2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID)

An Efficient Memory Zeroization Technique Under Side-Channel Attacks

Ankush Srivastava* and Prokash Ghosh†

*Computer Architecture and Dependable Systems Lab (CADSL), IIT Bombay, Mumbai, India Email: ankushs@ee.iitb.ac.in
†NXP Semiconductor India Pvt Ltd, Noida, India Email: prokash.ghosh@nxp.com

***Abstract*—Protection of secured data content in volatile memories (processor caches, embedded RAMs etc) is essential in networking, wireless, automotive and other embedded secure applications. It is utmost important to protect secret data, like authentication credentials, cryptographic keys etc., stored over volatile memories which can be hacked during normal device operations. Several security attacks like cold boot, disclosure attack, data remanence, physical attack, cache attack etc. can extract the cryptographic keys or secure data from volatile memories of the system. The content protection of memory is typically done by assuring data deletion in minimum possible time to minimize data remanence effects. In today's state-of-the-art SoCs, dedicated hardwares are used to functionally erase the private memory contents in case of security violations. This paper, in general, proposes a novel approach of using existing memory built-in-self-test (MBIST) hardware to zeroize (initialize memory to all zeros) on-chip memory contents before it is being hacked either through different side channels or secuirty attacks. Our results show that the proposed MBIST based content zeroization approach is substantially faster than conventional techniques. By adopting the proposed approach, functional hardware requirement for memory zeroization can be waived.**

I. Introduction

Random access memory (RAM) is a type of data storage that stores machine code and user content with almost equal read or write access time. Though it is volatile in nature, however, it does not loose content immediately after power supply is removed [1]. Ordinary volatile memory (including SRAM) typically loses their content gradually over the period, and data may persist for several minutes at the standard operating temperature. The memory data discharge may extend to an hour if it is kept at very low temperature or under frozen nitrogen [2]. Residual data can be recovered using simple, nondestructive techniques that require only momentary physical access to the machine [2], [3]. Attacker can recover the original cryptographic keys or secure data even if they find stored content which is 20-25% corrupted [2].

In complex designs, including system-on-chips (SoCs), there are different mechanisms present, such as tamper detection circuits [4], [5], memory bus monitoring [6], secure state monitoring using trust architecture (e.g. ARM trust architecture [7], [8]), secure supervisor circuits [9] etc., for detecting attempt of unauthorized accesses to the data storage. There are several tamper detection circuits available, such as voltage sensor [6], radiation sensor, probe sensor, wire sensor, temperature sensors [5] etc., which are usually adopted in practice to counter security attacks. Such type of circuits detect the temperature or speed variations during functional operation and asserts hardware interrupt, whenever it finds unexpected change in operating temperature [4] or system performance due to security violations. Similarly there are on-chip voltage sensors which detect sudden change in operating voltage and flag security violation (tamper) event. Such attacks are often called glitch attack [4]. *On detection of such tamper events, on-chip and off-chip memories must be initialized to all zeros* (zeroized) using various techniques [4], [7], [10]–[12] presented in the literature. In rest of the paper, memory zeroization is referred as complete memory data initialization with all zeros. It is a process of erasing data from storage element as soon as tamper detection response is sensed by the system.

Typically in complex designs, additional on-chip finite state machine (FSM) based fixed data generators are used for memory zeroization [9]. Each secured intellectual properties (IPs) have dedicated such FSM based data generators to initialize secured memory contents to zero. However, Ho et. al [13], [14] attempted to make data non-imprintable in SRAMs, without using zeroization technique, which is never adopted in practice due to significant area and power overhead. The only practical approach is to completely zeroize the memories as soon as security violation is detected. *Even in today's state-of-the-art SoCs, FSM based data generators are used to initialize memory locations of each memory cluster by issuing several thousands of sequential write transactions*. The memories are accessed via functional data paths for initialization purposes, which is practically implemented in [7]. The major drawback is, it incurs huge latency in erasing big memory clusters as each address location is written sequentially.

Embedded memories usually occupy more than 40% area of the device and out of which, approximately 10-30% of memories are critical from security point of view. If such secured memories of different IPs can be erased using much faster approach, then the data stealing probability due to secuirty attacks can be minimized significantly. *The limitations with the conventional memory zeroization techniques (either using on-chip controller or software based approach) are*: a) functional address space is initialized sequentially even if it consists of multiple hard macros of compiled or custom memories (e.g. 64MB space can be made up of 8 hard macros of 8MB SRAMs), b) dedicated hardware at each IP level

2380-6923/19/$31.00 ©2019 IEEE
DOI 10.1109/VLSID.2019.00032

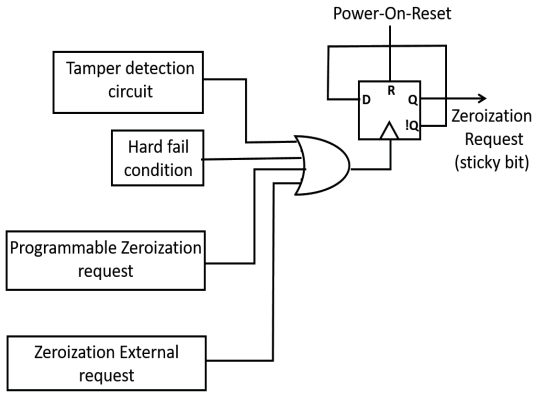

Fig. 1. Source of memory zeroization requests

to zeroize memory contents in case of security violations. It is utmost desirable in today's electronic system design to reduce such large latency in memory zeroization. Additionally, it should also reduce the hardware requirements to implement such system.

This paper proposes, for the first time, a *novel approach of memory zeroization using existing memory built-in self-test (MBIST) structure in such a way that it performs the intended function once security violation(s) is(are) detected, in addition to memory testing while in test mode*. It is shown in the paper that the adopted new approach is much faster than the traditionally implemented solutions for memory zeroization, which reduces data stealing probability under security attacks. Furthermore, the proposed technique does not require any additional hardware to perform complete memory zeroization.

In nutshell, this paper introduces a *scalable and structured system level solution to perform memory zeroization under several security violations*. The contents of the paper are organized as follows. Section II discusses the background and motivation of this work. It includes overview of previous works and discusses the recent implementations at system level. Section III discusses the implementation of the proposed MBIST based zeroization technique for on-chip embedded memories. Section IV evaluates the efficiency of the proposed technique using existing mainstream MBIST solution. Finally, section V concludes the paper.

II. Background and Motivation

A. Overview of the Previous Work

The freezing attacks have been reported on various memory systems as discussed in [2], [3], [15], which typically consist of SRAM and DRAM cells respectively. Several studies show that the data remanence period becomes larger at lower temperature, which can range from several seconds to several minutes at very low temperature like -50^{o}C. Some devices are therefore designed with on-chip temperature sensors to report any drop below a certain threshold value. It usually generates a trigger event which is treated as a security violation and must be categorized as high priority hardware interrupt by the system to perform memory zeroization immediately. National Institute of Standards and Technology (NIST) [11] sets the compliance to zeroize data storage which stores secured keys or secret data. The type and class of storage that need to be erased, under security attacks, is further discussed in [16].

There are multiple sources of global memory zeroization request which may originates from tamper detection circuit as well and shown clearly in Fig. 1. The zeroization request can also be generated by hard fail conditions occurred in system-on-chip [7]. When such SoCs observe a breach of security while operating in ARM^{TM} trust mode [8], then it mandatorily generates the zeroization request. The running software can also put a request by programming a register in the SoC. The zeroization request can also be triggered by an external event outside the device. This enables on-board devices to put zeroization request in case of intrusion detection.

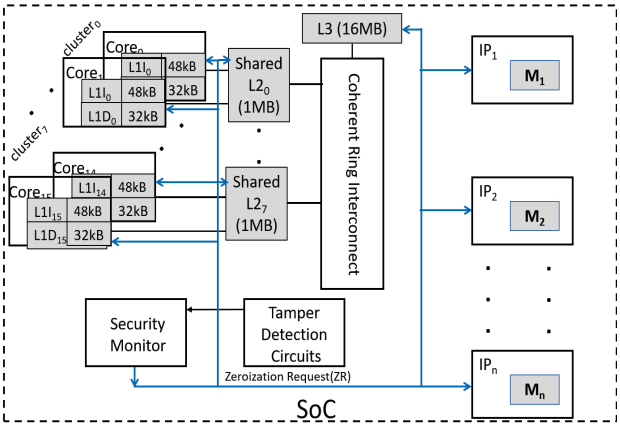

Fig. 2. Implementation of Memory Zeroization Scheme: An Architectural View

It can be seen in Fig. 1 that all different sources of security violations are ORed together and sampled by a flip-flip whose reset is controlled by system power-on-reset. We will show later that how the zeroization request is being honored at system level and how it helps the hardware to erase the memory content. Usually, additional hardware is deployed to erase data from the memory system that constitutes the functional address space. A larger memory system typically built using a set of smaller embedded memories. This is due to the fact that a single large memory implementation is impractically with low yield. The another bottleneck is huge area overhead for such larger memories and may not be practical in meeting timing specifications. In state-of-the-art SoCs, a single large memory typically built using a clustered set of smaller memories with required functional address space.

B. Recent Implementation

Fig. 2 shows the typical implementation for initializing memories in a SoC. On-chip *tamper circuits* detect possible intrusion of adversaries and asynchronously inform *Security*

Monitor regarding the possible security violation(s). For example, if on-chip temperature tamper detection circuit detects unexpected change in operating temperature, then it generates a tamper event. This event is sampled by the on-chip security monitor and it sends *zeroization request* (ZR) to all secured memory system instantiated inside the various IPs, including different levels of caches. The memory system (M_1 to M_n) inside the IPs (IP_1 to IP_n) are clearly shown in Fig. 2, including L1 instruction caches ($L1I_0$ to $L1I_{15}$), including L1 data caches ($L1D_0$ to $L1D_{15}$), shared L2 caches ($L2_0$ to $L2_7$) and shared L3 cache. The ZR signal, shown in blue line, reaches to all IPs and various cache memories of different processors to initiate memory zeroization process. When each IP samples the signal activity, embedded SRAM controller aborts ongoing memory operations and immediately starts writing zeros to full addressable space.

For example, assume that IP_1 has 1MB memory (M_1) with 32bit aligned data width. The memory would require (1024*1024)/32 = 32,768 cycles to completely zeroize the content, assuming each address location requires one cycle to write all zeros. In practice, a memory of size 1MB is made up of smaller memories and with the traditional scheme, it will access each location sequentially without exploiting the physical partitioning of the memory system. Once memory zeroization gets completed, the SoC state machine will reboot the device as per specifications. If we assume that all memory system have started zeroization at the same time with identical functional clock, then the time required to completely zeroize the memory system would be limited by the largest memory size. Assuming L3 memory size is largest among all, then it would be the bottleneck in the fastest zeroization requirement. At present, memories are initialized sequentially over the functional address space and this motivated us to explore a new approach which can make them initialize faster by exploiting parallelism in addressing physical memory space.

III. Leveraging Existing MBIST Architecture for Faster Zeroization

Fig. 3 shows a memory system having a cluster of 32 memories, each having a size of 72 KB, to form 2304 KB sized single memory. The individual memories are shown from M1 to M32, where memory size and logical address mapping are shown clearly. It is important to notice that a single memory instance of size 2304 KB is not practically feasible to be built by the memory compiler. Hence, designers typically use multiple instances of smaller memories to form a larger one which is shown in Fig. 3. It can be seen that each memory can be accessed by two different paths: a) functional path which is shown in red color, where memories can be accessed sequentially, b) test path accessed via Memory BIST engine and each memory can be accessed parallelly. The drawbacks of using functional path to initialize each memory are: a) longer zeroization time as memories are accessed sequentially, b) additional hardware to generate logical addresses via functional path.

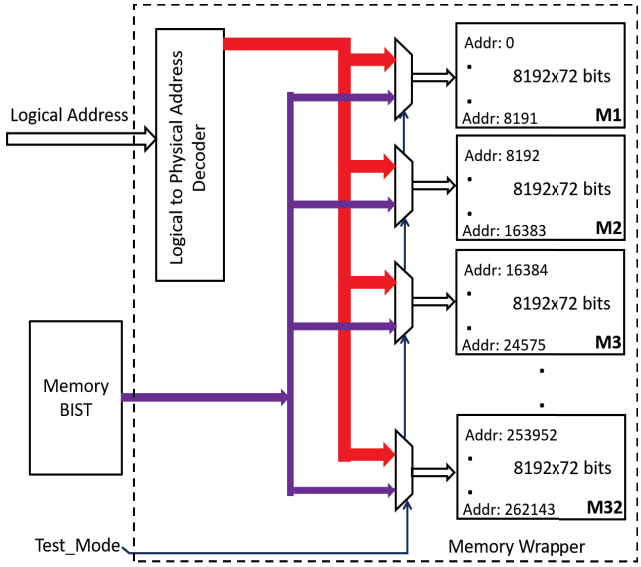

Fig. 3. A Typical Memory cluster with functional and test paths

Looking at the current example, we have an opportunity to identify alternative faster solution, instead of using conventional approach that uses dedicated hardware, to zeroize memory system. A faster zeroization scheme can be achieved by adopting those solutions which are already present in almost all designs, however, for testing purposes only. Generally, MBIST engine has direct access to each memory via test path. Memory testing is typically done in normal functional mode by switching the control signal (*Test_Mode* is our case) of the multiplexers that fall along data and control path of the memory. We can leverage this property for quickest possible zeroization by doing small changes in MBIST engines and system level operations, which would be discussed in next few sub-sections. In Fig. 3, the total number of address locations are 262144 and assuming that functional hardware is zeroizing the memories one cycle per address, then it would take 262144 clock cycles. However, if we choose to initialize (or zeroize) the memories through MBIST engine via test path then it would take only 8192 clock cycles (as opposed to 262144 cycles) as all 32 memories can be zeroized parallelly. In our new proposed approach, we intend to use Memory BIST engines to initialize embedded clustered memories parallelly which would be substantially faster than zeroizing such memories through functional data generators.

A. Method of Operation: System Level

Fig. 4 shows a generic state diagram that may be adopted in system-on-chip (SoC) for device boot and slightly modified to zeroize memory system once security violation is sensed. Here "POR" denotes Power-on-Reset which is externally controlled through general purpose input-output (GPIO) pad with active high input. "ZR" and "ZD" represent zeroization request and zeroization done signal respectively. As long as *POR* is asserted, FSM will continue to remain in "Power-On Reset"

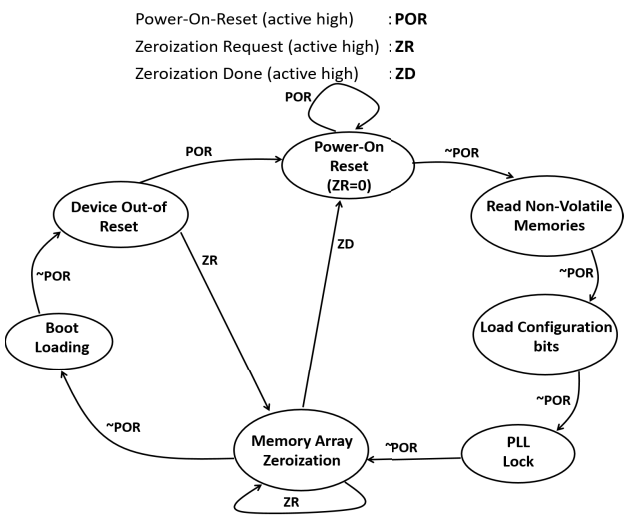

Fig. 4. A generic state diagram of system-on-chip to support memory initialization (or zeroization)

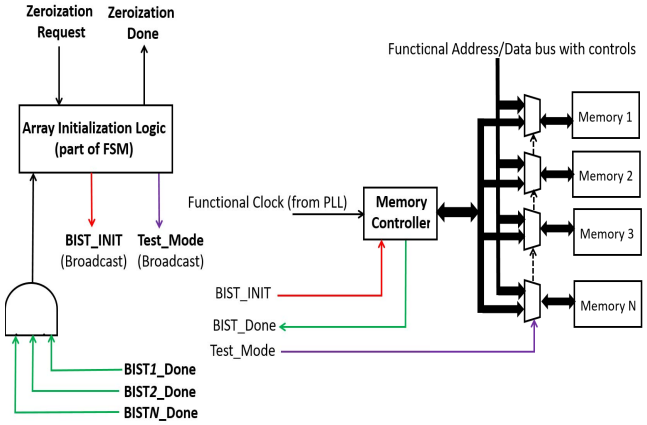

Fig. 5. Zeroization Hardware Architecture using Memory BIST

state. Once POR is de-asserted, FSM will read the content of non-volatile memories such as one time programmable (OTP) memories, embedded flash, etc during the state "Read Non-Volatile Memories". After reading the contents of OTP memories, SoC specific configuration bits are required to boot the IO peripherals which are fetched in state "Load Configuration bits". Then device waits for all PLLs to lock and reach "PLL Lock" state.

PLL lock is an important step as it confirms that device is now getting ready to work at rated frequency. It might be a case where different memory systems are using different PLLs on the same device, however, our proposed architecture covers all such cases. After all PLLs get locked, default memory array zeroization is initiated in state "Memory Array Zeroization". After array initialization, SoC boots the IO peripherals to get out-of-reset under state "Device Out-of-Reset". During the functional mode, the SoC continues to be in "Device Out-of-Reset" state and if security related circuits sense any violation(s) then it apparently asserts *ZR* request such that the state machine can move immediately to state "Memory Array Zeroization". Once memory systems are zeroized, a zeroization done *ZD* acknowledgment is sent back to SoC state machine. After receiving *ZD*, FSM moves to "Power-On-Reset" state in order to clear *ZR* signal. Next sub-section covers the system-level architectural implementation to zeroize memories using MBIST engines.

B. The Proposed Architecture Exploiting Existing MBIST Solution

This sub-section explains the architectural implementation which can exploit the capabilities of Memory BIST engines in zeroizing the embedded memories. A system level implementation of our proposed zeroization technique to erase sensitive data on memories is shown in Fig. 5. The state machine, as shown in Fig. 4, keeps polling for the zeroization request (ZR) issued from the device. Once ZR request is captured by the FSM, it jumps to "Memory Array Zeroization" , it disconnects the functional path of memories, and broadcast a set of signals to first activate test path (asserting *Test_Mode* signal) to access each memory and enable MBIST engine, by issuing *BIST_INIT*, to write all zeros to corresponding memories. Memory BIST engine starts performing zeroization as soon as *BIST_INIT* is being asserted (shown in red color) and it acknowledges back to system state machine by issuing BIST done (*BIST_DONE*) signal (shown in green color). Device exits this state and in this state all memories are inaccessible via functional path. Hence any kind memory transactions conflict attacks are also not possible with proposed scheme.

It can be seen that all BIST done are ANDed together such that it would only be honored by the system if all MBIST engines are done with zeroization. We have shown a generic implementation of memory BIST that may be changed based on the clocking structure and SoC state machine respectively. In the adopted approach, we have exploited the fact that the memory BIST engine can be used to zeroize memories without considering the fundamental limitations of functional memory zeroization schemes. MBIST engine can initialize all memories which are clustered together to form a bigger functional memory as discussed earlier using Fig. 3. *Our novel proposed solution can overcome the traditional latency in initializing secured memories once security violation(s) is(are) detected and it is transaparent to software*. As memories goes under the control of MBIST(s), it would be almost impossible for any adversary to stop zeroization except cutting power supply.

C. Execution of Memory Zeroization

This sub-section outlines a generic execution flow that must be followed when a system (or SoC) encounters a security violation under normal condition, which is intuitively covered in Fig. 4 and described further in Fig. 6. It clearly shows that the system (SoC state machine) automatically enters into

TABLE I
ZEROIZATION RUNTIME ANALYSIS ON INDUSTRIAL DESIGNS

IP name	No. of memory instances	Total memory size (KB)	Functional address space	Largest Memory		Cycles required for zeroization (conventional)	Cycles required for zeroization (proposed)	Zeroization speed-up
				Memory Size (KB)	Address space			
DMA Cntrl.	30	318	2368	36	256	2368	256	9
PEB	32	18432	262144	590	8192	262144	8192	32
Qbman	32	8866	71600	592	2048	71600	2048	35
DBG	10	1096	8192	35	256	8192	256	32
OCRAM	21	3328	65536	426	8192	65536	8192	8
CCN	64	121094	114688	1180	8192	114688	8192	14
DDR Cntrl.	12	402	12416	66	2048	12416	2048	6

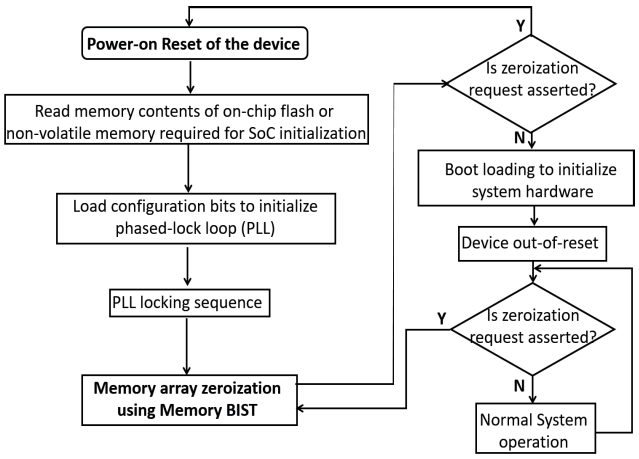

Fig. 6. Execution Flow of the proposed zeroization technique

zeroization mode, every time a ZR request is issued by the tamper detection circuit. Once the content from memories are erased, device moves to next SoC state (based on the value of ZR). The logic 0 of ZR request is considered as normal system operating condition. Once ZR request is asserted, the flowchart shows the various conditions under which FSM takes a jump, which can not be controlled externally through software.

Assume a memory system which consists of *N* memory blocks where each memory block can be of same or different size. If memory blocks are named as M1, M2,..Mn with corresponding number of address locations are S1,S2...Sn then the total number of addresses available in a memory system can be written as:

$$AL = S1 + S2 + ... + Sn \quad (1)$$

where *AL* is the total number of address location available in a memory system. Now assume that the maximum size of any memory block (based on address location) inside a memory system is, **Smax** = max(S0,S1,Sn). Using memory BIST, the time required to completely zeroize the memory system would be the time required to completely erase the content of largest memory block. This is due to the fact that all memories are initializing concurrently and does not require sequential accesses. In comparison to the functional approach, the improvement in zeroization time can be expressed as,

$$Zeroization\ Speed\text{-}up = AL/Smax \quad (2)$$

We can also conclude that the zeroization time reduction factor is roughly equal to the number of clustered memories in a memory system.

IV. EXPERIMENTAL RESULTS: EXPLORING THE FEASIBILITY

This section demonstrates the efficiency of our proposed approach by validating the results on various industrial intellectual properties (IPs). The seven IPs, as shown in Table II, are practically instantiated on state-of-the-art networking SoC. The IPs are synthesized using 28nm library and guaranteed to work at their functional clock periods, typically generated using phase locked loop (PLL). Total gate and flip-flop count of each IP are shown in column 2 and 3 of Table II. The column 3 shows the PLL to system clock ratio. For example, if the system clock period is 10 ns then the *DMA Controller* would get clock pulse at a period of 1.25 ns as the PLL locking ratio is 8. The absolute zeroization time (in ns) of each IP is shown under column 5.

As discussed earlier, our main objective is to reduce the zeroization time of embedded memories, of a complex design, in case of security violation(s) has(have) been detected by the tamper detection circuits. Traditionally, functional data generators are used to sequentially initialize the memory system (to all zeros) of IPs through functional address space. In order to parallelize the memory zeroization, we can easily exploit the existing Memory BIST solution which can initialize individual memories much faster than conventional zeroization approach. The additional benefit is, we can easily remove the dedicated hardware for memory initialization and completely rely on MBIST engines for two purposes: a) zeroizing memories in case of security violation(s), b) test and repair memories for hard failures.

The zeroization run-time analysis of various IPs, whose characteristics are mentioned in Table II, is shown in Table I. The IP name is listed in column 1, where each IP consists of multiple smaller memories to form a memory system. For example, packet express buffer (PEB) IP has total memory size of 18432 KB which is built using 32 memory instances of different sizes. The number of memory instances and total

TABLE II
CHARACTERISTICS OF STANDARD INTELLECTUAL PROPERTIES AND TRADITIONAL ZEROIZATION TIME

IP name	#Gates (in million)	#Flops (in million)	PLL to clock ratio	Zeroization Time (ns)
DMA Cntrl.	0.48	0.16	8	2960
PEB	0.6	0.13	6	436906
Qbman	1.7	0.47	8	89500
DBG	1.5	0.24	4	20480
OCRAM	0.4	0.05	6	109226
CCN	4.4	1.1	8	143360
DDR Cntrl.	0.7	0.1	16	7760

memory size is shown in column 2 and 3 respectively. Column 4 is the number of functional address locations that must be accessed to initialize memory locations with all zeros. Practically, the number of cycles require to initialize memory and the number of addresses are same. That is why, column 7 has same entries as column 4 and the absolute cycle count is used to calculate the zeroization time mentioned earlier in column 5 of Table II.

As mentioned earlier, MBIST engine would start initializing the memories at the same time, however, the complete zeroization time will be limited by the largest memory in the memory cluster. In Table I, the largest memory size and corresponding address space are shown under column 5 and 6 respectively. It is worth to notice that the clock cycles requirement for zeroization is equal to the number of address locations of the largest memory, only if we choose to use existing MBIST engines for zeroization purpose. Column 9 shows the speed-up obtained using the proposed approach (shown in column 8) over the conventional zeroization technique (shown in column 7). The zeroization speed-up, as established in Equation 2, is dependent on the number of address locations of the largest memory inside the cluster. This is typically due to the reason that MBIST engine sends the *BIST_DONE* to system state machine, only once all memories are initialized.

V. CONCLUSION AND FUTURE WORK

This paper firstly highlights the importance of protecting secret data, like authentication credentials, cryptographic keys etc., stored in the volatile memories that can be hacked during normal device operations. Secondly, we have highlighted the use of dedicated hardware in practice to functionally erase the private memory contents in today's state-of-the-art SoCs, once security violation(s) is(are) detected. The only major limitation with the conventional approach is to initialize complete memory system via sequential address space. Thirdly, this has motivated us to explore and propose, for the first time, an effective technique of zeroizing the memories which is substantially faster than the traditional techniques. Additionally, the proposed approach does not uses any additional hardware circuitry as it re-uses the functionality of Memory BIST engines for zeroization process. The proposed approach supports rapid zeroization for secure and non-secure systems having larger chunks of memory. At the end, we have shown that the proposed on-chip ram zeroization technique is much faster than traditional approach. In future work, we will focus on exploiting the current zeroization concept for external memories like DRAM and SD-MMC memories, or any other off-chip memories.

VI. ACKNOWLEDGMENTS

We would like to express our gratitude for all the valuable and constructive comments received from Prof. Virendra Singh, Computer Architecture and Dependable Systems Lab (CADSL), IIT Bombay, Mumbai, India.

REFERENCES

[1] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "Lest we remember: Cold boot attacks on encryption keys," in *Proc. of 17th USENIX Security Symposium, San Jose, CA,USA*, pp. 45–60, August, 2008.

[2] S. Skorobogatov, "Low temperature data remanence in static ram," in *Tech. Report No. UCAM-CL-TR-536, Computer Laboratory, University of Cambridge*, 2002.

[3] S. Yitbarek, M.T.Aga, R.das, and T.Austin, "Cold boot attacks are still hot: Security analysis of memory scramblers in modern processors," in *Proc. of 23rd International Symposium on High Performance Computer Architecture, Austin, TX*, pp. 313–324, 2017.

[4] E. Dubrova, "Anti-tamper techniques." KTH Royal Institute of Technology, Sweden, accessed on 06 Jan 2018.

[5] D. C. Vasile and P. M. Svasta, "Temperature sensitive active tamper detection circuit," in *Proc. of 23rd International Symposium for Design and Technology in Electronic Packaging (SIITME)*, pp. 175–178, 2017.

[6] S. H. Weingart, "Physical security devices for computer subsystems : A survey of attacks and defenses," in *Lecture Notes in Computer Science, vol 1965, Cryptographic Hardware and Embedded Systems (CHES), Springer, Berlin, Heidelberg*, pp. 302–317, 2000.

[7] "Ls2080 product description and documentation." www.nxp.com/products/processors-and-microcontrollers/arm-based-processors-and-mcus/qoriq-layerscape-arm-processors, accessed 04 July, 2018.

[8] "Trustzone architecture for soc and microcontrollers," in *www.arm.com/products/security-on-arm/trustzone*.

[9] T. E. Tkacik and A. Askenazi, "Encryption apparatus with diverse key retention schemes," in *United States patent, US 8,175,276 B2*, 2009.

[10] "Software configuration guide for the cisco 5900 embedded services routers." Zeroization Chapter: Cisco Family of Router, www.cisco.com, accessed 26 May 2018.

[11] "Implementation guidance for FIPS-140-2 and the cryptographic module validation program." National Institute of Standards and Technology Communications Security Establishment, Dept. of commerce, USA, accessed 26 May 2018.

[12] S. Malliaros, C. Ntangonian, and X. Christos, "Protecting sensitive information in the volatile memory from disclosure attacks," in *Proc. of International Conference on Availability, Reliability and Security (ARES)*, pp. 687–693, 2016.

[13] W. et. al., "Area-efficient and low stand-by power 1k-byte transmission-gate-based non-imprinting high-speed erase (tnihe) sram," in *Proc. of 48th International Symposium on Circuits and Systems (ISCAS), DOI: 10.1109/ISCAS.2016.7527336*, 2016.

[14] W. et. al., "A dynamic-voltage-scaling 1kbytex8-bit non-imprinting master-slave sram with high speed erase for low-power operation," in *Proc. of 14th International Symposium on Integrated Circuits (ISIC);DOI: 10.1109/ISICIR.2014.7029479*, 2014.

[15] A.Rahmati, M. Salajegheh, W. P. Burleson, K. Fu, and J. Sorber, "Tardis: Time and remanence decay in sram to implement secure protocols on embedded devices without clocks," in *Proc. of 21st Usenix Secuirity Symposium, Bellevue, WA*, pp. 221–236, 2012.

[16] "Security techniques: "security requirements for cryptographic modules" standard, fips 140 level 3," in *National Institute of Standards and Technology, Dept. of commerce, USA,www.nist.gov*.