Instruction Sequence Identification and Disassembly Using Power Supply Side-Channel Analysis

Deepak Krishnankutty, *Graduate Student Member, IEEE*, Zheng Li, Ryan Robucci, *Member, IEEE*, Nilanjan Banerjee, and Chintan Patel, *Member, IEEE*

Abstract—Embedded systems are prone to leak information via side-channels associated with their physical internal activity, such as power consumption, timing, and faults. Leaked information can be analyzed to extract sensitive data and devices should be assessed for such vulnerabilities. Side-channel power-supply leakage from embedded devices can also provide information regarding instruction-level activity for control code executed on these devices. Methods proposed to disassemble instruction-level activity via side-channel leakage have not addressed issues related to pipelined multi-clock-cycle architectures, nor have proven robustness or reliability. The problem of detecting malicious code modifications while not obstructing the sequence of instructions being executed needs to be addressed. In this article, instruction sequences being executed on a general-purpose pipelined computing platform are identified and instructions that make up these sequences are classified based on hardware utilization. Individual instruction classification results using a fine-grained classifier is also presented. A dynamic programming algorithm was applied to detect the boundaries of instructions in a sequence with a 100 percent accuracy. A unique aspect of this technique is the use of multiple power supply pin measurements to increase precision and accuracy. To demonstrate the robustness of this technique, power leakage data from ten target FPGAs programmed with a prototype of the pipelined architecture was analyzed and classification accuracies averaging 99 percent were achieved with instructions labeled based on hardware utilization. Individual instruction classification accuracies above 90 percent were achieved using a fine-grained classifier. Classification accuracies were also verified when a target FPGA was subjected to different controlled temperatures. The classification accuracies on discrete (ASIC) pipelined-architecture microcontrollers was 97 percent.

Index Terms—Side-channel analysis, power analysis, hardware security, instruction disassembly

1 Activity Tracking in Embedded Processors via Side-Channels

Assurance and *security* [1] of operations and data in microcrontrollers and embedded processors are essential for our rapidly growing electronic infrastructure. Lightweight, embedded processors are the basis of networking infrastructure. They are additionally used as part of smart sensors and actuators for industrial control systems and cyber-physical systems. Power management and boot processes for computer systems from phones, laptops, and servers are other applications. Each of these devices execute unique software/firmware and usually have an isolated task such as controlling the angle of an aircraft rudder by sensing, computing, and reporting the angle. The quantity and diversity of their usage represent a large attack surface, complicated by the need to support in-field updates. As the number of embedded systems and their connectivity increase, the implications of any compromise can be unpredictable and hazardous.

Going forward in the design of new computing building blocks of electronic systems, *security* and *assurance* must be a cornerstone. An representative effort to meet new diverse demands is the rising family RISC-V processors that have a with a range of configurations from a 9-stage pipeline processor used by Western Digital in hard-drives [2] to a two-stage pipeline design which Google is supporting for future products [3]. Also related to the direction of emerging embedded technology is the NIST lightweight cryptographic block competitions [4], which includes selection criteria based on *side-channel* vulnerability. However, assurance and security should be understood holistically, including general software execution. In work presented herein, techniques for analysis of *side-channels* related to activity within a microcontroller, driven by software execution, on a minimal computing building block, two-stage pipeline microcontroller, are presented. The dual goal is to examine the use of side-channels for *assurance*, and to understand the security of software execution in this building block.

Side-channels are unintended information pathways across layers of design abstraction [5] and represent *mutual information* between protected data or hidden activity and observable channels. Complex systems involve several layers, both within and across hardware and software. *Design abstraction* is required to allow parallel design and development that meets the competitive demands for rapid growth. However, Kocher *et al.* [5] formalized *side-channels*, vulnerabilities that arise when security is only addressed at each layer individually. While there is no doubt that the

• *The Authors are with the Department of Computer Science and Electrical Engineering, University of Maryalnd Baltimore County, Baltimore, MD 21250. E-mail: {deepakk1, zhengli1, robucci, nilanb, cpatel2}@umbc.edu.*

Manuscript received 15 Jan. 2020; revised 10 June 2020; accepted 26 July 2020. Date of publication 19 Aug. 2020; date of current version 8 Oct. 2020. (Corresponding author: Deepak Krishnankutty.)
Digital Object Identifier no. 10.1109/TC.2020.3018092

achievements in function and performance through design abstraction are impressive, addressing security remains a complex challenge tied to abstraction. As we address assurance and security within the building blocks of technology and develop practices to improve security across design layers, study of side-channels is key.

Side channels are exemplified by the analysis of power-supply consumption to infer secret data and, as presented further herein, to infer activity related to software execution. Particularly, Kocher *et al.* [5] famously described a linkage between power-supply transient consumption and activity within processing circuity, catalyzing a field of research. In that work, the authors presented a successful inference of secret keys involved in cryptographic processes using indirect yet accessible observations and analysis of power supply, called power-supply side-channels. In addition to inferring data, the authors point out the potential use of power-supply analysis to reverse-engineer computational processes, but generalized automated methods to accomplish this is an open research problem.

Several variations of attacks based on side-channel leakage have been explored including Power analysis attacks [5], Timing attacks [6], [7], Electromagnetic (EM) attacks [8], [9], Differential fault attacks [10], Scan-based attacks [11], Cache-based attacks [12], [13], Bus-snooping attacks [14], [15], Acoustic attacks [16], [17], etc. In one side-channel based technique [18], the authors use power profiling of a Device Under Attack (DUA) to locate trojans using a circuit partitioning technique. EM attacks against public-key algorithms have been demonstrated in [19]. The papers [20], and [21] discuss EM based secret-key recovery from RSA implementations. The authors, in [22], introduce a practical technique to measure EM emanations for instruction-level events based on a metric called SAVAT (Signal Available to Attacker). Although this technique can be useful in distinguishing on-chip from off-chip memory access instructions such as load/store instructions, fine-grained instruction level profiling is difficult from EM emanations. The methodology and techniques presented in [23], and [24] by Baki, Yilmaz *et al.*, cover EM based side-channel techniques that distinguish instructions from a pre-determined instruction sequence or micro-benchmark. EM based side-channel analysis, however, is not generically applicable for precise determination of instruction-level events. In this work, a methodology is established, one that is capable of retrieving information regarding instruction-level events via side-channel leakage. The issue of detecting instructions executed on an MSP430 microcontroller from its power side-channel leakage, was approached by implementing this methodology and yielded reliable results. In [23], [25], and [24], statistical models of code were used to infer instruction execution, and in some cases, models are specialized to the expected code. In this work, the information channel is explored, assuming no prior statistics about the code execution. Priors may not always be representative of what is actually being executed and therefore can adversely bias results for the characterization of abnormal code.

In [26], the authors propose an add-on monitoring system (WattsUpDoc) that non-intrusively monitors power consumption signals to detect the presence of malware on state-constrained embedded systems. The paper presents a coarse-grained approach to detect malicious code, utilizing machine learning techniques to discover patterns within the power-consumption waveform of a sequence. Numerous countermeasures have also been proposed to defend against these attacks by means of introducing preventive measures to obfuscate the leakage [27], [28], [29].

Studies into instruction level power leakages were proposed in [30], [31]. However, these studies did not account for instruction-level power leakage signatures and instead explored models that optimized the total power leakage of processors. Recently, researchers developed a tool (ELMO) for modeling instruction-level leakages [32]. ELMO models the power leakage only for a subset of the ARM instructions primarily selected for use in cryptographic algorithms. The model achieves a classification accuracy of 100 percent when the subsets of instructions were classified into five broad groups. This study also accounted for the data dependency among sequences of instructions being analyzed. A side-channel leakage based disassembler that incorporated Hidden Markov Models (HMM) with a prediction accuracy between 35 and 70 percent for individual instructions was proposed in [33]. Power data from a PIC16F687 microcontroller having a two-stage pipeline with 35 instructions was discussed. The disassembler leveraged profiling of existing code to generate models and then detected the same instruction sequences using these models. In [34] and [35], the authors present an instruction classifier with 100 percent accuracy when the same instruction was compared against the templates generated for that particular instruction. Measurements were derived from an ATMega163 microcontroller which has two pipeline stages. Templates generated from power-supply side-channel leakage data from a small subset of the instruction set architecture (ISA) are classified. In [36], a disassembler that utilizes Kullback-Leibler divergence and dimensional reduction using PCA was proposed. The authors report that the disassembler recognizes test instructions with 99.03 percent success. They also present some analysis of detecting register names.

The above-mentioned works do not fully address some key challenges of multi-clock-cycle instruction sets (discussed further in Section 2) and the identification of instruction sequences being executed on the processing platform. The number of clock cycles required to execute an instruction is referred to as Clock Cycles Per Instruction (CCPI). Furthermore, building classifiers and using those to classify individual instructions against themselves, even with close to 100 percent accuracy, does not necessarily translate into a highly accurate instruction recovery and identification for unknown code. In FISCAL [37], the authors of this work presented a method to identify instruction sequences on an instance of the openMSP430 synthesized on an FPGA. Owing in part to the great number of MSP430s deployed in both industrial and commercial applications and its versatility in terms of its ultra-low power capabilities [38], this architecture is further explored in this work.

In this paper, the mutual information between power supply consumption and code execution is established to represent a fundamentally useful side-channel to a high-level abstraction of system behavior that can be leveraged as a foundation for new methods for both attacking and protecting systems. In particular, how an attacker can use this

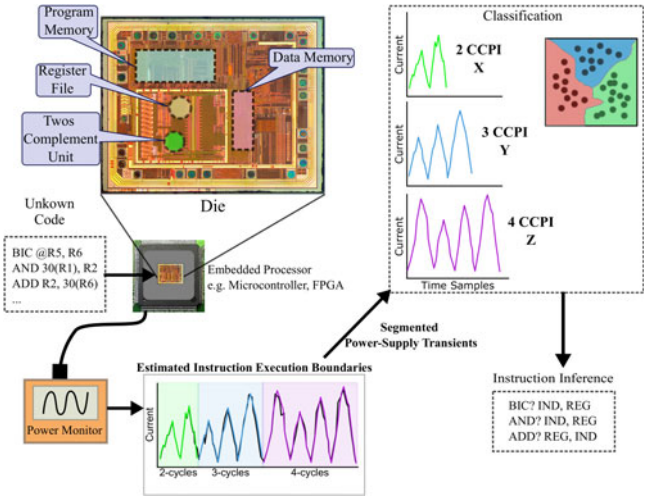


Fig. 1. A sequence of instructions being executed on a microcontroller is processed by a number of built-in hardware units. Power-supply side-channel leakage is observed via the power pin on the microcontroller. Power templates representing instructions are created and a dynamic programming technique is used to match the templates to power-supply leakage observations corresponding to the code sequence being executed on the microcontroller. Once the instruction execution boundaries are determined, a hardware-based classification of the observed instructions is performed. In the example, a sequence of CCPI, $< 2, 3, 4 >$, and hardware utilization bin labels, $< X, Y, Z >$, is inferred.

information to monitor code execution is demonstrated as well as countermeasures that can detect clandestine code execution. In solving this problem, instruction classification was investigated; instruction execution boundary determination was solved; and a coarse-grained classification method to efficiently perform segmentation was created. The method proposed, as illustrated in Fig. 1, is useful for a number of applications including efficient classification of code without exact disassembly (coarse-grained classifier) and working with larger code sequences to not only analyze but also find regions of critical code execution for attack or protection schemes. The proposed technique can be the foundation of new and improved attacks, and be used to search for and identify code as a precursor to other attacks. For instance, the proposed algorithm can be used to find where code is executed in order to launch an attack or as a directory to perform advanced code timing attacks.

Research Contributions. The design and implementation of our system present the following research contributions.

- A novel power-supply side-channel technique that uses two training loops and dynamic programming without requiring statistical assumptions about code being executed to infer instruction boundaries from observations of code execution on a *pipelined multi-cycle* general-purpose microcontroller, and demonstrated experimentally the ability to characterize and identify code.
- A low-overhead, coarse-grained SVM-based classifier to determine the hardware unit being utilized within a processor.
- A fine-grained SVM-based classifier that identifies actual instruction-level details.
- Demonstrated robustness and repeatability of both classifiers, by running analysis on power-supply traces collected from four power pins on 10 Spartan3 FPGAs that are programmed with a soft-core MPS430 as well as a single power pin on 4 discrete MSP430 chips.
- Demonstrated robustness to temperature variations by subjecting the FPGA package to multiple controlled temperatures.

2 Power-Supply as a Side-Channel for Code Disassembly

As compared to hardware implementations, software represents a level of abstraction very disparate from the low-level circuits generating power-supply signals, and thus inference involves additional complexity. A general purpose processor's hardware is complex and designed to support many possible operations at any given time. Challenges arise when directly modeling the relationship of software and power-supply signals, and are complicated by unknown timing of operations. Machine-learning techniques to infer timing and activity, grounded in basic knowledge of hardware and power-supply transients as well as architecture, are used in this work to overcome modeling challenges.

As mentioned in Section 1, a key challenge in power-supply-based disassembly for many types of processors is that they not only have multiple clock-cycles per instructions, but also that the number of cycles per instruction varies. In general, software is made of a sequence of instructions with varied and unknown CCPI. While some processors use instructions that are mostly 1 CCPI, and others have a larger but moderately varied CCPI, many processors have a highly variable CCPI (as illustrated in Fig. 1). This uncertainty inherently complicates side-channel based disassembly since knowing the clock signal cannot be used alone to infer when instructions to be classified begin and end. In Section 6, a solution to the problem of finding execution boundaries of instructions within a sequence is presented. Once instruction execution boundaries are determined, the power-supply transients within the intervals between can be used to infer which instruction was likely executed during that time. For a given CCPI several instructions are likely, and thus a classifier is built for each CCPI group.

The predominant components of power-supply transients as a side-channel arise from how transistor-based digital circuits operate. Digital computation involves transitioning values between logical 1 and 0 values, and each such transition represents a charge transfer from or to a power-supply connection, e.g., power pin. While some approaches to model power-supply transients from a gate-level abstraction involve accounting for the set of transition events, along with a minimal set of attributes about the physical circuit, e.g., load capacitance, in practice, however, the problem is complex, with factors such as R-C attenuation in the power grid and nonlinear interaction of circuits, each dependent on the location of gates and power-pads [39]. This complexity and difficulty to simulate or derive low-order empirical models led us to investigate this problem through experimentation and data-driven analysis. Inference models are built by data as well as knowledge of a hardware processor. In general, instruction power-supply traces vary based on (i) hardware use and options (ii) data (iii) other instructions in the pipeline. Herein a training framework is presented for creating a

robust classifier in this context, including data and analysis across several dies and temperatures.

3 Security Model

Two scenarios are of interest, wherein for one an attacker seeks to discover and exploit obscured code, and in the other an attacker seeks to modify code without detection. In the first scenario, a system manufacturer assumes code obscurity (security through obscurity), yet an attacker has access to a microcontroller to measure power-supply transients and therefore can infer operations, algorithms, and specific segments of vulnerable code using our technique and thereby stage further exploits and attacks (e.g., timing attacks, bug exploits). Knowledge of our technique therefore should be considered going forward in the design of systems, and appropriate countermeasures should be developed. Further discussion on reverse-engineering is provided in Section 10. In the second scenario, code execution assurance is desired; side-channel leakage can serve to build a system that detects malicious code insertions or modifications. In this section, the adversary model, threat model, and trust assumptions are described for the scenario wherein the side-channel leakage information is used to provide assurance in the system.

3.1 System Hardware Assumptions

The System Under Test (SUT) is a processing unit which is a low-power micro-controller architecture with multiple pipeline stages. The micro-controller architecture supports instructions that require a variable number of execution cycles within a pipeline stage.

3.2 Adversary and Threat Model

The adversary

- has physical or remote access to the SUT.
- has knowledge of processing unit architecture.
- has apriori access to the code or can retrieve it from the program memory.
- may modify existing code by re-arranging or inserting instructions in the program memory.
- has the ability to manipulate sensitive information in the data memory.

Modification to code and/or data memory can go undetected by other countermeasures implemented in the system

3.3 Trust Model

We make the following trust assumptions regarding the user of the SUT, the provider of the power monitoring system and the manufacturer of the processing unit(s) used in the SUT.

- The user and provider of the power monitoring system trusts that the manufacturer provides a discrete-chip or a soft-core for the processing unit, that adheres to the implementation and Instruction Set Architecture (ISA) specifications.
- The user and the manufacturer trusts that the provider of power monitoring system does not alter the original specifications of the processing unit.
- The user trusts the provider of the power monitoring system to report the instruction execution sequence and the type of each instruction.
- The user trusts the provider of the power monitoring system to ensure that it is non-intrusive and does not alter the normal operation of the SUT.
- The user trusts that the processing unit and other hardware components of the SUT cannot be altered after final system implementation.

4 Side-Channel Disassembly Overview

Given the physical implementation of computation as described in the previous section, it is apparent why power supply fluctuations are indicators of computational activity and can be used to infer instructions. An observation of a power trace during instruction execution can be used to infer which instruction was being executed by the processor. However, if observing a processor with a highly variable number of cycles per instruction, knowing the actual begin and end times in order to define the temporal boundary for classification is not trivial. Therefore a solution to identifying instruction boundaries is required so that the power-supply transient data can be temporally segmented in alignment with actual instruction execution.

The presented solution involves inference of instructions based on the execution cycles (excluding fetch cycles), and so a set of instruction execution boundaries (IEB), defined as the times in terms of clock-cycles of the start of an instruction execution stage, are first established. Clock-cycles not in this set are a continuation of execution. The number of possible answers for a sequence of $\mathcal{K}$ cycles is $2^{\mathcal{K}-1}$ assuming that the start of observation coincides with the beginning of the first instruction execution. If the power trace examination is advanced forward in time, each subsequent IEB defines the proceeding segment of instruction execution after the preceding IEB. The number of clock-cycles in this segment limits which sets of instructions are potentially being executed since there are varying clock-cycles per instruction execution (CCPI). The cost of this decision is attributed to how mismatched the observed power trace is from any template traces for the given CCPI. The identification of each IEB also defines the optimization subproblem of determining remaining IEB using the power trace.

For each CCPI (1-6), one or more representative templates can be used. One approach would be to build a template for each possible instruction (152 for the MSP430), allowing the best match to define the cost for a particular segment. Another approach would be to provide a well-suited template set designed to solve the segmentation problem. An intuitive choice is to provide a few representative templates for each CCPI group, and in the presented approach, representative sets were empirically built based on knowledge of discernible types of hardware use.

Once the temporal segmentation has been performed to prepare the power-supply segments, instruction inference can be performed within a single CCPI group. The number of such instructions per CCPI varies. Herein, this classification for which each instruction belongs to its own class is called a fine-grained classification and is discussed in Section 9. However, a less complex, coarse-grained classifier,

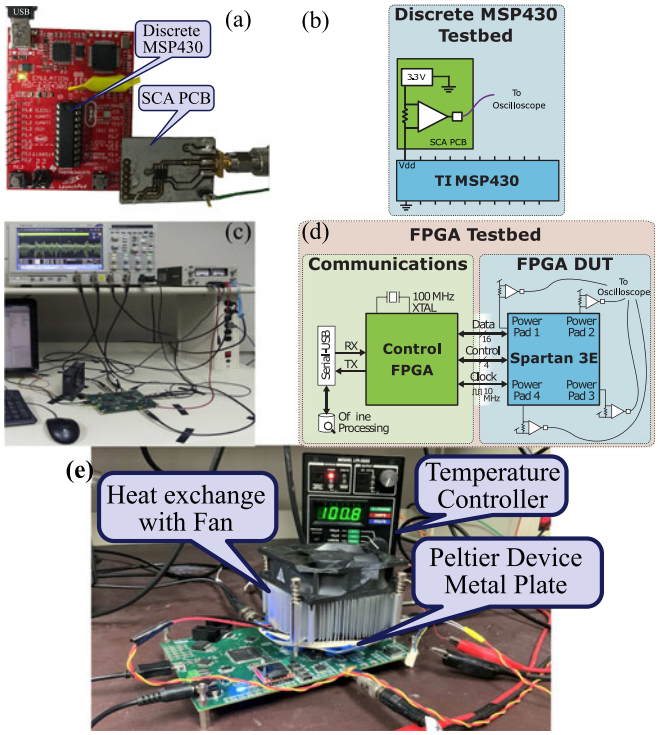


Fig. 2. The MSP430 LaunchPad along with a custom-designed board for power supply measurement is shown in the figures (a) & (b). The power pin is sampled using a high-bandwidth oscilloscope and analysis on the captured power traces is performed offline. Custom-designed board using FPGAs for communication/control and testbed [37] is illustrated in (c) & (d). Temperature setup with heat exchange and fan is shown in (e).

for which multiple instructions belong to a class based on common hardware usage traits is useful for detecting modifications in the code being executed. The coarse-grain instruction representation as used for the dynamic programming solution is related to this classifier and discussed in Section 7.

It is important to note that the core of this technique does not require the actual deployed code for any part of the training, nor does the instruction inference rely on prior expectation of code being executed, unlike [22]. In specific problems where the exact prior code is known, such information could be used to augment our technique, but the intent of this work is a solution to a harder problem of extracting useful knowledge about the code being executed when the exact code or compilation is not known. A solution to this foundational problem has many applications including non-intrusive detection and analysis of malicious code insertions, as well as understanding potential vulnerabilities of code execution to side-channel analysis. The technique is applied to well-established code modules (e.g., 128-bit AES encryption and PID controller) prediction accuracy is presented.

5 Experimental Hardware Setup

To evaluate the robustness of the proposed technique, experimentation was performed on both a discrete TI MSP430 microcontroller LaunchPad [40] and a *soft-processor* implementation of an open-MSP430 instantiated on an FPGA [41]. In this section, the experimental setup for power measurement on both platforms is detailed.

For experimentation with a discrete microcontroller, a custom auxiliary board (Figs. 2a and 2b) was designed and fabricated to measure power-consumption data from an MSP430G2553 chip housed on a TI LaunchPad kit. The custom PCB includes an on-board regulator (3.3V) to power the LaunchPad. An on-board amplifier (powered independently) amplifies the power signal which is high-pass filtered (cutoff frequency of 1MHz) and then passed through a coax cable to an oscilloscope.

For experimentation with an FPGA-based *soft-processor* implementation, a custom board (Fig. 2c) was designed and fabricated that both houses FPGAs and includes power measurement circuitry. The board comprises one control FPGA, which facilitates timing and controls for experimentation, and a Xilinx Spartan 3E as the Device Under Test (DUT) (Fig. 2d). The design macro, an openMSP430 [41] is instantiated on this FPGA. The control FPGA functions as the control/communication device to convey debug and control data to the DUT. All four power-supply pins available on a TQFP package of the DUT are simultaneously and separately measured producing four waveforms. Power-supply current probing is performed using 1Ω resistors in the supply path for each power pin. The voltage across each resistor due to the current consumption of the device is probed using a voltage buffer and passed through an amplification-stage before passing through coax cables to an oscilloscope. It should be noted that re-synthesis or replacement changes measured results, so the same bitstream was used for each subsequent FPGA (Section 7.2). Two versions of the board were developed and tested, one with a directly soldered DUT and a socketed version to show results across many FPGAs. For reliability analysis under varying temperatures, the socketed version of the board was fit with a Peltier module (operating temperature range: 0°C-80°C), a metal interface plate with thermistor sensor holes providing feedback to a multimeter that provided temperature readings (Fig. 2e). Temperature on the Peltier device was modulated using an external temperature controller. Although in this work an MSP430 soft-processor was synthesized on the DUT, the platform has been designed to prototype multiple processor architectures, as well as hardware-based computation macros (e.g., encryption cores, PID controller).

For both setups, the processor was driven with a 10 MHz clock. The power supply was sampled using a Tektronix *DPO7354C* oscilloscope at 500MSPS, which resulted in 50 samples per processor clock-cycle. A 2.5GHz bandwidth-limit setting was used.

6 Mixed-Instruction Sequence Analysis

The primary requirement towards the goals of assurance and security is to be able to identify instruction boundaries followed by classification. The MSP430 ISA consists of instructions that use 1 to 6 *clock-cycles per instruction* in the execute (EXE) stage, as specified in the MSP430 User Guide [42]. The processor's word access instruction set is segmented into six mutually exclusive sets, each respectively labeled N-CCPI, where $N = \{1, 2, 3, 4, 5, 6\}$. Each instruction power profile is labeled $I_{N,i}$, corresponding to the ith instruction in the set N-CCPI. The number of instructions per CCPI set is 16 (1-CCPI), 44 (2-CCPI), 22 (3-CCPI), 19 (4-CCPI), 39 (5-CCPI), and 12 (6-CCPI).

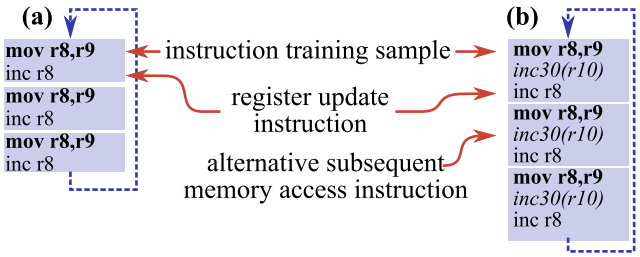


Fig. 3. Three-iteration training loops: (a) Interleaved register access instruction (b) Interleaved memory and register access instructions. These account for data-dependencies and influence of nearby instructions.

6.1 Generation of Robust Training Data

To ensure robustness of the classifiers, a training set with representative variations of each instruction must be generated. The power-supply fluctuates according to several factors including the following: (i) instruction operation (ii) addressing modes (`reg`, `mem`, `imm`, `symbolic`); (iii) data (i.e. data dependency); and (iv) other instructions in the pipeline.

To account for data-dependencies and influences of nearby instruction types in the sequence, a training loop was developed that included three instances of the instruction under characterization and data was varied within and between the loop iterations as shown in Fig. 3. A *reg* and a *mem* instruction were interleaved into each sequence. There was a difference among the 1^{st}, 2^{nd} and 3^{rd} instance of the instruction within each loop, with the power profile for the first instance showing the most variance. This variance can be attributed to the fact that the preceding instruction for the first instance is different from the other two. Classification using training samples from the 2^{nd} and 3^{rd} instance independently provided similar accuracy to use of samples from the 1^{st} instance. In the aforementioned case, testing samples were selected randomly from three instances of the instruction, with power profiles acquired during an independent run of the experiment. In this work, data from all observations, was used to create the training sample pool and selected training samples randomly.

The training set that was constructed included each non-emulated word-mode instruction in the MSP430 instruction set. To account for clock jitter within power captures, a cross-correlation is performed on power samples for each loop iteration. Waveforms created from an average of 10 000 single-pin power measurements of instructions are depicted in Fig. 4, for 1-CCPI and 4-CCPI. The figures show that a peak and valley in the power profile for an instruction correlate to one instruction execution cycle for the data from an FPGA device. In the template building phase, we utilize the

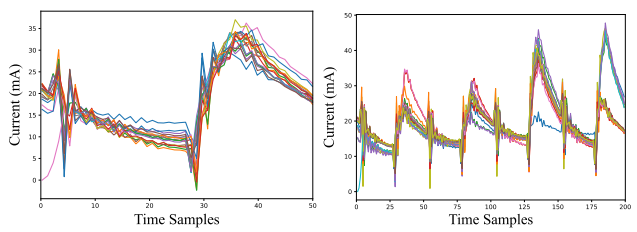


Fig. 4. Cases (a) and (b) depict the raw power profile averages for various clusters of 1-CCPI and 4-CCPI. (a) Power profiles for 1-CCPI templates comprising 16 different instructions. (b) Power profiles for 4-CCPI templates comprising 19 different instructions.

TABLE 1
Hardware Usage Classes With Their Description and Instruction Count

Label	Interpretation (Associated Traits)	N-CCPI					
		1	2	3	4	5	6
reg_reg (RR)	Source Register, Destination Register	10					
reg_reg_sub (RRS)	Source Register, Destination Register Includes Arithmetic and logical instructions	5					
mem_mem_sub (MMS)	Source Memory, Destination Memory (Complement hardware involved - includes 'bic' instruction)					9	4
mem_mem_nosub (MMNS)	Source Memory, Destination Memory (No complement hardware)					9	8
mem_idx_sub (MIS)	Source Memory, Destination Index (No complement hardware)					6	
mem_idx_bic (MIB)	Source Memory, Destination Index 'bic' (No complement hardware)					2	
mem_idx_nosub (MINS)	Source Memory, Destination Index No Subtraction (No complement hardware)					9	
reg_const_ind_sub (RCInS)	Source Register or a constant, Destination Memory Destination includes a constant and involves indirect addressing				3		
reg_const_ind_nosub (RCInNS)	Source Register or a constant, Destination Memory Indirect addressing at source, No subtraction hardware				4		
reg_const_ind_push (RCInP)	Source Register or a constant, Destination Memory Push Instruction Indirect addressing at source, No subtraction hardware				2		
reg_ind_nosub (RInNS)	Source Register, Destination Memory Indirect addressing at source, No subtraction hardware				9		
ind_reg (InR)	Source Memory, Destination Register Indirect addressing at source		24	10			
ind_reg_log (InRL)	Source Memory, Destination Register Indirect addressing at source includes logical op			2			
mem_reg (MR)	Source Memory, Destination Register Includes Arithmetic and logical instructions		8				
const_reg (CR)	Source is a generated constant, Destination Register Includes Arithmetic and logical instructions	1					
imm_reg (ImR)	Source involves a constant, Destination Register Includes Arithmetic and logical instructions		12	10			
imm_ind_sub (ImInS)	Source involves a constant, Destination Memory Indirect addressing involving constant at destination					4	
imm_ind_nosub (ImInNS)	Source involves a constant, Destination Memory Indirect addressing involving constant at destination, No complement hardware				1		

salient deviations among the power profiles of instructions requiring the same number of clock-cycles to determine a proper fit for the classification model.

6.2 Hardware-Utilization-Based Labeling

Within each N-CCPI group, we empirically group instructions based on hardware utilization and choose to define a class label for each group, Table 1. Hardware utilization, for example, includes addressing mode (e.g., memory, register), computational operation (e.g., multiplier, twos-complement unit), and status register updates. As shown under the column heading N-CCPI, instructions in a given N-CCPI group can be labeled with different hardware utilization. For example, within 1-CPPI `RRC` was included in the group `const_reg` while the remaining 15 were in the `reg_reg` and `reg_reg_sub` groups.

6.3 Determining IEBs in Complex Instruction Sequences

In this subsection, two main steps for IEB determination for random instruction sequences, with instructions of heterogeneous CCPI, are discussed. The first goal of determining the IEB is to correctly segment the power supply waveform. A correct segment is defined by a true starting and a true ending execution clock-cycle (Fig. 5a), such that the corresponding epoch comprises exactly one instruction. Initially, every possible valid window from length 1- to 6- clock-cycles must be assigned to determine whether it is the correct segment. This involves extracting overlapping windows (i.e. sliding window) for evaluation. Windows overlap by a unit step of 1-clock cycle for each of the lengths, 1-6, as depicted in Fig. 5b.

In [37], the authors presented the algorithm, summarized in Algorithm 1, to determine a likely sequence of IEBs in an instruction sequence based on an optimal fit of instruction templates to build a waveform to match the observed data. The observed test data for a sequence is defined to be the average power-supply transient of 5 000 observations of the sequence execution. The power traces for test sequences need to be sampled more than once, otherwise the same accuracy is not guaranteed. In an experimental sweep, it was possible to maintain accuracies above 90 percent for the

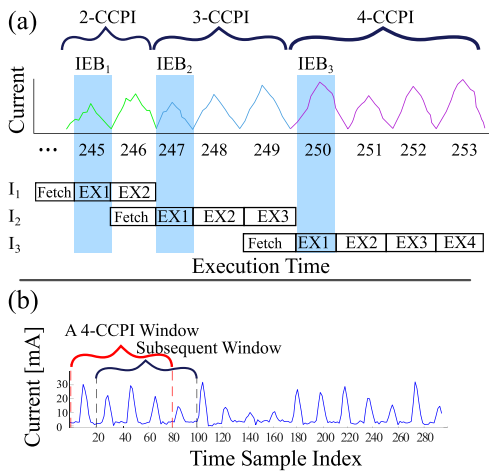


Fig. 5. Given the power profile for a test sequence, the first step is to locate the last clock cycle of execute stage for each instruction: Instruction Execution Boundary (IEB) (specified by EX1, EX2 etc. for execution cycle 1, execution cycle 2 respectively in the instruction pipeline). In the example shown (a), the sequence of CCPI is identified, i.e. 2, 3, and 4 for instructions I1 - BIC @R5,R6, I2 - AND 30(R1), R2, and I3 - ADD R2,30(R6), once the IEBs are determined at the 245th, 247th, and 250th clock-cycle. In (b), the sliding window scheme is shown for a 4-CCPI where the next subsequent window is shifted by 1-clock-cycle.

coarse-grained classifier, with a lower bound of 1 000 traces, below which the accuracies dropped. For the experiments presented in this work, the test power transients are averaged over 5 000 traces. The applicability of the technique presented in this paper is to sequences that are executed more than once, such as in a loop.

The IEB identifications involve solving two problems. The first is deciding the optimal tagging of Instruction Execution Boundaries (IEB) in a clock-cycle sequence c_1, c_2, ..., c_n. For any solution, it is assumed that the instruction starts in the first clock-cycle, so c_1 is always an IEB. As an example, for a 2-clock-cycle sequence choice, a choice for the set for labeling as IEBs could be $\{c_1, c_2\}$, meaning that there are two instructions, starting at clock-cycle 1 and clock-cycle 2 and are each 1-clock-cycle instructions. The alternative choice is $\{c_1\}$, meaning that there is only a single 2-CCPI. For a 3-CCPI observation the choice could be $\{c_1\}$, $\{c_1, c_2\}$, $\{c_1, c_2, c_3\}$, or $\{c_1, c_3\}$. These decisions correspond to CCPI sequences of (3), (1,2), (1,1,1) and (2,1). In general, the search space for the optimal tagging of IEBs is of size $2^{\mathcal{K}-1}$ where $\mathcal{K}$ is the number of observed instruction clock-cycles. It can be reasoned that determining the IEBs is the same as determining an N-CCPI sequence.

The second problem is in determining the cost function for deciding the optimal labeling of IEBs, and thus the sequence of clock-cycle lengths, $\vec{L} = (L_1, \ldots, L_k)$. For each entry L_n, in the series $\vec{L}$, a clock-cycle offset $\mathcal{O}_k$ can be determined from the summation of the previous values, $\sum_{n=1}^{k} L_n$. Taking k to be the number of power supply waveform samples taken per clock-cycle, the cost of each label is the minimum of the total squared distance between any template waveform in the template book that comprises $N \times k$ samples and the observed power supply waveform over the samples $\mathcal{O}_j \times k + 1$ to $(\mathcal{O}_j + L_j) \times k$. In other words, for every possible span of clock-cycles, the minimum cost is the squared sum distance to the best matching function in the template book. The solutions to these minimal cost computations are precomputed and labeled as $M_{\mathcal{O},\mathcal{N}}$, which denote the cost of deciding there is an instruction of clock-cycle-length N at clock-cycle offset $\mathcal{O}$:

$$M_{\mathcal{O},N} = \min_{\forall i \in 1..length(C_N)} \sqrt{\sum_{n=0}^{k \times l} (T_{N,i}[n] - D[\mathcal{O} \times N + n])^2}, \tag{1}$$

where k is the number of power supply data points taken in a clock-cycle; $T_{N,i}[n]$ is data point n of the ith template in a book of template waveforms for each instruction that takes N clock-cycles; $D[n]$ is data point n in the captured power supply waveform; and thus $M_{\mathcal{O},N}$ represents the minimum euclidean distance from an instruction with clock-cycle offset $\mathcal{O}$ and clock-cycle-length N. With these pre-computed results, a dynamic programming solution can be used to decide the optimal sequence of clock-cycle lengths.

Starting at the first position, the challenge is to decide the optimal choice of instruction length N and thus the second instruction-initiating-cycle label. The cost of a given decision for N is the summation of the cost $M_{1,N}$ and the cost of the remaining optimally decided labels chosen from clock-cycle c_{N+1} onward that includes the IEB labeling on c_{N+1}. Thus each decision introduces the subproblem of computing the remaining labels and their cost, which represents a solution to an *optimal-substructure* [43] to find the best overall solution. The number of choices for $M_{2,N}$ depend on the size of the set of possible clock-cycle-lengths. The decision with the lowest sum cost is chosen. Among the $2^{\mathcal{K}-1}$ possible choices sets of IEBs, a given clock-cycle can be labeled as an IEB c_j in multiple possible sets. Labeling of c_j represents a subproblem of an *optimal-substructure* solution beginning at that clock-cycle. Therefore, *overlapping subproblems* [43] exist in choosing between overall solutions, making the overall problem solution appropriate for dynamic programming. For a given clock-cycle offset, there is a minimum distance-based cost (sum squared error, COST^2, where $\mathrm{COST}(\mathbf{P}, \mathbf{x}, \mathbf{y}) = \|\mathbf{P}^T x - \mathbf{P}^T y\|_2$) in representing the remaining waveform using template waveforms in the collected dictionary. This minimum cost is denoted $C_{\mathcal{O}}$. Let $\mathcal{K}$ be the total number of clock-cycles in the overall observed sequence. Then, the subproblem may be solved recursively as

$$C_{\mathcal{O}} = \begin{cases} \min\limits_{\forall \mathcal{N} \in 1..\mathcal{K}-\mathcal{O}+1} M_{\mathcal{O},N} + C_{\mathcal{O}+N} & \mathcal{O} \leq \mathcal{K} \\ 0 & \text{otherwise} \end{cases}. \tag{2}$$

The recurrence relation in (2) for $C_{\mathcal{O}}$ allows for the *memoization* of optimal solutions to subproblems $C_{\mathcal{O}+N}$, where $C_{\mathcal{O}}$ *remembers* the most current optimal solution to a subproblem. The optimal solution is available at $C_{\mathcal{K}}$. For readability, clock-cycle offsets $\mathcal{O}_i$, $\mathcal{O}_j$ and $\mathcal{O}_l$, are represented by i, j and l respectively. *split* and *psplit* are temporary placeholders where s holds the indexes for *substructures* utilized during the memoization step to determine the optimal solution. L_{seq} is the clock-cycle length for the instruction sequence being observed. L_{win} is the maximum clock-cycle window length possible for the current instruction set. The optimal sequence of IEBs is available from *optimalIEB*.

Algorithm 1, determines the optimal sequence of IEBs for an instruction sequence by evaluating the recurrence relation defined in (2).

Algorithm 1. Optimal IEB Sequence Determination

procedure DETERMINEOPTIMALIEB(M)
for $i = 1$ to L_{seq} ▷ Initialization of cost matrix **do**
  for $j = 1$ to $i - 1$ **do**
    $C_{i,j} = \infty$
  $j = i$
  while $j < L_{seq}$ **and** $j < L_{win} + i$ **do**
    $C_{i,j} = M_{i,(j-i+1)}$
    $j = j + 1$
  while $j < L_{seq}$ **do**
    $C_{i,j} = \infty$
    $j = j + 1$
for $l = 2$ to L_{seq} **do** ▷ Dynamic programming
  for $i = 1$ to $L_{seq} - l + 1$ **do**
    $j = i + l - 1$
    for $s = i$ to $j - 1$ **do**
      if $C_{i,s} + C_{s+1,j} < C_{i,j}$ **then**
        $C_{i,j} = C_{i,s} + C_{s+1,j}$
        $split_{i,j} = s$ ▷ For recovery
$optimalIEB_1 = split_{1,L_{seq}}$ ▷ Recover optimal path
$index = 2$
$i = split_{1,L_{seq}} + 1$
$psplit = split_{1,L_{seq}}$
while $split_{i,L_{seq}} \neq 0$ **do**
  $optimalIEB_{index} = split_{i,L_{seq}} - psplit$
  $psplit = split_{i,L_{seq}}$
  $i = split_{i,L_{seq}} + 1$
  $index = index + 1$
$optimalIEB_{index} = L_{seq} - psplit$

7 Coarse-Grained Classification

In this section, one-instruction classification using Support Vector Machine (SVM) is evaluated. Evaluation of Algorithm 1 for classifying instruction sequences is also presented. In Table 1, instructions are binned into 18 coarsely defined classes based on hardware utilization. These are further categorized into six groups based on CCPI. In the training phase, power traces for every instruction, are captured on the oscilloscope for each loop as explained in Section 6 using Fig. 3. The number of such power traces captured is proportional to the number of loop iterations (7 000) and instances of an instruction within a loop sequence. For example, per Table 1, a five-clock-cycle, immediate-to-indirect addressing instruction is uniquely labeled and requires 61 clock cycles per loop iteration. With three instances of an instruction being observed per iteration, there are 5 050 instances of this instruction in each capture. Hence, two power captures on the scope will fulfill the requirement for classification.

The power templates are then categorized into classes based on hardware utilization (Table 1) and CCPI. To compare inter-pin relationships with prior observations, we construct an aggregate template (`Agg.`) by combining templates from each power pin. As a result, there are four times the number of data points for a template as compared to an individual pin. In addition, we determine $\mathcal{P}_{sum}[n] = \sum_{i=1}^{Npad} \mathcal{P}_i[n]$, where $Npad$ (=4) is the number of power supply pins. $\mathcal{P}_i[n]$ is the power data from power supply pin i at data point n, $\mathcal{P}_{sum}[n]$ is the *power data sum* result (`PSUM`) from all power supply pins at data point n. Additional power supply transients are acquired for a testing dataset of size 3 000 samples for each category and instructions from each loop in Fig. 3. SVM classification per CCPI group is then performed over training and test power templates.

TABLE 2
Confusion Matrices for N-CCPI Instructions After SVM Classification

(a) 1-CCPI Classification (%)

DUT	Actual Class	Inferred Class: CR	RR	RRS
F	CR	99.38	0.23	0.39
	RR	0.00	100.00	0.00
	RRS	0.17	0.00	99.83
D	CR	100.00	0.00	0.00
	RR	0.02	99.98	0.00
	RRS	0.00	0.00	100.00

(b) 2-CCPI Classification (%)

DUT	Actual Class	Inferred Class: ImR	InR	MR
F	ImR	99.99	0.00	0.01
	InR	0.00	100.00	0.00
	MR	0.00	0.00	100.00
D	ImR	99.99	0.01	0.00
	InR	0.00	100.00	0.00
	MR	0.00	0.07	99.93

(c) 4-CCPI Classification (%)

DUT	Actual Class	Inferred Class: RCInNS	ImInNS	RCInP	RInNS	RCInS
F	RCInNS	100.00	0.00	0.00	0.00	0.00
	ImInNS	0.00	100.00	0.00	0.00	0.00
	RCInP	0.00	0.00	100.00	0.00	0.00
	RInNS	0.00	0.00	0.00	100.00	0.00
	RCInS	0.00	0.00	0.00	0.00	100.00
D	RCInNS	99.85	0.00	0.00	0.00	0.15
	ImInNS	0.00	100.00	0.00	0.00	0.00
	RCInP	0.00	0.00	100.00	0.00	0.00
	RInNS	0.00	0.00	0.00	100.00	0.00
	RCInS	0.01	0.01	0.00	0.01	99.97

(d) 5-CCPI Classification (%)

DUT	Actual Class	Inferred Class: MMS	MMNS	MIB	MInS	MINS	MIS
F	MMS	100.00	0.00	0.00	0.00	0.00	0.00
	MMNS	0.00	100.00	0.00	0.00	0.00	0.00
	MIB	0.00	0.00	100.00	0.00	0.00	0.00
	MInS	0.00	0.00	0.00	100.00	0.00	0.00
	MINS	0.00	0.00	0.00	0.00	100.00	0.00
	MIS	0.01	0.00	0.00	0.00	0.00	99.99
D	MMS	99.84	0.16	0.00	0.00	0.00	0.00
	MMNS	0.00	100.00	0.00	0.00	0.00	0.00
	MIB	0.00	0.00	100.00	0.00	0.00	0.00
	MInS	0.00	0.00	0.00	100.00	0.00	0.00
	MINS	0.03	0.00	0.00	0.00	99.97	0.00
	MIS	0.13	0.00	0.00	0.00	0.00	98.87

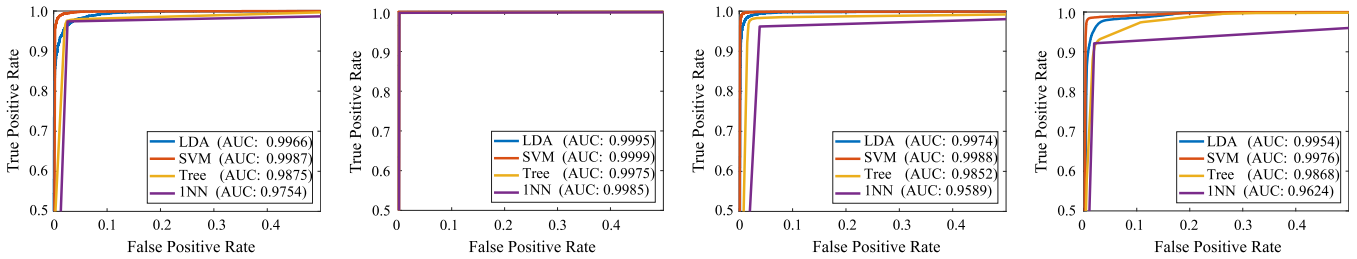
Cases (a)-(b) show classification results for 1-, 2-, 4-, and 5-CCPI. Aggregated results from all four pins of the FPGA as the DUT is indicated under the letter 'F'. The letter 'D' for results from discrete MSP430 chip.

7.1 Cross-Validation of Coarse-Grained Classifier

A *10-fold cross-validation* was performed over a data set size of 10 000 samples for every label based on hardware utilization. This includes all instructions in the N-CCPI group. Classification is performed using a subset of the training data collected from the devices as described in Section 6. The dataset for cross-validation comprises of 100 000 samples of power data acquired from each loop in Fig. 3 for every label in an N-CCPI group. Hence, there is an entry from each loop in Fig. 3 per sample in the dataset. In Table 2 (a-d), confusion matrices for 1, 2, 4, and 5-CCPI groups are presented based on data acquired from four power pins when the DUT is an FPGA (designated *F*) and when data is from the single power supply pin on a discrete MSP430 chip (designated *D*). For 3 and 6-clock-cycle instructions, the average classification accuracy is 99.99 percent for both cases, when data is from the FPGA. These accuracies are 99.97 percent and 100 percent respectively when the DUT is a discrete MSP430 chip. The confusion matrices in Table 2 (a-d) demonstrate high levels of classification accuracies for classifying instructions based on hardware utilization.

7.1.1 Classification Techniques

In [37], the nearest neighbor algorithm is utilized to classify instructions to different groups of hardware utilization based on the Manhattan distance (l_1 norm) to the templates



(a) 1 CCPI Classification (b) 2 CCPI Classification (c) 4 CCPI Classification (d) 5 CCPI Classification

Fig. 6. Receiver operating characteristic (ROC) curve of different classification techniques on the N-CCPI tasks. Accuracy Under Curve (AUC) in the legend indicates the overall discriminative performance. SVM shows better discriminative performance across 1- to 6-CCPI instructions.

(centroids) of different instructions in that specific group. The group label G is classified with (3):

$$G = \arg\min_{g}(\min_{i} \mathrm{DPCA}(\mathbf{P}, \mathbf{x}, \mathbf{t}_i^g)) \ , \tag{3}$$

where t_i^g is the template for instruction i after component reduction is achieved using principal component analysis (DPCA). Instruction i belongs to group g of hardware utilization. Other classification techniques are explored here and compared with SVM-based classification on the dataset described earlier.

To evaluate their discriminative ability, the ROC curve of different classifiers over 1, 2, 4, and 5-CCPI groups are compared in Fig. 6. Three additional classifiers, 1 Nearest Neighbor (1NN), Linear Discriminant Analysis (LDA), and Decision Tree, are used for comparison. In all cases, SVM classifiers show the highest Area Under Curve (AUC) (> 0.9976). This indicates that power measurements of instructions with different hardware utilization are mostly linearly separable with clear margins in its high-dimensional feature space.

7.2 Validation of IEB Using Real Code Sequences

In this section, we demonstrate the effectiveness of the technique, discussed in Section 6, w.r.t. CCPI reported in the MSP430 manual. The technique determines an optimal IEB sequence across a number of random sequences of instructions. Although extensive power data has been captured from multiple random as well as predefined sequences (e.g., AES, PID controller, etc.), for the purpose of brevity, given the constraints of this manuscript, just two sample sequences are listed in Tables 3 and 4, with correct predictions marked in bold with a blue background.

Multi-CCPI templates constructed from individual power supply pin data, majority decision from these individual results (MV), aggregate template (Agg.) and power sum (PSUM) are presented. CAX values depict the classification accuracy for the templates from the discrete MSP430 chip.

Most often, 2-clock-cycle instructions are mis-predicted as two 1-clock-cycle instructions (Tables 3 and 4). This can primarily be attributed to the two-stage pipeline architecture of the openMSP430. The *decode and fetch* stage of any subsequent instruction is in the pipeline with the last *execute* clock-cycle of an instruction that is currently being executed. For 2-clock-cycle instructions, this translates to higher power consumption during the second clock-cycle of the *execute* stage. Furthermore, if a 1-clock-cycle instruction is in its *execute* stage, the *decode and fetch* stage for the next instruction is in pipeline. The two 1-clock-cycle templates tend to be similar to two execute clock-cycles of a 2-clock-cycle instruction as opposed to the 2-CCPI templates for 2-clock-cycle instructions. A similar argument can be made for the low predictability of the 4-clock-cycle `XOR` instruction followed by a 1-clock-cycle `INC` instruction.

For improved IEB sequence recovery accuracy, templates are generated (as outlined in Section 6) from two different training loops, depicted in Fig. 3, and the template book, T_N, comprises templates from both loops. Minimal distances, $M_{\mathcal{O},N}$ from 2 are then recomputed to decide the presence of an instruction at offset $\mathcal{O}$. The minimum cost, $C_{\mathcal{O}}$ in (2), based on dynamic programming then provides the optimal sequence of IEBs. As Table 5 shows, using the resulting

TABLE 3
Optimal Sequence Of IEBs (Sequence 1)

Sequence 1	CCPI	Optimal IEB Sequences: Power Pin 1	2	3	4	MV	Agg.	PSUM	CA1	CA2	CA3	CA4	AGG	PSUM	CAX
pop_MR	2	**2**	**1**	**2**	**2**	**2**	**2**	**2**	98.98	98.98	98.98	98.98	100.00	99.99	98.72
add_MMNS	5	**5**	**5**	**5**	**5**	**5**	**5**	**5**	100.00	100.00	100.00	100.00	100.00	100.00	96.34
inc_RCInNS	4	**4**	5	**4**	**4**	**4**	**4**	**4**	99.69	96.79	92.95	99.99	99.99	97.27	97.46
mov_MMNS	5	**5**	**5**	6	**5**	**5**	**5**	**5**	98.82	98.99	96.53	98.99	100.00	99.99	99.99
add_RR	1	**1**	6		**1**	-	**1**	**1**	100.00	98.78	94.21	98.65	97.65	100.00	98.81
sub_MMS	6	**6**	**1**	**6**	**6**	**6**	**6**	**6**	97.16	85.35	88.28	95.19	96.35	89.57	97.56
dec_CR	1	**1**			**1**	-	**1**	**1**	96.35	95.35	96.24	97.35	100.00	97.56	98.96
mov_RInNS	3	**3**	6	6	**3**	-	**3**	**3**	100.00	100.00	100.00	100.00	100.00	100.00	98.44
subc_ImR	2	**2**			**2**	-	**2**	**2**	97.89	98.68	97.89	97.89	98.67	98.56	96.32
bit_MMNS	6	**6**	**6**	**6**	**6**	**6**	**6**	**6**	98.78	96.23	95.65	97.30	98.35	95.46	96.53
cmp_MMS	5	**5**	**5**	**5**	**5**	**5**	**5**	**5**	97.56	97.56	93.24	95.14	98.79	97.56	99.99
xor_RCInNS	4	**4**	**4**	5	**4**	**4**	**4**	**4**	93.25	94.26	95.35	96.54	98.45	97.35	96.55
inc_CR	1	**1**	**1**		**1**	**1**	**1**	**1**	97.26	95.28	98.34	99.99	100.00	100.00	95.45
Total	45	45	45	45	45		45	45	98.13	96.63	95.97	98.15	99.10	97.95	97.78

TABLE 4
Optimal Sequence Of IEBs (Sequence 2)

Sequence 2	CCPI	Optimal IEB Sequences: Power Pin 1	2	3	4	MV	Agg.	PSUM	CA1	CA2	CA3	CA4	AGG	PSUM	CAX
pop_MR	2	**2**	**1**	**2**	**2**	**2**	**2**	**2**	98.98	98.98	98.98	98.98	99.34	99.99	100.00
inc_RCInNS	4	**4**	5	**4**	**4**	**4**	**4**	**4**	99.89	96.79	97.15	99.99	99.99	98.67	97.46
mov_MMNS	5	**5**	6	**5**	**5**	**5**	**5**	**5**	100.00	100.00	100.00	100.00	100.00	100.00	95.29
add_RR	1	**1**		**1**	**1**	**1**	**1**	**1**	98.99	98.99	96.53	98.99	100.00	99.99	100.00
sub_MMS	6	**6**	**6**	**6**	**6**	**6**	**6**	**6**	99.99	96.97	99.99	99.99	100.00	98.26	95.23
dec_CR	1	**1**	**1**		**1**	**1**	**1**	**1**	94.24	85.35	83.28	92.58	96.35	89.57	93.48
add_MMNS	5	**5**	**5**	6	**5**	**5**	**5**	**5**	99.97	95.35	96.24	97.35	100.00	97.66	98.96
dec_CR	1	**1**			**1**	-	**1**	**1**	95.89	96.68	97.89	97.89	98.67	98.56	96.32
mov_RInNS	3	**3**	6	6	**3**	-	**3**	**3**	100.00	100.00	100.00	100.00	100.00	100.00	98.53
subc_ImR	2	**2**			**2**	-	**2**	**2**	98.78	97.88	95.65	97.30	96.56	95.44	95.24
bit_MMNS	6	**6**	**6**	**6**	**6**	**6**	**6**	**6**	97.56	97.56	93.24	95.14	98.79	96.55	98.18
cmp_MMS	5	**5**	5	5	**5**	-	**5**	**5**	93.25	94.27	95.35	96.19	98.45	97.35	95.36
xor_RCInNS	4	**4**	5	5	**4**	-	5	**4**	99.54	99.54	99.54	99.53	97.57	100.00	97.36
inc_CR	1	**1**			**1**	-		**1**	96.99	96.97	98.98	97.45	98.26	100.00	98.25
Total	46	46	46	46	46		46	46	98.15	96.81	96.63	97.96	98.86	98.00	97.12

TABLE 5
Prediction Sensitivity Per Instruction Using Codebook of Templates

CCPI	Instructions	IEB Sensitivity [%]					Freq. %
		Pin 1	Pin 2	Pin 3	Pin 4	Power Sum	
2	pop_mem_reg	100.00	100.00	90.00	100.00	100.00	7.52
5	add_mem_mem_nosub	100.00	100.00	100.00	100.00	100.00	8.27
4	inc_reg_const_ind_nosub	100.00	100.00	100.00	100.00	100.00	6.01
5	mov_mem_mem_nosub	100.00	100.00	87.50	100.00	100.00	6.01
1	add_reg_reg	100.00	100.00	80.00	100.00	100.00	7.52
6	sub_mem_mem_sub	100.00	100.00	100.00	100.00	100.00	8.27
1	dec_const_reg	100.00	100.00	100.00	100.00	100.00	12.03
3	mov_ind_reg_nosub	100.00	100.00	100.00	100.00	100.00	7.52
2	subc_imm_reg_sub	100.00	100.00	100.00	100.00	100.00	6.77
6	bit_mem_mem_nosub	100.00	100.00	100.00	100.00	100.00	6.77
5	cmp_mem_mem_sub	100.00	100.00	81.81	100.00	100.00	8.27
4	xor_reg_const_ind_nosub	100.00	100.00	36.36	100.00	100.00	8.27
1	inc_const_reg	100.00	100.00	12.50	100.00	100.00	6.01
1	nop_reg_reg	100.00	100.00	100.00	100.00	100.00	0.76
	Overall Sensitivity [%]	**100.00**	**100.00**	**84.96**	**100.00**	**100.00**	

codebook of templates allow for 100 percent sensitivity (percent detected) for each instruction from individual power supply pins.

CA values 1-4 (for templates from power pins 1-4 respectively) depict the classification accuracies once the IEB sequence has been determined which is the final step in the instruction sequence recovery process. Test data comprising of 10 000 instances of power traces is prepared based on IEB determined in the test sequence. The training data prepared in advance and described in Section 7 based on power data from two loop sequences is utilized for SVM classification on the test data. A further improvement in classification is achieved when a majority decision is made on the selection of a training dataset from one of the loop sequences in Fig. 3. When computing Minimal distances $M_{\mathcal{O},N}$, if a majority of the minimal distances from the test data to the training templates match a template derived from power traces of one of the two loops, then the test data at an offset would classify well based on training data derived exclusively from that loop. Accuracy of classification varied between 95 and 99 percent.

8 Testing for Robustness of the Coarse-Grained Classifier

Both a soldered and socketed version of the testbed were built to test a quad flat package IC. Thorough reliability tests were conducted on the socketed version of the testbed which allowed for DUT replacements, yielding results incorporating multiple FPGAs. Observations from the target FPGA were later analyzed when the package surface was subjected to controlled temperature settings (Fig. 7). Results when the FPGA package was set to an uncontrolled temperature were compared against those set to 12°C, 20°C, 40°C, and 60°C Utilizing the SVM classifier, classification accuracies ranging from 92 to 100 percent were achieved as training samples were captured at the specified temperatures. When training data samples, captured at a specific temperature, were tested for classifying instructions from datasets at other temperatures, the classifier provides results with above 92 percent accuracy. Similar results were achieved for other CCPI groups. This experiment was repeated on a different FPGA chip and similar results were achieved.

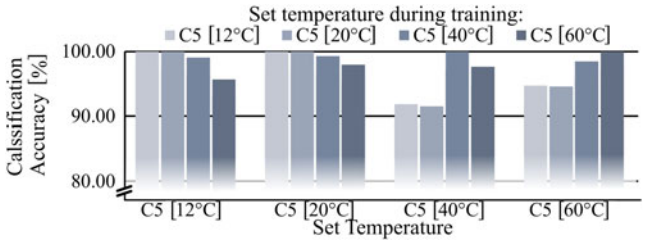


Fig. 7. Results after performing SVM classification over temperature controlled 4-CCPI datasets with power sum metric applied on power data from all four power pins. Color shades represent the training sets used in classifying test data observed at different temperatures. Results show that good classification accuracy is achieved across operating temperatures.

Robustness of the technique was evaluated on training data over ten different FPGAs and summarized in Table 6. The values in Table 6 represent average accuracy based on SVM-classification performed with the 6-CCPI group. Each row summarizes independent results with the training datasets from ten FPGAs (labeled as C1-C10). Test datasets from ten FPGAs were classified into hardware classes in Table 1. The classification accuracies were averaged and given under columns labeled as C1-C10. From the classifier results, it is evident that the training data prepared from one FPGA can be utilized for classifying test data observed from different FPGA with high accuracies ($> 91\%$). However, it can be noted that for a specific case, FPGA C9, the lower accuracies (50 percent) may be attributed to socket wear accumulated over time. It was later determined that the socket to pin contacts were not as firmly established as compared to other FPGAs. Similar results were achieved when the SVM-classification was performed with the other CCPI sets.

The technique's robustness was evaluated on test data based on random sequences using ten different FPGAs and is summarized in Table. 7. The table depicts classification accuracy across multiple FPGA observations when SVM classification was performed over random sequence datasets. Labels C1-C10 represent ten different FPGA chips. Training data for both aggregate and power sum from C1 was used to classify random test sequences for every chip including itself. Results show that the training data from one FPGA can be used in the classification of random sequences on other FPGAs with relatively high accuracies ($> 83\%$).

TABLE 6
SVM Classification Accuracy [%] Over 6-CCPI Datasets Using 10 FPGAs for Training and Classification

Training FPGA Dataset	Test FPGA Dataset									
	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
C1	100.00	91.38	98.19	98.48	91.85	99.24	97.52	97.06	50.00	98.13
C2	99.57	99.99	99.89	97.84	99.98	99.88	99.90	99.89	50.00	99.32
C3	99.91	99.26	100.00	99.93	99.70	99.96	99.97	99.98	50.00	99.90
C4	99.99	99.14	99.96	99.99	99.17	99.96	99.92	100.00	50.00	100.00
C5	98.57	99.97	99.65	92.90	100.00	99.88	99.44	99.68	50.00	98.78
C6	100.00	99.90	99.98	99.94	99.92	99.99	99.94	99.98	50.00	99.96
C7	99.98	98.84	99.99	99.38	99.48	99.98	100.00	100.00	50.00	99.98
C8	99.92	97.84	99.93	99.52	95.54	99.94	99.95	100.00	50.00	99.95
C9	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	99.99	50.00
C10	99.99	93.87	99.98	99.98	96.88	99.94	99.90	99.99	50.00	100.00

TABLE 7
Classification Accuracies [%] of Random Sequences Across 10 FPGAs

Training FPGA Dataset	Test FPGA Dataset									
	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Aggregate	98.53	86.34	96.54	97.61	88.50	97.43	98.25	95.48	83.46	94.64
Power Sum	96.54	83.56	93.32	95.23	89.64	97.50	97.30	94.67	84.67	93.12

8.1 Interpolating Power Templates to Match Device Specific Variations in Frequency

To compare the classification results from each of the four discrete MSP430 devices, the internal DCO (Digitally Controlled Oscillator) was set to 10 MHz by specifying control registers RSEL and DCO. However, due to device-level process variations, the clock frequencies among each device exhibit deviations of approximately 1 percent from normal. In Fig. 9a, the time dilation of power-supply waveforms observed on chips LP2, LP3, LP4 w.r.t chip LP1 is evident after the 40th time sample. These device-specific variations are accounted for by interpolating the power waveforms observed on those devices by one to two time-samples (Fig. 9b).

With the interpolated power waveforms, trained templates constructed using our methodology was tested for reliability across four different discrete MSP430 chips (Fig. 8). Results after interpolation are above 92 percent. Training datasets are interchangeable among different chips with classification accuracy over 90 percent in cross validation among different FPGAs. Similar performance was observed with other CCPI groups.

9 Validation of Fine-Grained Classifier

The eventual goal of side-channel power-leakage-based disassembly is to precisely determine instructions being executed, including on op-code and the operands being processed. To achieve the latter, it would necessitate enforcing tighter constraints on control flow, operand data being processed, and higher data sampling rates for the measurement device. A practical approach could restrict disassembly to determining the addressing modes, which was addressed in part by the coarse-grained classifier discussed in Section 7.

To further explore the discriminating capabilities of the methodology presented in Section 6 and possibly distinguish instructions based on their op-code, the SVM-classifier was applied on power data for individual instructions on a case-by-case basis for each N-CCPI group. In Table 8, classification results for the 6-CCPI group are presented

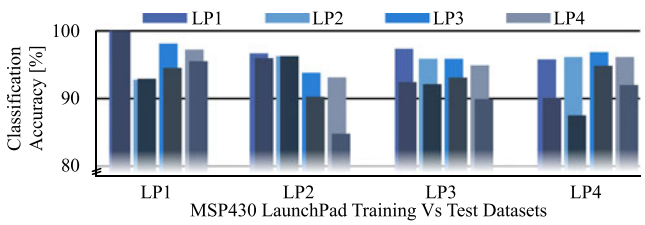
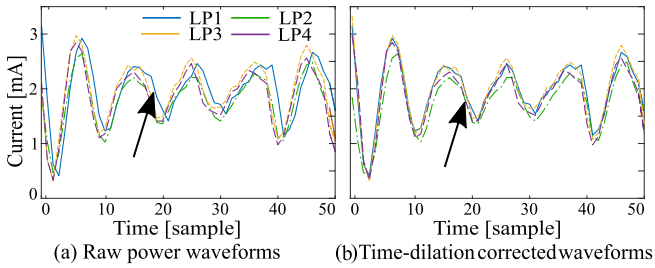


Fig. 8. Results of SVM classification across four discrete MSP430 chips over their 4-CCPI datasets. Color shades represent the training sets used in classifying test data for each discrete chip. Variation in frequency due to process variations among the devices is accounted for by means of interpolation. Results prior to performing interpolation (shaded) have lower classifier accuracies.



(a) Raw power waveforms (b)Time-dilation corrected waveforms

Fig. 9. In (a) the time dilation shown among power traces from four discrete MSP430 chips (LP1, LP2, LP3, and LP4). The arrow near time sample 20 indicates where the waveforms are exceedingly different. After interpolation (b), the time dilation is corrected.

based on aggregate templates. In both cases, the training dataset comprised 7 000 samples and was classified over a test dataset comprising 3 000 samples. Average classification accuracies were 97.58, 93.45, 99.74, 99.20, 99.7, and 99.97 percent for 1, 2, 3, 4, 5, 6-CCPI groups respectively based on four-pin-aggregate templates. Average classification accuracies were 90.52, 89.91, 97.69, 98.45, 99.47, and 99.82 percent for 1, 2, 3, 4, 5, 6-CCPI groups respectively based on the power sum templates. In general, discriminating capabilities improve as the number of features increase for longer execute-cycle instruction sets.

Instructions that incorporate the two's complement unit (e.g., `SUB`, `SUBC`, and `CMP`) were misclassified as `XOR`, or the `BIC` instructions on occasion. In hardware, `SUB`, `SUBC`, and `CMP` instructions perform a bit inversion followed by an `XOR` operation to determine two's complement. The `XOR` instruction, on the other hand, performs its namesake operation. Hence, they are very similar in operation which accounts for 0.05-10 percent misclassification rates across different CCPI groups. To take one particular example, a 4-CCPI `CMP` is misclassified as an `XOR` at a rate of 2.20 percent and as a `SUB` with a rate of 0.03 percent. A finer classification that isolates these instructions is seen to reduce misclassification rates. When a subset of 4-CCPI instructions is classified as `CMP`, `XOR`, and `SUB` a 100 percent classification is achieved. Thus, one could use hierarchical SVM classification that involves running the coarse-grained classifier to identify the hardware utilization bin followed by a fine-grained classification to identify individual instructions.

Robustness of the technique is evaluated on test data based on random sequences using ten different FPGAs and

TABLE 8
Fine-Grained Classification Accuracy [%] of 6-CCPI Instructions From a Dataset Aggregating Data From all Power Pins for 10 FPGAs

Training Instruction	Test Instruction											
	mov	sub	subc	xor	cmp	and	bic	bis	bit	add	dadd	addc
mov	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
sub	0.00	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
subc	0.00	0.00	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
xor	0.00	0.00	0.00	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
cmp	0.00	0.00	0.00	0.00	99.93	0.00	0.07	0.00	0.00	0.00	0.00	0.00
and	0.00	0.00	0.00	0.00	0.00	100.00	0.00	0.00	0.00	0.00	0.00	0.00
bic	0.00	0.00	0.00	0.00	0.00	0.00	100.00	0.00	0.00	0.00	0.00	0.00
bis	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00	0.00	0.00	0.00	0.00
bit	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	99.90	0.10	0.00	0.00
add	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.17	99.83	0.00	0.00
dadd	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00	0.00
addc	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00

TABLE 9
Fine-Grained Classification Accuracy [%], Random Sequences, 10 FPGAs

Training FPGA Dataset	Test FPGA Dataset									
	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
C1	92.72	72.78	89.24	89.15	76.12	86.86	85.34	90.36	77.01	82.25
C2	73.08	90.44	74.67	76.04	79.73	72.72	74.69	77.40	70.81	77.89
C3	83.27	70.42	91.35	82.21	70.04	88.72	82.56	90.18	74.75	88.17
C4	89.05	77.72	83.03	90.27	77.93	84.85	82.01	87.33	74.00	85.32
C5	79.22	80.90	77.00	74.59	90.47	74.47	76.80	70.75	70.47	73.66
C6	90.21	75.01	86.17	83.30	78.65	91.99	85.44	90.96	79.36	82.04
C7	83.96	73.17	89.84	88.51	80.67	83.47	92.27	89.29	73.62	85.19
C8	82.50	77.66	83.22	84.01	78.18	85.20	84.65	90.05	75.12	84.73
C9	74.04	73.25	77.84	77.25	74.23	70.37	75.30	74.93	90.41	74.71
C10	87.68	71.90	82.25	90.53	77.08	90.86	88.45	83.41	70.24	92.38

is summarized in Table 9. When classifying instructions in random sequences using training data from the same FPGA, the accuracies range from 90.05 – 92.72 percent. Robustness across discrete four MSP430 chips is summarized in Table 10. When classifying instructions in random sequences using training data from the same discrete MSP430 chip, the accuracies range from 91.71 – 94.88 percent. For both results, aggregate templates were utilized for classification.

9.1 Comparison of Fine-Grained Classifier Results to Coarse-Grained Classifier

Information entropy is utilized to quantify the complexity of the instruction classification problem and performance of classifiers. Let X be the random variable of the instruction to be classified assuming following a uniform distribution over the set $\{c_1, \ldots, c_K\}$, where K is the number of classes. The entropy of the classification problem $H(X)$ in units of bits is $H(X) = \sum_{i=1}^{K} p_i \log_2 \frac{1}{p_i} = \log_2 K$, where p_i is the probability of $X = c_i$.

The performance of classifier Y can be quantified in entropy reduction (how many bits of information the classifier provided about X) using mutual information as:

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) \log_2 \frac{p(x,y)}{p(x)p(y)}.$$

In Table 11, it can be seen that the fine-grain classifier recovers more information than the coarse-grain classier. However, the accuracy of the coarse-grain classifier shows that the division of classes is well-suited to hardware and thus provides an accurate and meaningful label for hardware use. The coarse-grain classifier could be used as the basis for hierarchical classification or other algorithms where a different fine-grain classifier could be created for each hardware utilization bin for detailed discrimination.

For each instruction, one dataset was collected from the same set which training were drawn. Based on this manner of data collection, it is difficult to determine if differentiating features the classifier picked up on where from the inter-capture variations or instructions themselves. Therefore for final validation, the classifier was used to identify random instruction sequences as well as representative codes like AES and PID, as discussed in Section 10, with high classification accuracies.

TABLE 10
Fine-Grained Classification Accuracy [%], Random Sequences, 4 Discrete MSP430 Chips

Training Dataset	Test Dataset			
	LP1	LP2	LP3	LP4
LP1	91.71	85.84	89.43	88.23
LP2	90.59	94.82	85.77	83.17
LP3	83.29	89.64	94.88	82.97
LP4	90.52	88.94	87.37	94.58

TABLE 11
Number of Classes, H(X) and I(X;Y) for Each N-CCPI Group

		CCPI					
		1	2	3	4	5	6
Coarse	# Classes	3	3	3	5	6	2
	H(x) [bits]	1.59	1.59	1.59	2.32	2.58	1.00
	I(x) [bits]	1.45	1.58	1.27	1.75	2.19	0.97
Fine	# Classes	16	44	22	19	39	12
	H(x) [bits]	4.00	5.46	4.46	4.25	5.29	3.58
	I(x) [bits]	2.91	3.78	3.54	3.97	4.63	2.56

10 Applications Discussion

In this section, real-world applications of the IEB determination and classification technique are presented. However, it must be noted that this technique is not limited to the presented subset of problems; the approach can be utilized in detecting the presence of malicious code sequences (as the first line of defense) and in mitigating faults that might occur in a broader class of systems.

10.1 Feasibility of IEB Sequence Determination

Two sequences, Sequence 1 (Table 3) and Sequence 2 (Table 4) are evaluated for demonstrating the robustness of IEB sequence detection and the SVM classifier. Sequence 1 is an untampered sequence of instructions which contains only one `dec` instruction, Sequence 2 comprises a tampered and re-ordered sequence of instructions with a second `dec` instruction introduced after the seventh instruction. For both instruction sequences, the IEB sequence determination technique recovered the sequence of clock cycles with a 100 percent accuracy. Classification accuracy with templates from power pins 1-4 of the FPGA vary between 95.97 and 98.95 percent. There is an improvement (97.78 for sequence 1 and 97.12 percent for sequence 2) of 1-2 percent in classification accuracy from the discrete MSP430 chip when compared to results from an FPGA. An improvement of nearly 1 percent comes with the cost of retaining more components as discussed in Section 7. Sequence 2 is detected as modified when the IEB determination of the two sequences is compared. Thus, this technique covers a wide range of anomalies, ranging from code insertions, and re-arrangement of instructions in a code segment where the sequence of IEBs in a code segment is affected. This shows that our technique is applicable to provide system assurance.

In the analysis of several sample sets of unknown sequences, CCPI sequence recovery rates (defined as % of actual CCPI correctly determined) ranged from 97.24 to 99.1 percent using the optimal IEB sequence algorithm on the templates constructed using the aggregate power profile data using a single codebook of templates. With the combined codebook of templates (Table 5), the sensitivity is 100 percent. Furthermore, the application of this technique is a first for microcontrollers with many-clock-cycle instructions. This shows that

TABLE 12
Grouped Instruction Classification Accuracy (PID)

#	H/W Class	Classification Rates							Freq. (%)
		CA1	CA2	CA3	CA4	AGG	PSUM	CAX	
1	RR	98.98	98.98	98.98	98.98	99.34	99.99	100.00	12.07
2	RRS	99.89	96.79	97.15	99.99	98.35	98.89	99.37	6.90
3	MMS	99.89	96.32	98.27	99.99	97.14	98.45	95.29	6.90
4	MMNS	100.00	100.00	100.00	100.00	100.00	100.00	100.00	3.45
5	MIS	99.99	96.97	99.99	95.65	100.00	98.26	98.36	5.17
6	MIB	94.24	85.35	83.28	92.58	96.35	89.57	95.17	3.45
7	MINS	99.97	95.15	96.24	97.35	98.34	97.66	99.57	1.72
8	RCInS	95.89	96.68	97.89	97.89	98.67	98.56	96.32	1.72
9	RCInNS	100.00	100.00	100.00	100.00	100.00	100.00	98.53	1.72
10	RCInP	98.78	97.88	95.65	97.30	96.56	95.43	95.24	1.72
11	RInNS	97.56	97.56	93.24	95.14	97.45	96.55	98.18	1.72
12	InR	93.25	94.27	95.35	96.19	98.45	97.35	97.26	1.72
13	InRL	97.18	98.14	99.54	99.63	97.57	100.00	97.88	1.72
14	MR	96.99	96.97	98.98	99.47	97.25	100.00	99.46	3.45
15	CR	99.89	96.79	97.15	99.99	98.14	98.67	98.25	25.86
16	ImR	100.00	95.65	96.79	100.00	95.25	100.00	98.25	10.34
17	ImINS	93.25	97.33	96.56	97.57	99.44	98.56	98.25	8.62
18	ImInNS	98.37	98.16	98.36	99.44	98.56	99.56	98.25	1.72
	Total	98.01	96.61	98.86	98.18	98.16	98.19	97.98	100.00

TABLE 13
Grouped Instruction Classification Accuracy (AES)

#	H/W Class	Classification Rates							Freq. (%)
		CA1	CA2	CA3	CA4	AGG	PSUM	CAX	
1	RR	98.98	98.98	98.98	97.45	98.34	98.26	100.00	25.31
2	RRS	99.89	95.13	95.25	94.35	99.04	93.48	97.46	2.68
3	MMS	99.89	97.25	98.32	98.65	95.25	98.66	97.27	16.02
4	MMNS	100.00	100.00	100.00	100.00	100.00	100.00	100.00	3.99
5	MIS	98.46	97.31	98.45	94.14	100.00	97.88	95.23	0.48
6	MIB	87.25	88.32	81.26	94.26	96.25	88.25	93.48	0.60
7	MINS	90.44	91.58	93.26	90.23	97.21	91.24	98.96	0.48
8	RCInS	95.89	95.34	96.17	98.86	98.39	99.79	96.32	7.03
9	RCInNS	100.00	100.00	100.00	100.00	100.00	100.00	98.53	0.95
10	RCInP	95.23	98.26	95.19	94.35	96.31	95.33	98.35	2.08
11	RInNS	96.16	93.28	96.27	95.14	94.26	96.65	98.18	1.43
12	InR	92.11	93.39	94.27	96.19	98.25	94.17	98.61	0.48
13	InRL	96.28	98.36	95.39	97.36	97.47	94.53	97.36	0.36
14	MR	95.26	95.24	96.47	97.45	96.25	93.42	98.51	2.98
15	CR	98.03	96.79	97.15	98.63	99.68	98.77	98.25	17.99
16	ImR	93.25	95.65	96.79	97.57	96.19	93.23	98.19	13.10
17	ImINS	94.33	92.16	97.36	94.28	95.39	97.18	98.25	2.03
18	ImInNS	95.56	96.35	89.94	87.45	97.12	89.35	98.25	2.03
	Total	95.95	95.74	95.58	95.91	97.52	95.57	97.84	100.00

unknown code can be disassembled to identify individual instructions.

10.2 Results From Targeted Algorithms in Security Applications

The IEB sequence determination was performed on code for an AES encryption (1679 instructions - 4231 clock cycles) and a PID controller sequence (58 instructions - 177 clock cycles). The CCPI sequence recovery rate when using discrete MSP430 chips was 100 percent. In the case of a soft-core implementation on FPGA, the CCPI sequence recovery rates were 95 percent for AES and 97 percent for the PID controller code. Towards determining code modification, anomaly detection was performed with the PID controller code by injecting malicious instructions which would modify the setpoint values after the derivative gain is applied. The modified instruction sequence could be distinctly identified from the unmodified controller code.

With the IEB sequence for the AES and PID controller determined, the coarse-grained classification was performed using the combined codebook of power templates for the FPGA-based soft implementation and discrete MSP430 chip testbed. The results for the PID controller and AES are shown in Tables 12 and 13 respectively. Individual instructions that belong to the same hardware utilization class were grouped together and the data shown in the table is the average classification accuracy within each group. The average classification accuracy (CA 1-4 for the FPGA dataset and the CAX for the discrete MSP430 dataset) for most classes are above 94 percent. Classification accuracy for the discrete MSP430s are comparable to those from the FPGA for both sequences.

TABLE 14
Fine-Grained Classification Accuracy [%],PID, 10 FPGAs

Training FPGA Dataset	Test FPGA Dataset									
	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
C1	91.49	76.29	83.45	90.31	84.74	82.93	86.44	86.41	82.10	84.21
C2	75.72	93.32	79.48	78.81	82.05	79.86	74.46	78.44	82.79	76.40
C3	83.96	76.71	91.80	82.72	82.86	85.43	84.62	88.31	83.10	86.98
C4	88.53	84.93	85.64	91.35	80.10	89.19	88.75	86.80	79.16	90.94
C5	81.39	82.52	75.78	74.36	91.97	77.67	75.88	77.56	75.78	77.36
C6	90.82	79.00	88.85	83.00	83.38	93.92	89.79	89.97	75.55	86.99
C7	90.69	77.50	86.48	85.49	78.04	89.27	94.57	89.69	84.61	85.87
C8	90.90	84.67	87.83	88.30	83.16	85.57	89.83	91.62	84.89	86.76
C9	80.71	74.42	81.81	76.13	76.41	79.62	74.66	84.55	92.59	81.49
C10	84.95	79.60	90.88	82.79	81.17	83.59	87.55	83.68	77.84	91.59

TABLE 15
Fine-Grained Classification Accuracy [%],AES, 10 FPGAs

Training FPGA Dataset	Test FPGA Dataset									
	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
C1	94.81	74.90	82.13	83.41	82.77	85.54	87.80	82.25	81.97	89.28
C2	79.93	94.17	81.88	82.45	82.63	79.87	77.00	82.95	76.02	72.63
C3	85.85	77.48	93.94	82.65	75.18	86.17	90.37	85.13	73.72	85.75
C4	89.27	84.99	90.11	92.20	74.60	88.76	90.66	88.77	75.71	88.38
C5	79.22	79.53	71.51	78.28	94.04	75.90	78.52	72.71	73.96	71.93
C6	90.91	84.84	83.74	85.21	84.82	92.56	86.17	85.22	81.76	87.74
C7	86.47	80.77	82.55	85.07	80.52	90.61	93.22	85.88	71.91	87.84
C8	86.19	73.66	84.55	87.64	70.07	82.08	88.57	94.78	75.69	86.80
C9	84.52	70.79	79.76	73.16	81.39	84.67	83.56	82.97	94.26	70.90
C10	87.64	83.54	85.67	85.31	81.19	85.53	85.82	86.37	70.98	94.65

TABLE 16
Fine-Grained Classification Accuracy [%],PID , 4 Discrete MSP430 Chips

Training Dataset	Test Dataset			
	LP1	LP2	LP3	LP4
LP1	95.09	89.32	87.96	88.53
LP2	89.57	94.44	88.86	88.85
LP3	89.85	86.86	93.26	89.76
LP4	89.01	87.73	86.18	95.82

Furthermore, we performed fine-grained classification on the PID controller and AES code. The accuracy obtained using aggregate data for both the PID controller and AES codes using ten different FPGAs is summarized in Tables 14 and 15 respectively. The table shows the average cross-validation results for all hardware utilization bins when the training dataset for each FPGA was used to classify test data for all ten FPGAs. When classifying instructions in the PID controller and AES instruction sequences using training data from the same FPGA, the accuracies range from 91.35 – 94.57 percent and 92.2 – 94.81 percent respectively. The classification accuracy is above 74 and 70 percent for PID controller and AES respectively, when the training dataset and test data are from different FPGAs. Similarly, the classification accuracies for the PID controller and AES codes for four discrete MSP430 chips are summarized in Tables 16 and 17 respectively. When classifying instructions in the PID controller and AES instruction sequences using training

TABLE 17
Fine-Grained Classification Accuracy [%],AES , 4 Discrete MSP430 Chips

Training Dataset	Test Dataset			
	LP1	LP2	LP3	LP4
LP1	94.88	88.79	88.06	90.17
LP2	88.11	93.96	91.99	89.43
LP3	90.12	86.97	93.42	87.71
LP4	89.38	91.30	89.09	94.09

data from the same discrete MSP430 chip, the accuracies are in the ranges 93.26 – 95.82 percent and 93.42 – 94.88 percent respectively. The classification accuracy is above 86 percent for both codes when the training data and test data are from different MSP430 chips.

11 Conclusion

This work presents a novel non-intrusive technique to detect multi-clock-cycle instruction sequences on a pipelined architecture, based on the analysis of power-supply side-channel analysis. The presented technique employs a dynamic programming algorithm that uses observations from multiple power supply pins. Using a codebook of templates based on two training loop sequences, IEB sequences were recovered accurately. This technique can detect modifications to the firmware running on a general-purpose pipelined embedded platform. Future work includes further investigation of the trade-off between implementation complexity and accuracy of advanced hierarchical classification techniques. Also, the compensation of broader clock frequency variations could explored with further signal processing, since temporal dynamics may or may not be decoupled from the clock frequency variations.

It has been demonstrated experimentally that the proposed multi-clock-cycle instruction classification technique can be seamlessly applied to a discrete instance of a multi-clock-cycle pipelined architecture, a TI LaunchPad MSP430 and also on a soft-core implementation on an FPGA. An evaluation of the reliability of our technique across four discrete MSP430 micro-controllers yielded accuracies between 92 and 100 percent for classifying instructions into hardware bins. This non-intrusive technique provided CCPI sequence recovery rates between 95 and 100 percent from random, AES, and PID instruction sequences. For the same sequences running on a soft-core FPGA implementation, the technique provided CCPI sequence recovery rates between 95 and 100 and over 98 percent accuracy for classifying instructions into hardware utilization bins. An evaluation using several metrics based on power-supply side-channel measurements show improvements in classification accuracy. The proposed technique yielded accuracies 92 and 100 percent for classifying instructions into hardware bins even as the target FPGA package was subjected to different controlled temperatures. When classifying individual instructions using a fine-grain classifier in random, PID controller, and AES instruction sequences using training data from the same FPGA, the accuracies were above 90.05 percent. For the discrete chip, the accuracies were above 91.71 percent.

Acknowledgments

This work was supported by the U.S. Office of Naval Research under Award N00014-15-1-2179 and Award N00014-18-1-2450. This work was also supported by the National Science Foundation under Award 1813945.

References

[1] A. Blyth and G. L. Kovacich, *Information Assurance: Surviving in the Information Environment*. Berlin, Germany: Springer, 2013.
[2] W. Digital, "Growing the RISC-V ecosystem," 2020. [Online]. Available: https://www.westerndigital.com/company/innovations/risc-v
[3] Google, "Opentitan – Open sourcing transparent, trustworthy, and secure silicon," Accessed: Jan. 15, 2020. [Online]. Available: https://opensource.googleblog.com/2019/11/opentitan-open-sourcing-transparent.html
[4] Status report on the first round of the NIST lightweight cryptography standardization process, Accessed: Jan. 13, 2019, 2019. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/ir/2019/NIST.IR.8268.eps
[5] P. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Proc. Annu. Int. Cryptol. Conf.*, 1999, pp. 388–397.
[6] P. Kocher, "Timing attacks on implementations of diffie-hellman, RSA, DSS, and other systems," in *Proc. Annu. Int. Cryptology Conf.*, 1996, pp. 104–113.
[7] J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestré, J.-J. Quisquater, and J.-L. Willems, "A practical implementation of the timing attack," in *Smart Card Research and Applications*, J.-J. Quisquater and B. Schneier, Eds. Berlin, Germany: Springer, 2000, pp. 167–182.
[8] J.-J. Quisquater and D. Samyde, "Electromagnetic analysis (EMA): Measures and counter-measures for smart cards," in *Smart Card Programming and Security*. Berlin, Germany: Springer, 2001, pp. 200–210.
[9] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, "The em side-channel(s)," in *Proc. Int. Workshop Cryptographic Hardware Embedded Syst.*, 2003, pp. 29–45.
[10] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *Proc. Annu. Int. Cryptol. Conf.*, 1997, pp. 513–525.
[11] B. Yang, K. Wu, and R. Karri, "Scan based side channel attack on dedicated hardware implementations of data encryption standard," in *Proc. Int. Conf. Test*, 2004, pp. 339–344.
[12] D. Page, "Theoretical use of cache memory as a cryptanalytic side-channel," *IACR Cryptol. ePrint Archive*, vol. 2002, 2002, Art. no. 169.
[13] Y. Tsunoo, T. Saito, T. Suzaki, M. Shigeri, and H. Miyauchi, "Cryptanalysis of des implemented on computers with cache," in *Proc. Int. Workshop Cryptographic Hardware Embedded Syst.*, 2003, pp. 62–76.
[14] M. Kuhn, "Cipher instruction search attack on the bus-encryption security microcontroller DS5002FP," *IEEE Trans. Comput.*, vol. 47, no. 10, pp. 1153–1157, Oct. 1998.
[15] T. Gilmont, J. Legat, and J.-J. Quisquater, "Enhancing security in the memory management unit," in *Proc. 25th EUROMICRO Conf.*, 1999, pp. 449–456.
[16] D. Asonov and R. Agrawal, "Keyboard acoustic emanations," in *Proc. IEEE Symp. Secur. Privacy*, 2004, pp. 3–11.
[17] D. Genkin, A. Shamir, and E. Tromer, "RSA key extraction via low-bandwidth acoustic cryptanalysis," in *Proc. Annu. Cryptol. Conf.*, 2014, pp. 444–461.
[18] M. Banga and M. Hsiao, "A region based approach for the identification of hardware trojans," in *Proc. IEEE Int. Workshop Hardware-Oriented Secur. Trust*, 2008, pp. 40–47.
[19] D. Genkin, I. Pipman, and E. Tromer, "Get your hands off my laptop: Physical side-channel key-extraction attacks on PCs," in *Proc. Int. Workshop Cryptographic Hardware Embedded Syst.*, 2014, pp. 242–260.
[20] M. Alam *et al.*, "One&done: A single-decryption em-based attack on openssl's constant-time blinded RSA," in *Proc. 27th USENIX Secur. Symp.*, 2018, pp. 585–602.
[21] J. V. Bulck *et al.*, "Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution," in *Proc. 27th USENIX Secur. Symp.*, 2018, pp. 991–1008.
[22] R. Callan, A. Zajić, and M. Prvulovic, "A practical methodology for measuring the side-channel signal available to the attacker for instruction-level events," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2014, pp. 242–254.

[23] B. B. Yilmaz, R. L. Callan, M. Prvulovic, and A. Zajić,"Capacity of the EM covert/side-channel created by the execution of instructions in a processor," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 3, pp. 605–620, Mar. 2018.
[24] B. B. Yilmaz, M. Prvulovic, and A. Zajić, "Electromagnetic side channel information leakage created by execution of series of instructions in a computer processor," *IEEE Trans. Inf. Forensics Security*, vol. 15, no. 7, pp. 776–789, Jul. 2020.
[25] B. B. Yilmaz, E. M. Ugurlu, F. Werner, M. Prvulovic, and A. Zajic, "Program profiling based on Markov models and EM emanations," in *Cyber Sensing*, I. V. Ternovskiy and P. Chin, Eds., vol. 11417, Bellingham, WA, USA: International Society for Optics and Photonics, 2020, pp. 69–83.
[26] S. S. Clark *et al.*, "Wattsupdoc: Power side channels to nonintrusively discover untargeted malware on embedded medical devices," in *Proc. USENIX Conf. Safety Secur. Privacy Interoperability Health Inf. Technol.*, 2013, pp. 9–9.
[27] T. Popp, S. Mangard, and E. Oswald, "Power analysis attacks and countermeasures," *IEEE Design Test Comput.*, vol. 24, no. 6, pp. 535–543, Nov./Dec. 2007.
[28] E. Laohavaleeson and C. Patel, "Current flattening circuit for DPA countermeasure," in *Proc. IEEE Int. Symp. Hardware-Oriented Secur. Trust*, 2010, pp. 118–123.
[29] X. Chen, R. Dick, and A. Choudhary, "Operating system controlled processor-memory bus encryption," in *Proc. Des. Autom. Test Eur.*, 2008, pp. 1154–1159.
[30] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: A first step towards software power minimization," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 1994, pp. 384–390.
[31] V. Tiwari, S. Malik, A. Wolfe, and M. T. C. Lee, "Instruction level power analysis and optimization of software," in *Proc. 9th Int. Conf. VLSI Des.*, 1996, pp. 326–328.
[32] D. McCann, E. Oswald, and C. Whitnall, "Towards practical tools for side channel aware software engineering: 'grey box' modelling for instruction leakages," in *Proc. 26th USENIX Secur. Symp.*, 2017, pp. 199–216.
[33] T. Eisenbarth, C. Paar, and B. Weghenkel, *Transactions on Computational Science X: Special Issue on Security in Computing, Part I*. Berlin, Germany: Springer, 2010, pp. 78–99.
[34] M. Msgna, K. Markantonakis, and K. Mayes, "Precise instruction-level side channel profiling of embedded processors," in *Proc. Int. Conf. Inf. Secur. Practice Experience*, 2014, pp. 129–143.
[35] M. Msgna, K. Markantonakis, D. Naccache, and K. Mayes, "Verifying software integrity in embedded systems: A side channel approach," in *Proc. Int. Workshop Constructive Side-Channel Anal. Secure Des.*, 2014, pp. 261–280.
[36] J. Park, X. Xu, Y. Jin, D. Forte, and M. Tehranipoor, "Power-based side-channel instruction-level disassembler," in *Proc. 55th Annu. Des. Autom. Conf.*, 2018, pp. 1–6.
[37] D. Krishnankutty, R. Robucci, N. Banerjee, and C. Patel, "FISCAL: Firmware Identification using side-channel power analysis," in *Proc. IEEE 35th VLSI Test Symp.*, 2017, pp. 1–6.
[38] J. H. Davies, *MSP430 Microcontroller Basics*. Amsterdam, The Netherlands: Elsevier, 2008.
[39] S. Kadiyala Rao, R. Robucci, and C. Patel, "Simulation based framework for accurately estimating dynamic power-supply noise and path delay," *J. Electron. Testing*, vol. 30, no. 1, pp. 125–147, Feb. 2014.
[40] T. Instruments, "TI launchpa," 2010. [Online]. Available: http://www.ti.com/tool/MSP-EXP430G2
[41] O. Girard, "openMSP430," Accessed: Mar. 1, 2016, 2016. [Online]. Available: http://opencores.org/project,openmsp430
[42] *MSP430x1xx Family User's Guide*, (rev. f) ed., Texas Instruments, 2016. [Online]. Available: http://www.ti.com/lit/ug/slau049f/slau049f.eps
[43] T. T. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, 3rd ed. Cambridge, MA, USA: MIT Press, 1990.

Deepak Krishnankutty (Graduate Student Member, IEEE) is working toward the PhD degree in computer science and electrical engineering at the University of Maryland Baltimore County, Baltimore, MD.

Zheng Li is working toward the PhD degree in computer science and electrical engineering at the University of Maryland Baltimore County, Baltimore, MD.

Ryan Robucci, (Member, IEEE) associate professor of Computer Science and Electrical Engineering with the University of Maryland Baltimore County, Baltimore, MD.

Nilanjan Banerjee is a professor of Computer Science and Electrical Engineering with the University of Maryland Baltimore County, Baltimore, MD.

Chintan Patel (Member, IEEE) is a associate professor of Computer Science and Electrical Engineering with the University of Maryland Baltimore County, Baltimore, MD.