Context Switching Semaphore with Data Security Issues using Self-healing Approach

M. Anand
Research Scholar, Dept of ECE
St. Peters University
Chennai - 54 India

Dr. S. Ravi
Professor and Head, Dept of ECE
Dr M. G. R. University
Chennai – 95 India

Kuldeep Chouhan
Research Scholar, Dept of ECE
Dr M. G. R. University
Chennai – 95 India

Syed Musthak Ahmed
Head, Dept of ECE
S. R. Engineering College
Warangal India

***Abstract*— The main objective of a self healing scheme is to share and secure the information of any system at the same time. "Self-healing" techniques ultimately are dependable computing techniques. Specifically self-healing systems have to think for itself without human input, able to boot up backup systems. However, sharing and protection are two contradictory goals. Protection programs may be completely isolated from each other by executing them on separate non-networked computer, however, this precludes sharing.**

Keywords- Self-healing; Semaphore; Data Security.

I. INTRODUCTION

Self-healing mechanisms complement approaches that stop attacks from succeeding by preventing the injection of code, transfer of control to injected code, or misuse of existing code. Approaches to automatically defending software systems have typically focused on ways to proactively or at runtime protect an application from attack. These proactive approaches include writing the system in a "safe" language, linking the system with "safe" libraries, transforming the program with artificial diversity, or compiling the program with stack integrity checking. The technique of program shepherding is validates branch instructions to prevent transfer of control to injected code and to make sure that calls into native libraries originate from valid sources. Control Flow Integrity (CFI), observing that high-level programming often assumes properties of control flow that is not enforced at the machine level [3,4]. The use of CFI enables the efficient implementation of a software shadow call stack with strong protection guarantees. However, such techniques generally focus on integrity protection at the expense of availability. Control flow is often corrupted because input is eventually incorporated into part of an instruction's opcode, set as a jump target, or forms part of an argument to a sensitive system call.

II. SELF HEALING SYSTEMS

A. Self Healing Approach

Self-healing is an approach to detect improper operations of software applications, transactions and business processes, and then to initiate corrective action without disrupting users [8]. Healing systems that require human intervention or intervention of an agent external to the system can be categorized as assisted-healing systems. The key focus or contrasting idea as compared to dependable systems is that a self-healing system should recover from the abnormal (or unhealthy) state and return to the normative (healthy) state and function as it was prior to disruption. Some scholars treat self-healing systems as an independent one while others view as a subclass of traditional fault tolerant computing systems.

The system monitors itself for indications of anomalous behavior. When such behavior is detected, the system enters a self-diagnosis mode that aims to identify the fault and extract as much information as possible with respect to its cause, symptoms, and impact on the system [9]. The system tries to adapt itself by generating candidate fixes, which are tested to find the best target state. Self-healing systems can support decision making in a large way for managerial and organizational situations [11]. Many of the decision support systems (DSS) offer passive forms of decision support, where the decision-making process depends upon the user's initiative. Such active involvement is especially needed in complex decision-making environments.

B. Architecture of Self-Healing (S-H)

The term 'self' in self-healing architecture is referred to the action or response initiated automatically within the system. A general architecture of a self-healing system is shown in Fig. 1.

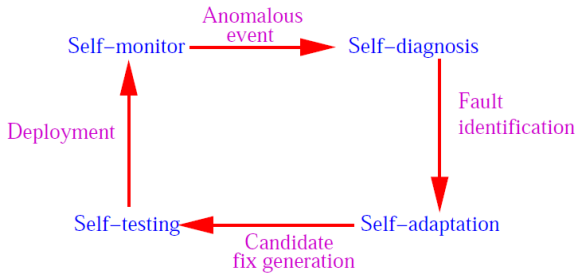

Fig. 1 General architecture of a self-healing system

C. Self Healing technique

The effective remediation strategies include failure-oblivious computing, error virtualization, rollback of memory updates, and data-structure repair [5]. These approaches may cause a semantically incorrect continuation of execution attempts to address this difficulty by exploring semantically safe alterations of the program's environment. The technique is subsequently introduced in a modified form as failure-oblivious computing, because the program code is extensively rewritten to include the necessary checks for every memory access, the system incurs overheads for a variety of different applications. Data-structure Repair is the most critical concerns with recovering from software faults and vulnerability exploits is ensuring the consistency and correctness of program data and state.

D. Why Self Healing Systems

The software notoriously buggy and crash-prone is despite considerable work in the fault tolerance and reliability [1]. The current approach to ensuring the security and availability of software consists of a mix of different techniques:

a) ***Proactive techniques*****:** seek to make the code as dependable as possible, through a combination of safe languages, libraries and compilers, code analysis tools, formal methods and development methodologies.

b) ***Debugging techniques*****:** aim to make post-fault analysis and recovery as easy as possible for the programmer that is responsible for producing a fix.

c) ***Runtime protection techniques*****:** try to detect the fault using some type of fault isolation, which address specific types of faults or security vulnerabilities.

d) ***Containment techniques*****:** seek to minimize the scope of a successful exploit by isolating the process from the rest of the system, *e.g.*, through use of virtual machine.

e) ***Byzantine fault-tolerance and quorum techniques*****:** rely on redundancy and diversity to create reliable systems out of unreliable components.

E. Elements of Self-Healing model

In the Self-healing process model, there are different categories of aspects to the self-healing system,

a) ***Fault model*****:** Self-healing systems have the tenets of dependable computing is that called a fault model must be specified for any fault tolerant system. The fault model answers the question of what faults the system is to tolerate. Self-healing systems have a fault model in terms of what injuries (faults), which are expected to be able to self-heal.

b) ***Fault duration*****:** Faults can be permanent, intermittent or transient due to an environmental condition. It is important to state the fault duration assumption of a self-healing approach to understand what situations it addresses.

c) ***Fault manifestation*****:** The severity of the fault manifestation, it affects the system in the absence of a self-healing response. The faults cause immediate system crashes, but, many faults cause less catastrophic consequences, such as system slow-down due to excessive CPU loads, thrashing due to memory hierarchy overloads, resource leakage, file system overflow, and so on.

d) ***Fault source*****:** Thee source of faults can affect self-healing strategies due to implementation defects, requirements defects, operational mistakes, and etc. Self-healing software is designed only to withstand hardware failures such as loss of memory [6] or CPU capacity and not software failures.

e) ***Granularity*****:** The granularity of a failure is the size of the component that is compromised by that fault. Different self-healing mechanisms are probably appropriate depending on the granularity of the failures [7] and hence, the granularity of recovery actions.

f) ***Fault profile expectations*****:** The source of the fault is the profile of fault occurrences that is expected. It considered for self-healing might be only expected faults that is based on design analysis or faults that are unexpected [12]. Additionally, faults might be random and independent, might be correlated in space or time, or might even be intentional due to malicious intent.

F. Conventional Methods of Security

The conventional methods can overcome only the effects of passive threats and not the active threats for the authenticate users. They reduce user-friendliness and also, the amount of OS resources required to provide security is high. Different protocol architectures are used for providing security for each layer of the OSI model and it may not be generic. Alternately, in this work, the information is allowed to flow freely through the fetch and decode cycles while an access or authentication is made only between the decode and execute cycle before the data is permanently written into the memory by the user (if authenticated). This is shown in Fig. 2.

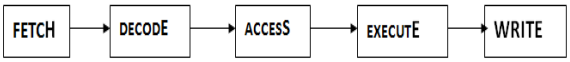

Fig. 2 Process of Execution

G. Proposed features of Security issues

The proposed hardware is shown in Figure 3. The features of the proposed hardware are that it is (i) PCI compliant and (ii) Mounted in a single chip

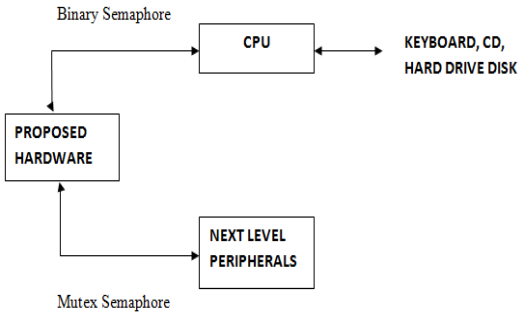

Fig. 3 Hardware Process System

a) ***Robustness*:** It provides defence against vulnerabilities with few false positives or false negatives.

b) ***Flexible*:** It adapts easily to cover the continuously evolving threats.

c) ***End-to-End*:** The security policy flows throughout all the seven layers of the OSI model.

d) ***Scalable*:** It can co-exist with the existing circuitry without any modification [2].

III. METHODOLOGY

A. Semaphores and Resource Sharing

A semaphore is a protected variable or an abstract data type which restricts the access to shared resources such as shared memory in a multiprogramming environment. It is a primitive synchronization mechanism for sharing CPU time and resources. It is a classic solution to prevent race conditions.

B. Operation of Semaphore

The 'value' of a semaphore is the number of units of resources which are free. To avoid busy-waiting, a semaphore has an associated queue of processes (usually First in First Out). If a process performs a 'P' operation on a semaphore which has the value zero, the process is added to the semaphore's queue. When another process increments the semaphore by performing a 'V' operation, and there are processes on the queue, one of them is removed from the queue and resumes operation.

C. Binary Semaphore

In binary semaphore, if there is only one resource, the semaphore takes value '0' or '1'.This is explained in Fig. 4.

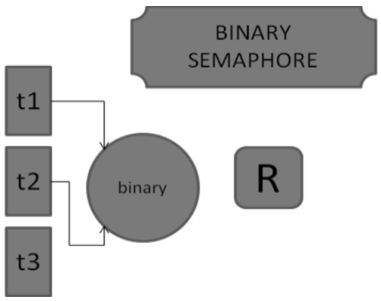

Fig. 4 Binary Semaphore

Suppose a 'P' operation busy-waits (uses its turn to do nothing) or maybe sleeps (tells the system not to give it a turn) until a resource is available, where upon it immediately claims one. Now, let 'V' be the operation that simply makes a resource available again after the process has finished using it. The 'P' and 'V' operations must be atomic, i.e., no process may be preempted in the middle of one of those operations to run another operation on the same semaphore. When a semaphore is being used, it takes value '0' and when it takes the value '1', the process directly starts execution without waiting.

D. Counting Semaphore

The counting semaphore concept can be extended with the ability of claiming or returning more than one unit from the semaphore. When multiple resources are to be shared by many operations, such a semaphore is used. All the resources must be of the same type. This is shown in Fig. 5.

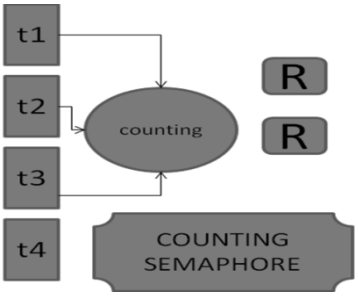

Fig. 5 Counting Semaphore

In this, the initial value of semaphore is set equal to number of resources available. As the semaphores are being used, the value keeps decrementing. As the semaphores are being released, after use, its value keeps incrementing. 'Zero' value refers to empty semaphore.

E. Mutex Semaphore

A mutex is a binary semaphore with extra features like ownership or priority inversion protection. Mutexes are meant to be used for mutual exclusion only. This is shown in Figure 6.

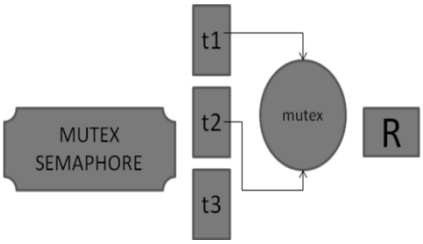

Fig. 6 Mutex Semaphore

Initially the semaphore value is set to zero. Once a task attains ownership, it can access the resources as many times as it wants and each time, it accesses, the semaphore value increases.

F. Characteristics of Semaphore

The characteristics of the three semaphores are shown in TABLE I.

TABLE I. CHARACTERISTICS OF THREE SEMAPHORES

Binary Semaphore	Counting Semaphore	Mutex Semaphore
Anyone can release the semaphore. Used for mutual exclusion and event notification.	Only after a task has attained access, it can release the semaphore.	Only the owner can release the semaphore. Used only for mutual exclusion.

IV. RESULTS AND DISCUSSION

A. Implementation

In the existing architecture of computers, the bottleneck of connecting a high speed low memory device to a low speed high memory device is solved by using an intermediate memory module called the cache. The cache exists between the high speed CPU and the lower speed memories. However, there is no security between the data link layer of the CPU and the cache. The proposed card is placed on the PCI bus at the maximum possible speed and a direct connection is established to the CPU via snooping. To understand how the security layer is to be implemented, the knowledge of the three basic terms is required: subject, object and capability. Subject refers to the user or entity which acts on behalf of the user on the system. Objects may be defined as resources within the system. The main term however is capability which is basically a 'token'. The possession of a capability by a subject confers access right for an object. They cannot be easily modified, but they can be reproduced.

The capabilities of the objects are to be stored in the non-readable section of the HDD. For a subject to access a particular object, it must possess the capability for doing so. Hence, before a subject accesses a resource via the CPU, it will first go through a screening check from the hardware on whether or not its capabilities allow it to access such resources. Hence, a security layer is now added to the data link layer between the CPU and the cache. Additionally, user also must be prevented from creating arbitrary capabilities. This can be accomplished by placing the capabilities in special 'Capability Segments' which users cannot access. Another approach is to add a tag bit to each primary storage location. This bit, inaccessible to the user is 'ON' if the location contains a capability. It should be noted that the hardware restricts the manipulation of the location contents to appropriate system routines. If the last remaining capability is destroyed, then that object cannot be used in any manner. In this work, special provisions are made for controlling the copying and movement of capabilities (as well as interpretation) depending on the hardware involved.

B. Evolved Function is a Semaphore Selector

The schematic of the control block performing the evolved function is shown in Fig. 7.

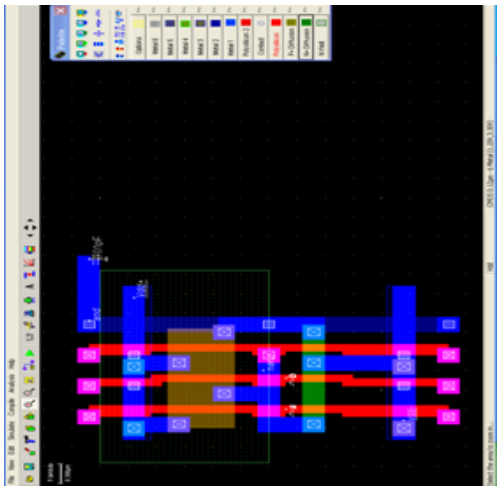

Fig. 7 Schematic representation of Semaphore Selector

The power consumed by the block under read and write mode is shown in Fig. 8 and Fig. 9 respectively.

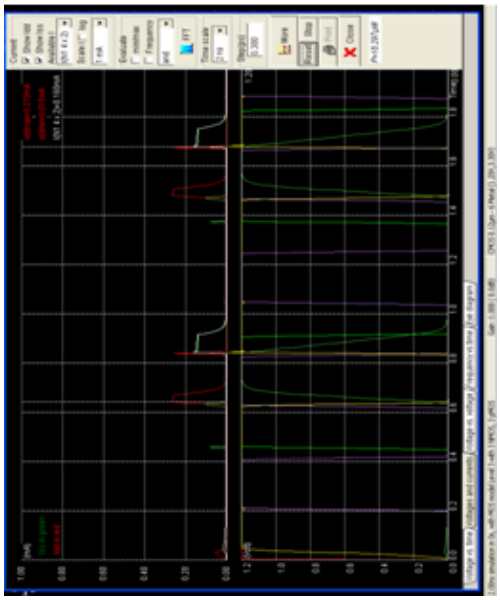

Fig. 8 Current and Voltage Variations of semaphore selector during read cycle

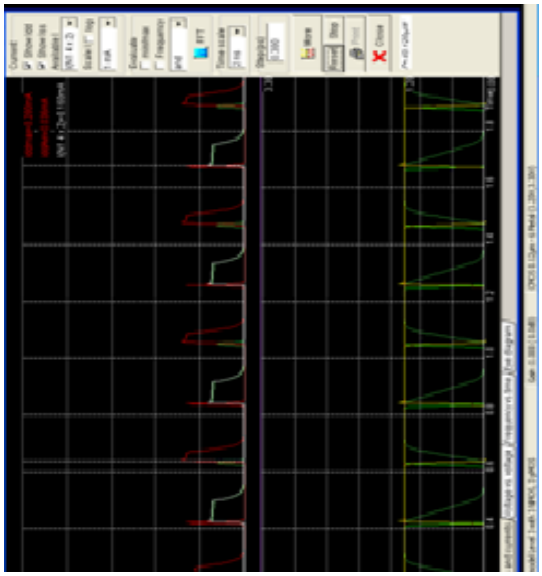

Fig. 9 Current and Voltage Variations of Semaphore selector during write cycle

C. Evolved Function is a Resource Sharing Selector

The control block performing the evolved function of "resource sharing selector*"* along with its power consumtion is shown in Fig. 10. Similar graph corresponding to read and write cycle is shown in Fig. 11 and Fig. 12 respectively.

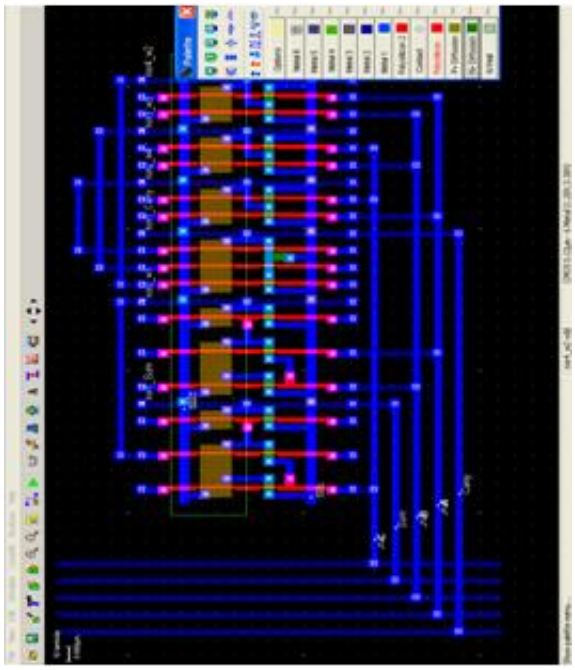

Fig. 10 Schematic of Evolved resource sharing selector

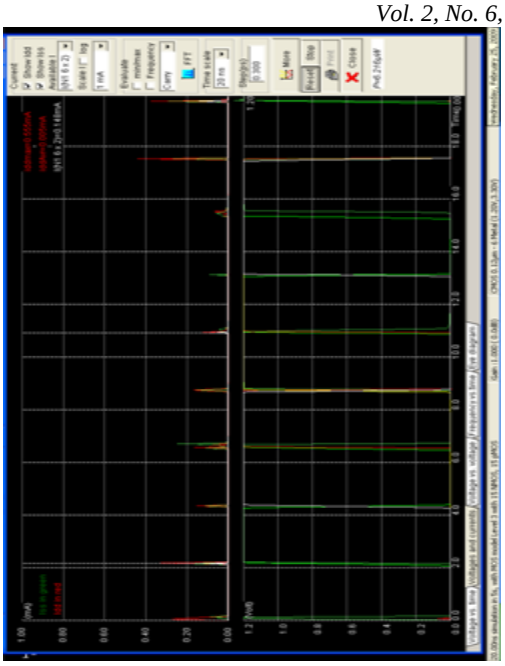

Fig. 11 Current and Voltage Variations of evolved resource sharing selector during Read cycle

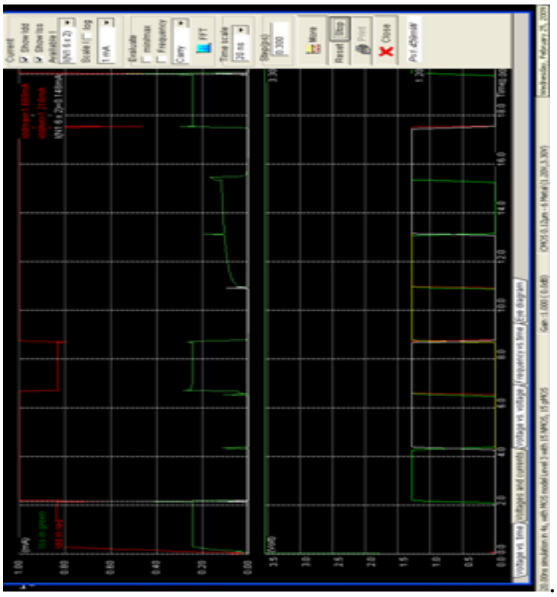

Fig. 12 Current and Voltage Variations of evolved resource sharing selector during write cycle

D. Evolved Function is a Snoop Selector Function

The schematic of the control block performing the evolved function of a snoop selector circuit is shown in Fig. 13. The power consumed by the PE under read and write cycles is shown in Fig. 14 and Fig. 15 respectively.

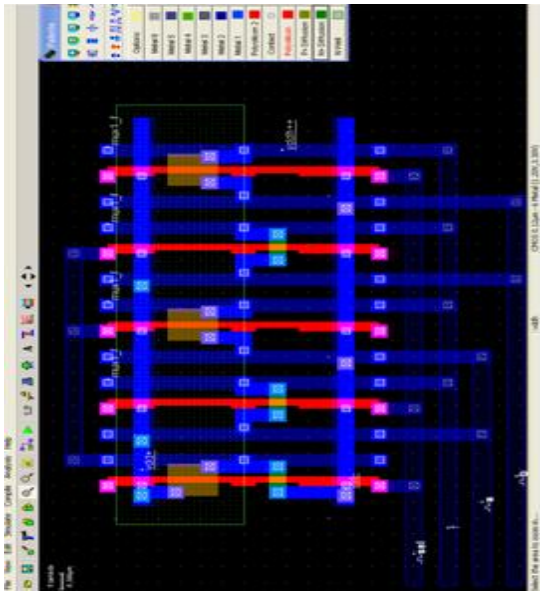

Fig. 13 Schematic of Evolved Snoop selector circuit

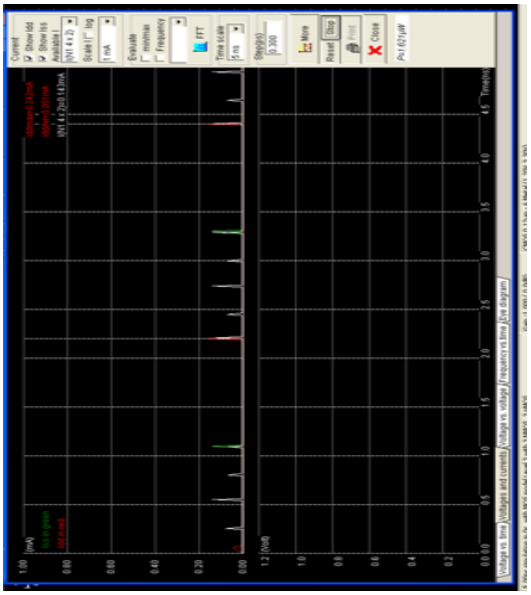

Fig. 14 Current and Voltage Variations of evolved Snoop selector during read
Cycle

E. Evolved Function is a Context Switching Semaphore

The schematic of Context switching semaphore block of the evolved function is shown in Fig. 16. The power consumed by the PE during read and write cycle is shown in Fig. 17 and Fig. 18 respectively

Fig. 15 Current and Voltage Variations of evolved snoop selector during write
cycle

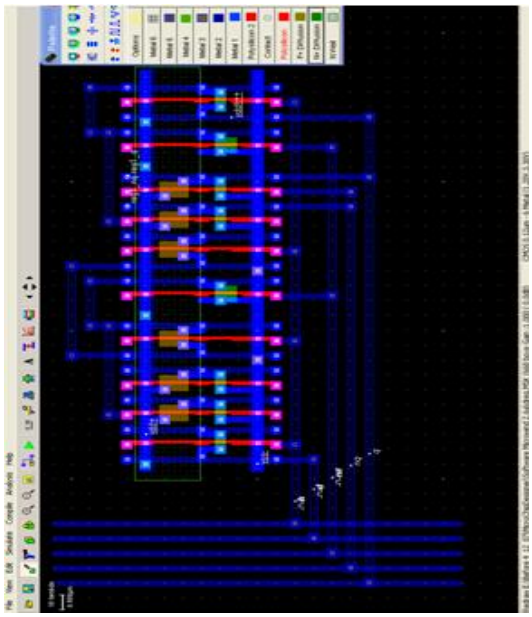

Fig. 16 Schematic of Evolved Context switching semaphore

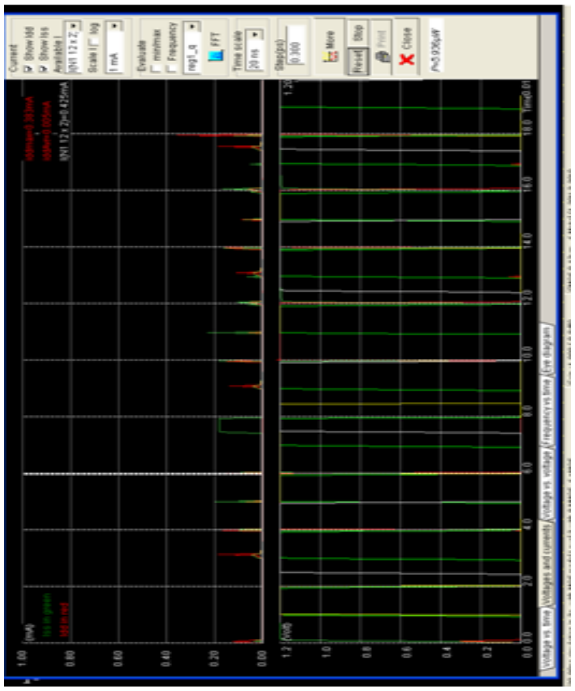

Fig. 17 Current and Voltage Variations of evolved Context switching semaphore during read cycle

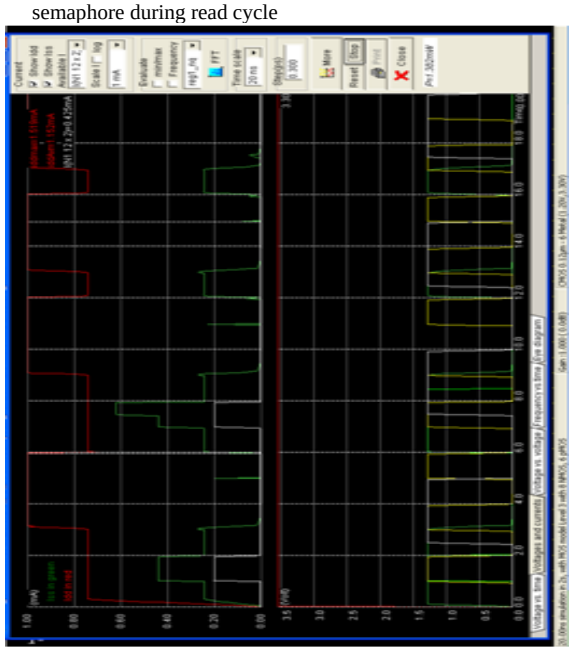

Fig. 18 Current and Voltage Variations of evolved Context switching semaphore during write cycle

The results of the discussion are tabulated in TABLE II. It can be inferred from the table that a power level variation exists between the read and write cycles.

TABLE II. POWER CONSUMED BY EVOLVED PE

Evolved Function of PE	**Average Power Consumed by Evolved blocks**	
	read	**write**
Semaphore selector	.3mW	2.325mW
Resource sharing selector	.54mW	3.5mW
Snoop selector	.12mW	3.665mW
Context switching semaphore	.456mW	3.5mW

V. CONCLUSION

In this work, self-healing systems prove increasingly important in countering system software based attacks, which recover and secure to the data from interrupted services. Self-healing systems offer an active form of decision support, without human intervention that can detect the fault and recover from the fault. Also, with intelligent architectural models, a self-healing system can select the proper repair plan to deploy the broken component, if there is more than one component that needs to be healed, can prioritize a fault component over the others, etc.

ACKNOWLEDGMENT

This work was supported in part by Micro Logic System, Chennai-600017.

REFERENCES

[1] Bouricius, W.G., Carter, W.C. & Schneider, P.R, "Reliability modeling techniques for self-repairing computer systems", in proceedings of 24th National Conference, ACM, 1969, pp. 395-309.

[2] Shelton, C., Koopman, P.&Nace, W., "A framework for scalable analysis and design of system-wide graceful degradation in distributed embedded systems", WORDS03, January 2003.

[3] G. Edward Suh, J. W. Lee, D. Zhang, and S. Devadas, "Secure Program Execution via Dynamic Information Flow Tracking", in proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-XI), October 2004.

[4] Martin Abadi, Mihai Budiu, Ulfar Erlingsson, and Jay Ligatti, "Control-Flow Integrity: Principles, Implementations, and Applications", in proceedings of the ACM Conference on Computer and Communications Security (CCS), 2005.

[5] B. Demsky and M. C. Rinard, "Automatic Detection and Repair of Errors in Data Structures", in proceedings of ACM OOPSLA, October 2003.

[6] M. Locasto, S. Sidiroglou, and A.D Keromytis, "Software Self-Healing Using Collaborative Application Communities", in proceedings of the Internet Society (ISOC) Symposium on Network and Distributed Systems Security (SNDSS), February 2006.

[7] J. Kong, X. Hong, J.-S. Park, Y. Yi, and M. Gerla, "L'Hospital: Self-healing Secure Routing for Mobile Ad-hoc Networks", in Technical Report CSD-TR040055, Dept. of Computer Science, UCLA, January 2005.

[8] Michael E. Shin and Jung Hoon An, "Self-Reconfiguration in Self-Healing Systems", in proceedings of the Third IEEE International Workshop on EASE'06, pp 89-98 (2006).

[9] E. M. Dashofy, A. V. D. Hoek, and R. N. Taylor, "Towards architecture-based self-healing systems", in p*roceedings of the first workshop on Self-healing systems*, Charleston, South Carolina, pp. 21-26, 2002.

[10] H.You, V.Vittal, Z.Yang, "Self-healing in power systems: an approach using islanding and rate of frequency decline based load shedding", IEEE Transaction on Power System, Vol.18, No.1, 2003, pp.174 -181.

[11] T.A. Ramesh Kumaar and Dr.I.A.Chidambaram, "Self-Healing Strategy for Dynamic Security Assessment and Power System

Restoration", in International Journal of Computer Science & Emerging Technologies, (E-ISSN: 2044-6004) Vol. 2, Issue 2, April 2011.

[12] J. Newsome and D. X. Song, "Dynamic Taint Analysis, and Signature Generation of Exploits on Commodity Software", in the Proceedings of the Network and Distributed System Security Symposium, San Diego,
CA, Feb.2005

AUTHORS PROFILE

1. Mr. M. Anand has completed his B.Tech. in the faculty of ECE and M.Tech. in the area of Applied Electronics. He is presently a research scholar in the faculty of ECE at St. Peters University, Avadi, Chennai. Mr. M.Anand shows interest in the areas of Embedded system design, Co-design of embedded systems, Scheduling etc.

2. Dr. S. Ravi was born on July 24th, 1971 at Chennai, India and got A.M.I.E. (Electronics Engineering) from Institution of Engineers, Calcutta, M.Tech. (Communication systems) from Anna University, Chennai in the Year 1994 and Ph.D. from Anna University in the year 2003. He has got 20 Years of Teaching Experience and 3 Years of Industrial Experience. Presently, he is working as Professor and Head of the Department of Electronics Engineering in Dr. M.G.R. University, Chennai. He has so far published nearly thirty four papers in referred International Journals and successfully guided four Ph.D. scholars. Dr. S. Ravi is a Life Member IETE and Life Member of Indian Society for Technical Education

3. Mr. Kuldeep Chouhan has completed his M.Tech. in the department of Computer Science and Engineering (CSE). He is presently a research scholar in the faculty of ECE at Dr. MGR University, Chennai. His area of interest is Wireless Sensor Network, Embedded system, MultiAgent System in Sensor Security

4. Dr. Syed Musthak Ahmed, completed his BE(Electronics) from Bangalore University, ME(Electronics), UVCE Bangalore, Bangalore University, Ph.D.(ECE), Vinayaka Missions University, Salem. He has 25 years of teaching experience working in reputed engineering colleges and is presently working as Professor and HOD (ECE), SR Engineering College, Warangal. He is a member of Various Professional Societies like SMIEEE, FIETE, MISSS, MISTE, MIAENG, MIAMT. He has various publications in National and International Journals / Conferences