EMSim: A Microarchitecture-Level Simulation Tool for Modeling Electromagnetic Side-Channel Signals

ABSTRACT

Side-channel attacks have become a serious security concern for computing systems, especially for embedded devices, where the device is often located in, or in close proximity to, a public place, and yet the system contains sensitive information. To design systems that are highly resilient to such attacks, an accurate and efficient design-stage quantitative analysis of side-channel leakage is needed. For many system properties (e.g., performance, power, etc.), cycle-accurate simulation can provide such an efficient-yet-accurate design-stage estimate. Unfortunately, for an important class of side-channels, electromagnetic emanations, such a model does not exist, and there has not even been much quantitative evidence about what level of modeling detail (e.g., hardware, microarchitecture, etc.) would be needed for high accuracy.

This paper presents *EMSim*, an approach that enables simulation of the electromagnetic (EM) side-channel signals cycle-by-cycle using the detailed micro-architectural model of the device. To evaluate *EMSim*, we compare the simulated signals against actual EM signals emanated from real hardware (a RISC-V processor implemented on an FPGA), and find that they match very closely. To gain further insights, we also experimentally identify how the accuracy of the simulation degrades when key micro-architectural features (e.g., pipeline stall, cache-miss, etc.) and other hardware behaviors (e.g., data-dependent switching activity) are omitted from the simulation model. We further evaluate how robust the simulation-based results are, by comparing them to real signals collected in different conditions (manufacturing, distance, etc.). Finally, to show the applicability of *EMSim*, we demonstrate how it can be used to measure side-channel leakage through simulation at design-stage.

1. INTRODUCTION

Information leakage through side-channels has become a serious concern in securing computing systems that are located where potential attackers may gain enough physical access to carry out the attack [1, 2, 3, 4, 5, 6, 7]. This is especially a problem for embedded and Internet-of-Things (IoT) systems which often contain sensitive data, such as sensor data, login information for over-the-network management of the system and/or accessing back-end cloud infrastructure, and are often placed in publicly accessible locations. For some side-channels, such as electromagnetic (EM) emanations, physical proximity can be leveraged to attack systems that are considered to be physically secure but are located near publicly accessible locations, e.g., in-wall "smart building" sensors, security cameras, etc [8, 9, 10, 11].

To design these systems to be highly resilient to side-channel attacks, side-channel leakage needs to be quantitatively assessed and attributed to specific hardware and software components, relatively early in the design of the system. For other system properties, such as performance, power consumption, reliability, etc., such early quantitative assessment and attribution typically relies on cycle-accurate simulation [12, 13, 14, 15, 16, 17, 18, 19] that models the relevant activity in the system with enough detail to achieve good accuracy, but without modeling most of the low-level activities that would render the simulation too slow to be useful for evaluating relevant scenarios. If such efficient-yet-highly-accurate simulation would exist for EM emanations, hardware designers and architects could include EM side-channel leakage among their design considerations [20, 21, 22, 23, 24, 25], compilers could use simulation models to optimize for reduced leakage [26, 27, 28], software designers could detect and mitigate information leakage problems for security-sensitive applications [29, 30, 31], etc.

Unfortunately, for assessment and attribution of EM side-channel emanations, no such efficient-yet-highly-accurate simulation exists, and there has *not* even been much quantitative evidence about what level of modeling detail would be needed for high accuracy, how much accuracy is sacrificed when not modeling certain aspects of the hardware and hardware/software interaction, e.g., whether high accuracy for estimating EM emanations can be achieved without modeling the underlying physics (how magnetic and electric fields change in response to current flows through semiconductor and metal structures that form the circuitry of a system), whether it is sufficient to model ISA-level properties without modeling the microarchitecture, etc.

While there are some tools and metrics to quantify EM side-channel leakages [32, 33, 34, 35, 36], ***they are limited due to three main reasons***: *First*, they are mainly focused on developing metrics to estimate the information leakage itself, i.e., mutual information between the signal and the program secrets, rather than modeling the actual analog signal. Relying only on these metrics rather than analyzing the actual signal, may not be sufficient since these metrics inherently make assumptions about the aspects of the signal the attacker may exploit, i.e., they may not reveal *all* of the information the signal may contain. *Secondly*, existing methods only model the system at *architecture-level*, i.e., associating a (leakage) value to individual instructions based on the ISA, and ignore the micro-architecture activities such as pipeline stages, stall cycles, etc. on the signal. As we will show in this paper, this can lead to significant inaccuracy, mainly because the model, by staying at the ISA-level, neglects to account for how that instruction interacts with other instructions and the underlying hardware. *Thirdly*, by neglecting the impact of micro-architecture, these methods implicitly assume that the entire hardware design is a *single source*, and then only model the EM emanations based on this single source. Such an assumption can lead to large inaccuracies since different micro-architecture components (e.g., cache, register-file, etc.) generate different electromagnetic waves with different polarization and/or phase, and hence, they may have *constructive or destructive* impacts on each other and the overall received signal.

To address these challenges, in this work we take a different approach. We develop a simulator, *EMSim*, that is able to simulate the EM side-channel signals cycle-by-cycle using the detailed micro-architectural model of the device. We then quantitatively assess the accuracy of simulator-generated signals by comparing them to the EM signals collected while an actual processor executes the same program code, and we further identify how much accuracy suffers when key micro-architectural features and hardware behaviors are omitted from the simulation model. By using a systematic and carefully designed set of measurements, along with *regression-based parameter estimation and multiple-input-single-output (MISO) communication system modeling*, we show that *EMSim* is able to produce an EM signal that closely matches the actual signal, and it is robust against different sources of variability.

To assess the accuracy of *EMSim* itself, i.e., to avoid discrepancies between simulated and real behaviors that are caused by differences between the *assumed* and the *actual* (often proprietary) microarchitecture of the processor, we implement a RISC-V based [37] in-order processor on an FPGA. We will show throughout the paper that the knowledge of key micro-architectural details is critical in achieving good accuracy for simulation-generated EM signals. We envision that, to provide fairly accurate simulation while having an access to a detailed micro-architecture model, EMSim can be integrated into a cycle-accurate simulator which, as we will show in this paper through some use-case examples, can be used by a variety of users (e.g., hardware designers, architects, software and/or compiler developers, etc.) to *estimate the EM-related side-channel leakages without requiring to physically measure any signals.*

To the best of our knowledge, ***this is the first microarchitecture-level EM side-channel model***, which allows it to provide much more precise estimates of leakage not only for individual instructions but also for sequences of instructions (when the goal is to assess and improve leakage from a particular piece of code on a set of hardware platforms), and also the first model that can asses leakage from a particular part of the system (when the goal is to make the design less "leaky") while maintaining the performance advantages of a cycle-accurate simulation relative to a physics-based model.

The remainder of the paper is organized as follows: In §2, we provide a brief overview of our setup. In §3, we outline our methodology on modeling EM signals as a multi-input-single-output system. §4 demonstrates our approach in modeling micro-architectural events namely pipeline stalls, mispredictions, and cache miss. §5 presents the experimental setup and evaluations on model accuracy and robustness. Real-world use-cases are presented in §6. Related work is briefly reviewed in §7. The paper is concluded in §8.

2. EXPERIMENTAL METHODOLOGY

Before describing how *EMSim* simulates side-channel signals, in this section we first explain our experimental methodology to generate, collect, and analyze the real EM signals generated by the target hardware. Using this information, in the next section we will then describe our method to accurately model these signals.

2.1 Hardware and Measurement Setup

To generate real EM signals, rather than attempting to use an off-the-shelf processor and model such a system, where some of the microarchtiectural details (of even entire microarchitectrual blocks) are proprietary and thus not disclosed in publicly available literature, we implement a 5-stage, single-issue, in-order processor equipped with a branch predictor, a cache, and a forwarding unit using RISC-V ISA [37] and Verilog HDL on an FPGA to have full control and knowledge about the target system and its microarchitecture.

We implement a 32-bit *base* RISC-V ISA [38] which provides a convenient ISA and software toolchain "skeleton" suitable for resource-constrained design scenarios such as embedded systems [37]. We also implement the multiplication/division extension ("M") to support these activities. Prior to using the processor for our measurements, we extensively tested the correctness of the processor's implementation. The designed processor has five pipeline stages namely: `Fetch, Decode, Execute, Memory,` and `Writeback`. The processor also has **(i)** a branch prediction unit with a *2-level predictor* [39] and a *branch-target-buffer* (BTB), **(ii)** a 32-entry 32-bit *register-file*, and **(iii)** a 32KB cache. *Cache-hit* takes one extra cycle and reading from memory takes extra 2 cycles (in addition to the cache latency). Note that these delays can be increased to any arbitrary value (e.g., to study the effect of delay on side-channels). The processor also has *forwarding* and *hazard detection* units.

Using the implemented processor, we then use a probe and a signal acquisition device (an oscilloscope) to receive and record the EM signals (more details on §5).

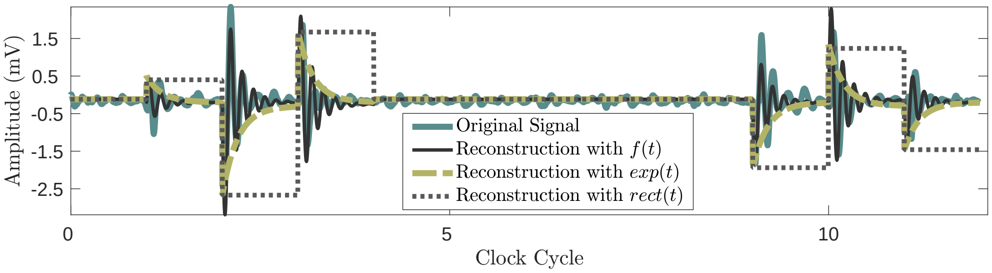


Figure 1: Reconstructing the original signal using three different approaches. Using a combination of a sinusoidal and an exponential function ($f(t)$ in Equ. 5) can achieve the best accuracy.

2.2 Signal Acquisition

Capturing the emanated signal is the first step to properly modeling the signal. Unfortunately, measuring the ideal emanated signal (i.e., not corrupted by additive channel white noise) is not possible if only one-time-run of any instruction sequence is considered. One option is to collect many one-time-run signals and take the average. The problem with this approach is capturing synchronized signals, i.e., the starting points of captured signals may not correspond to the same processing-time of the given instruction sequence. To address this problem, we use a novel signal-processing method called "*modulo operation*" [40] to create a highly accurate estimate of the ideal emanated signal, i.e., to remove most of the noise and distortion that was present in the actual signal due to under-sampling, synchronization, noise, and other imperfections that are unavoidable during practical collection of signals.

The main parameters for the *modulo operation* are the number of clock cycles to execute a given sequence (`noc`), the sampling-rate of the instrument, and the clock frequency of the device. After having these parameters, the next step is to collect the emanated signal. For that, a given sequence is executed several times (1000 times in our measurements). Each set of measurements consists of a sequence of instructions (called `sequence`). The goal is to retrieve the emanated signal for the `sequence`.

The next step is to utilize the *modulo operation* to map each received samples to average these many measurements. Assuming T_s is the total time to execute the sequence once (i.e., $T_s = \texttt{noc} \times T_{clk}$), it applies the following operation to the sampling time of each sample to map each sample to its fundamental period:

$$\Delta_m = mod(T_m, T_s), \tag{1}$$

where T_{clk} is the clock time, Δ_m is called the *modular offset*, and T_m is the sampling time of m^{th} sample. After obtaining the modular offsets for each sample, the *modulo operation* takes the mean of the samples that have same modular offset, to produce the desired signal. Further signal-processing techniques such as moving average, Gaussian filtering, etc., can be applied to this generated signal to obtain smoother reference signals.

2.3 Signal Reconstruction

Simulating an analog signal can be considered a signal-processing reconstruction problem where a continuous signal (i.e., EM in this case) needs to be determined from a sequence of equally-spaced samples with sampling-rate T, where T is preferably much smaller than T_{clk}. Such a signal can be ideally reconstructed using Whittaker-Shannon interpolation formula [41], however, it is well-known that such a method is not feasible in practice. Instead, a popular method for signal reconstruction is zero-order hold (ZOH) technique where a continuous signal, y, can be reconstructed from a sample sequence $x[n]$, assuming one sample per time interval is T:

$$y(t) = \sum_{n=-\infty}^{+\infty} x[n] \times rect(\frac{t - T/2 - nT}{T}). \tag{2}$$

To improve the ZOH accuracy, in this paper we make an observation that *switching activity in a processor is synchronized to its clock* and most of the switching happens right after the positive/negative edge of the clock. Thus, instead of using a rectangular function (which implies that activity is evently spread over a cycle), a decaying function can be used:

$$f_1(t) = e^{-\theta t}\texttt{u}(t), \tag{3}$$

where θ is a positive normalization factor that changes the width of the signal, and $\texttt{u}(t)$ is the unit step function. Substituting $rect()$ in Equ. 2 by the exponential we get:

$$y(t) = \sum_{n=0}^{+\infty} x[n] \times e^{-\theta(t-nT)} \times \texttt{u}(t - nT). \tag{4}$$

However, we observed that the received signal is also exposed to oscillations with decreasing magnitude. To meet the requirements for both oscillations and decreasing amplitude, combining sinusoidal with exponential can increase the accuracy further:

$$f(t) = \sin(2\pi t/T_0) \times e^{-\theta t} \times \texttt{u}(t), \tag{5}$$

where T_0 is the periodicity of the sinus function. Again, substituting $rect()$ in Equ. 2 by $f(t)$, we will have:

$$y(t) = \sum_{n=0}^{+\infty} x[n] \sin(\frac{2\pi(t - nT)}{T_0}) e^{-\theta(t-nT)} \texttt{u}(t - nT). \tag{6}$$

In Figure 1, we plot a measured signal and its reconstructions with these options. We observe that $f(t)$

explains the behavior of the received signal much better. Thus, by finding $x[n]$ for each cycle and using Equ. 6, the analog side-channel signal can be modeled.

3. EMSIM MODELING

3.1 Overview

Fundamentally, EM side-channel signals are created due to *bit-flips* at the transistor-level [9,42]. In principle, all transistors and metal-layer interconnect components contribute to the signal, and so the signal could be modeled using activity of all transistors and on-chip wires as predictor variables, which *should* be highly accurate but is practically infeasible. Thus the main challenge in model-building is to select (or discard) potential predictors in a systematic manner, to achieve a trade-off where feasibility (or even efficiency) is achieved without a major sacrifice in accuracy.

To achieve a simple yet accurate model, existing methods are mainly focused on individual instructions and their operands, and they attribute an "average" behavior to each of the instructions, rather than model its cycle-by-cycle effect on the processor's hardware. In effect, these methods model a simplified *one-instruction-at-a-time* implementation, essentially ignoring pipeline effects and other important aspects of the micro-architecture.

To more accurately simulate EM signals, we model micro-architectural components as ***independent*** sources of EM emanations and then further group these units in each pipeline stage as an individual source. We used pipeline stages as the sources mainly because we observed that each instruction has different footprint in each cycle, and the side-channel generated at each cycle is a combination of these activities in ALL stages. Using this methodology, we model a multi-input (pipeline stages), single-output (EM signal) system (MISO).

Using this approach, the challenges are ***a) how to model the signal for individual sources***, and ***b) how to properly combine the signals*** generated by each source to accurately form the side-channel signal.

3.2 Signal Amplitude for Individual Sources

In practice, there are two contributors in creating EM side-channel signals for each pipeline stage. The first group of contributors, which we call *instruction-dependent* activities, are caused by the switching activities of the micro-architectural units (e.g., register-file, ALU, etc.) that are utilized in that stage (e.g., whether the register-file is being written or not).

The second group, *data-dependent* activities, are created due to bit-flips on the data-bus, address-bus, and any other registers that hold operand's values. These bit-flips are independent from the instruction-type but are dependent to the previous state of the bus. In the following, we will describe how we *independently* measure each of these two groups.

Instruction-Dependent Activities. To independently measure these groups, we first minimize the effect of the *data-dependent* activities by setting all the operands, addresses, and immediate values to zero. This approach enables us to measure the *baseline* signal for each stage that is *only* caused by the switching activities of the micro-architectural units used in that stage.

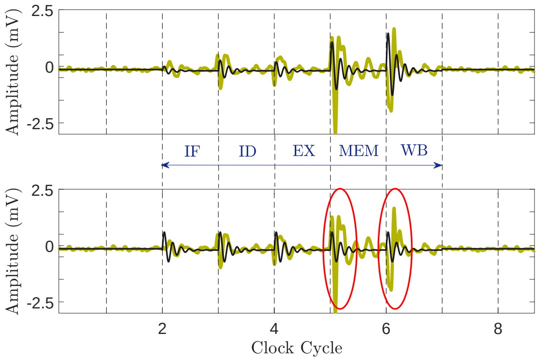


Figure 2: The signal amplitude for an `ADD` as it progress in the pipeline (while all other instructions are `NOP`). The actual signal is shown in light color (green). Darker color (black) shows the simulated signal when considering each pipeline stage as a separate source (top), and when considering the entire processor as a single source (bottom), and the largest differences between the two are pointed out using red ellipses.

After decoupling the data-related activities from the signal, the second challenge is to minimize the effects of other stages on the generated EM signal. Recall that we mentioned ALL pipeline stages contribute to the overall signal, however, ideally we want to be able to measure the effect of each stage *separately* so that we can use them as the basic-blocks to reconstruct the overall signal. To achieve that, we use `NOP` instruction as the *baseline* since it has the minimum possible switching activity, and then create `NOP` → `inst` → `NOP` instruction sequence (for all instructions), while operands for `inst` are all set to $r1$ (and $r1 = 0$). Using this method, no data/operand-dependent bit-flips are created, but register-file, ALU, etc. may be used (depending on the instruction type). We then measure the signal amplitude for all instructions and every pipeline stages. We call this ***baseline hardware amplitude*** or A.

Figure 2 shows how the (actual) EM signal (shown in green/light color) changes as an `ADD` instruction progresses through the pipeline while all the other instructions are `NOP`. Using Equ. 6 and `NOP`→ `inst`→ `NOP` instruction sequence, we used our simulator to generate the signal. Further, to show why individual stages should be modeled separately, Figure 2 (bottom) shows the simulated signal when the "average" amplitude is used for all stages. As can be seen, failing to model each stage individually (as used in previous work [34]) can lead to significant inaccuracies in some stages (note that using *max* instead of *average* also leads to similar inaccuracies).

Data-Dependent Activities. Once the baseline amplitude is measured, the next step is to find how this amplitude changes as the number of bit-flips changes

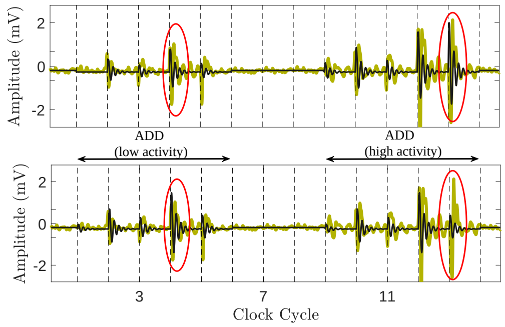


Figure 3: Effect of the *activity factor* on the amplitude. The actual signal shown in green. The simulation is shown in black when activity factor is modeled using a linear regression model (top) and when an *average* activity is used (bottom).

due to value/operand used in the instruction and the previous state of the bus/register. Intuitively, the more bit-flips, the higher the amplitude should be thus we define *activity-factor*, α, as a ***scaling factor*** to the baseline activity, A. To find α, we first treat each bit-flip *equally*, and assume that each bit-flip has similar effect on the signal amplitude. We then calculate α as:

$$\alpha = 1 + \frac{(flips_{new} - flips_{base})}{flips_{total}}, \tag{7}$$

where $flips_{new}$ is the total number of flips for the current instruction, $flips_{base}$ is the total number of flips when previous instruction is `NOP`, and $flips_{total}$ is the maximum possible number of flips for the current instruction. Using this equation, we then define $A' = \alpha \times A$, and use it to simulate the signal. Figure 3 (bottom) shows the original signal (shown in light green), and the simulated signal using this approach (shown in black) for the similar `NOP`→ `inst`→ `NOP` instruction sequence discussed in the previous section. As can be seen in the figure (bottom), this *"averaging"* modeling can not accurately predict the amplitude of the signal which indicates that *not all the bit-flips have the similar impact on the amplitude.* Our further investigation confirmed this theory. Particularly, we found that flips in the output of the ALU and memory have the most significant impacts on the signal. We believe this difference is mainly due to the different physical parameters of transistors and/or lengths of the connecting wires. Using this observation, to systematically calculate the activity factors for each stage, we use a *linear regression* model:

$$\alpha = \delta + \mathcal{T} \times c + \epsilon, \tag{8}$$

where $\mathcal{T}$ is a vector of transition bits across all the existing registers in the targeted pipeline stage, δ and ϵ are the vector of scalar intercept and error terms respectively, c is the vector of activity factors to be predicted by the model. As mentioned before, α is the scaling factor for the baseline amplitude, A, thus $\alpha = A_{meas}/A_{simul}$. Note that to find $\mathcal{T}$, a detailed micro-architecture model is needed to track all the bit-flips for every gate in the processor (except cache/memory). However, to significantly reduce the complexity and simulation time, the size of $\mathcal{T}$ can be reduced using the step-wise regression method [43] where, iteratively, the size of the fitted model (i.e., α and $\mathcal{T}$ in our case) is reduced using standard statistical metrics such as F-tests [43]. In other words, since *not all the bit-flips have statistically significant impact on the emanated signal*, the non-contributing factors can/should be removed from the model. In our processor, using this method we managed to reduce the size of $\mathcal{T}$ by more than 65%.

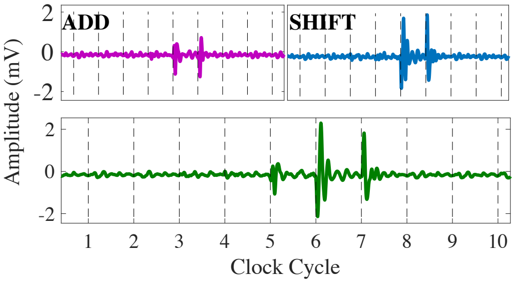


Figure 4: An example of how individual sources (pipeline stages) are combined to form the final signal. *Top*: how the actual EM signal looks like when the instructions are executed in isolation (`NOP, inst, NOP`). *Bottom:* The actual EM signal when the instruction sequence is `NOP, ADD, SHIFT, NOP` (i.e., a combination of multiple instruction in the pipeline).

Figure 3 (top) shows the simulated signals when the linear regression (LR) model is used for activity factors. Compared to the averaging method (bottom), using LR has significantly improved the simulation accuracy.

3.3 Multi-Input Modeling

Once the signal amplitude for individual sources are calculated, the next step is to combine the signals generated by these individual sources to create the simulated EM signal. In principle, the generated EM signal is the *superposition* of individual waves thus depending on each source's phase, the superposition of each pair can be either *constructive* or *destructive*. Using this fact, the overall signal can be approximated as a *linear* combination of these individual sources where the coefficients may vary between ±1, depending on the phases.

Due to the complex nature of the generated EM signals, accurately modeling each and every source mathematically is significantly time-consuming and often infeasible in practice. To tackle this problem and find coefficients for each source, a *model-fitting* approach can be used. We use a *linear-regression* model to find (predict) the overall EM signal. Specifically, we use:

$$X = \delta_s + (\alpha A) \times M + \epsilon_s, \tag{9}$$

where αA is the vector of individual sources amplitudes (α is the activity factor and A is the baseline amplitude), δ and ϵ are the intercept and error vectors, M is the predicted coefficients, and X is the final amplitude which will be used in Equ. 6 to simulate the signal.

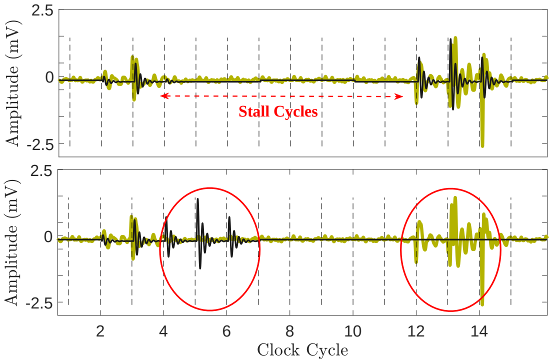


Figure 5: Effect of stalls on the signal. The actual signal shown in green, while simulated signals are shown in black when modeling pipeline stalls (top) and not modeling it (bottom).

Figure 4 shows an example of how two individual sources are combined in each cycle to form the final signal. Figure 4 (top) shows `ADD` and `SHIFT` instructions when they are executed in isolation (i.e., `NOP, inst, NOP`), and Figure 4 (bottom) shows how the final signal looks like when the executed sequence is `NOP, ADD, SHIFT, NOP`. Specifically, cycle 6 is when the `ADD` instruction is in `WB` stage and `SHIFT` is in `MEM`, and the resulting signal is a linear combination of these two sources. Note that to find M, we need to measure all the possible combinations of the entire instructions in the ISA, however, as we will show in Section 5, the number of required measurements can be significantly reduced using standard *clustering* algorithms.

4. MODELING MICRO-ARCHITECTURAL EVENTS

The last step of simulating an EM side-channel signal is adding the signatures of different micro-architectural events to the signal. We particularly add the signatures of the following three events to the signal:

Pipeline Stall. Stalling is a common event in a processor which prevents successor instructions from advancing in the pipeline and preserves the instruction and operands in the stalled stages. Due to this preservation no bit-flips occur in the stalled stages. In addition, to save power, a control signal is typically used to disable (e.g., through power-gating) hardware components in the stalled stages. As a result, stalling typically has a dramatic impact on the switching activities of the stalled stages and, consequently, results in a significant reduction in the amplitude of the generated side-channel signals. Figure 5 shows the effect of stalling on the signal where a `MUL` instruction has stalled the pipeline for eight cycles (we intentionally increased the stall cycles in `MUL` for clarity). As can be seen from the figure, not properly simulating stalls (bottom) results in a significant deviation from the original signal (shown in light green).

Note that stalling does not have any impact on prior instructions thus they still generate side-channel signals as they advance through the pipeline. Thus during the stall, the received side-channel signal is only generated by the instructions in the non-stalled stages (if any).

To properly model stalling, the simulator should be able to detect when stalls are happening (using the micro-architecture model), and ignores the signals generated by the stalled stages during the stall phase. In our model, this is done by setting the amplitudes of stalled stages to zero in Equ. 9.

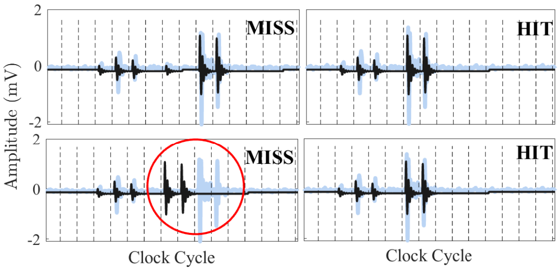
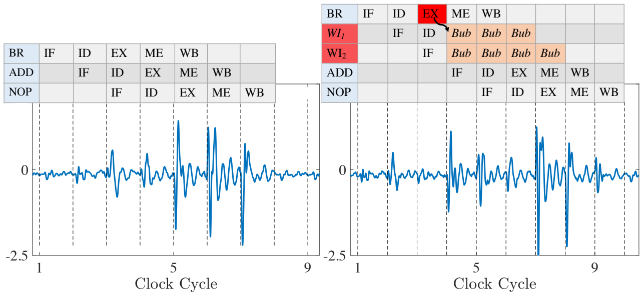


Figure 6: Effect of cache-miss (left) and cache-hit (right) on the signal. Miss causes two extra stall cycles. The actual signal (light blue) and simulated signals (black) with (top) and without (bottom) modeling cache misses are shown.

Cache miss. Similar to pipeline stalls, due to a data-dependency, cache miss can also cause stalls. In our design, accessing the cache stalls the pipeline for one cycle. Further, cache miss and access to the memory causes extra two stall cycles. These two signals and their difference is shown in Figure 6. As can be seen in the figure, two extra stall cycles (total of three) can be seen in the `LD` instruction. Similar to stalls, the cache activity should be properly simulated using the micro-architecture model. Figure 6 illustrates how without properly modeling the cache misses the simulated signal will be deviated from the original signal (bottom left).

Misprediction. In addition to stalls, we observed that branch misprediction also has noticeable impact on the side-channel signals. Depending on the pipeline design, the correct outcome of a branch instruction can be resolved after a few cycles (2 cycles in our design), and if a *misprediction* is detected, the processor has to *flush* the incorrectly fetched instructions, and begin executing the correct ones after that. In order to do that, processors typically substitute the incorrect instructions with `NOP` instructions. It is expected that executing these *bubble* instructions temporarily changes the side-channel signals since they change the switching activities of each stage. Figure 7 shows the received EM signals with and without a misprediction along with instructions present at each cycle in each pipeline stage.

Similar to pipeline-stall, using the micro-architecture model, mispredictions can be detected and simulated in our simulator. It is important to mention that we also studied the impact of using different branch-predictors on the side-channel signals (e.g., always not-taken, 2-level, g-share, etc.) and we did not observe any sta-



BR	IF	ID	EX	ME	WB		
ADD		IF	ID	EX	ME	WB	
NOP			IF	ID	EX	ME	WB

BR	IF	ID	EX	ME	WB				
WI₁		IF	ID	*Bub*	*Bub*	*Bub*			
WI₂			IF	*Bub*	*Bub*	*Bub*	*Bub*		
ADD				IF	ID	EX	ME	WB	
NOP					IF	ID	EX	ME	WB

Figure 7: Effect of misprediction (right) on the signal. It causes two instructions being flushed from the pipeline and hence affect the signal in those cycles.

tistically significant difference between these predictors mainly because they have relatively small switching activities (especially for low-end processors).

It is also important to mention that *we tested the effect of other micro-architectural events* such as data-forwarding on the signal and did not observe any significant difference in the presence and/or absence of them. Also note that, as we mentioned before, in this paper, we limited the modeling to bare-metal, and left system-level activities modeling such as interrupts, exceptions, context-switch, etc. to future work.

5. EVALUATIONS

We divide our evaluations into two main parts. First to show the correctness, accuracy, and robustness of our simulator, we present our experimental evaluations on how well the simulated signal *matched* with the original side-channel signal generated by the target hardware for *ALL* possible combinations of the instructions. We then explore the impact of variations such as manufacturing, environmental, etc. on the accuracy of *EMSim*.

The second part of our evaluations (presented in §6) is focused on the *EMSim* use-cases and its application in different domains such as security, debugging, etc.

5.1 Evaluating Model Accuracy

Setup. We implemented a RISC-V based processor on a Terrasic DE0-CV board with an Altera Cyclone-V FPGA [44] with 50 MHz clock-rate. To record side-channel signals, we used a Keysight digital oscilloscope (DSOS804A), with 1 GHz bandwidth and 10 GSa/s rate. We further studied the effect of changing the sampling-rate on the accuracy and found that similar accuracy can be achieved with much lower sampling-rate (about 200 MHz in our measurements). As a result, similar results can be achieved using a less expensive device (e.g., TBS1032B Tektronix Digital Oscilloscope [45] costs around $300) and/or a high sampling-rate device can be used for modeling devices with faster clock-rates. To receive EM signals, we used a magnetic probe [46], placed 5 cm above the FPGA. Signal processing is done in Matlab2017-b and the simulator is implemented in standard C++ programming language.

Cluster	Type	Inst.	No. Inst.
1	ALU	`ADD,XOR,JAL, ...`	13
2	Shift	`SLLI,SRT, SRA, ...`	10
3	MUL/DIV	`MUL, DIV, REM, ...`	8
4	Load	`LB, LW, LH, ...`	5
5	Store	`SB, SH, SW`	3
6	Cache	`LB, LW, LH, ...`	5
7	Branch	`BEQ, BLT, BGE, ...`	6

Table 1: RISC-V (R32IM) instruction-set and their cluster used in this paper.

Model Building. In §3, we discussed that in order to fit a model, *ALL* possible combinations of instructions should be measured (i.e., about three hundred million combinations in RISC-V ISA). Clearly such a requirement makes the model building extremely time-consuming in practice. However, intuitively, we expect instructions with similar behaviors (e.g., ALU-type, memory-type, etc.) have similar side-channel signals since they share identical hardware activities. Using this intuition, we used the *hierarchical agglomerative algorithm* [47] with the *cross-correlation* as the distance metric to cluster instructions with similar EM pattern into a same cluster. We found that RISC-V ISA can be *clustered* into 7 categories (when the operands are similar) where a single instruction in each category can be a representative of all instructions in that category.

These categories are shown in Table 1. Using this table, we then used only the representative instruction of the cluster for model building which, in turn, reduce the model building complexity significantly. In our setup, the number of measurements was reduced from 300 million to only 16 thousands. Note that while the clustering algorithm did not use the micro-architecture model as a prior knowledge, the clusters confirmed that instructions with similar micro-architecture activities should be clustered in a same group.

Results. To prove that our approach provides accurate simulated signals for *ALL* possible instruction combinations thus can be applicable to *ANY* complex program that uses the mixture of the implemented ISA (R32IM), we created a microbenchmark using all possible combinations of the representative instructions shown in Table 1. Particularly, for a 5-stage pipeline and 7 distinct clusters, there are $7^5 = 16807$ possible combinations that can appear together (in the pipeline) in a cycle. We created a program to generate all these combinations with random operands. We then manually modified branch instructions and assigned the target address and branch condition to create loops with random instruction and iteration sizes. To limit the execution time, we then randomly put these instructions into groups of 1024 combinations (i.e., 5120 instructions in each group which were executed one after another similar to a real program). To cover all the combinations, 17 of such groups were needed (no two groups were similar). We then executed these randomly-generated groups on the processor normally, and recorded the real and simulated signals. To further prove the validity and correctness of our simulator, we also randomly created another 17 groups, this

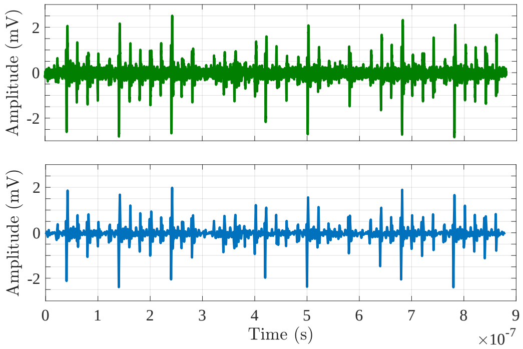


Figure 8: A comparison between the signal generated by a real hardware (top) and the simulated signal (bottom) in EMSim.

time from all instructions in the ISA and not just the representatives. Using these 34 groups/applications, we then compared the simulated signals with the actual ones using the *normalized cross-correlation* method. Each group/application takes about 9000 cycles to finish on average. The execution-time varied depending on the instructions used and microarchitectural events.

Figure 8 shows the simulated and actual EM-side-channel signals for one of the groups tested in our evaluation (for clarity, only the first 50 cycles are shown in the figure). As can be seen from the figure, the simulated signal matches the real signal with high accuracy. We found that, on average, ***EMSim has about 94.1% accuracy in simulating side-channel signals across all possible instruction combinations***.

5.2 Manufacturing Variability

To investigate the impact of manufacturing variability on the model's accuracy, i.e., to determine if training is needed for each physical instance of a (same) device, we repeated our measurements for three physical instances of the Terrasic development board. We then compared the signals received form each device using the normalized correlation method. We observed that, for each cycle, the signals for board #2 and #3 are slightly shifted (compared to the board #1), mainly due to the slight shift in the actual clock frequency of the boards. However, we found that such a shift has no statistically significant impact on the accuracy.

In general, it is important to mention that while a shift in the clock frequency could cause the side-channel signal to be *scaled* in each cycle (depending on the amount of the shift), we observed that this scaling has (almost) the same impact (in terms of shape and amplitude) on all the instructions thus, effectively, makes the true mutual information unchanged.

5.3 Board Variability

To study the effect of board variability, mainly the effect of CMOS technology and board design on the signals, and further evaluate the model accuracy on boards with different manufacturing conditions, we used two other boards: a Terrasic DE1 board with an Altera Cyclone-II FPGA [48] and a Digilent ARTY board with a Xilinx Artix-35T FPGA [49] (both clocked at 50MHz). We then repeated the measurements described in §5.1 for these two new boards and found that, using the same processor design and applications, to correctly model the signal both A and c parameters (cf. Equ. 9 in §3.3) should be retrained to achieve the similar accuracy. Other parameters such as M remained the same since the position of the antenna and the physical and logic design of the processor stays the same. We envision that for different designs, the baseline amplitude and activity factors should be re-trained (only once) and then can be included by the developers as a library (similar to that of for other properties such as power, timing, etc.).

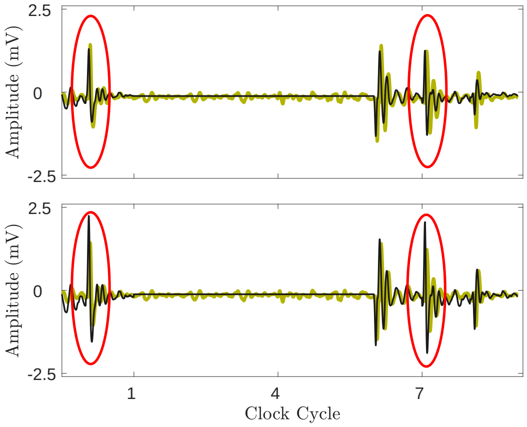


Figure 9: Effect of distance on the signal amplitude. For both figures, the plots with darker color correspond to reconstructed signal, and the other ones correspond to the original signal.

5.4 Effects of Distance

Transferring the ideas from communication theory literature, to find the effect of the distance (i.e., the position of the probe and its distance to the center of borad) on each source, a parameter, called ***loss-coefficient*** or β, can be considered as the channel coefficient of a flat-fading channel. Here, we need to note that, regardless of the position of the probe, the baseline amplitude, A, can not be measured solely because we do not have any control on the power distribution of the board for each instruction at each pipeline stage. Hence, *the resulting signal power is always a combination of the actual signal amplitude and the corresponding loss coefficient (i.e., $A\beta$)*. However, to deal with this problem, we choose the probe's location at the center of the processor as the *base point*, and define β_0 as the loss coefficient at this point. Further denoting A_0 is the actual emanated signal amplitude, we assume the amplitude of the signal can be written as $A = A_0\beta_0$ with respect to the base point. Therefore, with these assumptions, β is assumed to be one for all the measurements done in this paper.

To further investigate the effect of β on the amplitude, we measured the signal (with the same input trace) at a different location, and compared the results with the base case. Figure 9 illustrates the effect of the antenna location on the loss coefficient factor β. Here, the training signals for the reconstruction are obtained from the base measurement, and the figure at the bottom is obtained by neglecting the effect of β (i.e., $\beta = 1$) during the simulation. The figure at the top is generated by solving the same linear regression model given in Equ. 9 this time by substituting A by $A\beta$, where β is not constrained to one (while A is the signal obtained in the base case). We can conclude that considering the effect of β is crucial to explain the changes due to antenna location since better correlation and root mean square (RMSE) results are obtained with the adjusted β. Note that, adjusting the β is only required during the model building (i.e., during measurements if the position of the probe changes), however, the user of the tool does not require to change/adjust β for his/her leakage estimation and can use the base case or numerous cases (depending on the availability) to obtain an "average" leakage estimation, or "worst" case for $\beta = 1$.

6. PRACTICAL USE-CASES FOR EMSIM

In the previous section, we showed *EMSim*'s ability in accurately modeling all possible combinations of instruction in RISC-V ISA. We also demonstrated its robustness against different sources of variability. In this section, we present various use-cases for *EMSim*.

6.1 Side-Channel Leakage Estimation

An important step for defending against side-channel attacks (SCA) is estimating how much (sensitive) information can be possibly leaked (through a specific or set of side-channels) during the execution of an application. To estimate this leakage, different metrics can be used. Particularly, for EM side-channels the state-of-the-art methods are Test Vector Leakage Assessment (TVLA) [50,51] and Signal Available to Attacker (SAVAT) [33] methodologies.

Due to the lack of simulation tools, to properly calculate these metrics, several actual measurements should be performed. Unfortunately, these measurements often require sophisticated equipment and experts with various skills which, in turn, makes them expensive and difficult in practice. Using our approach, however, we show that *EMSim* is capable of generating highly-accurate simulated signals which can be used to calculate these metrics precisely which eliminates the need for an actual measurement infrastructure.

The following describes how *EMSim* can be used to model TVLA and SAVAT. It is important to mention that unlike prior work [34,35,52,53], *EMSim is NOT limited to a specific metric or analysis, and it can be used for ANY analysis based on the EM signal.*

Test Vector Leakage Assessment (TVLA). This metric is based on the *T-test* statistical test which determines if there is a significant difference between the *means* of two groups, which may be related in certain features. In the context of SCA, TLVA shows how much the side-channel signal (e.g., EM) is correlated with specific values in the code. The values can be either some known intermediate nodes (e.g., the output of *Sbox* in AES) or more generally, some *fixed* (or *semi-fixed*) input parameters of a specific function. If the metric exceeds a threshold (i.e., a confidence value in the statistical test), it indicates that there is a (potential) leakage for an SCA such as DPA [54,55], template attacks [6], etc.

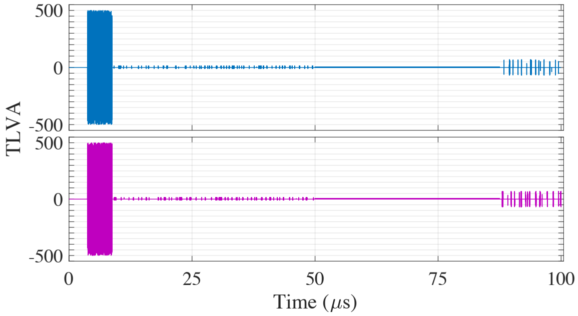


Figure 10: AES-128 leakage assessment using TLVA methodology on the measured/actual signal (top) and the simulated signal (bottom).

To compare the accuracy of TLVA metric based on the measured vs. simulated signals, we ran AES-128 on our RISC-V processor. We then used our setup described in §5.1, to measure the signal. Figure 10 shows these measurements for the real signal (top) and the simulated signal (bottom). As can be seen from the figure, the TLVA metric based on the simulated signal is highly matched with the real measurement and follows the same pattern (and values) as the actual one (i.e., *no-activity→high→low→no→medium* sequence).

Signal Available to Attacker (SAVAT). This metric measures the side-channel signal created by a specific single-instruction difference in program execution, i.e., the amount of signal made available to an attacker who wishes to decide whether the program has executed instruction/event A or instruction/event B.

To measure this metric, Callan *et al.* [33] developed a microbenchmark which creates a controlled alternation between A and B instructions many times. Such alternation creates a periodic signal with period $t_p = t_A + t_B$, where for the half of the period A is executing and for the other half B. Such a periodic activity can then be observed in the frequency domain as a spike at $f_p = 1/t_p$. The key insight is that the corresponding energy of the spike (i.e., area under the curve) indicates how different A and B are from each other (in terms of side-channel signals) hence reveals how much signal would be available to an attacker when the difference between two samples is whether A was executed or B.

To compute SAVAT in both real measurements and simulated signals, we implemented the microbenchmark proposed by Callan *et al.* [33], and used the setup explained in §5.1 to collect the signals. Table 2 shows SAVAT values in our processor for 6 pairs of instructions. As can be seen, the values retrieved from simulations are highly matched with the values computed using the real measurements. SAVAT can then be utilized to reveal the information leakage capacity of the system [56].

	LDM		LDC		NOP		ADD		MUL		DIV	
	R	S	R	S	R	S	R	S	R	S	R	S
LDM	0.02	0	3.71	3.91	5.34	5.32	5.24	5.20	5	5.02	4.98	4.98
LDC	3.72	3.91	0.04	0	0.81	0.85	0.74	0.74	0.21	0.24	0.21	0.23
NOP	5.35	5.32	0.8	0.86	0.01	0	0.08	0.1	0.67	0.69	0.66	0.69
ADD	5.24	5.20	0.74	0.75	0.07	0.1	0.03	0	0.98	1.05	1.03	1.1
MUL	4.98	5.01	0.22	0.21	0.66	0.68	0.94	1	0.03	0	0.04	0.01
DIV	4.97	4.99	0.21	0.21	0.65	0.68	1.05	1.13	0.03	0.01	0.02	0

Table 2: Signal Available to Attacker metric [33] for Real measurements (R) and Simulations (S).

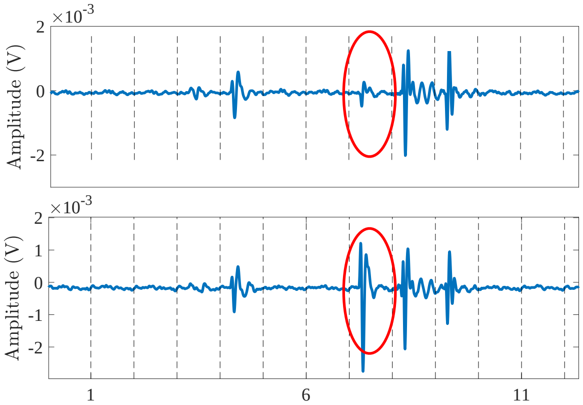


Figure 11: A case-study to show how EMSim can be used for debugging. The measured signal (top) does not match with the reference model obtained by the simulation model (bottom) which indicates that there is a potential error/issue in the hardware.

6.2 Application to Debugging/Profiling

While so far we have shown how EMSim can be utilized to accurately model EM side-channel signals and thus can be used for leakage estimation during developing secure software, in this section, we present another potentially useful use-case of EMSim and show how hardware designers and computer architects can also leverage this framework during hardware development.

Given that EMSim can accurately model the system for each pipeline stage and each micro-architecture event, it can potentially be used as a debugging tool in the chip-design flow such as a debugging tool for finding design bugs in post place&route stage and/or for finding manufacturing bugs/defects in post-fabrication. In contrary with signal modeling, in this scenario, the signals simulated by the simulator can be assumed as the "ground-truth" or "expected" signal where the signals emanated by the hardware have to be matched to these *reference* models. A deviation from the *reference* model obtained by the simulations indicates that there is an unwanted change/error in the hardware.

The main advantage of this approach compared to existing standard testing methods is that the proposed approach is ***zero-overhead*** and does not require any testing infrastructure on the system which, in turn, saves a significant amount of area and reduces complexity.

To further demonstrate the feasibility of this approach, Figure 11 shows a scenario where there is a bug in designing a multiplier in the Execution stage. The multiplier is designed such that it calculates the result of multiplying two 16-bits operands in three cycles where the majority of the activity (i.e., writing the output register, etc.) takes place in the last (third) cycle. However, as seen from the figure, the amplitude of the measured signal (top) in the third cycle of the execution (shown in a red circle) is significantly lower than that of in the simulation (bottom). Further investigation reveals that instead of properly multiplying two 16-bits data, the designed multiplier only uses the lower half (i.e., 8-bit data) of each operand and ignores the upper half of those inputs hence results in a significantly lower activity factor and thus much smaller signal strength.

7. RELATED WORK

Much work has been done to prevent particular side-channel attacks [2, 32, 57, 58, 59, 60, 61], either by removing the tie between sensitive information and the side-channel signal, or by trying to make the signal more difficult to measure. However, such work mostly focuses on preventing a particular side channel attack in a very specific piece of code and are less focused about the fundamental relationship between the hardware, software, and the side-channel signal.

Strategies for quantifying potential side channel exposure at the micro-architectural and architectural levels are still an open problem. Existing work proposed different methods and/or metrics to estimate the leakage either for a specific type of side-channel (e.g., cache, power, EM, etc.) or alternatively, as a generic framework to estimate the overall leakage for any given side-channel.

Side-Channel Vulnerability Factor (SVF) [32] measures how the side-channel signal correlates with high-level execution patterns (e.g., program phase transitions). While this metric allows overall assessment of the "leakiness" of a particular system and application over a given side-channel, it provides limited insight to 1) computer architects about which architectural and microarchitectural features are the strongest leakers, and to 2) software developers about how to reduce the side-channel leakiness of their code.

To address these limitations, Signal Available to Attacker (SAVAT) method [33] was proposed. SAVAT measures the side-channel signal (particularly EM and power from laptops) created by a specific single-instruction difference in program execution, i.e., the amount of signal made available to a potential attacker who wishes to decide whether the program has executed instruction/event A or instruction/event B. These measurements can be used to determine the potential for information leakage when execution of individual instructions or even sections of code depend on sensitive information. Unfortunately, SAVAT only models the system at ISA-level and ignores the underlying relation of each instruction to the hardware or other instructions in the sequence.

Similar to SAVAT, McCann *et al.* [34] proposed a modeling technique capable of producing a leakage metric at instruction-level for power (and/or EM) side-channel signals on ARM M0/M4 cores. To estimate the leakage for individual instructions, the proposed method only requires knowledge about different characteristics of the system at ISA-level such as data-dependent effects of neighboring instructions in a sequence, register effects, bit-flips, etc. Similar to SAVAT, while the method proposed by McCann *et al.* [34] provides interesting insights about possible sources of leakage, it also ignores the effects of micro-architecture events such as cache miss, branch mis-prediction, etc. on the signal which, as shown in this paper, may lead to making wrong conclusions about the leakage model of the software/system.

Another related work to EMSim is the method proposed by Barenghi and Pelosi [35] where the leakage for individual instructions was calculated by measuring the power consumption between two consecutive cycles and employing the Pearson correlation coefficient between the two measurements. To calculate the leakage, in addition to leveraging the ISA-level information, pipeline model was also used. However, the framework did not consider any micro-architecture events, nor pipeline stalls and only accounts the number of cycles that takes for each instruction to execute. It also did not model the individual effect of each stage on the others and the overall signal. In this work, however, we took a more systematic and accurate approach by considering different micro-architecture events and hardware effects.

Also related to this work are work on leveraging EM signals for profiling [28, 62, 63]. Spectral Profiling [62] compares short-term spectra of EM emanations to those obtained during training to recognize which loop-granularity region of code the signal corresponds to. EMPROF [63] analyzes the system's EM emanations to identify processor stalls that are associated with last-level cache (LLC) misses. Compared to these frameworks, instead of leveraging the existing EM side-channel signal for profiling, our work takes a more holistic approach and systematically models the underlying relation between the software and hardware and provides insights on how and why these EM side-channel signals are created due to a variety of micro-architectural and hardware activities. Using this approach, EMSim can be used for a variety of purposes beyond only for program/memory profiling (e.g., hardware modeling, leakage estimation, compiler development, etc.).

Another body of work related to this paper are the cycle-accurate models/tools to simulate power and/or microarchitecture [12,13,14,15,16,17,18,19]. While these models can accurately model the power consumption at each cycle, they are different from this work and hence may not be a proper tool for simulating EM side-channel signals for two main reasons: *First*, while these methods do consider the activity factor to calculate power, they often treat all the bit-flips equally. However, as shown in this paper, depending on the design, not all flips equally contribute to the overall signal. Ignoring this fact can lead to inaccurate modeling (see Figure 3). *Second*, depending on the architecture, different stages might have different effect on each other and the overall signal. Without properly modeling these effects, the overall signal can not be modeled (see Figure 9).

Loosely related to our work are methods for instruction tracking/intrusion detection based on side-channel signals [64, 65, 66, 67]. These methods often use different signal processing methods (e.g., Markov Model, Statistical tests, etc.) and/or machine learning techniques to find the most likely executed instruction based on the side-channel trace. While they are effective in providing a non-intrusive, zero-overhead method for profiling/intrusion detection, they are not designed to model the signal and/or provide any information or insight about how these signals are generated.

8. CONCLUSIONS

This paper presented *EMSim*, an approach that enables simulation of the EM side-channel signals cycle-by-cycle using the detailed micro-architectural model of the device. To evaluate *EMSim*, we compared its simulation-derived signals to signals measured from real hardware and found that they match very closely. To gain further insight, we also experimentally identified how the accuracy of the simulated signals degrades when key micro-architectural features and other hardware behaviors are omitted from the simulation model.

We envision a variety of uses for EMSim. For hardware, software, and compiler developers, it allows EM leakage to be quantified without having to build actual hardware and/or actually measure signals. More importantly, it allows simulated signals to be broken down and attributed to specific parts of the hardware design and program code. Furthermore, when hardware prototypes are available, significant discrepancies between the signal generated by EMSim and actual EM emanations can be used to identify where the actual hardware design differs from the simulated microachitecture, which can be used to debug the hardware and/or to help refine the simulation model to more closely match the hardware.

To extend *EMSim* to simulate more complex processors, we believe similar multi-input-single-output methodology can be used, where each pipeline stage acts as a single source. For out-of-order processors we expect higher *baseline hardware amplitude* for each stage (specifically `fetch` and `decode` due to the more complex hardware design of these stages). We also expect different values for activity factors and coefficients for individual stages. However, given that regardless of the implementation, the root cause of creating side-channel signals are bit-flips at the gate-level, we do not expect any fundamental *modeling* difference between in-order and OoO designs.

9. REFERENCES

[1] M. Alam, H. A. Khan, M. Dey, N. Sinha, R. Callan, A. Zajic, and M. Prvulovic, "One&done: A single-decryption em-based attack on openssl's constant-time blinded RSA," in *27th USENIX Security Symposium (USENIX Security 18)*, (Baltimore, MD), pp. 585–602, USENIX Association, 2018.

[2] M. Backes, M. Dürmuth, S. Gerling, M. Pinkal, and C. Sporleder, "Acoustic side-channel attacks on printers," in *Proceedings of the 19th USENIX Conference on Security*, USENIX Security'10, (Berkeley, CA, USA), pp. 20–20, USENIX Association, 2010.

[3] J. Balasch, B. Gierlichs, O. Reparaz, and I. Verbauwhede, "Dpa, bitslicing and masking at 1 ghz," *IACR Cryptology ePrint Archive*, vol. 2015, p. 727, 2015.

[4] D. J. Bernstein, J. Breitner, D. Genkin, L. Groot Bruinderink, N. Heninger, T. Lange, C. van Vredendaal, and Y. Yarom, "Sliding right into disaster: Left-to-right sliding windows leak," in *Cryptographic Hardware and Embedded Systems – CHES 2017* (W. Fischer and N. Homma, eds.), (Cham), pp. 555–576, Springer International Publishing, 2017.

[5] J. Brouchier, T. Kean, C. Marsh, and D. Naccache, "Temperature attacks," *IEEE Security Privacy*, vol. 7, pp. 79–82, March 2009.

[6] S. Chari, J. R. Rao, and P. Rohatgi, "Template attacks," in *Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '02, (London, UK, UK), pp. 13–28, Springer-Verlag, 2003.

[7] D. Genkin, L. Pachmanov, I. Pipman, E. Tromer, and Y. Yarom, "Ecdsa key extraction from mobile devices via nonintrusive physical side channels," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, (New York, NY, USA), pp. 1626–1638, ACM, 2016.

[8] D. Genkin, L. Pachmanov, I. Pipman, and E. Tromer, "Stealing keys from pcs using a radio: Cheap electromagnetic attacks on windowed exponentiation," in *Cryptographic Hardware and Embedded Systems – CHES 2015* (T. Güneysu and H. Handschuh, eds.), (Berlin, Heidelberg), pp. 207–228, Springer Berlin Heidelberg, 2015.

[9] A. Zajić and M. Prvulovic, "Experimental demonstration of electromagnetic information leakage from modern processor-memory systems," *IEEE Transactions on Electromagnetic Compatibility*, vol. 56, pp. 885–893, Aug 2014.

[10] G. Camurati, S. Poeplau, M. Muench, T. Hayes, and A. Francillon, "Screaming channels: When electromagnetic side channels meet radio transceivers," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, (New York, NY, USA), pp. 163–177, ACM, 2018.

[11] Z. Hadjilambrou, S. Das, M. A. Antoniades, and Y. Sazeides, "Leveraging cpu electromagnetic emanations for voltage noise characterization," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 573–585, Oct 2018.

[12] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, (New York, NY, USA), pp. 469–480, ACM, 2009.

[13] S. Li, K. Chen, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "Cacti-p: Architecture-level modeling for sram-based structures with advanced leakage reduction techniques," in *Proceedings of the International Conference on Computer-Aided Design*, ICCAD '11, (Piscataway, NJ, USA), pp. 694–701, IEEE Press, 2011.

[14] S. J. E. Wilton and N. P. Jouppi, "Cacti: an enhanced cache access and cycle time model," *IEEE Journal of Solid-State Circuits*, vol. 31, pp. 677–688, May 1996.

[15] E. K. Ardestani and J. Renau, "Esesc: A fast multicore simulator using time-based sampling," in *Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, HPCA '13, (Washington, DC, USA), pp. 448–459, IEEE Computer Society, 2013.

[16] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, pp. 1–7, Aug. 2011.

[17] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, S. Sarangi, P. Sack, K. Strauss, and P. Montesinos, "SESC simulator," January 2005. http://sesc.sourceforge.net.

[18] A. Patel, F. Afram, S. Chen, and K. Ghose, "Marss: A full system simulator for multicore x86 cpus," in *2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1050–1055, June 2011.

[19] T. E. Carlson, W. Heirman, S. Eyerman, I. Hur, and L. Eeckhout, "An evaluation of high-level mechanistic core models," *ACM Transactions on Architecture and Code Optimization (TACO)*, 2014.

[20] M. Taram, A. Venkat, and D. Tullsen, "Mobilizing the micro-ops: Exploiting context sensitive decoding for security and energy efficiency," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 624–637, June 2018.

[21] A. Althoff, J. McMahan, L. Vega, S. Davidson, T. Sherwood, M. B. Taylor, and R. Kastner, "Hiding intermittent information leakage with architectural support for blinking," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, ISCA '18, (Piscataway, NJ, USA), pp. 638–649, IEEE Press, 2018.

[22] M. Andrysco, A. Nötzli, F. Brown, R. Jhala, and D. Stefan, "Towards verified, constant-time floating point operations," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, (New York, NY, USA), pp. 1369–1382, ACM, 2018.

[23] J. Yu, L. Hsiung, M. E. Hajj, and C. W. Fletcher, "Data oblivious isa extensions for side channel-resistant and high performance computing," in *Proceedings of the Annual Network and Distributed System Security Symposium (NDSS)*, NDSS '19, 2019. `https://eprint.iacr.org/2018/808`.

[24] A. Rane, C. Lin, and M. Tiwari, "Secure, precise, and fast floating-point operations on x86 processors," in *Proceedings of the 25th USENIX Conference on Security Symposium*, SEC'16, (Berkeley, CA, USA), pp. 71–86, USENIX Association, 2016.

[25] K. Nayak, C. W. Fletcher, L. Ren, N. Chandran, S. V. Lokam, E. Shi, and V. Goyal, "Hop: Hardware makes obfuscation practical.," in *Proceedings of the Annual Network and Distributed System Security Symposium (NDSS)*, NDSS '17, 2017.

[26] C. Liu, A. Harris, M. Maas, M. Hicks, M. Tiwari, and E. Shi, "Ghostrider: A hardware-software system for memory trace oblivious computation," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '15, (New York, NY, USA), pp. 87–101, ACM, 2015.

[27] A. Rane, C. Lin, and M. Tiwari, "Raccoon: Closing digital side-channels through obfuscated execution," in *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC'15, (Berkeley, CA, USA), pp. 431–446, USENIX Association, 2015.

[28] D. I. Gorman, M. R. Guthaus, and J. Renau, "Architectural opportunities for novel dynamic emi shifting (demis)," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-50 '17, (New York, NY, USA), pp. 774–785, ACM, 2017.

[29] J. Wichelmann, A. Moghimi, T. Eisenbarth, and B. Sunar, "Microwalk: A framework for finding side channels in binaries," in *Proceedings of the 34th Annual Computer Security Applications Conference*, ACSAC '18, (New York, NY, USA), pp. 161–173, ACM, 2018.

[30] J. Chen, Y. Feng, and I. Dillig, "Precise detection of side-channel vulnerabilities using quantitative cartesian hoare logic," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, (New York, NY, USA), pp. 875–890, ACM, 2017.

[31] M. Wu, S. Guo, P. Schaumont, and C. Wang, "Eliminating timing side-channel leaks using program repair," in *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2018, (New York, NY, USA), pp. 15–26, ACM, 2018.

[32] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. Stolfo, "On the feasibility of online malware detection with performance counters," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, (New York, NY, USA), pp. 559–570, ACM, 2013.

[33] 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 *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-47, (Washington, DC, USA), pp. 242–254, IEEE Computer Society, 2014.

[34] D. McCann, E. Oswald, and C. Whitnall, "Towards practical tools for side channel aware software engineering: Grey box' modelling for instruction leakages," in *Proceedings of the 26th USENIX Conference on Security Symposium*, SEC'17, (Berkeley, CA, USA), pp. 199–216, USENIX Association, 2017.

[35] A. Barenghi and G. Pelosi, "Side-channel security of superscalar cpus: Evaluating the impact of micro-architectural features," in *Proceedings of the 55th Annual Design Automation Conference*, DAC '18, (New York, NY, USA), pp. 120:1–120:6, ACM, 2018.

[36] 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 Transactions on Information Forensics and Security*, vol. 13, no. 3, pp. 605–620, 2017.

[37] A. Waterman and K. Asanovic, "Risc-v instruction reference." `https://content.riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf`, 2019 (accessed Nov. 6, 2019).

[38] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanovi, "The risc-v instruction set manual. volume 1: User-level isa, version 2.0," tech. rep., CALIFORNIA UNIV BERKELEY DEPT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCES, 2014.

[39] T.-Y. Yeh and Y. N. Patt, "Two-level adaptive training branch prediction," in *Proceedings of the 24th Annual International Symposium on Microarchitecture*, MICRO 24, (New York, NY, USA), pp. 51–61, ACM, 1991.

[40] B. B. Yilmaz, E. M. Ugurlu, A. Zajic, and M. Prvulovic, "Instruction level program tracking using electromagnetic emanations," in *Proceedings of the SPIE*, vol. 11011, International Society for Optics and Photonics, 2019.

[41] A. V. Oppenheim, A. S. Willsky, and S. H. Nawab, *Signals &Amp; Systems (2Nd Ed.)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1996.

[42] W. van Eck, "Electromagnetic radiation from video display units: An eavesdropping risk?," *Computers and Security*, vol. 4, no. 4, pp. 269 – 286, 1985.

[43] D. J. Hand, "Statistical concepts: A second course, fourth edition by richard g. lomax, debbie l. hahs-vaughn," *International Statistical Review*, vol. 80, no. 3, pp. 491–491, 2012.

[44] Terassic, "De0-cv." `http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=364`, 2019 (accessed Nov. 6, 2019).

[45] Tektronix, "tbs1000-digital-storage-oscilloscope." `https://www.tek.com/oscilloscope/tbs1000-digital-storage-oscilloscope`, 2019 (accessed Nov. 6, 2019).

[46] AARONIA, "Datasheet: Rf near field probe set dc to 9ghz." `http://www.aaronia.com/Datasheets/Antennas/RF-Near-Field-Probe-Set.pdf`, 2019 (accessed April. 1, 2019).

[47] W. B. Frakes and R. Baeza-Yates, *Information retrieval: Data structures & algorithms*, vol. 331. Prentice Hall Englewood Cliffs, NJ, 1992.

[48] Terassic, "De1 evaluation board." `https://www.intel.com/content/www/us/en/programmable/solutions/partners/partner-profile/terasic-inc-/board/altera-de1-board.html`, 2019 (accessed Nov. 6, 2019).

[49] Xilinx, "Digilent evalaution board." `https://store.digilentinc.com/arty-a7-artix-7-fpga-development-board-for-makers-and-hobbyists/`, 2019 (accessed Nov. 6, 2019).

[50] G. Becker, J. Cooper, E. DeMulder, G. Goodwill, J. Jaffe, G. Kenworthy, T. Kouzminov, A. Leiserson, M. Marson, and P. Rohatgi, "Test vector leakage assessment (tvla) methodology in practice," in *International Cryptographic Module Conference*, 2013.

[51] G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi, "A testing methodology for side-channel resistance validation," 2011.

[52] A. Heuser, W. Schindler, and M. Stöttinger, "Revealing side-channel issues of complex circuits by enhanced leakage models," in *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1179–1184, March 2012.

[53] T. Ming, W. Pengbo, M. Xiaoqi, C. Wenjie, Z. Huanguo, P. Guojun, and J. Danger, "An efficient sca leakage model construction method under predictable evaluation," *IEEE Transactions on Information Forensics and Security*, vol. 13, pp. 3008–3018, Dec 2018.

[54] S. Mangard, E. Oswald, and F. . Standaert, "One for all - all for one: unifying standard differential power analysis attacks," *IET Information Security*, vol. 5, pp. 100–110, June 2011.

[55] P. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, (Berlin, Heidelberg), pp. 388–397, Springer-Verlag, 1999.

[56] 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 Transactions on Information Forensics and Security*, pp. 1–1, 2019.

[57] A. Nazari, N. Sehatbakhsh, M. Alam, A. Zajic, and M. Prvulovic, "Eddie: Em-based detection of deviations in program execution," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, (New York, NY, USA), pp. 333–346, ACM, 2017.

[58] Y. Han, S. Etigowni, H. Liu, S. Zonouz, and A. Petropulu, "Watch me, but don't touch me! contactless control flow monitoring via electromagnetic emanations," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, (New York, NY, USA), pp. 1095–1108, ACM, 2017.

[59] Y. Liu, L. Wei, Z. Zhou, K. Zhang, W. Xu, and Q. Xu, "On code execution tracking via power side-channel," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, (New York, NY, USA), pp. 1019–1031, ACM, 2016.

[60] M. Taha and P. Schaumont, "Key updating for leakage resiliency with application to aes modes of operation," *IEEE Transactions on Information Forensics and Security*, vol. 10, pp. 519–528, March 2015.

[61] Z. He and R. B. Lee, "How secure is your cache against side-channel attacks?," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-50 '17, (New York, NY, USA), pp. 341–353, ACM, 2017.

[62] N. Sehatbakhsh, A. Nazari, A. Zajic, and M. Prvulovic, "Spectral profiling: Observer-effect-free profiling by monitoring em emanations," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1–11, Oct 2016.

[63] M. Dey, A. Nazari, A. Zajic, and M. Prvulovic, “Emprof: Memory profiling via em-emanation in iot and hand-held devices,” in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 881–893, Oct 2018.

[64] S. S. Clark, B. Ransford, A. Rahmati, S. Guineau, J. Sorber, K. Fu, and W. Xu, “Wattsupdoc: Power side channels to nonintrusively discover untargeted malware on embedded medical devices,” in *Proceedings of the 2013 USENIX Conference on Safety, Security, Privacy and Interoperability of Health Information Technologies*, HealthTech’13, (Berkeley, CA, USA), pp. 9–9, USENIX Association, 2013.

[65] N. Sehatbakhsh, M. Alam, A. Nazari, A. Zajic, and M. Prvulovic, “Syndrome: Spectral analysis for anomaly detection on medical iot and embedded devices,” in *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 1–8, April 2018.

[66] C. R. Aguayo González and J. H. Reed, “Power fingerprinting in sdr integrity assessment for security and regulatory compliance,” *Analog Integr. Circuits Signal Process.*, vol. 69, pp. 307–327, Dec. 2011.

[67] C. Song, H. Moon, M. Alam, I. Yun, B. Lee, T. Kim, W. Lee, and Y. Paek, “Hdfi: Hardware-assisted data-flow isolation,” in *2016 IEEE Symposium on Security and Privacy (SP)*, pp. 1–17, May 2016.