Reconfiguration-Based VLSI Design for Security

Bao Liu and Brandon Wang

Abstract—**Reconfigurable computing is a critical technology for achieving nanoelectronic systems of yield and reliability. In this paper, we present that reconfigurable computing is further a critical technology for achieving hardware security in the presence of supply chain adversaries. Specifically, reconfigurable implementation of a given logic function achieves design obfuscation, while reconfiguration for difference logic functions further achieves moving target defense. We further present reconfigurable reversible computing-based cryptography, and a generic reconfiguration-based VLSI design-for-security methodology. In our case studies based on a SPARC V8 LEON2 processor, we prevent software- or hardware-based code injection attacks at cost of 0.72% area increase, negligible power consumption increase and no performance degradation; we further prevent a hardware Trojan from gaining unauthorized memory access at cost of 4.42% area increase, negligible power consumption increase, and 11.30% critical path delay increase.**

Index Terms—**Carbon nanotubes, computer security, cryptography, integrated circuits, nanoelectronics, security.**

I. Introduction

A critical challenge for nanoelectronic systems is to achieve yield and reliability. As very-large-scale integration (VLSI) technology scales into the nanometer scale, devices and interconnects are subject to increasingly prevalent defects and significant parametric variations. Based on photolithography, we are making layout features of smaller dimensions than the wavelength of the light, which requires increasingly complex OPC and other DFM techniques at increasing layout area cost. Future nanoelectronic systems are expected to be based on self-assembly manufacture of a regular physical structure [22], [30], and achieve functionality by reconfiguration [55]. Reconfiguration is further critical for nanoelectronic systems to achieve yield and reliability by bypassing defective or degraded devices and interconnects, which occurrence cannot be avoided or reduced below a certain level as is determined by the uncertainly principle of quantum physics [38].

In this paper, we present that reconfigurable computing is further a critical technology to achieve hardware security in the presence of supply chain adversaries.

Manuscript received April 05, 2014; revised July 20, 2014; accepted August 04, 2014. Date of publication February 09, 2015; date of current version March 09, 2015. This paper was recommended by Guest Editor S. Bhunia.

B. Liu is with the Electrical and Computer Engineering Department, University of Texas, San Antonio, TX 78249 USA. (e-mail: bliu@utsa.edu).

B. Wang is with Cadence Design Systems, Inc., San Jose, CA 95134 USA. (e-mail: bwang@cadence.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JETCAS.2014.2372431

Hardware is the foundation and the root of trust of any security system. In recent years, a growing number of software-based security solutions have been migrated to hardware-based security solutions for much enhanced resistance to software-based security threats. Such systems range from smartcards to specialized secure co-processing boxes, wherein hardware provides the source of security and trust for a number of security primitives [4], [36], [37], [42], [56], [59].

However, in recent years, it has been brought into light that hardware is also subject to a number of security threats. The existing techniques mostly focus on information leak from a hardware system: An adversary may extract cryptographic keys and confidential information from a system by testing [2], [63], reverse engineering [20], or side-channel analysis [10], [32], [33], [61].

More critical threats come from the supply chain and compromise hardware integrity. In today's global IC industry, a supply chain adversary, such as an IP provider, an IC design house, a CAD company, or a foundry may have access to the source code of the design, and may easily tamper a hardware system by planting time bombs which compromise hardware computation integrity, or creating back doors which enable information leak, bypassing access control mechanisms at higher (e.g., OS and application) levels [29]. The recently-released Comprehensive National Cyber Security Initiative has identified this supply chain risk management problem as a top national priority [44].

A supply chain adversary's capability is rooted in his knowledge on the hardware design. Successful hardware design obfuscation would severely limit a supply chain adversary's capability if not preventing all supply chain attacks. However, not all designs are obfuscatable in traditional technologies. In this paper, we propose to achieve VLSI design obfuscation by reconfigurable implementation of a given logic function, which is determined by the end user and unknown to any party in the supply chain. We further propose to achieve moving target defense in VLSI design by reconfiguration for different logic functions. Moving target defense has been proposed against software-based attacks. Obfuscated implementation of a moving target defense scheme further prevents a supply chain adversary or a hardware Trojan from tampering or gaining knowledge on the scheme and launching an attack without being detected. We further propose reconfigurable reversible computing (RRC)-based cryptography and present a generic reconfiguration-based supply chain risk management methodology. In our case studies based on a LEON2 processor, we prevent software- or hardware-based code injection attacks at cost of 0.72% area increase, negligible power consumption increase and no performance degradation; we further prevent a hardware Trojan from gaining unauthorized memory access at cost of 4.42% area increase, negligible power consumption increase, and 11.30%

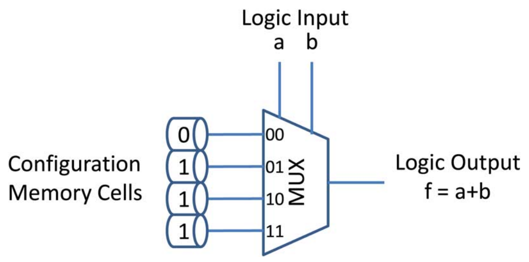

Fig. 1. LUT-based reconfigurable logic as in FPGA.

critical path delay increase. In comparison, the existing processor protection techniques against supply chain attacks either lack an obfuscation foundation so that the protection scheme is subject to attack [60] or base on certain logic encryption technique which does not achieve obfuscation [47].

The rest of this paper is organized as follows. We present a supply chain adversary attack model, and analyze the existing VLSI obfuscation techniques as well as the theoretical obfuscation results in Section II. We propose reconfiguration-based VLSI obfuscation, reconfiguration-based VLSI moving target defense, reconfigurable reversible computation (RRC)-based cryptography, and a generic reconfiguration-based VLSI design-for-security methodology in Section III. We present two case studies based on an open source SPARC V8 processor LEON2 in Section IV, and conclude in Section V.

II. Background

A. *Reconfigurable Computing Technologies*

A hardware system is reconfigurable if an user can direct the system to perform different computations by writing to the memory of the system. In a broad sense, a general-purpose computer is reconfigurable in that an user can direct the system to perform different computations by writing to the instruction memory. While for reconfigurable computing most people today refer to programmable logic devices (PLD) such as field programmable gate arrays (FPGA), where a configuration memory directs an array of configuration multiplexers for different interconnect structures and logic functions. Typical technology of today for a reconfigurable logic function is based on a lookup table, including a 2^n-to-1 multiplexer and 2^n configuration memory cells for a n-input logic function (Fig. 1), while alternative reconfigurable computing fabrics have been proposed for emerging technologies [40].

A configuration memory can be based on one-time programmable anti-fuses, SRAM, flash memory, or a number of emerging technologies such as STT RAM [45], carbon nanotube transistors [39], or molecular electronic devices [46]. Reconfiguration is critical for a computing system in an emerging technology to achieve yield and reliability.

B. *Supply Chain Attack Model*

A supply chain adversary is an insider who is involved in the design and manufacturing of a hardware device. His tamper capability is based on his role in the supply chain, specifically, his read and write permission in the design and the manufacturing process of a specific device. An IP provider or a designer for a specific module may have limited access to the design, while a foundry or a chip-level integration designer has access to the whole device design. The general lack of access control in today's supply chain further facilitates an adversary to gain knowledge of a design and launch attacks. Besides based on his role in the supply chain, a supply chain adversary may gain further knowledge of a design by probing, testing, side-channel analysis, or reverse engineering. For example, commercial tools are available to help convert a layout to a gate-level netlist, and further convert a gate-level netlist to a higher-level abstraction. To locate an adder in a gate-level netlist, one can resynthesize the logic based on a revised cell library which includes a zero-cost adder cell. As a result, a supply chain adversary (e.g., in a foundry) may have read and write permission to the whole design of a particular device.

A supply chain adversary may install a hardware Trojan that is triggered at system runtime. A hardware Trojan can be a logic bomb that compromises hardware computation integrity by altering the authentic computation result, or a back door that compromises hardware data confidentiality by leaking out secrets or confidential information. A back door may launch an attack by performing more (e.g., to leak out information) or less (e.g., to bypass existing security checks) than expected, while keeping the authentic computation results intact. Such a back door cannot be detected by testing or concurrent checking of the computation results.

C. *VLSI Encryption/Locking*

The state-of-the-art VLSI logic encryption/locking techniques include combinational logic locking and finite-state machine (FSM) locking. Combinational logic locking augments a combinational logic network with an additional group of lock inputs such that the augmented combinational logic network has the same function as the original combinational logic network only if a specific vector (aka a valid key) is applied to the lock inputs [50]. This is achieved, for example, by inserting a group of XOR logic gates or LUTs [9] or an additional logic cone [18] to the combinational logic network. FSM locking augments an FSM by introducing a group of extra finite states, which form an obfuscated mode. Only a correct sequence of inputs transit the FSM out of the obfuscated mode and set the FSM to the correct initial state in the normal operation mode [3], [17]–[19], [21].

These techniques hide combinational logic or FSM functionality by introducing additional inputs or finite states. While they succeed in preventing an adversary from operating a hardware system, these techniques cannot guarantee to prevent a supply chain adversary from understanding and tampering a VLSI design. For example, an adversary can understand and tamper the design once he has the logic encryption/locking key. An adversary at a foundry may gain knowledge of the key if the foundry needs to receive a key and activates an IC to perform manufacturing test. Or, an adversary having knowledge of the function of the chip can obtain the key or remove the lock. Such an adversary can be anyone having an activated IC in his possession. We assume that the adversary has the knowledge of the design, for example, by reverse engineering, and has the capability to test the chip, e.g., by applying stimuli and observing responses.

We categorize the combinational logic encryption/locking techniques as follows.

XOR/XNOR-Based: The simplest combinational logic locking technique is to insert XOR and XNOR gates into a combination logic network [9], [50]. An adversary knows which inputs are functional inputs and which inputs are lock inputs. He can then identify the lock gates connected to the lock inputs. If a total of M lock gates are inserted in a combinational logic network, the complexity for an adversary to find the correct logic may not be 2^M. For example, for a logic output, if its fanin cone contains m_i lock gates, the complexity to find the correct logic function for that logic output is at most 2^{m_i}.

MUX-Based: Another combinational logic locking technique is to insert multiplexers or combine logic functions based on Shannon expansion [18]. The reason is as follows. If a lock input is connected to a lock gate that is not a XOR or XNOR gate, the key to the lock input is implied to be the noncontrolling logic value of the lock gate. An adversary can then easily obtain the key, unless the lock input is connected to multiple lock gates and is implied to have conflicting logic values—for example, the lock input is connected to a group of AND gates and a OR gate which have the same function as a XOR or XNOR gate.[1] A more general design paradigm is to have a lock input connected to multiple logic cones, which provide different logic functions. These logic cones then need to be combined by multiplexers or based on Shannon expansion at the logic output. The select signal needs to be provided by the lock inputs.

For example, one can partition a combinational logic network into 2^n sub-networks, where n is the number of lock inputs, and combine the sub-networks by a multiplexer with the lock inputs providing the select signal. Each sub-network needs to take all the logic inputs, otherwise, an adversary can identify that sub-network is not valid. This increases the size of the combinational logic network exponentially.

For cost reduction, the sub-networks may share common logic cones. This gives the next paradigm.

Permutation-Based: An extension of the previous paradigm is to permute the logic inputs and the logic outputs (e.g., in [51]). A permutation logic block is a group of multiplexers, wherein each output is given by any of the inputs based on the select signal. A further extension is to cut a combinational logic network into two parts, and permute the signals crossing the cut line.

Reconfigurable Logic Barrier-Based: Another technique cuts a combinational logic network into two parts with all the inputs in one part and all the outputs in another part, and implements all the gates in the cut line in reconfigurable logic, i.e., forming a reconfigurable logic barrier [9].

An adversary has the following techniques to unlock a locked combinational logic network.

Key Propagation: An adversary may apply an input vector and propagate a bit in the key to a logic output based on an ATPG algorithm [48]. In case that propagating a key bit to a primary output depends on some other key bits, an ATPG algorithm can be applied to find the logic value of an internal node in the netlist which is a function of the interdependent key bits.

[1]Or, the gate is hidden in reconfigurable logic as in [9].

Path Analysis: Similarly, in testing, an adversary flips one bit at the logic input, and observes a flipped bit at the logic output. He then finds the signal propagation path(s) in the combinational logic network. The inversion of the signal propagation path must match the inversion between the flipped logic input and the flipped logic output. If there is a single XOR or XNOR gate in the signal propagation path, the lock input or the side input of the XOR or XNOR gate is determined by the inversion of the path. If there are multiple XOR or XNOR gates in the signal propagation paths, the adversary needs to find more signal propagation paths to determine the logic values of the lock inputs. If there is a multiplexer in a signal propagation path, the select signal is implied by the signal propagation path. If multiple signal propagation paths cross a multiplexer, any signal propagation path of an incorrect inversion can be eliminated. A logic network with permuted inputs and outputs [51] may not be unlocked by this technique if the adversary cannot observe any internal signal, which can be alternatively unlocked by graph isomorphism analysis as follows.

Graph Isomorphism: Since an adversary knows the function of the logic network (e.g., by testing), he can synthesize a combinational logic network of the same function based on the same cell library, compare with the locked combinational logic network, and find any isomorphic graph. An isomorphic graph can be found by, for example, first identifying the node with the largest degree of connection, and proceeding to its neighboring nodes and so on (e.g., as in formal verification [14]). This technique is capable to unlock any locked combinational logic network with XOR/XNOR gates, multiplexers, permutation blocks, or reconfigurable logic barriers.

For a reconfigurable logic barrier [9], graph isomorphism analysis gives the function of the reconfigurable logic barrier. Although the key may still be unknown, an adversary can proceed to reconfigure the logic barrier for the same logic function, which will remove the lock from the design.

Based on these combinational logic unlocking techniques, an adversary can unlock a locked FSM similarly. A locked FSM has two separate sets of finite states in the obfuscated state space and the normal state space respectively, only the correct sequence of passwords transits the FSM to the normal state space [3], [17]–[19], [21]. To de-obfuscate a locked FSM is to find the finite state encodings in the normal state space. This can be achieved by applying key propagation [48].

D. Theoretical Results on Obfuscation

Obfuscation is a long-standing problem in computer security and cryptography. To obfuscate a function f is to create an implementation of f that reveals nothing about f except its input–output behavior. Intuitively, a circuit obfuscator $\mathcal{O}$ is an efficient algorithm that, given a circuit C implementing some function f, outputs another circuit $\mathcal{O}(C)$ such that 1) (preserving functionality) it computes (perhaps approximately) the same function as f, 2) (polynomial slowdown) its size is within a polynomial factor of c, and, 3) ("virtual black-box" property) for any efficient adversary that computes some predicate on $\mathcal{O}(C)$, there exists an efficient simulator that computes the same predicate with black-box access to an oracle that evaluates f [5], [16], [43].

Software and hardware design obfuscation has long been studied as a potentially powerful tool against design tamper. Recent theoretical studies show that, 1) there exist functions that cannot be obfuscated, and 2) there exist functions that can be obfuscated. Barak *et al.* showed the existence of (contrived) classes of functions which are not obfuscatable, or, a general purpose obfuscator does not exist [5]. Goldwasser and Kalai showed that there exist many natural classes of functions that cannot be obfuscated with respect to auxiliary input (intuitively, one may think auxiliary input as the history or previous executions of the circuit) [26]. In contrast, Hohenberger *et al.* presented an obfuscation scheme of a public key-based re-encryption program [27]. Besides, the only positive obfuscation result is of point functions, which are Boolean functions that return 1 on exactly one input, for example, a password check program. Canetti and Wee separately showed how to obfuscate a point function based on a random oracle, e.g., a hash function that hides all details [16], [62]. An obfuscated point function queries the random oracle on an input, and compares the answer with a stored value. For example, a password check program encrypts an input, and compares the encryption result with a stored value, which is an encrypted password. As a result, it achieves the virtual black box property of obfuscation. This scheme is based on a weaker definition of obfuscation, which says that there is a negligible probability to distinguish an adversary circuit based on the obfuscated scheme and a simulator based on a black box of the function. As a result, this obfuscation scheme of point functions cannot be extended to obfuscate arbitrary Boolean functions [43], except some specific classes of functions such as d-CNFs [13]. The latest theoretical obfuscation research has been on weaker definitions of obfuscation, e.g., "indistinguishability obfuscation" or "best-possible obfuscation" [24], and "extractability obfuscation" [12] or "differing-inputs obfuscation" [6], i.e., such an obfuscator has the property that if an adversary can distinguish between obfuscations of two programs, then he must "know" an input on which they differ.

E. Access Control-Based VLSI Obfuscation

Although a generic obfuscator does not exist, VLSI obfuscation can be achieved based on access control. In such an IC design and manufacturing process, an untrusted party may gain knowledge on the function of a module, e.g., based on his knowledge on the rest of the system, while his restricted read/write permission gives him only a black box access to the specific module. For example, a hardware system may consist of several chips. An untrusted foundry has no read/write access to and cannot tamper a separate chip that is manufactured by a trusted foundry. Specifically, the traditional secure co-processors are manufactured by a trusted foundry on a separate chip after being designed by a trusted design house. Another example is that in 3-D IC technology, a passive silicon interposer containing only interconnects can be manufactured at an untrusted foundry, while the logic chips are manufactured at a trusted foundry and mounted on the interposer at a later stage [15]. As a result, the untrusted foundry has only black box access to the logic chips and cannot tamper them.

Obfuscation of part of a system does not lead to obfuscation of the entire system without significant cost. For example, leading fabless semiconductor companies such as AMD and research agencies such as Intelligence Advanced Research Projects Agency (IARPA) have proposed split manufacturing for VLSI layout obfuscation to prevent an untrusted foundry from tamping VLSI layout [49]. In split manufacturing, the logic gates and the interconnects at the lower metal layers are manufactured separately with the interconnects at the higher metal layers. In a traditional semi-customized IC manufacturing technology, while the logic gates and the interconnects at the lower metal layers are mass produced, the interconnects at the higher metal layers are customized at a later stage. In 3-D IC technology, the interconnects at the higher metal layers can be implemented on a separate die. Tezzaron, a leading 3-D integration capability provider, has published a white paper stating that "A multi-layer circuit may be divided among the layers in such a way that the function of each layer becomes obscure. Assuming that the TSV connections are extremely fine and abundant, elements can be scattered among the layers in apparently random fashion [57]." Imeson *et al.* have proven the complexity for an adversary to re-construct a gate-level netlist by finding an isomorphic subgraph after a subset of the interconnects are lifted onto the higher metal layers in a semi-customized technology or a separate die in 3-D IC technology and become obscure [28]. A significant percentage of all the interconnects (e.g., $\approx 47\%$ for an ISCAS85 test circuit c432) need to be lifted to achieve any meaningful security. Further, Rajendran *et al.* pointed out that an attacker may exploit the heuristics in physical design, e.g., re-construct the gate-level netlist by finding nearest neighbors [49]. To defeat such an attack requires degraded placement and routing quality and resultant performance degradation and area and power consumption increase.

III. Reconfiguration-Based VLSI Design for Security

A. Reconfiguration-Based VLSI Obfuscation

1) Theorem: A reconfigurable implementation of a logic function f or a sequential module is an obfuscated implementation for any supply chain adversary.

Proof: A reconfigurable logic module of m configuration bits has 2^m possible configurations. A logic function or a sequential design has numerous implementation choices such as in high-level synthesis, logic synthesis and physical design such as cell placement and interconnect routing. A reconfigurable logic module is configured by an end user after the design and the manufacturing process. A reconfigurable logic module typically has its own verification and testing procedures for each possible configuration. To enables design verification or manufacturing test of the whole system, a reconfigurable logic module can be configured in any way such that the required function is achieved, while an end user can choose any configuration for a given function.

A supply chain adversary may know the function or the input-output behavior of a reconfigurable logic module based on his role in the design process or from the rest of the design, but he has no knowledge on how a reconfigurable logic module is synthesized, placed and routed, and he cannot locate an internal signal in a reconfigurable logic module and tamper it. He cannot gain such knowledge by reverse engineering, testing or probing the internal nodes of a reconfigurable logic module because the reconfigurable logic module has yet to be constructed by an end user.

As a result, a reconfigurable implementation of a logic function f is an obfuscated implementation for any supply chain adversary because it possesses the three properties of obfuscation: 1) preserving functionality, 2) polynomial slowdown, and 3) "virtual black-box." ■

A supply chain adversary may install a hardware Trojan which probes all the internal nodes of a reconfigurable logic module after it is configured by an end user, locate an internal signal by analysis (e.g., observing an expected logic value for all possible input combinations), and tamper the internal signal. However, such a hardware Trojan would be too costly to stay stealthy.

Alternatively, a hardware Trojan may reconfigure the entire reconfigurable logic module. Such a hardware Trojan needs to save the authentic configuration bits and the Trojan configuration bits, which would not make it stealthy.

Even a field engineer cannot gain knowledge on the implementation of a reconfigurable logic module or tamper a reconfigurable logic module if the module is reconfigured right after the field engineer's visit. For success of a critical mission, reconfiguration may be performed right before the mission launch time.

In comparison, split manufacturing does not achieve obfuscation of the entire netlist, as Imeson *et al.* have shown the complexity for an adversary to reconstruct the netlist [28], which may be further reduced by exploiting physical design heuristics as is shown by Rajendran *et al.* [49].[2].

B. *Reconfiguration-Based VLSI Moving Target Defense*

1) Theorem 1: Reconfiguration for different logic functions or sequential modules achieves VLSI moving target defense.

Moving target defense has been proposed against software-based attacks. For example, Address Space Layout Randomization (ASLR) is to randomize memory addresses of known attack targets [23], for example, the stack of a program which could be the target of a buffer overflow attack [35]. Instruction Set Randomization (ISR) was proposed to prevent software-based code injection attacks [7], [8], [11], [31]. For example, the instruction opcodes may be randomized, or the instruction bits may be permuted or XOR'ed. Without knowledge of the instruction set, an attacker cannot inject Trojan code or execute a malicious program.

Moving target defense is more effective in preventing attacks from hardware Trojans which have only limited computation resource in order to stay stealthy. For example, ISR has its limitation in preventing software-based code injection attacks as a software-based attacker may gain knowledge of the instruction set by try and trial [23], [54]. However, launching such a complex attack from a hardware Trojan would make the hardware Trojan easy to be detected. As a result, ISR is more effective in preventing hardware-based code injection attacks.

Against a supply chain adversary or a hardware Trojan, a moving target defense scheme must be concealed by obfuscated implementation.

[2]Alternative to reconfiguration, VLSI design obfuscation can also be achieved by integrating a trusted die on a 3-D chip [41]. The trusted die is a black box to any untrusted foundry, design house or CAD tools. This trusted die technology is an extension of split manufacturing. In traditional split manufacturing, only interconnects are obfuscated. While in trusted die technology, the entire logic netlist on a trusted die is obfuscated, including logic gates and interconnects. Still, obfuscation of part of the design does not guarantee that the rest of the design is obfuscated

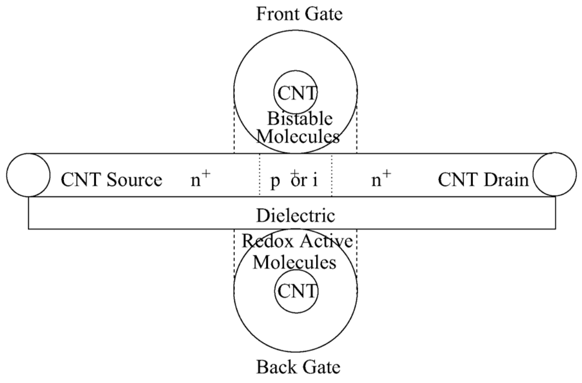

Fig. 2. A n-type reconfigurable double gate carbon nanotube field effect transistor (RDG-CNFET).

C. *Reconfiguration-Based Security-Aware VLSI Design in Emerging Technologies*

Besides hardware design security, reconfiguration is further critical to yield and reliability in emerging technologies, as we present in Section I. Emerging technologies further provide more cost-efficient implementation platforms for the proposed reconfiguration-based security-aware VLSI design methodologies. For example, a carbon nanotube crossbar provides a 2-D array of reconfigurable double-gate carbon nanotube field-effect transistors (RDG-CNFETs), each of which can be configured to open, short, or a field-effect transistor (Fig. 2) [39]. In such a computing platform, implementing an n-input simple logic function requires only $2n^2$ RDG-CNFET transistors (for a $n \times n$ reconfigurable PMOS network and a $n \times n$ reconfigurable NMOS network). While in a traditional lookup table-based reconfigurable logic module, implementing an n-input simple logic function requires $(6 + 2n)2^n$ MOS transistors (for each of the 2^n truth table entry there needs to be a 6-transistor SRAM cell and $2n$ pass transistors) (Fig. 3) [39], [40]. Emerging technologies further enable encryption/decryption hardware cost efficiency improvement as follows.

D. *Reconfigurable Reversible Computing (RRC)-Based Cryptography*

All the existing cryptographic primitives have proofs of security based on two assumptions: 1) read-proof hardware; that is, hardware that prevents an enemy from reading anything about the information stored within it; and 2) tamper-proof hardware; that is, hardware that prevents an enemy from changing anything in the information stored within it. In particular, existing

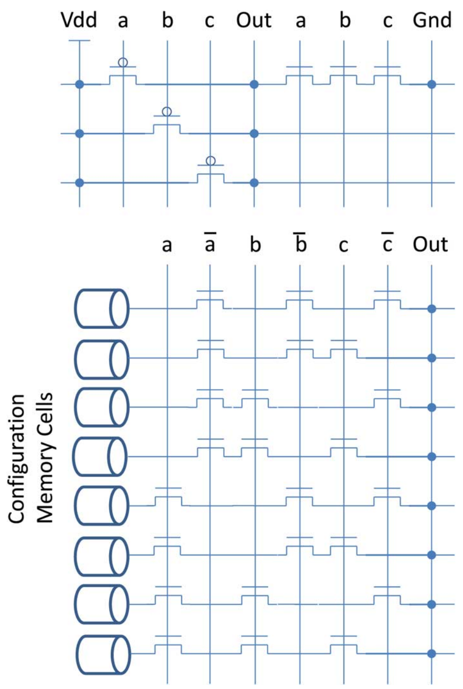

Fig. 3. A 3-input NAND gate implemented in (top) a 2-D RDG-CNFET array, and (bottom) a traditional lookup table-based reconfigurable logic module.

cryptographic schemes consist of an algorithm which the adversary knows, but cannot change (i.e., stored in tamper-proof hardware), and a secret key, which the adversary does not know and cannot change (i.e., stored in hardware which is both read-proof and tamper-proof) [25].

Here we propose alternative cryptographic schemes consisting of an algorithm which the adversary does not know. For example, we propose to encrypt by a reconfigurable logic function f, and decrypt by its reverse function f^{-1}. Such a logic function f provides reversible computing. A computation is reversible if it can be 'undone' in the sense that the output contains sufficient information to reconstruct the input, i.e., no input information is erased [58]. For example, bit permutation is reversible, and simple logic gates can be constructed by selecting a subset of input and output bits of permutation. Although study of reversible computing can be traced back to the early 1960s when Landauer pointed out that irreversible erasure of an information bit consumes power and dissipates heat [34], reversible computing has received much renewed research attention recently for low power computing, signal processing, cryptography, computer graphics, program inversion, reversible debugging, networks on chip, nano- and photonic circuits, and quantum computation [52]. In particular, some cryptographic primitives are based on reversible computing. For example, the AES algorithm includes iteration of four steps: substitute bytes based on S-boxes, shift rows, mix columns, and add round key by bitwise XOR. A much simpler encryption scheme is to perform one-time bitwise XOR with a key. This is however very vulnerable to adversary attacks, as an adversary can easily derive the key from a pair of input and output vectors. A stronger encryption scheme can be based on a more complex

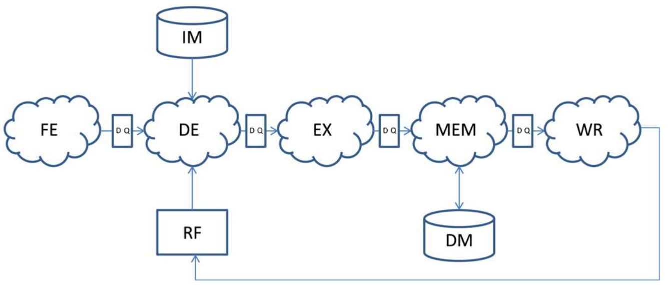

Fig. 4. Control/data flow diagram for the SPARC V8 LEON2 processor of a simple five-stage instruction pipeline.

combinational logic network which achieves a one-to-one mapping (i.e., reversible computing) between the inputs and the outputs. For n-bit inputs, n-bit outputs, and a k-bit key, there exist $2^k n!$ possible reversible computing logic functions. The number of possible reversible computing schemes can be further increased in a reconfigurable sequential design (such as AES which has a key expansion scheme and applies a different key in each iteration). Reconfiguration further conceals such a reversible computing scheme from any supply chain adversary or hardware Trojan.

E. *Obfuscation-Based VLSI Design-for-Security Methodology*

We do not propose to obfuscate an entire hardware system either by implementation in reconfigurable logic or manufacture at a trusted foundry after design at a trusted design house. For a variety of design objectives, it is preferred to obfuscate a minimum part of a hardware system while ensuring all the required security properties. For example, reconfigurable logic has larger area, higher power consumption, and lower performance compared to ASIC, and reconfigurable IP is also subject to security threats and needs to be protected, for example, by encryption and authentication. In access control-based VLSI obfuscation, it is preferred to minimize the part of the hardware system which is designed and manufactured by trusted parties, because an untrusted party is involved in an IC design and manufacturing process for certain competitive advantage, for example, a foreign foundry of a more advanced technology, an IP provider with a ready-to-use IP, or a CAD vendor who offers to shorten a design's time-to-market.

Consequently, we propose to obfuscate a minimum part of a hardware system which provides the root of design confidentiality and subsequently ensures design integrity and data confidentiality of the entire system at minimum cost. A generic methodology is to 1) identify and evaluate each possible attack scheme, 2) develop and evaluate each potential defense scheme, e.g., by selecting and obfuscating a subset of modules in the system, and 3) choose a defense scheme. Based on a control/data flow diagram, an obfuscated module may be chosen to be at the entry point of an attack, or a module which is necessary or critical for an attack to succeed. For example in a processor, we obfuscate the instruction decode unit against code-injecting attacks, and some of the memory stage logic against unauthorized memory access attacks (Fig. 4). We illustrate based on an open source SPARC V8 processor LEON2 as follows.

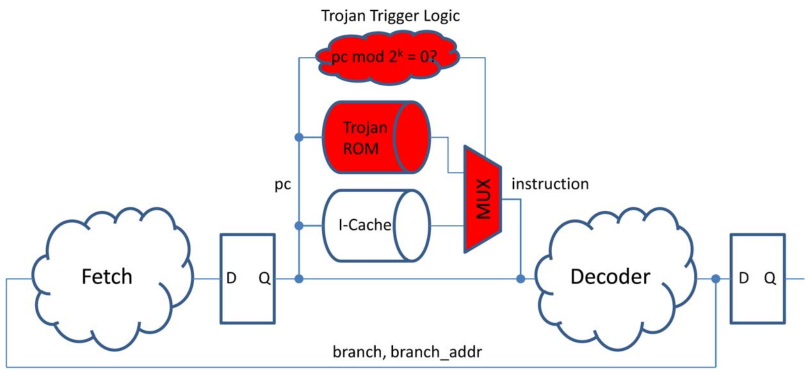

Fig. 5. Code injection hardware Trojan (shaded in red) including a Trojan ROM, a group of multiplexers, and a trigger logic module.

IV. Case Study: LEON2 SoC

LEON2 is an open source SPARC V8 architecture processor based on a simple five-stage instruction pipeline. The five pipeline stages are instruction fetch, decode, execute, memory and write (Fig. 4). The instruction fetch unit (FE) provides the instruction addresses or program counts (PC), based on which the decode unit (DE) reads instructions from the instruction memory (IM) and reads the register file (RF). The memory (MEM) stage logic loads the registers from the data memory (DM) based on load instructions. The function units in the execute (EX) stage perform the CPU operations. The write (WR) stage logic writes the results back to the register file (RF). Finally the memory (MEM) stage logic updates the data memory (DM) based on store instructions.

Such a computing system is subject to two categories of adversary attacks: 1) that compromise computation integrity and 2) that compromise data confidentiality. We evaluate possible attack and defense schemes as follows.

A. On Computation Integrity

1) Hardware Trojans That Compromise Computation Integrity: An adversary may install a hardware Trojan which once triggered tampers an authentic computation by executing more (e.g., injecting a malicious program) or less (e.g., skipping security checks) or altering the authentic computation results (e.g., acting as a logic bomb). A hardware Trojan may tamper any control signal. A hardware Trojan of simple logic may be implemented as a Finite State Machine. However, a hardware Trojan of more complex logic needs to be based on a malicious program. The malicious program can be stored in a Trojan nonvolatile memory to avoid detection by any static memory check. The hardware Trojan may send the malicious program instructions to the authentic instruction decoder rather than having a Trojan instruction decoder for minimum footprint.

Fig. 5 illustrates a code-injecting hardware Trojan (shown as colored modules) in a LEON2 processor, which includes a Trojan ROM containing the Trojan program, a group of multiplexers at the instruction fetch unit inputs, and a trigger logic module. The Trojan trigger logic module monitors the next program count (npc) in the instruction fetch unit. When the trigger condition is met, for example, the lower n bits of the next program count are all zero's, the Trojan multiplexers direct the instruction fetch unit to fetch instructions from the Trojan ROM other than from the instruction cache. Since the Trojan ROM is very small, it can be addressed by the lower n bits of the program count. The Trojan instruction sequence starts by saving the program count and the other processor internal states, and ends by restoring the processor internal states including the program count. When the low n bits of the program count equal to the address of the last Trojan instruction (that restores the program count), the Trojan multiplexers direct the instruction fetch unit to fetch instructions from the instruction cache. This resumes the authentic operation.

2) Reconfigurable Instruction Decoder for Instruction Set Randomization: To prevent attacks by such a powerful code-injecting hardware Trojan, we implement the instruction decode unit in reconfigurable logic, which achieves not only obfuscation but also moving target defense, specifically, Instruction Set Randomization (ISR).

A supply chain adversary has only black box access to the obfuscated IDU. Consequently, he has no knowledge on the instruction set and cannot tamper the instruction decoder. A hardware Trojan may carry out the following attacks, but cannot stay stealthy at the same time.

1) To carry out a deputy attack [23] or send Trojan instructions to the obfuscated IDU, a Trojan needs knowledge of the instruction set, and it further needs to be reconfigurable to translate the Trojan instructions to the local dialect. A Trojan may apply stimuli to the instruction decoder, collect responses and analyze them for knowledge on the instruction set. However, for n-bit instructions and a k-bit key, bit permutation and bitwise XOR with the key give $2^k n!$ possible instruction encryption schemes. Further, to probe an internal signal (e.g., the opcode check logic output), a Trojan needs to probe all the nodes in the reconfigurable IDU, which has a prohibitive cost. Or, a Trojan may probe the other signals which are in the hardware system but are not in the obfuscated module. For a microprocessor, these signals include the program count, the ALU inputs, the memory stage inputs, the register file inputs, etc. The cost is also prohibitive for a Trojan to stay stealthy. For example, an ALU add operation may be caused by an add, subtract, multiple, load, store, jump, or return instruction.
2) To carry out a circumvention attack [23] or bypass the instruction decoder, a Trojan needs to duplicate the instruction decoder unit, which would be too costly. Or, a Trojan

may reconfigure the IDU, which requires that the Trojan includes all the configuration bits for the IDU. There would further be significant performance degradation when the Trojan reconfigures the IDU.

3) Or, an adversary with physical access to a device may reprogram the reconfigurable instruction decoder and install a code injection Trojan in it. To install a Trojan while keeping the authentic reconfigurable instruction decoder is not easy, and the size of the reconfigurable module limits the size of the Trojan.
4) Or, an adversary may save the configuration bits, replace the instruction decoder by a code injection Trojan, e.g., including a Trojan memory and a Trojan instruction decoder, and restore the authentic configuration after the attack. Such an attack must be launched when there is no authentic computation in the device to avoid detection. Further, such an attack is limited by the capacity the reconfigurable module in terms of such as size and port access. For example, the reconfigurable module may only include part of the instruction decoder, such that a Trojan in the reconfigurable module cannot inject malicious code into the system.

3) Evaluation: We evaluate the code injection Trojan and reconfiguration-based IDU obfuscation based on an five-stage in-order open source SPARC processor LEON2 [1], which is configured to include a five-cycle multiplier, a 35-cycle divider, a floating-point unit, a memory management unit, a PCI interface, and a network unit with no co-processors. We perform logic synthesis based on the Synopsys Design Compiler and the 45 nm Nangate open cell library [53]. For the lookup table-based reconfigurable technology (Fig. 1), we modify the 45 nm Nangate cell library such that a n-input logic gate has the same area as a 2^n-input multiplexer plus 2^n latches.

We have implemented a minimum code injection Trojan with a 1 KB ROM, a few multiplexers at the instruction fetch unit input, and a trigger logic network. We have further implemented an IDU in the lookup table-based reconfigurable technology as in FPGA. We did not implement a deputy attack Trojan because of its complexity and prohibitive cost in the presence of $2^k n!$ possible instruction encryption schemes for n-bit instructions and a k-bit key. We estimate the cost of a circumvention attack Trojan by summing up the cost of the minimum code injection Trojan and the standard IDU or the configuration memory cells of a reconfigurable IDU.

Table I gives the hardware overhead of these designs. Compared with the LEON2 processor, the minimum code injection Trojan with a 1 KB ROM leads to a layout area increase of only 2.5%. Reconfiguration-based IDU obfuscation increases the IDU area from 926.3 μm^2 to 15718.3 μm^2, and increases the overall LEON2 processor area by 34.7%. For a circumvention attack, to reconfigure the IDU, including all the configuration memory cells would increase the minimum code injection Trojan area by 9.5× as the configurable memory cells take 67.7% of the area in our 2-input LUT-based reconfigurable logic. Alternatively, including an additional IDU would increase the minimum code injection Trojan area by 82.7%. For tradeoff between cost and security, we may implement x% of the IDU in reconfigurable logic starting from the inputs, which increases the Trojan area by $0.827x$% at the cost of $0.347x$% area increase for the LEON2 processor.

TABLE I
HARDWARE OVERHEAD OF THE LEON2 PROCESSOR, A CODE INJECTION TROJAN WITH 1 KB ROM, THE IDU, AN OBFUSCATED IDU, AND THE LEON2 PROCESSOR WITH AN OBFUSCATED IDU, RESPECTIVELY

	Area (μm^2)	Power (mW)	Delay (ns)
LEON2	4.52×10^4	2.25	7.79
Trojan w/ 1KB ROM	1.12×10^3	2.22×10^{-3}	0.41
IDU	9.26×10^2	1.40×10^{-2}	6.79
Obfuscated IDU	1.57×10^4	1.46×10^{-1}	18.01
LEON2 w/ Obfuscated IDU	6.09×10^4	2.40	18.01
Inst Enc by RRC	3.24×10^2	4.55×10^{-3}	0.88
IDU w/ Inst Enc by RRC	1.25×10^3	1.86×10^{-2}	7.67
LEON2 w/ Inst Enc by RRC	4.55×10^4	2.25	7.79

Alternatively, we may encrypt and decrypt the instruction bits by reconfigurable reversible computing (RRC). In the presence of $2^k n!$ possible instruction encryption schemes for n-bit instructions and a k-bit key (where $n = 32$ for LEON2 and $k = 16$ in our implementation), a hardware Trojan cannot carry out a deputy attack or gain any knowledge on the RRC scheme because of its complexity and prohibitive cost. Including such an RRC module at the input of the IDU increases the IDU area by 3.77% and the LEON2 processor area by 0.72% without affecting the overall power consumption and performance of the LEON2 processor.

B. On Data Confidentiality

1) Memory Access Trojans: Other than code injection, signal injection is another mechanism that a hardware Trojan may rely on to perform stealth operation or gain unauthorized access s to confidential data in a hardware system. Specifically, a hardware Trojan may gain unlimited access to the memory space by sending memory access requests in supervisor mode, bypassing any memory access checking mechanism.

In SPARC V8 architecture, memory access instructions include load, store, and swap instructions of different sizes (byte, half word, word, or double word). An effective memory address is generated by adding the contents of two registers, or the content of a register and a signed immediate number. Each memory address is tagged with an Address Space Identifier (ASI) which can be 8, 9, 0×A or 0×B, indicating an instruction or data access in user or supervisor mode. The ASI bits are generated based on a supervisor signal su, which is given by the supervisor bit s in the Processor State Register (PSR). The PSR also contains bits for processor interrupt level, trap enable, co-processor enable and floating-point processor enable. A Write PSR (WRPSR) instruction updates the PSR.

A hardware Trojan may request memory access in supervisor mode by (Fig. 6):

1) injecting a memory access instruction;
2) injecting memory access request signals including a memory address and an ASI which can be in supervisor mode;
3) injecting memory access request signals including a memory address while tampering the supervisor signal;

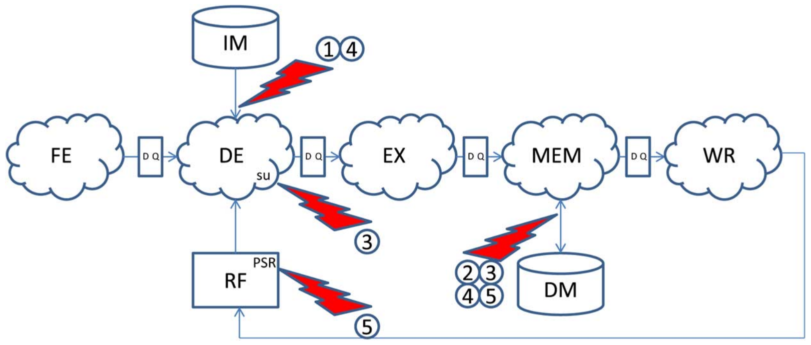

Fig. 6. Attack targets of five unauthorized memory access hardware Trojans in a LEON2 processor.

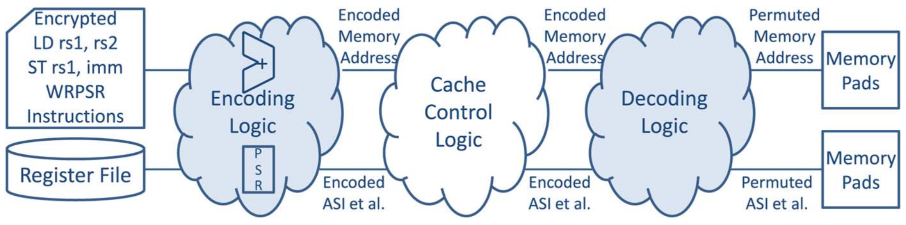

Fig. 7. Obfuscated memory access logic (shaded in blue) including memory access instruction decoding logic, WRPSR decoding logic, PSR, supervisor signal logic and ASI logic.

4) injecting memory access request signals including a memory address after tampering the PSR, e.g., by injecting a WRPSR instruction; or
5) injecting memory access request signals including a memory address after tampering the PSR directly.

2) Reconfigurable Memory Access Module Against Illegal Memory Access: We prevent a hardware Trojan from gaining unauthorized memory access as follows (Fig. 7).

1) We first encrypt the instructions by reconfigurable reversible computing (RRC). We insert an RRC module including a 16-bit key at the input of the instruction decoder for this. As a result, without knowledge on the instruction encryption scheme, an adversary or a hardware Trojan cannot inject a memory access instruction or a WRPSR instruction or any other instruction.
2) Next, we encrypt the memory address by RRC. We insert an RRC module including a 16-bit key at the register file output for this. We also encrypt the register file addresses by RRC.We insert an RRC module including a 5-bit key at the register file input for this. As a result, without knowledge of the memory and register file address encryption schemes, an adversary or a hardware Trojan cannot inject a memory address or tamper a register.
3) Further, we obfuscate the PSR, the supervisor signal logic, and the ASI logic by implementing them in reconfigurable logic. This obfuscated logic module outputs encoded ASI bits. A supply chain adversary or a hardware Trojan cannot tamper the obfuscated module, the ASI bits, the supervisor signal or the PSR bits.
4) We connect the memory access pads to a reconfigurable logic module which decodes the memory access request signals including the memory address and the ASI bits, and send the memory access request signals to the memory access pads after permutation. Without knowledge on the encoded memory access signals and the permuted memory access pads, a supply chain adversary or a hardware Trojan cannot inject memory access request signals to the inputs of the memory access signal decoding logic or the memory access pads.

We have a one-to-one mapping between an encoded memory address and a plain text memory address. As a result, the cache control logic does not need any redesign for encoded memory addresses, as long as we do not mix cache block address bits with block offset bits.

3) Evaluation: We designed a memory access Trojan, which injects Trojan memory access signals when the authentic processor is not accessing the memory. It generates memory addresses based on a 32-bit counter, and generates the supervisor ASI bits to gain unauthorized memory access. It searches for a given key in the memory, and stores the next word.

We further design a LEON2 processor which prevents unauthorized memory accesses by encrypting instructions, memory addresses, ASI bits and register file addresses by RRC, obfuscating the ASI logic module including the PSR register, and permuting the memory access request signals at the memory access pads. As a result, a hardware Trojan cannot gain unauthorized memory access by 1) injecting a memory access instruction or a WRPSR instruction in the presence of $2^{16}32!$ possible instruction encryption schemes, 2) cannot inject memory access request signals in the presence of 40! possible memory address and ASI bit permutations, and 3) cannot tamper the ASI logic module including the PSR register in the presence of 2^m possible configurations for the reconfigurable ASI logic module.

Table II gives the hardware overhead of the LEON2 processor, the memory access Trojan, and the memory Trojan-resistant LEON2 processor, respectively. We observe that the memory access Trojan is very small. It takes only 0.08% of the LEON2 processor area. It has a negligible power consumption

TABLE II
HARDWARE OVERHEAD OF THE LEON2 PROCESSOR, A MEMORY ACCESS HARDWARE TROJAN, AND A MEMORY TROJAN RESISTANT LEON2 PROCESSOR, RESPECTIVELY

	Area (μm^2)	Power (mW)	Delay (ns)
LEON2	4.52×10^4	2.25	7.79
Memory Trojan	3.49×10^2	8.89×10^{-3}	6.85
Inst Enc by RRC	3.24×10^2	4.55×10^{-3}	0.88
Reg. Enc by RRC	4.97×10^1	0.74×10^{-3}	0.87
Obf. PSR, su, ASI logic	2.94×10^2	2.79×10^{-3}	0.88
Memory Trojan Resistant LEON2	4.72×10^4	2.25	8.67

and does not degrade the LEON2 processor performance, although it has a large critical path delay which comes from the 32-bit counter. With all the patches, a memory Trojan-resistant LEON2 processor has an area increase of 4.42%, negligible power consumption increase, and a critical path delay increase of 11.30%.

V. CONCLUSION

In this paper, we propose 1) reconfiguration-based VLSI obfuscation, 2) reconfiguration-based VLSI moving target defense, 3) reconfigurable reversible computing (RRC)-based cryptography, and 4) a generic reconfiguration-based VLSI design-for-security methodology. As case studies, we present a reconfigurable instruction decoder for instruction set randomization (ISR) against code injection attacks, and a reconfigurable memory access module for memory access signal encoding against unauthorized memory access in an open source LEON2 processor [1]. By obfuscating part of the instruction decoder in reconfigurable logic, we prevent program monitoring Trojan attacks and increase the area of a minimum code injection Trojan with a 1 KB ROM by 2.38% for every 1% area increase of the LEON2 processor. Alternatively, encrypting instructions by a simple reconfigurable logic module prevents code injection attacks at the cost of 0.72% area increase, negligible power consumption increase and no performance degradation for the LEON2 processor. We further achieve a memory access Trojan-resistant LEON2 processor at the cost of 4.42% area increase, negligible power consumption increase, and 11.30% critical path delay increase. Emerging technologies such as carbon nanotube crossbar-based reconfigurable computing provide further cost reduction for the proposed methodology.

ACKNOWLEDGMENT

The authors would like to thank the reviewers and Dr. R. Karri and J. Rajendran at NYU-Poly for their valuable comments.

REFERENCES

[1] Aeroflex Gaisler, LEON SPARC V8 Processors [Online]. Available: http://www.gaisler.com/
[2] M. Agrawal, S. Karmakar, D. Saha, and D. Mukhopadhyay, "Scan based side channel attacks on stream ciphers and their counter-measures," in *Proc. Int. Conf. Cryptol. India (INDOCRYPT)*, 2008, pp. 226–238.
[3] Y. M. Alkabani and F. Koushanfar, "Active hardware metering for intellectual property protection and security," in *Proc. USENIX Security Symp.*, 2007, pp. 291–306.
[4] T. Alves and D. Felton, "Trustzone: Integrated hardware and software security," *Inf. Q.*, vol. 3, no. 4, pp. 18–24, 2004.
[5] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang, "On the (im)possibility of obfuscating programs," in *Proc. Int. Conf. Cryptogr.*, 2001, pp. 1–18.
[6] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang, "On the (im)possibility of obfuscating programs," *J. ACM*, vol. 59, no. 2, p. 6, 2012.
[7] E. G. Barrantes, D. H. Ackley, S. Forrest, and D. Stefanovic, "Randomized instruction set emulation," *ACM Trans. Inf. Syst. Security*, vol. 8, no. 1, pp. 3–40, 2005.
[8] E. G. Barrantes, D. H. Ackley, T. S. Palmer, D. Stefanovic, and D. D. Zovi, "Randomized instruction set emulation to disrupt binary code injection attacks," in *Proc. ACM Conf. Comput. Commun. Security*, 2003, pp. 281–289.
[9] A. Baumgarten, A. Tyagi, and J. Zambreno, "Preventing IC piracy using reconfigurable logic barriers," *IEEE Design Test Comput.*, vol. 27, no. 1, pp. 66–75, Jan./Feb. 2010.
[10] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *Proc. Ann. Int. Cryptogr. Conf. Adv. Cryptogr.*, 1997, pp. 513–527.
[11] S. W. Boyd, G. S. Kc, M. E. Locasto, A. D. Keromytis, and V. Prevelakis, "On the general applicability of instruction-set randomization," *IEEE Trans. Dependable Secure Comput.*, vol. 7, no. 3, pp. 255–270, 2010.
[12] E. Boyle, K.-M. Chung, and R. Pass, "On extractability obfuscation," *Cryptogr. ePrint* 2013 [Online]. Available: http://eprint.iacr.org/2013/650
[13] Z. Brakerski and G. N. Rothblum, "Black-box obfuscation for d-cnfs," *Cryptogr. ePrint* 2013 [Online]. Available: http://eprint.iacr.org/2013/557
[14] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Computers*, vol. 35, no. 8, pp. 677–691, Aug. 1986.
[15] I. Cadence Design Syst., 3-D ICs with TSVs-design challenges and requirements [Online]. Available: www.cadence.com/rl/resources/white papers/3dic wp.pdf
[16] R. Canetti, "Towards realizing random oracles: Hash functions that hide all partial information," in *Proc. Int. Conf. Cryptogr.*, 1997, pp. 455–469.
[17] R. S. Chakraborty and S. Bhunia, "Hardware protection and authentication through netlist level obfuscation," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 2008, pp. 674–677.
[18] R. S. Chakraborty and S. Bhunia, "Harpoon: An obfuscation-based SoC design methodology for hardware protection," *IEEE Trans. Computer-Aided Design*, vol. 28, no. 10, pp. 1493–1502, Oct. 2009.
[19] R. S. Chakraborty and S. Bhunia, "Security against hardware Trojan through a novel application of design obfuscation," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 2009, pp. 113–116.
[20] Chipworks [Online]. Available: http://www.chipworks.com/
[21] A. R. Desai and M. S. Hsiao *et al.*, "Interlocking obfuscation for anti-tamper hardware," in *Proc. Cyber Security Inf. Intell. Res. Workshop*, 2012, pp. 1–4.
[22] C. Dwyer, L. Vicci, and J. Poulton *et al.*, "The design of DNA self-assembled computing circuitry," *IEEE Trans. Very Large Scale (VLSI) Syst.*, vol. 12, no. 11, pp. 1214–1220, Nov. 2004.
[23] D. Evans, A. Nguyen-Tuong, and J. Knight, "Moving target defense: An asymmetric approach to cyber security," in *Effectiveness of Moving Target Defenses*. New York: Springer, 2011.
[24] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters, "Candidate indistinguishability obfuscation and functional encryption for all circuits," in *Proc. IEEE Symp. Foundat. Comput. Sci.*, Oct. 2013, pp. 40–49.
[25] R. Gennaro, A. Lysyanskaya, T. Malkin, S. Micali, and T. Rabin, "Algorithmic tamper-proof (ATP) security: Theoretical foundations for security against hardware tampering," in *Theory Cryptogr. Conf.*, 2004, pp. 258–277.
[26] S. Goldwasser and Y. T. Kalai, "On the impossibility of obfuscation with auxiliary input," in *Proc. IEEE Symp. Foundat. Comput. Sci.*, 2005, pp. 553–562.
[27] S. Hohenberger, G. N. Rothblum, A. Shelat, and V. Vaikuntanathan, "Securely obfuscating re-encryption," in *Theory Cryptogr. Conf*, 2007, pp. 233–252.
[28] F. Imeson, A. Emtenan, S. Garg, and M. V. Tripunitara, "Securing computer hardware using 3d integrated circuit (IC) technology and split manufacturing for obfuscation," in *Proc. 22nd USENIX Security Symp.*, 2013, pp. 495–510.

[29] C. E. Irvine and K. Levitt, “Trusted hardware: Can it be trustworthy?,” in *Proc. ACM/IEEE Design Automat. Conf*, 2007, pp. 1–4.

[30] S. J. Kang, C. Kocabas, and T. Ozel *et al.*, “High-performance electronics using dense, perfectly aligned arrays of single-walled carbon nanotubes,” *Nature Nanotechnol.*, vol. 2, pp. 230–236, 2007.

[31] G. S. Kc, A. D. Keromytis, and V. Prevelakis, “Countering code-injection attacks with instruction-set randomization,” in *Proc. ACM Conf. Comput. Commun. Security*, 2003, pp. 272–280.

[32] P. Kocher, J. Jaffe, and B. Jun, “Differential power analysis,” in *Proc. Int. Cryptogr. Conf. Adv. Cryptogr.*, 1999, pp. 388–397.

[33] P. C. Kocher, “Timing attacks on implementations of Diffie-Hellman RSA, DSS, other systems,” *Advances in Cryptology*, vol. 1109, Lecture Notes Comput. Sci., pp. 104–113, 1996.

[34] R. Landuer, “Irreversibility and heat generation in the computing process,” *IBM J. Res.*, pp. 183–191, 1961.

[35] R. B. Lee, D. K. Karig, J. P. Mcgregor, and Z. Shi, “Enlisting hardware architecture to thwart malicious code injection.,” in *Proceedings of the 2003 International Conference on Security in Pervasive Computing*. New York: Springer Verlag, 2003, pp. 237–252.

[36] R. B. Lee, P. C. S. Kwan, J. P. McGregor, J. Dwoskin, and Z. Wang, “Architecture for protecting critical secrets in microprocessors,” in *Proc. Int. Symp. Comput. Archit.*, 2005, pp. 2–13.

[37] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz, “Architecture support for copy and tamper resistant software,” in *Proc. Int. Conf. Archit. Support Programm. Languages Operat. Syst.*, 2000, pp. 168–177.

[38] B. Liu, “Defect mapping and adaptive configuration of nanoelectronic circuits based on a CNT crossbar nano-architecture,” *Workshop Nano Molecular, Quantum Commun. (NanoCom)*, 2009.

[39] B. Liu, “Reconfigurable double gate carbon nanotube transistor based nanoelectronic architecture,” in *Proc. Asian South Pacific Design Automat. Conf.*, 2009, pp. 853–858.

[40] B. Liu, “Architecture exploration of crossbar-based nanoscale reconfigurable computing platforms,” *Nano Commun. Netw. J.*, vol. 1, no. 3, pp. 232–241, 2010.

[41] B. Liu and B. Wang, “Embedded reconfigurable logic for asic design obfuscation against supply chain attacks,” in *Proc. Conf. Design Automat. Test Eur.*, Dresden, Germany, 2014, pp. 1–6.

[42] Microsoft, Next-generation secure computing base [Online]. Available: www.microsoft.com/resources/ngscb/default.mspx

[43] A. Narayanan and V. Shmatikov, On the limits of point function obfuscation 2006 [Online]. Available: http://eprint.iacr.org/2006/182

[44] National Security Council, The comprehensive national cybersecurity initiative [Online]. Available: http://www.whitehouse.gov/cybersecurity/ comprehensive-national-cybersecurity-initiative

[45] S. S. P. Parkin, “Spintronics materials and devices: Past present and future,” in *IEEE Int. Electron Devices Meet. Tech. Digest*, 2004, pp. 903–906.

[46] A. R. Pease, J. O. Jeppesen, J. F. Stoddart, Y. Luo, C. P. Collier, and J. R. Heath, “Switching devices based on interlocked molecules,” *Acc. Chem. Res*, vol. 34, pp. 433–444, 2001.

[47] J. Rajendran, A. K. Kanuparthi, M. Zahran, S. K. Addepalli, G. Ormazabal, and R. Karri, “Securing processors against insider attacks: A circuit-microarchitecture co-design approach,” *IEEE Design Test Comput.*, vol. 30, no. 2, pp. 35–44, 2013.

[48] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, “Security analysis of logic obfuscation,” in *Proc. ACM/IEEE Design Automat. Conf.*, 2012, pp. 83–89.

[49] J. Rajendran, O. Sinanoglu, and R. Karri, “Is split manufacturing secure,” in *Proc. Conf. Design Automat. Test Eur.*, 2013, pp. 1259–1264.

[50] J. Roy, F. Koushanfar, and I. Markov, “EPIC: Ending piracy of integrated circuits,” in *Proc. Conf. Design Automat. Test Eur.*, 2008, pp. 1069–1074.

[51] J. A. Roy, F. Koushanfar, and I. L. Markov, “Protecting bus-based hardware IP by secret sharing,” in *Proc. ACM/IEEE Design Automat. Conf.*, 2008, pp. 846–851.

[52] M. Saeedi and I. L. Markov, “Synthesis and optimization of reversible circuits-a survey,” *ACM Comput. Surv.*, vol. 45, no. 2, pp. 21:1-21:34–851, 2013.

[53] Silicon Integration Initiative (SI2), Nangate Open Cell Library [Online]. Available: http://www.si2.org/openeda.si2.org/projects/nangatelib/

[54] N. Sovarel, D. Evans, and N. Paul, “Where's the FEEB? The effectiveness of instruction set randomization,” in *Proc. 14th USENIX Security Symp.*, Baltimore, MD, 2005, pp. 1–16.

[55] M. R. Stan, P. D. Franzon, S. C. Goldstein, J. C. Lach, and M. M. Ziegler, “Molecular electronics: From devices and interconnect to circuits and architecture,” *Proc. IEEE*, vol. 91, no. 11, pp. 1940–1957, Nov. 2003.

[56] G. E. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas, “Aegis: Architecture for tamper-evident and tamper-resistant processing,” in *Proc. Int. Conf. Supercomput.*, San Francisco, CA, 2003, pp. 160–171.

[57] Tezzaron semiconductor, 3-D ICs and Integrated Circuit Secuirty 2008 [Online]. Available: http://www.tezzaron.com/about/papers/3-D-ICs and Integrated Circuit Security.pdf

[58] T. Toffoli, “Reversible computing,” 1980, Tech. Memo MIT/LCS/TM-151.

[59] Trusted computing group, Trusted Platform Module (TPM) Specifications [Online]. Available: http://www.trustedcomputinggroup.org/resources/tpm main specification

[60] A. Waksman and S. Sethumadhavan, “Tamper evident microprocessors,” in *Proc. IEEE Symp. Security Privacy*, 2010, pp. 1–16.

[61] Z. Wang and R. B. Lee, “New cache design for thwarting software cache-based side channel attacks,” in *Proc. Int. Symp. Comput. Archit.*, 2007, pp. 494–505.

[62] H. Wee, “On obfuscating point functions,” in *Proc. ACM Symp. Theory Comput.*, Baltimore, MD, 2005, pp. 523–532.

[63] B. Yang, K. Wu, and R. Karri, “Scan based side channel attack on dedicated hardware implementations of data encryption standard,” in *Proc. IEEE Int. Test Conf.*, 2004, pp. 339–344.

Bao Liu is an Assistant Professor in Electrical and Computer Engineering Department at the University of Texas, San Antonio, TX, USA. His research areas include hardware security, nano-computing and VLSI CAD including physical design, statistical timing analysis and optimization, power rail and signal integrity analysis, reliable and resilient design, and delay testing. He has published over 60 journal articles and conference papers, and received three U.S. and international patents.

Prof. Liu is the receiver of a Best Paper Award in International Conference on Computer Design in 2005, a Best Research Award in UCSD Research Review in 2002, a China ICCAD Best Member Award in 1996, and a China Mathematics Olympiad Honor Medal in 1988. He has served as Chair for an invited session “Emerging Nano-Circuits and System” in Midwest Symposium on Circuits and Systems in 2010, co-Chair for the “Hardware and System Security” track in International Symposium on Quality Electronic Design (ISQED) in 2014, co-Chair for the “Emerging Design and Technology” track in ISQED since 2006, and a panelist on “CAD for Nanoelectronics” in International Symposium on Nanoscale Architectures in 2010.

Brandon Wang received the MSEE degree from New Jersey Institute of Technology, Newark, NJ, USA, and the MBA degree from the Wharton School at the University of Pennsylvania, Philadelphia, PA, USA.

He oversees the overall Cadence 3DIC solution marketing and product management activities, as well as other enablement efforts with strategic foundry partners. Prior to joining Cadence, he spent over six years at ARM, managing the Interface IP Group, and later the PHY product line, where he also served as a member of the Corporate Patent Review Committee, responsible for low-power and high-speed circuits. Before that, he was with UBICOM, a network processor startup that is now part of Qualcomm. He holds six U.S. patents, and has published at a number of IEEE conferences and in journal papers.

Mr. Wang serves as a Director on the Board of CASPA, a nonprofit semiconductor organization.