Remote Side-Channel Disassembly on Field-Programmable Gate Arrays

Brandon R. Baggett

REMOTE SIDE-CHANNEL DISASSEMBLY ON FIELD-PROGRAMMABLE GATE ARRAYS

A Dissertation

Submitted to the Graduate Faculty of the
University of South Alabama
in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

in

Computing

by
Brandon R. Baggett
B. S., University of South Alabama, 2019
December 2023

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CPA	Correlation Power Analysis
DPA	Differential Power Analysis
DUT	Device Under Test
EM	Electromagnetic
FPGA	Field-Programmable Gate Array
GPP	General Purpose Processor
IP	Intellectual Property
LUT	Look-Up Table
RISC	Reduced Instruction Set Architecture
RO	Ring Oscillator
SPA	Simple Power Analysis
TDC	Time Delay Counter

ABSTRACT

Baggett, Brandon, R., Ph.D., University of South Alabama, December 2023. Remote Side-Channel Disassembly on Field-Programmable Gate Arrays. Chair of Committee: Todd R. Andel, Ph.D.

Over the last two decades, side-channel vulnerabilities have shown to be a major threat to embedded devices. Most side-channel research has developed our understanding of the vulnerabilities to cryptographic devices due to their implementation and how we can protect them. However, side-channel leakage can yield useful information about many other processes that run on the device. One promising area that has received little attention is the side-channel leakage due to the execution of assembly instructions. There has been some work in this area that has demonstrated the idea's potential, but so far, this research has assumed the adversary has physical access to the device. In recent years, researchers have developed methods for remote side-channel attacks using power monitors implemented in reprogrammable hardware.

In this work, we test if similar power monitors are capable of disassembling code running on a general purpose processor on the same chip die as the reconfigurable hardware fabric. We train a sequence of decision tree classifiers to first predict which group an instruction belongs to, then its type, and finally, the individual instruction.

Our results demonstrate that our field-programmable gate array-based power monitors are correlated to the executed instructions and correctly classify individual instructions at about 13-15% depending on the clock rate. This is better than randomly guessing but far from being useful in practice. We also train coarse-grain models for classifying instructions based on their functional unit utilization. This approach improves the accuracy to about 20-30% depending on the clock rate and grouping used.

CHAPTER I

INTRODUCTION

New technologies are often adopted to address the limitations of existing systems. System designers must consider the vulnerabilities they may introduce when implementing these technologies. One emerging technology with great potential is hybrid general purpose processors (GPPs) packaged with field-programmable gate array (FPGA) fabric on the same chip die. These products are readily available from Intel and AMD. FPGAs are reprogrammable hardware in which the designer can implement arbitrary circuits. The FPGA fabric consists of many interconnected look-up tables (LUTs) that can be programmed with a bitstream to specify their input and output to simulate virtually any circuit design [1].

Thompson and Spanuth [2] argue that due to the rapid advancements in GPPs, specialized processors have not had the opportunity to emerge. As shown in Fig. 1(a), GPPs have been in a positive feedback loop where their rapid improvements led to more users buying new processors, which gives companies more money to continue improving their devices. However, as technological advancements slow, users are less inclined to upgrade their processors, which means less money for development, further slowing the technological advancements. This has created a negative feedback loop for GPPs (Fig. 1(b)), which creates a space for specialized processors to fill.

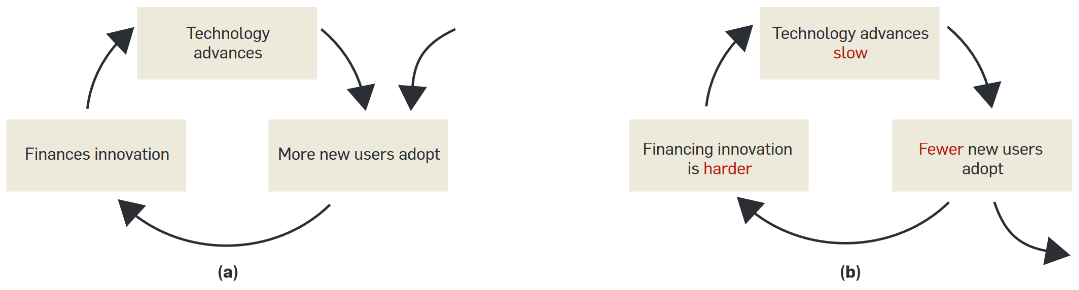


Figure 1. Feedback loops for general purpose processor advancements [2].

Creating specialized processors is expensive, and many applications do not have enough users to adopt the technology to justify the development cost. We believe that hybrid GPP/FPGA processors could address this limitation by allowing a single chip die to be used for many applications. The individual applications can benefit from the speedup offered by implementing their design in the FPGA fabric without going through the fabrication process. This concept is supported by Intel's acquisition of Altera in 2015 [3] and AMD's recent acquisition of Xilinx [4]. Both companies talk about how these hybrid devices will enable more flexible options for developers.

Despite the benefits that hybrid processors offer, it is crucial to understand the vulnerabilities these devices introduce so that we can protect against them. A major vector for attacking FPGAs is their bitstream. Potential vulnerabilities exist in the entire bitstream lifecycle from generation to end-of-life [5]. If attackers compromise the bitstream, they can upload any design and compromise other processes running on the same chip. Researchers have shown that lightweight FPGA-based power monitors are capable of performing side-channel attacks [6]. A natural extension of this work would be to see what other information can be leaked through these on-chip power monitors.

In this research, we introduce a system and methodology to implement power monitors in the FPGA fabric and test if we can disassemble code running on a GPP on the same chip die using power side-channel data. Side-channel disassembly has shown to be viable [7]–[9]. However, these approaches have assumed the attacker has physical access to the device. The emergence of hybrid processors potentially introduces a new threat that system designers must address. That is, remote side-channel disassembly attacks. This research is primarily focused on determining the feasibility of the attack, so our approach has a few limitations that should be considered. First, we are using a softcore GPP, which operates at a much lower clock rate than a hardcore processor. Additionally, we are only considering the executed instructions and not the registers. In Chapter V, we outline potential solutions to address these issues if this design proves feasible.

Our results show that there exists a correlation between our power monitors' data and the executed instructions. Our models' accuracies range from approximately 13% to 30%. Even in the best-case scenario, where we use a coarse-grained model that groups similar instructions based on their suspected functional unit utilization, our approach is still far from being a salient threat. However, this does not mean the threat is nonexistent. As FPGA power monitors inevitably continue to improve, this threat will become more feasible.

The rest of the paper is organized as follows. In Chapter II, we discuss previous side-channel research and various FPGA power monitors. In Chapter III, we present our research objectives, proposed system, and methodology for evaluating FPGA-based side-channel disassembly. In Chapter IV, we discuss the results of our tests. Finally, in Chapter V, we conclude this dissertation by discussing the implications of our research and outlining potential future works.

CHAPTER II

BACKGROUND

Over the last few decades, side-channel research has received a lot of attention from the academic and professional community. This is primarily due to the threat they pose to cryptographic hardware. Although cryptographic side-channels are among the most promising applications of side-channel research, we should not limit our understanding to just cryptographic side-channels. Other promising side-channel applications have emerged, such as malware detection and code disassembly. These areas have received varying levels of attention from the academic community, with side-channel code disassembly receiving little attention. However, due to recent advancements in remote side-channel attacks, we believe there is new potential for side-channel disassembly.

2.1 Side-Channels

A side-channel is any unintended leakage of information. This can be anything from using the flashing lights in someone's window to detect that they are home watching TV to measuring the electromagnetic (EM) emanations from a monitor to reconstruct the image [10]. However, the most commonly discussed side-channels are those that leak information about secret cryptographic keys. Any measurable physical property that varies based on the secret key can potentially be used in a side-channel attack to break a cryptographic implementation. The most exploited side-channels are power and EM.

2.1.1 Cryptographic Side-Channels

Fig. 2 (based on the work by Zhou and Feng [11]) gives a basic overview of cryptographic side-channel attacks. In the traditional understanding of cryptography, a message is passed through an encryption algorithm with a key to output a ciphertext. If anyone intercepts this ciphertext, it should be indistinguishable from random data unless they have the corresponding key to decrypt the message. However, it was shown in the late 1990s that there are measurable physical changes in the device's behavior based on the secret key that can be used in a side-channel attack to break the encryption [12].

The most commonly discussed side-channels are power and EM. Fig. 3 [13] shows the basic steps for performing power and EM side-channel attacks. First, the attacker calculates an intermediate step of the algorithm (such as an S-box output in DES) for multiple messages and key candidates. These key candidates are smaller than the full key since the attacker only needs to iterate over the key bits that are directly affecting the targeted intermediate value (such as the 6 bits going into a DES S-box). Then, the attacker uses a power model, such as Hamming Weight, to approximate the consumed power for each calculated intermediate value. Next, the attacker measures the side-channel leakage from the device under test (DUT) when encrypting the same messages from step 1.

The next step is to use statistical analysis to compare the results from the physical device against the hypothetical power consumptions. This will yield the final results matrix that shows which key guess best matches the true power measurements. The two most popular statistical methods are differential power analysis (DPA) and correlation power analysis (CPA). The underlying math used in these approaches differ, but in both cases, the attacker is looking for a result that is significantly better than the others. Due to noise, the results will never be perfect, but as the attacker increases the number of traces, the correct result will start to differentiate itself from the others.

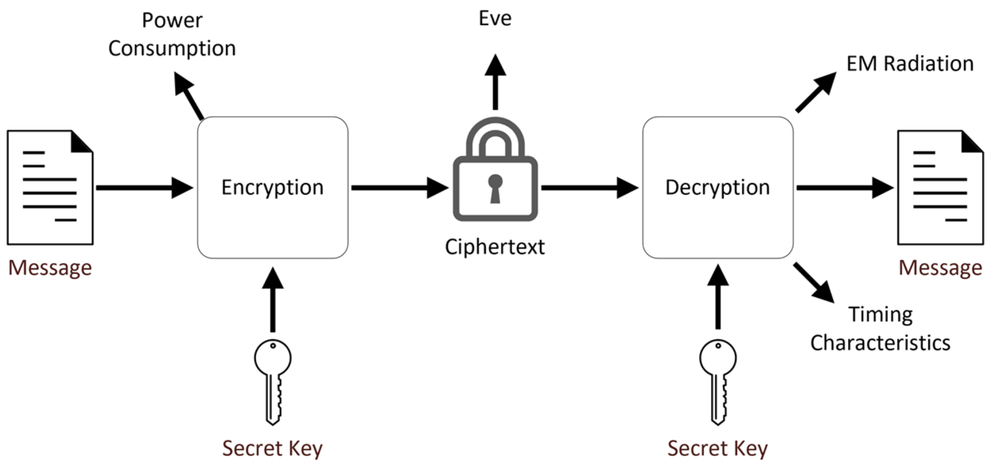


Figure 2. Side-channel attack model [11].

Functionally, power and EM attacks work very similarly. However, they differ in how they collect their side-channel data [13], [14]. Power attacks monitor the power consumption of the entire device. In EM attacks, the attacker scans each part of the device to look for hotspots and then measures the localized EM emissions from the region with the most activity. EM attacks require more sophisticated equipment, but their localized nature allows them to isolate the collection from most of the noise from other activities on the chip.

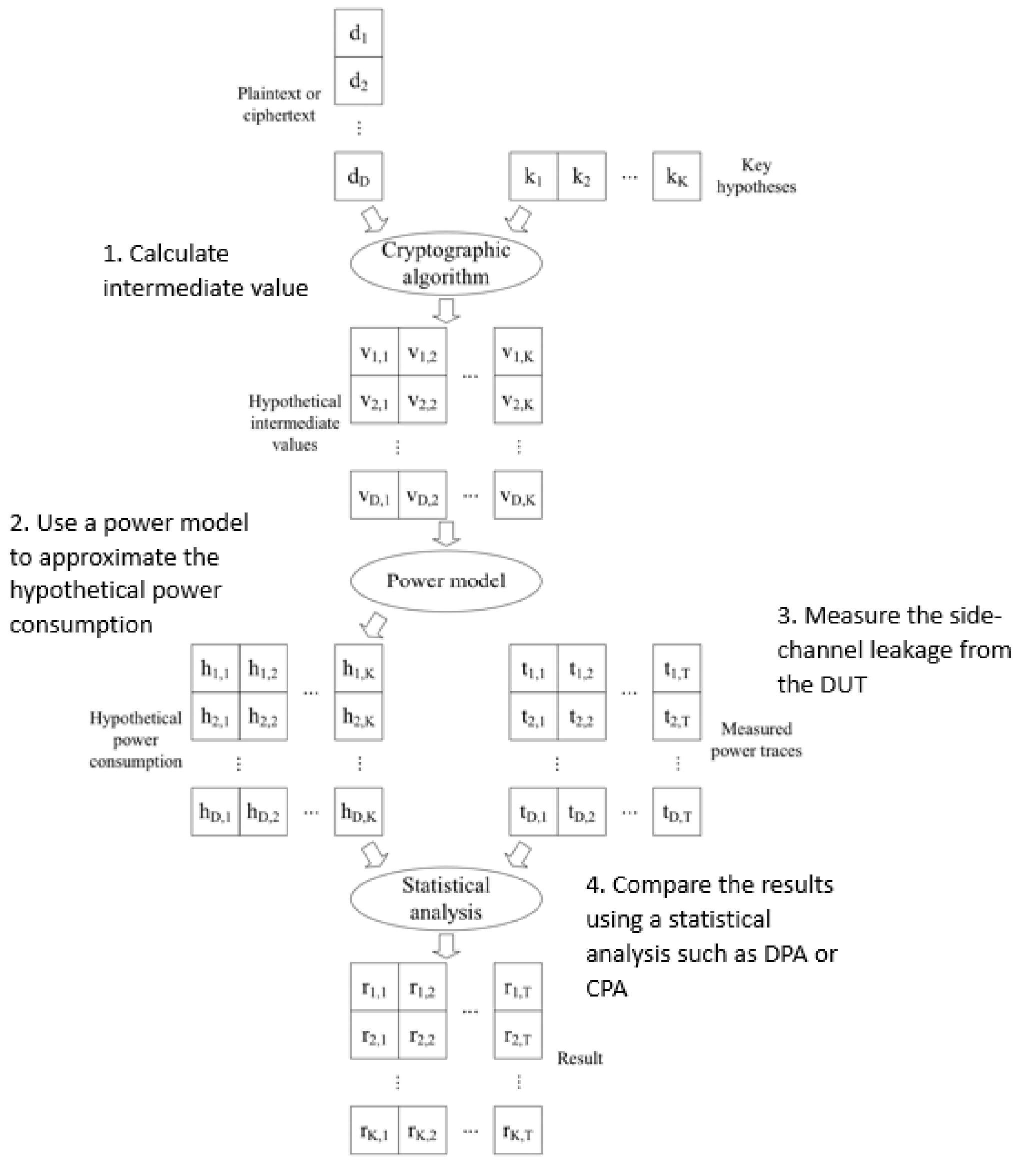

Plaintext or ciphertext
d_1
d_2
d_D
k_1
k_2
k_K
Key hypotheses
1. Calculate intermediate value
Cryptographic algorithm
Hypothetical intermediate values
2. Use a power model to approximate the hypothetical power consumption
Power model
3. Measure the side-channel leakage from the DUT
Hypothetical power consumption
Measured power traces
Statistical analysis
4. Compare the results using a statistical analysis such as DPA or CPA
Result

Figure 3. Side-channel analysis steps [13].

2.1.2 Side-Channel Malware Detection

Another application for side-channel research is malware detection. Since side-channel leakage is due to the physical properties of the device, in theory, malware cannot conceal itself from a side-channel-based detection mechanism. This is a desirable option for malware detection since the detection system can be outside the influence of the infected system, so the malware cannot interfere with it [15].

Clark et al. [16] introduce WattsUpDoc, a system for detecting malware on medical devices using power side-channels. Using system-wide power measurements, their approach detected about 94% of known malware and 85% of zero-day attacks. They used various binary classification algorithms but had the highest results with the 3-nearest neighbor, multilayer perceptron, and random forest classifiers. Dawson et al. [15] used a nonlinear phase-space algorithm to detect rootkits based on power measurements.

Malware can also be detected using EM side-channels. Khan et al. [17] demonstrated that they could detect malware on embedded devices based on their EM side-channel leakage. They used a neural network to detect when the device was behaving abnormally with an accuracy of nearly 100% for DDoS attacks, ransomware, and code modifications. EM side-channels are better at detecting localized variations than power side-channels. Other researchers [18] used this property to detect hardware Trojans in FPGA-based AES and RSA cores. Using a k-means clustering algorithm, they accurately classified about 80% of hardware Trojans.

There have been other proposed side-channel-based malware detection systems that work similarly. Although these goals are different from breaking cryptographic implementations, they further demonstrate the many uses of side-channel research. Since all applications run on hardware, there will be physical byproducts leaking information about the application. We can try to minimize the variance between different applications and data, but unless we can completely remove these changes, a dedicated adversary can exploit this leakage.

2.1.3 Side-Channel Disassembly

One of the earliest forms of side-channel analysis was simple power analysis (SPA) [12]. There are multiple versions of this attack that leak different information about the encryption algorithm, but the SPA attack on RSA is of particular interest. In a naïve implementation of RSA, the algorithm iterates through each bit of the secret key. For every loop iteration, a square operation is performed. When there is a 1-bit in the secret key, a multiplication operation is also performed. Since multiplications consume more power than squares, an attacker can easily extract the key after a single encryption, as shown in Fig. 4 [19].

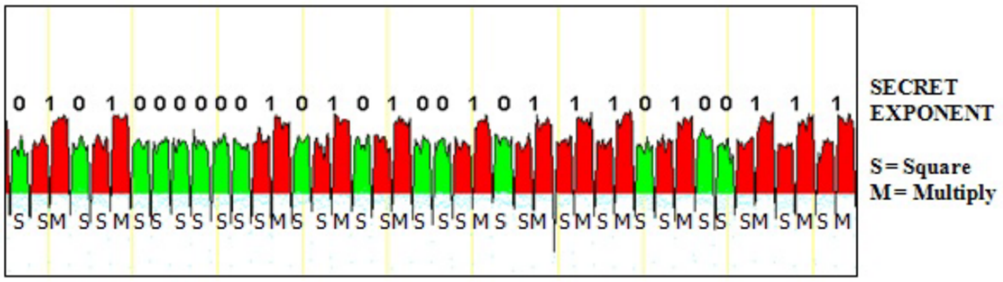


Figure 4. Simple power analysis attack on RSA [19].

This SPA attack on RSA is still a cryptographic side-channel attack, but it reinforces the idea behind side-channel disassembly by demonstrating that different assembly instructions have measurable power variations that are exploitable in side-channel analysis. Although side-channel disassembly is similar to the side-channel malware detection discussed previously, it has a few key distinctions that will make it more challenging.

In malware detection, the goal is to detect suspicious behavior, whereas in side-channel disassembly, the goal is to classify each of the assembly instructions. This introduces two main obstacles. The binary classification of malware detection means that the analysis can look at the power consumption trend and compare it against the baseline. On the other hand, disassembly needs to accurately classify the instruction based on a single clock cycle's power trace. Not only is this a shorter window for analysis, but the number of classes is much higher.

Until recently, there has been little research on side-channel disassembly despite its potential. The earliest work was by Eisenbarth et al. [7], where they used power side-channels to reconstruct the code running on a microcontroller. Their work empirically demonstrated the feasibility of the idea. They used a frequency analysis to model the distribution of instructions and common instruction tuples. They then made a template for each instruction and built a Markov model. They were able to correctly classify about 70% of the instructions when they knew the instructions' distribution. When evaluating other applications, their classification accuracy dropped to approximately 58%. Although these results are not great, they provide a good starting point by showing that side-channel data leaks information about the executed assembly code.

In the last two years, there has been an influx of side-channel disassembly research. Although it is still a young research area, this further demonstrates its potential. However, there is still more work to be done. Most of these studies have had a high classification accuracy (>90%), but so far, no work has demonstrated the work is feasible in practice.

Many of these works have begun exploring the potential of using EM side-channels for disassembly [9], [20]–[23]. Although EM analyses are more sophisticated, their localized nature is beneficial since they can place probes over each functional unit. However, EM side-channels require more sophisticated setups than power analysis attacks. As with most decisions in computing, there are trade-offs that prevent one approach from being the objective best.

Another trend that has emerged is the use of hierarchical models. The first was by Park et al. [8], in which the authors used a hierarchical approach to first classify a power trace into different groups based on the operands that it used. Next, the instruction was classified from within that group, and finally, the operands were classified. Using this approach, they were able to achieve an accuracy of 99%. Then, Glamočanin et al. [24] used a 3-tiered model where the first layer classifies the instruction's group based on the structure of the instruction. The second layer classifies instructions within the group, and the final layer predicts the bits of the operands of the instruction. A bit-level approach was also used by Cristiani et al. [20], where they achieved an accuracy of ~99% in predicting the instruction and its operands based on EM side-channel data. This was further corroborated by Lie et al. [25], [26] where they found that the processed data (i.e., the bits in the instructions and operands) had a larger impact on the power consumption than the instruction did. These works also highlight one of the major limitations of the current state of side-channel disassembly research. Most research is only concerned with disassembling the instructions, but not the operands.

Most of the previous side-channel disassembly research has only dealt with disassembling the instructions. This intuitively makes sense for a starting point since it stands to reason that the operations would have a larger impact on the side-channel leakage than the operands. In addition to the papers mentioned previously, Maillard et al. [23] trained models for 3 levels of granularity. Their most coarse-grained model classified EM side-channel data into the functional unit it used. Their second model aimed to reconstruct the opcode, and their fine-grained model aimed to reconstruct the entire instruction with its operands. Their coarse grained-models had near 100% accuracy, but the fine-grained model only achieved an accuracy of 53-64% (some bits had a higher accuracy than others). This is consistent with the work bt Krishnankutty et al. [27] where the authors used two granularities for training. Their coarse-grained model grouped instructions based on the functional units they used and achieved an accuracy of 92-100% depending on the target processor. Their fine-grained model, which predicted the instruction (but not the operands), achieved an accuracy of 90-91%.

Some researchers are using a side-channel disassembly-like approach to detect the integrity of software [25], [26]. These authors propose "side-channel programming," in which parts of a program are rewritten so that modifications can be detected using the side-channel leakage of individual instructions. This work is similar to side-channel disassembly but differs since their goal is to detect changes to the software and not reconstruct the program itself.

Until now, one area that has remained unaddressed in this field is that side-channel attacks traditionally require physical access to the device. If an attacker has direct access, then there are simpler methods to extract the code than performing a side-channel attack. However, some remote side-channel attacks have emerged that could potentially address this limitation, allowing for remote code disassembly.

2.2 Remote Side-Channel Attacks

Originally, the main limitation of side-channel attacks was that they required physical access to the device. Over the years, techniques for remote side-channel attacks have been developed. Timing variations were especially conducive to remote side-channel monitoring, with the timing differences due to cache hits/misses being one of the most targeted vulnerabilities. However, FPGAs have introduced a new avenue for remote side-channel attacks due to their ability to implement on-chip power monitors.

2.2.1 Timing Side-Channels

Timing side-channel attacks rely on the slight variations in timing characteristics due to different program inputs. This concept was introduced by Kocher [28], where he demonstrated that the modular exponentiation algorithm used in many RSA and Diffie-Hellman implementations is vulnerable to a timing attack. Similar to the SPA attack described in Fig. 4, this attack exploits the differences between a square and multiplication operation in hardware. The authors point out that other timing variations, such as those due to cache hits, processor instructions, branching statements, or performance optimizations, could also lead to timing side-channel leakage.

Since then, more sophisticated versions of this attack have been developed. In the Flush and Reload attack [29], the attacker infers information about a targeted process that is running shared code. Many processes use the same code (i.e., two programs accessing a standard library). This shared memory is read-only, so the two processes should be isolated in theory. However, the timing differences between cache hits and misses can leak information to the attacker. In this attack, the attacking process empties (flushes) the cache and waits a predetermined amount of time before running the shared code. If the access happens relatively quickly, that means the code has been reloaded into the cache, which tells the attacker that the targeted process has used that code. If there are segments of code that only run based on secret data, then the attacker can use these timing variations to infer the secret value.

A similar attack is the Prime and Probe attack [30]. The main difference in this attack is that the attacker loads their own data to completely fill the cache, waits for the targeted process to run, and then reloads all the data into the cache again. If the data loads quickly, then the targeted process did not access any addresses that would load into that cache line. This can leak information about the structure of the data access of the targeted process. If a secret value determines the data access, then the attacker can use this side-channel to infer that secret value.

Spectre [31] and Meltdown [32] are other forms of timing side-channel attacks. These attacks are functionally very similar. They try to access protected memory with a conditional statement; if the statement is true, then the process will load some arbitrary value to memory. Functionally this code cannot work since the processor will not allow the process to access the protected memory. However, these protections are not enforced during speculative execution. The attacking process can then attempt to load in the same data that would have been loaded if the conditional statement was run. If the data loads relatively quickly, then it was already loaded into the cache, which implies that it was accessed during the speculative execution stage for the blocked conditional statement. An attacker can use this to deduce a password by trying each character for the nth char of a password until the arbitrary data has been loaded into memory. Similar methods can leak information about other protected data.

2.2.2 Power Side-Channels

Although, power side-channel attacks originally assumed the attacker had physical access to the device. Zhao and Suh [6] evaluated the effectiveness of FPGA-based ring oscillators (RO) as power monitors for remote side-channel attacks (ROs will be discussed in greater detail in section 2.3 below). In their first attack, they implemented a naïve implementation of RSA (similar to the one discussed in Fig. 4) in the FPGA fabric. As the power consumption of the device increases, the oscillation rate will decrease. Fig. 5 demonstrates how they used this setup to break RSA. They summed the RO counts for a given period, and a lower count corresponds to higher power consumption. Since a 1-bit in the secret key causes the extra multiply operation, an attacker can deduce the secret key based on the RO outputs. However, due to noise on the device, it would sometimes take additional traces to find the key. If they assumed that the attacker had complete control over the placement and routing of the power monitors, then they could break the key in at most five traces. In the worst-case scenario, if the attackers had no control over the placement and routing, it could take up to 22 traces to break the key.

Their research focused on SPA, but other works have extended this idea for more sophisticated attacks. Schellenberg et al. [33] used delay lines (also referred to as time delay counters (TDC)) to attack an AES core implemented in the FPGA fabric. Gravellier et al. [34] proposed a modified RO design that addresses some of the limitations of RO that led many designers to use TDC power monitors instead. Their new RO design can perform a CPA attack on an AES core running in the same FPGA fabric. Since the quality of FPGA-based power monitors continues to improve, we believe this approach is ideal for remote side-channel disassembly.

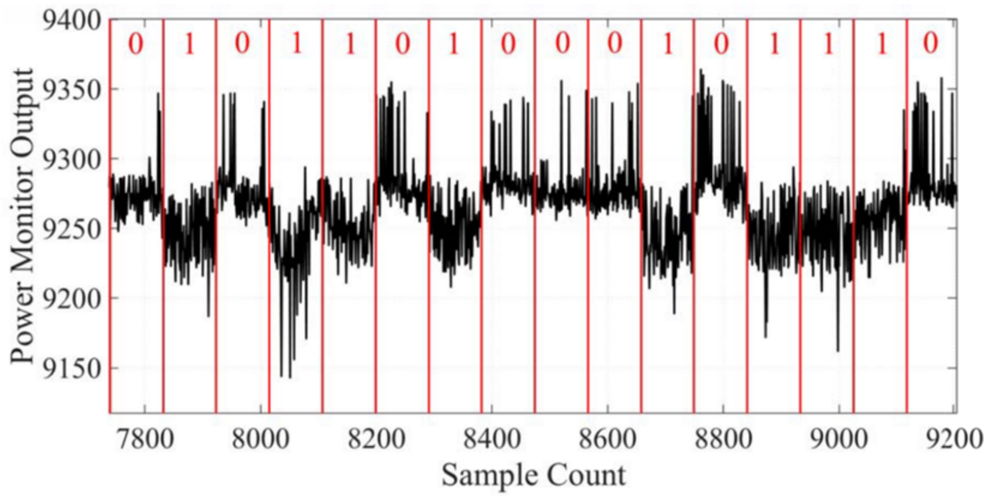


Figure 5. SPA power trace from FPGA ring oscillators [6].

2.3 FPGA-Based Power Monitors

Although previous studies have focused on cryptographic side-channels, these works demonstrated the feasibility of FPGA-based power monitors. A natural extension of these works is to see what other information can be detected through these on-chip power monitors. In particular, side-channel disassembly would benefit from the remote possibilities of on-chip power monitors.

Both ROs (Fig. 6) and TDCs (Fig. 7) work on the same fundamental principle; that is, the device's power consumption affects the propagation delay of its signals. ROs work by having an odd number of inverter gates in series that get fed back into an AND gate with an enable signal. Every time the signal propagates through the inverter chain, its output will switch between a 0 and a 1. At the end of the inverter chain is a counter for how many times the RO has gone through a complete cycle. As the device consumes more power, the oscillation rate will decrease, so the counter will be lower for the given period. This means that a lower count corresponds to higher power consumption.

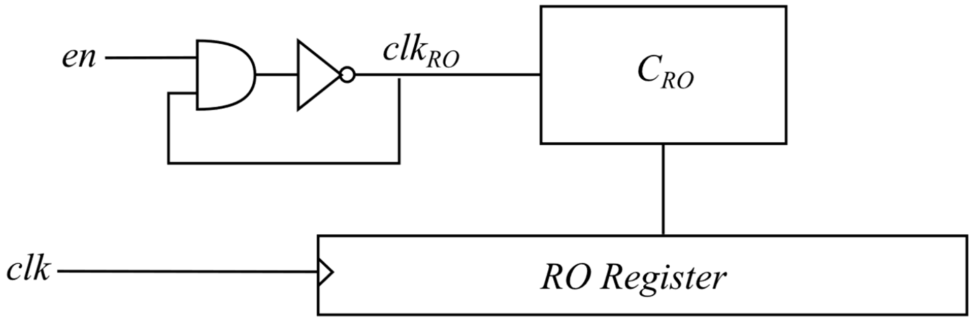


Figure 6. Ring Oscillator based power monitor circuit diagram [34].

TDCs work similarly, but instead of measuring the oscillation count, they measure the propagation depth through a series of inverters. In TDCs, the clock initializes the inverter chain. The signal will propagate through the chain based on the power consumption of the device. At the next clock cycle, the TDC register records how far down the inverter chain the signal propagated to determine the power consumption of the device.

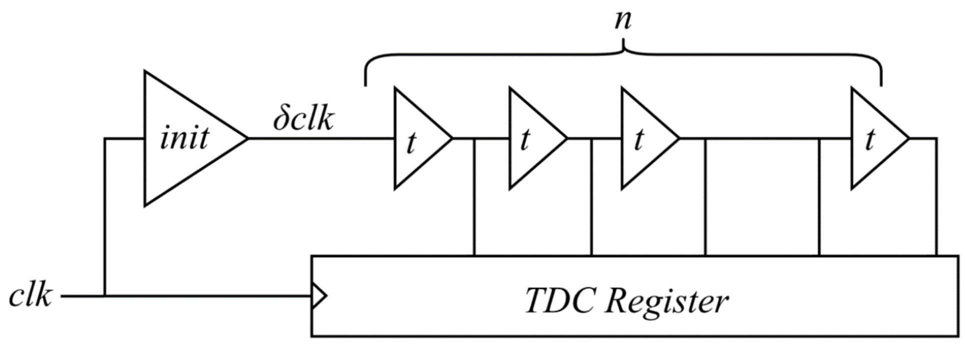


Figure 7. Time Delay Counter based power monitor circuit diagram [34].

Both TDCs and ROs can measure a device's power, but neither is objectively superior since they each have their tradeoffs. ROs are a simpler design and can measure at theoretically any frequency, but as their sampling rate increases, their accuracy decreases. TDCs do not have this accuracy/frequency tradeoff, but they cannot sample at a higher rate than the clock. Additionally, TDCs require more resources and are more susceptible to placement and routing than ROs.

CHAPTER III

METHODOLOGY

The goal of this research is to develop a system for remote power monitors on FPGAs using ring oscillators and to determine if it can be used to disassemble code running on a softcore processor on the same chip die. This chapter proposes a methodology for completing these objectives and gives an overview of the required resources, system design, and evaluation metrics.

3.1 Research Objective

Previous research has shown that different assembly instructions have measurable power variations that are exploitable by side-channel analysis. However, these studies all assumed the adversary has physical access to the device, which limits the threat of this attack. In recent years, researchers have demonstrated that FPGAs allow for remote side-channel attacks since an adversary can modify the bitstream without physical access to the device. This development introduces new threats that must be considered when designing secure systems. Up to this point, these attacks have focused on cryptographic side-channels. Since remote power monitors can measure and exploit the minute power variations due to the secret key's bits, then it stands to reason that they can measure other side-channel leakages on the device. In particular, we suspect they can detect power variations caused by different assembly instructions.

This research aims to determine if side-channel data collected from FPGA-based power monitors can be used to disassemble code running on a GPP on the same chip die. We propose a system that uses four RO-based power monitors placed around the perimeter of a softcore Microblaze processor running in the FPGA fabric on a Diligent Genesys 2 Kintex-7 FPGA Development Board [35]. There is another Microblaze processor on the same chip die that is responsible for collecting the RO power data and scheduling the testing programs to run on the target processor.

Since our goal is to determine if programs can be disassembled using FPGA-based ROs, we perform the classification training and testing on our desktop instead of the development board. If the approach proves feasible, then future works will involve refining the technique to be applicable in real-world scenarios. The controller dynamically creates the training code during the data collection stage. Table 1 outlines the major objectives of this research.

For this project, we evaluate the feasibility of remote side-channel disassembly. Due to this scope, there are some limitations of the work that should be considered. The first is that we are using a softcore processor to run the target assembly code. We chose a softcore processor since it gives us more control when configuring the processor and minimizes the background signal noise that would be generated by a hardcore GPP. A more practical implementation of this attack would be a hybrid device that uses a hardcore GPP running with FPGA fabric on the same chip die. For these devices, the hardcore processors generally run at a much higher clock rate than the FPGA fabric. This would increase the complexity of attacks when using either TDC or RO-based power monitors. Another major limitation is that we are only considering the executed instruction and not the registers.

Table 1. Research objectives for side-channel disassembly.

Objectives	Description
1	**Configure hardware design**
1.1	Configure Target processor
1.2	Configure custom IP
1.3	Configure Controller
2	**Create embedded software**
2.1	Create Target C code
2.2	Create Controler C code
3	**Train the machine learning algorithms**
3.1	Create pre-processing data script
3.2	Train the models
3.3	Evaluate the models' accuracy

If this study shows that the approach is feasible, then both of these limitations could potentially be addressed by using more robust power monitors and machine learning techniques that account for the system's specifics. For example, we could try accounting for instruction distributions or microarchitectural differences, such as different length pipelines.

3.2 Design Overview

Our design is implemented on a Genesys2 Kintex-7 FPGA [35] using the Xilinx Vivado Design Suite 2018.2. The target is the Xilinx Microblaze 32-bit softcore processor. Specifically, we are targeting Vivado's "Real-time Preset" Microblaze configuration for the attacked processor. Our implementation of the Microblaze uses a 3-stage pipeline (Fetch, Decode, and Execute), has 8kB for both the instruction and data caches, and keeps all other settings at their default values.

We use a second Microblaze as a controller for reading the power monitor counts, writing them to memory, and scheduling the code to run on the target processor. The controller is responsible for generating the training data, scheduling when the target processor runs the code, and recording the values from the power monitors. The training and evaluation of the classification models are performed off-device. The controller is another 32-bit Microblaze softcore processor. It is configured using Vivado's "Microcontroller" preset with the optimization set to "Performance." Using placement and routing constraints, we place the ROs, Target processor, and Controller as shown in Fig. 8 and let Vivado manage the placement of the rest of the Intellectual Property (IP).

We chose ROs for our power monitors since they have sufficient resolution for performing side-channel attacks, so they should also work for code disassembly. Additionally, they require less stringent placement and routing constraints when compared to TDC power monitors, which is a more realistic representation of an attacker's capabilities. Although ROs have lower accuracy at higher clock rates, we do not anticipate this being an issue for our design since we are considering a softcore processor that runs at the same clock rate as the power monitors.

The system's sequence is shown in Fig. 9. The ROs continuously send their data to the controller. First, the controller generates the next code snippet. Next, it sends the program to a shared BRAM memory region, and then interrupts the Target. Next, the target freezes the Controller Clock, unfreezes the RO, and then waits for 100 μseconds to clear out the RO counts and let the heat from the Controller dissipate. Next, the target runs the code snippet. It then freezes the RO so that their data is not replaced before their values are saved. Next the Target unfreezes the Controller Clock. Then,the controller sends the data to the main computer. Finally, the system loops back to step 1 until all of the data is collected.

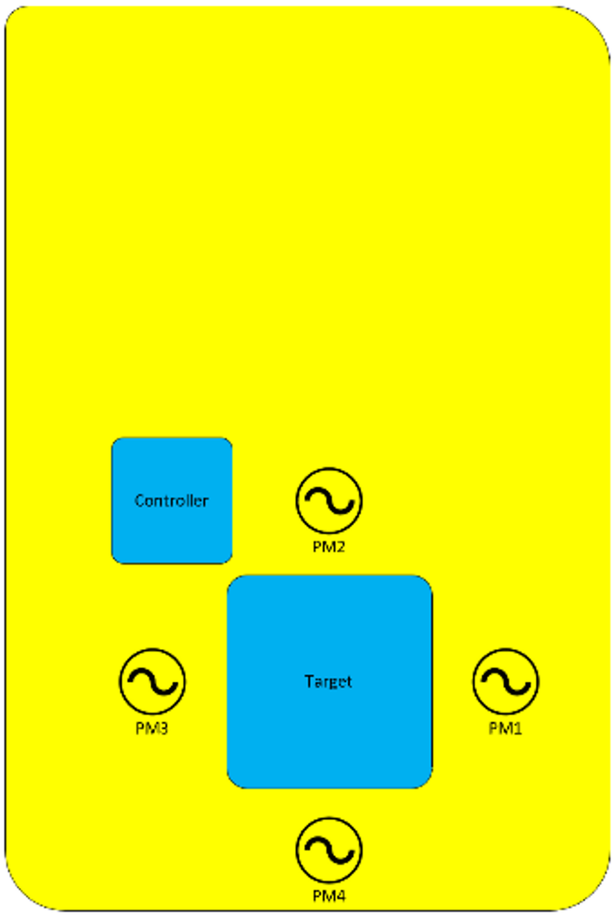


Figure 8. Processors and power monitors placement diagram.

In a real-world scenario, the target processor would not aid in data collection. Our approach is similar to cryptographic side-channel research, where a trigger is used to help with the data collection. By using this approach, we know that our power data corresponds to the training data and reduces the amount of data preprocessing required when training our models.

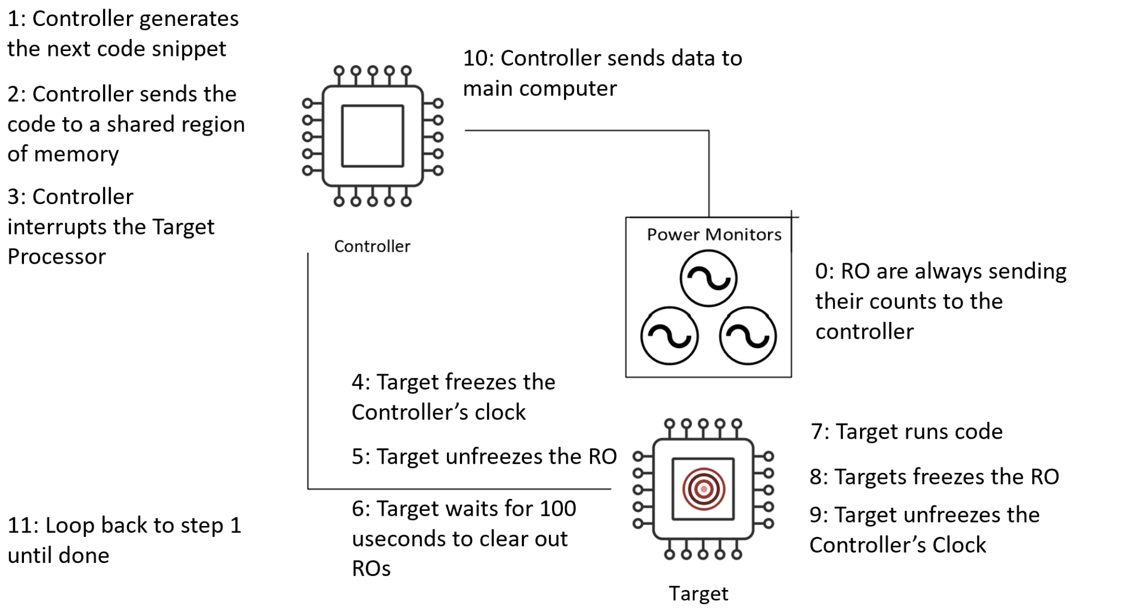

1: Controller generates the next code snippet
2: Controller sends the code to a shared region of memory
3: Controller interrupts the Target Processor
Controller
10: Controller sends data to main computer
Power Monitors
0: RO are always sending their counts to the controller
4: Target freezes the Controller's clock
5: Target unfreezes the RO
6: Target waits for 100 useconds to clear out ROs
7: Target runs code
8: Targets freezes the RO
9: Target unfreezes the Controller's Clock
Target
11: Loop back to step 1 until done

Figure 9. Sequence of events for side-channel data collection.

To facilitate this design, we created four custom AXI peripherals. The first is a RO, but unlike other works where the ROs were either measuring at clock rates faster than the targeted circuit or were measuring for entire program executions, our ROs need clock cycle accuracy. To account for this requirement, we modified the design of the ROs used in [36]. Fig. 10 shows this design. The actual RO is in the top left corner, and its output is fed into two 16-bit counters. We use a D flip flop that switches states every clock cycle as a counter selector so that while one counter is recording the output of the RO, the other is put into a reset state. This dual-counter setup was chosen so that we do not lose any data while a counter is resetting. Finally, the outputs are OR'd together and fed into a chain of 50 16-bit D flip flops acting as an output buffer (only 1 flip flop is shown for simplicity). When the counters are in their reset state, their output is set to 0, so the inactive counter does not have an effect on the OR gate.

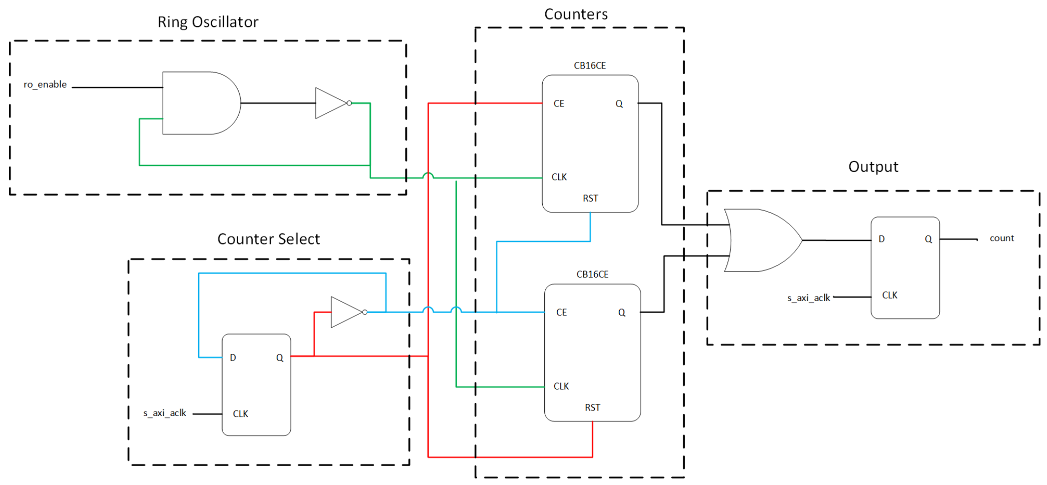
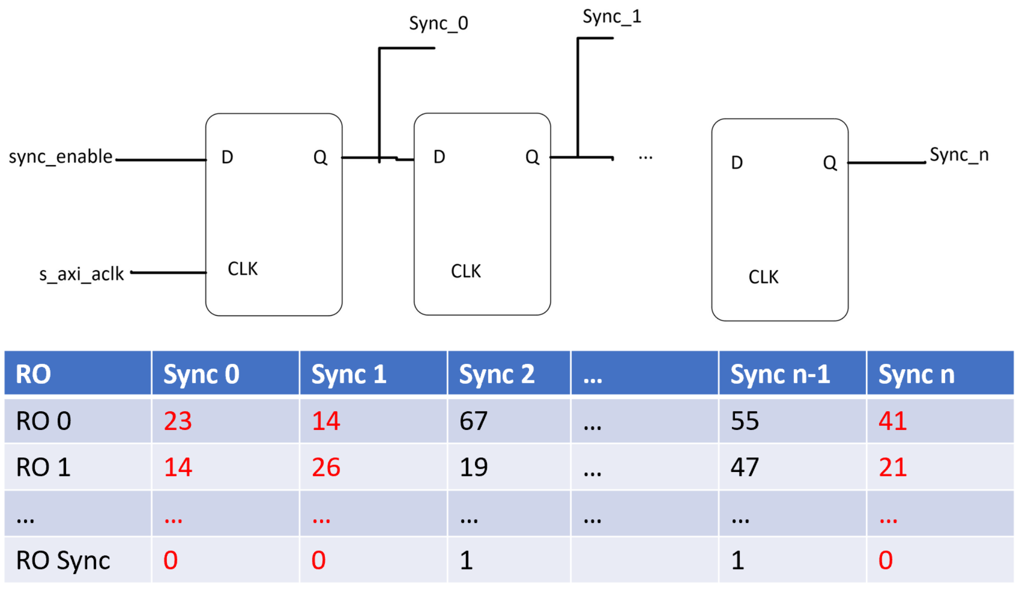


Figure 10. Custom ring oscillator.

We made a gated clock IP that is a simple AND gate combining an enable signal with the clock signal so that we can effectively freeze certain IP in their current state by disabling the gated clock. We used two of these in our design. The first is to freeze the controller while the target processor is running the code snippets so it does not affect the power consumption. The second is for the ring oscillators so that we can hold their output buffer values while the controller sends them to the main computer.

We also created an Interrupt Scheduler IP, which is two AXI writable D flip flops that are linked to the interrupt signals for each processor. Our design uses interrupt handlers as the handshaking mechanism for passing control between the processors. In the final design, only the Target interrupt is used, but both were used for testing purposes.

Fig. 11 shows the RO sync IP with some mock data. The RO sync IP is a buffer of 50 D flip flops. This IP is triggered by the target processor immediately before running the code snippets and then deactivated immediately following the code snippet's execution. The data from this IP signals which RO output values correspond to the code snippets running and which can be trimmed during the data pre-processing stage.



RO	Sync 0	Sync 1	Sync 2	...	Sync n-1	Sync n
RO 0	23	14	67	...	55	41
RO 1	14	26	19	...	47	21
...	...	...	...	...	...	...
RO Sync	0	0	1		1	0

Figure 11. RO Sync IP.

Fig. 12 shows the Vivado block diagram for the final design. The IP in red boxes are the most significant IP for the design, and the others are clocking infrastructure, AXI interconnects, and memory IP added by Vivado to facilitate the design. The highlighted green path shows the Controller clock signal, and the purple path is the RO gated clock signal. These paths show which IP are frozen when the gated clocks are triggered.

3.3 Evaluation

For data collection, we follow a similar approach to previous works [7], [9] and generate a set of instructions as shown in Fig. 13. For each instruction, we generate 1,000 assembly code snippets. The code snippets follow the pattern outlined in Fig. 13, that is, two or-immediate instructions to randomize the data in registers R8 and R9, which serve as the operands for the test instructions. Then, two instructions are randomly selected to precede and follow the targeted instruction. Finally, there is a branch instruction to return to the interrupt handler. There are slight modifications to this snippet for branching and memory access instructions to ensure proper code execution.

In this work, we are only disassembling the instructions, but as shown in [8], the registers also affect the power consumption. We use the 54 instructions that are used in the Dhrystone benchmark for our configuration of the Microblaze. The instructions are listed in Appendix A [37]. We chose to use the subset of instructions for the benchmark since, as previous works have shown, there is often a difference between testing done on training data and a real application. The Dhrystone benchmark was chosen since it is readily available and does not use floating point instructions, allowing it to work with the reduced

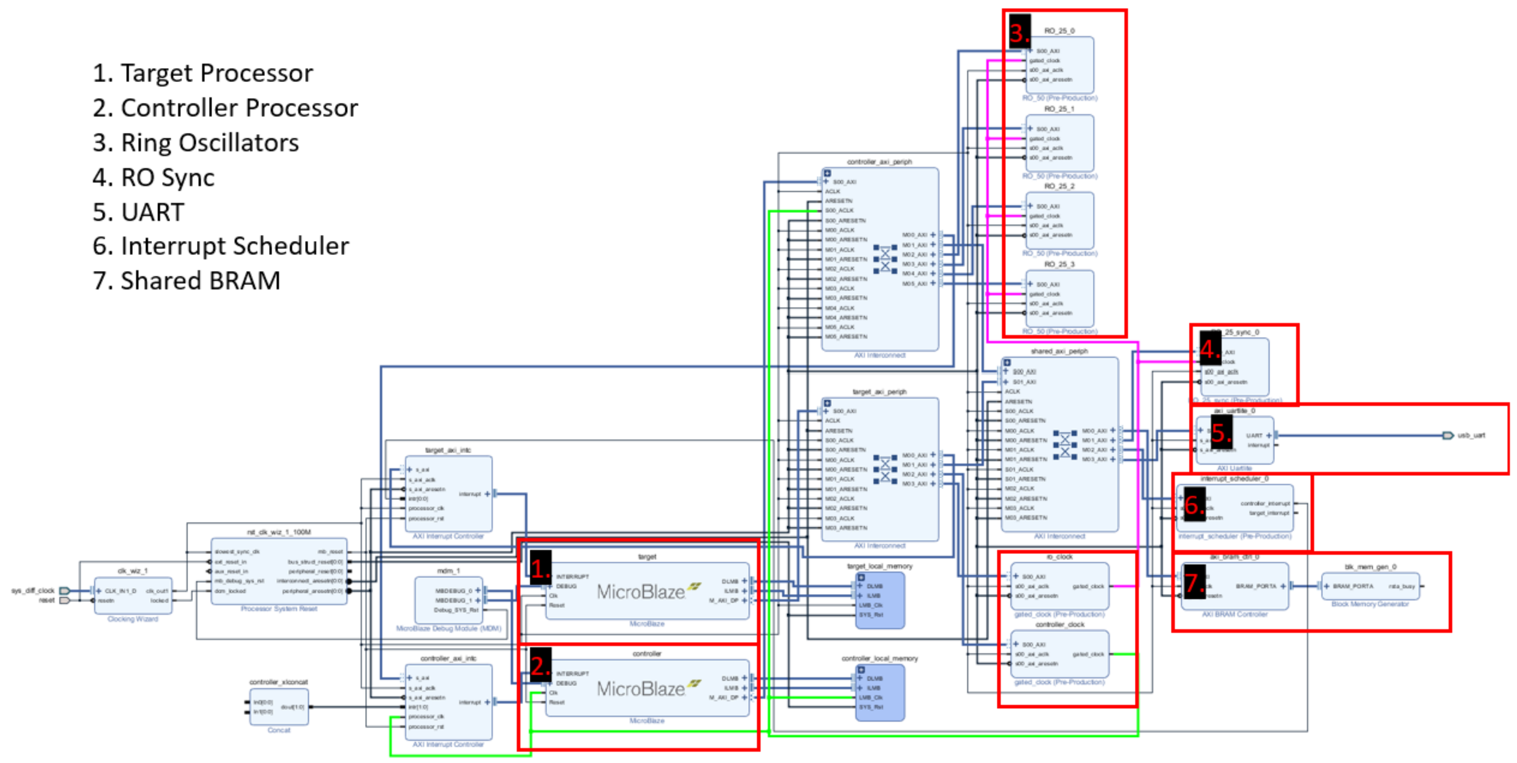


Figure 12. Side-channel data collection system block diagram.

instruction set as implemented in the Microblaze. If our approach proves feasible, then our next step will be to test its accuracy on the benchmark and compare the results.

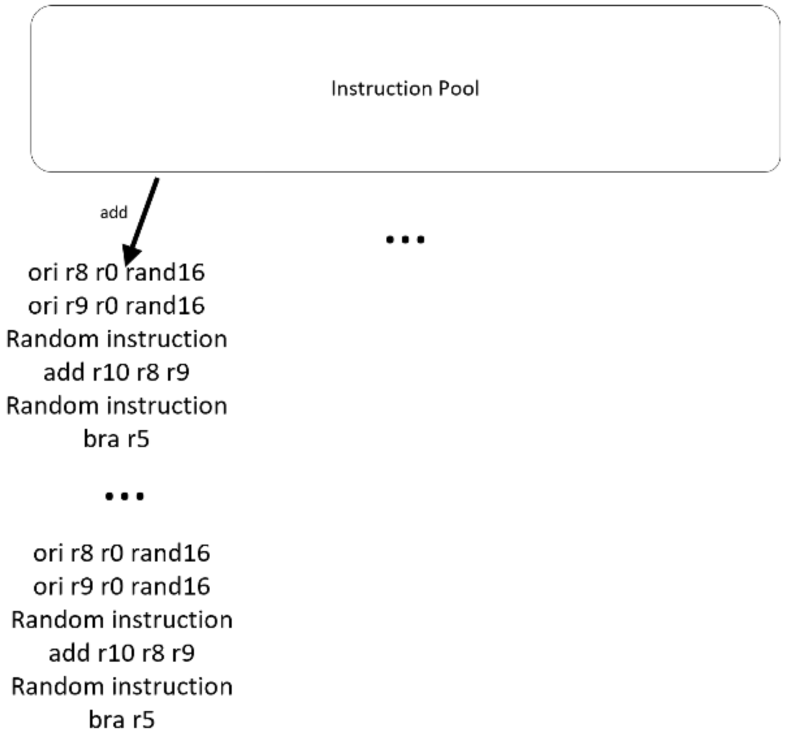


Figure 13. Microblaze instruction's training code snippet example.

Most of the instructions are single clock-cycle instructions, but a few are multi-cycle. Multi-clock-cycle instructions are split into pseudo-instructions. *Mul* and *muli* are 3-cycle instructions, so we split the second and third cycles into separate instructions *mul_1* and *mul_2*. These instructions are classified as A-type arithmetic instructions since the power difference in A and B type should come from the initial stage when the data is read into the functional unit and not during the later clock cycles that are exclusively computations. Similarly, the barrel shifting instructions (*bsll*, *bsrli*, *bsrai*, *bslli*) take two clock cycles so the second cycle is labeled as an Arithmetic B-type instruction *bs_1*. We chose B-type for *bs_1* since three of the four barrel shifting instructions are B-tpye. Finally, the memory access instructions (*lw*, *sw*, *lhui*, *lwi*, *swi*, *sbi*, *shi*) cause a 1-clock-cycle delay, so this delay slot is split into the pseudo instruction *mem_stall*.

To account for the static power consumption of the device and the power consumption of the controller, the ROs, and the communication between the different cores, we filter the results. We use a similar approach to [6]. After implementing the design and uploading it to the FPGA, we generate 1000 random code snippets. Everything on the device is enabled except for the target processor. We then average the counts for each sensor. We can model the system's power consumption by:

$$P_{total} = P_{static} + P_{RO} + P_{controller} + P_{target} + P_{noise}$$

P_{total} is the total power consumed by the device, P_{static} is the baseline power consumption of the device when nothing is running on it, P_{RO} is the power consumed by the ROs, $P_{controller}$ is the power consumption of the controller core, P_{target} is the power consumed by the target processor, and P_{noise} is the noise in the power consumption traces. By running the system as described above, we can filter out the average power consumption of the ROs, and the controller by subtracting the averaged values from the RO counts when the whole system is enabled.

To generate the classification model, we use three layers of decision tree classifiers on the instructions but not the registers. This approach is partially based on the work in [8]. First, we classify the instruction based on its class as defined in Appendix A (arithmetic, logical, branching, memory access, and other) [38]. From there, we classify the instruction as either an A or B type for arithmetic, logical, and memory access instructions. All branching instructions are B-type, so their second tier is based on conditional vs. unconditional instructions. The other instructions do not have a type or a natural way to divide them and only have two tiers. The final tier classifies the individual instructions. The full model can be seen in Fig. 14.

We chose a hierarchical approach since it is consistent with the work in [8], is representative of how we suspect the instructions should affect the power consumption, and leads to a natural way to evaluate a partially correct guess. We expect that the different classes of instructions should be one of the main contributors to the power variations. For example, we suspect that an "OR" and "XOR" instruction should be more similar than an "OR" and "LW" instruction. Even if we cannot accurately classify individual instructions, this approach gives an easy way to evaluate partial accuracy by considering the accuracy of the models at each tier. We use an 80/20 train/test split for training and evaluating the models. We also train a single-layer model that takes the power data and directly predicts the instructions to use as a baseline.

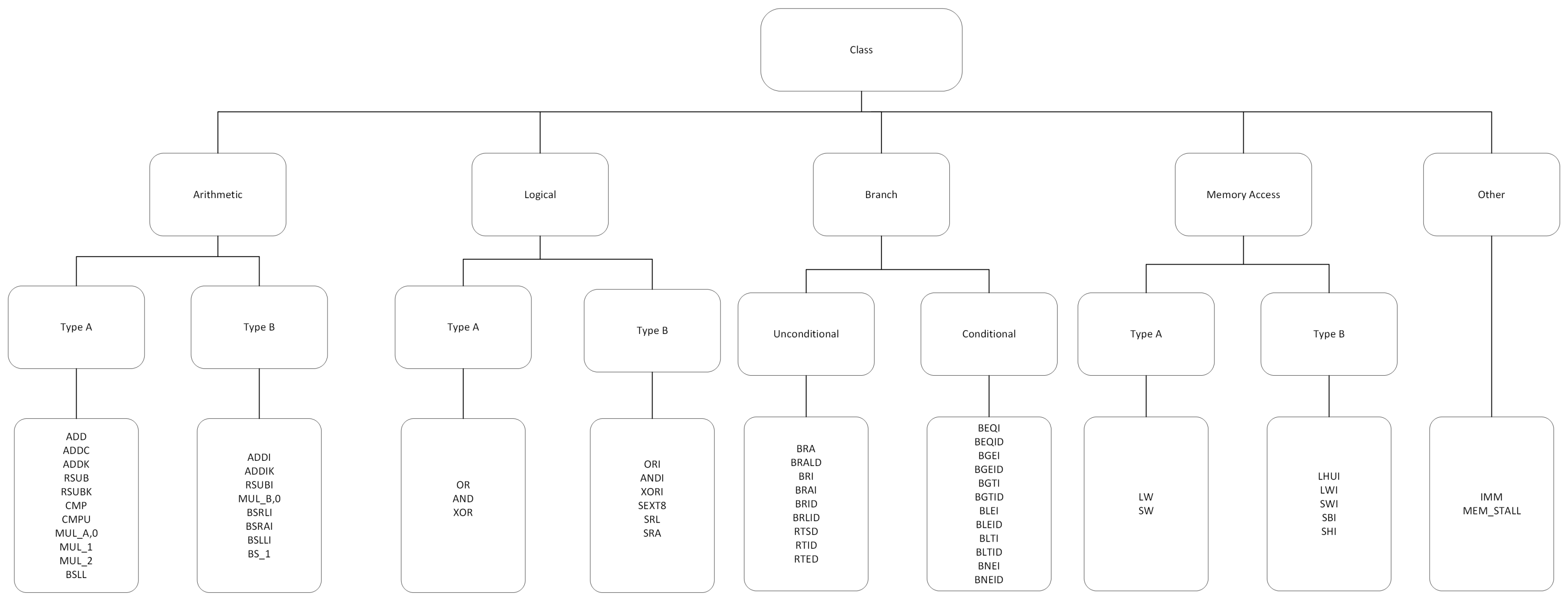

Class
Arithmetic
Logical
Branch
Memory Access
Other
Type A
Type B
Type A
Type B
Unconditional
Conditional
Type A
Type B
ADD ADDC ADDK RSUB RSUBK CMP CMPU MUL_A,0 MUL_1 MUL_2 BSLL
ADDI ADDIK RSUBI MUL_B,0 BSRLI BSRAI BSLLI BS_1
OR AND XOR
ORI ANDI XORI SEXT8 SRL SRA
BRA BRALD BRI BRAI BRID BRLID RTSD RTID RTED
BEQI BEQID BGEI BGEID BGTI BGTID BLEI BLEID BLTI BLTID BNEI BNEID
LW SW
LHUI LWI SWI SBI SHI
IMM MEM_STALL

Figure 14. Proposed machine learning model.

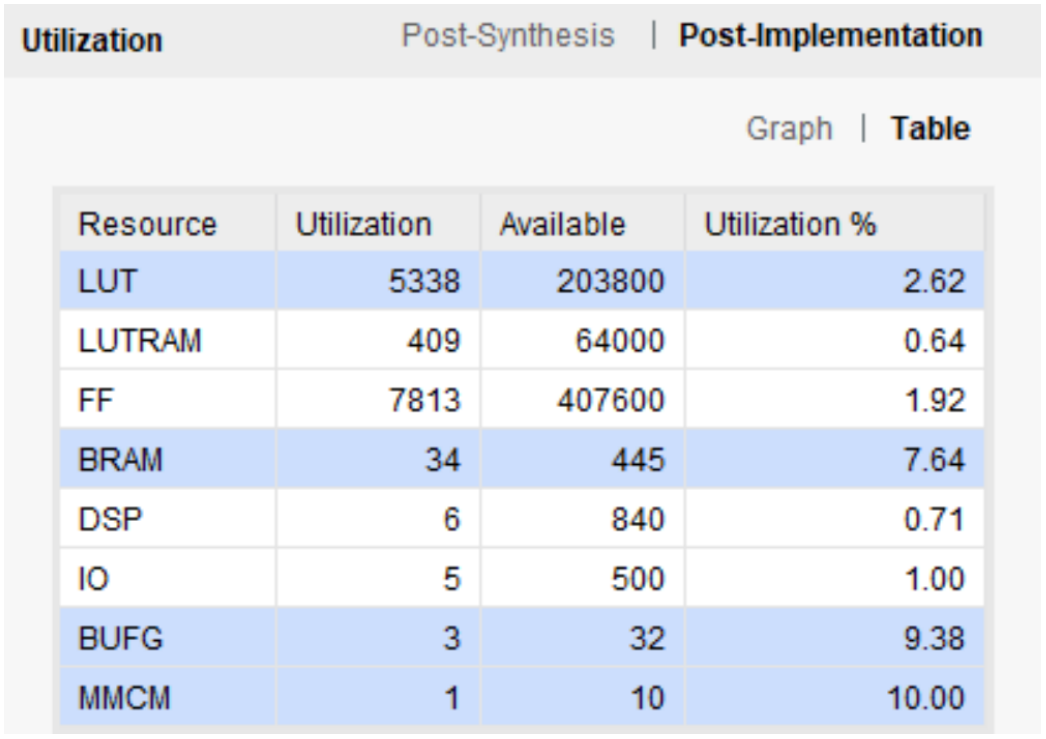
CHAPTER IV

RESULTS

In this section, we give an overview of the cost of implementing our countermeasure and outline the results of the analysis. Since the initial results show that our approach works but is not feasible in practice we also introduce some ad hoc tests and their results.

4.1 Implementation Cost

New designs will always incur overhead in resource utilization, which could make the proposed model prohibitively expensive regardless of its accuracy. Fig. 15 shows the system resource utilization for the final design, with the most important resources highlighted. LUTs are the basic building block of FPGAs, and the total system used less than 3%. Additionally, all of the resources were less than or equal to 10% utilization on our Kintex-7. The only resources that use a noteworthy amount of resources (by percentage of total utilization) are the BUFG, MMCM, and BRAM. The BUFG and MMCM are clocking resources and would remain mostly the same regardless of the design complexity unless more clocks were added. However, their relatively high utilization is mostly independent of our power monitoring system. The BRAM is also high since we have two processors and a shared memory region that accounts for most of the BRAM utilization. These regions have little to do with RO power monitors and are there to make testing easier.

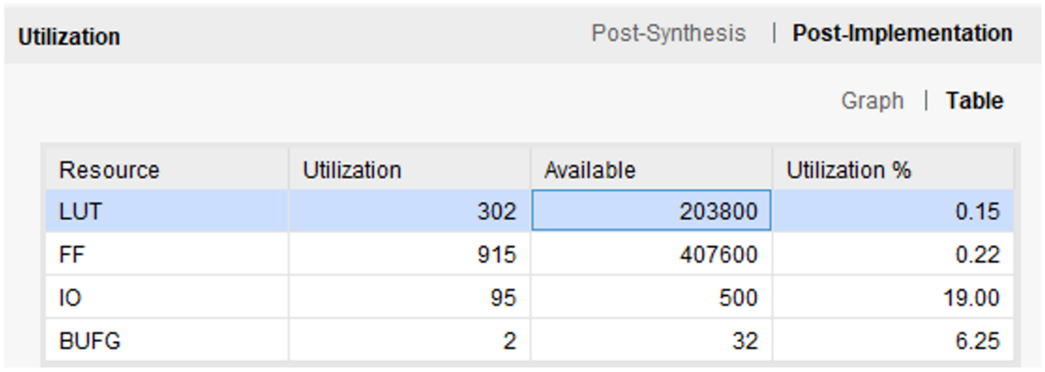

Utilization Post-Synthesis | Post-Implementation

Graph | Table

Resource	Utilization	Available	Utilization %
LUT	5338	203800	2.62
LUTRAM	409	64000	0.64
FF	7813	407600	1.92
BRAM	34	445	7.64
DSP	6	840	0.71
IO	5	500	1.00
BUFG	3	32	9.38
MMCM	1	10	10.00

Figure 15. System under test resource utilization.

Things begin looking even better when we consider the utilization of just the ROs. Fig. 16 shows the resource utilization for a single RO. The LUT utilization is at .15% utilization. The IO utilization is higher here since Vivado tied all of the IO for the RO to physical pins on the device, but these would normally be tied to its control IP. Based on these utilization percentages, we can conclude that individually, the ROs have a minimal impact on the overall resource utilization. In a real-world scenario, one of the major concerns is that there is enough remaining logic to implement the power monitors and their control logic. These low utilization percentages show that this should not be a concern for most targets.



Resource	Utilization	Available	Utilization %
LUT	302	203800	0.15
FF	915	407600	0.22
IO	95	500	19.00
BUFG	2	32	6.25

Figure 16. RO resource utilization.

4.2 Disassembly Results

As outlined in Section 3.3, we trained a series of decision tree classifiers to match the model in Fig. 14. We also trained a single-layer decision tree classifier that takes the RO counts and directly predicts the instructions. All parameters are kept at their default values, except for the variable weights, to account for the split multi-clock-cycle instructions that are unequally represented in the training data.

There are 58 instructions in our model, so randomly guessing would yield an accuracy of 1/58 = 1.72%. Since our data is unequally represented in the training data set, we also need to ensure that the models are not just selecting the largest class every time. Each *true* instruction has 1000 samples in the training data set, but the multi-clock-cycle instructions are overrepresented. The total number of samples in the dataset is 69,000, and the most represented instruction is the *mem_stall* instruction with 7,000 samples. So if the models are exclusively selecting *mem_stall* then we would expect the accuracy to be 7,000/69,000 = 10.14%. Anything higher than this number suggests that the approach is working.

Fig. 17 gives a summary of the results from running our system at 100 MHz (The maximum clock rate for the Kintex-7 FPGA). The red numbers on the right are the accuracy of each model, and the black numbers on the left are the percentage of instructions going to each model that belong to that model. This number is an upper bound to the accuracy of each model since a model cannot accurately predict an instruction that belongs to a different branch. At the top level, we can see that our model only predicts class correctly 32.36% of the time, and this error propagates through the model. Due to this propagation of errors, our combined model achieves an accuracy of 15.14%, which is almost the same as the single-layer model's accuracy of 15.12%.

These results are not great, but they are better than randomly guessing and are a good starting point to try and refine our approach. Since ROs have a strong tradeoff between measurement frequency and accuracy, we also tested the system at 50 MHz. Fig. 18 shows these results. The combined 50 MHz model has an accuracy of 13.09%, and the single layer model has an accuracy of 13.01%. Surprisingly, lowering the clock rate does not improve accuracy but makes the models worse. We have two hypotheses for what could be causing this. The first is that the RO accuracy loss for 100 MHz to 50 MHz is so small that it doesn't significantly affect the results. The other possibility is that the additional logic used to divide the clock frequency is codependent with the power monitors and affects the results. Further testing needs to be done to isolate this cause. For both frequencies, we ran

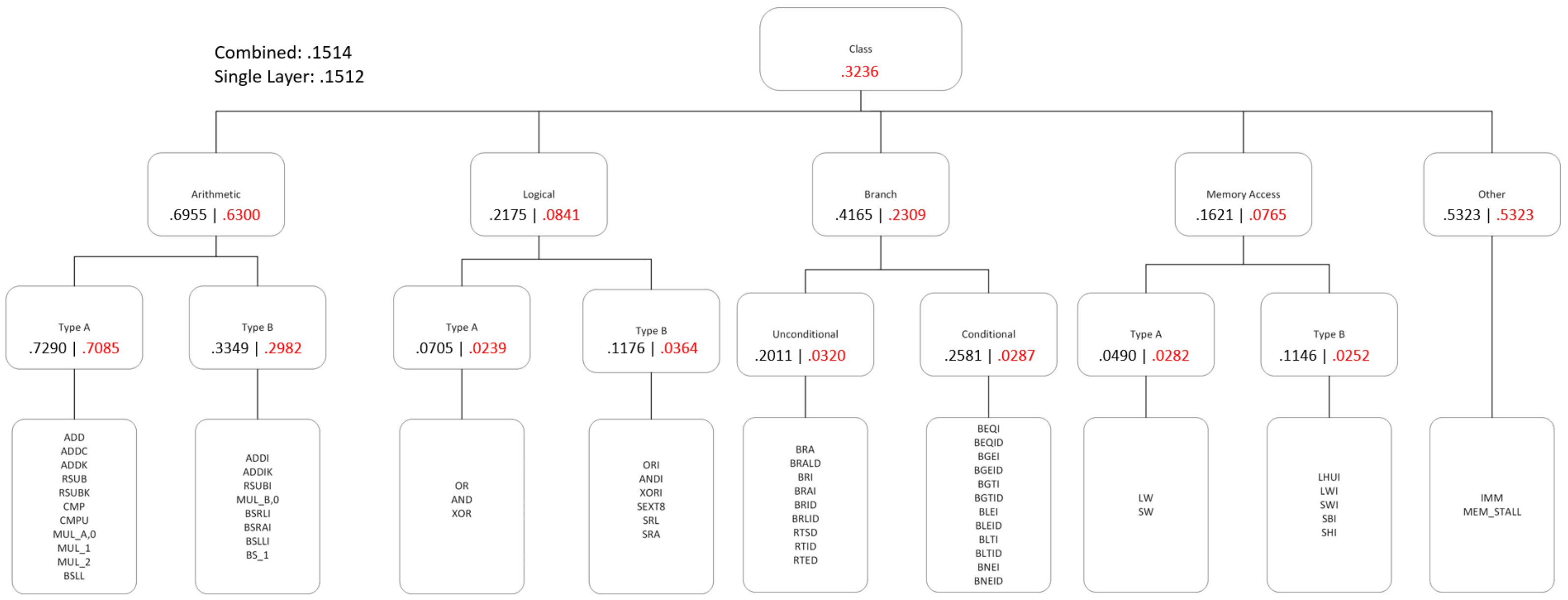


Figure 17. Classification accuracies for the 100 MHz test system data.

a 5-fold cross-validation on the single-layer model and found that the results were within +-.7%. This suggests that the results were representative of our data and not due to overfitting the data.

Naturally, some of the models are more accurate than others. For example, of the data correctly sent to the Other models, almost all of it is classified correctly. Of the data the conditional branching models correctly receive, they can only classify about 10% correctly. Most of the models are somewhere between these two extremes. This difference is most likely due to the similarities of some instructions. Although we do not have access to the source files for the Microblaze, we can make assumptions on which instructions share resources based on their semantics and knowledge of other processors.

4.3 Binning Results

Similar to previous works [23], [27], we decided to train coarse-grained models based on the instructions' hardware utilization. As mentioned previously, the Branching Conditional models have the worst accuracy. We don't know specifically how these instructions are implemented, but it stands to reason that they all use the same circuit to compare the numbers and then update the program counter. This could explain why those models are only slightly better than randomly guessing. To test this, we create two coarse-grained datasets by grouping similar instructions that we suspect are using the same functional units.

The first grouping is outlined in Table 2, the instructions not listed were kept as individual instructions. We chose these groupings based on the instructions we suspect are

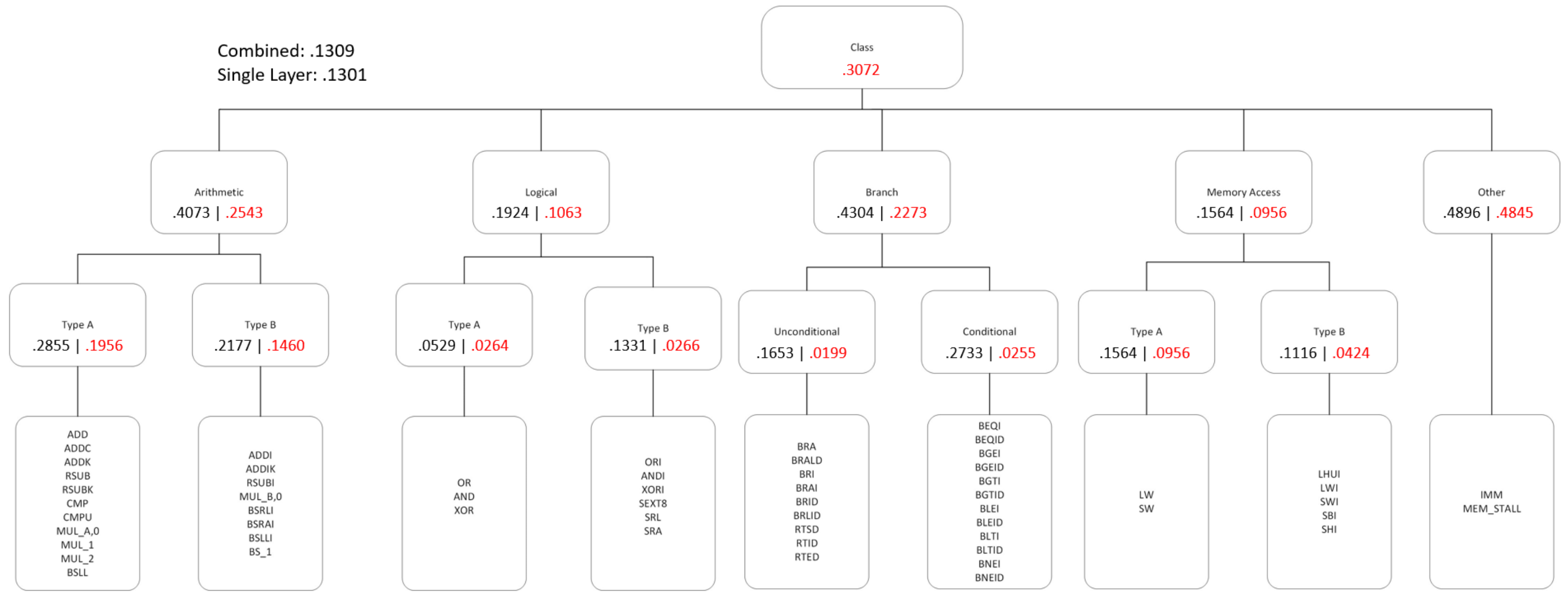


Figure 18. Classification accuracies for the 50 MHz test system data.

using the same functional unit within each type (i.e., we keep Type A and Type B instructions separate even if they share functional units.) This approach reduced the total number of instructions to 18, with the most over-represented group being the conditional pseudo-instruction with 12,000 samples. For this group, randomly guessing would yield an accuracy of 1/18 = 5.56%, and exclusively picking the largest class has an accuracy of 12000/69000 = 17.39%.

The results for the 100 MHz Group 1 data can be seen in Fig. 19. Unlike with the original data, there is a large difference between the accuracy of the combined model at 20.52% and the single-layer model at 28.68%. The 50 MHz Group 1 results are shown in Fig. 20. These results were much closer at 20.51% for the combined model and 20.62% for the single-layer model. This gives seemingly contradicting implications. For the 100 MHz data, it suggests that the single-layer model scales better than the combined model. This seems to be the case since the top level model that predicts class only achieves accuracies in the low to mid 30s, and the errors propagate from there. However, the 50 MHz data suggests that the models grow similarly. Further testing needs to be done to fully understand these discrepancies and discover the true relationship.

For the second grouping, outlined in Table 3, we group the instructions by their functional unit utilization, but this time we ignore the instruction's type. Grouping 2 reduces the total number of instructions down to 13, which puts the accuracy of randomly guessing at 1/13 = 7.69%, and the pseudo-instruction with the largest percentage of representation is still the conditional branching pseudo-instruction at 17.39%. For this grouping, we only use the single-layer model since removing the Type makes the

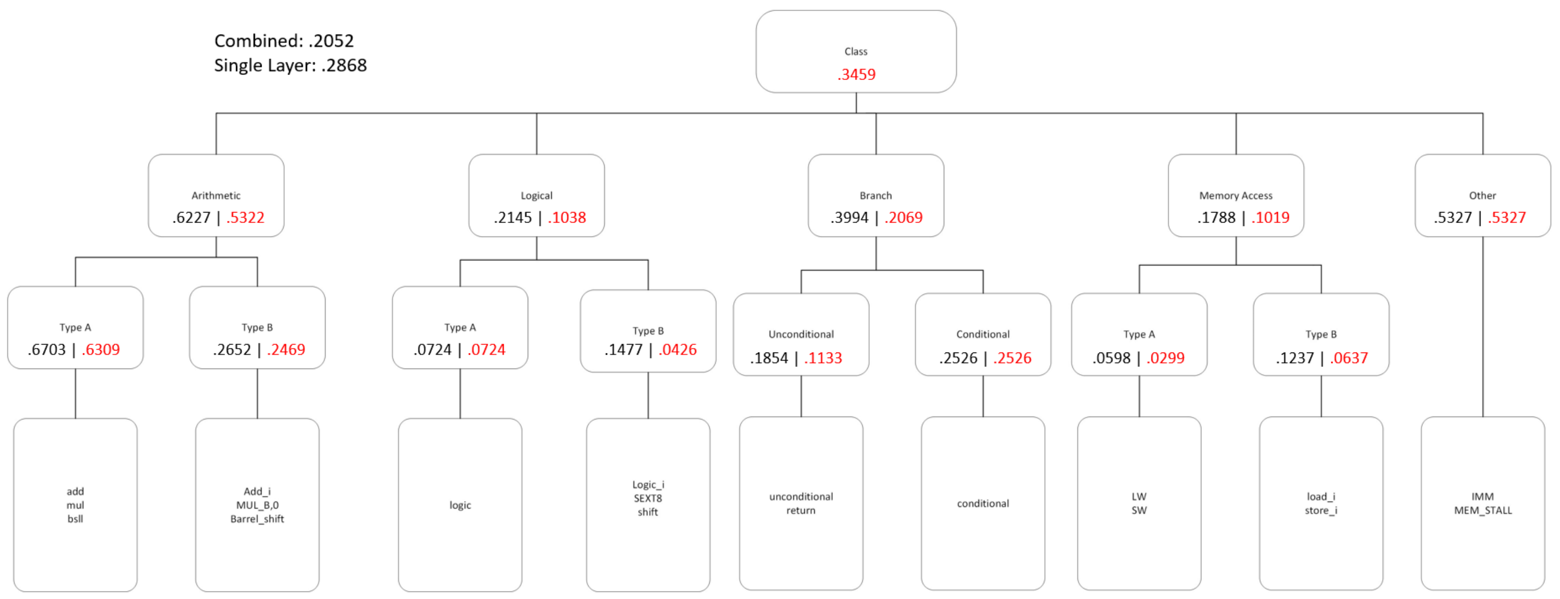


Figure 19. Classification accuracies for the 100 MHz Group 1 data.

Table 2. Microblaze instructions: grouping 1 summary.

Original	Group 1
add, addk, rsub, addc, cmp, cmpu, rsubk	add
mul, mul_1, mul_2	mul
addi, addik, rsubi	add_i
bsrli, bsrai, bslli, bs_1	barrel_shift
and, or, xor	logic
andi, ori, xori	logic_i
srl, sra	shift
bra, brald, bri, brai, bird, brlid	unconditional
rtid, rted, rtsd	return
beqi, beqid, bgei, bgeid, bgti, bgtid, blei, bleid, blti, bltid, bnei, bneid	conditional
lhui, lwi	load_i
sbi, shi, swi	store_i

Table 3. Microblaze instructions: grouping 2 summary.

Original	Group 2
add, addk, rsub, addc, cmp, cmpu, rsubk, addi, addik, rsubi	add
mul, mul_1, mul_2, muli	mul
bsll, bsrli, bsrai, bslli, bs_1	barrel_shift
and, or, xor, andi, ori, xori	logic
srl, sra	shift
bra, brald, bri, brai, bird, brlid	unconditional
rtid, rted, rtsd	return
beqi, beqid, bgei, bgeid, bgti, bgtid, blei, bleid, blti, bltid, bnei, bneid	conditional
lhui, lwi, lw	load
sbi, shi, swi, sw	store

combined model unusable. This grouping achieves an accuracy of 29.90% for the 100 MHz data and 23.66% for the 50 MHz data. For both groupings, we used 5-fold cross-validation and found that all folds produced the same results within +-1.3%, suggesting that this is representative of our data and not due to overfitting.

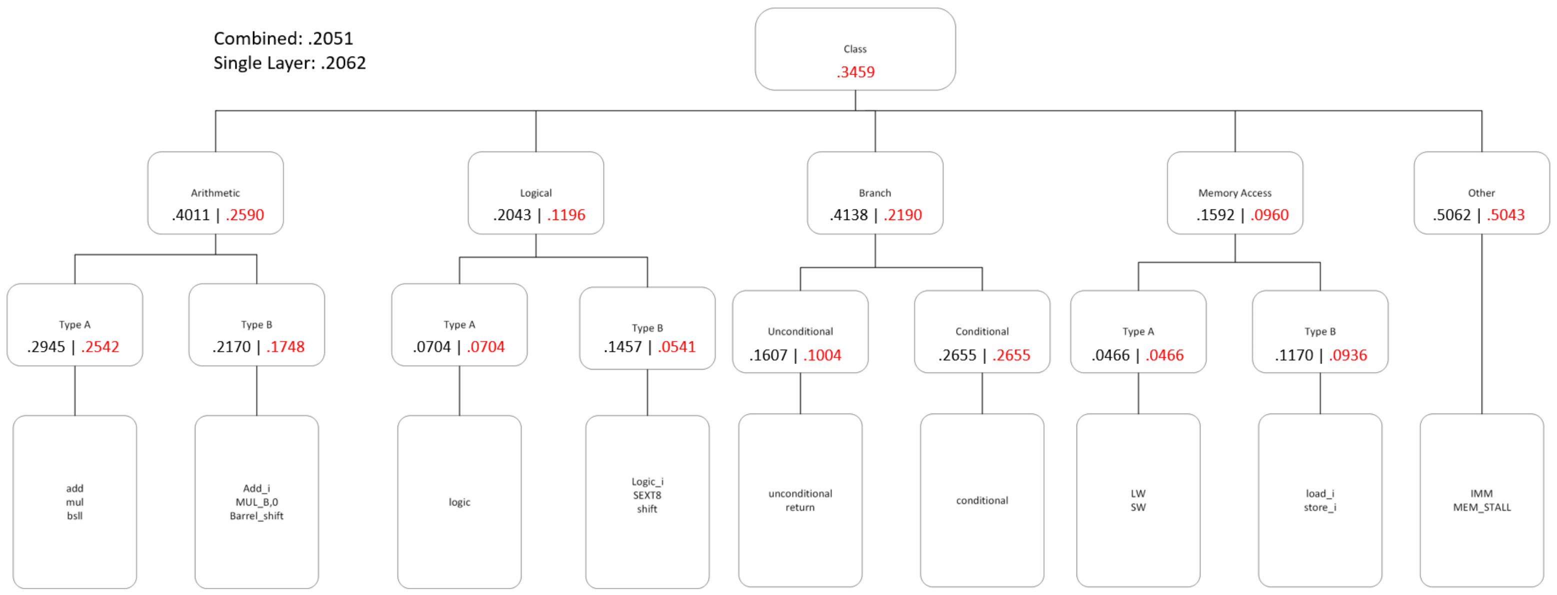


Figure 20. Classification accuracies for the 50 MHz Group 1 data.

4.4 Results Summary

A summary of our findings our outlined in Table 4. Our data proves that assembly instructions have a noticeable effect on the output of our ring oscillator power monitors. However, our combined three-tiered model offers little to no benefit over using a single-layer model for the original data and the 50 MHz Group 1 data and is about 8% worse than the single-layer model for the 100 MHz Group 1 data. In all cases, the models are better than randomly guessing and exclusively selecting the most represented instruction. We also created two coarse-grained models that combined instructions based on their suspected functional unit utilization. The first grouping preserved the instruction's type, while the second did not. Using the first group offered a substantial improvement over the original data, and using the second group offered slight improvements over the Group 1 model. The implications of this work and potential improvements will be discussed in the next chapter.

Table 4. Comparison on instruction classification accuracies of each model.

	Original	Group 1	Group 2
Random guess	1/58 = .0172	1/18 = .0556	1/13 = .0769
Largest class	7/69 = .1014	12/69 = .1739	12/69 = .1739
100 MHz: single layer	.1512	.2868	.2990
100 MHz: combined	.1514	.2052	-
50 MHz: single layer	.1301	.2062	.2366
50 MHz: combined	.1309	.2051	-

CHAPTER V

CONCLUSION AND FUTURE WORK

5.1 Conclusions and Key Contributions

As new technologies continue to emerge, it is important that we consider the security vulnerabilities that they may introduce. With the recent acquisitions of Alterra by Intel and Xilinx by AMD, we are likely to start seeing more hybrid GPP/FPGA processors. Although these devices can offer many benefits to developers, they can also introduce new vulnerabilities. In this work, we tested to see if FPGA-based RO power monitors can be used to disassemble code running on a GPP on the same chip die.

Our fine-grained models that classify individual instructions achieved accuracies from about 13-15%. The coarse-grained models that grouped instructions based on their suspected functional unit utilization achieved accuracies ranging from about 20-30%. These are all better than randomly guessing but are far from being useful in practice.

There are limitations to our approach. First, our design used a lot of control logic to make the data collection easier, which an attacker would not have access to. The gated clocks and the RO_sync are the primary culprits. For this work to be feasible in practice, an adversary would need to be capable of collecting the power data in real time and trimming the RO counts that don't correspond to the targeted code. Additionally, our models only consider the instruction and not the registers or immediate values they process. Some works [25], [26] suggest that these values have a significant effect on the power consumption, but for full code disassembly, the models would need to be updated.

Finally, This design used a softcore processor running at the same clock rate as the ROs. We chose this approach since it gave us more control of the target processor for testing the system and reduced the background signal noise compared to a hardcore GPP. However, a more realistic scenario would be a hardcore GPP as the target processor, but these tend to run at much higher clock rates than FPGA fabric. For example, on the Zedboard, which runs a Zynq processor, [39] the GPP operates at 667 MHz while the FPGA fabric operates at 100 MHz. This means sampling the RO like we did once per clock cycle would cause each data point to correspond to 6 2/3 instructions. Creating further issues with implementing this attack.

Despite these limitations and low accuracies, this work still highlights that an attacker can create an FPGA-based power monitor whose output is correlated to the executed instructions. Although the threat is not feasible at the moment, it could become a salient threat as the technology matures. The following section outlines future works for improving the attack and potential countermeasures.

5.2 Future Works

Even though our approach is not currently feasible, it does not mean that we should ignore the threat. In this section we outline future works to refine the attack and potential countermeasures to protect against it.

The largest hurdle for implementing this attack is the quality of the power monitors. Future works could explore ways to improve the quality of their data. Also, in this work, we looked at ROs, but future works could test if TDC-based power monitors yield better results. Researchers could also look at offsetting the clock for sampling. For example, sampling two power monitors on the rising clock edge and the other two on a falling clock edge. With this approach, the falling clock edge-triggered power monitors would be covering two instructions, but they could also help with capturing power variations that occur in the first/last half of the clock cycle.

More robust machine learning could also improve the results. Similar to the work in [7], future works could identify common instruction tuples and work those into a Markov Model or Recurrent Neural Network. We manually grouped instructions based on their suspected functional unit utilization, but clustering could be used to identify better groupings to improve accuracy for coarse-grained models. As mentioned previously, the operands/immediate values also need to be incorporated into the model for it to be useful in practice.

Finally, this is the perfect opportunity to begin researching countermeasures so that we have strategies to protect against the attack as it matures. However, protecting against this attack will be challenging. Traditional side-channel countermeasures like masking [40] could work, but protecting the entire processor would likely be prohibitively expensive in terms of money and performance. Future works need to look into methods to protect against this attack.

REFERENCES

[1] T. Huffmire, C. Irvine, T. D. Nguyen, T. Levin, R. Kastner, and T. Sherwood, *Handbook of FPGA Design Security*, en. Dordrecht: Springer Netherlands, 2010, ISBN: 978-90-481-9156-7 978-90-481-9157-4. DOI: 10.1007/978-90-481-9157-4. [Online]. Available: http://link.springer.com/10.1007/978-90-481-9157-4 (visited on 04/21/2022).

[2] N. C. Thompson and S. Spanuth, "The decline of computers as a general purpose technology," *Communications of the ACM*, vol. 64, no. 3, pp. 64–72, Feb. 2021, ISSN: 0001-0782. DOI: 10.1145/3430936. [Online]. Available: http://doi.org/10.1145/3430936 (visited on 06/21/2021).

[3] *Intel Acquisition of Altera*, en-US, Dec. 2015. [Online]. Available: https://newsroom.intel.com/press-kits/intel-acquisition-of-altera/ (visited on 05/10/2022).

[4] *AMD Acquires Xilinx*, en, 2022. [Online]. Available: https://www.amd.com/en/corporate/xilinx-acquisition (visited on 05/10/2022).

[5] A. Duncan, F. Rahman, A. Lukefahr, F. Farahmandi, and M. Tehranipoor, "FPGA Bitstream Security: A Day in the Life," en, in *2019 IEEE International Test Conference (ITC)*, Washington, DC, USA: IEEE, Nov. 2019, pp. 1–10, ISBN:

978-1-72814-823-6. DOI: 10.1109/ITC44170.2019.9000145. [Online]. Available: https://ieeexplore.ieee.org/document/9000145/ (visited on 06/26/2021).

[6] M. Zhao and G. E. Suh, "FPGA-Based Remote Power Side-Channel Attacks," in *2018 IEEE Symposium on Security and Privacy (SP)*, ISSN: 2375-1207, May 2018, pp. 229–244. DOI: 10.1109/SP.2018.00049.

[7] T. Eisenbarth, C. Paar, and B. Weghenkel, "Building a Side Channel Based Disassembler," en, in *Transactions on Computational Science X*, Series Title: Lecture Notes in Computer Science, vol. 6340, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 78–99, ISBN: 978-3-642-17498-8 978-3-642-17499-5. DOI: 10.1007/978-3-642-17499-5_4. [Online]. Available: http://link.springer.com/10.1007/978-3-642-17499-5_4 (visited on 03/12/2022).

[8] J. Park, X. Xu, Y. Jin, D. Forte, and M. Tehranipoor, "Power-based Side-Channel Instruction-level Disassembler," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, Jun. 2018, pp. 1–6. DOI: 10.1109/DAC.2018.8465848.

[9] D. Strobel, F. Bache, D. Oswald, F. Schellenberg, and C. Paar, "SCANDALee: A side-ChANnel-based DisAssembLer using local electromagnetic emanations," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, ISSN: 1558-1101, Mar. 2015, pp. 139–144. DOI: 10.7873/DATE.2015.0639.

[10] E. Williams, *TEMPEST: A Tin Foil Hat For Your Electronics And Their Secrets*, en-US, Oct. 2015. [Online]. Available: https://hackaday.com/2015/10/19/tempest-a-tin-foil-hat-for-your-electronics-and-their-secrets/ (visited on 06/28/2021).

[11] Y. Zhou and D. Feng, “Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing.,” *IACR Cryptology ePrint Archive*, vol. 2005, p. 388, Jan. 2005.

[12] P. Kocher, J. Jaffe, and B. Jun, “Differential Power Analysis,” en, in *Advances in Cryptology — CRYPTO’ 99*, M. Wiener, Ed., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 1999, pp. 388–397, ISBN: 978-3-540-48405-9. DOI: 10.1007/3-540-48405-1_25.

[13] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks Revealing the Secret of Smart Cards*, 1st ed. Springer-Verlag US, 2007. [Online]. Available: 10.1007/978-0-387-38162-6.

[14] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, “The EM Side—Channel(s),” en, in *Cryptographic Hardware and Embedded Systems - CHES 2002*, B. S. Kaliski, ç. K. Koç, and C. Paar, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2003, pp. 29–45, ISBN: 978-3-540-36400-9. DOI: 10.1007/3-540-36400-5_4.

[15] J. A. Dawson, J. T. McDonald, J. Shropshire, T. R. Andel, P. Luckett, and L. Hively, “Rootkit detection through phase-space analysis of power voltage measurements,” in *2017 12th International Conference on Malicious and Unwanted Software (MALWARE)*, Oct. 2017, pp. 19–27. DOI: 10.1109/MALWARE.2017.8323953.

[16] S. S. Clark, B. Ransford, A. Rahmati, *et al.*, “{WattsUpDoc}: Power Side Channels to Nonintrusively Discover Untargeted Malware on Embedded Medical Devices,”

en, 2013. [Online]. Available: https://www.usenix.org/conference/healthtech13/workshop-program/presentation/clark (visited on 05/30/2022).

[17] H. A. Khan, N. Sehatbakhsh, L. N. Nguyen, M. Prvulovic, and A. Zajić, "Malware Detection in Embedded Systems Using Neural Network Model for Electromagnetic Side-Channel Signals," en, *Journal of Hardware and Systems Security*, vol. 3, no. 4, pp. 305–318, Dec. 2019, ISSN: 2509-3436. DOI: 10.1007/s41635-019-00074-w. [Online]. Available: https://doi.org/10.1007/s41635-019-00074-w (visited on 04/22/2022).

[18] J. He, H. Ma, Y. Liu, and Y. Zhao, "Golden Chip-Free Trojan Detection Leveraging Trojan Trigger's Side-Channel Fingerprinting," *ACM Transactions on Embedded Computing Systems*, vol. 20, no. 1, 6:1–6:18, Dec. 2020, ISSN: 1539-9087. DOI: 10.1145/3419105. [Online]. Available: https://doi.org/10.1145/3419105 (visited on 07/09/2022).

[19] T. L. Cayton, *Differential Analysis Attacks and Countermeasures in Elliptic Curve Cryptography*, en, 2002. [Online]. Available: http://koclab.cs.ucsb.edu/teaching/ecc/project/2015Projects/Cayton.pdf.

[20] V. Cristiani, M. Lecomte, and T. Hiscock, "A Bit-Level Approach to Side Channel Based Disassembling," en, in *Smart Card Research and Advanced Applications*, S. Belaïd and T. Güneysu, Eds., ser. Lecture Notes in Computer Science, Cham:

Springer International Publishing, 2020, pp. 143–158, ISBN: 978-3-030-42068-0. DOI: 10.1007/978-3-030-42068-0_9.

[21] V. M. Vaidyan and A. Tyagi, "Instruction Level Disassembly through Electromagnetic Side-Chanel: Machine Learning Classification Approach with Reduced Combinatorial Complexity," in *Proceedings of the 2020 3rd International Conference on Signal Processing and Machine Learning*, ser. SPML 2020, New York, NY, USA: Association for Computing Machinery, Dec. 2020, pp. 124–130, ISBN: 978-1-4503-7573-3. DOI: 10.1145/3432291.3432300. [Online]. Available: https://dl.acm.org/doi/10.1145/3432291.3432300 (visited on 05/26/2023).

[22] H. Fendri, M. Macchetti, J. Perrine, and M. Stojilović, "A Deep-Learning Approach to Side-Channel Based CPU Disassembly at Design Time," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, ISSN: 1558-1101, Mar. 2022, pp. 670–675. DOI: 10.23919/DATE54114.2022.9774531.

[23] J. Maillard, T. Hiscock, M. Lecomte, and C. Clavier, "Towards Fine-grained Side-Channel Instruction Disassembly on a System-on-Chip," in *2022 25th Euromicro Conference on Digital System Design (DSD)*, ISSN: 2771-2508, Aug. 2022, pp. 472–479. DOI: 10.1109/DSD57027.2022.00069.

[24] O. Glamocanin, R. Islambouli, and D. Mahmoud, "Machine learning for side-channel disassembly," *work*, vol. 5, p. 3,

[25] H. Liu, H. Li, and E. Y. Vasserman, "Practicality of Using Side-Channel Analysis for Software Integrity Checking of Embedded Systems," en, in *Security and Privacy*

in Communication Networks, B. Thuraisingham, X. Wang, and V. Yegneswaran, Eds., Series Title: Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 164, Cham: Springer International Publishing, 2015, pp. 277–293, ISBN: 978-3-319-28864-2 978-3-319-28865-9. DOI: 10.1007/978-3-319-28865-9_15. [Online]. Available: http://link.springer.com/10.1007/978-3-319-28865-9_15 (visited on 10/26/2022).

[26] H. Liu and E. Vasserman, "Side-channel Programming for Software Integrity Checking," *EAI Endorsed Transactions on Security and Safety*, vol. "8", no. 28, Jun. 2021, ISSN: 2032-9393. [Online]. Available: https://eudl.eu/doi/10.4108/eai.2-6-2021.170013 (visited on 11/02/2022).

[27] D. Krishnankutty, Z. Li, R. Robucci, N. Banerjee, and C. Patel, "Instruction Sequence Identification and Disassembly Using Power Supply Side-Channel Analysis," en, *IEEE Transactions on Computers*, vol. 69, no. 11, pp. 1639–1653, Nov. 2020, ISSN: 0018-9340, 1557-9956, 2326-3814. DOI: 10.1109/TC.2020.3018092. [Online]. Available: https://ieeexplore.ieee.org/document/9171473/ (visited on 05/26/2023).

[28] P. C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," en, in *Advances in Cryptology — CRYPTO '96*, N. Koblitz, Ed., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 1996, pp. 104–113, ISBN: 978-3-540-68697-2. DOI: 10.1007/3-540-68697-5_9.

[29] Y. Yarom and K. Falkner, "{FLUSH+RELOAD}: A High Resolution, Low Noise, L3 Cache {Side-Channel} Attack," en, 2014, pp. 719–732, ISBN: 978-1-931971-15-7. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/yarom (visited on 04/12/2022).

[30] Y. A. Younis, K. Kifayat, Q. Shi, and B. Askwith, "A New Prime and Probe Cache Side-Channel Attack for Cloud Computing," in *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*, Oct. 2015, pp. 1718–1724. DOI: 10.1109/CIT/IUCC/DASC/PICOM.2015.259.

[31] P. Kocher, J. Horn, A. Fogh, *et al.*, "Spectre Attacks: Exploiting Speculative Execution," in *2019 IEEE Symposium on Security and Privacy (SP)*, ISSN: 2375-1207, May 2019, pp. 1–19. DOI: 10.1109/SP.2019.00002.

[32] M. Lipp, M. Schwarz, D. Gruss, *et al.*, "Meltdown: Reading Kernel Memory from User Space," en, 2018, pp. 973–990, ISBN: 978-1-939133-04-5. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/lipp (visited on 03/20/2022).

[33] F. Schellenberg, D. R. Gnad, A. Moradi, and M. B. Tahoori, "An inside job: Remote power analysis attacks on FPGAs," in *2018 Design, Automation Test in Europe*

Conference Exhibition (DATE), ISSN: 1558-1101, Mar. 2018, pp. 1111–1116. DOI: 10.23919/DATE.2018.8342177.

[34] J. Gravellier, J.-M. Dutertre, Y. Teglia, and P. Loubet-Moundi, "High-Speed Ring Oscillator based Sensors for Remote Side-Channel Attacks on FPGAs," in *2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, ISSN: 2640-0472, Dec. 2019, pp. 1–8. DOI: 10.1109/ReConFig48160.2019.8994789.

[35] *Genesys 2 Reference Manual - Digilent Reference*. [Online]. Available: https://digilent.com/reference/programmable-logic/genesys-2/reference-manual (visited on 03/22/2022).

[36] A. M. Gautier, T. R. Andel, and R. Benton, "On-Device Detection via Anomalous Environmental Factors," in *Proceedings of the 8th Software Security, Protection, and Reverse Engineering Workshop*, ser. SSPREW-8, New York, NY, USA: Association for Computing Machinery, Dec. 2018, pp. 1–8, ISBN: 978-1-4503-6096-8. DOI: 10.1145/3289239.3289246. [Online]. Available: https://doi.org/10.1145/3289239.3289246 (visited on 04/24/2022).

[37] Xilinx, *MicroBlaze Processor Reference Guide*, en, 2021.

[38] *MicroBlaze Instruction Set*, Lund University Electrical and Information Technology. [Online]. Available: https://www.eit.lth.se/fileadmin/eit/courses/eit070/Laborationer/MicroBlazeIS.pdf (visited on 08/24/2022).

[39] L. H. Crockett, R. A. Elliot, M. A. Enderwitz, and R. W. Stewart, *The ZYNQ book: embedded processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 all programmable SoC*. [S.l.]: Strathclyde Academic Media, 2014, The book can be consulted by contacting: DGS-RP-IL : Bellotta, Antonio. [Online]. Available: https://cds.cern.ch/record/2001018.

[40] E. Prouff and M. Rivain, "Masking against Side-Channel Attacks: A Formal Security Proof," en, in *Advances in Cryptology – EUROCRYPT 2013*, T. Johansson and P. Q. Nguyen, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2013, pp. 142–159, ISBN: 978-3-642-38348-9. DOI: 10.1007/978-3-642-38348-9_9.

APPENDIX

Microblaze Instructions

A full list of Microblaze instructions can be found in the Microblaze Processor Reference Guide [37]. For this research, we are only using the instructions that are in the Dhrystone benchmark for our configuration of the Microblaze processor. The classes (arithmetic, branching, logical, memory access, and other) are based on the divisions outlined by Lund University [38]. Microblaze instructions are divided into two types [37]. Type A instructions are used for register-to-register instructions. Type B instructions are for register-immediate instructions. The Microblaze is a reduced instruction set architecture (RISC), so all instructions are 32 bits. The subset of instructions that are used in this research are listed in Table 5.

Appendix Table 5. Microblaze instructions Summary.

Instruction	**Type**	**Class**	**Description**
add	A	Arithmetic	Add
addc	A	Arithmetic	Add with carry
addk	A	Arithmetic	Add with keep carry
addi	B	Arithmetic	Add immediate
addik	B	Arithmetic	Add immediate and keep carry
and	A	Logical	And
andi	B	Logical	And with immediate
beqi	C	Branching	Branch immediate if equal
beqid	C	Branching	Branch immediate if equal with delay
bgei	C	Branching	Branch immediate if greater or equal
bgeid	C	Branch	Branch immediate if greater or equal with delay
bgti	C	Branch	Branch immediate if greater than
bgtid	C	Branching	Branch immediate if greater than with delay
blei	C	Branching	Branch immediate if less or equal
bleid	C	Branching	Branch immediate if less or equal with delay
blti	C	Branching	Branch immediate if less than
bltid	C	Branching	Branch immediate if less than with delay
bnei	C	Branching	Branch immediate if not equal
bneid	C	Branching	Branch immediate if not equal with delay
bra	U	Branching	Branch absolute
brald	C	Branching	Branch absolute with link and delay
bri	C	Branching	Branch immediate
brai	C	Branching	Branch absolute immediate
brid	C	Branching	Branch immediate with delay
brlid	C	Branching	Branch and link immediate with delay
bsrli	B	Logical	Barrel shift right logical immediate
bsrai	B	Logical	Barrel shift right arithmetic immediate
bslli	B	Logical	Barrel shift left logical immediate
cmp	A	Arithmetic	Compare (signed)
cmpu	A	Arithmetic	Compare (unsigned)
imm	NA	Other	Immediate
lbui	B	Memory	Load byte unsigned immediate
lhui	B	Memory	Load halfword unsigned immediate
lw	A	Memory	Load word
lwi	B	Memory	Load word immediate
mul	A	Arithmetic	Multiply
muli	B	Arithmetic	Multiply immediate
or	A	Logical	Or
ori	B	Logical	Or immediate
rsub	A	Arithmetic	Subtract

Table 5 cont.

Instruction	Type	Class	Description
rsubk	A	Arithmetic	Subtract and keep carry
rsubi	B	Arithmetic	Subtract immediate
rtid	U	Branching	Return from interrupt
rted	U	Branching	Return from exception
rtsd	U	Branching	Return from subroutine
sbi	B	Memory	Store byte immediate
sext8	B	Logical	Sign extend byte
shi	B	Memory	Store halfword immediate
sra	B	Arithmetic	Shift right arithmetic
srl	B	Arithmetic	Shift right logical
sw	A	Memory	Store word
swi	B	Memory	Store word immediate
xor	A	Logical	Xor
xori	B	Logical	Xor with immediate

BIOGRAPHICAL SKETCH

Name of Author: Brandon R. Baggett

Graduate and Undergraduate School Attended:

University of South Alabama, Mobile, Alabama

Degrees Awarded:

Bachelor of Science in Mathematics and Statistics, 2019

Bachelor of Science in Computer Science, 2019

Doctor of Philosophy in Computing, 2023

Publications:

Baggett, B., Andel T., McDonald, J. T., (2021). Location Distribution as a Side Channel Countermeasure. *SoutheastCon 2021*, 01-05. https://doi.org/10.1109/SoutheastCon45413.2021.9401916

Mullins T., Baggett, B., Andel T., McDonald, J. T., (2022). Circuit-Variant Moving Target Defense for Side-Channel Attacks. *International Conference on Cyber Warfare and Security*, 17(1), 219-226. https://doi.org/10.34190/iccws.17.1.14