. K. (2007). Cryptographic algorithms on reconfigurable hardware. *Springer Science & Business Media*. https://books.google.com/books?hl=en&lr=&id=qIY9RTzemr8C&oi=fnd&pg=PR13&dq=Cryptographic+Algorithms+on+Reconfigurable+Hardware&ots=FYxGLCWXpC&sig=DxnAwqDv-V7Otmkq7U52zoarj64#v=onepage&q=Cryptographic%20Algorithms%20on%20Reconfigurable%20Hardware&f=false

Vivado. (n.d.). AMD Xilinx. Retrieved April 23, 2023, from https://www.xilinx.com/products/design-tools/vivado.html

Zhao, M., & Suh, G. E. (2018). FPGA-Based Remote Power Side-Channel Attacks. *2018 IEEE Symposium on Security and Privacy (SP)*. https://doi.org/10.1109/SP.2018.00049

Zybo Z7: Zynq-7000 ARM/FPGA SoC Development Board. (n.d.). Digilent. Retrieved April 24, 2023, from https://digilent.com/shop/zybo-z7-zynq-7000-arm-fpga-soc-development-board/