Power Side-Channel Attacks on BNN Accelerators in Remote FPGAs

Shayan Moini, Shanquan Tian, Daniel Holcomb, Jakub Szefer, and Russell Tessier

***Abstract*—To lower cost and increase the utilization of Cloud Field-Programmable Gate Arrays (FPGAs), researchers have recently been exploring the concept of multi-tenant FPGAs, where multiple independent users simultaneously share the same remote FPGA. Despite its benefits, multi-tenancy opens up the possibility of malicious users co-locating on the same FPGA as a victim user, and extracting sensitive information. This issue becomes especially serious when the user is running a machine learning algorithm that is processing sensitive or private information. To demonstrate the dangers, this paper presents a remote, power-based side-channel attack on a deep neural network accelerator running in a variety of Xilinx FPGAs and also on Cloud FPGAs using Amazon Web Services (AWS) F1 instances. This work in particular shows how to remotely obtain voltage estimates as a deep neural network inference circuit executes, and how the information can be used to recover the inputs to the neural network. The attack is demonstrated with a binarized convolutional neural network used to recognize handwriting images from the MNIST handwritten digit database. With the use of precise time-to-digital converters for remote voltage estimation, the MNIST inputs can be successfully recovered with a maximum normalized cross-correlation of 79% between the input image and the recovered image on local FPGA boards and 72% on AWS F1 instances. The attack requires no physical access nor modifications to the FPGA hardware.**

***Index Terms*—Remote Attacks, Deep Neural Networks, Convolutional Neural Networks, Side-channel Attacks, Power Attacks, Time-to-Digital Converters (TDCs)**

I. INTRODUCTION

Cloud FPGAs have recently emerged as an important computing paradigm where users can rent access to high-end FPGA resources on-demand from public cloud providers. Most major cloud providers now offer some form of remote, pay-per-use access to FPGAs [1], [2], [3], [4], [5]. Furthermore, recent proposals for multi-tenancy have the promise of increasing FPGA utilization, especially in data center settings, by fitting multiple users' designs onto a single FPGA at the same time. A number of research projects [6], [7], [8], [9], [10], [11] have explored how to implement FPGA multi-tenancy. The sharing of an FPGA by many users, unfortunately, opens up multi-tenant FPGA platforms to many new, potential attacks in which a malicious user can be co-located next to a victim user.

Once co-located, a malicious user can try to learn information about the victim through a side channel. When multi-tenant FPGAs are deployed in a remote data center, the malicious user is limited to only using side channels that do not require physical access. For example, previous work [12], [13], [14], [15], [16] has shown that crosstalk between long routing wires on an FPGA can be used to leak sensitive information from cryptographic circuits using remote attacks. Meanwhile, voltage and power-based attacks [17] have been used to remotely extract encryption keys for both RSA [18] and AES [19] using circuits implemented on an FPGA by a malicious user.

The danger of such attacks becomes especially worrisome as there is more and more interest in the FPGA acceleration of machine learning for image recognition, or other tasks, where sensitive information is processed. Existing work on machine learning (ML) algorithm accelerators, and especially deep neural networks, using FPGAs [20], [21], [22], [23] has shown that these algorithms, when deployed on FPGAs, can significantly speed up the inference operations. Further, many cloud providers tout FPGAs for acceleration of ML workloads [24].

To show potential threats when machine learning accelerators are combined with multi-tenant FPGA deployment, this work demonstrates a remote power-based side-channel attack on a binarized convolutional neural network (BNN) in an FPGA. In our attack, voltage fluctuations, caused by the changes in the power consumption of the convolution unit in the BNN, are used to accurately reconstruct images that are input into the BNN accelerator during the inference operations. Being able to recover the images that are processed by the ML algorithm could reveal sensitive imagery [25]. To highlight the dangers of the potential attacks, this work shows how to recover such input images remotely, where an attacker uses a time-to-digital (TDC) converter as a remote power sensor in a multi-tenant FPGA setting. Outside of multi-tenant scenarios, the same attack could be used whenever the ML accelerator resides on an FPGA alongside untrusted 3rd-party intellectual property (IP) cores that might contain unknown sensing circuits.

Our attack can be performed *remotely with no physical hardware access by the attacker*. Furthermore, the attack works without knowledge of the neural network parameters that could facilitate attacks involving power dissipation templates. We demonstrate the details of our attack on the convolution unit of a BNN-based circuit that is used to recognize the handwriting images from the MNIST handwritten digit database [26]. Our attack and corresponding image recovery is successfully demonstrated on multiple generations of Xilinx FPGAs including a ChipWhisperer CW305 board [27] (Artix-7), a ZCU104 board [28] (Zynq UltraScale+), a VCU118 board [29] (Virtex UltraScale+), and Amazon AWS F1 instances [30] (Virtex

Shayan Moini, Daniel Holcomb, and Russell Tessier are with the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003 USA (e-mail: smoini@umass.edu; dholcomb@umass.edu; tessier@umass.edu). Shanquan Tian and Jakub Szefer are with the Department of Electrical Engineering, Yale University, New Haven, CT 06511 USA (email: shanquan.tian@yale.edu; jakub.szefer@yale.edu)

UltraScale+). Based on the evaluation we show that clearly recognizable images can be retrieved for all tested input images from the MNIST database. A maximum cross-correlation of 79% is observed between the original and recovered images on local FPGA boards and 72% on AWS F1 instances.

In summary, our work makes the following contributions:

- We demonstrate a side channel attack on an ML accelerator implemented in remote FPGAs. Input images to the accelerator are reconstructed using TDCs that are logically isolated from the accelerator.
- Our attack is shown to work effectively on cloud FPGAs that are part of AWS F1 instances.
- We characterize the effectiveness of the attack using quantitative metrics and examine its robustness to noise.

A. Paper Organization

The remainder of this manuscript is organized as follows. Section II provides background on deep neural networks and existing attacks. Section III gives details of our attack and our experimental approach is described in Section IV. Attack characterization with a ChipWhisperer board is described in Section V. Image extraction results generated from commodity FPGA boards and AWS F1 instances are presented in Section VI. Section VII concludes the manuscript and offers directions for future work.

II. Background and Related Work

In this section, we provide an overview of convolutional neural network models and review previous attacks against FPGA accelerators for deep neural networks.

A. Convolutional Neural Networks

Deep neural networks (DNNs) [31] are a class of artificial neural networks that use multiple layers. In a DNN, each layer is responsible for extracting relevant features, and the output of each layer is passed as the input to the next layer. DNNs combine feature extraction with the classification capability of classical neural networks to map input data to a set of predictions. DNNs can be used to perform, for example, image classification tasks.

Convolutional neural networks (CNNs) [31] are a subset of DNNs that are mostly used for classifying multi-dimensional data (e.g., images or video). The main distinctive property of CNNs is the convolution layer, which implements feature extraction by performing a convolution operation between the high-dimensional input data (called input feature maps) and kernels (small matrices of parameters that are computed during the training phase) to generate the output of the layer (called output feature maps). As shown in Figure 1, the output feature maps of each layer are passed to the next layer as the input feature maps. Other layers in a typical CNN include a non-linear function (creating complex input-output mappings), pooling (reducing the dimensionality of input feature maps by different methods, e.g., max pooling), batch normalization (normalizing input feature maps to decrease their variance), and fully-connected layers (where each element of an output feature map is calculated by point-wise multiplication between a whole input feature map and a kernel of the same size).

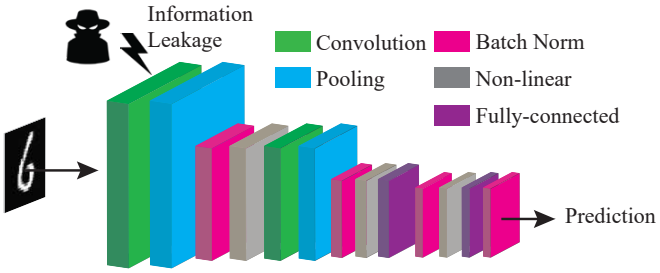


Fig. 1: Overview of steps in the CNN used in this work. The details of the architecture are explained in Table I, and the threat model is shown in Figure 2.

B. Binarized Neural Networks

Binarized neural networks (BNNs) [32] use aggressive quantization so that each element of the convolution kernel can be represented as either -1 or $+1$. This quantization helps reduce the memory bandwidth needed to load network parameters from off-chip memory during the execution of each layer and replaces multipliers with simple add and subtract operations. In BNNs, all convolution input feature maps and kernels are comprised of binary values except for the first input layer which generally receives its input feature maps as matrices of integers, e.g., representing the pixels of input images.

For this work, we assume the input to the BNN is a grayscale image with each pixel being represented by an integer (0 to 255). This image is the input feature map to the first convolution layer which convolves the input with $n \times n$ binary kernels. To perform the convolution, each element of the convolution output (an output feature map) is calculated by multiplying a kernel with a $n \times n$ window of the input feature map and summing the resulting values. Sweeping an $n \times n$ kernel across the input feature map generates an output feature map. The convolution operation is followed by a maximum pooling operation which reduces the size of its input feature maps by choosing the largest value out of each $k \times k$ window of each input feature map and discarding the rest. The next layer, batch normalization, normalizes its input feature maps value by value. Here, the numbers are represented as fixed-point values between -1 and +1. The non-linear function layer truncates the output feature map values of the batch normalization layer into either -1 or +1 based on their sign. This process is replicated for other convolution steps with the exception that their input feature maps are the binary outputs of the previous non-linear function layer.

For this work, the BNN is pre-trained with the MNIST database on an Nvidia GTX 1080 GPU, and the derived parameters, including convolution kernel values, are used in the BNN accelerator on an FPGA. We used the Keras framework [33] to train the BNN. The trained network is used during the inference stage to classify the input images of digits into one of ten categories (0 to 9). The BNN contains two convolution layers and two fully connected layers. Convolution is performed with a standard 64 kernels per layer [33]. All convolution and fully-connected layers, except for the first layer, receive binary inputs, have 3×3 binary kernels (e.g., $n = 3$), and generate

TABLE I: Details of the trained BNN. The accuracy of the trained network with the MNIST test set is 98.24%.

Layer #	Layer Type	Input Size	Kernel Size
1	Convolution	28×28	3×3×64
2	Pooling	28×28×64	2×2
3	Batch norm	14×14×64	-
4	Non-linear function	14×14×64	-
5	Convolution	14×14×64	(3×3×64)×64
6	Pooling	14×14×64	2×2
7	Batch norm	7×7×64	-
8	Non-linear function	7×7×64	-
9	Fully-connected	7×7×64	500 × (7×7×64)
10	Batch norm	500	-
11	Non-linear function	500	-
12	Fully-connected	500	10×500
13	Batch norm	10	-

output feature maps in integer format. The first convolution layer receives the input image, a 28×28 pixel grayscale image of a handwritten digit, as a matrix of integer values between 0 and 255 and performs the convolution operation with binary kernels. The output of the network is a ten element array that shows the likelihood of the input image being each of the ten digits with the highest number being the predicted digit for the input image. Table I shows the details of the BNN architecture used in this work.

C. Attacks on DNN FPGA Accelerators

Several researchers [25], [34], [35], [36] have explored side-channel attacks on DNN accelerators on FPGAs. All of these approaches used physical access to the FPGA to collect needed information for the attacks. Meanwhile, we present a remote, power-based side channel that does not require physical access to FPGA supply voltage pins, uses on-chip voltage sensors to detect voltage fluctuations, and is demonstrated to work with four different FPGA boards.

Wei et al. [25] used power traces recovered from FPGA voltage supply pins to extract the input image data of a BNN. An oscilloscope was used to measure the voltage drop across a 1Ω resistor placed on the power supply rail of a SAKURA-G board [37]. Their attack method relies on per-clock cycle power consumption of convolution operations. Dubey et al. [34] targeted an FPGA accelerator of a fully-connected BNN. They were able to successfully extract the parameter values of the model by finding the highest correlation of the model power consumption for a collection of known input values. Voltage traces gathered by an oscilloscope connected to the supply voltage pin of a Kintex-7 FPGA on a SAKURA-X board [38] were used to perform this attack.

Yoshida et al. [36] used FPGA side-channel electromagnetic leakage measurements to extract the kernel values of a multi-layer perceptron (MLP) accelerator in the presence of weight encryption. An external probe was used to collect these measurements. Hua et al. [35] extracted the structure of a CNN, including the size of the input feature map and kernels of each layer by studying the off-chip memory access patterns of the FPGA accelerator while the operations of each layer were performed. Their attack revealed the structure of neural networks in the presence of weight encryption. However, they did not reverse engineer the input feature map values.

Boutros et al. [39] recently performed a fault-injection attack on a CNN implemented in a remote Intel Stratix 10 FPGA. Their experiments showed that the deliberate use of excessive power consumption on the FPGA was not sufficient to cause classification errors in the CNN due to large timing margins in the circuit implementation and redundancy in the CNN model.

D. Voltage Sensing Using TDCs

In FPGAs, small drops in supply voltage occur in the vicinity of power consumption due to both resistive and inductive drops in the power distribution network and the chip packaging [40]. Given that the propagation delay of combinational logic varies as a function of supply voltage, circuit delay in a specially designed sensor circuit can be used as a proxy for measuring the changes in the supply voltage. This approach is commonly used in voltage sensors based on ring oscillator (RO) [41] or TDC [19] circuits. ROs need long measurement periods for precision and are therefore unsuitable for side channels that rely on fast transients. Meanwhile, TDCs are often used to overcome the limitations of ring oscillator-based sensors [41] and have been shown to effectively obtain side channel information on FPGAs [19]. In TDCs, each measurement reflects the delay of a circuit within a single clock cycle by observing how far through a tapped delay line a signal can travel during the cycle. This makes TDC sensors suitable for sensing short transient voltage fluctuations on the order of a single clock cycle. Because delay changes are only observable if they cause the signal to reach the next tap in the delay line, the precision of a TDC is limited by the delay between successive taps. As we show in Section III-B, following others who previously exploited TDC designs [19], [41], the high-speed carry logic in modern FPGAs makes a suitable delay line with taps that are on the order of 5-25 picoseconds (ps) apart, depending on the FPGA technology and architecture.

This manuscript significantly extends an earlier work that examined FPGA image extraction from a BNN model using TDCs [42]. We comprehensively explore the issue of TDC-based image extraction from BNNs in FPGAs by applying denoising to recovered images and deliberately stressing the FPGA power distribution network. Unlike earlier work, our attack is applied to AWS F1 platforms, a commercial cloud FPGA environment.

III. DETAILS OF THE ATTACK

In this section, we provide an overview of our threat model. We then focus on the details of the attack and its implementation in a multi-tenant FPGA setting.

A. Threat Model

This work focuses on a multi-tenant FPGA scenario where the victim user is running a machine learning inference algorithm on a hardware module that is co-located on the same FPGA with the malicious user's modules. The adversary simultaneously uses the FPGA platform without sharing logic

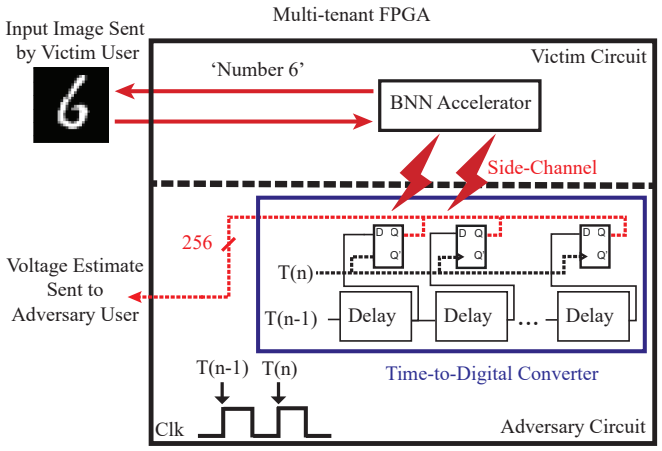


Fig. 2: Overview of attack implementation. The TDC outputs voltage estimates for each clock cycle of the first convolution layer. These estimates are used to reconstruct the input image.

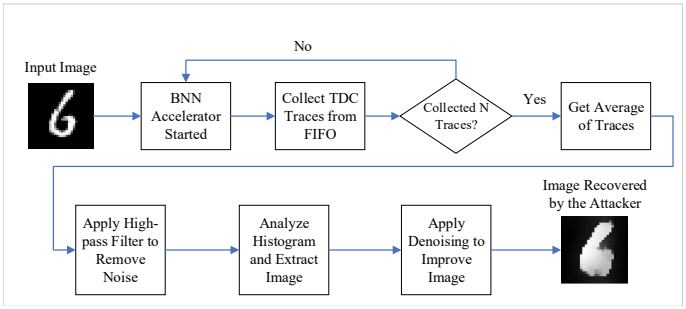


Fig. 3: Steps of the attack.

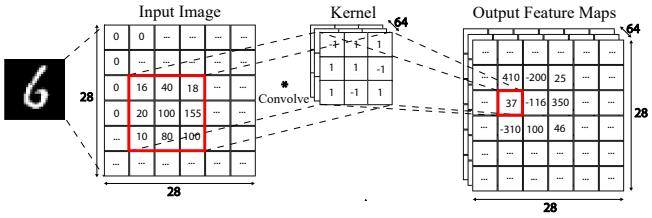


Fig. 4: Detailed view of the first convolution layer in BNN. The value of 37 shown in the output feature map is generated from the 3×3 input image on the left and the kernel.

or I/O resources with the victim. The victim circuit's input (e.g., the input image) is sent to the BNN accelerator in the FPGA in a secure manner (e.g., the input may be encrypted). The same input image is sent by the victim to the FPGA multiple times, a common case in video processing (e.g., of surveillance images). It is assumed that the adversary is not able to access the inputs. Hence the goal of the adversary is to learn the inputs. Further, the adversary is not able to learn the inputs through information leakage (e.g., crosstalk) on the input wires, which would make the attack trivial. The output is likewise assumed to be securely sent back to the user, and the adversary is not able to learn the output directly (if they did, they again would not need the attack).

In this work the focus is on using a TDC to measure voltage changes. The data from the TDC is used by the adversary to estimate the voltage drop across the FPGA power distribution network (PDN) during the execution of the convolution layer, as the BNN accelerator does the image classification. The acquired voltage estimates serve as a side channel that can be used to extract the victim's input image data. The recovered image approximates the input image by distinguishing between foreground and background pixels of the image.

B. Attack Implementation

The attack implementation details are shown in Figure 2. In this setup, there is a victim circuit and an attacker circuit co-located on same FPGA. To extract the input image from the BNN accelerator, the adversary focuses on the first convolution layer which directly processes the input image. The TDC outputs voltage estimates during each clock cycle of the interval when the BNN accelerator processes the first convolution layer. The estimates are measured using the TDC sensor.

In the first convolution layer, an image is convolved with multiple distinct kernels to generate multiple output feature maps. In our attack, we use a voltage estimate trace from the execution of the first kernel of the first convolution layer for an input image. Since we assume that the same image is evaluated by the FPGA accelerator multiple times, multiple (N) similar traces are collected using the same input image. After collecting multiple traces, the adversary takes the mean of the data values in the traces to obtain a single average trace of the voltage estimates during the execution of the first kernel of the first convolution layer. A high-pass filter is then used to remove noise. We leverage the observation that the background and foreground pixels can then be distinguished by analyzing the different magnitudes of the voltage in a trace of measurements. This information can be represented by a histogram of instance counts of magnitude values in the filtered trace. Points in the histogram are used to label pixels as belonging to the image foreground or background based on the magnitude of their voltage measurement. An image denoising filter is applied to this preliminary recovered image to improve clarity. The result of the analysis is a reconstructed image that approximates the input image that was input to the BNN. The procedure is shown in Figure 3 and discussed in more detail in Section V.

The convolution operation can be represented as [25]:

$$O^{j}_{x,y} = \sum_{i=1}^{M} \left(\sum_{a=0}^{K_x-1} \sum_{b=0}^{K_y-1} \omega^{i,j}_{a,b} \times I^{i}_{xS_x+a,yS_y+b} \right) \quad (1)$$

The $O^{j}_{x,y}$ parameter represents the location (x,y) in the jth output feature map which is calculated by convolving a window (same size as the kernel) of the ith input feature map (I^i) and the corresponding kernel ($\omega^{i,j}_{a,b}$) and then adding the M results together where M equals the number of input feature maps. The S_x and S_y values represent the convolution step sizes which are equal to 1 in our BNN implementation. The K_x and K_y values represent kernel sizes in the x and y dimensions.

For the first convolution layer of a BNN trained on the MNIST handwritten digit database with a 28×28 grayscale

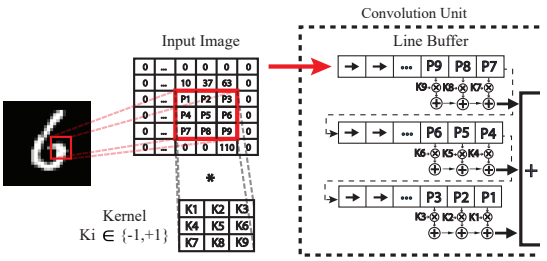


Fig. 5: Detailed view of the convolution unit. Adapte Output is generated from the 3×3 input image, shown in and the kernel.

image as the input and 64 kernels of size 3×3, Equ be simplified to:

$$O^{j}_{x,y} = \sum_{a=0}^{2}\sum_{b=0}^{2} \omega^{j}_{a,b} \times I_{x+a,y+b}, \quad j \in [1,64] \qquad (2)$$

and represented by the operations shown in Figure 4.

The convolution unit uses a line buffer architecture to hold and provide data values to the convolution [25]. As shown by the line buffer at the right in Figure 5, the line buffer is arranged in three rows, each of which processes one line of the convolution operation. The line buffer is a shift register that receives one pixel from the input feature map (the image) per clock cycle and shifts its values to the right. The length of each row in the line buffer matches the length of the input feature map of the convolution operation (28 for the first layer in our implementation). The rightmost word of each row of the line buffer enters the next row from the left, and the rightmost word of the last row is discarded. The rightmost three words of each of the three rows of the line buffer constitute the image window whose values are multiplied with values from the 3×3 kernel. Since binary kernels are used in a BNN, each image pixel in the current image window is added to or subtracted from (based on a kernel value of +1 or -1) the other pixels in one clock cycle using a combinational adder tree. One output feature map value is generated every clock cycle.

An adversary can take advantage of the shared FPGA PDN to sense local supply voltage changes, which can reveal information about the per-cycle power consumption in the convolution unit. The power consumption is due in part to the switching activity in the BNN accelerator, including the convolution unit, which causes supply voltage to be correlated to the data processed (larger magnitude data values lead to increased switching). The small PDN fluctuations are reflected in the sampled values of the time-to-digital converter (TDC), and the TDC samples are then used to recover a facsimile of the input image. The 256-stage TDC architecture is shown in Figure 6. The 256-stage TDC contains an adjustable delay followed by a chain of fast fixed-purpose FPGA elements typically used to perform timing-critical carry operations in arithmetic circuits (*Carry4* or *Carry8* depending on FPGA family). TDC elements are manually placed in the FPGA for controlled and predictable delay that is matched to the clock frequency at which the TDC operates.

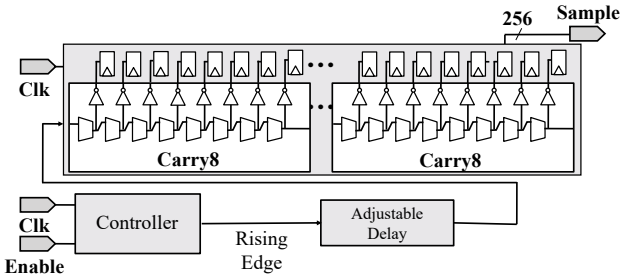


Fig. 6: Architecture of the TDC.

TABLE II: Details of the evaluation boards used for the experiments. The system clock generates the clock for the BNN accelerator and the TDC module.

Board Name	Device	FPGA Family	Clk. (MHz)
ChipWhisperer	XC7A100T	Artix 7	50
ZCU104	XCZU7EV	Zynq UltraScale+	120
VCU118	XCVU9P	Virtex UltraScale+	100
AWS F1	XCVU9P	Virtex UltraScale+	120

The TDC is activated by sending the rising edge of a clock through the adjustable delay and the carry chain to the flip-flops attached to the 256 stages of the carry logic. The Hamming weight of the sample indicates how far through the carry chain the rising edge has propagated by the time the next rising clock edge arrives. When the supply voltage drops, the propagation delay of the circuit increases, and the rising edge will have propagated through fewer carry stages before the next rising clock edge, and hence the sample captured in the flip flops will have a lower Hamming weight. Conversely, if the supply voltage is higher, the propagation delay decreases, and the Hamming weight of the sample increases. The adjustable delay stages before the carry chain calibrate the TDC for process variation which ensures that the sensor will not saturate under small voltage fluctuations that increase or decrease the Hamming weight of the samples. TDC calibration by the attacker is required before the first time an FPGA is used for an attack or following significant changes in device operating conditions (e.g., temperature). The 256-bit TDC measurements are saved in on-chip FIFOs (256-bit word width) at run-time and collected by the adversary after the convolution operation is finished.

IV. Experimental Approach

In this section, we describe the experimental platforms and implementations used to evaluate the efficacy of our attack. Four Xilinx FPGA-based boards, listed in Table II, were used for experimentation. The first three boards in the table, locally situated in the authors' laboratories, were used for characterization and testing. AWS F1 instances listed in the last row of the table were used for cloud-based experiments.

A. Experimental Platforms

The ChipWhisperer CW305 board [27], [43] provides a platform for examining power side-channel attack scenarios.

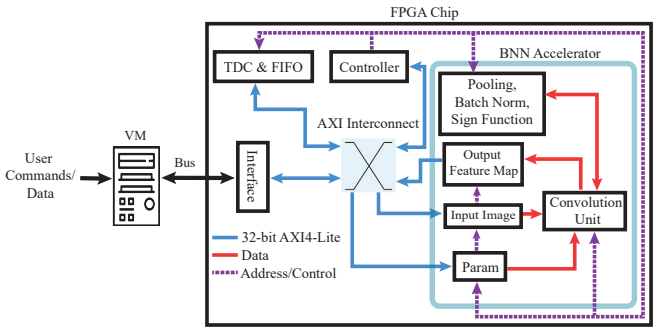


Fig. 7: Overview of the architecture implemented on the ChipWhisper, ZCU104, VCU118 and AWS F1. For AWS F1, the Bus is a PCIe bus and the $Interface$ is a Shell. For the three local boards, the Bus is JTAG and the $Interface$ is a JTAG-to-AXI converter.

The board supports low-noise off-chip voltage measurement using an oscilloscope or a capture board via a low-noise and high-bandwidth connection to the main FPGA 1V DC supply pin. Voltage measurements can be obtained by an adjacent ChipWhisperer-Lite capture board [44] that contains a 10-bit analog-to-digital converter (ADC) with a 105 mega-samples per second (MS/s) sampling rate. As described in Section V, both the capture board and an on-FPGA TDC were used on the ChipWhisperer to obtain voltage traces.

Xilinx ZCU104 [28] and VCU118 [29] evaluation boards were also used for evaluation, with on-FPGA TDC-based sensors used to collect voltage traces. Off-chip FPGA supply voltage measurements were not collected for these two boards. The latter board contains an FPGA that is the same as the one located on AWS F1 instances. For the AWS F1 instances, likewise, an on-FPGA TDC-based sensor was used since there is no physical access for making off-chip supply voltage measurements.

B. Implementation on Local Boards

The implementation of the attack architecture for the three local boards is similar. The BNN accelerator and supporting test circuitry, as well as the TDC and FIFO used for performing the attack, are shown in Figure 7. The data movement between different components of the design takes place through an AXI4-Lite on-chip communication protocol. Off-chip communication (data movement and control commands) uses a Xilinx JTAG-to-AXI converter module that provides direct access to the on-chip AXI bus for the user through a JTAG interface.

The on-chip controller in Figure 7 sets memory addresses and controls the operation of the convolution unit. This controller has registers that set the parameters of the three on-chip block memories used to store on-chip data. *Input Image* stores the input feature map, *Output Feature Map* stores the result of the convolution, and $Param$ stores the binary values of the convolution kernels for the current layer with +1 represented by the bit value 1 and -1 by the bit value 0. For each layer of the BNN, the input feature map and corresponding kernel values are loaded into *Input Image* and $Param$ memories by the user, then the convolution operation is performed, and finally the results are collected from *Output Feature Map*.

C. Implementation on AWS F1

In addition to the local boards, the attack architecture was implemented on AWS F1 instances. The architecture on AWS F1 with functional modules *TDC & FIFO* and *BNN Accelerator* matches the implementations tested on the local FPGAs (Figure 7). The AWS virtual machine (VM) is able to send input images to the FPGA and read TDC outputs from the FPGA, using built-in `peek()` and `poke()` functions. The *TDC & FIFO* modules are physically separated from the *BNN Accelerator* without any direct communication.

Since AWS F1 instance FPGAs currently only support use by a single customer at a time, this setup approximates a multi-tenant scenario. Our attack does work in the presence of the $Shell$ interface circuitry and server environment that are not under user control.

V. Attack Analysis on ChipWhisperer

In this section, we describe characterization experiments using the ChipWhisperer CW305 board. These experiments use both on- and off-FPGA voltage measurements to examine voltage fluctuations during the convolution operation as input images are processed. The ChipWhisper is an *ideal* board in that its bypass capacitors have been removed and dedicated voltage measurement resources are provided. With information gathered from the ChipWhisperer, the attack was then deployed on other, more realistic boards.

A. Off-Chip Characterization of Convolution

In an initial set of experiments, the ADC on the ChipWhisperer-Lite capture board was used to sample the FPGA core supply voltage level at the rising edge of each convolution unit clock cycle. The supply voltage level drops of all 28×28 (784) convolution operations for the first kernel applied to the input image, illustrated in Figure 8a, were measured. The experiment was run 10 times and the mean values of the voltage drops observed at the FPGA supply input at each clock cycle versus the steady state supply voltage were used to generate the trace shown in Figure 9. The 125 orange circles in Figure 9 show clock cycles during which the 125 pixels from the image foreground are used in convolution for the first time (the clock cycle when the foreground pixel is in location $P9$, multiplied by $K9$ in Figure 5). Figure 9 shows that the clock cycles corresponding to foreground pixels have a higher voltage drop compared to other clock cycles. These differences can be used to differentiate between foreground and background pixels.

The voltage drops induced by the foreground pixels can be explained by examining Equation 2. Each pixel of the output feature map ($O_{x,y}$) is calculated using an image window and a kernel. The image (a grayscale picture of a digit with each pixel an integer between 0 and 255) has low-valued pixels (close to 0) for background and high-valued pixels (close to 255) for the foreground. For the calculation of the output feature map, the kernel values are constant. However, the values of the processed input image pixels in specific locations in the line buffer change between background and foreground pixels during the convolution operation. The dynamic power consumption

(a) (b) (c)

Fig. 8: (a) Input image to the convolution unit from the MNIST database, (b) recovered image with supply voltage traces from the ChipWhisperer board, (c) recovered image after applying a denoising algorithm.

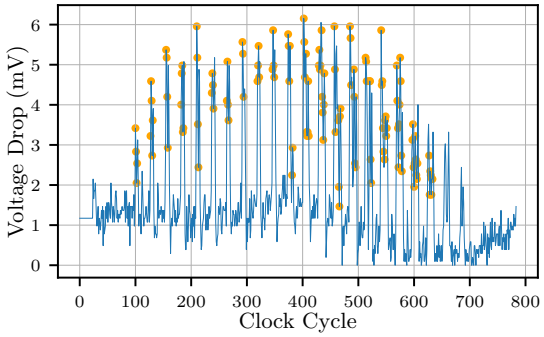


Fig. 9: Voltage trace from the ChipWhisperer FPGA while running the convolution unit shown in Figure 5. The y axis illustrates the absolute value of the measured voltage drop due to convolution unit activity. The 125 orange circles correspond to the clock cycles that process foreground pixels of the input image (Figure 8a).

(and resulting voltage drop) of processing foreground pixels is larger than for background pixels. Specifically, foreground pixels result in the generation of larger magnitude results for the multiply and accumulate operations when the convolution operation processes these pixels. As a result of generating these values, significant switching activity takes place in the adder tree of the convolution unit and resultant voltage drops can be observed.

To illustrate the range of voltage changes due to the convolution of the input image, a histogram of the absolute value of voltage drop measurements in Figure 9 is shown in Figure 10. The histogram contains 40 bins evenly distributed in value between 0 to 6 mV. The boundary between foreground and background pixels can be distinguished with a threshold.

Generally, the processing of background pixels leads to small voltage drops that are clustered on the left of the histogram and the processing of foreground pixels leads to a range of larger voltage drops on the right of the histogram. The threshold can be identified by locating a downward gradient in occurrence counts over multiple voltage bins. In the ChipWhisperer, this transition took place over five bins located just before 2 mV.

In Figure 10, the dashed red line shows the chosen threshold value. All voltage drops created by input pixels that fall to the left of the line are classified as background pixels, while the ones to the right are classified as foreground pixels. To decrease noise, remove stray pixels, and improve the quality of the recovered image, the Rudin-Osher-Fatemi denoising

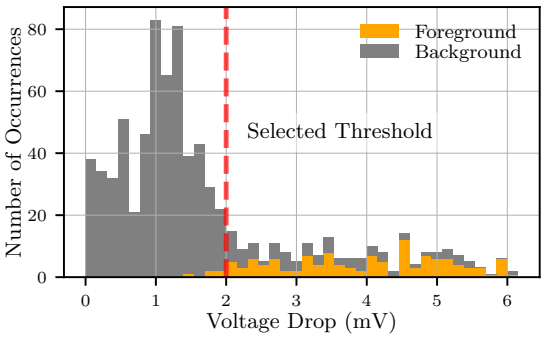


Fig. 10: Histogram of the voltage drop due to convolution unit operation for convolution operations with the same input image and kernel. Each occurrence in the histogram represents the average of ten trials of processing the same pixel and kernel. The bars corresponding to foreground pixels are colored in orange and those corresponding to background pixels are colored dark gray. The selected threshold (boundary) between foreground and background pixels is marked in the histogram.

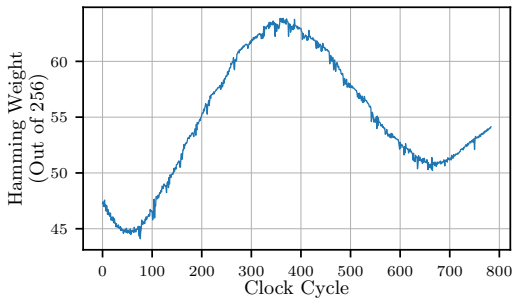
algorithm [45], [46] with τ equal to 0.1 and tv_weight of 40 is applied to this result to generate a recovered image. The input image to the BNN accelerator and the two recovered images using the threshold are shown in Figures 8a, 8b and 8c, respectively. Unlike the input image which has a range of grayscale pixels, the recovered images prior to denoising are binary with 0 value for background pixels and 255 value for foreground pixels. Following denoising, the recovered image has a range of grayscale pixels.

B. TDC-Based Characterization of Convolution Operations

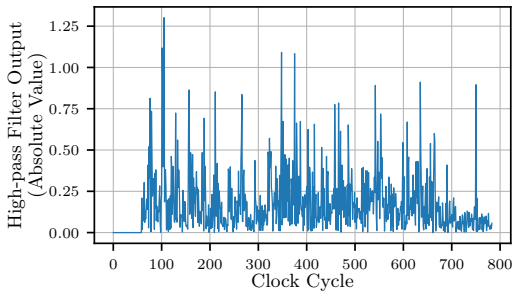
The characterization of convolution unit voltage drops described in the previous subsection was performed using voltage traces obtained by the ChipWhisperer-Lite capture board. In this section, we describe characterization experiments that use voltage measurements obtained by a TDC sensor implemented in the ChipWhisperer FPGA. The TDC architecture was described in Section III-B. The 256-bit TDC carry chain for the Artix-7 FPGA on the ChipWhisperer board consists of $Carry4$ carry primitives. The sensitivity for each TDC stage, as determined by the Xilinx Vivado 2019.1 software [47], is close to 25 ps.

For each clock cycle, the flip-flop values from the TDC were saved in a 256-bit wide FIFO, forming one voltage estimate. This experiment was performed 100 times using the same input image and kernel. The voltage estimates at each clock cycle for the 100 traces were then averaged to minimize noise, forming a collection of 784 Hamming weights, one for each pixel. The resulting Hamming weights are shown in Figure 11a.

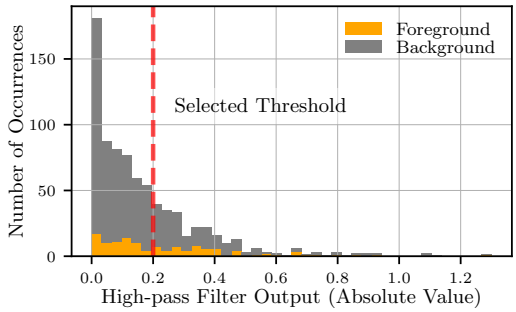
The plot in Figure 11a contains a low-frequency envelope due to the lack of bypass capacitors on the ChipWhisperer that affects supply voltage behavior. A high-pass Butterworth digital filter was applied to the values shown in the plot to remove the envelope and retain voltage fluctuations due to convolution unit activity. For each point in the plot, the filter determines an average Hamming weight value over the previous ten clock cycles (a running average window). This value is then



(a) Average TDC Hamming weights, 100 runs.



(b) Recovered trace after applying a high-pass filter to the TDC Hamming weight values (absolute value).



(c) Histogram of the filtered TDC trace with the selected threshold shown.

Fig. 11: TDC data recovered from ChipWhisperer, (a) Unfiltered trace recovered from TDC, (b) Trace after applying high-pass filter and removing low-frequency envelope (absolute value), (c) Histogram of the filtered TDC trace with the selected threshold.

subtracted from the Hamming weight value at the current clock cycle, leading to the plot shown in Figure 11b. Subsequently, the image was recovered with the histogram threshold shown in Figure 11c and Rudin-Osher-Fatemi denoising steps described earlier in Section V-A. Figure 12c shows the recovered image obtained after applying the denoising algorithm.

C. TDC-Based Attack Summary

To summarize, the following steps are performed to recover a reconstructed image using the on-FPGA TDC:

1) Voltage estimates are collected for each input pixel during operation of the convolution unit for the first kernel of the first convolution layer.
2) Voltage estimates for each pixel are averaged across all runs with the image to generate a single trace. The averaged estimates are represented using Hamming weights.
3) A Butterworth high-pass filter is used to remove low-freqency power supply ripple from the averaged Hamming weights.
4) A histogram of the resulting values is created and a threshold is used to differentiate foreground and background pixels, forming a preliminary recovered image.
5) A Rudin-Osher-Fatemi denoising algorithm is used to improve the quality of the recovered image.

(a) (b) (c)

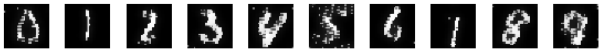
Fig. 12: Recovered image from Chipwhisperer using TDC after applying filter. (a) Input image (same as 8a), (b) recovered image, (c) recovered image after denoising.

(a) Input images.

(b) Recovered images from ZCU104 before denoising.

(c) Recovered images from ZCU104 after denoising.

(d) Recovered images from VCU118 before denoising.



(e) Recovered images from VCU118 after denoising.

(f) Recovered images from AWS F1 before denoising.

(g) Recovered images from AWS F1 after denoising.

Fig. 13: Input images and recovered images before and after denoising from all boards. The images were recovered using only TDC measurements.

VI. Image Extraction Using the Attack

After initial experimentation with the ChipWhisperer CW305, our attack was applied to the two local boards and AWS F1 instances described in Section IV to see how well the attack can perform on commercial off-the-shelf boards that were not designed to study side channel attacks. The hardware for

these platforms was not modified for our experimentation. The experimental setup for these platforms including the BNN accelerator is shown in Figure 7. The clock speeds of the BNN accelerators in the FPGAs are listed in Table II. Our experiments consider the quality of the recovered images, the proximity of the TDC to the convolution unit on the FPGA chip, and the number of times each input image is used to create a recognizable recovered image (e.g., number of runs).

A. Image Recovery with Local Boards

Recovered images, both before and after denoising, for the ZCU104 and VCU118 boards using TDC measurements are shown in Figure 13. For these experiments, the TDC was placed adjacent to the BNN accelerator in the FPGA fabric (in the next row of logic blocks) to increase the accuracy of the voltage estimates. For example, the relative positions of the BNN accelerator and TDC in the ZCU104's UltraScale+ FPGA for these experiments are shown in Figure 14a. The images were recovered after applying the steps outlined in Section V-B. For the ZCU104 and VCU118, the same input image and kernel were used 3,000 times.

The TDC's ability to detect the small voltage drops caused by the convolution unit as it processes the input image is critical to image recovery. To study the importance of TDC location on the FPGA die relative to the location of the BNN accelerator, the BNN was moved to a location on the opposite side of the die, as shown in Figure 14b for the ZCU104's UltraScale+ FPGA. The experiments from Section VI-A were rerun for the digital image shown in Figure 8a.

To compare the quality of the recovered images with cross-die placement of the TDC versus the results from adjacent placement for the selected digit, the normalized cross-correlation (CCR_norm), derived from cross-correlation (CCR), between the recovered images and the input image for both adjacent and cross-die TDC placements were calculated using Equations 3 and 4. Here, $\bar{A}$ and $\bar{B}$ represent the mean pixel values of the images. The CCR_norm value provides a quantitative metric for comparing the similarity of the input image and a recovered image.

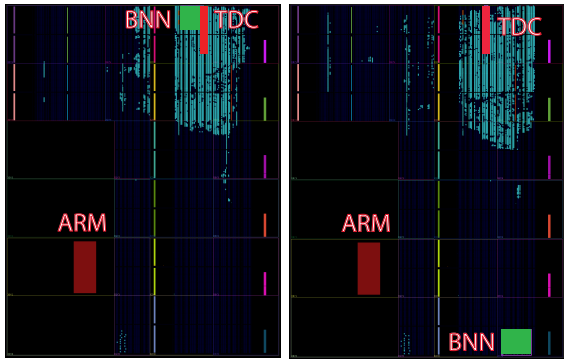
$$CCR = \sum_{(i,j)\in N^{28\times 28}} \left[(A[i,j]-\bar{A})\times(B[i,j]-\bar{B})\right] \quad (3)$$

$$CCR\_norm = \frac{CCR}{\sqrt{\sum\left(A[i,j]-\bar{A}\right)^2 \times \sum\left(B[i,j]-\bar{B}\right)^2}} \quad (4)$$

Recovered images both before and after denoising were considered. Table III shows the normalized cross-correlations of the recovered images on the target boards. Figure 15 shows the recovered images for different placement strategies, both before and after denoising, for the two boards.

This experiment shows that cross-die placement leads to the recovery of a lower-quality image compared to adjacent placement, which was predictable. However, the recovered image is still recognizable and the attack can be performed even if the BNN accelerator and TDC are not in close proximity.

To obtain recognizable reconstructed images, the same input image is processed by the same kernel numerous times. To evaluate the effect of number of runs on image quality, we again used the image shown in Figure 8a. For both local FPGA boards, the normalized cross-correlation (Equation 4) of the recovered image and the original image versus the number of times the input image was processed by the first kernel was calculated. Results from these experiments are shown in Figure 16. Denoising the recovered images clearly improves the image quality. Figure 17 shows recovered images for an increasing number of runs, before and after denoising. This figure clearly shows that after about 200 runs, the recovered image is recognizable.

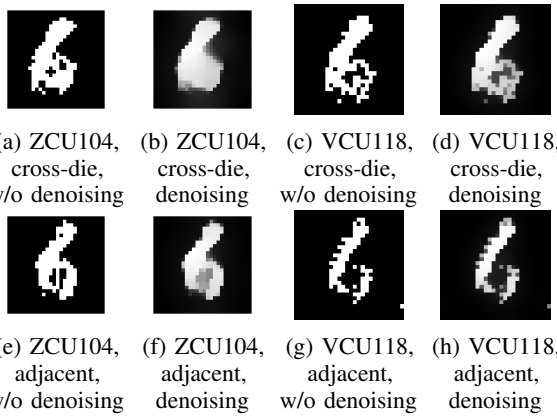


(a) Adjacent Placement (b) Cross-Die Placement

Fig. 14: Floorplan of the ZCU104 UltraScale+ FPGA for adjacent and cross-die placement. Green rectangle: BNN accelerator. Red rectangle: TDC sensor. Brown rectangle: ARM processor.

TABLE III: Normalized cross-correlation between original and recovered images before and after denoising under adjacent and cross-die TDC placement. The ZCU104 and AWS F1 FPGA floorplans are shown in Figures 14 and 20a.

Board	Adjacent Placement		Cross-die Placement	
	w/o denoise	w/ denoise	w/o denoise	w/ denoise
ZCU104	0.745	0.791	0.594	0.655
VCU118	0.678	0.738	0.646	0.697
AWS F1	0.671	0.716	0.426	0.547



(a) ZCU104, cross-die, w/o denoising (b) ZCU104, cross-die, denoising (c) VCU118, cross-die, w/o denoising (d) VCU118, cross-die, denoising

(e) ZCU104, adjacent, w/o denoising (f) ZCU104, adjacent, denoising (g) VCU118, adjacent, w/o denoising (h) VCU118, adjacent, denoising

Fig. 15: Recovered images with adjacent and cross-die placement for 3,000 runs. The input image is shown in 8a.

B. Analysis of Image Mean Structural Similarity

To further contrast the perceptual similarity of recovered and original input images, the mean structural similarity index

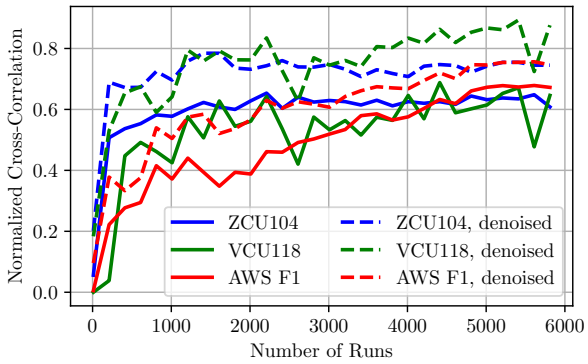


Fig. 16: Normalized cross correlation of the recovered image versus number of runs for all boards, before and after denoising. Input image shown in Figure 8a.

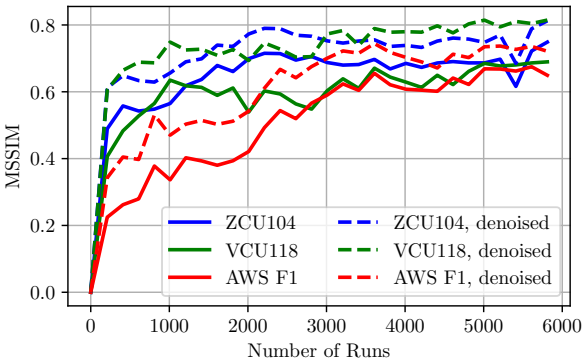
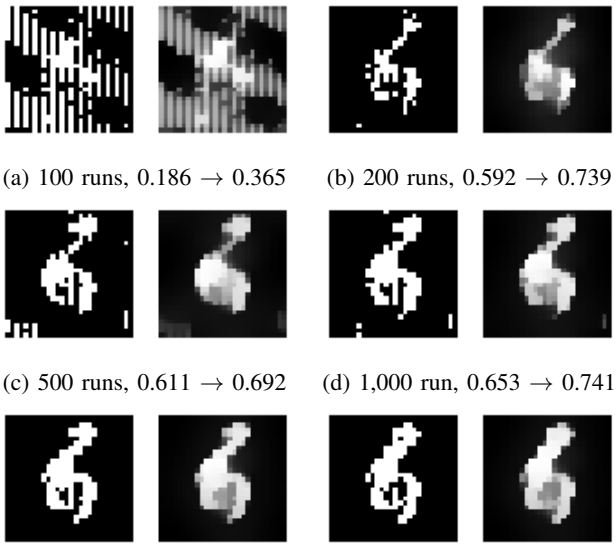


Fig. 18: Mean structural similarity (MSSIM) of the recovered image versus number of runs for all boards, before and after denoising. Input image shown in Figure 8a.

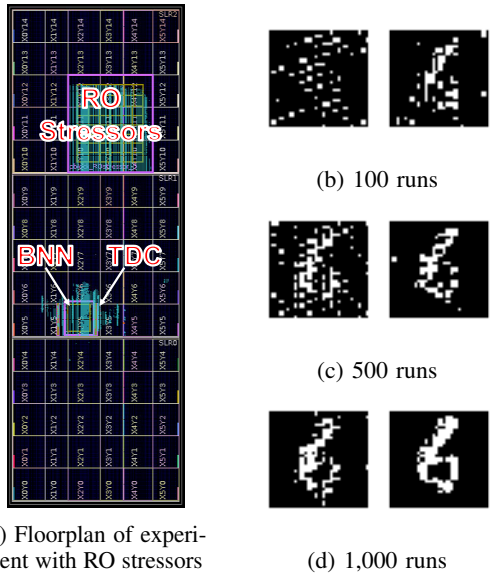


(a) 100 runs, 0.186 → 0.365 (b) 200 runs, 0.592 → 0.739

(c) 500 runs, 0.611 → 0.692 (d) 1,000 run, 0.653 → 0.741

(e) 3,000 runs, 0.744 → 0.785 (f) 6,000 runs, 0.735 → 0.779

Fig. 17: Recovered images for the ZCU104 board for different numbers of runs before (left figure) and after (right figure) denoising. Normalized cross-correlation with the original image is included in each caption (CCR_norm without denoising → CCR_norm with denoising). Input image shown in Figure 8a.



(a) Floorplan of experiment with RO stressors

(b) 100 runs

(c) 500 runs

(d) 1,000 runs

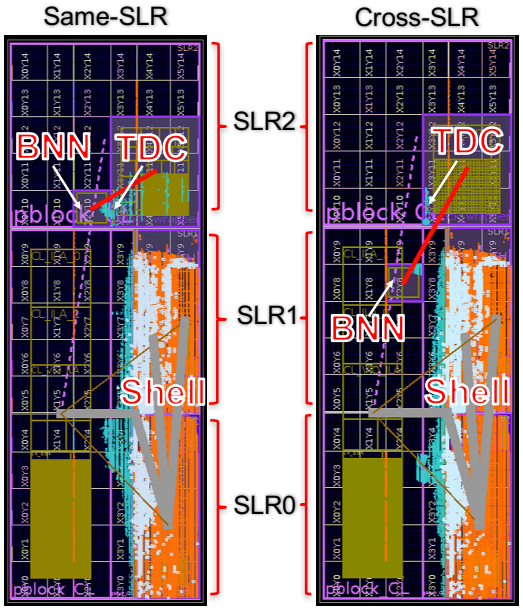
Fig. 19: The effects of instantiated stressor circuits on local VCU118. In (b) - (d), the images recovered without and with 50,000 stressors are shown on the left and right, respectively.

(MSSIM [48]) was calculated. The MSSIM of two images is determined by taking the mean of the structural similarity index values between fixed-size windows of the two images rather than comparing individual pixels. Structural similarity index provides a quantitative comparison between the two image windows. MSSIM calculations for two images generate a value between -1 and 1, with values close to 1 indicating a close match and values close to -1 indicating a complete mismatch. A sliding window size of 11 pixels was chosen for calculating MSSIM values [48]. The mean structural similarity index between the input image in Figure 8a and the recovered image for different numbers of runs is shown in Figure 18. The plots in the figure closely follow the normalized cross correlation trends shown in Figure 16 as the MSSIM index increases when the number of runs increases.

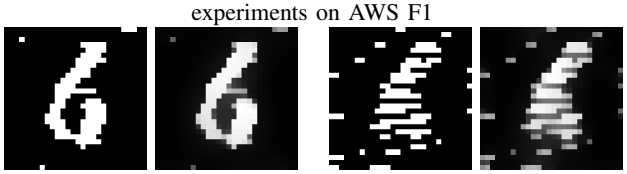
C. Effect of Voltage Stressing Circuits on Local Board Image Recovery

It has previously been shown that an attacker's ability to detect small on-FPGA voltage changes is enhanced if significant steady-state power is simultaneously drawn from the device [49]. In addition to the TDC and associated control circuitry, an attacker may instantiate circuits that deliberately consume significant power in an effort to stress the power distribution network of the supply voltage. A common voltage stressor circuit is a ring oscillator (RO), a combinational loop that contains an odd number of inverters. This type of stressor can be efficiently implemented in an FPGA using one logic element.

In an experiment with the VCU118, RO-based voltage stressors were added to the UltraScale+ FPGA and enabled during the extraction of voltage estimates from the convolution of the input image and first kernel. As shown in the floorplan in Figure 19a, the TDC and BNN accelerator were located in



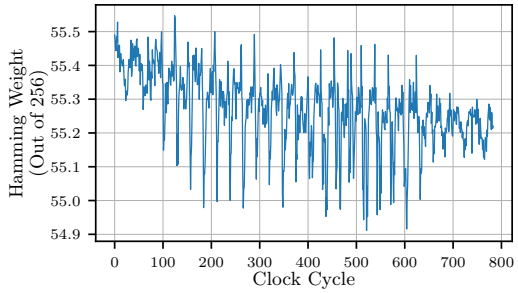
(a) Floorplan of same-SLR and cross-SLR experiments on AWS F1



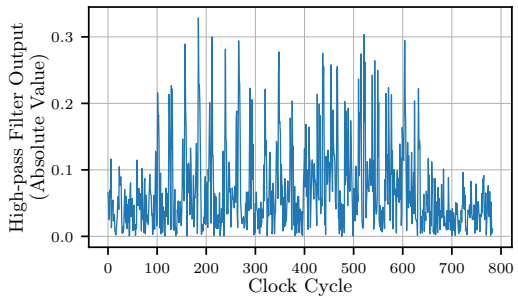
(b) Same-SLR, w/o denoising (left) and w/ denoising (right)

(c) Cross-SLR, w/o denoising (left) and w/ denoising (right)

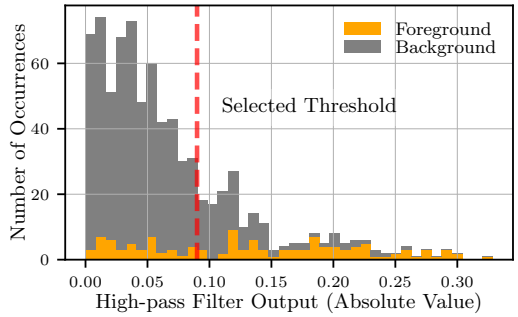
Fig. 20: Floorplan and recovered images of same-SLR and cross-SLR experiments on AWS F1: (a) The *Shell* logic occupies the right part of SLR0 and SLR1. Left: The TDC and BNN accelerator are both on SLR2; Right: The TDC is on SLR2, and the accelerator BNN is on SLR1. (b) The recovered image of same-SLR experiment on AWS F1 for 6,000 runs. (c) The recovered image of cross-SLR experiment on AWS F1 for 6,000 runs.



(a) Average TDC Hamming weights, 6,000 runs.



(b) Recovered trace after applying a high-pass filter to the TDC Hamming weight values (absolute value).



(c) Histogram of the filtered TDC trace with the selected threshold shown.

Fig. 21: Average of TDC voltage drop traces averaged over 6,000 runs on AWS F1 for same-SLR experiment. (a) Unfiltered trace recovered from the TDC; (b) Trace after applying high pass filter (absolute value); (c) Histogram of the filtered TDC trace and the selected threshold.

adjacent columns on the device and the stressors were located in a different region of the device to reduce their effect on on-die temperature. Fifty groups of RO stressors were used, each with 1,000 ROs, for a total of 50,000 (shown in Figure 19a). This stressor count was found to be sufficient to impact the appearance of the recovered images.

To examine effects of using stressors, the image shown in Figure 8a was input into the FPGA for separate experiments in which the stressors were activated or not activated. The recovered images for an increasing number of runs during the experiments are shown at the right in Figure 19. The images indicate the visual improvement as a result of stressor deployment.

D. Image Reconstruction on AWS F1

To show that our attack could be deployed on existing cloud FPGAs if multi-tenancy was allowed, our attack infrastructure was migrated to and tested on AWS F1 instances. The experimental setup for the attack was described in Section IV-C. The UltraScale+ FPGA used in AWS F1 contains three *super logic regions* (SLRs) (Figure 20a). Each SLR is a separate die containing logic and memory resources. As shown in Figure 20a, the *Shell* interface is located in the right-hand area of SLR0 and SLR1. Since the *Shell* has significant power consumption, it can influence the accuracy of TDC measurements. To assess these effects, experiments were performed with the BNN accelerator and TDC on SLR2 (same-SLR) and on separate SLRs (cross-SLR). In the same-SLR experiment (Figure 20a left), the TDC and the BNN are placed next to each other. Figure 20a (right) depicts the cross-SLR experiment, in which the BNN accelerator is on SLR1 and the TDC is on SLR2.

Figure 21 shows the averaged Hamming weights obtained for the same-SLR case using the digit image from Figure 8a for 6,000 runs. As shown in Figure 21a, the averaged values collected by the TDC are influenced by environmental noise,

a decreasing voltage envelope. After high-pass filtering, identifiable peaks, indicating foreground pixels, can be identified, as shown in Figure 21b. The histogram and selected threshold used to extract the recovered image are shown in Figure 21c.

The recovered images of the same-SLR experiment are shown in Figures 13f and 13g. The normalized cross-correlations between the input and recovered images for the same-SLR case before and after denoising are 0.671 and 0.716, respectively. As shown in Figure 16, the normalized cross-correlations for denoised images for the same-SLR experiment on AWS F1 increase as the number of runs are increased.

Similar to the local board experiments, the positioning of the TDC at a distant location from the BNN accelerator results in a reduction in recovered image quality. The experiment described in the previous paragraph was rerun on AWS F1 for the cross-SLR case. The recovered images of digit 6 are shown in Figure 20c, and the averages of normalized cross-correlation are listed in Table III. The results indicate that the normalized cross-correlation of the denoised image for the same-SLR experiment (0.716) is superior to the value for the cross-SLR experiment (0.547). The presence of the $Shell$ in the same SLR as the BNN accelerator for the cross-die experiment influences the quality of the recovered image to a modest extent.

E. Limitations

Although our image reconstruction attack has been shown to be effective on multiple FPGA-based boards, it does have limitations. All presented results thus far were generated using the MNIST handwritten digit database which includes images with background pixels of 0 and foreground pixels with values up to 255. Additionally, our previously reconstructed images have used multiple repetitions (runs) with the exact same image. In this subsection, we examine the performance of the attack on ZCU104 and VCU118 boards if these constraints are relaxed.

As described in Section V, the use of pixel values of 0 for the background minimizes adder tree activity in Figure 5, leading to a significant difference in voltage drops caused by foreground and background pixels. For experimentation, four new versions of the image shown in Figure 8a were created with all background pixels converted to 1, 10, 30, and 50, respectively. Reconstructed images were generated for each modified input image after 6,000 runs each. Table IV indicates that deviation from a zero-valued background does indeed reduce normalized cross-correlation in all cases, although, as shown in Figure 22, the reconstructed images are still recognizable after denoising.

For further experimentation, we created groups of 6,000 distinct images of Figure 8a by flipping the least significant bit (LSB) or two least significant bits of each input pixel with a fixed probability. Effectively, this process mimics analog-to-digital converter noise that may be present during image sampling. Then, we collected TDC measurements from each noisy image once and used the mean of all the 6,000 traces to recover the input image.

Table V shows the normalized cross-correlation of the experiments for six different bit flipping probabilities (0 indicates that no bits were flipped and 100 indicates that all LSB or least significant two bits were flipped). In cases of two-bit flips, both bits of the pixel were flipped from their original values. The results show that bit flipping has a limited effect on normalized cross correlation, although always flipping the LSBs does show some degradation. Sample reconstructed images shown in Figure 23 indicate continued visual recognition.

TABLE IV: Normalized cross-correlation of the recovered image and the original image shown in Figure 8a after replacing the background pixels with a non-zero constant value prior to BNN processing. Results are generated with 6,000 runs.

	Background Pixel Value				
Board	0 (default)	1	10	30	50
ZCU104 (w/o denoising)	0.75	0.46	0.53	0.56	0.56
ZCU104 (w/ denoising)	0.82	0.62	0.65	0.67	0.66
VCU118 (w/o denoising)	0.71	0.59	0.59	0.60	0.58
VCU118 (w/ denoising)	0.76	0.70	0.63	0.64	0.64

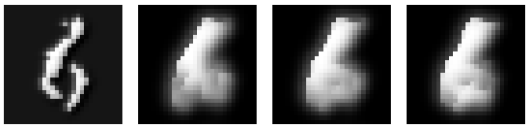


Fig. 22: Recovered images from ZCU104 board when background pixels are replaced with non-zero values, after applying the denoising algorithm. The background values are 0 (default), 1, 30, and 50 from left to right. Each experiment performed with 6,000 runs of the same modified image.

In a final experiment to explore limitations, we evaluated the reconstruction of several images from the Fashion MNIST dataset [50] of black-and-white garment images with the same input image size of 28×28 pixels as the MNIST handwritten digit database. These images of garments have a broader range of textures than digits. As seen in Figure 24, image reconstruction of a sample of input images shows a distinct recognizable garment outline although internal garment textures are missing.

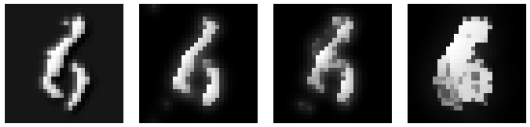
VII. CONCLUSION

This paper presents a remote power side-channel attack on binarized convolutional neural networks targeting multi-tenant FPGAs. We show that it is possible to accurately extract image inputs to a BNN accelerator by collecting and analyzing on-chip voltage estimates. Time-to-digital converters are leveraged to obtain voltage estimates on the FPGA chip during execution of the algorithm. Our approach has been successfully applied to four FPGA boards, including on Xilinx UltraScale+ FPGAs located on Amazon AWS F1 cloud servers. Our experiments successfully recovered recognizable images for all ten digits from the MNIST handwritten digit database.

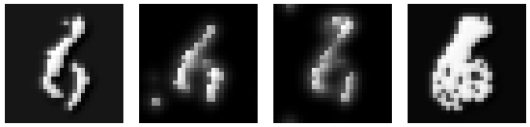
This research opens up significant avenues for future exploration. Additional attacks to extract kernel values are needed to identify both BNN image inputs and parameters. The collection and analysis of voltage estimates for multi-kernel processing in the first convolution layer could be used in this effort. Additional layers of the BNN may also be vulnerable to the extraction of voltage estimates. It may also be possible to use

TABLE V: Normalized cross-correlation of the denoised recovered image and the original image shown in Figure 8a if the least significant bit or least significant two bits of all pixels are flipped prior to input to the BNN accelerator. Results are generated with 6,000 runs.

	Flipping Probability					
Board (Bit Number)	0	20	40	60	80	100
ZCU104 (LSB)	0.83	0.77	0.80	0.75	0.76	0.63
VCU118 (LSB)	0.76	0.76	0.73	0.67	0.67	0.64
ZCU104 (2 Bits)	0.83	0.76	0.76	0.73	0.74	0.62
VCU118 (2 Bits)	0.76	0.75	0.70	0.63	0.65	0.63



(a) Flip LSB, Probability from left to right = 0, 40, 80, 100%



(b) Flip lower 2 bits, Probability from left to right = 0, 40, 80, 100%

Fig. 23: Recovered images from the ZCU104 board for 6,000 runs: (a) denoised recovered images after flipping the least significant bit of pixels in the BNN input image with fixed probabilities, (b) denoised recovered images for probabilistic flips of the bottom two pixel bits.

a similar approach to extract input images for CNNs with non-binary kernel values. Such an approach would require the use of multipliers for weight scaling, possibly leading to increased power consumption. More complex datasets, including color images, and applications, such as face recognition, could also be considered. More research is also needed to determine exactly when BNN processing starts so that TDC sampling can be synchronized with BNN processing.

Countermeasures are also needed to reduce the effectiveness of on-chip voltage measurement attacks. The extraction of voltage estimates could be impeded by the significant circuit switching of interfaces or other design components in the proximity of the convolution unit (e.g., active fences [51]). Additionally, the pixel order of convolution unit processing could be scrambled on a per-image basis to make image reconstruction more difficult.

REFERENCES

[1] Amazon Web Services, "Amazon EC2 F1 instances," https://aws.amazon.com/ec2/instance-types/f1/, Accessed: 2020-8-23.

[2] Alibaba Cloud, "Elastic compute service: Instance type families," https://www.alibabacloud.com/help/doc-detail/25378.htm#f1, Accessed: 2020-03-29.

[3] Xilinx, Inc., "Xilinx powers Huawei FPGA accelerated cloud server," https://www.xilinx.com/news/press/2017/xilinx-powers-huawei-fpga-accelerated-cloud-server.html, Accessed: 2020-03-29.

[4] Baidu Cloud, "FPGA cloud compute," https://cloud.baidu.com/product/fpga.html, Accessed: 2020-03-29.

[5] Tencent Cloud, "FPGA cloud computing," https://cloud.tencent.com/product/fpga, Accessed: 2020-03-29.

(a) Input Image, left to right: coat, dress, pullover, T-shirt

(b) Recovered images using the ZCU104 board

(c) Recovered images using the VCU118 board

Fig. 24: Recovered images for coat, dress, pullover, and T-shirt images from the Fashion-MNIST dataset using the ZCU104 and VCU118 boards for 6,000 runs.

[6] E. El-Araby, I. Gonzalez, and T. El-Ghazawi, "Virtualizing and sharing reconfigurable resources in high-performance reconfigurable computing systems," in *International Workshop on High-Performance Reconfigurable Computing Technology and Applications (HPRCTA)*, 2008.

[7] S. Byma, J. G. Steffan, H. Bannazadeh, A. L. Garcia, and P. Chow, "FPGAs in the cloud: Booting virtualized hardware accelerators with OpenStack," in *IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2014.

[8] F. Chen, Y. Shan, Y. Zhang, Y. Wang, H. Franke, X. Chang, and K. Wang, "Enabling FPGAs in the cloud," in *ACM Conference on Computing Frontiers (CF)*, 2014.

[9] J. Weerasinghe, F. Abel, C. Hagleitner, and A. Herkersdorf, "Enabling FPGAs in hyperscale data centers," in *IEEE International Conference on Ubiquitous Intelligence and Computing, Autonomic and Trusted Computing, Scalable Computing and Communications (UIC-ATC-ScalCom)*, 2015.

[10] A. Khawaja, J. Landgraf, R. Prakash, M. Wei, E. Schkufza, and C. J. Rossbach, "Sharing, protection, and compatibility for reconfigurable fabric with AMORPHOS," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.

[11] A. Vaishnav, K. D. Pham, and D. Koch, "A survey on FPGA virtualization," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2018.

[12] I. Giechaskiel, K. B. Rasmussen, and K. Eguro, "Leaky wires: Information leakage and covert communication between FPGA long wires," in *ACM ASIA Conference on Computer and Communications Security (ASIACCS)*, 2018.

[13] I. Giechaskiel, K. Eguro, and K. B. Rasmussen, "Leakier wires: Exploiting FPGA long wires for covert- and side-channel attacks," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 12, no. 3, pp. 1–29, Sep. 2019.

[14] I. Giechaskiel, K. B. Rasmussen, and J. Szefer, "Measuring long wire leakage with ring oscillators in cloud FPGAs," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2019.

[15] C. Ramesh, S. B. Patil, S. N. Dhanuskodi, G. Provelengios, S. Pillement, D. Holcomb, and R. Tessier, "FPGA side channel attacks without physical access," in *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2018.

[16] G. Provelengios, C. Ramesh, S. B. Patil, K. Eguro, R. Tessier, and D. Holcomb, "Characterization of long wire data leakage in deep submicron FPGAs," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2019.

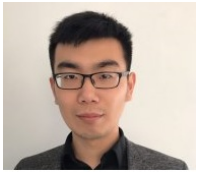
[17] O. Glamocanin, L. Coulon, F. Regazzoni, and M. Stojilović, "Are cloud FPGAs really vulnerable to power analysis attacks?" in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020.

[18] M. Zhao and G. E. Suh, "FPGA-based remote power side-channel attacks," in *2018 IEEE Symposium on Security and Privacy (S&P)*, 2018.

[19] F. Schellenberg, D. R. Gnad, A. Moradi, and M. B. Tahoori, "An inside

job: Remote power analysis attacks on FPGAs," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018.
[20] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta, and Z. Zhang, "Accelerating binarized convolutional neural networks with software-programmable FPGAs," in *ACM/SIGDA International Symposium on FPGAs (FPGA)*, 2017.
[21] Y. Chen, J. He, X. Zhang, C. Hao, and D. Chen, "Cloud-DNN: An open framework for mapping DNN models to cloud FPGAs," in *ACM/SIGDA International Symposium on FPGAs (FPGA)*, 2019.
[22] S. Zeng, G. Dai, H. Sun, K. Zhong, G. Ge, K. Guo, Y. Wang, and H. Yang, "Enabling efficient and flexible FPGA virtualization for deep learning in the cloud," in *IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2020.
[23] S. Moini, B. Alizadeh, M. Emad, and R. Ebrahimpour, "A resource-limited hardware accelerator for convolutional neural networks in embedded vision applications," *IEEE Transactions on Circuits and Systems (TCAS) II: Express Briefs*, vol. 64, no. 10, pp. 1217–1221, 2017.
[24] Microsoft Azure, "What are Field-Programmable Gate Arrays (FPGA) and how to deploy," https://docs.microsoft.com/en-us/azure/machine-learning/how-to-deploy-fpga-web-service, Accessed: 2020-05-14.
[25] L. Wei, B. Luo, Y. Li, Y. Liu, and Q. Xu, "I know what you see: Power side-channel attack on convolutional neural network accelerators," in *ACM Computer Security Applications Conference*, 2018, pp. 393–406.
[26] Y. LeCun, C. Cortes, and C. Burges, "MNIST handwritten digit database." http://yann.lecun.com/exdb/mnist/, Accessed: 2020-05-19.
[27] C. O'Flynn and Z. D. Chen, "ChipWhisperer: An open-source platform for hardware embedded security research," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*, 2014.
[28] Xilinx, Inc., "ZCU104 evaluation board," https://www.xilinx.com/products/boards-and-kits/zcu104.html, 2020, Accessed: 2020-05-19.
[29] ——, "VCU118 evaluation board," https://www.xilinx.com/products/boards-and-kits/vcu118.html, 2020, Accessed: 2020-05-19.
[30] Amazon.com, Inc., "Amazon AWS F1," https://aws.amazon.com/ec2/instance-types/f1, 2020, Accessed: 2020-05-19.
[31] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295 – 2329, Dec. 2017.
[32] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in Neural Information Processing Systems*, 2016.
[33] K. Ding, "Binarized dense and Conv2D layers for Keras," https://github.com/DingKe/BinaryNet, 2020, Accessed: 2020-05-19.
[34] A. Dubey, R. Cammarota, and A. Aysu, "Maskednet: A pathway for secure inference against power side-channel attacks," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2020.
[35] W. Hua, Z. Zhang, and G. E. Suh, "Reverse engineering convolutional neural networks through side-channel information leaks," in *ACM/IEEE Design Automation Conference (DAC)*, 2018.
[36] K. Yoshida, T. Kubota, M. Shiozaki, and T. Fujino, "Model-extraction attack against FPGA-DNN accelerator utilizing correlation electromagnetic analysis," in *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019.
[37] Satoh Lab., "SAKURA-G FPGA board," http://satoh.cs.uec.ac.jp/SAKURA/hardware/SAKURA-G.html, 2020, Accessed: 2020-05-19.
[38] ——, "SAKURA-X FPGA board," http://satoh.cs.uec.ac.jp/SAKURA/hardware/SAKURA-X.html, 2016, Accessed: 2020-05-19.
[39] A. Boutros, M. Hall, N. Papernot, and V. Betz, "Neighbors from Hell: Voltage attacks against deep learning accelerators on multi-tenant FPGAs," in *International Conf. on Field Programmable Technology (FPT)*, 2020.
[40] D. R. E. Gnad, C. D. K. Nguyen, S. H. Gillani, and M. B. Tahoori, "Voltage-based covert channels in multi-tenant FPGAs," Cryptology ePrint Archive, Report 2019/1394, 2019, https://eprint.iacr.org/2019/1394.
[41] K. M. Zick, M. Srivastav, W. Zhang, and M. French, "Sensing nanosecond-scale voltage attacks and natural transients in FPGAs," in *ACM/SIGDA International Symp. on Field Programmable Gate Arrays (FPGA)*, 2013.
[42] S. Moini, S. Tian, D. Holcomb, J. Szefer, and R. Tessier, "Remote power side-channel attacks on BNN accelerators in FPGAs," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021.
[43] NewAE Technology, Inc., "CW305 ChipWhisperer Artix FPGA target board," https://wiki.newae.com/CW305_Artix_FPGA_Target, 2020, Accessed: 2020-05-22.
[44] ——, "CW1173 ChipWhisperer-Lite capture board," https://wiki.newae.com/CW1173_ChipWhisperer-Lite, 2020, Accessed: 2020-05-22.
[45] L. I. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Physica D: Nonlinear Phenomena*, vol. 60, no. 1-4, pp. 259–268, 1992.
[46] W. Zhu, "A first-order image denoising model for staircase reduction," *Advances in Computational Mathematics*, vol. 45, Nov. 2019.
[47] Xilinx, Inc., "Ug973: Vivado design suite user guide," https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug973-vivado-release-notes-install-license.pdf, June 2019.
[48] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.
[49] I. Giechaskiel, K. Rasmussen, and J. Szefer, "Capsule: Cross-FPGA covert-channel attacks through power supply unit leakage," in *IEEE Symposium on Security and Privacy (S&P)*, 2020.
[50] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.
[51] J. Krautter, D. R. Gnad, F. Schellenberg, A. Moradi, and M. B. Tahoori, "Active fences against voltage-based side channels in multi-tenant FPGAs," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.

Shayan Moini (S16) received his B.Sc. degree In Electrical Engineering at Sharif University of Technology, Tehran, Iran, in 2014. In 2017, he completed his M.Sc. degree in Electrical Engineering at the University of Tehran, Iran. He is currently studying towards a Ph.D. degree in the Department of Electrical and Computer Engineering at the University of Massachusetts, Amherst, MA, USA.



Shanquan Tian (S18) received his B.S. degree in Applied Physics from University of Science and Technology of China, Anhui, China in 2017, and the M.S. degree in Electrical Engineering from Yale University, New Haven, CT, USA in 2019, where he is currently studying towards a Ph.D. degree in Computer Architecture and Security Laboratory, Department of Electrical Engineering at Yale.

Daniel Holcomb (M07) received the B.S. and M.S. degrees in electrical and computer engineering from the University of Massachusetts Amherst, Amherst, MA, USA, and the Ph.D. degree in electrical engineering and computer sciences from the University of California at Berkeley, Berkeley, CA, USA. He is currently an Associate Professor with the Department of Electrical and Computer Engineering, University of Massachusetts Amherst.

Jakub Szefer (S08–M13–SM19) received B.S. with highest honors in Electrical and Computer Engineering from University of Illinois at Urbana-Champaign, and M.A. and Ph.D. degrees in Electrical Engineering where his research focused on secure hardware architectures. He is currently an Associate Professor of Electrical Engineering at Yale University where he leads the Computer Architecture and Security Laboratory (CASLAB).

Russell Tessier (M00–SM07) received the B.S. degree in computer and systems engineering from Rensselaer Polytechnic Institute, Troy, NY, USA and the S.M. and Ph.D. degrees in electrical engineering from the Massachusetts Institute of Technology, Cambridge, MA, USA. He is currently Professor of Electrical and Computer Engineering with the University of Massachusetts, Amherst, MA.