SCFI: State Machine Control-Flow Hardening Against Fault Attacks

Pascal Nasahl[*], Martin Unterguggenberger[†*], Rishub Nagpal[†*],
Robert Schilling[*], David Schrammel[*], Stefan Mangard[†*]
[*]*Graz University of Technology*
{firstname.lastname}@iaik.tugraz.at
[†]*Lamarr Security Research*

***Abstract*—Fault injection (FI) is a powerful attack methodology allowing an adversary to entirely break the security of a target device. As finite-state machines (FSMs) are fundamental hardware building blocks responsible for controlling systems, inducing faults into these controllers enables an adversary to hijack the execution of the integrated circuit. A common defense strategy mitigating these attacks is to manually instantiate FSMs multiple times and detect faults using a majority voting logic. However, as each additional FSM instance only provides security against one additional induced fault, this approach scales poorly in a multi-fault attack scenario.**

In this paper, we present SCFI: a strong, probabilistic FSM protection mechanism ensuring that control-flow deviations from the intended control-flow are detected even in the presence of multiple faults. At its core, SCFI consists of a hardened next-state function absorbing the execution history as well as the FSM's control signals to derive the next state. When either the absorbed inputs, the state registers, or the function itself are affected by faults, SCFI triggers an error with no detection latency. We integrate SCFI into a synthesis tool capable of automatically hardening arbitrary unprotected FSMs without user interaction and open-source the tool. Our evaluation shows that SCFI provides strong protection guarantees with a better area-time product than FSMs protected using classical redundancy-based approaches. Finally, we formally verify the resilience of the protected state machines using a pre-silicon fault analysis tool.

***Index Terms*—Fault Attacks, Finite-State Machines, Control-Flow Integrity**

I. Introduction

Fault attacks are active, physical attacks that allow an adversary to manipulate the execution of a digital circuit. In these attacks, one or multiple faults are injected into certain gates, wires, or registers of a logical hardware block. The effects of these faults, which comprise transient bit-flips or permanent stuck-at effects, manipulate the execution of the hardware block and an adversary can exploit this malfunctional behavior [7]. Finite-state machines (FSMs) are lucrative fault targets, as these fundamental hardware blocks are responsible of controlling systems and their datapaths. By hijacking the execution flow of the FSM using faults, an adversary can manipulate the FSM to enter states which cannot be reached from the current state. Hence, due to the severity of these attacks, security-sensitive state machines need dedicated protection against faults.

A common fault defense strategy is to encode the FSM states such that they are separated with a certain Hamming Distance [1] [4], [22]. However, this can only mitigate attackers aiming to induce faults into the state registers. Other defense strategies [20] introduce monitors which check whether the conducted state transition is in the list of valid state transitions. Leveugle et al. [13] dynamically verifies that the state transitions stay within the intended execution flow, which is determined during synthesis using the control-flow graph (CFG) of the FSM. There, on each state transition, a signature is derived, and a monitor checks whether the signature matches the predetermined signature of the CFG. However, faults induced either into the next-state logic or into the FSM's control signals still enable adversaries to redirect the control-flow within the bounds of the CFG. Moreover, the fault detection latency of monitor-based schemes is high and the error coverage is often insufficient [24].

Redundantly instantiating the next-state logic and comparing the resulting states typically requires manual effort by the RTL designer. Moreover, this approach requires an additional redundant next-state logic for each additional fault protection layer. Hence, the area overhead of redundancy-based protection mechanisms scales poorly, especially when considering multi-fault attacks, e.g., quadruple laser fault injection [19].

Contribution

In this paper, we introduce SCFI, a scalable mitigation approach probabilistically protecting the control-flow of finite-state machines against multi-fault attacks. SCFI ensures that any control-flow deviation from the intended control-flow is detected with a high probability by substituting the unprotected next-state logic of the controller with a fault-hardened next-state logic. Internally, this hardened logic absorbs the control signals and the execution history and only generates a valid next state when these inputs are not tampered by faults. When either the control signals, the current state (*i.e.*, the execution history), or the next-state logic itself is targeted by faults, the logic ensures that these faults corrupt the next state output to a degree which can be detected. To ensure this behavior, SCFI uses a lightweight diffusion layer, which is based on a maximum distance separable (MDS) matrix multiplication. We integrate SCFI into the Yosys synthesis suite to automatically protect arbitrary FSMs against fault attacks without any user interaction and open-source[1] the modified toolchain. In order to

This project has received funding from the Austrian Research Promotion Agency (FFG) via the AWARE project (grant number 41091245).

[1]https://extgit.iaik.tugraz.at/sesys/scfi

978-3-9819263-7-8/DATE23/© 2023 EDAA

evaluate the area and timing overhead, we synthesized several FSMs used in an industry-driven open-source project with our modified synthesis suite. Our comparison with a redundancy-based protection approach of the FSM's next-state logic shows that SCFI scales better in terms of area-time product for different fault protection levels than classical redundancy-based protection approaches. Finally, we utilize a pre-silicon fault analysis tool to formally verify the fault resiliency of the hardened FSMs.

II. Background

This section provides fundamental background on fault attacks and finite-state machines required for the subsequent chapters.

A. Fault Attacks

Fault attacks are commonly used to break the security of embedded devices. In these physical attacks, one or multiple faults are induced into the circuit, causing several side effects at the electrical level. These electrical effects comprise timing violations and other disturbances [12] and they influence the execution of the target. By exploiting the effects of a fault, an adversary is capable of hijacking the control-flow of software [5], [6], [23], bypassing security measures, such as secure-boot [10], [14], or extracting secret keys used by cryptographic primitives [3], [15].

Originally, fault attacks were pure physical attacks requiring an adversary to have physical access to the target device. To induce a fault, attackers interrupt the supply voltage or the clock signal, decapsulate the chip and shoot with a laser directly into the die, or use electromagnetic pulses [17]. However, recent publications, such as Plundervolt [8], CLKSCREW [9], or VoltJockey [2], demonstrated that faults also could be induced remotely in software, increasing the attack surface of fault attacks even more.

In general, a fault $f \in F$ is described using the set $K = \{e, s, t\}$ where e is the effect of a fault, s the spatial, and t the temporal dimension of the fault. Typically, the fault effect e comprises transient, *i.e.*, bit-flips, or stuck-at effects. The spatial s and temporal t dimensions of a fault describe where (which gate or wire) and when (which clock cycle) a fault is induced. The set F consists of all possible fault combinations and an adversary typically can inject up to a certain number of faults into the circuit.

B. Finite-State Machines

Finite-state machines (FSMs) are sequential circuits responsible for controlling systems and their datapaths. Internally, an FSM maintains a finite set of states, and a state-transition into the next state that is controlled by the input signals, *i.e.*, the control signals and the current state. The outputs of a Mealy-type FSM are defined by the current state and the input signals, and the outputs of a Moore-type FSM only depend on the current state.

As depicted in Fig. 1, an FSM is described using the 5-tuple $\{S, X, Y, \phi, \lambda\}$. The $|S|$ states of an FSM are represented as a s-bit symbol S, where the size s needs to be at least

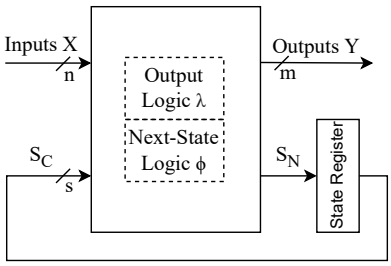

Fig. 1. General structure of a state machine.

$s = \lceil log_2(|S|) \rceil$ bits to comprise the entire state space. Furthermore, the FSM consists of n-bit control signals X and m-bit output signals Y. The FSM uses the next-state function $S_N = \phi(X, S_C)$ to derive the next state S_N from the current state S_C and the control signals X. For a Mealy machine, the output Y depends on the current state S_C and the input signals X and is described using the output function $Y = \lambda(X, S_C)$. The execution-flow of an FSM can be described using a directed

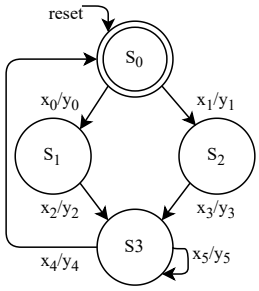

Fig. 2. Control-flow graph of an FSM.

graph, as shown in Fig. 2. This graph, which is also called a control-flow graph (CFG), comprises all valid transitions $t \in CFG$ the FSM can perform. A valid transition is defined by the valid $\{S_C, X\}$ pairs and the next-state function ϕ.

III. Threat Model

We consider a powerful adversary capable of injecting $N-1$ faults in different clock cycles and at different locations into the device under attack. These faults can be induced independently of the used fault methodology, *i.e.*, we consider local and remote injecting techniques. Similar to related work, we model the impact of a fault as a transient, *i.e.*, a bit-flip, or a permanent, *i.e.*, a stuck-at, effect. The spatial dimension of the induced fault comprises wires as well as combinational and sequential elements of the logic.

A. Attacker Description

Within this threat model, an attacker aims to hijack the execution-flow of a security-sensitive state machine in the circuit. Based on the general description of a state machine provided in Section II-B, the adversary can achieve this goal by inducing faults into the next-state logic. A fault into the next-state logic allows an adversary to hijack the execution flow of the FSM and to indirectly change the output signals.

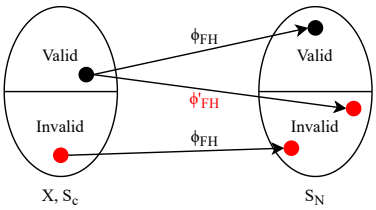

Fig. 3. Mapping of valid and invalid input tuples to a valid or invalid next state.

This fault target can be modeled using the modified next-state logic $S_N = \phi(S_C, X, F_N)$, where F_N describes one or multiple faults. Based on this formula, an adversary can induce faults into different fault targets (FT):

FT1 State Registers: A fault into the state registers allows the adversary to arbitrarily redirect the control-flow of the FSM inside $t \in CFG$ or outside $t \notin CFG$ the control-flow graph. For the CFG in Fig. 2, the adversary could flip bits in the state registers to directly jump from S_0 to S_3.

FT2 Control Signals: By inducing bit-flips into the control signals, the adversary can manipulate the control-flow of the FSM within the borders of the CFG. For example, a fault into the control signal x_0 or into the comparison logic can hijack the execution $S_0 \rightarrow S_1$ to $S_0 \rightarrow S_2$ in Fig. 2.

FT3 Next-State Logic: When directly targeting the logic of the next-state function, the adversary can arbitrarily redirect the control-flow of the FSM within or outside the CFG.

B. Goal - Fault Secure FSM

In order to comprehensively protect the control-flow of finite-state machines against fault attacks, dedicated fault countermeasures must consider all fault targets *FT1*, *FT2*, and *FT3*. The goal is that a fault-protected controller FSM_F influenced by faults detects any control-flow deviations from the control-flow of an identical copy $FSM_{\bar{F}}$ which is not affected by faults, *i.e.*, $\phi_F(S, X, F_N) =? \ \phi_{\bar{F}}(S, X, 0)$.

IV. Design

To comprehensively protect finite-state machines against control-flow hijacks, with SCFI, we maintain the integrity of the control-flow by introducing a fault-hardened next-state logic ϕ_{FH}. This hardened next-state logic prevents that a fault into *FT1*, *FT2*, or *FT3* enables the adversary to redirect the control-flow inside or outside the boundaries of the CFG. This function ϕ_{FH} is internally constructed using a multi-input signature register (MISR) and it links the entire execution history in a compressed format to detect control-flow deviations. To enter the next valid state, the execution history as well as the control signals need to be genuine. As shown in Fig. 3, ϕ_{FH} maps a valid tuple $\{X, S_C\}$, which includes the execution history in S_C, into a valid next state S_N. When an adversary induces faults into *FT1...FT3*, *i.e.*, either into the tuple $\{X, S_C\}$ or into the ϕ_{FH} logic, ϕ_{FH} forces the FSM into a non-escapable terminal error state. Fig. 4 depicts the transformation of an unprotected next-state logic of an example FSM into a protected version. The unprotected FSM is susceptible to faults, as a

```
unique case(SC)
 S0: begin
  SN = S0;
  if(x0)
   SN = S1;
  else if(x1)
   SN = S2;
 end
 S1: begin
  SN = S1;
  if(x2)
   SN = S3;
 end
```

```
unique case(SC)
 S0: begin
  SN = φ_FH(SC, X);
 end
 S1: begin
  SN = φ_FH(SC, X);
 end
 ERROR: begin
  SN = ERROR;
 end
 default: begin
  fsm_alert = err_signal;
  SN = ERROR;
 end
```

Fig. 4. Unprotected and protected next-state logic of an example FSM.

single fault into the state registers, the comparison logic, or the control signals can change the execution-flow of the FSM. SCFI closes these attack vectors by deriving the next state using ϕ_{FH}. If the current state, the control signals, or the next-state logic is tampered with a fault, ϕ_{FH} produces an invalid state and enters the non-escapable default error state. To achieve this protection degree, the next-state function and its inputs and outputs need to fulfill requirements **R1** to **R3**:

R1 Encoded Control Signals: All control signals X are encoded to X_e. The encoding needs to guarantee that the attacker needs at least N bit-flips to manipulate a valid control-signal codeword to another valid codeword.

R2 Encoded States: All states S are encoded to s_e-bit states S_e. Similar to the control signals, the encoding needs to guarantee a minimum Hamming Distance between valid states of N.

R3 Hardened Next-State Function: The hardened next-state function ϕ_{FH} generates an encoded next state S_{Ne} fulfilling **R2** for each encoded control signal and encoded current state tuple $\{S_{Ce}, X_e\}$ (**R3.1**). Moreover, ϕ_{FH} needs to ensure that up to $N-1$ bit-flips into its circuit or into the input space affect the output in such a way, that the faults can be detected, *i.e.*, an invalid state S_{Ne} is generated (**R3.2**).

Due to requirement **R3**, the state derived in different paths merging at some point also produces different encoded states. For example, the path $S_1 \rightarrow S_3$ in Fig. 2 derives a different state than the path $S_2 \rightarrow S_3$. As maintaining different state symbols for a single state is costly, we add an additional requirement:

R4 Collision Capability: The hardened next-state function needs to produce the same encoded next state for different paths using a modifier, *i.e.*, $\phi_{FH}(S_{C1e}, X_{1e}, Mod_1) == \phi_{FH}(S_{C2e}, X_{2e}, Mod_2)$ for $S_{C1e} \neq S_{C2e}$ and $X_{1e} \neq X_{2e}$. The modifiers Mod_1 and Mod_2 are used to produce a state collision.

In the following section, we discuss one possible selection for the hardened next-state function.

A. Selection of the Hardened Next-State Function

In SCFI, the hardened next-state function ϕ_{FH} is based on a lightweight diffusion function used in cryptographic primitives.

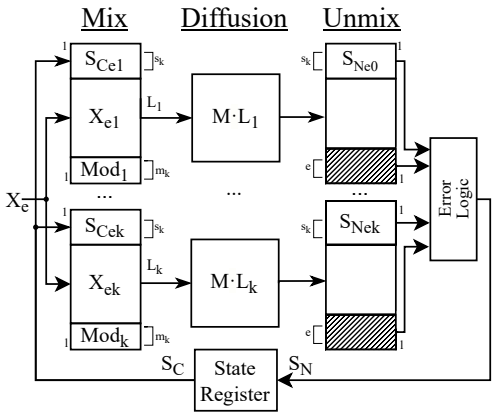

Fig. 5. SCFI hardened next-state function.

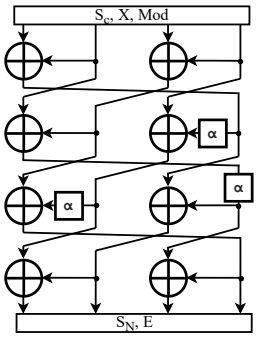

Fig. 6. Internal structure of the MDS matrix multiplication [16]. All elements operate on 1-bytes each.

This function ϕ_{FH}, as shown in Fig. 5, maps the input space consisting of the encoded control signals X_e (**R1**), the encoded current state S_{Ce} (**R2**), and the modifier Mod (**R4**) to a next encoded state S_{Ne} (**R3.1**). The properties of the underlying diffusion function imply that any fault at the input space or within the logic maximally affect the output, thereby substantially decreasing the probability of a successful fault attack and probabilistically fulfilling (**R3.2**). Overall, SCFI's hardened next-state function consists of three layers:

a) Mix Layer: In this layer, the input triple is split into k l-bit vectors L. For this, the encoded current state, the encoded control signals, and the modifier are split into k shares and each share is placed into the vectors, as shown in Fig. 5.

b) Diffusion Layer: Then, in the diffusion layer, the vectors L are absorbed by k diffusion functions. These functions conduct a linear transformation $D(L) = M \cdot L$ which is a matrix multiplication of vector L with matrix M in a specific field. This transformation, depending on the choice of matrix M, yields a strong diffusion. Ideal choices of this matrix are called maximum distance separable (MDS) matrices, maximizing the diffusion property. Hence, by choosing such an ideal MDS matrix, a fault at the input or within the function maximally propagates to the output, destroying the next state with a high probability (**R3.2**).

c) Unmix Layer: The output of the diffusion layer is stored into k l-bit vectors. The concatenation of the first $s_e/k = s_k$-bits of each output vector results in the encoded next state S_{Ne} (**R3.1**). As the size $k \cdot l$ of the output space is larger than the size s_e of the encoded state, $k \cdot l - s_e$ bits are free, providing the collision property (**R4**). Additionally, SCFI uses, depending on the required fault security, the e topmost free bits of each output vector as error detection bits E. Here, by choosing a corresponding modifier Mod, ϕ_{FH} sets these bits to a predefined value, *i.e.*, 1. In the error logic, the logical `AND` of S_{Ne} and E infects the next state when a fault-induced error happens.

V. Implementation

We open-source a modified version of the Yosys [18] open synthesis suite capable of automatically protecting arbitrary FSMs with SCFI. The protection can be enabled globally or selectively for the unprotected FSMs in the design flow with a certain fault protection level N. Our implementation adds a new Yosys pass to the suite operating in between of other optimization passes before the design is mapped to the logic gate level. Note that the RTL designer only needs to manually encode the control signals with a Hamming Distance of N-bits in the modules driving these signals.

A. Next-State Logic

First, our custom FSM protection pass identifies the unprotected FSM by utilizing the existing Yosys FSM passes. Then, the FSM's state variables are re-encoded so that the Hamming Distance between these variables is N. Afterwards, our pass extracts the CFG of the FSM and stores the current state, the next state, and the control signals for each control-flow edge. With this information, the modifier Mod for state transition is determined, satisfying the equation $MDS(S_{Ce}, X_e, Mod) = S_{Ne}$.

For the MDS diffusion function, we use a lightweight construction with a minimal gate count proposed by Duval et al. [16]. As shown in Fig. 6, this function splits the 32-bit input space into 4 8-bit chunks, performs the matrix multiplication, and returns 4 8-bit vectors which form the 32-bit output. In SCFI, we selected the $M^{8,\,3}_{4,\,6}$ [16] MDS matrix operating in the field $\mathbb{F}_2[\alpha]$ with $\alpha = X^8 + X^2 + 1$. This particular matrix has a low `XOR` count with a slightly larger logical depth compared to other matrices in the 4×4 category. We note that the choice of MDS matrix can be changed according to design requirements, *i.e.*, area or timing constraints.

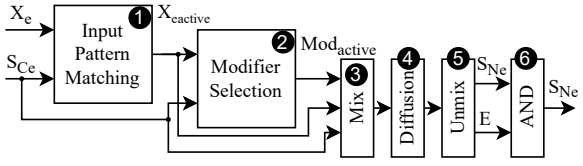

Fig. 7. The next-state logic hardening pass.

Having the modifiers, our pass describes the logic of the next-state function in the internal Yosys register-transfer level intermediate language (RTLIL). As depicted in Fig. 7, first ❶,

the active control signal $X_{e_{active}}$ is determined by performing a pattern match of the control signal and the current state S_{Ce}. Then, using this signal and S_{Ce}, the modifier for this input is selected ❷. In the mix layer ❸, the wires of the triple $\{S_{Ce}, X_{e_{active}}, Mod_{active}\}$ are distributed to the k 32-bit input MDS diffusion functions. These lightweight diffusion functions ❹ consist of only `XOR` gates. In the unmix layer ❺, the next state S_{Ne} is concatenated and the error bits E are selected. By connecting S_{Ne} and E using `AND` gates ❻, a fault infectively destroys the next state.

VI. Evaluation

To evaluate the effectiveness of SCFI in terms of area, timing, and security when protecting security-sensitive FSMs of an industry-driven project, we integrate our custom Yosys pass into the design flow of the OpenTitan [21] secure element. This chip, which is entirely open-source, acts as a secure root-of-trust and provides a key storage and cryptographic accelerators.

A. Area Overhead

In order to evaluate the area overhead introduced by SCFI, we analyzed unprotected *(i)*, manually protected *(ii)*, and automatically protected *(iii)* FSMs. As the reference *(i)* for our evaluation, we selected several FSMs of OpenTitan and synthesized the entire corresponding module with Yosys using the open-source Nangate45 standard cell library. For the manually protected *(ii)* FSMs, we encoded the control signals with a Hamming Distance of N-bits and instantiated the next-state logic of the FSM N times. To detect control-flow hijacks triggered by faults, we designed a small error logic monitoring the state registers of the redundant FSMs and raising an error signal when one or more state values mismatch. Finally, we automatically protected *(iii)* the reference *(i)* FSMs by calling the SCFI Yosys pass in the design flow. Similar to the manually protected *(ii)* FSMs, we encoded the control signals with a HD of N-bits and configured SCFI that at least N faults are required to hijack the FSM. Table I illustrates the area overheads for the three configurations for different FSMs and different protection levels N ranging from 2 to 4. For the manual redundancy approach, the geometric mean of the area overhead is 17.5 % for $N = 2$, 42.9 % for $N = 3$, and 67.6 % for $N = 4$. In comparison, the geometric mean area overhead for the FSMs protected with SCFI is 9.6 % for $N = 2$, 21.8 % for $N = 3$, and 27.1 % for $N = 4$. Note that for smaller input spaces $\{S_{Ce}, X_e, Mod\}$ the area overhead for SCFI could be higher than for a redundancy approach (cf. `otbn_controller` in Table I) as SCFI needs to instantiate a MDS matrix with a 32-bit input.

TABLE I
Area overhead for protecting different FSMs using redundancy or SCFI.

	Unprotected	Redundancy			SCFI		
	Area [GE]	Area [%]			Area [%]		
Protection Level		2	3	4	2	3	4
adc_ctrl_fsm	1019	38	76	121	14	27	42
aes_control	632	13	44	77	6	22	32
i2c_fsm	2729	38	70	109	20	21	27
ibex_controller	537	29	75	122	13	34	43
ibex_lsu	933	10	21	32	2	13	16
otbn_controller	2857	1	4	5	5	5	6
pwrmgr_fsm	301	89	184	334	33	71	84
Geometric Mean		17.5	42.9	67.6	9.6	21.8	27.1

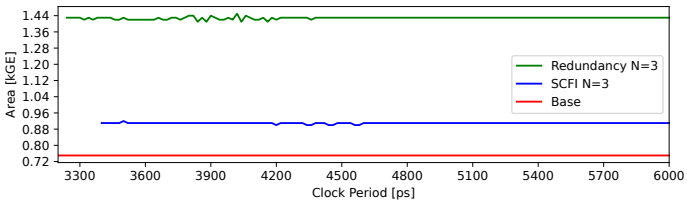

Fig. 8. Area-time product for the `adc_ctrl_fsm` module in different configurations.

B. Timing Overhead

SCFI affects the timing of the next-state logic by introducing the fault-hardened next-state function ϕ_{FH}. However, the timing overhead is minimal, as the logical depth of ϕ_{FH} comprises four `XOR` layers for the MDS multiplication and an `AND` layer for the error masking. We successfully synthesized all modules in all configurations depicted in Table I for OpenTitan's target frequency of 125 MHz with Yosys and the open-source standard cell library.

Fig. 8 illustrates the area-time (AT) product for the unmodified, the redundancy-protected, and the SCFI-hardenend `adc_ctrl_fsm` module. In this plot, we increased the clock period from 3200 ps to 6000 ps and measured the area in kGE of the design synthesized by Cadence Genus and a proprietary cell library. For this experiment, we switched from Yosys to the Cadence synthesis suite as Yosys and the internally utilized yosys-abc tool only provides basic area and time optimization functionality. As shown in Fig. 8, Cadence was able to meet the timing for a maximum frequency of 312 MHz for the base design, 308 MHz for the design using redundancy, and 294 MHz for SCFI. However, this slightly decreased frequency is typically not problematic, as the critical path of a design is usually not in an FSM. Moreover, as depicted, SCFI achieves a better AT product for protecting the next-state logic of the FSM in the `adc_ctrl_fsm` module than the redundancy approach.

C. Security Evaluation

By encoding the control signals and the state variable, an adversary cannot hijack the state machine by inducing faults into fault targets *FT1* and *FT2*. As the input pattern matching logic ❶ of ϕ_{FH} operates on these encoded signals, the attacker needs to induce N faults into this block to manipulate the active, encoded control signal. While a fault into the modifier selection block ❷, which consists of multiplexers, could select a different modifier, the attacker cannot exploit this injected fault. More specifically, a fault would yield a combination of control signal, state, and modifier which creates a non-valid next state. Internally, the mix layer ❸ consists of a rewiring of the encoded control signals, the state, as well as the modifier. Hence, this layer can resist up to $N - 1$ faults. The idea of the diffusion layer ❹ is that a small change

at the input causes a significant change at the output, *i.e.*, the avalanche effect. To achieve this property, SCFI internally uses MDS matrix multiplication yielding optimal diffusion guarantees. These MDS matrices propagate a bit-flip in a single input byte to all four output bytes, *i.e.*, they have a branch number of 5. Hence, one or multiple bit-flips into the input triple $\{S_{Ce}, X_{e_{active}, Mod_{active}}\}$ propagate through this function affecting multiple output bits. By effecting the next state S_{Ce} or the error bits E, an invalid state is generated in the unmix ❺ and error ❻ layer and the FSM enters the default error state. Precisely, there are only $|S_{Ne}| + |E|$ valid output states; an attacker who induces N faults on the next-state function inputs, $\{X, S_C\}$, would have a success probability of $P = \frac{|S_{Ne}|+|E|}{k \cdot 2^{32-(|S_{Ne}|+|E|)}}$. However, considering that $|S_{Ce}| + |E| << k \cdot 2^{32-(|S_{Ne}|+|E|)}$, the success probability is very small. For attacks within the next-state function, the MDS property of the diffusion layer ensures that the success probability still remains quite low, albeit it is higher than the previous case. As shown in Fig. 6 depicting the construction of the MDS matrix, faults in the first three `XOR` layers propagate to at least two output bytes. Although a fault at the last layer only affects one output byte, all valid output states S_N are still encoded with a Hamming Distance of N, requiring that the adversary needs to induce N bit-flips.

D. Formal Security Analysis

We formally analyzed the resilience of the diffusion layer consisting of the MDS matrix multiplication by utilizing SYNFI [11], a recently introduced pre-silicon fault analysis tool operating at the netlist. For the analysis, we synthesized an FSM with 14 state transitions and configured SCFI with a protection level of 2 bits (HD). We used SYNFI to analyze whether it is possible to hijack one of the state transitions and enter another next state using faults. In total, we injected 7644 single bit-flips exhaustively into all available gates in the MDS matrix multiplication and 32 (0.42 %) of these faults enable an adversary to hijack the execution-flow of the FSM.

Note that analyzing the resilience of FSMs against faults is also necessary when using other protection approaches. For example, when redundantly instantiating the next-state logic to mitigate faults, a synthesis tool aiming to meet timing and area constraints could weaken the security when optimizing the design.

VII. Limitation & Future Work

A potential future work could extend SCFI to adapt the MDS matrix size to the size of the $\{S_C, X, Mod\}$ input triple to further improve the area-time product. In addition, the formal analysis could be integrated into the Yosys pass to increase security guarantees of SCFI. Finally, a future work could investigate how SCFI could be extended to also provide protection for the output logic.

A limitation of the current prototype implementation is that the selector signals of the MUXes used in the input pattern matching logic ❶ are 1-bit signals. This would allow an adversary to redirect the control-flow within the bounds of the CFG. To mitigate this attack vector, an updated version of the SCFI Yosys pass could introduce encoded selector signals.

VIII. Conclusion

In this paper, we presented SCFI, a methodology capable of protecting the control-flow of finite-state machines against fault attacks. SCFI substitutes the next-state logic of FSMs with a fault-hardened function only deriving a next valid state in a fault-free scenario. We integrated SCFI into the Yosys synthesis suite and open-sourced our modified toolchain. Our evaluation shows that the area overhead for FSMs protected with SCFI is lower than for traditional protection approaches.

References

[1] K. Akdemir, H. Ghaith, and B. Sundar, "Non-linear Error Detection for Finite State Machines," WISA, 2009.

[2] P. Qiu, D. Wang, Y. Lyu, and G. Qu, "VoltJockey: Breaching TrustZone by Software-Controlled Voltage Manipulation over Multi-core Frequencies," CCS, 2019.

[3] E. Biham, and A. Shamir, "Differential Fault Analysis of Secret Key Cryptosystems," CRYPTO, 1993.

[4] M. Choudhury, D. Forte, and S. Tajik, "A Pragmatic Approach for Encoding Laser Fault Injection Resistant FSMs," DATE, 2021.

[5] N. Timmers, and C. Mune, "Escalating Privileges in Linux Using Voltage Fault Injection," FDTC, 2017.

[6] N. Timmers, A. Spruyt, and M. Witteman, "Controlling PC on ARM Using Fault Injection," FDTC, 2016.

[7] I. Verbauwhede, D. Karaklajic, and J. Karaklajic, "The Fault Attack Jungle - A Classification Model to Guide You," FDTC, 2011.

[8] K. Murdock, D. Oswald, J. Bulck, D. Gruss, and F. Piessens, "Plundervolt: Software-based Fault Injection Attacks against Intel SGX," S&P, 2020.

[9] A. Tang, S. Sethumadhavan, and S. Stolfo, "CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management," Usenix, 2017.

[10] A. Cui, and R. Housley, "BADFET: Defeating Modern Secure Boot Using Second-Order Pulsed Electromagnetic Fault Injection," WOOT, 2017.

[11] P. Nasahl et al., "SYNFI: Pre-Silicon Fault Analysis of an Open-Source Secure Element," TCHES, 2022.

[12] J. Brockmann, P. Sasdrich, and T. Guneysu, "Revisiting Fault Adversary Models - Hardware Faults in Theory and Practice," IACR Cryptol. ePrint Arch., 2021.

[13] J. Leveugle, and G. Saucier, "Optimized Synthesis of Concurrently Checked Controllers," IEEE Trans. Computers, 1990.

[14] A. Vasselle, H. Thiebeauld, Q. Maouhoub, A. Morisset, and S. Ermeneux, "Laser-Induced Fault Injection on Smartphone Bypassing the Secure Boot-Extended Version," IEEE Trans. Computers, 2020.

[15] C. Dobraunig, M. Eichlseder, T. Korak, and S. Mangard, "SIFA: Exploiting Ineffective Fault Inductions on Symmetric Cryptography," CHES, 2018.

[16] S. Duval, and G. Leurent, "MDS Matrices with Lightweight Circuits," IACR Trans. Symmetric Cryptol., 2018.

[17] D. Karaklajic, J. Schmidt, and I. Verbauwhede, "Hardware Designer's Guide to Fault Attacks," IEEE Trans. Very Large Scale Integr. Syst., 2013.

[18] C. Wolf, "Yosys Open SYnthesis Suite," https://yosyshq.net/yosys, 2022.

[19] ALPhANOV, "PILAS - Advanced Laser Injection for Security Analyses," https://www.alphanov.com/en/collaborative-projects/pilas-advanced-laser-injection-security-analyses, 2022.

[20] GL. Djordjevic, TR. Stankovic, and MK. Stojcev, "Concurrent Error Detection in FSMs using Transition Checking Technique," TELSIKS, 2005.

[21] S. Johnson, D. Rizzo, P. Ranganathan, J. McCune, and R. Ho, "Titan: enabling a transparent silicon root of trust for Cloud," Hot Chips, 2018.

[22] C. Muhtadi, T. Shahin, and F. Domenic, "SPARSE: Spatially Aware LFI Resilient State Machine Encoding," HASP, 2021.

[23] P. Nasahl, and N. Timmers, "Attacking AUTOSAR using Software and Hardware Attacks," escar USA, 2019.

[24] R. Rochet, R. Leveugle, and G. Saucier, "Efficiency comparison of Signature Monitoring Schemes for FSMs," ASP-DAC, 1995.