Hardware Trojans – Prevention, Detection, Countermeasures (A Literature Review)

ABSTRACT

A Hardware Trojan is a malicious, undesired, intentional modification of an electronic circuit or design, resulting in the incorrect behaviour of an electronic device when in operation – a back-door that can be inserted into hardware. A Hardware Trojan may be able to defeat any and all security mechanisms (software or hardware-based) and subvert or augment the normal operation of an infected device. This may result in modifications to the functionality or specification of the hardware, the leaking of sensitive information, or a Denial of Service (DoS) attack.

Understanding Hardware Trojans is vital when developing next generation defensive mechanisms for the development and deployment of electronics in the presence of the Hardware Trojan threat. Research over the past five years has primarily focussed on detecting the presence of Hardware Trojans in infected devices. This report reviews the state-of-the-art in Hardware Trojans, from the threats they pose through to modern prevention, detection and countermeasure techniques.

1 Introduction

Electronic systems have proliferated over the past few decades to the point that most aspects of daily life are aided or affected by the automation, control, monitoring, or computational power provided by Integrated Circuits (ICs). The ability to *trust* these ICs to perform their specified operation (and only their specified operation) has always been a security concern and has recently become a more active topic of research (Chakraborty, Paul & Bhunia 2008, Tehranipoor & Koushanfar 2010, Rajendran et al. 2010). ICs are the building blocks for all modern electronic systems; they form the information backbone of many critical sectors including the financial, military, industrial, and transportation sectors. Without trust in these ICs, the systems they support cannot necessarily be trusted to perform as specified and may even be susceptible to attack by a malicious adversary.

A new disruptive threat has surfaced over the past five years[1], a hardware-based security threat known as the *Hardware Trojan*. A Hardware Trojan is a malicious, undesired, intentional modification of an electronic circuit or design, resulting in the incorrect behaviour of an electronic device when in operation. Akin to a software Trojan Horse program (Thompson 1984), a Hardware Trojan is a back-door that can be inserted into hardware. The added advantages of a Hardware Trojan include residing at the lowest level of information processing, and being persistent – the threat is present as long as the infected hardware is in use. A Hardware Trojan may be able to defeat any and all security mechanisms (software or hardware-based) and subvert or augment the normal operation of an infected IC. The modification can affect any type of IC, including processors, memory, Digital Signal Processors (DSPs), Application Specific Integrated Circuits (ASICs), and even configuration bit-streams for reconfigurable logic. The undesired behaviour can

include modifications to the functionality or specification of the hardware, the leaking of sensitive information, or a Denial of Service (DoS) attack. Effects may range from a subtle degradation of service through to a complete and permanent shut-down of a system. Hardware Trojans can affect a system by themselves alone, or provide a foothold for software based attacks (King et al. 2008), where colluding software is aware of the inserted Trojan. Hardware Trojans most often remain dormant and are only activated when triggered by, for example, the passing of time, some specific activity, or external events. The spectrum of Hardware Trojans – their capabilities, size, trigger mechanisms, and payloads – is enormous, and the state-space of IC development both physically and procedurally provides ample opportunity for concealment.

The ease with which Hardware Trojans can make their way into modern ICs and electronic designs is concerning. Modifications to hardware can occur at any stage during the design and manufacturing process, including the specification, design, verification and manufacturing stages. Hardware Trojans may even be retro-fitted to existing ICs post-manufacture, as proposed by Abramovici & Bradley (2009). Maintaining tight control over the IC design life-cycle is costly; the current trend is towards separation of design from manufacture, relying on a handful of large CMOS fabrication facilities world-wide, mostly located within Asia. This trend towards out-sourcing is not limited to manufacture; designers trust third-party Electronic Design Automation (EDA) tools, utilise third-party Intellectual Property (IP) cores – many of which are provided only as a binary net-list –

[1] Existence of such threats has been mooted since the early 1990s (Schwartau 1994).

and outsource to contract-design houses. With so many entities and individuals involved, there is ample opportunity to insert a Trojan into the final hardware. A Hardware Trojan might be as simple as a paragraph change in the specification, an extra source-code line in the Hardware Description Language (HDL), a modification of the silicon die at the fabrication plant, or just a modulation of the CMOS geometries used.

Ensuring an IC is authentic and does not contain any Trojans is a very difficult problem (Tehranipoor & Koushanfar 2010). Hardware Trojans are poorly observable during traditional IC design-verification and testing phases; their payload and trigger states are difficult to find, becoming exponential in the number of circuit nodes (Chakraborty, Paul & Bhunia 2008). Most ASIC designs are too large for either exhaustive testing or formal verification, so it is likely that many Trojans will never be detected. The life-cycle stage at which a Trojan is inserted and its logical structure also affects the effectiveness of existing detection methods.

For a country like Australia, keeping the entire design and manufacturing process in-house is infeasible for all but the smallest ASIC designs. Our reliance on the globalisation of the electronics industry is critical for developing both our commercial and military capabilities. The Hardware Trojan threat must therefore be taken seriously and our approach must ensure that we are able to either trust the operation of ICs used or, better still, be able to safely utilise Commercial-Off-The-Shelf (COTS) electronic devices and ICs regardless of their trust.

Current research is investigating all aspects of Hardware Trojans. The Embedded Systems Challenge (Polytechnic Institute of New York University 2010) actively promotes the development of Hardware Trojans and detection mechanisms through an annual adversarial challenge against a specific embedded system. DARPA has initiated its Trust-in-ICs program (Microsystems Technology Office, DARPA 2007), an R&D program developing tools and techniques to ensure ICs are authentic and free of Trojans post-manufacture. The Australian Department of Defence (Anderson, North & Yiu 2008) raised an awareness of the threat and proposed broad classes of Hardware Trojans and countermeasures. Most other public research is conducted by university groups, focusing on techniques to prevent the insertion of Hardware Trojans into a design or detect the presence of Trojans post-manufacture.

This report introduces the threats posed by Hardware Trojans and the variety of manners in which they might be activated. It also presents the state-of-the-art in prevention, detection, and countermeasure techniques currently being researched. Unfortunately, current methods are not suitable for providing adequate protection – at best statistical guarantees can be provided on preventing or detecting a Hardware Trojan. In the near future, if not now, it will become necessary for hardware systems to be able to operate securely in the presence of Hardware Trojans. Recently, researchers have looked at op- erating securely despite the potential presence of Hardware Trojans. Some of this work is sponsored by DARPA, AFRL and Rockwell Automation. Future research will need to focus on combining the best prevention, detection, and countermeasure techniques to provide the requisite security through a defence-in-depth approach. We conclude that it is likely, however, that a small subset of hardware will always need to be trusted to provide the root of trust for any Hardware Trojan resistant system developed.

2 Threats

Due to the large number of insertion vectors and potential low-level actions a Hardware Trojan could perform, the danger posed by this threat is grave. Hardware Trojans are persistent; once a system has been infected the threat remains whenever the system is powered on. Hardware Trojans have the potential ability to undermine confidence in all modern electronic systems, by infecting and changing the behaviour of any integrated circuit. The effects of a Hardware Trojan can range from simple targeted attacks through to sophisticated attacks that provide footholds for higher level software attacks. Example targeted attacks could include performing a bit-flip that changes the integrity of stored data, weakening the functionality of cryptographic cores, or leaking confidential information. Systems may also be infected by multiple Hardware Trojans that can together subvert a supposedly secure system.

Although the threat is ultimately bounded by the actions a Hardware Trojan can take, considering the insertion vectors and activation mechanisms at the same time gives a greater insight into the power and stealth of a Hardware Trojan. Hardware Trojan threats, insertion vectors, and activation mechanisms will continue to develop concurrently and will need to be countered by the best prevention, detection, and countermeasure techniques to maintain the security of deployed ICs.

When examining the threat posed by a Hardware Trojan, one must first consider its inherent attributes in order to determine its effect on an information system. There have been several Hardware Trojan taxonomies proposed to describe such attributes, with the aim to enable a systematic study of Hardware Trojan characteristics, to aid the development of detection and mitigation techniques for given classes, and to facilitate benchmarking for detection and mitigation strategies.

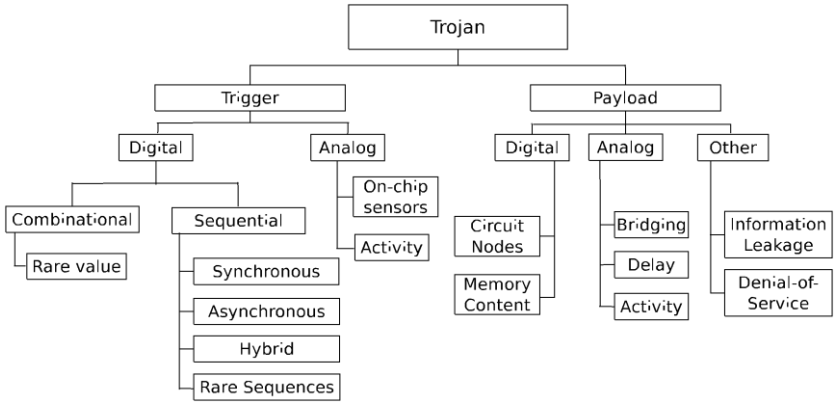

Figure 1: *Hardware Trojan Taxonomy: Chakraborty, Narasimhan & Bhunia (2010)*

Chakraborty, Narasimhan & Bhunia (2010) proposed a classification (Fig. 1), extended from Wolff, Papachristou, Bhunia & Chakraborty (2008) that is based upon trigger and payload mechanisms. Wang, Tehranipoor & Plusquellic (2008) proposed a taxonomy (Fig. 2) based upon physical characteristics, triggering mechanism and functionality. Rajendran et al. (2010) reorganise and extend this taxonomy further (Fig. 3), by consideration of design phase, abstraction level and location.

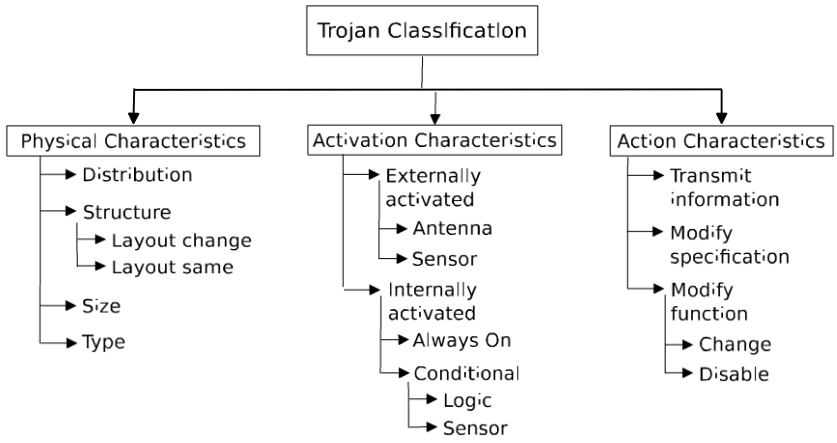

Figure 2: *Hardware Trojan Taxonomy: Wang, Tehranipoor & Plusquellic (2008)*

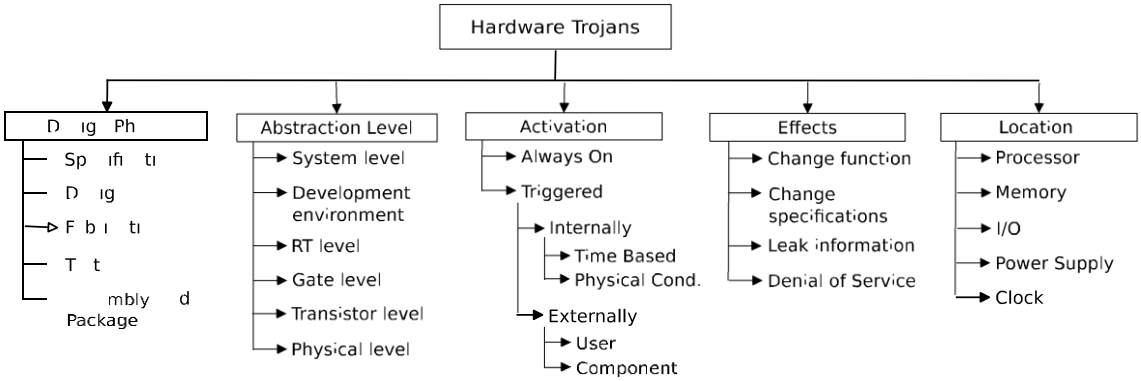

Figure 3: *Hardware Trojan Taxonomy: Rajendran et al. (2010)*

The majority of research in the Hardware Trojan area glosses over how a Hardware Trojan can influence its environment, with two primary exceptions. The Embedded Systems Challenge, run by the Polytechnic Institute of New York University (2010) gives the students the "source code" to an IC implemented in reconfigurable logic to attempt to infect with a Hardware Trojan that has to leak specific data. Anderson, North & Yiu (2008) speculate about some of the payloads that are possible when the Trojan is located in different ICs; for example, in a CPU, a hard disk, attached to network hardware or on a memory bus. In this section, the functionality or actions that a Hardware Trojan may enact – thus representing the true threat to an information system – is presented.

2.1 Insertion Phase and Location

There are numerous stages associated with the design and manufacture of an IC. These are typically regarded as specification, design, fabrication, testing and assembly (Karri et al. 2010) and directly influence how an adversary might introduce a Hardware Trojan. During the specification stage, system characteristics such as usage model and expected functionality are defined. The specification is then realised into specific tar-

get technologies with consideration of functional and physical constraints during the design phase. In the fabrication phase, mask sets are created and silicon wafers produced and probed to verify both functional and physical characteristics. Wafers are then finally cut into die, packaged and tested in readiness for deployment and monitoring. Chakraborty, Narasimhan & Bhunia (2010) assert that the only stages not vulnerable to the insertion of Hardware Trojans by an adversary are during specification, package testing, and deployment and monitoring (Fig. 4). All other stages, in practice, are vulnerable to security attacks due to the reliance on third party vendors for design tools, intellectual property design, manufacture and test facilities. For these same reasons, however, it can be argued that all stages in reality could be influenced by an untrusted party, for example by Trojan retrofitting during supply chain or testing. Thus the complete design cycle needs to be examined when considering effective prevention and detection strategies.

A Hardware Trojan may be added to an information system in a variety of locations. The location is not necessarily limited to a single component but may be distributed across multiple components such as the processor, memory, IO, power supply or clock grid (Karri et al. 2010). A particular location influences the complexity of design, difficulty of insertion, as well as the actions or system effect of a Trojan. The emphasis for this section, is to examine such possible actions and effects and to present specific real world examples highlighting the threat of the Hardware Trojan.

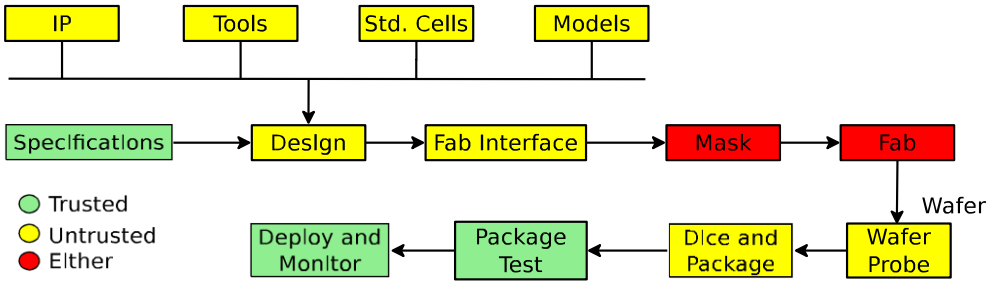

Figure 4: *Vulnerable phases of IC development cycle: Chakraborty, Narasimhan & Bhunia (2010)*

2.2 Hardware Trojan Actions

Hardware Trojans are a relatively new system security threat that extend the information system attack surface traditionally focused on software vulnerabilities. Given software security layers are written on the premise of trust in the underlying hardware, software security mechanisms may be bypassed by malicious hardware, thus presenting a complex challenge to ensure security in systems. Hardware Trojans can be implemented as hardware modifications to ASICs, COTS components, microprocessors, DSPs, or as firmware modifications to Field Programmable Gate Array (FPGA) bit-streams (Wang, Tehranipoor & Plusquellic 2008). Given the low insertion level of a Hardware Trojan, a broad range of actions are possible. These actions can be categorised into classes of modify functionality, modify specification, leak information and denial of service (Wang, Tehranipoor & Plusquellic (2008)). A particular Hardware Trojan implementation could perform any or all of these actions.

2.2.1 Modify Functionality

A Hardware Trojan that modifies a device's functionality, either through additional logic or by removing or bypassing existing logic, directly compromises the integrity of an information system. Examples include modifying stored data, or affecting a computation operation or communications channel. Functionality modifications are limitless; the actions resulting from this class of Trojan are only constrained by the resources, imagination, and skill of an adversary. Agrawal et al. (2007) present a scenario whereby a simple yet destructive Trojan could insert a fault in the Chinese Remainder Theorem (CRT) inversion step of an RSA signature computation leading to the compromise of the RSA key. Karri et al. (2010) refer to a modification to cause an error detection module to accept inputs that should be rejected.

It is certainly conceivable that errors in ICs, such as the Intel FDIV bug[2], could be reproduced by a Hardware Trojan with selective triggering to avoid detection. Karri et al. (2010) indicate how a particular Trojan could be designed to alter the order in which CPU instructions are executed, leak data through side-channel effects, and change the contents of programmable read only memory, thus introducing integrity issues. Modifying functionality could also be used to support more generic attacks. King et al. (2008) noted that "a single hard coded attack in hardware greatly understates the power of malicious circuitry" and developed an example of a modified CPU to support a raft of software attacks. These included a memory access and firmware modification that facilitated privilege escalation, login back-door and password stealing attacks.

2.2.2 Modify Specification

The modify specification class describes Hardware Trojans that attack by changing the target IC's parametric properties or an IC's non-functional specification. Parametric properties that could be affected include clock or timing parameters and power usage. This is achieved by directly influencing the intrinsic IC properties including that of wire and transistor geometries. In contrast to the modify functionality class, this category of Trojan as described by Wang, Tehranipoor & Plusquellic (2008), modifies only existing wires and transistors thus their disruptive actions would normally be restricted to those trending toward system failure. It could be hypothesised, however, that additive Trojan hardware could be coupled in such a manner as to similarly influence intrinsic IC properties yet provide extended capabilities for triggering and action characteristics.

This class of Hardware Trojan could perform a variety of actions, including limiting the processing capability of a system by modifying system clock, or by replacing computational or IO units that are functionally equivalent but have reduced throughput performance. Other specification changes that might affect performance include gate placement and routing, functionally equivalent circuits, or extraneous passive components. These performance impacts could be introduced in a load-based degradation approach whereby system performance is degraded as a result of the introduction of timing errors during high load activity. Chakraborty, Narasimhan & Bhunia (2010) provide circuit examples whereby

[2] The FDIV bug involved certain floating point divisions returning incorrect results beyond four significant digits

a bridging fault is introduced by insertion of a resistor and by increasing net delay by increasing a capacitive load property.

2.2.3 Leak Information

This class of Trojan encompasses hardware modifications that aim to transmit sensitive information from an information system to an adversary without the knowledge or cooperation of the affected information system or system user. Transmission mechanisms could use existing internal or external system paths, or alternatively, exfiltrate via side-channels. For example, Rajendran et al. (2010) note that information could be leaked by means such as radio frequency, optical, thermal, power and timing side-channels and also via interfaces such as RS232 and JTAG. Transmissions may also be hidden within the noise margins of either functional or physical features of the IC. For example, Jin & Makris (2009) leak encryption keys via wireless transmission amplitude or frequency margins that occur due to process variations, and Lin, Burleson & Paar (2009) leak data below the noise floor of the CMOS process using a spread spectrum side-channel technique.

2.2.4 Denial of Service

Low-level modifications to the hardware provides a broad range of possibilities for implementing DoS actions that range from partial service degradation to complete and permanent disabling of a device by the introduction of a "kill switch" (Adee 2008). Rajendran et al. (2010) refer to Trojans that affect service by exhausting scarce resources such as bandwidth, computation, and battery power and note that physical effects that disable or alter the configuration of a device could be temporary or permanent. A Hardware Trojan could be designed to consume excess battery energy by preventing circuits from going to sleep (Wolff et al. 2008) or by insertion of excess buffers in IC interconnections (Karri et al. 2010), thus limiting the service life of a device between charges. A Trojan could also be designed to exert control of a memory Write-Enable signal, over-writing an existing value with a random value, and causing a service side effect on a system or simply disable partial or all power supply to a device (Karri et al. 2010). Other forms of service degradation could be induced by early failure of a device. Chakraborty, Narasimhan & Bhunia (2010) provide an example circuit that generates excessive activity accelerating the aging process of an IC and thus shortening a device's life span without otherwise affecting functionality. Similarly, Rajendran et al. (2010) infer that chemical compositions may be altered to increase the electron-migration in critical circuitry like power supply and clock trees which could accelerate failures.

2.3 Hardware Trojan Implementations

Hardware Trojans have only recently received research attention, thus to date very few actual published implementations exist. Those referred to in current publications typically are simplistic, "single hard-coded" solutions that have been used solely for the purpose of experimenting with detection and countermeasures verification. Very little in-depth consideration has been given to the system-wide effects of a single or coordinated Hardware

Trojan attack, or the practicalities associated with implementing (or detecting) command and control for such attacks. The more interesting published Hardware Trojan implementations are further examined here.

2.3.1 Illinois Malicious Processor

King et al. (2008) implement two general purpose mechanisms for designing malicious CPUs. The authors show how Hardware Trojan circuits can be embedded into a CPU to realise attacks such as stealing passwords, enabling privilege escalation, and allowing automatic logins into compromised systems. The work represents a general platform to support a wide variety of attacks with the possibility of dynamic upgrades. Two malicious modifications on a CPU are performed: a memory access mechanism that allows an attacker to access protected memory regions, and a shadow mode that allows an attacker to execute hidden "firmware". One of the attacks presented that exercises the properties of these malicious modifications, a login attack that allowed an attacker complete and high level access to the machine, was implemented in only 1341 gates.

This work represents the first published Hardware Trojan implementation that can be used as a generic programmable platform for attacks. The authors introduced modifications at the VHDL level and provided both simulation and synthesised results using a 40MHz Leon 3 SPARC target platform. Some analysis of detection by consideration of analogue and digital perturbations of introduced hardware was given: the software component of the memory access mechanism is visible to the Operating System; and timing effects introduced due to the use of debugging style trap mechanisms are detectable. Additionally, a brief section on general defence against malicious processors was provided.

2.3.2 Cyber Security Awareness Week

During the 2008 Cyber Security Awareness Week (CSAW) Embedded System Challenge held at the Polytechnic Institute of NYU a hypothetical scenario was posed whereby teams were tasked to compromise an FPGA-based cryptographic device, "Alpha", by insertion of a set of Trojans but still pass validation testing. The teams were provided HDL source code and given a month to present designs. The top teams, Baumgarten, Steffen, Clausman & Zambreno (2011) and Jin, Kupp & Makris (2009), provided a mechanism to leak secret keys from an IO channel and a DoS attack respectively. Upon examination of all entries, 90% of Trojans were inserted in the design phase, 50% were activated by user input and 75% were located in IO units (Rajendran et al. 2010).

2.3.3 Malicious Off-chip Leakage Enabled by Side-channels

Lin, Burleson & Paar (2009) explore the design space of Hardware Trojans and propose a design that is less than 50 gates in size to generate power side-channels, suitable for covertly leaking secret information. The technique, Malicious Off-chip Leakage Enabled by Side-channels (MOLES), is implemented in an AES cryptographic circuit targeting a 45nm CMOS technology. Spread spectrum techniques were employed in the MOLES design capable of leaking multi-bit information below the noise floor of the power level

of the host IC to avoid detection. The authors claim the technique would be resistant to most detection strategies such as optical inspections, functional tests and physical fingerprinting analysis. Although the leakage circuit has a low gate count, computational effort for recovering the leaked data given the low signal to noise ratio and process variation aspects is noted to be a critical issue.

The authors provide a generalised design methodology for implementing the described MOLES circuits, backed by a mathematical description of detection theory for differential power analysis required for multi-bit key extraction. Results are based upon simulations of short key length (8-bit) extractions only, falling well short of realistic key lengths. The authors, however, note practical issues for the approach relating to large key sizes (256-bit) due to the number of power traces required to generate an acceptable level of SNR for reliable recovery.

2.3.4 Cryptographic Hardware Trojans

Agrawal et al. (2007) experimented with two simplistic Hardware Trojans embedded in an RSA encryption circuit to analyse side-channel effects. The circuits were comprised of a simple counter that disabled the IC after a set threshold and a comparator that monitors a data bus or register against a fixed value and alters computation upon a match. The authors theorise how such circuits could be difficult to detect, and be used to take actions such as disabling the circuit, leaking secrets or creating glitches to compromise the integrity and security of the larger system to which the IC belongs.

Jin & Makris (2009) provide an example of an information leakage Hardware Trojan that targets a DES encryption core. The design extracts the 56-bit encryption key one bit at a time, and leaks it by hiding one bit in each 64-bit block of transmitted data. After the transmission of only 56 ciphertext blocks, the entire key will have been broadcast, thus compromising the encryption. The extracted key is physically hidden in the wireless transmission amplitude or frequency margins allowed because of process variations, thus ensuring adherence to designed functional specifications.

2.3.5 Exploiting Semiconductor Properties

Shiyanovskii et al. (2009) describe a new type of Hardware Trojan, a "reliability based Trojan", that can be induced by intentional modification of fabrication processes to accelerate wearing in CMOS devices. These process modifications can keep the initial performance parameters of the circuit within the accepted process variation, thus typical production tests would not detect the modified properties. Such Trojans can exploit the following wear processes: Hot Carrier Injection(HCI), Oxide Breakdown (OB), Negative Bias Temperature Instability (NBTI) and Electron-Migration(EM) and could be used to construct a DoS class Hardware Trojan, including a gradual degradation of performance, or early wear-out of certain parts of the IC.

3 Trigger Mechanisms

There are many ways Hardware Trojans can be inserted into a design, and many different direct and leveraged threats that they pose, as covered in Section 2. Once inserted into a system most Hardware Trojans will lie dormant until activated (or triggered) to perform malicious activity. Activation can be any mechanism, overt or covert, random, directed, or predetermined that elicits a change in state or behaviour of a Trojan. This activation phase is important as it provides a vector for detecting and countering Hardware Trojans. During different verification phases of IC design, an attempt can be made to trigger Hardware Trojans. Typically this is through functional validation testing, or state-space exploration involving the inputs, outputs, and internal logic of the design. Triggering a Trojan during testing may help to identify the presence of the Trojan in the design. Much research has been done in detecting Hardware Trojans based on their trigger function; this is covered in more detail in Section 5. If a Hardware Trojan lies undetected in a system, then countermeasures can be deployed to protect against activation; typically this might include utilising data guards or hardening the architecture cognisant of specific triggers. Section 6 covers methods for countering Hardware Trojans by understanding potential trigger mechanisms.

Understanding the manner in which Hardware Trojans can be triggered is important if we want to fully understand the threat that they pose, and a brief taxonomy of trigger mechanisms, based on that proposed by Rajendran et al. (2010) and Wang, Tehranipoor & Plusquellic (2008) is presented here.

3.1 Internally Triggered

Internally triggered Hardware Trojans rely on some specific internal state of the target device being reached. The most common methods are combinational and sequential activation.

3.1.1 Combinational Activation

A Hardware Trojan is activated when certain values are detected simultaneously at specific internal circuit nodes within a device – a trigger state. This type of trigger mechanism can be implemented *solely* by combinational logic. Waksman & Sethumadhavan (2011), who call this a "single-shot cheat code", give the example of a specific address on a bus, e.g., 0xdecafbad, triggering a Hardware Trojan. In reality the combinational activation may require a much larger set of nodes to be simultaneously activated to a particular state, e.g., a particular state of a set of internal registers, combined with a specific word on the data bus, combined with a specific word on the address bus. Tehranipoor & Koushanfar (2010) also describe particular input patterns being used to activate Hardware Trojans, e.g., combining data, control, address, and self-test inputs.

Jin, Kupp & Makris (2009) used a number of combinational triggers in their design for the 2008 Embedded Systems Challenge, including detecting the overflow of an input buffer and also triggering whenever an encryption key was changed.

3.1.2 Sequential Activation

Sequentially triggered Hardware Trojans rely on a sequence of events occurring for activation. Compared with combinational activation, a sequentially triggered Trojan has a massively increased state-space to use – this comes from the fact that the trigger mechanisms can now be implemented using state machines. Chakraborty, Narasimhan & Bhunia (2010) point out that this sequence is usually of rare logic values at internal nodes and, because of the logic depth provided by the state machines, these sequences prove a lot more difficult to detect during testing and verification of an IC.

The simplest sequential trigger is a synchronous hardware counter within the design that activates after a certain number of clock cycles; Waksman & Sethumadhavan (2011) refer to these as *ticking time-bombs*. Chakraborty, Narasimhan & Bhunia (2010) build on this, discussing asynchronous sequence counters that are incremented by specific events, e.g., a rising transition at the output of an AND gate. Chakraborty, Narasimhan & Bhunia go on to suggest combining the synchronous and asynchronous triggers to form hybrid activation mechanisms.

Waksman & Sethumadhavan (2011) refer to these as "sequence cheat codes" and give the example of the bytes 0xd, 0xe, 0xc, 0xa, 0xf, 0xb, 0xa, 0xd, arriving over eight different clock cycles triggering a Hardware Trojan. The bytes need not arrive over consecutive cycles and a patient Trojan could monitor inputs and internal state for much more complex sequences of events.

The complexity of a sequential based trigger is in the hands of a Trojan designer. The side-effects of the complexity include the power drawn by the Trojan and the number of logic gates required to implement it. Internal sequential triggers have also been proposed that take advantage of physical or analogue effects within an IC. For example, Rajendran et al. (2010) indicate that monitoring a chip's temperature or power consumption could be included within trigger hardware. Further, Chakraborty, Narasimhan & Bhunia (2010) give the specific example of extra inserted capacitance that is charged through a resistor. The capacitor is charged depending on activity of the surrounding logic, which in turn could be induced through arranging specific IC activities. The Trojan is triggered based on the charged capacitor reaching a certain threshold. The notion of triggers being either analogue or digital has been discussed further by Chakraborty, Narasimhan & Bhunia (2010).

Jin, Kupp & Makris (2009) used a number of individual sequential triggers to activate different Trojans, including detecting the keyboard input string "New Haven", detecting a key-code for the *F12* key, detecting the pattern "moscow" within input data, detecting the key-code for the CAPS lock key, and counting a specific number of characters transmitted out an RS232 port.

Chen et al. (2008) investigated a "content and timing" based Hardware Trojan trigger, where the Trojan can only be activated if the correct content is observed at the correct time. Interestingly for a small design, they show that the testing time required to reliably activate their trigger is in the order 3×10^{35} years – they use the combination of detecting certain key-codes for keyboard presses over specific time intervals. Chen et al. further develop a thermal trigger, where an input pattern that generates a lot of activity is used to drive an inverter-based ring oscillator to generate heat. A similar ring oscillator is then

used to detect delays caused by this heat and subsequently activate a Hardware Trojan. Similarly Electro-Magnetic Interference (EMI), logic activity, and logic circuit power draw could be used as internal triggers.

3.2 Externally Triggered

External triggers rely on some interaction with the outside world, distinct from the system that the target device is integrated within. The power of external triggers, according to Wang, Tehranipoor & Plusquellic (2008) is that the activation can come at any time from a source that is external to, and independent of, the target device. Wang, Tehranipoor & Plusquellic go on to give specific examples of embedding a receiver or antenna within a target device. Tehranipoor & Koushanfar (2010) specifically identify on-chip sensors that could monitor the external environment, including sensing temperature, voltages, EMI, humidity, and altitude. These triggers are known as side-channel triggers, akin to techniques for obtaining information from a target electronics device, without interfering with the device (e.g., Fan et al. (2010)). Side-channels also provide a method for detecting the presence of Hardware Trojans; these methods are further developed in Section 5.

Other external triggers include physical interaction with the target device. Rajendran et al. (2010) include external input provided by a user, e.g., buttons or switches that could be attached to a target device. A trigger may also come from another component that is externally connected, e.g., a connected memory device. Another specific example is detailed by Jin, Kupp & Makris (2009), who connect to an extra (unused) external port within the target device to communicate with an embedded Hardware Trojan.

3.3 Always On

Some Hardware Trojans are always active and are not turned on or off by a specific trigger. Other Hardware Trojans may only make a subtle change to the specification, functionality, or timing of a system and hence not require a trigger. For example, leaking data through a circuit-activity-based side-channel could always be occurring inside a particular IC.

Other *Always-On* Hardware Trojans may have a more subtle trigger mechanism. Wang, Tehranipoor & Plusquellic (2008) discuss modifying an IC's geometry so that certain nodes or paths have a higher susceptibility to failure – here the trigger mechanism is a gradual occurrence as circuit performance degrades. Shiyanovskii et al. (2009) delve much deeper into these mechanisms, where devices on a wafer are modified to wear out after a certain time period, typically within a few months to years of operation – these are the so-called "reliability based" Hardware Trojans. Shiyanovskii et al. give specific examples of intentional modifications of the fabrication process that can affect a number of wear-out parameters, as detailed in Section 2. They show the fabrication factors that affect these mechanisms and also note that post-fabrication testing does not test for these time-based early wear-out effects. The triggering of these *Always-On* Trojans is probabilistic.

These time-based early wear-out trigger mechanisms are very difficult to detect not only because of the time-based nature of their activity, but because they can be implemented in the noise margin of the CMOS semiconductor manufacturing process. Wang, Tehranipoor

& Plusquellic (2008) note that there are no activation side-effects for *Always-On* Trojans, e.g., no power or temperature effects, as the Trojan is always on.

3.4 Trigger Design Issues

A Hardware Trojan designer can easily create a trigger mechanism that will prove difficult to detect. This is due to the massive state-space that exists for an adversary to design a trigger within. This state-space includes all the internal nodes of a logic design, the input and output of the device, process variations, modified CMOS geometries, and analogue electronic effects. Hybrid triggers that combine some or all of these trigger mechanisms make the job of finding Hardware Trojans even more difficult.

A common assumption among researchers is that Hardware Trojan payloads will most likely be hidden behind complex triggers designed to prevent accidental activation, or activation during acceptance testing (Tehranipoor & Koushanfar 2010, Chakraborty, Narasimhan & Bhunia 2010). This is so that the Trojan payload is not able to be detected prior to deployment. Interestingly, however, having such a complex trigger may in itself prove to assist in the detection process as discussed in Section 5.

4 Prevention

Given the considerable threats posed by the presence of Hardware Trojans, one way to ensure they cannot affect a design is by preventing them from being inserted at any stage of the IC development cycle.

The best way to prevent the insertion of a Hardware Trojan into an IC is to tightly control the process from end to end. A small, trusted design team, using self built tools, will be able to specify an IC design that is free of Trojans. Taking this design to a trusted foundry (run by a small team of trusted individuals) will produce faithful, trusted instances of the specified design. Having only trusted people assemble the final product, and having the product used only by trusted users will allow for a reasonable level of faith that the original design is being used with no malicious modifications.

This chain of trust is impractical for most products, even high-grade military products. The cost, in terms of both money and time, of only using ICs fully developed in-house is prohibitive. The commercial sector's use of packaged IP blocks (analogous to software libraries) has seen the rapid development of newer, more capable electronic products (e.g. smartphones). If the Defence sector were to develop all IP in-house it would see their capabilities fall far behind those offered by the commercial sector. There may be some ICs, such as high-grade crypto chips, where the fully trusted life-cycle scenario is feasible, especially if the IC in question has a low number of logic gates. In this instance, however, it is still possible for the ICs to be stolen, reverse-engineered, and re-birthed as modified ICs if the supply chain is untrusted.

Prevention is the first chance to counter the threat of Hardware Trojans. It is a vital link in a defence-in-depth strategy. There are all the usual policies, procedures, and best practices that can be used to maintain control over the IC development process: utilising trusted individuals, design tools, and trusted fabrication facilities as already described. Some specific research describing novel methods for preventing Hardware Trojans at different stages of the IC development life-cycle has been done. This research has looked at prevention during the design, fabrication, and post-fabrication stages of an IC.

4.1 Prevention at Design

During the design stage, Hardware Trojans may be added by an adversarial member of the design team, by untrusted EDA tools or by including untrusted third-party Intellectual Property (IP) modules in the design.

The ability to create trusted circuits using untrusted EDA tools is addressed by Potkonjak (2010). The proposed solution fully accounts for the use of all hardware resources at all times, i.e., on all clock cycles. Not only must all resources be used, but they must also be *required* to be used for correct functionality of the IC. This allows no room within the hardware for additional Trojan hardware.

This technique capitalises on the observation that while it may be difficult to completely specify a design so that all resources are fully utilised, it is relatively simple to check whether or not a given design satisfies this requirement. As such, the author proposes using untrusted, commercial CAD tools to create the design, and a small, self-built (hence

trusted) tool to check that a given design does indeed satisfy the requirements. The primary problem with this approach is that it is entirely possible (see (Baumgarten et al. 2011) for an example) to build a Hardware Trojan almost entirely from logic that already exists in a design. While it may be relatively simple to develop a tool to check that a given design does use all available hardware resources, ensuring that their are no malicious effects of that design would seem to be a more difficult proposition.

A similar technique is detailed by Chakraborty & Bhunia (2009) but includes the use of obfuscation techniques. The correct functionality ("normal mode") is hidden behind a secret initialisation sequence. Any deviation from this sequence transitions the IC into an unrecoverable "obfuscation mode" in the state graph. One difference between this technique and the one described by Potkonjak is that this obfuscated technique uses dead-end states rather than attempting to soak up all available logic gates into correct operation. This makes it easier to produce, but would also make it easier to modify without consequence. It does not protect against someone modifying the source, nor does it prevent a dedicated reverse-engineer from analysing the design at a post-fabrication stage.

4.2 Prevention at Fabrication

The issue of untrusted fabrication is addressed by Love, Jin & Makris (2011). They propose a system that has an IP Consumer providing both a hardware specification and a list of "security-related properties". Both the IP Consumer and the IP Producer have to agree on a translation of these properties into a formal mathematical codification in a theorem-proving language. As the IP Producer writes the Hardware Description Language (HDL), they also produce the formal proof that the specified hardware fulfils all required properties. This can then be checked by a theorem prover when the IP is delivered to the IP Consumer. This idea is similar to the software process Proof Carrying Code (Necula 1997).

Tying a formal model of a design to the specification of the design may result in a more correct implementation. However, as noted by Love, Jin & Makris, it is left to the IP Vendor to create the formal model. This assumes that the IP Vendor is reasonably trustworthy and will not add Trojans to either the design or the proof. An untrustworthy IP Vendor may add a Trojan to both the design and proof that is difficult to detect. This is similar to the code produced for The Underhanded C Contest (XcottCraver 2009). Not only is it difficult to specify all the security-related properties that you want to be addressed by the hardware, but there are also new attack techniques being developed all the time, techniques that could bypass all identified security properties.

4.3 Prevention at Post-Fabrication

An approach whereby some of the IC's design is implemented by reconfigurable logic (to be specified post fabrication) is described by Baumgarten, Tyagi & Zambreno (2010). Reconfigurable logic is placed between the outputs of some ICs and the inputs of other ICs, disguising some of the design from an attacker who has access to the RTL. This approach may be seen as either a preventative measure or a technique for operating in the presence of Hardware Trojans; as such it is also detailed in Section 6.3. In terms of

its preventive attributes, it leaves an attacker uncertain of the exact workings of the IC until after the reconfigurable logic has been programmed. This cuts down the attacker's window of opportunity.

Even given best efforts it is very difficult to completely prevent the addition of Hardware Trojan logic to ICs. The best that can be achieved is a first step in a combination of steps to counter the presence of Hardware Trojans.

5 Detection

As detailed in Section 4, it is not possible to completely prevent the insertion of a Hardware Trojan into an IC during a typical design flow. Where preventative measures are used to protect against Hardware Trojans being inserted into a design or device, detection mechanisms are used to discover the presence of a Hardware Trojan.

Once detected, a Hardware Trojan may be removed from a design (if detected in the RTL), the IC could be set aside so that it is not used, or the IC may still be used, operating in the presence of the Hardware Trojan (see Section 6 for more details on this). Depending on the detection mechanism used, a Hardware Trojan may be either definitively identified, or a statistical measure may be provided indicating the probability that the design or IC has been tampered with.

Traditional IC test and verification is targeted at performing acceptance tests and ensuring an IC performs as specified. Generally, however, it does not test for the addition of extra functionality. Given the state-space that extra functionality can hide within, this would be a tough task for all but the smallest logic designs. There is still a very real possibility that a Hardware Trojan will not be picked up during testing, but will be activated once the chip is in use. Abramovici & Bradley (2009) provide a strong argument that "we cannot guarantee that ICs deployed in the field are Trojan-free".

There is no "magic bullet" for detecting all Hardware Trojans. Most current research focuses on detecting Trojans post-fabrication; the fabrication process is currently seen as the weakest link in the IC development cycle. Little research has been done in detecting Trojans within RTL, prior to synthesis[3], or during fabrication. At this stage of the development cycle the designers are thought to be trusted; detecting Hardware Trojans therefore requires a trusted audit process, including an RTL design review and simulation.

There are many and varied techniques to detect Hardware Trojans, however, these techniques are really only capable of identifying a specific class of Trojan. Any individuals designing Hardware Trojans would be trying to evade existing and new detection mechanisms as they are researched and developed, in an arms race similar to that now being experienced in the anti-virus industry. Presented in the remainder of this section are some of the state-of-the-art detection methods, with reference to Chakraborty, Narasimhan & Bhunia's (2010) taxonomy (Fig. 5).

5.1 Destructive

Destructive methods of Hardware Trojan detection completely destroy the IC that they examine, lessening the usefulness of such techniques.

In order to have a very high degree of assurance that there is no Hardware Trojan in a given IC, it can be completely reverse-engineered, however reverse-engineering a complex modern IC is a time consuming and expensive process. Reverse-engineering is generally performed by Chemical Metal Polishing followed by Scanning Electron Microscope (SEM)

[3] Converting a design described at the RTL level (HDL source, or schematic) into a binary technology level net-list.

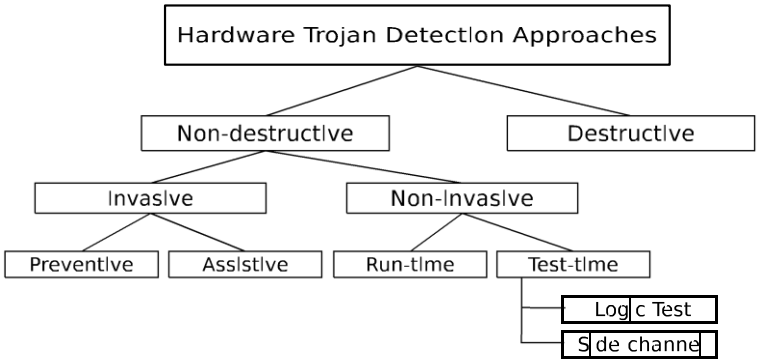

Figure 5: *Hardware Trojan Detection Techniques: Chakraborty, Narasimhan & Bhunia (2010)*

image reconstruction and analysis. Generally, the determination of the "correctness" of a chip is performed through visual comparison with a known good example or "golden reference". However, if a Trojan has been added prior to fabrication (and is therefore to be found in all manufactured ICs), visual comparison will not work. In this case the IC would have to be completely reverse engineered through the reading of the logic gate layout and reconstruction of an RTL description. This makes the reverse engineering problem much more difficult.

Hardware Trojan modifications might only be placed in a handful of distinct IC instances. Under these circumstances, reverse-engineering can be used to determine if an IC is free of Trojans only up to a certain level of assurance. For example, Agrawal et al. (2007) use destructive reverse engineering to find known good ICs. Before being reverse-engineered, a random sample of ICs from a batch are "fingerprinted" using side-channel information such as power, temperature and electromagnetic profiles. Once a consistent set of parameters is obtained, all of the sampled chips are then reverse-engineered to ensure that they are not infected by Trojans. The fingerprint can then be used in a non-destructive test on the rest of the chips in the batch. This approach suffers from several problems. A Hardware Trojan may be realised by the addition, deletion or modification of as few as two logic gates (Sturton et al. 2011), while modern ICs may consist of *billions* of such gates. Finding this "needle in a haystack" requires complete reverse-engineering at the gate level of the IC. In addition to this, there is no guarantee that ICs that have a Hardware Trojan will generate a different fingerprint to those without.

5.2 Non-destructive

Non-destructive methods of Hardware Trojan detection do not destroy the IC being examined, and are classified as being either invasive, or non-invasive. Non-invasive techniques leave the design unaltered, while the invasive techniques modify the design in order to embed features to assist with Trojan detection.

5.2.1 Invasive

Invasive techniques can be split into two branches, preventive and assistive. Measures that are used to prevent the insertion of Hardware Trojans prior to manufacture are detailed in Section 4.

Assistive techniques are used to make Hardware Trojans easier to detect in post-fabrication testing. Chakraborty, Paul & Bhunia (2008) have proposed a design that aims to expose the presence of a Hardware Trojan in a multi-module design[4]. This is achieved through additional inputs and outputs that are added to each module. The extra inputs provide a "key" which transitions the module into "transparent mode". In this mode, the module executes self-testing circuitry, designed specifically to test rare events and low-probability values. It then outputs a signature, which is a combination of the provided input key and the results of its self-test. This signature is then provided to the next module in line as its input key. In this manner a single "special" input key provided at the multi-module design's primary input tests the entire module, and the result can be determined by a single value at the design's primary output. The authors claim that this method is useful against an attacker who has "information about the functionality and logic structure" of the IC.

Using such specialised logic, designed to test the extended state-space where a Hardware Trojan may lie, will, in practice, provide very little protection from a targeted Hardware Trojan. As discussed elsewhere in this section, the likelihood of detecting a well-crafted Hardware Trojan is very small. In addition to the low probability of detection, this technique also relies on the Trojan having been inserted into the design at a very specific stage. The attacker would have to insert the Trojan *after* the functional design of a given module, but *before* the module designer then designed the fingerprinting logic for that module.

Salmani, Tehranipoor & Plusquellic (2009) propose a procedure to insert dummy flip-flops into logic to increase Hardware Trojan activity, making for easier detection using side-channel techniques. Other researchers also suggest logic additions that will make it easier to detect a Hardware Trojan utilising side-channel analysis, e.g., Li & Lach (2008) add extra logic for characterising delay times within an IC.

Das et al. (2010) focus on preventing malicious writes to external memory using a combined hardware and software model. They add extra Gate-keeper logic and modified software, that can check all writes to memory, thus allowing illegal writes to be detected and acted upon. This is a run-time detection mechanism detailed further as a counter-measure in Section 6.1.

5.2.2 Non-invasive

Non-invasive Hardware Trojan detection is done by comparing the performance characteristics of an IC with a known good copy. Detecting Hardware Trojans in a non-invasive manner can be done either at runtime or at test-time. The run-time detection mechanisms cross-over into the countermeasures, as once a Trojan is detected at run-time there is the

[4] e.g., a design consisting of an ALU, memory, control logic, and address decoder.

opportunity to try and continue operating, working around the Trojan. The test-time detection methods attempt to enhance traditional IC testing, or use side-channel analysis.

5.2.2.1 Runtime Bloom, Narahari & Simha (2009) detail a Hardware Trojan detection approach that uses both hardware and software. This strategy only attempts to detect two attacks. The first is a DoS attack, which they detect by using a small custom hardware guard which sits on the memory bus. The guard is programmed to respond to periodic "liveness" pings. Failure to respond in a timely manner is treated as successful detection of a DoS attempt. The second attack that they can detect is a combined hardware and software attack whereby the Hardware Trojan disables memory protection so that a colluding software process can escalate its privileges. This is detected by testing whether or not unprivileged software can access memory that it should not be able to access. This approach requires the Operating System to be altered to work with the guarding hardware.

Abramovici & Bradley (2009) added reconfigurable DEsign-For-ENabling-SEcurity (DEFENSE) logic to the functional design to implement real-time security monitors. After the ICs have been fabricated, the reconfigurable logic is programmed, detailing how the device should behave. Variations from this norm are then able to be detected. Subsection 6.2 further explores this mechanism and associated proposed countermeasures.

McIntyre et al. (2009) detect the presence of Hardware Trojans by executing functionally equivalent processes on multiple hardware processing elements. The output from each of these elements can then be compared to others, allowing processes that may be affected by Hardware Trojans to be re-computed on other processing elements. The mechanics are detailed further as a countermeasure in Section 6.4. The concept of being able to detect a Hardware Trojan through a subtask calculating an incorrect output is limited. The infected hardware may provide correct outputs but also egress information in some other manner, or it may degrade the performance of the IC gradually without drawing attention to itself.

5.2.2.2 Logic Testing Given the huge logical space in a modern IC, constructing a test vector that covers the entire IC logic space is computationally infeasible. Chakraborty, Narasimhan & Bhunia (2010) offer the statistic that, even restricted to a Hardware Trojan with a maximum of four trigger nodes and a single payload node, an ISCAS-85 benchmark circuit c880 (an 8-bit ALU) with 451 gates can have ~10^9 triggers and ~10^{11} possible Trojan instances. With numbers like that, the most promising logic-testing schemes take a statistical approach.

Jha & Jha (2008) present a randomisation-based technique which probabilistically compares the functionality of the design of the circuit with the implemented circuit. The results given in this paper relate to their "random" modification of ISCAS Benchmark circuits (to "infect" the circuit) and the authors claim that their technique was able to detect 10 out of 12 modifications.

The motivation behind Chakraborty et al.'s (2009) study is to test rare occurrences on an IC rather than testing for correctness. The tester determines rare states that can occur within a circuit module. Testing then focuses on repeating test vectors that excite these

states. The authors claim that this technique reduces test length by ~85% over a range of benchmark circuits and achieves better coverage than a purely random test set. This technique is based on the assumption that Hardware Trojans will more likely be activated by combinations of rare states within a design.

5.2.2.3 Side-Channel Analysis Rather than attempting to trigger a Hardware Trojan directly in order to detect its presence, side-channel analysis uses the fact that the Trojan trigger mechanism itself changes some characteristics of the IC, whether or not it activates the Trojan payload. The amount of power that a section of the IC uses (power draw), the amount of heat produced in certain locations or the length of time that certain parts of the IC take to perform their processing (path delay) are examples of the secondary IC characteristics used to perform side-channel analysis. This type of analysis appears to have the best detection likelihood, as the Trojan does not need to be activated (deliver its payload) in order to be detected.

Agrawal et al. (2007) present a broadly representative example of this type of detection mechanism. Some known good copies of the IC are obtained and "fingerprinted" using one or more side-channel parameters. Other chips can then be tested against these fingerprints. Various statistical techniques can then be used to pick out statistically significant (but well hidden) differences. The authors specifically use power draw as the primary side-channel. The obvious difficulty is in ensuring that the ICs used to generate the initial fingerprint are Trojan-free. Power supply transient signal analysis is used as the side channel by Rad, Plusquellic & Tehranipoor (2008). They aim to determine the smallest Hardware Trojan that they can find using this technique, which turns out to be three additional gates. The tests were performed on a simulator of a particular benchmark circuit.

Banga & Hsiao (2009) propose a technique that is able to magnify the side-channel differences (based on power draw) between circuits infected with Hardware Trojans and those that are not. A "sustained vector technique" is used, which repeats (sustains) certain inputs in order to allow genuine circuits time to reach a stable state – a process that the authors call Toggle Minimisation. Next, infected regions within the design are isolated by looking at the differential power draw when a new test vector is applied. Large changes in the differential power draw could be indicative of extraneous hardware, i.e., they are trying to identify circuits that are active when they are not supposed to be.

Path delay was the measurement used as the fingerprint – the side-channel analysed – by Jin & Makris (2008). The authors categorise Hardware Trojans as having either implicit or explicit payloads. Explicit payloads directly affect the circuits that they are attached to (e.g., altering the value of a control or data signal). Implicit Hardware Trojan payloads do not make changes to the circuitry that they are attached to; they may instead leak data via a side-channel, or perform a DoS attack once triggered. Jin & Makris claim to be able to detect 100% of explicit Hardware Trojans and 36% of implicit Hardware Trojans. Their experiments were conducted on a simulator, and their Trojans were simple modifications designed specifically to affect power draw and path delay. Similarly, both path delay and leakage current are used as the side-channel for analysis by Potkonjak et al. (2009). Wang et al. (2008) use current charge integration from multiple current measurement points on an IC, and then localised current analysis to detect Trojan circuitry. The current analysis is once again compared with a golden reference and the authors claim to be able to detect

added Trojans "as small as a few gates"; about 0.1% of the circuit area.

The primary problem with the side-channel analysis methodology for detecting Hardware Trojans is that it depends entirely on having an authentic golden reference IC that can be used for comparison and benchmarking. Where an IC has had a Trojan added to it at any stage up to manufacture, and hence is to be found in each instance of an IC, these approaches will not work. Additionally, the search space for these detection methods may be very large. Although good work has been done to manipulate inputs and use advanced statistical methods to amplify differences, the likelihood of detecting such a difference is very small, especially against a well written, targeted Hardware Trojan.

6 Countermeasures

The combination of state-of-the-art Hardware Trojan prevention and pre-deployment detection mechanisms still cannot provide complete certainty that manufactured ICs or reconfigurable logic designs are free of Trojans. Given the large number and types of threats and the massive state space for Hardware Trojan triggers, some researchers have focussed on the problem of maintaining secure operation in the presence of Hardware Trojans. Specifically, these researchers have been looking at implementations that can prevent activation of certain Trojans, or still allow useful trustworthy operations to be completed in the presence of an *unknown* Hardware Trojan.

Successful countermeasures should allow hardware to be oblivious to inserted Trojans and even allow COTS components to be used to construct Hardware Trojan resistant, trustworthy computing systems. To date no single, generic countermeasure has yet been developed or proposed that would allow an IC to operate in a trustworthy manner in the presence of an arbitrary Hardware Trojan. This section further examines the state-of-the-art in Hardware Trojan countermeasures, including protection profiles, implementation details, operational details and analysing the general applicability of the countermeasure.

There are software based mechanisms that can protect the confidentiality and integrity of data as it is stored, processed or transmitted by some ICs within a system, e.g., using an encrypted file-system can protect data from Hardware Trojans that might reside within an ATA controller or hard disk IC. There are currently no commercial ICs that integrate any Hardware Trojan countermeasures. Those countermeasures that are currently being researched often only protect against a single class or sub-class of threats and/or triggers. Broader protection is usually achieved via a defence-in-depth strategy, targeting specific Trojan actions and trigger mechanisms with independent countermeasures and then combining these measures into a protection strategy. The proposed and experimental mechanisms that exist can be broadly categorised as either data guards, new RTL-level architectures, reconfigurable architectures, or part of a replication, fragmentation and voting strategy.

6.1 Data Guards

By guarding data (including CPU instructions), a designer is attempting to prevent a Hardware Trojan from being activated and/or prevent a Hardware Trojan from directly accessing and utilising any (unencrypted) sensitive data. A guard can control what form data takes as it is stored or transmitted within or between ICs or logic modules, affecting the manner in which any Trojans can interact with the data.

6.1.1 Generic Guards

Waksman & Sethumadhavan (2011) introduce a number of guarding techniques to prevent Hardware Trojan activation. Some of the proposed techniques have been implemented in the Zesto x86 simulator.

Bus scrambling is used to prevent Hardware Trojans receiving activation codes. It is used for non-computational units that handle data, e.g., a memory controller or DMA controller. Waksman & Sethumadhavan propose using simple encryption schemes (e.g., XOR with a pseudo-random number) to obfuscate data, only looking to secure it for a short period of time – reasoning that a lack of hardware resources would prevent decryption by a Hardware Trojan. A manually verifiable (trustworthy) circuit needs to implement the scrambling.

Waksman & Sethumadhavan only emulated the performance effects of scrambling by adding parameterisable delays in the caches and memory controller. Whilst bus scrambling will prevent simple Trojans from being activated, it can still lead to probabilistic activation, e.g., if a simple 32-bit data trigger is used, activation will likely occur within 2^{32} bus cycles. A more controlled approach might be to re-map all inputs into a completely functionally validated state-space. A strong encryption scheme should also be used to prevent sensitive data from being leaked.

For computational units within an IC, obfuscating the data will affect the unit's ability to produce a correct result as the results of the computation will be affected by the scrambled inputs. Waksman & Sethumadhavan propose to use homomorphic cryptography (Gentry 2010) to allow computational units to operate on encrypted data inputs. An encryption function is defined that is homomorphic over the computational function, allowing the computational unit to calculate the correct result while only processing encrypted data. The result can then be decrypted to obtain the desired computational result.

Implementing and computing these homomorphic functions is inefficient and non-trivial; it is difficult to construct a general purpose homomorphic encryption scheme (Gentry 2010). Similar to bus scrambling, the encryption and decryption units would need to be implemented in trustworthy (verifiable) hardware. Waksman & Sethumadha- van did not simulate the homomorphic encryption scheme proposed, but instead gave a simplified case study based on an RSA cryptography unit. The use of Garbled Circuits (Järvinen et al. 2010) might be considered as an alternative approach for the obfuscation of data for computational units.

Waksman & Sethumadhavan also propose a time-guard to prevent Hardware Trojan activation within a validated functional state space. An IC is fully functionally validated for a given number of cycles, and is then periodically powered on and off within this testing epoch, ensuring that no timer based Trojans can be activated. A light-weight context saving mechanism is used to ensure processing continuity. Waksman & Sethumadhavan reason that any Hardware Trojans that are dormant for the complete testing state-space (a state-space of time and inputs) will forever be dormant during this same period under operational conditions.

This works for input-based and time-based triggers. However for non-volatile memory triggers, accumulation triggers (e.g., Chakraborty, Narasimhan & Bhunia's (2010) capacitor charging trigger), side-channel triggers, degradation effects and external triggers (e.g., RF) this type of countermeasure will not be adequate. The authors get around the first of these issues by burning-out any non-volatile memories post-tapeout or by visual inspection. Another solution might be preventing the use of an ASIC process technology that can be used to implement non-volatile storage. Any hardware used to perform the context saving or IC resetting will need to be trusted. A simple emulation of flushing the processor

state was performed by the authors.

Further guards are proposed that would randomly reorder events and insert dummy events into input sequences to different modules, e.g., to a memory controller so that the sequence of stores/loads is disrupted. These guards protect against sequential triggers being activated. Patient Hardware Trojans should still be able to get sequences through; however using complete validation and power resets could prevent this from happening. Sequences that naturally have breaks in them may also escape this countermeasure.

Finally, Waksman & Sethumadhavan propose utilising multiple versions of an untrusted module or IC from multiple different designers. The output from the modules can then be checked and compared cycle-by-cycle – effectively voting on the output. This has the drawback of high cost in terms of silicon area and power consumption. Replication and voting are further covered in Section 6.4. Whilst Waksman & Sethumadhavan attempt a complete coverage against Trojan activation they completely ignore side-channel, timing, and external trigger mechanisms.

6.1.2 Specific Bus Guards

Kim, Villasenor & Koç (2009) developed a custom System on Chip (SoC) bus architecture that proposes changing master/slave bus architectures to detect Hardware Trojans that try to lock the bus utilising standard activities, e.g., master not releasing the bus, blocking interrupts, slaves continuously waiting. Basic updateable counters are used to implement simple heuristics to detect these types of activities and then blacklist suspect master/slave devices on the bus and provide reporting options. The architecture was tested using the Advanced Micro-controller Bus Architecture (AMBA) from ARM.

This countermeasure is targeted at a specific architectural feature and a subset of Hardware Trojan activities, i.e. preventing Trojans from interfering with the correct operation of the SoC bus and hence affecting the performance of the SoC.

A number of researchers have looked at placing guards on the memory bus in a processor architecture. The reasons for guarding the memory bus include both Hardware Trojan activation and data leakage.

Das et al. (2010) insert *shadow* writes, i.e., companion writes for all *store* instructions into binaries. The addresses of these *shadow* writes are an encrypted version of the original addresses. A Gate-Keeper core (hardware) residing on the memory bus checks that all memory writes are followed by their corresponding write to an encrypted address. Thus, the Gate-Keeper can ensure that the only writes that occur are legitimate in the original binary, preventing a Hardware Trojan utilising the bus to leak sensitive information. A full prototype was developed that could execute x86 static binaries and have the Gate-Keeper detect all shadow writes. The Gate-Keeper would necessarily need to be trusted. This approach recognises that data egress in any large amount will generally require writing that data to memory (to send across a network for example). However, data egress may also occur utilising other methods, e.g., through altering power draw or changing timing characteristics. This mechanism crosses over between detection and countermeasure, i.e., the IC is operating in the presence of Hardware Trojans and the detection occurs at run-time.

Bloom et al. (2009) introduce a double guard between the CPU and data bus produced using independently keyed adversarial hardware. The two guards check each other for correctness. Executables are encrypted once with each key for the separate guards, with data being decrypted on its way to the CPU and encrypted on its way back to memory. The system relies on there being no collusion between the two guards. An instrumented compiler is also a crucial part of the system to generate the binaries, BIOS, and Operating System images prior to run-time. The double guard relieves some of the need to trust either guard; of more importance is ensuring that there is no collusion between the two guards. The countermeasure was evaluated using SimpleScalar (Austin, Larson & Ernst 2002), an open-source computer architecture simulator.

As early as 2003, Suh, Clarke, Gassend, van Dijk & Devadas had proposed their AEGIS processor architecture that could utilise untrusted peripheral components and run an untrusted Operating System. The processor needs to be trusted to perform encryption primitives, acting as a guard between itself and all untrusted peripherals. The difficulty lies with getting the AEGIS IC free from Hardware Trojans. The idea of utilising a minimal TCB, or trusted hardware circuitry to control Hardware Trojan countermeasures is vital to every countermeasure presented in this section.

These memory guard mechanisms target quite specific Trojan trigger mechanisms and threats, however some of the principles can be carried over to other buses and modules within an IC.

Beaumont et al. (2011) propose a guard that sits on the ATA bus between a CPU and a hard disk. The guard can encrypt or scramble data to prevent simple codes or sequences of data from activating a Trojan. The guard can also encrypt data so that untrusted ICs within the hard disk cannot store or leak any sensitive information. For more complete coverage, the guard would also need to mitigate against timing-channel triggers. Anderson, North & Yiu (2008) expand on this idea, introducing the concept of the Silicon Security Harness. The Silicon Security Harness involves one or more gates and monitors that can be retrofitted to hardware or system components, or that are installed as part of the architecture. The Silicon Security Harness uses these protective measures to increase the resistance against Hardware Trojans.

6.2 Novel RTL-Level Architectures

Some researchers have identified specific modifications that can be made to modern processor and IC architectures to protect against Hardware Trojans. The premise is the addition or modification of logic gates to specifically identify the presence of, or prevent the activation of Hardware Trojans.

Hicks et al. (2010) developed a hybrid hardware/software, compile-time/run-time Hardware Trojan countermeasure called *BlueChip*. *BlueChip* is a defensive strategy that includes both a design-time and a run-time component to deal with RTL designs that may have arbitrary Trojans inserted at unknown locations. An Untrusted Circuit Identification (UCI) algorithm and tool-set automatically identifies and removes potential malicious circuits, in this case from an RTL design destined for a processor IC. Suspicious circuits are identified and removed during design verification testing. Any circuits that are included in the design but that do not affect any outputs during testing are identified and

removed. The removed hardware is replaced by logic that will trigger an exception if the removed hardware is ever activated. This may occur due to potential Trojan activation or a false-positive from correctly operating circuitry that may have been removed. Low-level software will then try and recover and move forward by emulating what the hardware was trying to achieve – akin to a software trap for performing floating point emulation. Unrecoverable exceptions (nesting) are addressed by the proposal to use a very small set of 'trusted' instructions to perform the emulation.

The *BlueChip* concept has been prototyped using the Leon3 processor (Aeroflex Gaisler AB 2010) design on a Xilinx Virtex5 FPGA. *BlueChip* moves some of the onus from trusted hardware to trusted software components, leaving a truly small subset of hardware that needs to be fully verified to be used to emulate the removed hardware. The strategy is most applicable to processor designs where the exception handling and software executing mechanisms exist to handle the emulation requirements. Extending the strategy to generic IC design may require a trusted co-processor that could handle the exceptions and lead the way in emulating around any removed hardware. Since the development of *BlueChip*, Sturton et al. (2011), have developed malicious circuits that can be inserted into an IC that will evade UCI detection and also pass design-time testing.

Abramovici & Bradley (2009) identified that no existing pre-deployment mechanisms can guarantee detection of all Hardware Trojans. They propose detecting post-manufacturing tampering attacks at run-time through the addition of extra logic to an IC – an integrated self-checking IC; based on detection mechanisms. Additional DEFENSE (DEsign-For-ENabling-SEcurity) logic is added to a SoC to perform real-time, configurable security checks of the behaviour of different parts of the system, multiplexing the different parts through checking hardware. Examples include, checking illegal accesses and illegal states, DoS checks, and consistency checks (e.g., processor enters an inconsistent mode for the current state). When an attack is detected, real-time countermeasures are deployed, such as disabling suspect logic blocks. Abramovici & Bradley also propose using fail-safe states, spare logic, and check-pointing to counter any detected attacks in real-time. The DEFENSE platform has not been prototyped and providing complete coverage of Hardware Trojans during run-time security checks would be difficult. Introducing countermeasures in real-time to provide continuity of operation of the IC would also be difficult.

Deng, Chan & Suh (2009) propose a time-bounded, unique hardware checksum that is based on checking the authenticity of trusted hardware. The hardware checksum is based on low-level micro-architectural implementation details of the processor. The hardware is challenged and a checksum needs to be returned within a specific time limit. The authors suppose that an authentic checksum will not be able to be emulated or simulated within the time limit and only the authentic hardware will be able to respond correctly. The mechanism ensures that no Trojans have been added to the circuit post-fabrication. New instructions have been added to the processor architecture to support the Micro-Architecture Signature Function (MSF), which utilises on-chip architectural features to generate a unique response to a challenge. While the mechanism can be used to guarantee that the hardware is authentic, it cannot guarantee that no Trojans have been inserted into the design during the specification, design, verification, or manufacturing stages.

Specifically targeting DoS Hardware Trojan threats, Bloom et al. (2009) developed

a heartbeat function to check for continued operation of an IC. Non-cacheable memory accesses are inserted into the software, these then present themselves on the memory bus at regular, but random intervals and are used to detect whether or not the IC has been subject to a DoS attack.

6.3 Reconfigurable Architectures

Using reconfigurable logic to counter Hardware Trojans can bring significant benefits, but also presents a new set of design problems and challenges. There is a spectrum of reconfigurable logic devices, including high logic density FPGAs where most of the device is reprogrammable; platform targeted FPGAs that contain fixed semiconductor elements, e.g., PCIe endpoints, memory controllers, even complete processor cores; and custom ASICs that may contain small reconfigurable portions to perform specific functions, e.g., implementing glue logic, or implementing a custom co-processing element.

The main benefit of reconfigurable logic is the separation provided between the hardware implementation and the design implementation. In a typical IC, the design is implemented directly in the semiconductor process, whereas utilising reconfigurable logic, a standard programmable logic element or macro is implemented in the semiconductor process and the design is later implemented by programming these logic elements using a configuration bit-stream. This separation means a design can be developed almost completely independently of the hardware, and in a trusted environment, although there is still some reliance on specific peripheral features attached to the generic logic elements.

Whilst the RTL design can now be controlled, the design and implementation of the reconfigurable logic is subject to many of the same Hardware Trojan insertion threats as a standard ASIC. The main difference is that an attacker can only perform generic attacks against the reconfigurable logic architecture, making it more difficult to seamlessly interfere with, or modify the logical operation of the *configured* design. Hardware Trojans can still perform the full range of attacks, modify functionality, modify specification, leak sensitive information, and DoS. For example, logic elements can be modified to perform augmented operations, potentially introducing subtle errors into designs, or data can be leaked through peripheral devices.

The newly defined problem becomes how to best implement trustworthy designs knowing that the underlying reconfigurable logic may be infected with arbitrary Trojans, and how to protect the integrity of the design (configuration bit-stream) once it has been created, i.e. prevent the bit-stream from becoming corrupted or infected by Trojans. A generic three-step measure for securing the integrity of an FPGA bit-stream is proposed by Webb (2006). First, the integrity of the configuration is checked by reading it back, secondly, the FPGA is partially reconfigured (from an authenticated partial bit-stream) if an incorrect configuration is found, and thirdly, the FPGA uses a challenge-response protocol to notify a third party if the system has been compromised.

Dutt & Li (2009) identify recent work on protecting FPGA bit-streams and configuration memories from "upset" events. They also identify recent work indicating that while FPGA Fabrication is separated from FPGA design, reliance on third-party integration and IP modules can introduce external influences, and that while configuration

bit-streams provide inherent protection and are difficult to reverse engineer, they are not impossible to reverse engineer. Further they identify that while bit-stream encryption provides good protection, it removes the ability to do partial reconfiguration and also cannot protect against tampered IP that makes its way into the design.

Dutt & Li propose using ECC-based parity groups to guarantee the authenticity of the design. Additional logic is added to output parity from groups of Configurable Logic Blocks (CLBs) within an FPGA. This parity is checked using a trusted checking phase to uncover any design tampering. Two-stage randomisation of the parity generation is used to ensure non-predictability in the result of any given parity group.

Baumgarten, Tyagi & Zambreno (2010) protect against Hardware Trojans that may be inserted during the fabrication stage. By placing reconfigurable logic blocks between crucial elements within the design, a foundry only sees reconfigurable architecture in some parts of the design. These barriers are then programmed, or unlocked, using a securely distributed key post-manufacture to complete the design. If the location and functionality of these barriers is carefully chosen then any inserted Trojans will have difficulty activating and affecting the operation of the IC. Careful consideration would need to be given to the type of reconfigurable logic, key management, and different barrier placement heuristics. Combining fixed and reconfigurable logic can provide unique solutions to Hardware Trojan infections; the reconfigurable logic could even be used to implement local protection mechanisms.

Utilising reconfigurable logic for Hardware Trojan protection introduces a new set of design and verification challenges, moving the focus from the semiconductor process to the RTL design. Significant collusion would be required between the fabrication process and CAD tool vendors to implement an effective IC-level Hardware Trojan. Implementing a complex reconfigurable logic design undoubtedly relies on integration of many IP modules. Huffmire et al. (2007) proposed the idea of *Moats* and *Drawbridges* as isolation primitives to ensure that when connected together, the interfaces of the IP cores have not been tapped or illegally routed within an FPGA.

Further features of reconfigurable logic that may be used to counter Hardware Trojans include partial and dynamic reconfiguration of logic (Silva & Ferreira 2010), encryption of configuration bit-streams (Trimberger 2007), replication and lock-stepping of logic (Newgard & Hoffman 2010), design of architecturally variant but functionally identical logic modules (McIntyre et al. 2009), and generating unique hardware based random numbers (Kumar et al. 2008).

6.4 Replication, Fragmentation, and Voting

Effective Hardware Trojans rely on understanding the operation of an electronic circuit design at some level, from process and gate-level, through RTL-level, IC-level and electronic design. A generic Hardware Trojan countermeasure that can be deployed at many of these levels involves: replication, or duplication of logic and/or data; slicing, or fragmentation of logic and/or data; scattering, or distribution of logic and/or data; and gathering and combining of logic and/or data results, e.g., using a voting mechanism.

The effectiveness of these general countermeasures comes on three fronts: protecting

against Hardware Trojans leaking sensitive information by splitting the data up and processing it with independent logic elements; protecting against functional or specification modifications to elements by using multiple replicated, or duplicated logic; and protecting against DoS attacks by providing redundancy in operation of logic elements within the design.

The countermeasures can be deployed at various levels from gate, RTL, logic design, functional modules, and IP cores, through to the IC and macro-level devices. The protection mechanisms rely on a non-collusion assumption[5] between the replicated, or duplicated elements within the design.

Waksman & Sethumadhavan (2011) propose using logic duplication, where multiple versions of an untrusted module or IC are used from multiple different designers. The outputs from the modules are then checked on a cycle-by-cycle basis, effectively voting on the correct output. Waksman & Sethumadhavan indicate the high cost in terms of silicon area and power consumption.

McIntyre et al. (2009) utilise a method for dynamically evaluating the trust in hardware at run-time. The premise is to detect the presence of Hardware Trojans at run-time and then continue processing by removing or relying less on the suspicious elements. They propose using a multi-core processing system to take advantage of in-built redundancy, and the ability to discard cores if they are found to be untrustworthy. Functionally-equivalent, but variant, processes are spawned on multiple processing elements and the results compared. The variation in processes may be obtained from different compilation, implementation, or algorithms used. If the compared results differ, a third processing element is introduced and the three results are compared. This process is continued until agreement is reached between at least two processing elements. Processing elements that give inconsistent (wrong) results are dynamically penalised, i.e., they become less trusted and are less likely to be used.

No thought was given to the trust of the software or hardware that performs the comparison. This method could be further extended to using functionally equivalent hardware implemented as randomised variants. The method could also be applied at different abstraction levels, e.g., instruction level, gate-level, program-level, or IC-level. If performed at the instruction-level this activity could become transparent to higher levels, with the hardware, including a small TCB, taking care of the instruction-level scheduling, replication, variant processing, and voting.

Newgard & Hoffman (2010) introduce a tightly-coupled dual-processor lock-step configuration implemented inside an FPGA – an implementation of replication and voting at the macro level. Both processors receive and process the same instructions at the same time. Hardware check logic examines and compares all bus control signals on every bus transaction. If an error is detected, the system is forced into an error recovery sequence. Further development and verification of a TCB to implement the trusted checking and error recovery mechanisms would be needed to adequately counter an FPGA infected with Hardware Trojans. The method could be expanded for a larger number of processors that are either discrete ICs or embedded within a reconfigurable fabric.

[5] A reasonable assumption given the variant processes and channels available for the design, manufacture and procurement of electronic design components.

In 1991, Trouessin et al. investigated the provision of high-reliability and availability, and the preservation of data-confidentiality in large-scale distributed systems. Whilst this work did not address the Hardware Trojan threat, the techniques for fault tolerance are very relevant. The authors introduce a general approach that fragments the data into small pieces, such that on its own each fragment contains little information. The approach can be used for both data storage and data processing, Trouessin et al. cite Fray, Deswarte & Powell (1986) and Fray & Fabre (1989) as references for data storage and data processing fragmentation respectively. The fragments are then scattered for either storage or processing. Replication (redundancy) of the fragments is used to gain reliability in the system. Threshold schemes, akin to secret sharing (e.g., Shamir (1979)) are also proposed to recombine the stored or processed data. The determination of fragmentation functions for general purpose processing could be difficult and expensive to compute. The same mechanism could be implemented as discrete hardware processing elements. A TCB would be needed to handle the input and output of the processing and storage operations.

To date, research into Hardware Trojan countermeasures has focussed on preventing the activation of Trojans and detecting Trojan activity at run-time. Any countermeasure developed should protect the confidentiality of data and maintain integrity of operation of the device being protected. Existing software techniques that focus on reliability and confidentiality may be applicable, or may be applied in concert with newly developed countermeasures. There is currently no single solution that can provide complete coverage against the range of threats and triggers presented earlier. It is unlikely that such a solution will be developed, instead a combination of countermeasures will be needed to combat specific classes of Hardware Trojans within specific application domains. These countermeasures will need to be developed cognisant of both the systems within which they will be deployed and the level of protection they aim to provide. As demonstrated by Sturton et al. (2011), when new countermeasures are developed, researchers will also find ways to bypass them. This Hardware Trojan arms-race further highlights the need for a defence-in-depth approach.

7 Summary

Hardware Trojans are a present and ongoing threat to the security of electronic systems world-wide. The Australian Military has particular concern due to the outsourcing of design and manufacture of integrated electronic components and our reliance on COTS components to maintain capability. Hardware Trojans threaten to compromise the integrity of data and operations performed by any system containing integrated electronic components. The threats include functional and specification modifications, leaking of sensitive information and Denial of Service attacks.

Hardware Trojan payloads and their activation mechanisms can take advantage of the massive state-space formed by the combination of parallel logic, internal routing, and input and output, that exists within a modern IC. The resulting Hardware Trojans can remain hidden deep within the design of an IC, having very poor observability.

Efforts to prevent Hardware Trojans being designed or manufactured into ICs are still in their infancy. Much current research is focused on post-fabrication detection of Hardware Trojans, with current efforts looking at destructive techniques, side-channel analysis, and assistive logic testing techniques. Detection mechanisms are often focussed on a specific class of Hardware Trojans, with no single or combination of techniques able to provide complete coverage. A sprinkling of novel countermeasures have been developed that will allow trusted operation to continue in the presence of Hardware Trojans. These countermeasures focus on: guarding data, inputs and outputs; new RTL-level architecture features; reconfigurable logic; and replication, fragmentation and voting schemes.

Future research will have to focus on combining the best prevention and detection techniques to provide the greatest guarantee of Hardware Trojan free devices. New countermeasure techniques will have to be developed to allow these still untrusted devices to perform trusted operations. It is likely that a combination of countermeasures will be required to provide the best coverage depending on the device or system being protected. A defence-in-depth approach should allow COTS ICs to be combined with some customised logic to operate in a trusted manner. Ultimately some small trusted hardware will need to be developed to provide the root-of-trust upon which the integrity of any countermeasures are built.