REDACTOR: eFPGA Redaction for DNN Accelerator Security

Yazan Baddour
Electrical Engineering
California State Uni. Long Beach
Long Beach, CA, USA
Yazan.Baddour01@student.csulb.edu

Ava Hedayatipour
Electrical Engineering
California State Uni. Long Beach
Long Beach, CA, USA
Ava.Hedayatipour@csulb.edu

Amin Rezaei
Computer Engineering & Computer Science
California State Uni. Long Beach
Long Beach, CA, USA
Amin.Rezaei@csulb.edu

***Abstract*—With the ever-increasing integration of artificial intelligence into daily life and the growing importance of well-trained models, the security of hardware accelerators supporting Deep Neural Networks (DNNs) has become paramount. As a promising solution to prevent hardware intellectual property theft, eFPGA redaction has emerged. This technique selectively conceals critical components of the design, allowing authorized users to restore functionality post-fabrication by inserting the correct bitstream. In this paper, we explore the redaction of DNN accelerators using eFPGAs, from specification to physical design implementation. Specifically, we investigate the selection of critical DNN modules for redaction using both regular and fracturable look-up tables. We perform synthesis, timing verification, and place & route on redacted DNN accelerators. Furthermore, we evaluate the overhead of incorporating eFPGAs into DNN accelerators in terms of power, area, and delay, finding it reasonable given the security benefits.**

***Index Terms*—Hardware Security; Deep Neural Networks; Hardware Accelerators; eFPGA Redaction; Physical Design**

I. Introduction

Deep Neural Network (DNN) technologies continue to evolve, propelled by the quest for enhanced accuracy. Achieving superior accuracy in DNNs necessitates high-performance computing resources, such as hardware accelerators. As accuracy and throughput become increasingly critical, the importance of security for DNN models and accelerators also escalates.

Attackers have the capability to reverse engineer the Intellectual Property (IP) of hardware accelerators, enabling them to create unauthorized substitute models. Additionally, they can tamper with the weights of the DNN, resulting in alterations to accuracy and misclassification of inputs [1]. To address these security concerns, logic locking [2]–[7] has emerged using various types of key gates in which only authorized users can unlock the circuit. However, logic locking faces challenges from SAT-based attacks [8]–[13] that exploit an activated Integrated Circuit (IC) as a reference point along with the leaked locked netlist, allowing them to extract the correct key. In response to these threats, programmable fabrics exemplified by embedded Field-Programmable Gate Arrays (eFPGAs) offer robust defenses against reverse engineering and hardware IP theft [14]–[17]. The configuration bitstream remains accessible solely to the designer or authorized user, even if an untrusted foundry manufactures the design. The challenges in eFPGA redaction encompass identifying which modules within the IP should be redacted and minimizing the delay and area overheads associated with replacing these specific modules with eFPGAs.

To the best of our knowledge, **there is a gap in the existing literature on how eFPGA redaction affects the security of DNN accelerators.** Specifically, there is limited exploration of the implications on timing, area, and power overheads when integrating eFPGA to replace a segment of a hardware accelerator. The contributions of this paper are as follows:

- Proposing an eFPGA redaction flow to hide sensitive components of a DNN accelerator with low overhead;
- Providing guidance on selecting a critical module to be replaced with regular or fracturable LUTs;
- Evaluating the overhead of the proposed redaction method in terms of area, power, and delay as well as the security against oracle-guided SAT-based attacks.

The rest of the paper is organized as follows: Section II discusses the background, preliminaries, and related works. Section III proposes our contributions to redact critical IPs of a DNN accelerator via eFPGAs, followed by verification, synthesis, and place & route steps. Section IV depicts the comprehensive experimental results on the overhead and security of the redacted accelerator. Finally, conclusions are given in Section V.

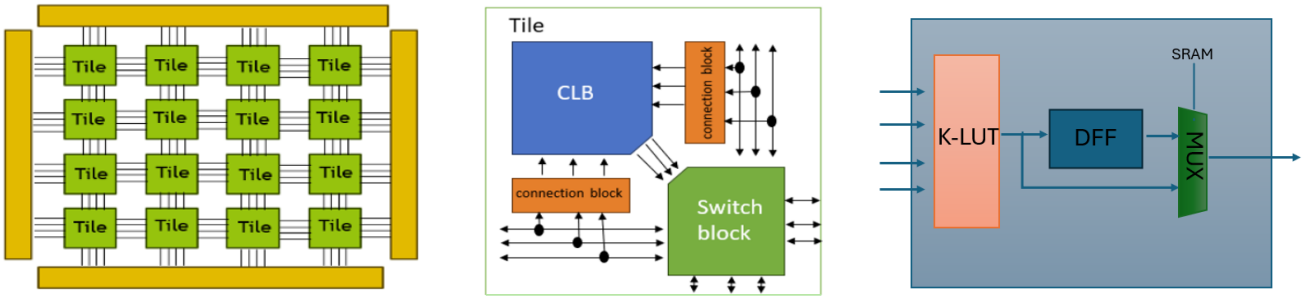
II. Background and Preliminaries

In this section, we discuss the background on DNNs, accelerators, eFPGAs, and existing work on securing hardware accelerators.

A. Deep Neural Networks

DNNs are employed in various tasks such as image classification [18] and speech recognition [19]. The input layer of DNNs receives inputs, and outputs are generated through weighted sums followed by non-linear activation functions. These outputs then proceed to subsequent layers. Initially set randomly, weights and biases are adjusted during the training phase to minimize disparities between expected and actual outputs.

A Convolutional Neural Network (CNN) is a type of DNN comprising four main layers: convolutional, normalization, pooling, and fully connected layers. In the convolutional layer, inputs and outputs are processed as 2D or 3D arrays, with dimensions represented by height, width, and number of channels. The convolutional layer uses 3D filters to calculate



(a) 4x4 fabric (b) Tile components (c) BLE components

Fig. 1: eFPGA architecture

the inner product with sub-arrays of the input, moving a sliding window of the filter size across the input with a fixed stride. Normalization layers adjust activation levels within each feature map, aiding in stabilizing the learning process by keeping inputs within a reasonable range. Pooling layers in CNNs down-sample input feature maps to create lower-resolution versions, retaining essential information while excluding irrelevant details. Operations such as average pooling and max pooling are applied independently to each feature map, reducing dimensions while maintaining the number of channels. Fully connected layers in CNNs determine class scores by processing outputs from preceding layers. Activation functions like Rectified Linear Unit (ReLU) introduce non-linearity.

B. Hardware Accelerators

DNN models demand substantial computational resources, with tasks such as image classification requiring millions of weights and over half a billion Multiply And Accumulate (MAC) operations, necessitating specialized accelerators for real-time execution. DNN architectures require Arithmetic Logical Units (ALUs) [20], [21] that focus on MAC units for computational resources and flexibility, dataflow optimization [22], [23] to minimize energy consumption, and sparsity [24] to skip zero multiplications and reduce power consumption.

Eyeriss [25] is a Row-Stationary (RS) dataflow-based accelerator designed to minimize data movement energy in spatial architectures. In the RS dataflow, a row of operands is stored in the Register File (RF). Eyeriss employs diagonal connections of Processing Elements (PEs) for input reuse and vertical accumulation of partial sums. Each PE includes local registers for storing at least one row of weights and activations, along with a MAC unit and controller responsible for the temporal reuse of MAC units in 1D convolution. A significant portion of the RF is allocated to weights, and input vectors are reused to calculate partial sums for multiple output feature maps.

C. Embedded Field Programmable Gate Arrays

Modern eFPGA architectures adopt a tile-based structure, with each tile containing configurable logic resources. Surrounding these tiles are the I/O blocks, which facilitate communication between the eFPGA and external devices or systems. Each tile within the eFPGA architecture comprises two fundamental elements that collectively enable its programmability and functionality. The first component is the Configurable Logic Block (CLB), which serves as the primary unit for implementing logic functions and user-defined designs. The second component is programmable routing, which facilitates the interconnection of various CLBs and enables the flow of data between them. Within each CLB, there is a Basic Logic Element (BLE) that incorporates a K-input Look-Up Table (LUT) capable of mapping a K-input single-output Boolean function, alongside a flip-flop and a 2-1 routing multiplexer used to toggle between sequential and combinational logic.

Fig. 1 provides an example of a 4x4 eFPGA architecture, the components inside each tile, and a visual representation of a basic BLE setup with a 4-input LUT. Studies have identified the optimal LUT size for balancing area and delay to be between 4 to 6, with 6-LUTs exhibiting superior performance and 4-LUTs occupying the smallest area [26]. To address these trade-offs and enhance LUT utilization, a refined version known as fracturable LUTs (FLUTs) has been introduced. FLUTs have more than one mode of operation: they can function as a K-input LUT or be fractured into smaller K-1 LUTs, with shared inputs [27]. Within the CLB, BLEs can be organized into logic clusters to optimize eFPGA speed and area efficiency. The ideal cluster size (N) is typically determined to be between 4 and 10 [28]. Crossbar routing interconnects the inputs and outputs of CLBs and BLEs using a series of programmable multiplexers, ensuring connectivity between all BLEs and every CLB pin. Switch Blocks (SBs) and Connection Blocks (CBs) serve as the global routing that connects CLBs together.

D. Related Works

With the recent boom in Artificial Intelligence (AI), developing new approaches to safeguard the DNN models and accelerators is of the utmost importance. In [29], fault injection attacks are examined that are capable of inducing misclassification in DNN by altering the bias in the output layer to favor the desired adversarial class. As a countermeasure in [30] the fault-sensitivity of individual neurons within a given DNN is

measured by effectively leveraging both external and internal redundancy within DNN models to balance system robustness against hardware overhead.

In addition, a Hardware Protected Neural Network (HPNN) framework is proposed [31] to safeguard DNNs from attackers with extensive knowledge. It conceals the weight space through a confidential HPNN key, controlling each neuron's functionality. In [32] a hardware key is introduced to secure the accelerator such that a wrong key increases memory access, inference time, and energy consumption, making it unsuitable for inference. This method requires modifying the activation function to prevent bypassing the hardware key controlled block. Moreover, they use a model key to obscure the model without the need of retraining the model. Although this approach resulted in decreased accuracy and higher memory access when incorrect keys were applied, it necessitates modifications in the DNN accelerator.

Initial logic locking solutions utilize XOR-based and MUX-based mechanisms [33], [34]. However, the oracle-guided SAT attack [8] exposes vulnerabilities in these methods, prompting the development of more robust techniques [3], [5], [6], [35]–[40], which increase the time complexity of SAT attacks.

Furthermore, a soft embedded FPGA redaction method is introduced [17] to hide critical IP functionality and routing within RTL designs. A critical IP in the Verilog file is identified and synthesized using Yosys [41], then place & route is performed to determine the smallest eFPGA fabric needed. The resulting fabric, devoid of critical IP information, is loaded with a bitstream to maintain functionality. In [42] the effects of varying parameters (K and N) on area, power, delay, and security of eFPGA architectures are compared. Increasing K influences CLB inputs, impacting LUT sizes and routing, while increasing N adds BLEs, affecting area and routing complexities. Findings challenge the assumption that fabric size directly correlates with security strength.

In [11], the assumption that eFPGA-based designs are inherently secure against oracle-guided attacks has been challenged. The researchers conducted two attacks: CycSAT [9], which aims to break cycles within the circuit, and IcySAT [10]. IcySAT took longer due to the unrolling process. However, CycSAT struggles to break hard loops, leaving at least one loop intact and causing the SAT solver to repeat iterations. To address this, the paper developed a two-phase attack called Break & Unroll. The first phase breaks cycles sequentially, creating a new non-cyclic constraint. If hard cycles persist, the second phase unrolls the circuit, duplicating gates and breaking feedback connections to prevent infinite loops.

III. REDACTOR

In this section, we propose **REDACTOR**, an eFPGA **RED**action framework for CNN **AC**celera**TOR**s shown in Fig. 2. Without loss of generality, we utilize the accelerator outlined in [43], which is a modification of the Broadcast, Stay, Migration (BSM) dataflow initially proposed in [44]. Although the procedure is independent of the chosen accelerator, this adaptation is considered energy-efficient as it effectively minimizes redundant access to off-chip memory.

TABLE I: Critical IPs redacted

Critical IP	# of Modules	# of I/Os	Description
OWMC	5	20	Controls the flow of DNN weights
MUXDC	4	15	Controls dataflow between PEs
OMDC	20	26	Controls the convolution process
PEDC	1	5	Sets/resets output register of PEs

A. Critical IP Selection and Fabric Generation

The goal of module selection is to choose a module that plays a crucial role in the overall functionality of the accelerator. In this regard, three options can be considered:

- Opting for a module that significantly impacts the outputs proves beneficial for causing output corruption when an unauthorized user loads the wrong bitstream.
- Selecting a module that causes error propagation throughout the system can amplify the impact of using incorrect bitstreams and decrease the accuracy of the DNN model.
- Choosing a module that can be implemented using a complex fabric with a large unroll factor can be beneficial for securing against existing attacks that target eFPGAs while ensuring it remains within the power, area, and delay budget.

The first critical IP selected is the On-Chip Weight Memory Controller (OWMC), which has 12 outputs, some of which directly and indirectly impact the loading of DNN weights from the weight memory to the data flow block comprising numerous Processing Elements (PEs). Protecting these weights ensures that sensitive information remains secure from unauthorized access or corruption. Subsequently, we iterate through the redaction process to redact several IPs in the accelerator using eFPGAs of varying sizes and architectures. These IPs include the Multiplexers Dataflow Controller (MUXDC), the On-Chip Memory Dataflow Controller (OMDC), and the Processing Elements Dataflow Controller (PEDC). Table I displays the characteristics of each critical IP.

To generate the eFPGA fabrics, we use OpenFPGA [45]. We input three files into OpenFPGA: the benchmark file, which contains the critical RTL Verilog portion of the selected module for redaction, and two architecture files. One is the OpenFPGA architecture file [46], and the other is the VPR architecture file [47]. Both files are in Extensible Markup Language (XML) format and together specify the architecture of the eFPGA.

We select a fabric architecture to achieve 100% block utilization and over 90% I/O utilization. This is accomplished by manually adjusting the number of BLEs (N) per CLB and the number of I/O pins per tile until the target utilization is reached, while maintaining a constant 4-input LUT/FLUT (K=4). The number of CLB inputs can be determined using

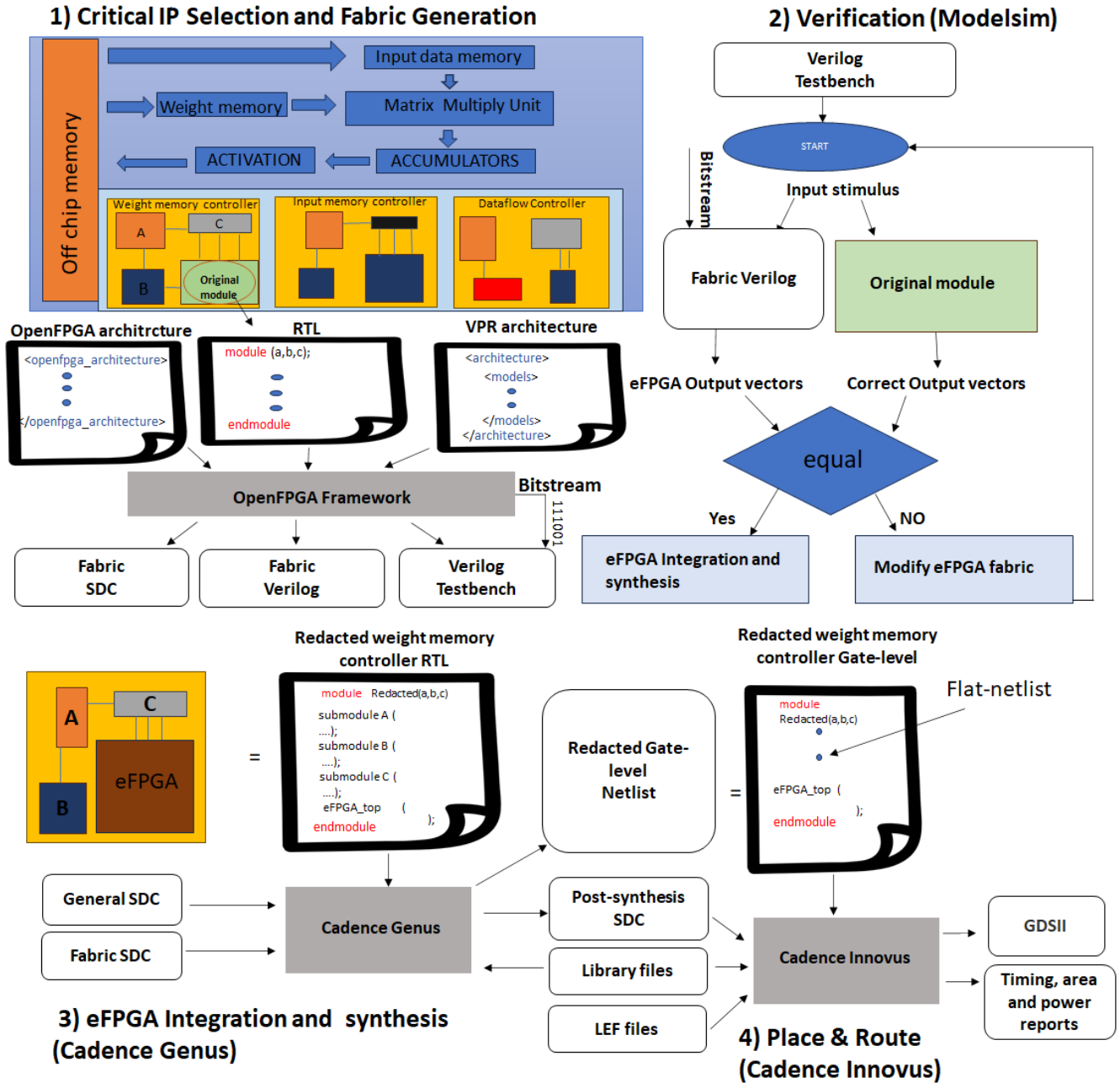


Fig. 2: The proposed eFPGA redaction flow for DNN accelerators

the following formula, known to provide favorable Power, Performance, and Area (PPA) characteristics [48].

$$I = \frac{K(N+1)}{2} \tag{1}$$

Next, we acquire the timing constraints (i.e., SDC files) for the fabric intended for use in the physical design phase. Additionally, we obtain the Verilog files for the fabric, along with a testbench designed to verify the functionality of the fabric and the bitstream necessary to program the fabric to operate as the chosen module for redaction. Table II presents the eFPGA parameters used.

B. Verification

We utilize ModelSim to simulate the functionality of the fabric. Both the original design and the eFPGA fabric receive shared input stimuli, and the correct bitstream is loaded into the fabric for evaluation. Subsequently, the output vectors of the eFPGA and the original design are compared. When the correct bitstream is applied, the output vectors should match, ensuring consistency and functionality between the eFPGA

TABLE II: eFPGA parameters used

Parameter	Value	Description
K	4	Input size of a LUT/FLUT
N	[1,9]	Number of BLEs per CLB
W	Auto	Number of routing tracks in a channel
Fc_{in}	0.15	Fraction of the routing tracks that a CLB input can connect
Fc_{out}	0.1	Fraction of the routing tracks that a CLB output can connect
Fs	3	Number of connections per incoming routing track in a switch block
L	4	The length of routing track (number of CLBs spanned)

and the original design. In Fig. 3, the waveforms depict the output vectors. The eFPGA output signals are visualized in green, while the original module's signals are represented in yellow. The verification process is repeated for the synthesized fabric to verify functionality after synthesis.

C. eFPGA Integration and Synthesis

We utilize Cadence Genus for synthesizing each redacted module. The redacted module serves as the top module in Cadence Genus for synthesis, where we replace the critical Verilog portion with the eFPGA at the RTL level. To map the Verilog netlist to a gate-level netlist, we employ the standard cell library file available on the Cadence website (i.e., slow_vdd1v2_basicCells.lib), which is a 45nm technology library characterized by slower operating speeds or slower process corners.

First, we read the Verilog and library files and elaborate the top module. Then, we analyze all the SDC files generated by OpenFPGA and command Genus to report timing. At this stage, Genus addresses any combinational loops that legally exist within the eFPGA fabric by inserting a buffer from the technology library onto the feedback loop (i.e., cdn_loop_breaker_cell). Additionally, it disables the timing arc from the input to the output, effectively breaking the timing loop. Fig. 4 shows an example of a loop that is broken. The buffer highlighted in yellow represents the feedback loop.

We choose to use a flattened netlist to simplify the place and route stage. However, during the process of flattening the

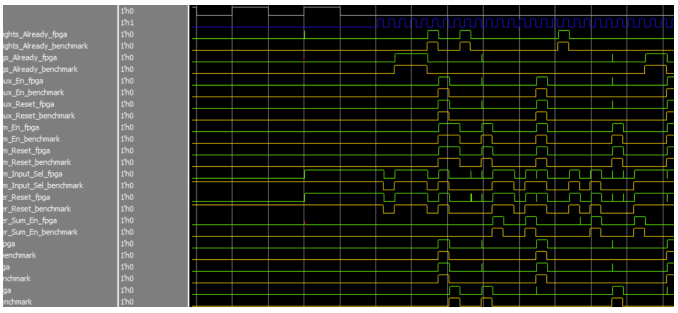


Fig. 3: eFPGA fabric and original module output vectors

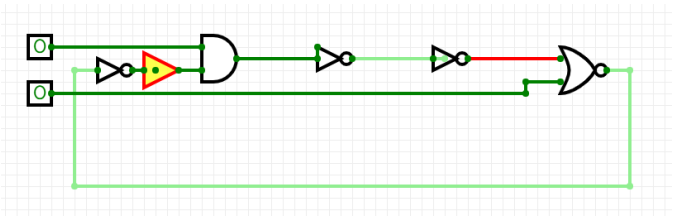


Fig. 4: Loop breaker cell example

netlists, Cadence Genus modifies the names of hierarchical instances, which results in the disregard of some initial SDC constraints. To address this issue, we instruct Genus to parse an additional SDC file, which contains instructions to disable timing for specific paths using the flattened names.

Finally, optimization is performed on the flattened netlist, where we instruct the tool to attempt optimization on all paths with negative slack, including the critical path. Reports are then generated to provide details on area, power, and timing. Additionally, the gate-level netlist Verilog file and a single SDC file containing all constraints are generated. These two files serve as inputs for the subsequent place & route stage. Genus does not incorporate the constraints necessary to break the timing loops in the final SDC file. Therefore, we manually modify the file to include these constraints.

D. Place & Route

The place & route stage in IC design is a critical process where logic gates, flip-flops, and other components undergo positioning (i.e., placement) and interconnection (i.e., routing). We utilize Cadence Innovus for the place & route of the redacted modules. Cadence Innovus takes inputs including the gate-level netlist and the SDC constraint file generated by Cadence Genus, along with technology files such as timing library files (.lib) and library exchange format (.lef) files.

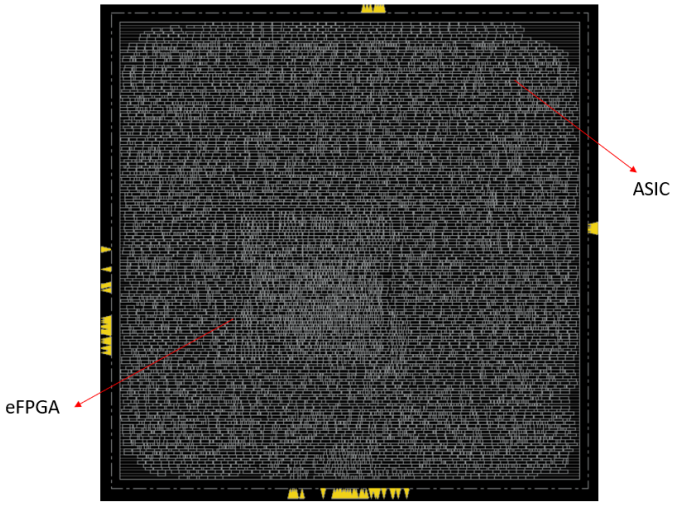


Fig. 5: Redacted module layout without metal layers

TABLE III: eFPGA fabric characteristics

eFPGA Fabric	Block Utilization	I/O utilization	Bitstream Size	Channel Width
2x2 K4N2	100%	94%	614	18
2x2 K4_frac_N1	100%	94%	458	18
1x1 K4N6	100%	100%	440	26
1x1 K4_frac_N3	100%	100%	256	18
2x2 K4N4	100%	95%	1160	30
2x2 K4_frac_N3	100%	95%	1059	30
1x1 K4N1	100%	100%	66	6
1x1 K4_frac_N1	100%	100%	79	14

The process begins with specifying the floor plan, followed by power planning, which involves inserting VDD and VSS rings, and power and ground stripes to connect them to standard cells. Standard cells are then placed within the specified floor plan. Additionally, we instruct Innovus to place the design with I/O pins, eliminating the need for manual I/O file creation. Subsequently, pre-Clock Tree Synthesis (CTS) optimization is performed to meet all timing constraints before CTS. We then perform CTS, followed by post-CTS optimization to further refine the design to meet timing constraints. Finally, routing and post-route optimization are carried out to complete the process, and timing, area, and power reports are generated. Fig. 5 shows the layout of the redacted module without metal layers. The eFPGA is noticeable in the middle, where it exhibits a slightly different pattern.

IV. Experimental Results

In this section, we evaluate the overhead and security of **REDACTOR**. The source codes, along with the created eFPGA fabrics, are available on our GitHub repository[1].

A. Overhead Analysis

As mentioned earlier, our goal is to find the simplest fabric using the parameters shown in Table II. We achieve this by increasing or decreasing the number of BLEs in each CLB and I/O pins in I/O tiles. The characteristics of the eFPGA fabrics utilized in the experiments are summarized in Table III. Fig. 6 compares the area, power, and delay overheads of LUT-based eFPGA and FLUT-based eFPGA for the redacted modules normalized according to the original designs.

OWMC: We redact the OWMC IP using two fabrics: The first one is a LUT-based fabric with 2 BLEs/CLB (2x2 K4N2). The second one is a FLUT-based fabric with 1 BLE/CLB (2x2 K4_frac_N1). Despite the expected larger bitstream for the FLUT-based fabric, it actually has a smaller size because of fewer BLEs/CLB. Both fabrics fully utilize blocks and I/O but have different bitstream sizes: 614 for the LUT-based and 458 for the FLUT-based. Despite FLUT's anticipated impact on channel width, it remains the same at 18 for both fabrics due to fewer BLEs. The LUT-based fabric introduces higher area, power, and delay overheads (33%, 16%, and 27% respectively) compared to the FLUT-based fabric (30%, 15%, and 24.5% respectively). *Opting for FLUT-based is preferable when integrating eFPGA for the OWMC IP.*

MUXDC: We test the MUXDC IP using two fabric configurations: a LUT-based with 6 BLEs/CLB (1x1 K4N6) and a FLUT-based with 3 BLEs/CLB using FLUT (1x1 K4_frac_N3). Both fabrics achieve full block and I/O utilization. The LUT-based has a larger bitstream size of 440 compared to the FLUT-based, which has 256 bits. Routing widths differ, with the LUT-based at 26 and the FLUT-based at 18 tracks per channel. Due to this variance, the area difference between the fabrics is notable. The FLUT-based shows significant improvement in power consumption and a slightly better critical path delay. The first fabric introduces higher area, power, and delay overheads (192%, 139%, and 2% respectively) compared to the second fabric (107%, 49%, and 1.6% respectively). *For the MUXDC IP, opting for FLUT proves preferable for redaction.*

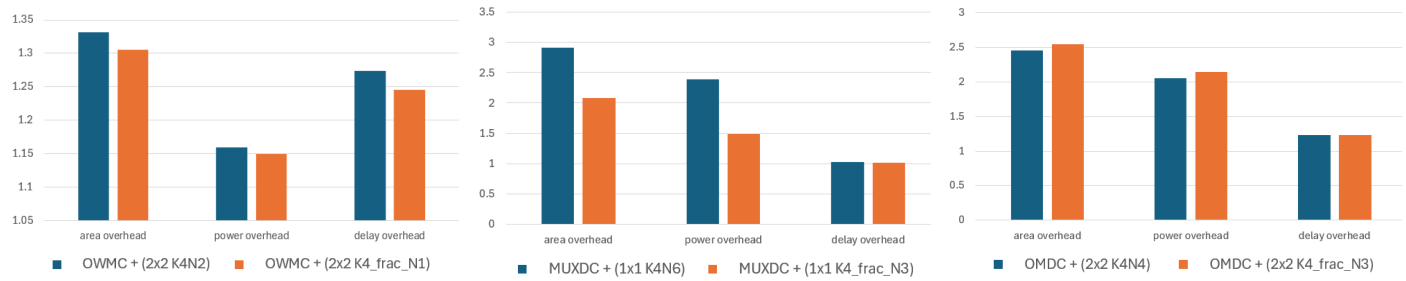
OMDC: In addition, the OMDC IP is tested using two fabrics: a LUT-based (2x2 K4N4) and a FLUT-based (2x2 K4_frac_N3), both achieving full block utilization and 95% I/O utilization. Despite the LUT-based fabric having a larger bitstream size of 1160 compared to the second fabric's 1059, the fabric utilizing regular LUTs shows slightly better performance in area, power, and delay, even though both fabrics routed with a channel width of 30. The LUT-based introduces an area overhead of 145%, a power overhead of 105%, and a critical path delay overhead of 23%, while the FLUT-based introduces an area overhead of 154%, a power overhead of 114%, and a critical path delay overhead of 23.6%. *For the OMDC IP, choosing a regular LUT is preferable.*

PEDC: Finally, the PEDC IP is evaluated with two fabrics: a basic LUT-based with one CLB and one BLE (1x1 K4N1), and an identical fabric with FLUT instead (K4_frac_N1). Both fabrics utilize I/Os and resources fully. The LUT-based fabric needs 66 bits for programming, while the FLUT-based one requires 79. The LUT-based fabric has a minimum channel width of 6, while the FLUT-based one needs a wider channel width of 14 due to its more complex routing. Because the original PEDC IP is a very small finite state machine, both the LUT-based and FLUT-based fabrics shows huge area and power overhead as well as high delay compared with the original IP. *In the case of PEDC IP, choosing the fabric with regular LUTs is preferable. However, it is not efficient to replace a small IP with eFPGA due to the overheads introduced.*

B. Security Analysis

We conduct a separate synthesis of the eFPGA fabric, this time constraining Genus to utilize only basic logic cells. Using Python, we convert the Verilog gate-level netlist from Genus to a *.bench* file. To establish an oracle, we program the fabric with the bitstream generated by OpenFPGA, assigning the outputs of the DFFs found in the scan chain with their corresponding bits. For creating a key-controlled netlist, we

[1]https://github.com/cars-lab-repo/REDACTOR



(a) Redacted OWMC (b) Redacted MUXDC (c) Redacted OMDC

Fig. 6: Normalized area, power, and delay overhead of LUT-based vs. FLUT-based eFPGAs redacted modules

TABLE IV: Security results

Fabric	Unroll	# Clauses	Time (s)	Key Reported?
2x2 K4N2	59	1610318	99	yes
2x2 K4_frac_N1	30	559536	7	yes
1x1 K4N6	36	875500	51	yes
1x1 K4_frac_N3	2	9291	0.26	yes
2x2 K4N4	112	N/A	Time out	no
2x2 K4_frac_N3	59	3299375	81	yes
1x1 K4N1	5	4189	0.1	yes
1x1 K4_frac_N1	2	2384	0.08	yes

expose the outputs of the DFFs in the scan chain as key inputs. Then, we utilized the break and unroll attack found in [11], which is then fed to NEOS [49] tool for extracting the bitstream.

The security results are presented in Table IV. We were able to extract the bitstream of all eFPGAs that used FLUTs in a shorter time compared to eFPGAs that used regular LUTs. This is because the LUT-based fabrics have a higher unroll factor. For instance, our 2x2 K4N4 fabric timed out after 6 hours of running the attack, which has an unroll factor of 112. To maintain the reduced overheads of FLUT-based eFPGA fabrics, we propose two methods for future research:

- Introducing more cycles within the eFPGA fabric, thereby increasing the unroll factor to the point where the attack times out.
- Introducing non-unrollable cycles within the eFPGA fabric, which consist of oscillating and stateful cycles [11] interlaced in a manner that ensures at least one cycle remains unbroken during a cyclic attack, causing it to fail by entering an infinite loop.

V. Conclusion

In this paper, we explored the importance of securing DNN accelerators and proposed an approach for redacting critical IPs with eFPGAs, from specification to physical design. Specifically, we focused on evaluating the impact of eFPGA fabrics with high I/O and block utilization and assessed the integration of these nearly fully utilized fabrics with regular LUTs and FLUTs.

While using FLUT-based eFPGAs can complicate routing, they offer an advantage by reducing the number of BLEs or CLBs, which generally translates to lower power, delay, and area overhead. As demonstrated by the experiments, the choice between LUT or FLUT depends on the specific IP to be redacted, and no general rule can be established. Additionally, we observed that FLUT-based eFPGAs have lower unroll factors, making it easier for attackers to extract the bitstream. Conversely, when we redacted the OWMC IP with a regular LUT-based eFPGA, the attack timed out, and we were able to maintain reasonable overhead while enhancing security.

For future research, systematically increasing the unrolling factor or adding non-unrollable cycles to eFPGA fabrics can be pursued to maintain low overhead while improving security.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Award No. 2245247.

References

[1] Sparsh Mittal, Himanshi Gupta, and Srishti Srivastava. A survey on hardware security of dnn models and accelerators. *Journal of Systems Architecture*, 117:102163, 2021.

[2] Jeyavijayan Rajendran, Youngok Pino, Ozgur Sinanoglu, and Ramesh Karri. Security analysis of logic obfuscation. In *Design Automation Conference (DAC)*, pages 83–89, 2012.

[3] Amin Rezaei and Hai Zhou. Sequential logic encryption against model checking attack. In *Design Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1178–1181, 2021.

[4] Jordan Maynard and Amin Rezaei. Dk lock: Dual key logic locking against oracle-guided attacks. In *International Symposium on Quality Electronic Design (ISQED)*, pages 1–7, 2023.

[5] Raheel Afsharmazayejani, Hossein Sayadi, and Amin Rezaei. Distributed logic encryption: Essential security requirements and low-overhead implementation. In *Proceedings of Great Lakes Symposium on VLSI (GLSVLSI)*, pages 127–131, 2022.

[6] Yeganeh Aghamohammadi and Amin Rezaei. Cola: Convolutional neural network model for secure low overhead logic locking assignment. In *Great Lakes Symposium on VLSI 2023 (GLSVLSI)*, pages 339–344, 2023.

[7] Kevin Lopez and Amin Rezaei. K-gate lock: Multi-key logic locking using input encoding against oracle-guided attacks. In *30th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2025.

[8] Pramod Subramanyan, Sayak Ray, and Sharad Malik. Evaluating the security of logic encryption algorithms. In *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 137–143, 2015.

[9] Hai Zhou, Ruifeng Jiang, and Shuyu Kong. Cycsat: Sat-based attack on cyclic logic encryptions. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 49–56, 2017.

[10] Kaveh Shamsi, David Z. Pan, and Yier Jin. Icysat: Improved sat-based attacks on cyclic locked circuits. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–7, 2019.

[11] Amin Rezaei, Raheel Afsharmazayejani, and Jordan Maynard. Evaluating the security of efpga-based redaction algorithms. In *2022 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–7, 2022.

[12] Yuanqi Shen, You Li, Shuyu Kong, Amin Rezaei, and Hai Zhou. Sigattack: New high-level sat-based attack on logic encryptions. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 940–943, 2019.

[13] Amin Rezaei, Ava Hedayatipour, Hossein Sayadi, Mehrdad Aliasgari, and Hai Zhou. Global attack and remedy on ic-specific logic encryption. In *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 145–148, 2022.

[14] Mustafa M. Shihab, Jingxiang Tian, Gaurav Rajavendra Reddy, Bo Hu, William Swartz, Benjamin Carrion Schaefer, Carl Sechen, and Yiorgos Makris. Design obfuscation through selective post-fabrication transistor-level programming. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 528–533, 2019.

[15] Jianqi Chen and Benjamin Carrion Schafer. Area efficient functional locking through coarse grained runtime reconfigurable architectures. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 542–547, 2021.

[16] Jitendra Bhandari, Abdul Khader Thalakkattu Moosa, Benjamin Tan, Christian Pilato, Ganesh Gore, Xifan Tang, Scott Temple, Pierre-Emmanuel Gaillardon, and Ramesh Karri. Exploring efpga-based redaction for ip protection. In *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9, 2021.

[17] Prashanth Mohan, Oguz Atli, Joseph Sweeney, Onur Kibar, Larry Pileggi, and Ken Mai. Hardware redaction via designer-directed fine-grained efpga insertion. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1186–1191, 2021.

[18] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009.

[19] Liang Lu and Steve Renals. Small-footprint highway deep neural networks for speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(7):1502–1511, 2017.

[20] 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 *ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, page 92–104, 2015.

[21] Daofu Liu, Tianshi Chen, Shaoli Liu, Jinhong Zhou, Shengyuan Zhou, Olivier Teman, Xiaobing Feng, Xuehai Zhou, and Yunji Chen. Pudiannao: A polyvalent machine learning accelerator. In *Proceedings International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, page 369–381, 2015.

[22] Norman Jouppi, Cliff Young, Nishant Patil, and David Patterson. Motivation for and evaluation of the first tensor processing unit. *IEEE Micro*, 38(3):10–19, 2018.

[23] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W. Keckler, and William J. Dally. Scnn: An accelerator for compressed-sparse convolutional neural networks. In *ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*, pages 27–40, 2017.

[24] Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos. Cnvlutin: Ineffectual-neuron-free deep neural network computing. In *Proceedings of International Symposium on Computer Architecture (ISCA)*, page 1–13. IEEE Press, 2016.

[25] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*, pages 367–379, 2016.

[26] Satwant Singh, Jonathan Rose, Paul Chow, and David Lewis. The effect of logic block architecture on fpga performance. *IEEE Journal of Solid-State Circuits*, 27(3):281–287, 1992.

[27] David Dickin and Lesley Shannon. Exploring fpga technology mapping for fracturable lut minimization. In *International Conference on Field-Programmable Technology (FPT)*, pages 1–8, 2011.

[28] Vaughn Betz and Jonathan Rose. Cluster-based logic blocks for fpgas: area-efficiency vs. input sharing and size. In *Proceedings of Custom Integrated Circuits Conference (CICC)*, pages 551–554, 1997.

[29] Yannan Liu, Lingxiao Wei, Bo Luo, and Qiang Xu. Fault injection attack on deep neural network. In *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, page 131–138. IEEE Press, 2017.

[30] Yu Li, Yannan Liu, Min Li, Ye Tian, Bo Luo, and Qiang Xu. D2nn: A fine-grained dual modular redundancy framework for deep neural networks. In *Proceedings of Annual Computer Security Applications Conference (ACSAC)*, page 138–147, 2019.

[31] Abhishek Chakraborty, Ankit Mondai, and Ankur Srivastava. Hardware-assisted intellectual property protection of deep learning models. In *ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2020.

[32] Jingbo Zhou and Xinmiao Zhang. Joint protection scheme for deep neural network hardware accelerators and models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(12):4518–4527, 2023.

[33] Jarrod A. Roy, Farinaz Koushanfar, and Igor L. Markov. Ending piracy of integrated circuits. *Computer*, 43(10):30–38, 2010.

[34] Jeyavijayan Rajendran, Huan Zhang, Cheng Zhang, Garrett S. Rose, Yier Pino, Ozgur Sinanoglu, et al. Fault analysis-based logic encryption. *IEEE Transactions on Computers*, pages 410–424, 2013.

[35] Yang Xie and Ankur Srivastava. Anti-sat: Mitigating sat attack on logic locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(2):199–207, 2019.

[36] Muhammad Yasin, Bodhisatwa Mazumdar, Jeyavijayan J V Rajendran, and Ozgur Sinanoglu. Sarlock: Sat attack resistant logic locking. In *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 236–241, 2016.

[37] Muhammad Yasin, Ozgur Sinanoglu, and Ramesh Karri. Provably-secure logic locking: From theory to practice. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1601–1618. ACM, 2017.

[38] Hai Zhou, Yuanqi Shen, and Amin Rezaei. Vulnerability and remedy of stripped function logic locking. Cryptology ePrint Archive, Paper 2019/139, 2019.

[39] Yeganeh Aghamohammadi and Amin Rezaei. Machine learning-based security evaluation and overhead analysis of logic locking. *Journal of Hardware and Systems Security*, 8:25–43, 2024.

[40] Amin Rezaei, Yuanqi Shen, Shuyu Kong, Jie Gu, and Hai Zhou. Cyclic locking and memristor-based obfuscation against cycsat and inside foundry attacks. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 85–90, 2018.

[41] Yosys open synthesis suite. https://yosyshq.net/yosys/, 2013.

[42] Jitendra Bhandari, Abdul Khader Thalakkattu Moosa, Benjamin Tan, Christian Pilato, Ganesh Gore, Xifan Tang, Scott Temple, Pierre-Emmanuel Gaillardon, and Ramesh Karri. Not all fabrics are created equal: Exploring efpga parameters for ip redaction. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 31(10):1459–1471, 2023.

[43] Cnn-accelerator. https://github.com/8krisv/CNN-ACCELERATOR/, 2021.

[44] Jihyuck Jo, Suchang Kim, and In-Cheol Park. Energy-efficient convolution architecture based on eescheduled dataflow. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 65(12):4196–4207, 2018.

[45] Xifan Tang, Edouard Giacomin, Baudouin Chauviere, Aurelien Alacchi, and Pierre-Emmanuel Gaillardon. Openfpga: An open-source framework for agile prototyping customizable fpgas. *IEEE Micro*, 40(4):41–48, 2020.

[46] Xifan Tang, Edouard Giacomin, Giovanni De Micheli, and Pierre-Emmanuel Gaillardon. Fpga-spice: A simulation-based architecture evaluation framework for fpgas. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(3):637–650, 2019.

[47] Jason Luu, Jason Helge Anderson, and Jonathan Scott Rose. Architecture description and packing for logic blocks with hierarchy, modes and complex interconnect. In *Proceedings of ACM/SIGDA International Symposium on Field Programmable Gate Arrays (ISFPGA)*, page 227–236, 2011.

[48] Elias Ahmed and Jonathan Rose. The effect of lut and cluster size on deep-submicron fpga performance and density. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(3):288–298, 2004.

[49] Kaveh Shamsi. Bitbucket — bitbucket.org. https://bitbucket.org/kavehshm/neos/src/master/.