Bake It Till You Make It

Heat-induced Leakage from Masked Neural Networks

Dev M. Mehta*[1], Mohammad Hashemi*[1], David S. Koblah[2], Domenic Forte[2] and Fatemeh Ganji[1]

[1] Worcester Polytechnic Institute, Worcester, USA,
dmmehta2@wpi.edu,mhashemi@wpi.edu,fganji@wpi.edu
[2] University of Florida, Gainesville, USA, dkoblah@ufl.edu,dforte@ece.ufl.edu

Abstract. Masking has become one of the most effective approaches for securing hardware designs against side-channel attacks. Irrespective of the effort put into correctly implementing masking schemes on a field programmable gate array (FPGA), leakage can be unexpectedly observed. This is due to the fact that the assumption underlying all masked designs, i.e., the leakages of different shares are independent of each other, may no longer hold in practice. In this regard, extreme temperatures have been shown to be an important factor in inducing leakage, even in correctly-masked designs. This has previously been verified using an external heat generator (i.e., a climate chamber). In this paper, we examine whether the leakage can be induced using the circuit components themselves. Specifically, we target masked neural networks (NNs) in FPGAs, with one of the main building blocks being block random access memory (BRAM) and flip-flops (FFs). In this respect, thanks to the inherent characteristics of NNs, our novel internal heat generators leverage solely the memories devoted to storing the user's input, especially when frequently writing alternating patterns into BRAMs and FFs. The possibility of observing first-order leakage is evaluated by considering one of the most recent and successful first-order secure masked NNs, namely ModuloNET. ModuloNET is specifically designed for FPGAs, where BRAMs are used for storing the inputs and intermediate computations. Our experimental results demonstrate that undesirable first-order leakage can be observed by increasing the temperature when an alternating input is applied to the masked NN. To give a better understanding of the impact of extreme heat, we further perform a similar test on the design with FFs storing the input, where the same conclusion can be drawn.

Keywords: Side-channel Analysis · Masking · Neural Networks · Heat Generation · T-test · FPGA.

1 Introduction

Deep learning (DL) accelerators have become an integral part of Internet of things (IoT) edge devices that support image classification, speech recognition, etc. [LeC19]. In this regard, mobile and wearable devices require a low-cost neural network (NN) accelerator to support DL inference in cameras, medical devices, ground maintenance systems, video games, and so on. In these applications, a NN is handed to the users, whose preparation requires a significant investment of money and time in order to train it against a (relatively) huge training dataset and tune its hyperparameters and parameters. Hence, a malicious user may attempt to extract the NN's parameters and hyperparameters (two valuable assets of the accelerator's designer) [BBJP19].

* These authors contributed equally to this work.

Standard protections, e.g., blocking JTAG access, blocking binary readback, code obfuscation, etc., could be taken into account against reverse-engineering and binary analysis attacks [BBJP19]. However, these protections cannot prevent an attacker from extracting the assets of NNs through side-channel analysis (SCA) [BBJP19, DCA20b, XCC+20, YKO+20]. Masking schemes are among the most widely studied countermeasures to protect cryptographic primitives against SCA and have been one of the first solutions discussed in the context of protecting accelerators [DCA20b, DCA20a, DAP+22]. Masking schemes have been proven to be secure against SCA, even with higher orders, although they impose a high price of overhead on the design, need careful construction and implementation, and may suffer from high latency as well as the fresh randomness requirement. Interestingly, recent masked NNs have overcome the challenge facing them with regard to the masking overhead and have proven that their approach is resilient against SCA in the order of million traces [DAP+22, DCA20b]. Nevertheless, despite the effort put into designing a masked scheme –irrespective of the underlying function of the scheme– unexpected leakage can be exhibited when realizing the masked design [DCEM18]. This failure has been mainly attributed to the fact that leakages of different shares are no longer independent under some specific conditions, e.g., high temperature, high clock frequency, etc. The work in [DCEM18] can be thought of as precise experimentation and *leakage detection* within the lab environment to give a better understanding of what has been reported before in a series of work [GOKT16, SGMT18, ZS18, RPD+18]. These studies have been devoted to (1) the impact of power consumption of circuitry placed in one region of a field programmable gate array (FPGA) on the fluctuations of the power supply voltage at other, even unrelated/unconnected, regions of the FPGA [GOKT16, DCEM18]; and (2) how this effect can be exploited to conduct SCA [SGMT18, ZS18, RPD+18]. While the latter has been extensively researched, the former topic, in particular, the conditions resulting in an unexpected leakage, is yet to be sufficiently investigated.

Specifically, although the impact of high temperature on the dependency of the shares has been considered in [DCEM18] by using a climate chamber, it is interesting to explore how a masked design can be intentionally exposed to such extreme heat without using an external generator, making even remote heat-induced SCA possible. When considering active attacks, i.e., fault attacks, the devastating effect of high temperature has been widely known and studied, see, e.g., [PBR17, MLS22, BBB+22, BH22]. On the other hand, little attention has been paid to how increasing the temperature could lead to leakage, and even in those relevant studies [GOKT16, DCEM18] as discussed above, an external heat generator has been employed. This is despite the fact that internal heat generators could be applicable as well. Such generators have been analyzed and developed by Happe et al. [HHAP12], and Agne et al. [AHH+14], who have demonstrated that utilizing and frequently reading/writing from/to some circuit components such as block random access memory (BRAM) and flip-flop (FF) *pipelines* cause a significant rise in circuit temperature. Therefore, a natural question to ask would be whether such heat generators *already exist and can be exploited* in masked designs to cause leakage. Our paper attempts to answer this question by considering FPGA-based accelerators, which inevitably involve BRAMs and FFs in their design as their main building blocks. Additionally, as explained before, masking has been further introduced to these accelerators, making them a viable option for our study.

Contributions: Our contributions are summarized below.

- As our first step, we focus on the design of ModuloNET as presented in [DAP+22]. Although the design has not been made available, we follow the precisely described design of modules involved in ModuloNET. Our design is further verified using one of the state-of-the-art tools, namely VERICA [RBFSG22]. During the verification phase, as a byproduct, we identify a vulnerability in the hardware implementation of Goubin's binary to arithmetic (B2A) conversion algorithm [Gou01] as used in ModuloNET, which has been overseen in the literature. We report that writing/reading sensitive variables into/from memory can cause a leakage, which we address by slightly changing the design of ModuloNET.[1] We emphasize that this detected vulnerability complements the set of issues with such conversion algorithms recently identified in [GPM22]. Using the t-test also helps us to demonstrate that no first-order leakage exists in our ModuloNET design after resolving the issue.

- Our second contribution is the design of the first internal heat generator, which relies on neither additional circuitry nor an external heat generator, but solely the design –precisely the memory used to store the inputs– and the inputs crafted by the user. Compared to the ones proposed in [AHH+14, HHAP12], our generators *do not* leverage BRAM and FF pipelines, but rather, rely on writing into the memory in a parallel manner. In doing so, no changes are made in the design of FPGA-based masked accelerators under attack. To assess its efficacy, we take advantage of the BRAMs used in ModuloNET to store the inputs. The extreme temperature is observed by simply writing alternating '1' and '0' patterns into *single-port* BRAMs (see Section 5.1 for more details). Under this condition, the leakage of the design is successfully changed, and at some points in time, the t-scores do not always remain within the desired threshold.

- Last but not least, we examine if first-order leakage can be observed if FFs, instead of BRAMs, are used to store the inputs. Our results demonstrate that writing the inputs into FFs also leads to an increase in the temperature, and consequently, first-order leakage is detectable.

Organization. The paper is organized as follows. Next, we give an overview of the most relevant work in Section 2. This is followed by the background information and adversary model provided in Section 3. Section 4 introduces the advantages and challenges inherent to FPGA-based accelerators. The design of ModuloNET as presented in [DAP+22] is also introduced in Section 4, whereas our methodology for generating heat is presented in Section 5. Our results are provided in Section 6 with a discussion in Section 7, after which Section 8 concludes the paper.

2 Related Work

2.1 Attacks against NNs

SCA and fault attacks. FPGAs are extensively used to implement DL accelerators and are supported by cloud providers; however, a major security concern about them has been side-channel analysis (SCA) and fault attacks [XAQ21] [BH22]. SCA against NNs has been successful, and therefore, many protected NN designs have emerged [DAP+22, DCA20a, DCA20b]. Some of the recent work relevant to SCA against NN include [XCC+20, YMY+20, BBJP19, DCA20b, YKO+20, XAQ21, BJP22, SGMT18, ZS18, DKAA22] – just to name a few. In this series of work, regardless of whether the attacker has physical or remote access to the device, her goal is to extract the NN model and/or parameters, being the NN's assets [TG22]. This has been achieved through observing physical leakages such as timing [BBJP19], power consumption [XCC+20], and electromagnetic emanation (EM) [BBJP19]. Another category of SCA conducted against NNs is fault-induced SCA. As an example of this, the authors of [LGFX21] have used SCA and power-wasting circuits in conjunction with each other for their attack. The power-wasting circuit used is a look-up tables (LUT) based combinatorial loop to bypass combinatorial loop checkers used by cloud

[1]We believe that this could have been observed and resolved by the authors of [DAP+22] since no leakage has been reported in their paper, although they could not report the vulnerability since no verification tool was applied in their study.

providers that prevent similar attacks based on ring oscillators (ROs). The advantage of this approach is that there is no need to have a priori information about the NN engine under attack due to the SCA module included to schedule the attack.

SCA, as a passive attack, is not the only type of attack mounted against NNs. Voltage and clock-based faults are widely used for fault injection. This type of tampering can cause bit-flips [KGT22], timing faults [MS19] and can even reset the design [KHEB14]. It can also lead to indirect or direct physical effects on the hardware like temperature change, timing delays, etc. [KHEB14].

NN reverse engineering attacks. Researchers have developed an attack using the ROs to steal NN parameters remotely [ZYC$^+$21]. They take advantage of the shared power resources present in a cloud FPGA setup. ROs are used as sensors to measure the power consumption of different NN operations carried out by the victim design. They train a machine learning algorithm with the implementation of different kinds of NNs. This model is used to infer the parameters of the victim circuit using the power traces. Similar results were obtained by Tian et al. using a time-to-digital converter (TDC) based SCA to reverse engineer the structure of NN remotely on the FPGA implementation of the Versatile Tensor Accelerator [TMW$^+$21]. They obtained parameters for a multi-tenant implementation of ResNet-18 and MobileNet with different layers configuration. Similarly, Moini et al. show the use of TDC in multi-tenant FPGA setup to extract image inputs for a binary NN accelerator [MTH$^+$21]. They have shown the results for multiple FPGAs, including Ultrascale+ FPGA from Amazon AWS F1 cloud server.

2.2 Temperature-based Attacks

Here we briefly explain how changes in the temperature have been leveraged by attackers targeting various designs. Faults can be injected by external temperature manipulation, as shown by Hutter et al. [HS13]. They have shown fault generation in RSA by heating the microcontroller outside its recommended temperature tolerance. Another advantage of heating is that glitches can be easily induced when the device is at a higher temperature [KHEB14]. This can increase the efficacy of the faults. External temperature-based attacks have also been mounted to inject faults in memory [GA03, Sko09]. These attacks can be translated to accelerators as they use similar hardware capabilities of FPGAs. As another example, Korak et al. showed a clock-glitching fault attack with artificial temperature control on a micro-controller platform using an external controller [KHEB14]. It results in changes in instructions, execution order, and value changes.

When it comes to temperature-based attacks against NNs, [ATG$^+$19] can serve as an example. Writing into dual-port BRAM has been used to inject faults in NNs [ATG$^+$19]. It has been shown that successive write-collisions lead to high voltage consumption and an increase in temperature. Using this technique, faults were injected successfully into a neural network implementation. The heating of the chip combined with voltage drop leads to bit-flips and timing violations.

Side-channel leakage dependency on temperature Hardware masking schemes' side-channel leakages are affected by various factors such as the supply voltage, frequency, and temperature [MRSS18]. To support this claim, Moradi et al. [MRSS18] investigated the impact of such factors and demonstrated that even with a correct implementation of a masking scheme, such implementations still exhibit unexpected leakage. They performed a wide range of experiments, targeting FPGA, and reported under what circumstances, a correct implementation of masked hardware shows unexpected leakage. They studied the effect of six factors, including number of shares, shunt resistor, voltage supply, circuit size, design frequency, and temperature on side-channel leakage, and showed that under specific circumstances (i.e., at a certain frequency and temperature), a well-designed hardware masking implementation exhibits leakage sooner than expected.

Summary. There have been multiple efforts to show the weaknesses of different implementations of NNs. On the other hand, temperature-based attacks have also been discussed in various contexts, including the security of NNs, although fault attacks can be seen as the main category in this matter. To the best of our knowledge, no study has considered circuit components-based heat generation to induce leakage, let alone in the case of a masked implementation of a NN.

3 Background

3.1 Brief Introduction to Masking Schemes

A masking scheme can be seen as a secret sharing one, where secret values are split into *shares*, and operations are conducted on them in such a manner that a specific security objective is achieved. In this regard, a masking scheme aims to offer security at a given order d by making a set of assumptions on the leakage behavior of the target device.

Boolean masking. One of the most common forms of masking is Boolean masking, where binary addition is adopted to share the sensitive variables. In doing so, a sensitive value $x \in GF(2^m)$ is divided to $d+1$ shares $(x_1, \cdots, x_{d+1})$ such that $x = \bigoplus_{i=1}^{d+1} x_i$. The security requirement for this scheme is the uniformity of the shares, guaranteed by drawing shares $x_1, \cdots, x_d$ from a uniform random distribution and by choosing $x_{d+1} = x \oplus \bigoplus_{i=1}^{d} x_i$ (so-called correctness property). It is straightforward to see that linear and affine functions can be securely evaluated when applying Boolean masking cf. [DCEM18]. On the other hand, non-linear functions need further attention. This becomes more evident if we take masked multiplication as an example into account. In that case, when computing $z = xy$, a total of $(d+1)^2$ terms contribute to the output, which should be reduced to $(d+1)$ shares. For this purpose, various Boolean masking schemes have been proposed in the literature, although we focus solely on Domain-Oriented Masking [GMK16] used in the design of masked NNs [DAP+22].

Domain-Oriented Masking (DOM). To reduce the number of shares contributing to the multiplication output, DOM involves two steps. First, the cross-products are computed, and randomness is added to specific ones; for instance, for the first-order security, we obtain cf. [DCEM18]:

$$\begin{aligned} p_1 &= x_1 y_1 \\ p_2 &= x_1 y_2 \oplus r_1 \\ p_3 &= x_2 y_1 \oplus r_1 \\ p_4 &= x_2 y_2 \end{aligned} \tag{1}$$

After that, in the second phase, the terms p_i in the Equation (1) are synchronized in a register and introduced to a compression stage to reduce the $(d+1)^2$ shares to $(d+1)$ shares in the output z_i, where $z_1 = p_1 \oplus p_2$ and $z_2 = p_3 \oplus p_4$. This reduction is obtained at the cost of an extra clock cycle and the independence requirement imposed on the input shares [GMK16, GMK17, GM17]. Nevertheless, the DOM multiplier is one of the commonly applied masking schemes whose security comes down to the independent power consumption of the component functions.

Arithmetic masking, and conversion to Boolean masking In practice, arithmetic operations may be needed for different cryptographic (e.g., SPECK [BSS+13]) and non-cryptographic primitives (e.g., NNs) [Cor17, DMRB18, DAP+22]. Under this scenario, it

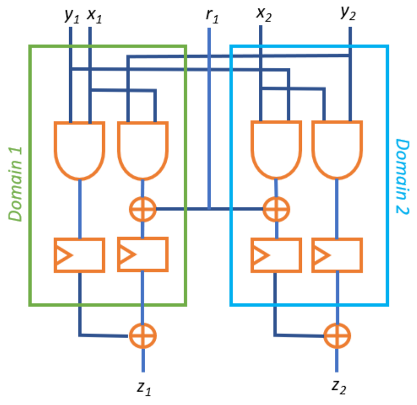


Figure 1: Domain-Oriented Masking (DOM) Multiplier for first-order secure computation. Here DOM-indep multiplier is illustrated, where the demand for fresh randomness and the area overhead in terms of gate count is significantly smaller than the DOM-dep multiplier; however, this is achieved by independently sharing the inputs.

can be advantageous to employ arithmetic masking. As a simple example, to compute $z = x + y \bmod 2^k$, a first-order scheme with arithmetic sharing performs the following operations. First, arithmetic shares A_1, A_2, B_1, B_2 are defined such that $x = A_1 + A_2$ and $y = B_1 + B_2$. Afterward, the shares are added separately, by letting $C_1 \leftarrow A_1 + B_1$ and $C_2 \leftarrow A_2 + B_2$ with two arithmetic shares C_1 and C_2 that can be directly added to obtain $z = x + y = A_1 + A_2 + B_1 + B_2 = C_1 + C_2$. Note that all additions and subtractions are performed modulo 2^k cf. [Cor17].

Under conditions where both Boolean and arithmetic masking schemes are needed, it is possible to convert one to the other. Specifically, a B2A conversion accepts Boolean shares and outputs arithmetic ones: Boolean shares of x that are $x_1, x_2, \cdots x_{d+1}$ should be converted to $(d+1)$ arithmetic shares a_i such that $x = a_1 + a_2 + \cdots + a_{d+1} \bmod 2^k$ without leaking any information about x. The first B2A algorithms have been introduced by Goubin [Gou01], with first-order security only (for higher-order conversion, see, e.g., [CGV14, Cor17, HT19]). The algorithm working the other way around and converting arithmetic to Boolean (A2B) has also been provided in [Gou01] and further improved in terms of complexity in [CGTV15]. As discussed in [DAP+22], conversion algorithms devised to be implemented in hardware have assumed the same field for inputs and outputs and attempted to reuse the randomness in the Boolean shares [MTMM07, Gol07]; nevertheless, [DAP+22] had to take another approach, namely concatenating each Boolean share (i.e., 1 bit) with $k-1$ fresh random bits to obtain k-bit Boolean shares. After that, the approach in [Gol07] has directly been applied as a B2A algorithm.

Leakage evaluation through t-test Test vector leakage assessment (TVLA) has been a standard test methodology used in the literature to detect side-channel leakage [DAP+22, DCA20a, DCA20b, DCEM18]. Relying on Welch's t-test, the TVLA test checks the similarity between two sets of traces captured from two populations of inputs. Holding these two sets of traces, the t-test calculates the t-score as $t = (\mu_1 - \mu_2)/\sqrt{(s_1^2/n_1^2) + (s_2^2/n_2^2)}$, where μ_1 and μ_2 are the means, s_1 and s_2 are the standard deviations, and n_1 and n_2 are the total number of the captured traces for first and second population, respectively. Based on the null hypothesis, the t-scores from two trace sets indicate if the design is susceptible to an SCA if it exceeds the threshold ± 4.5. If so, the design is vulnerable to an SCA with 99.99% confidence. Similar to the prior masked approaches, to evaluate the side-channel resiliency of our design, we choose the non-specific fixed vs. random

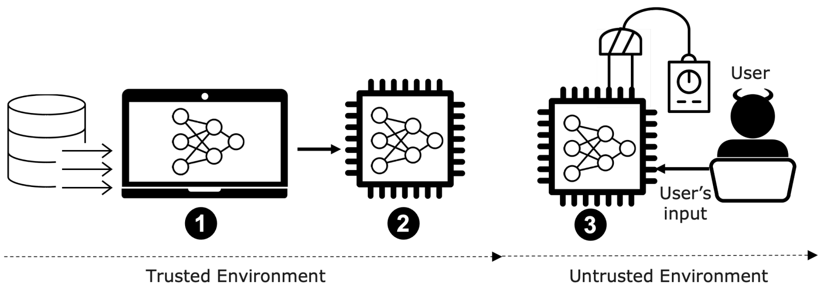


Figure 2: Adversary model considered in this work is similar to prior studies on SCA against NNs. In the trusted environment, first, the NN is trained on a given dataset. Second, the trained NN is implemented on FPGA. Third, in an untrusted environment, the user of the device acts maliciously and attempts to extract information about the NN's parameters and architecture by launching SCA.

t-test [DCEM18, DAP$^+$22].

3.2 VERICA

VERICA [RBFSG22] is a tool for the formal verification of hardware. Most verification tools focus on either SCA or fault attacks (FA) but VERICA has incorporated verification of a design under both attacks. It mainly uses the probing model introduced in [ISW03] to check security as one of the methods, especially for a masked design. This allows it to formally verify the circuits by taking in a netlist of a design and forming binary decision diagrams (BDD) for them. The tool also uses many other models like glitch-robust probing model [FGDP$^+$18], active security [DN20], extended fault model [RBSG22], etc. It is claimed to perform better than SILVER [KSM20] and FIVER [RBSS$^+$21] as it not only gives the verification for SCA and FA individually but can also verify combined SCA/FA attacks. It also has the ability to verify composability properties like PINI [CS20], SNI [BBD$^+$16], FNI [DN20], CINI [RBFSG22], etc. This enables the tool to verify gadgets and help reduce the development time for secure designs. It also uses a more advanced BDD engine and has a modular interface for testing different versions of a module. The modular interface is achieved by using a JSON file to annotate the shares for the netlist instead of editing the netlist. This way, with the same port for a module, even for different implementations, new annotations do not need to be generated. This allows for faster and more efficient experimentation.

3.3 Adversary Model

First and foremost, we stress that our adversary model is the same as what has been considered in [DAP$^+$22] and various studies devoted to SCA against NNs. As illustrated in Figure 2, the NN provider trains the model in an offline fashion, and the adversary is the user performing the inference. In this regard, valuable assets of NNs consist of their architectures and the parameters critical to achieving reasonable accuracy [BBJP19]. The countermeasure developed in [DAP$^+$22] has not been concerned with the former and attempted to protect the parameters in NNs (e.g., weights) solely. Therefore, without loss of generality, our adversary attempts to induce leakage by using the circuit components themselves. For this purpose, she collects power/EM traces from the device that she possesses either via direct access or remotely, see, e.g., [SGMT18, ZS18, DKAA22]. The adversary follows a chosen-plaintext-type attack model, where she sends her inputs to the device to be classified and captures multiple traces, being further used to launch power/EM

SCA [XCC$^+$20, YMY$^+$20, BBJP19, DCA20b, YKO$^+$20, XAQ21, BJP22]. Notice that none of the voltage/EM fault injection and template/profiled attacks has been in the scope of [DCA20a].

Moreover, the countermeasure proposed in [DCA20a] aims to ensure that the information about parameters never leaks during any intermediate computation through a *first-order* power/EM-based SCA. In other words, the proposed protection scheme masks all intermediate computations. Furthermore, the t-probing model [ISW03] and robust-probing model [FGDP$^+$18] are used to provide security guarantees for their proposed masking scheme. The former model takes into account an adversary who observes the values of at most t wires in the masked circuit. The security is achieved if and only if the value on each of those t wires can be simulated using solely randomness. To reflect the impact of physical faults in hardware, such as glitches, transitions, and coupling, the probing model is enhanced by considering glitch-extended probes [RBN$^+$15] to obtain the robust-probing model. The glitch-extended probes, as their name implies, are relevant to the notion of glitches, where the probes leak the value of the probed wires as well as all the wires in the fan-in until the last synchronization point.

4 FPGA-based Accelerators and ModuloNET

Before elaborating on our heat generation method, this section gives insight into aspects of modern FPGA-based accelerators that are crucial to understanding why our heat generation method can be applied to masked FPGA-based accelerators in practice. In the second part of this section, an example of masked NNs implemented on FPGAs has been given, namely ModuloNET [DAP$^+$22]. We stress that this example is selected thanks to its intrinsic characteristics, including theoretically sound, impressive side-channel resilience, and lightweight enough to be implemented on FPGAs.

4.1 FPGA-based Accelerators: Pros and Cons

As a result of decades of study and practice, implementation of NNs on FPGAs has become pervasive in various research fields and commercial applications and achieved satisfactory products [WGY$^+$16]. Parallelism, modularity, and dynamic adaptation are some of the main computational features of NNs, which can be met when implementing them on FPGAs [MHS08]. Although there are challenges to face, including scalability and precision, compared with graphics processing unit (GPU) acceleration, FPGA accelerators can achieve at least moderate performance with lower power consumption. The latter is of great importance since the ever-growing volume of data has been leading to exceedingly high power consumption (and consequently, temperature) in data centers [USA22]. Nonetheless, FPGAs have relatively limited computing resources, memory, and input/output (I/O) bandwidths; hence, a great deal of attention needs to be paid in order to develop complex and massive FPGA-based accelerators. In this respect, multiple approaches have been devised to optimize the design of FPGA-based accelerators in terms of throughput and latency. Here throughput means that more data can be analyzed in a given amount of time, whereas the latency should be within the range specified by a service objective.

Challenge 1: Throughput. To optimize the design of NNs by considering the throughput as a metric, the data should be accessed every clock cycle and fed into the network. Batching (i.e., processing a batch of multiple input samples together) is a technique taken as a step towards this, although it increases processing latency and implementation complexity. Therefore, in practice, using large batch sizes is not practical; see, e.g., [NSS$^+$16]. In response to this, processing a stream of input data on FPGA accelerators is required, in particular for streaming applications, including image/video processing applications, real-time vision algorithms, and network packets encryption algorithms that are all FPGA-friendly data-intensive applications [RBK19, RHCM+16, RHL+18]. In traditional applications of FPGA accelerators, e.g., for computation-intensive tasks, access of the user to the memory shared between her and the FPGA has been restricted by, e.g., employing a hierarchy of dynamic random access memory (DRAM)/BRAMs in OpenCL platforms [SGS10]. This has, obviously, dramatically impacted the throughput streaming application; hence, methods have been developed to allow FPGA BRAMs to transfer data point to point every clock cycle [RHL+18]. Clearly, when the user has direct access to the FPGA (no memory hierarchy like the one in OpenCL), achieving such a high throughput is even more straightforward, see, e.g., [ZSZ+17, CSJC10]. Therefore, accelerators attempt to store inputs inside the chips into memory resources before any calculation takes place [SFM17, GYSC17, ZLS+15]. This is indeed helpful to reduce *latency* as well.

Challenge 2: Latency. Latency can refer to the *inference* latency, namely the time taken to process one unit of data given that only one unit of data is processed at a time; however, another aspect of data processing is more critical from the perspective of our study: the memory access latency. What has been suggested in the literature is to generally balance the computation throughput and memory bandwidth [SSEM18, ZP17]. In doing so, using external memories to store weights, especially for convolutional neural networks (CNNs), cannot be recommended due to how the throughput of such a design is limited by the external memory bandwidth [MVZ+21, CLL+14, LFJ+16, CLL+14, DFC+15]. Additionally, frequent access to the off-chip memory also introduces high energy consumption and, consequently, higher temperature [Hor14]. Although one might think that careful scheduling of operations can result in a significant reduction in external memory access, the widely accepted remedy, i.e., data buffering, imposes another difficulty, which is the limited buffer size. Data buffering, in fact, has often been paired with using external memories, aiming to tackle the issue with the limited on-chip memory.

The challenges discussed above are generic in the sense that designers of FPGA accelerators have to tackle them irrespective of the security issues, for instance, resiliency to SCA. In fact, designing an FPGA accelerator with optimized throughput and latency is a hard-to-attain objective, let alone how this could be securely handled in the case of a masked NN. ModuloNET [DAP+22] is one of those proposals attempting to tackle all these aspects, including side-channel resiliency, together. At least for the networks showcased in their paper, no external memory has been used to store the weights so as not to cause harm to the throughput. Moreover, although masking imposes some additional cycles and consequently increases the latency, it is argued that the percentage of this increase is insignificant, thanks to the already high latency of the sequential design. At the time of writing this paper, the design of ModuloNET has not yet been made publicly available; therefore, we have had to follow the instructions and methodology given in [DAP+22] to implement it. This, however, even helps improve the design of ModuloNET[2].

4.2 ModuloNET: An Example of Masked NNs

The implementation uses masking for provable security against first-order attacks in the t-probing and glitch-extended probing models. Their binary NN, i.e., with binary weights and activation function (AF) [CHS+16], includes five layers, one input layer, three hidden layers, and one output layer. All the layers are fully connected, making the design a multi-layer perceptron (MLP). The input layer consists of 784 neurons, and each hidden layer has 1024 neurons, while the output layer has ten neurons, compatible with the size of images in the MNIST data set. The design incorporates calculations in both binary and arithmetic-sharing schemes. Therefore, masking of both natures has been used in

[2]Our design will be available upon acceptance of the paper.

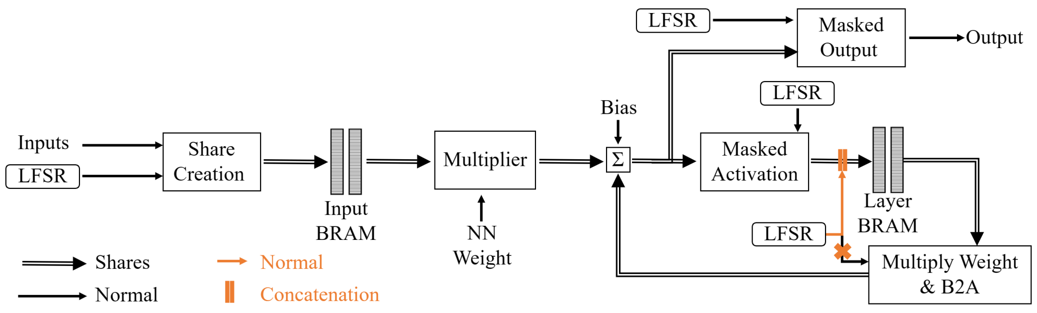


Figure 3: Design of ModuloNET cf. [DAP+22]. Input BRAM is placed after the shares are created and before the trigger to separate share creation from the Leakage Test, whereas layer BRAM is placed to store values per layer for computation. The orange-colored design shows changes that we made to the original design.

the design, and conversions between them have been applied. The masking scheme used in [DAP+22] is domain-oriented masking, where 2-input DOM-indep AND gates (refer to Section 3.1) are used in various places in the design.

Each neuron of the layer is shown in Figure 3. The design includes two BRAM-based memories, which store values used in computations for every neuron. In this way, the design sequentially calculates the values of the neuron one after another. Every neuron calculation includes Summation, Masked Activation, and B2A converter. The input layer neurons will additionally use the input BRAM, the Multiplier (shown in Figure 3), and the output layer neurons will use the Masked Output layer. Each of these modules has been discussed briefly below. Complete details regarding the operations and theoretical proofs for the security can be found in [DAP+22].

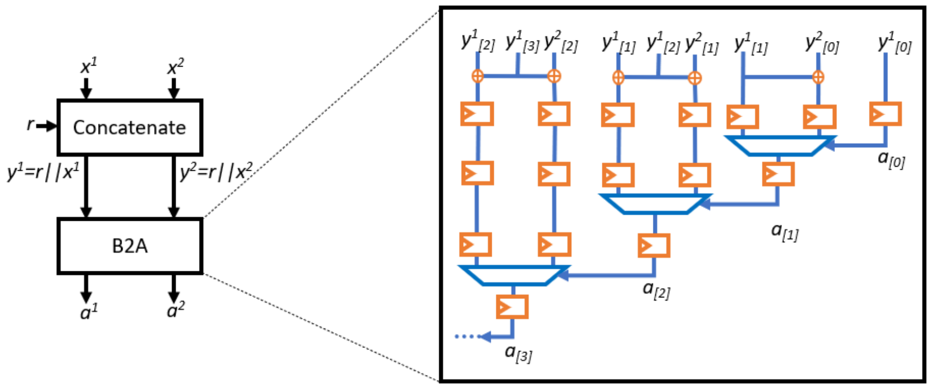
Input share creation. Pixels are 8-bit inputs for the NN. As the design is a masked implementation, the inputs need to be converted into shares. The shares are created using a linear-feedback shift register (LFSR) by subtracting the random value from the pixel value. So, one share is the random value, and the other share is the subtraction of those. These shares are arithmetic shares because they are integer values. These shares can be generated on-the-fly and used for each neuron calculation.

Input weight multiplication. First, we have the weight multiplication for the input layer. The weights here are binary weights. The multiplication with the weight results in either the same value or the complement value. This functionality is implemented as a multiplexer (MUX). Also, to keep the shares independent, the shares are calculated using a parallel implementation of the same modules.

Summation. After multiplication, shares are aggregated by the summation module. *In every cycle*, a new pair of shares are read from BRAM to be summed. This summation occurs for all the inputs for the current neuron, and then the bias is added. The summation module is also connected to the B2A module for the hidden layer and output layer computations and receives the input from the previous layer.

Masked activation. The summation results for the neuron are given as input to the activation layer. The activation layer is a non-linear function; therefore, it needs to be implemented using DOM gates. For this particular NN, we only need the carry-out of the summation for the result of the AF (more details about the AF used here can be found in [DAP+22]). This is accomplished by using a Kogge-Stone adder. The adder computes both the input shares and outputs the shares of carry-out. These output shares are Boolean shares and are stored in the layer BRAM. Layer BRAM is filled with the output of the AF for all the neurons in a layer before calculating the next layer.

B2A. Now, for the next layer, the input is the output from the previous layer, i.e., the values stored in the layer BRAM. For multiplication on these layers, XNOR-POPCOUNT



x^1
x^2
r
Concatenate
$y^1=r||x^1$
$y^2=r||x^2$
B2A
a^1
a^2
$y^1_{[2]}$ $y^1_{[3]}$ $y^2_{[2]}$ $y^1_{[1]}$ $y^1_{[2]}$ $y^2_{[1]}$ $y^1_{[1]}$ $y^2_{[0]}$ $y^1_{[0]}$
$a_{[0]}$
$a_{[1]}$
$a_{[2]}$
$a_{[3]}$

Figure 4: Boolean to arithmetic converter module in [DAP+22]. The B2A module consists of the concatenate module and Golic's B2A converter [Gol07].

is performed, which includes the B2A module. The B2A module converts the Boolean shares generated by the AF to arithmetic shares for summation. To convert the 1-bit Boolean share to 15-bit arithmetic shares, concatenation is performed before using the conversion algorithm as shown in Figure 4. Here, 1-bit Boolean share, x^1 *and* x^2 are concatenated with 14-bit random number, r to form 15-bit numbers, y^1 *and* y^2. These are converted to 15-bit arithmetic shares, a^1 *and* a^2 using Golic's protected design [Gol07]. After the conversion of shares, they are left shifted and fed to the summation module. This process is repeated for all the layers.

Masked output layer. For the output layer, the change is that instead of the masked activation module, the summation outputs are processed by the masked output layer. Similar to the masked activation layer, the output layer AF is a non-linear function. Thus, it also needs masking to keep the output shares independent. Three modules and a register file achieve this. Next, we will discuss all of these modules.

First comes the A2B Converter, which converts the Arithmetic summation shares to Boolean shares. This conversion is essential as the masked output layer performs the binary calculation. The values are stored in the register file. Before processing the output layer, we need values for all the output layer neurons. It is the reason for the register file after the A2B converter. Once all the values are ready, the output layer process starts with the threshold module.

For the NN, the threshold module is used to check if the scores are below a certain threshold (details on how the threshold is decided can be found in [DAP+22]). If it crosses the threshold, the output is 0. This is implemented with AND gates replaced by DOM gates to make it masked. The output is given to the masked comparator module using DOM gates to protect the module. The comparator module checks which class has the highest confidence score, so we compare it with the current global and local max results. The higher confidence score-based values are selected, and indexes for the same are also saved into local/global max registers. Masked MUX is used to select these values which use DOM gates for masking. This flow is used to select a class in the output layer.

5 Temperature-induced Leakage from NNs

What paves the way for possible leakage? As explained before in Section 4.1, in practice, accelerators' input introduces a crucial requirement for bandwidth and latency, making the solutions that use off-chip memory less favorable [LFJ+16]. Because accelerators are used in scenarios such as IoT or image processing [LeC19], accelerators must maintain a pre-

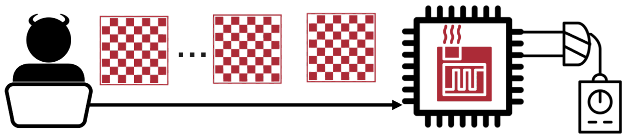


Figure 5: The adversary relies on the fact that at high temperatures, the power consumption associated with different shares is no longer independent of each other. In this regard, the adversary takes advantage of the memory allocated to store the inputs and, by writing alternating '0' and '1' patterns, attempts to increase the operating temperature of the FPGA and detect first-order leakage.

specified input communication bandwidth to meet the need for high throughput [LFJ+16]. To avoid this excessively restrictive requirement, the most recent approaches have stored input on-chip, i.e., in BRAMs or FFs [LFJ+16] to fulfill the requirement for input bandwidth. As an example, Shen et al. [SFM17] have used 1108 BRAMs to implement SqueezeNet on Xilinx Virtex-7 FPGA, which leads to 38% BRAM utilization. In another approach, Li et al.[LFJ+16] have utilized 1913 BRAMs (65.07% of available Xilinx VC709 BRAMs) for their AlexNET hardware accelerator. Zhang et al. [ZLS+15] have utilized 1024 BRAMs out of 2060 available BRAMS (50% BRAM utilization) in the Xilinx Virtex-7 FPGAs to implement their CNN accelerator on the FPGA. BRAMs mentioned above are partially used for input storage, where a minor portion of the BRAM utilization is for storing intermediate values. We emphasize that it is possible to use BRAMs to store the input, which can be updated from an off-chip source frequently in a pipeline manner [DFC+15, CLL+14]; nonetheless, this can result in an increase in the latency and/or a reduction in the throughput (see Section 4). When storing the input on-chip in BRAMs, writing the input (i.e., image pixels) in a BRAM row-by-row is a common practice, see, e.g., [KKL+19, GBS+19]. This allows reading a whole row in a single clock cycle and, consequently, reduces the latency.

It is worth mentioning that not all FPGA-based accelerators store inputs in BRAMs, and they might utilize a chain of FFs for their input storage [LFJ+16]. In this respect, some approaches have considered the usage of FFs to provide their accelerator with pipeline computation ability. For instance, Zhou et al.[ZJ15] replaced BRAMs with 46140 FF (7.6% of the FFs available on Xilinx Virtex-7 FPGA) to implement the ImageNET classifier accelerator to achieve the maximum pipeline calculation benefits. Guan et al. [GYSC17] introduced a long short-term memory recurrent NNs (LSTM-RNNs) accelerator. Thanks to the nature of the LSTM-RNNs accelerator, they require to have frequent access to the previous LSTM-RNN states and input values every clock cycle; hence, due to the limitation of BRAMs in this matter, they have mapped the BRAMs to a chain of FFs to have frequent access to the previous LSTM-RNN states and input values. They have utilized 181364 FF (29.91% of available Virtex7-485t FPGA FFs) and 112 out of 2060 BRAMs (5.44% BRAM utilization) to implement the LSTM-RNNs accelerator. Not only is storing inputs in FFs suggested for such a specific accelerator, but also, in general, memory-hungry NNs can benefit from that to offer frequent memory access. As a prime example, to benefit from pipeline calculation for CNN accelerators, the most recent approaches have highlighted using FF to store input images and their features instead of BRAMs [ZLS+15, GYSC17].

5.1 Inducing Leakage through Internal Heat Generators

The core idea underlying the heat-induced leakage is to generate heat inside the FPGA by flipping the input image (i.e., writing alternating '0' and '1' input patterns into memory) frequently to toggle the corresponding BRAMs/FF, thereby increasing dynamic power consumption and subsequently chip temperature. This simple but effective concept is realized by an adversary who solely feeds the inputs to the design in each and every clock cycle, see Figure 5. This type of adversary is completely agnostic about the design of the NN, but leverages the *internal* heat generator to make the leakage from the masked NNs detectable. The foundation of this has been laid by Happe et al. [HHAP12] and Agne et al. [AHH$^+$14], who have indicated that one of the primary sources of heat in FPGAs is a significant number of BRAMs and FFs pipelines. In addition, they showed that reading/writing from BRAMs and FFs also generate extreme heat.

To understand the effect of such heat generators on masked NNs, let us focus on ModuloNET [DAP$^+$22]. Similar to many other accelerators [CLL$^+$14, DFC$^+$15, DCA20a, DCA20b, DCA20a], input images and masked AF outputs in ModuloNET have been stored in BRAMs; therefore, ModuloNET meets the needs of the adversary for generating heat on FPGAs. As mentioned in Section 3.3, the adversary cannot control the input/output of the masked AF when the FPGA is operating. However, the adversary can continuously feed flipping images into the FPGA, as shown in Figure 5, which are stored in BRAMs. To make the most of the heat generator, it is necessary to change multiple bits every cycle. This can be achieved by writing alternating '1' and '0' patterns into BRAMs; hence, the adversary crafts image pixels with such a pattern to obtain, so-called flipping images. We emphasize that, in contrast to heat generators in [HHAP12, AHH$^+$14], we do not write into memory in a pipeline fashion, but in parallel, in compliance with the design of NNs.

Now the question is why exploiting such an internal heat generator would impact the masked design. In fact, the heat generated inside the FPGA is the result of high power consumption, which directly affects how (in)dependent the power consumption of the shares (and the functions being operated on them) are on each other. This has been explored in [DCEM18], where it is shown that "the power consumption of a function operating on a share influences the amount of power consumption of other functions simultaneously operating on other shares." De Cnudde et al. [DCEM18] have empirically examined this (in fact, in "an artificial lab environment") that such a phenomenon is observable within the temperature range between 50 °C and 70 °C. These are the ranges selected in [DCEM18] to express the extreme effect of temperature on first-order leakage. While they could enjoy the freedom of selecting these by employing the climate chamber, we attempt to find out if the internal heat generator executed by solely feeding the alternating inputs could have an influence on the first-order leakage from the design. In the same vein as [DCEM18], we aim to pinpoint the existence of leakage in correctly-implemented masked NN.

We have elaborated on how the BRAM-based heat generator can induce first-order leakage from masked NNs thus far, although the same can hold for other types of heat generators. As demonstrated in [HHAP12, AHH$^+$14], a chain of FFs can also generate heat effectively. Therefore, we examine whether, in both cases of utilizing BRAMs and FFs for storing NN inputs, the heat can be generated by flipping the value of inputs (e.g., image pixels) frequently.

Comparison with the most relevant heat generation methods. First and foremost, we stress that De Cnudde et al. [DCEM18] have investigated the impact of extreme temperature on masking in general. To this end, a climate chamber has been used to operate the device at higher temperatures. While the concept has been well understood, no practical heat generation has been proposed to increase the temperature internally to make the masked implementation leak. On the other hand, the heat generation in [ATG$^+$19] has leveraged a large number of simultaneous write-collisions to cause voltage drop and, consequently, a significant temperature rise. Here write-collision refers to the scenario where both ports of BRAMs in *dual-port* BRAMs are writing different data to the same memory address. It has been observed that FPGAs (i.e., including series crafted by Xilinx, Intel, Microsemi, etc.) contain dual-port random-access memory (RAMs) with concurrent writing possibility, which can be turned into a serious vulnerability if opposite logic values are written to an address to create a transient short circuit. While our heat generation

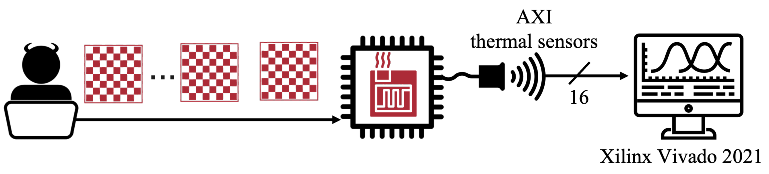


Figure 6: Experimental setup used to perform the thermal test.

Table 1: Hardware resource allocation in our implementation of ModuloNET used to perform t-test in the case of BRAM-based and FF-based heat generators (HGs).

Resource	Used		Utilization (%)		Available
	BRAM-based HG	FF-based HG	BRAM-based HG	FF-based HG	
LUT	15807	24072	24.93	37.96	63400
FF	7782	**24102**	6.13	**19**	126800
BRAM	**131**	71	**97.03**	52.59	135

relies on writing flipping pixels into BRAMs, as opposed to [ATG+19], single-port BRAMs are utilized in our case. The implication of this is that our heat generation is not restricted to designs with dual-port BRAMs.

6 Experimental Results

6.1 Measurement Setup

To evaluate how effective our heat generators are and understand their impact on first-order leakage, ModuloNET is implemented on Artix-7 FPGA. For this, We have used Vivado 2021 [Xil21]. The BRAMs are generated using the IP catalog of Vivado. To maintain the independence of shares and the security of the design, we had to tune some of the synthesis and implementation parameters. We disabled the LUT sharing and the optimization option as well as enabled `keep_equivalent_registers`. This setting forces Vivado to place the shares in different locations, as mentioned in the design, and not optimize them. Power traces have been captured using the Riscure setup, including LeCroy wavePro 725Zi as the oscilloscope. The design is loaded into the Chipwhisperer CW305 target, and the capturing process is controlled by the Chipwhisperer Lite board. The design frequency is set to 10 MHz while the oscilloscope capturing frequency is 100 mega samples per second (MS/s). The traces are collected with a 1s delay to prevent the bandwidth of the oscilloscope from becoming the bottleneck.

The CW305 target board is a customized board for effective side-channel evaluation purposes; however, it, unfortunately, lacks the system monitoring sensors. Hence, we have used the PYNQ-Z1 board with Artix-7 FPGA (package number FTG256) to perform thermal tests. Figure 6 illustrates our temperature evaluation experimental setup. We have used the XADC thermal sensors and read the 16-bit temperature channel analog to digital converter (ADC) via Xilinx Vivado 2021 local server. To communicate with the XADC module, we used the PYNQ-Z1 AXI streaming port with a refreshing period of 1s. To establish communication between the personal computer (PC) and the XADC and store the sensor data, Happe et al. [HHAP12] used the MicroBlaze processor embedded alongside the design. However, as the MicroBlaze processor generates heat on FPGA, we have used the Xilinx Vivado tool command language (TCL) to store XADC temperature sensor data with the bandwidth of 1s to prevent any heat generation from other sources rather than the design.

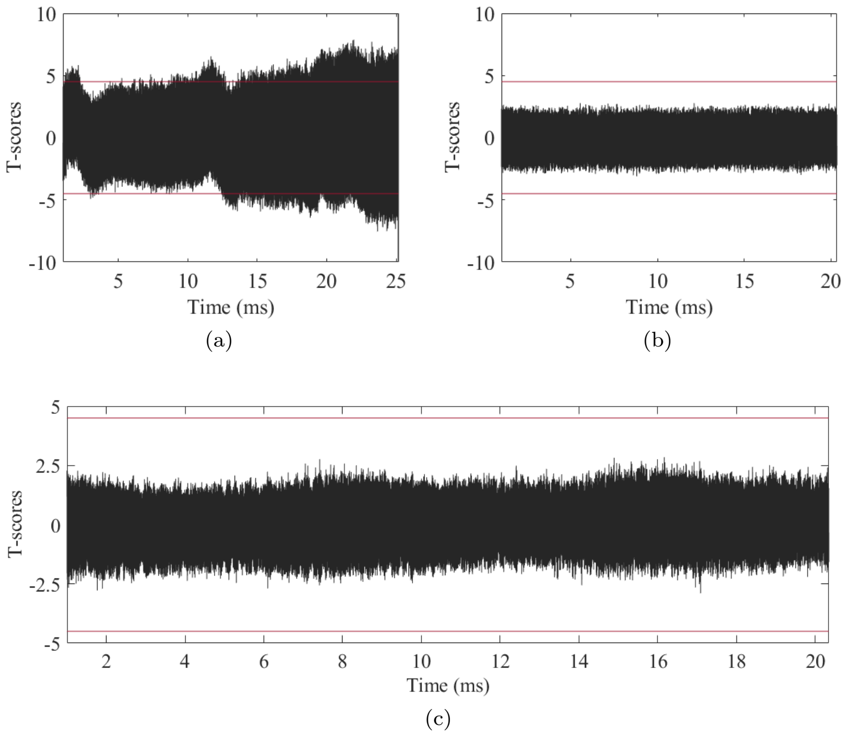


Figure 7: T-test results for the entire design, including all modules, explained in Section 4.2 based on (a) 100k traces captured from the original design (as depicted in Figure 8 in [DAP$^+$22]), (b) 100k traces from our improved design (see Figure 3), and (c) 1M traces captured from our design. The red lines in the figures show the threshold ±4.5 corresponding with 99.99% confidence.

6.2 Our implementation of ModuloNET

Our ModuloNET is implemented with regard to the description provided in [DAP$^+$22] and includes the modules explained in Section 4.2. As mentioned in [DAP$^+$22], the share creation during the t-test calculation would lead to leakage due to input correlation. To avoid this, they calculated the inputs and corresponding shares before calculation. We have employed a similar approach where we calculate the shares on the device and store them in the input BRAM before starting the t-test. Dubey et al. [DAP$^+$22] have implemented a multi-layer Perceptron (MLP) with 784 input nodes, 3 hidden layers with 1024 nodes, and 10 output layer nodes to argue about the scalability of their approach; however, to accelerate the trace collection process, the number of nodes in the hidden layers is reduced to 64. It is also justified that thanks to the repetitive nature of NN computations and the sequential design (i.e., computing one node at a time), the leakage assessment is generalizable to the larger NN. Similarly, we take a reasonable size of the NN into account to reduce the number of clock cycles and, consequently, the time taken to collect a trace, as well as the size of the data stored for a single trace. Our network has 100 input nodes, 64 nodes in the hidden layer, and 10 nodes in the output layer. This reduced design takes 20,348 clock cycles per trace and around 200kB of data per trace. We stress that if more input nodes were considered, the operating temperature would be even higher than observed in our experiments; hence, our results give an optimistic estimate of what can happen in practice.

Since we have considered 100 input nodes, we need 100 pairs of shares of all the inputs. We have allocated 2 BRAM, one for each share, with a size of 78,400 (corresponding to 784 pixels in MNIST images and each with 100 input nodes) with 15-bit width. The 15-bit width is the result of padding the 8-bit pixels to 15, as performed in the original design in [DAP+22]. While 2 BRAMs (FIFO36) are consumed by layer BRAM (see Figure 3), 69 BRAMs (FIFO36) are used to store only 100 pixels per image to further accelerate the trace collection process.

Moreover, calculating the t-scores requires capturing traces from a set of fixed and a set of random inputs, which prohibits us from having the luxury of flipping input images entirely during the t-test process. Hence, we have considered a set of 120 FIFO18 BRAMs (half the size of FIFO36 BRAMs) in the design and flipped them simultaneously with the t-test process. Further, when examining the impact of FF-based input storage on the operating temperature, we modified our design by replacing all the BRAMs with register files for storage. In this case, 120 FFs with a width of 8-bit have been added, similar to BRAMs configured to store 8-bit numbers (same as the size of pixels) with a depth of 16. Writing into memory occurs every cycle, and thus the address counter stays constant at zero. We stress that adding these BRAMs and FFs help us mimic the effect of the flipping input images without interfering with the trace-capturing process. Note that the total number of bytes in flipping and t-test inputs is much less than the number of input bytes that can be flipped by the adversary in a real-world scenario, where the actual input size of the MNIST image is 784 pixels. Furthermore, since the t-test inputs are not changed once the t-test is started, the change in the operating temperature reflects the situation where the adversary does not take full advantage of the heat generator. The complete list of resources used by the design when performing the t-test is provided in Table 1. In terms of the number of LUTs and FFs, although we tried to follow the original design as closely as possible, we have higher resource utilization compared to ModuloNET [DAP+22] (5635 LUTs and 5009 FF). Nonetheless, since we compare the operating temperature and the leakage solely by considering our own design under different conditions, this does not cause any issues.

Verifying the design using VERICA. As seen in Figure 7(a), the t-test results for the original design as described in [DAP+22] showed leakage. To figure out the cause of the leakage, we used a design verification tool, namely VERICA [RBFSG22]. For this purpose, first, we need to generate a gate-level netlist for the individual modules using the Synopsys design compiler [Syn20]. The Tcl script has to follow constraints for the VERICA tool. Next, we need to define the security annotations for the input and output, such as shares, refresh bits, and control signals. This information has to be defined in a JSON file which can be used to test different versions of the same module, which is a major improvement over SILVER [KSM20]. We tested different modules using VERICA and found that the B2A module failed the test. In fact, the B2A module passed the d-probing test but failed the glitch-extended probing test. There was one probing point inside the B2A module, where the leakage of the Boolean shares was detected. By looking at the netlist, we found the cause of the problem at the point where Boolean shares were written/read into/from BRAM. The point is that the prerequisite for Goubin's B2A algorithm is the field being the same for inputs and outputs. Therefore, if the Boolean shares have less than k bits (see Section 3.1), fresh random values can be used to pad them (concatenating with the random values as done in [DAP+22]). If this padding is performed after reading the values from BRAM, the glitch-extended probing test fails. By performing padding before writing the Boolean shares into the layer BRAM, we could resolve the problem. This change is shown by the orange color in Figure 3. The t-test results for 100k traces collected from the enhanced design are depicted in Figure 7(b). To further demonstrate the security of the design in terms of first-order leakage, we calculated the t-scores for 1M traces as well.

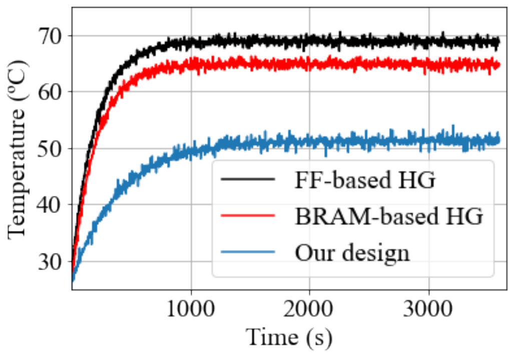


Figure 8: Temperature testing results for our implementation of ModuloNET when giving a normal (not flipping) image, flipping inputs are written into BRAMs (BRAM-based heat generator), and FFs (FF-based heat generator). Note that these heat generators (HG) are indeed part of the design, specifically, the memory storing the inputs.

The maximum t-score obtained for 1M traces in our design(Figure 7(c)) is 2.84.

6.3 Temperature Measurement Results

As mentioned earlier in Section 5, there are two ways of storing input data given to FPGA-based accelerators, namely, using BRAMs and FFs. To evaluate our internal heat generators, we have done experiments in two cases, namely giving (i) a normal (not flipping) image and (ii) one with alternating '0' and '1' patterns (flipping image). Note that the former also corresponds to the heat generated when processing the t-test inputs stored in the memory. In both cases, the XADC sensor is used to collect 3,600 temperature samples over 3,600s. Then, we let the board cool down for 60 minutes (m), even more than the suggested cool-down time recommended in [HHAP12, AHH+14], and provided ModuloNET with flipping images. Figure 8 shows the heat generation results in these two experiments.

It is observable in Figure 8 that giving the normal images to ModuloNET with BRAM input storage generates heat up to a maximum of 54.3 °C. While with the flipping images, its operating temperature reaches the maximum of 64.9 °C in an even shorter period. The changes in the input image increase the FPGA temperature by 10.6 °C. To investigate the effect of heat generation on the accelerator with the FF-based input storage, we have replaced BRAMs storing the input with a set of FFs. Figure 8 shows the results of ModuloNET in that case as well. Compared to ModuloNET equipped with BRAM to store the input, we can observe that using FFs for storing the input results in a higher temperature, namely the maximum of 70.2 °C, which means the frequent changes in the input image leads to 15.8 °C increase in the temperature.

6.4 Leakage Detection

After verifying the impact of giving flipping images on the operating temperature of the FPGA embodying ModuloNET, we investigate how the resulting temperature rise can affect the first-order leakage. The goal of experiments done in this regard is to understand whether a first-order secure design, i.e., our design of ModuloNET, exhibits first-order leakage if flipping images are fed into it and, thus, increases the operating temperature. For this purpose, we compare the t-scores calculated for traces collected from the design when providing normal (not flipping) and flipping images under both scenarios: BRAM-based and FF-based input storage. In these experiments, we first wrote the '0' and '1' alternating patterns into the memory long enough so that the FPGA reaches a sufficiently high temperature. Afterward, the traces were collected. More precisely, after about 20m (1,200s), the temperature becomes almost stable and reaches its maximum value (see Figure 8); hence, the traces were collected 20m after beginning the experiments. Note that an adversary interested in detecting the leakage does not need this information as she can simply give the flipping images for hours to make sure that the operating temperature is sufficiently high.

Furthermore, we should highlight that we start with several tens of thousands of traces to see after capturing how many traces the first-order leakage is detectable. In this regard, we collected between 100k and 300k traces from the device in our experiments. The effect of activating the heat generators, i.e., giving a flipping image to ModuloNET, can be seen in Figure 9. As can be seen in Figure 9(b) , under the scenario where flipping inputs are fed into ModuloNET, the t-scores calculated for 100k traces are much increased compared to Figure 9(a), where normal inputs (not flipping inputs) are considered. Notice that no first-order leakage has been seen for ModuloNET when conducting an evaluation with 1M traces as depicted in Figure 7. Further, Figure 9(b) illustrates a sample trace and the average of 1,000 traces showing how the power characteristics of the design are affected by the extreme heat. Moreover, as marked in Figure 9(b) (see the bottom row), the t-scores do not always fall within the desired threshold, implying the rejection of the null-hypothesis with the confidence of 99.99%.

We repeated this experiment for the ModuloNET with FF-based input storage. In that case, no first-order leakage was detected after 100k traces, although by increasing the number of traces to 150k, some t-scores exceed the threshold. As it is observable in Figure 9(b) and Figure 9(c), BRAM-based heat generation causes more t-scores exceeding the threshold even with fewer traces compared to the FF-based case. It is interesting since the device reached a higher temperature for FF-based heat generation; however, the first-order leakage is detectable after a larger number of traces. This phenomenon has indeed been observed in [DCEM18] as well. Our results also confirmed that when increasing the number of traces to 300k, the first-order leakage becomes even more visible; nevertheless, in Figure 9, we present the result for the minimum number of traces where the t-scores are not always within the threshold.

7 Discussion

Can we stop generating heat? Reducing the operating temperature of NNs goes hand in hand with designing energy-efficient NNs. In addition to methods devised to predict the energy consumption of a NN and enhance the structure of that accordingly [LWL+20, CJSM17], another line of research has been pursued to allocate memory more carefully to reduce the power consumption in general rather than solely in NNs [GBS+19, TBNG06, TBN+07, KRN+18]. Garcia et al. [GBS+19] have particularly focused on memory-constrained FPGAs in the context of image processing. They have proposed a partitioning method backed by theoretical analysis of its effects on resource usage and power consumption. In this way, instead of allocating either some blocks of BRAM with a considerable size or slicing data to smaller groups before memory allocation on a trial-and-error basis, their proposed procedure helps the designer select the right memory configuration and optimize on-chip memory usage. Consequently, their approach can decrease the operating temperature. Nevertheless, the effect of Garcia's partitioning method [GBS+19] should be studied more carefully when being applied against masked implementation. At the moment, it is unknown how it would counter the risk of unexpected leakage or make heat-induced leakage disappear.

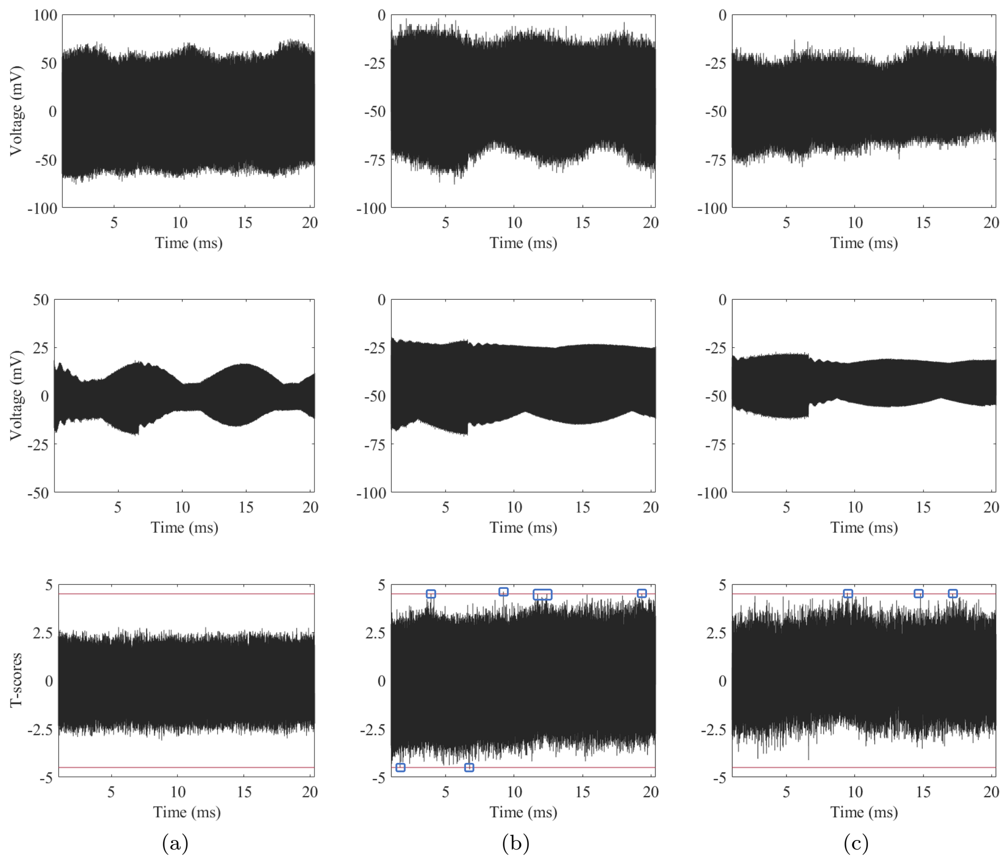


Figure 9: Results for ModuloNET (a) with normal (not flipping) inputs, and after writing flipping inputs into (b) BRAMs and (c) FFs. Sub-figures present (top) sample trace, (Middle) average of 1000 randomly chosen traces, and (bottom) t-test results for 100k traces for (a) and (b) and 150k traces for (c). In the bottom row, the red lines in the figures show the threshold ±4.5 corresponding with 99.99% confidence, whereas the t-scores exceeding the threshold are marked with blue squares.

Can off-chip memory provisioning be helpful? It is a valid point that one could store the *inputs* off-chip and then schedule how the NN should be provided with the inputs efficiently to account for the heat-induced leakage. This could prevent the adversary from generating heat on-chip and protect the design. Yet, this approach causes the bandwidth bottleneck and reduces the efficiency of the accelerator (see Section 4). In some cases, even if the problem with bandwidth would be resolved, off-chip resources for buffering the inputs might not be available due to device constraints. This introduces a trade-off between security and resource requirements. It might be a designer's choice as to whether to go for higher throughput by using on-chip memory and sacrificing security (refer to Section 5) or sacrificing performance for security.

What other heat generators are possible? Besides writing alternating patterns into BRAMs, several other circuit components could generate heat, including LUT pipeline, shift right logical (SRL) pipeline, FF pipeline, LUT-FF pipeline, digital signal processor (DSP) pipeline, BRAM pipeline, LUT oscillator, and SRL-FF pipeline [HHAP12]. Among all these, based on the results presented in [HHAP12, AHH+14], BRAM and LUT-FF pipeline could generate the most heat on FPGAs. The first option is similar to our heat generator (ours is not in a pipeline form), where having access to the design input is sufficient to increase the temperature. However, if BRAMs only store intermediate computations (e.g., in BoMoNet [DCA20a], AF BRAMs), generating heat is not under the control of the adversary, although changes in the values stored in those BRAMs might increase the temperature. In a nutshell, allocating a significant number of BRAMs/FFs could result in generating sufficient heat that leads to detectable leakage; we suggest that designs with these components could be assessed to avoid unexpected leakage.

Moreover, there are other circuit components that could cause high temperatures and are used in NNs generally, although, to the best of our knowledge, they have not been used in masked ones. In this regard, DSPs have been used in the NN design, which is a good source of heat generation as they operate in a combinational manner under a high clock frequency [HHAP12, AHH+14]. Hence, we highlight the possibility of heat-induced leakage if such a structure is used in masked NNs. As a recommendation, we refer to the results by Happe et al. [HHAP12] indicating that minimizing the interconnection between these components inside the core through pipelining can generate heat on an FPGA.

Leakage verification of B2A and A2B conversion algorithms. As discussed before, we have observed leakage if the B2A module is employed with respect to instructions given in [DAP+22] and as depicted in their Figure 8. For this purpose, we used VERICA [RBFSG22], which discovered the leakage caused by writing/reading the single-bit Boolean shares into/from BRAMs. The issue with verifying the leakage from verified A2B/B2A conversion algorithms has recently been addressed in [GPM22]. More specifically, their findings indicate glitch-based issues for hardware A2B as presented in [CGV14] and transition-based leakage in Goubin's schemes in software [Gou01]. They have, in particular, pointed out several registers overwrite leaks in Goubin's A2B/B2A algorithm implemented in software. Our results complement theirs by demonstrating that Goubin's B2A algorithm should be carefully implemented in hardware, in particular, when dealing with single-bit Boolean shares. We believe that this might have been addressed in Dubey's design as well, irrespective of the design shown in their Figure 8 [DAP+22]), although we could not verify it since their code was not available.

8 Conclusion

To support a large number of calculations in NN accelerators, many approaches have been proposed using FPGA to achieve the maximum benefit of parallel calculations. In order to protect such accelerators, among various proposed techniques, masking has received a great deal of attention in the recent work [DCA20a, DCA20b, DAP+22]. Our paper introduces a methodology for inducing first-order leakage in FPGA-based accelerators that offer first-order side-channel resilience through masking. Starting from the observation made in [DCEM18], our technique attempts to make the power consumption of different shares dependent on each other by increasing the temperature. For this purpose, in contrast to [DCEM18], we apply novel internal heat generators composed of the NN's components, making our heat generators cheap and inseparable part of the design. To verify the effectiveness of our method experimentally, we consider ModuloNET [DAP+22] as one of the most recent examples of masked NNs. We observe that by writing alternating patterns into BRAMs (or, alternatively, FFs), ModuloNET shows unexpected first-order leakage as the FPGA embodying it reaches the temperature between 50 °C and 70 °C. We emphasize that our paper aims to highlight the possibility of unexpected leakage in correctly-implemented masked NNs by means of t-test leakage assessment. We further note that the modules in our design are tested via VERICA [RBFSG22], where a new vulnerability in the hardware implementation of Goubin's B2A algorithm is identified and resolved in our design. Finally, we discuss possible countermeasure and their associated challenges as well as future direction. For the latter, specifically, security-aware memory allocation for masked NNs is suggested.

9 Acknowledgments

This work has been supported partially by Semiconductor Research Corporation (SRC) under Task IDs 2991.001 and 2992.001 and NSF under award number 2138420. David S. Koblah is supported by the Department of Defense through the Science, Mathematics, and Research for Transformation (SMART) Scholarship-for-Service Program.

References

[AHH+14] Andreas Agne, Hendrik Hangmann, Markus Happe, Marco Platzner, and Christian Plessl. Seven recipes for setting your fpga on fire–a cookbook on heat generators. *Microprocessors and Microsystems*, 38(8):911–919, 2014.

[ATG+19] Md Mahbub Alam, Shahin Tajik, Fatemeh Ganji, Mark Tehranipoor, and Domenic Forte. Ram-jam: Remote temperature and voltage fault attack on fpgas using memory collisions. In *2019 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 48–55, 2019.

[BBB+22] Anubhab Baksi, Shivam Bhasin, Jakub Breier, Dirmanto Jap, and Dhiman Saha. A survey on fault attacks on symmetric key cryptosystems. *ACM Computing Surveys*, 55(4):1–34, 2022.

[BBD+16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 116–129, 2016.

[BBJP19] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. CSI NN: Reverse engineering of neural network architectures through electromagnetic side channel. In *28th USENIX Security Symp. (USENIX Security 19)*, pages 515–532, 2019.

[BH22] Jakub Breier and Xiaolu Hou. How practical are fault injection attacks, really? *IEEE Access*, 10:113122–113130, 2022.

[BJP22] Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. On (in) security of edge-based machine learning against electromagnetic side-channels. In *2022 IEEE International Symposium on Electromagnetic Compatibility & Signal/Power Integrity (EMCSI)*, pages 262–267. IEEE, 2022.

[BSS+13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The simon and speck families of lightweight block ciphers. *cryptology eprint archive*, 2013.

[CGTV15] Jean-Sébastien Coron, Johann Großschädl, Mehdi Tibouchi, and Praveen Kumar Vadnala. Conversion from arithmetic to boolean masking with logarithmic complexity. In *International Workshop on Fast Software Encryption*, pages 130–149. Springer, 2015.

[CGV14] Jean-Sébastien Coron, Johann Großschädl, and Praveen Kumar Vadnala. Secure conversion between boolean and arithmetic masking of any order. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 188–205. Springer, 2014.

[CHS+16] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[CJSM17] Ermao Cai, Da-Cheng Juan, Dimitrios Stamoulis, and Diana Marculescu. Neuralpower: Predict and deploy energy-efficient convolutional neural networks. In *Asian Conference on Machine Learning*, pages 622–637. PMLR, 2017.

[CLL+14] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. Dadiannao: A machine-learning supercomputer. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 609–622. IEEE, 2014.

[Cor17] Jean-Sébastien Coron. High-order conversion from boolean to arithmetic masking. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 93–114. Springer, 2017.

[CS20] Gaëtan Cassiers and François-Xavier Standaert. Trivially and efficiently composing masked gadgets with probe isolating non-interference. *IEEE Transactions on Information Forensics and Security*, 15:2542–2555, 2020.

[CSJC10] Srimat Chakradhar, Murugan Sankaradas, Venkata Jakkula, and Srihari Cadambi. A dynamically configurable coprocessor for convolutional neural networks. In *Proceedings of the 37th annual international symposium on Computer architecture*, pages 247–257, 2010.

[DAP+22] Anuj Dubey, Afzal Ahmad, Muhammad Adeel Pasha, Rosario Cammarota, and Aydin Aysu. Modulonet: Neural networks meet modular arithmetic for efficient hardware masking. *IACR Trans. on Cryptographic Hardware and Embedded Systems*, pages 506–556, 2022.

[DCA20a] Anuj Dubey, Rosario Cammarota, and Aydin Aysu. Bomanet: Boolean masking of an entire neural network. In *2020 IEEE/ACM Intrl. Conf. On Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2020.

[DCA20b] Anuj Dubey, Rosario Cammarota, and Aydin Aysu. Maskednet: The first hardware inference engine aiming power side-channel protection. In *2020 IEEE Intrl. Symp. on Hardware Oriented Security and Trust (HOST)*, pages 197–208. IEEE, 2020.

[DCEM18] Thomas De Cnudde, Maik Ender, and Amir Moradi. Hardware masking, revisited. *IACR Trans. on Cryptographic Hardware and Embedded Systems*, pages 123–148, 2018.

[DFC+15] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. Shidiannao: Shifting vision processing closer to the sensor. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pages 92–104, 2015.

[DKAA22] Anuj Dubey, Emre Karabulut, Amro Awad, and Aydin Aysu. High-fidelity model extraction attacks via remote power monitors. In *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 328–331. IEEE, 2022.

[DMRB18] Lauren De Meyer, Oscar Reparaz, and Begül Bilgin. Multiplicative masking for aes in hardware. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 431–468, 2018.

[DN20] Siemen Dhooghe and Svetla Nikova. My gadget just cares for me-how nina can prove security against combined attacks. In *Cryptographers' Track at the RSA Conference*, pages 35–55. Springer, 2020.

[FGDP+18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable masking schemes in the presence of physical defaults & the robust probing model. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 89–120, 2018.

[GA03] S. Govindavajhala and A.W. Appel. Using memory errors to attack a virtual machine. In *2003 Symposium on Security and Privacy, 2003.*, pages 154–165, 2003.

[GBS+19] Paulo Garcia, Deepayan Bhowmik, Robert Stewart, Greg Michaelson, and Andrew Wallace. Optimized memory allocation and power minimization for fpga-based image processing. *Journal of Imaging*, 5(1):7, 2019.

[GM17] Hannes Groß and Stefan Mangard. Reconciling $d + 1$ masking in hardware and software. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 115–136. Springer, 2017.

[GMK16] Hannes Gross, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In *Proceedings of the 2016 ACM Workshop on Theory of Implementation Security*, TIS '16, page 3, New York, NY, USA, 2016. Association for Computing Machinery.

[GMK17] Hannes Groß, Stefan Mangard, and Thomas Korak. An efficient side-channel protected aes implementation with arbitrary protection order. In *Cryptographers' Track at the RSA Conference*, pages 95–112. Springer, 2017.

[GOKT16] Dennis RE Gnad, Fabian Oboril, Saman Kiamehr, and Mehdi B Tahoori. Analysis of transient voltage fluctuations in fpgas. In *2016 International Conference on Field-Programmable Technology (FPT)*, pages 12–19. IEEE, 2016.

[Gol07] Jovan Dj Golic. Techniques for random masking in hardware. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 54(2):291–300, 2007.

[Gou01] Louis Goubin. A sound method for switching between boolean and arithmetic masking. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 3–15. Springer, 2001.

[GPM22] Barbara Gigerl, Robert Primas, and Stefan Mangard. Formal verification of arithmetic masking in hardware and software. *Cryptology ePrint Archive*, 2022.

[GYSC17] Yijin Guan, Zhihang Yuan, Guangyu Sun, and Jason Cong. FPGA-based accelerator for long short-term memory recurrent neural networks. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 629–634. IEEE, 2017.

[HHAP12] Markus Happe, Hendrik Hangmann, Andreas Agne, and Christian Plessl. Eight ways to put your fpga on fire—a systematic study of heat generators. In *2012 International Conference on Reconfigurable Computing and FPGAs*, pages 1–6. IEEE, 2012.

[Hor14] Mark Horowitz. Computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 10–14. IEEE, 2014.

[HS13] Michael Hutter and Jörn-Marc Schmidt. The temperature side-channel and heating fault attacks. volume 8419, 11 2013.

[HT19] Michael Hutter and Michael Tunstall. Constant-time higher-order boolean-to-arithmetic masking. *Journal of Cryptographic Engineering*, 9(2):173–184, 2019.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *Annual International Cryptology Conference*, pages 463–481. Springer, 2003.

[KGT22] Jonas Krautter, Dennis R. E. Gnad, and Mehdi B. Tahoori. Remote fault attacks in multitenant cloud fpgas. *IEEE Design & Test*, 39(4):33–40, 2022.

[KHEB14] Thomas Korak, Michael Hutter, Baris Ege, and Lejla Batina. Clock glitch attacks in the presence of heating. In *2014 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 104–114, 2014.

[KKL$^+$19] Angelos Kyriakos, Vasileios Kitsakis, Alexandros Louropoulos, Elissaios-Alexios Papatheofanous, Ioannis Patronas, and Dionysios Reisis. High performance accelerator for cnn applications. In *2019 29th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pages 135–140. IEEE, 2019.

[KRN$^+$18] Inderpreet Kaur, Lakshay Rohilla, Alisha Nagpal, Bishwajeet Pandey, and Sanchit Sharma. Different configuration of low-power memory design using capacitance scaling on 28-nm field-programmable gate array. In *System and Architecture*, pages 151–161. Springer, 2018.

[KSM20] David Knichel, Pascal Sasdrich, and Amir Moradi. Silver–statistical independence and leakage verification. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 787–816. Springer, 2020.

[LeC19] Yann LeCun. 1.1 deep learning hardware: Past, present, and future. In *2019 IEEE Intrl. Solid-State Circuits Conf.-(ISSCC)*, pages 12–19. IEEE, 2019.

[LFJ$^+$16] Huimin Li, Xitian Fan, Li Jiao, Wei Cao, Xuegong Zhou, and Lingli Wang. A high performance fpga-based accelerator for large-scale convolutional neural networks. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–9. IEEE, 2016.

[LGFX21] Yukui Luo, Cheng Gongye, Yunsi Fei, and Xiaolin Xu. Deepstrike: Remotely-guided fault injection attacks on dnn accelerator in cloud-fpga. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 295–300, 2021.

[LWL$^+$20] Shengwen Liang, Ying Wang, Cheng Liu, Lei He, LI Huawei, Dawen Xu, and Xiaowei Li. Engn: A high-throughput and energy-efficient accelerator for large graph neural networks. *IEEE Transactions on Computers*, 70(9):1511–1525, 2020.

[MHS08] A Muthuramalingam, S Himavathi, and E Srinivasan. Neural network implementation using fpga: issues and application. *International Journal of Electrical and Computer Engineering*, 2(12):2802–2808, 2008.

[MLS22] Dina G Mahmoud, Vincent Lenders, and Mirjana Stojilović. Electrical-level attacks on cpus, fpgas, and gpus: Survey and implications in the heterogeneous era. *ACM Computing Surveys (CSUR)*, 55(3):1–40, 2022.

[MRSS18] Amir Moradi, Bastian Richter, Tobias Schneider, and François-Xavier Standaert. Leakage detection with the x2-test. *IACR Trans. on Cryptographic Hardware and Embedded Systems*, pages 209–237, 2018.

[MS19] Dina Mahmoud and Mirjana Stojilović. Timing violation induced faults in multi-tenant fpgas. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1745–1750, 2019.

[MTH$^+$21] Shayan Moini, Shanquan Tian, Daniel Holcomb, Jakub Szefer, and Russell Tessier. Remote Power Side-Channel Attacks on BNN Accelerators in FPGAs. *Proceedings -Design, Automation and Test in Europe, DATE*, 2021-February(2):1639–1644, 2021.

[MTMM07] Robert McEvoy, Michael Tunstall, Colin C Murphy, and William P Marnane. Differential power analysis of hmac based on sha-2, and countermeasures. In *International Workshop on Information Security Applications*, pages 317–332. Springer, 2007.

[MVZ$^+$21] Jian Meng, Shreyas Kolala Venkataramanaiah, Chuteng Zhou, Patrick Hansen, Paul Whatmough, and Jae-sun Seo. FixyFPGA: Efficient fpga accelerator for deep neural networks with high element-wise sparsity and without external memory access. In *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, pages 9–16. IEEE, 2021.

[NSS$^+$16] Eriko Nurvitadhi, David Sheffield, Jaewoong Sim, Asit Mishra, Ganesh Venkatesh, and Debbie Marr. Accelerating binarized neural networks: Comparison of fpga, cpu, gpu, and asic. In *2016 International Conference on Field-Programmable Technology (FPT)*, pages 77–84. IEEE, 2016.

[PBR17] Roberta Piscitelli, Shivam Bhasin, and Francesco Regazzoni. Fault attacks, injection techniques and tools for simulation. In *Hardware security and trust*, pages 27–47. Springer, 2017.

[RBFSG22] Jan Richter-Brockmann, Jakob Feldtkeller, Pascal Sasdrich, and Tim Güneysu. VERICA - verification of combined attacks: Automated formal verification of security against simultaneous information leakage and tampering. Cryptology ePrint Archive, Paper 2022/484, 2022.

[RBK19] Siavash Rezaei, Eli Bozorgzadeh, and Kanghee Kim. Ultrashare: Fpga-based dynamic accelerator sharing and allocation. In *2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–5. IEEE, 2019.

[RBN$^+$15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In *Annual Cryptology Conference*, pages 764–783. Springer, 2015.

[RBSG22] Jan Richter-Brockmann, Pascal Sasdrich, and Tim Guneysu. Revisiting fault adversary models–hardware faults in theory and practice. *IEEE Transactions on Computers*, 2022.

[RBSS+21] Jan Richter-Brockmann, Aein Rezaei Shahmirzadi, Pascal Sasdrich, Amir Moradi, and Tim Güneysu. Fiver–robust verification of countermeasures against fault injections. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 447–473, 2021.

[RHCM+16] Siavash Rezaei, Cesar-Alejandro Hernandez-Calderon, Saeed Mirzamohammadi, Eli Bozorgzadeh, Alexander Veidenbaum, Alex Nicolau, and Michael J Prather. Data-rate-aware fpga-based acceleration framework for streaming applications. In *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–6. IEEE, 2016.

[RHL+18] Zhenyuan Ruan, Tong He, Bojie Li, Peipei Zhou, and Jason Cong. St-accel: A high-level programming platform for streaming applications on fpga. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 9–16. IEEE, 2018.

[RPD+18] Chethan Ramesh, Shivukumar B Patil, Siva Nishok Dhanuskodi, George Provelengios, Sébastien Pillement, Daniel Holcomb, and Russell Tessier. Fpga side channel attacks without physical access. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 45–52. IEEE, 2018.

[SFM17] Yongming Shen, Michael Ferdman, and Peter Milder. Maximizing cnn accelerator efficiency through resource partitioning. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 535–547. IEEE, 2017.

[SGMT18] Falk Schellenberg, Dennis RE Gnad, Amir Moradi, and Mehdi B Tahoori. An inside job: Remote power analysis attacks on fpgas. In *2018 Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, pages 1111–1116. IEEE, 2018.

[SGS10] John E Stone, David Gohara, and Guochun Shi. Opencl: A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering*, 12(3):66, 2010.

[Sko09] Sergei Skorobogatov. Local heating attacks on flash memory devices. In *2009 IEEE International Workshop on Hardware-Oriented Security and Trust*, pages 1–6, 2009.

[SSEM18] Ahmad Shawahna, Sadiq M Sait, and Aiman El-Maleh. Fpga-based accelerators of deep learning networks for learning and classification: A review. *ieee Access*, 7:7823–7859, 2018.

[Syn20] Synopsys. v2020.09-sp4. `[Online]https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html` [Accessed: Jan.11, 2023], 2020.

[TBN+07] Russell Tessier, Vaughn Betz, David Neto, Aaron Egier, and Thiagaraja Gopalsamy. Power-efficient ram mapping algorithms for fpga embedded memory blocks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(2):278–290, 2007.

[TBNG06] Russell Tessier, Vaughn Betz, David Neto, and Thiagaraja Gopalsamy. Power-aware ram mapping for fpga embedded memory blocks. In *Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays*, pages 189–198, 2006.

[TG22] Shahin Tajik and Fatemeh Ganji. Artificial neural networks and fault injection attacks. In *Security and Artificial Intelligence*, pages 72–84. Springer, 2022.

[TMW+21] Shanquan Tian, Shayan Moini, Adam Wolnikowski, Daniel Holcomb, Russell Tessier, and Jakub Szefer. Remote Power Attacks on the Versatile Tensor Accelerator in Multi-Tenant FPGAs. *Proceedings - 29th IEEE International Symposium on Field-Programmable Custom Computing Machines, FCCM 2021*, pages 242–246, 2021.

[USA22] USA Today. Are us data centers fueling climate change? the best (and worst) regions for clean energy. [Online] https://www.usatoday.com/story/tech/2022/08/24/climate-change-data-center-oil-gas-wind/7875280001/?gnt-cfr=1 [Accessed: Jan.11, 2023], 2022.

[WGY+16] Chao Wang, Lei Gong, Qi Yu, Xi Li, Yuan Xie, and Xuehai Zhou. Dlau: A scalable deep learning accelerator unit on fpga. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(3):513–517, 2016.

[XAQ21] Qian Xu, Md Tanvir Arafin, and Gang Qu. Security of neural networks from hardware perspective: A survey and beyond. In *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 449–454. IEEE, 2021.

[XCC+20] Yun Xiang, Zhuangzhi Chen, Zuohui Chen, Zebin Fang, Haiyang Hao, Jinyin Chen, Yi Liu, Zhefu Wu, Qi Xuan, and Xiaoniu Yang. Open dnn box by power side-channel attack. *IEEE Trans. on Circuits and Systems II: Express Briefs*, 67(11):2717–2721, 2020.

[Xil21] Inc. Xilinx. v2021.1. [Online]https://www.xilinx.com/products/design-tools/vivado.html [Accessed: Jan.11, 2023], 2021.

[YKO+20] Kota Yoshida, Takaya Kubota, Shunsuke Okura, Mitsuru Shiozaki, and Takeshi Fujino. Model reverse-engineering attack using correlation power analysis against systolic array based neural network accelerator. In *2020 IEEE Intrl. Symp. on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2020.

[YMY+20] Honggang Yu, Haocheng Ma, Kaichen Yang, Yiqiang Zhao, and Yier Jin. Deepem: Deep neural networks model recovery through em side-channel information leakage. In *2020 IEEE Intrl. Symp. on Hardware Oriented Security and Trust (HOST)*, pages 209–218. IEEE, 2020.

[ZJ15] Yongmei Zhou and Jingfei Jiang. An fpga-based accelerator implementation for deep convolutional neural networks. In *2015 4th International Conference on Computer Science and Network Technology (ICCSNT)*, volume 1, pages 829–832. IEEE, 2015.

[ZLS+15] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays*, pages 161–170, 2015.

[ZP17] Chi Zhang and Viktor Prasanna. Frequency domain acceleration of convolutional neural networks on cpu-fpga shared memory system. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 35–44, 2017.

[ZS18] Mark Zhao and G Edward Suh. FPGA-based remote power side-channel attacks. In *2018 IEEE Symp. on Security and Privacy (SP)*, pages 229–244. IEEE, 2018.

[ZSZ+17] Ritchie Zhao, Weinan Song, Wentao Zhang, Tianwei Xing, Jeng-Hau Lin, Mani Srivastava, Rajesh Gupta, and Zhiru Zhang. Accelerating binarized convolutional neural networks with software-programmable fpgas. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 15–24, 2017.

[ZYC+21] Yicheng Zhang, Rozhin Yasaei, Hao Chen, Zhou Li, and Mohammad Abdullah Al Faruque. Stealing neural network structure through remote fpga side-channel analysis. *IEEE Transactions on Information Forensics and Security*, 16:4377–4388, 2021.