This thesis has been submitted in fulfilment of the requirements for a postgraduate degree (e.g. PhD, MPhil, DClinPsychol) at the University of Edinburgh. Please note the following terms and conditions of use:

This work is protected by copyright and other intellectual property rights, which are retained by the thesis author, unless otherwise stated.
A copy can be downloaded for personal non-commercial research or study, without prior permission or charge.
This thesis cannot be reproduced or quoted extensively from without first obtaining permission in writing from the author.
The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the author.
When referring to this work, full bibliographic details including the author, title, awarding institution and date of the thesis must be given.
Towards trustworthy computing on untrustworthy hardware

Heba Salem

Doctor of Philosophy
Institute of Computing Systems Architecture
School of Informatics
University of Edinburgh
2023
Abstract

Historically, hardware was thought to be inherently secure and trusted due to its obscurity and the isolated nature of its design and manufacturing. In the last two decades, however, hardware trust and security have emerged as pressing issues. Modern day hardware is surrounded by threats manifested mainly in undesired modifications by untrusted parties in its supply chain, unauthorized and pirated selling, injected faults, and system and microarchitectural level attacks. These threats, if realized, are expected to push hardware to abnormal and unexpected behaviour causing real-life damage and significantly undermining our trust in the electronic and computing systems we use in our daily lives and in safety critical applications. A large number of detective and preventive countermeasures have been proposed in literature. It is a fact, however, that our knowledge of potential consequences to real-life threats to hardware trust is lacking given the limited number of real-life reports and the plethora of ways in which hardware trust could be undermined. With this in mind, run-time monitoring of hardware combined with active mitigation of attacks, referred to as trustworthy computing on untrustworthy hardware, is proposed as the last line of defence. This last line of defence allows us to face the issue of live hardware mistrust rather than turning a blind eye to it or being helpless once it occurs.

This thesis proposes three different frameworks towards trustworthy computing on untrustworthy hardware. The presented frameworks are adaptable to different applications, independent of the design of the monitored elements, based on autonomous security elements, and are computationally lightweight. The first framework is concerned with explicit violations and breaches of trust at run-time, with an untrustworthy on-chip communication interconnect presented as a potential offender. The framework is based on the guiding principles of component guarding, data tagging, and event verification. The second framework targets hardware elements with inherently variable and unpredictable operational latency and proposes a machine-learning based characterization of these latencies to infer undesired latency extensions or denial of service attacks. The framework is implemented on a DDR3 DRAM after showing its vulnerability to obscured latency extension attacks. The third framework studies the possibility of the deployment of untrustworthy hardware elements in the analog front end, and the consequent integrity issues that might arise at the analog-digital boundary of system on chips. The framework uses machine learning methods and the unique temporal and arithmetic features of signals at this boundary to monitor their integrity and assess their trust level.
Computing and electronics systems used in consumer electronics, medicine, aviation, industry and manufacturing, and many other sectors are all composed of a hardware part that is the physical functional part of the system, and a software part, which is a set of codes and programs used to operate the hardware. Over the years, it has been shown that software, which is more exposed to the outside world, is a likely target for adversaries attempting to compromise the security of systems mostly for financial gain but also for political, social, and espionage reasons. This eventually led to a concentration of efforts in protecting and fortifying software and the data or information that it deals with, while the security of hardware was mostly pushed aside assuming that hardware is protected by its obscurity.

Over time, researchers started realizing the unique security issues of hardware and the types of attacks computer chips are prone to. Particular hardware security concerns are attacks stemming from unsolicited hardware modification initiated by third parties involved in the current-day globalized supply chain of computer chips. Another concern is counterfeiting, where unreliable poorly-manufactured computer chips are sold as genuine ones. More recently, the possibility of using the interaction between software and hardware in causing malfunction in hardware has also emerged. Several measures have been proposed by researchers and chip designers to combat these issues. However, due to the large variation in their potential shapes and realizations, the threat of the consequences of these issues arising during the operation of systems is real and possible.

This thesis aims to face the possibility of the initiation of untrustworthy behaviour during operation by assuming that the chips in a given system are potentially untrustworthy from the very beginning. Consequently, techniques are devised to constantly monitor their behaviour and detect any abnormal action that is effectively a breach of trust. Three different frameworks for the monitoring of hardware trust and security are introduced in this thesis where one framework is concerned with direct violations of trusted operation and the other two concerned with obscured and subtle violations. The violations of concern span the main security properties of confidentiality, integrity, and availability. The frameworks were tested and verified with essential hardware parts related to on-chip communication, storage, and signal acquisition.
Acknowledgements

I would like to thank my supervisor, Prof. Nigel Topham for his constant guidance and support throughout my PhD. I always left our meetings energised, full of ideas, and ready to tackle the research issues I was facing. I will always appreciate the lengthy discussions that we had on various research topics and the freedom I was given in exploring different research directions and approaches, which I think helped me grow as a more confident researcher. His patience and his supportive attitude has made my “pandemic-era” PhD journey mostly smooth and stress-free.

I would also like to thank my second supervisor, Prof. David Aspinall, and Prof. Mahesh Marina for the valuable feedback I received from them in the annual reviews. Their comments turned my attention to blind spots in my research and made me think deeper of my approaches, the objectives of my work, and its potential impact.

Throughout my education and career, I have been fortunate to meet and work with a number of people who believed in my potential and who were generous to pass down their knowledge and wisdom, I would therefore like to thank them all, especially those who encouraged me to pursue a career in research.

Finally, I would like to thank my family; my parents; Maryam and Ahmad and my siblings; Shatha and Mohammad. I am eternally grateful for my mother, who this thesis is dedicated to, for her support, unconditional love, selflessness, and countless sacrifices. She has always been my backbone and has, from a very young age, believed in me and encouraged me to pursue whatever I set my mind to. Words cannot express how thankful I am for her. I am also thankful for my sister, Shatha, for being a loving and encouraging sister, and for always listening to me and lifting my spirits when I am feeling down. A very special thank you goes for her understanding of my dislike of driving and taking it on her shoulders to drive me to/from university for the four years I was an undergraduate. I would also like to thank my brother, Mohammad, who by following his passion in engineering had inspired me to decide to become an engineer back when I was 6 or 7 years old.
Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified. Some of the material published in this thesis was published in the following papers:


(Heba Salem)
To my mother, Maryam
# Table of Contents

1 Introduction ........................................... 1  
   1.1 Motivation ....................................... 3  
   1.2 Aims and scope .................................... 7  
   1.3 Contributions and thesis outline ............... 9  

2 Background ............................................ 15  
   2.1 The Anatomy and security of computing systems .... 15  
   2.2 Hardware trust: definition and issues ............. 18  
   2.3 Why shouldn’t we trust our hardware systems? .... 20  
      2.3.1 Globalization in the supply chain and hardware Trojans .... 21  
      2.3.2 IP piracy ..................................... 25  
      2.3.3 Fault injection attacks ....................... 30  
      2.3.4 System-level hardware attacks ............... 33  
   2.4 Towards more trustworthy hardware ............... 38  
      2.4.1 Countermeasures to hardware Trojans ........ 38  
      2.4.2 Countermeasures to IP piracy ............... 46  
      2.4.3 Countermeasures to fault injection attacks .... 55  
      2.4.4 Countermeasures to system-level hardware attacks . 60  
   2.5 The last line of defence ......................... 63  

3 Run-time monitoring of untrustworthy on-chip communications ................. 69  
   3.1 Introduction ..................................... 69  
      3.1.1 Traits of a ‘last line of defence’ technique .... 71  
      3.1.2 Motivation .................................... 71  
      3.1.3 Contribution and chapter outline ............ 73  
   3.2 Related work ..................................... 73  
   3.3 Background and threat model ..................... 75
5.2.1 The attack vector .............................................. 142
5.2.2 Related work .................................................. 144

5.3 Approach ......................................................... 146
5.3.1 Temporal and mathematical analysis of ADC output .... 147

5.4 The framework .................................................. 151
5.4.1 Run-time integrity monitoring and incorrect output detection . 151
5.4.2 Trustworthiness assessment .................................. 154
5.4.3 Experimental setup .......................................... 156

5.5 Implementation ................................................ 158
5.6 Functional verification and evaluation .......................... 159
5.6.1 Use case 1: The electrocardiogram ......................... 159
5.6.2 Use case 2: The water treatment system .................. 162
5.6.3 Use case 3: The airplane fuel distribution system ...... 164

5.7 Evaluation ....................................................... 166
5.7.1 Limitations ................................................... 168

5.8 Conclusion ....................................................... 168

6 Conclusions ...................................................... 171
6.1 Lessons learned and future directions ......................... 176

A Supplementary material ........................................ 179
A.1 Algorithms ..................................................... 179
A.2 Data ........................................................... 192

Bibliography ........................................................ 193
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>The abstraction layers of computing systems, the associated attack difficulty and impact, and the attack mitigation difficulty</td>
<td>16</td>
</tr>
<tr>
<td>2.2</td>
<td>An example of a System on Chip (SoC)</td>
<td>17</td>
</tr>
<tr>
<td>2.3</td>
<td>The IC supply chain, the parties involved in it, and the IC resources available for HT attack facilitation</td>
<td>22</td>
</tr>
<tr>
<td>2.4</td>
<td>HT taxonomy [65]</td>
<td>26</td>
</tr>
<tr>
<td>2.5</td>
<td>Types of counterfeit ICs [85]</td>
<td>28</td>
</tr>
<tr>
<td>2.6</td>
<td>A taxonomy of HT countermeasures</td>
<td>39</td>
</tr>
<tr>
<td>2.7</td>
<td>A taxonomy of detective countermeasures to IC/IP piracy</td>
<td>47</td>
</tr>
<tr>
<td>2.8</td>
<td>A taxonomy of preventive countermeasures to IC/IP piracy</td>
<td>50</td>
</tr>
<tr>
<td>3.1</td>
<td>AXI3 write channels architecture (interconnect case) and signals</td>
<td>78</td>
</tr>
<tr>
<td>3.2</td>
<td>AXI3 read channels architecture (interconnect case) and signals</td>
<td>79</td>
</tr>
<tr>
<td>3.3</td>
<td>The deployment of the proposed scheme and wrappers in a SoC</td>
<td>85</td>
</tr>
<tr>
<td>3.4</td>
<td>W(AXI) transaction and data logs</td>
<td>87</td>
</tr>
<tr>
<td>3.5</td>
<td>The lifecycle of a transaction in the Address Read stage</td>
<td>88</td>
</tr>
<tr>
<td>3.6</td>
<td>The lifecycle of a transaction in the Data Read stage</td>
<td>89</td>
</tr>
<tr>
<td>3.7</td>
<td>Block diagram of the modelled SoC</td>
<td>93</td>
</tr>
<tr>
<td>4.1</td>
<td>The DRAM access latencies for the baseline and the DoS simulations</td>
<td>112</td>
</tr>
<tr>
<td>4.2</td>
<td>Heat map showing the range and distribution of latencies for different DRAM loadings</td>
<td>115</td>
</tr>
<tr>
<td>4.3</td>
<td>Training KDEs of RR memory accesses</td>
<td>118</td>
</tr>
<tr>
<td>4.4</td>
<td>Evaluation KDEs of RR memory accesses</td>
<td>118</td>
</tr>
<tr>
<td>4.5</td>
<td>Training KDEs of RW memory accesses</td>
<td>118</td>
</tr>
<tr>
<td>4.6</td>
<td>Evaluation KDEs of RW memory accesses</td>
<td>118</td>
</tr>
<tr>
<td>4.7</td>
<td>The J-S distances between the training (baseline) KDEs and the evaluation (run-time) KDEs</td>
<td>121</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>4.8</td>
<td>The proposed latency-extension detection framework and its implementation in real-life systems</td>
<td>124</td>
</tr>
<tr>
<td>4.9</td>
<td>The J-S distance between the training KDEs and the testing KDEs, in normal conditions and latency-extending DoS conditions</td>
<td>126</td>
</tr>
<tr>
<td>5.1</td>
<td>The control process in industrial control systems</td>
<td>136</td>
</tr>
<tr>
<td>5.2</td>
<td>Histograms of the temporal and arithmetic relationships calculated from the 80bpm ECG signal</td>
<td>150</td>
</tr>
<tr>
<td>5.3</td>
<td>Histograms of the temporal and arithmetic relationships calculated from the 80bpm ECG signal when bit 3 is stuck at logic ‘1’</td>
<td>150</td>
</tr>
<tr>
<td>5.4</td>
<td>The proposed analog front-end run-time integrity monitoring and trustworthiness assessment framework</td>
<td>152</td>
</tr>
<tr>
<td>5.5</td>
<td>Severity vs. flipping frequency vs. targeted bit</td>
<td>155</td>
</tr>
<tr>
<td>5.6</td>
<td>Severity vs. match-rate</td>
<td>156</td>
</tr>
<tr>
<td>5.7</td>
<td>Baseline and bit-flip (scenario 2) affected ECG signal</td>
<td>161</td>
</tr>
<tr>
<td>5.8</td>
<td>Histogram of the slopes of the bit-flipping affected (scenario 2) ECG signal</td>
<td>161</td>
</tr>
<tr>
<td>5.9</td>
<td>The baseline and bit-flip affected (scenario 1) LIT101</td>
<td>164</td>
</tr>
<tr>
<td>5.10</td>
<td>Slope histograms of the baseline and bit-flip affected (scenario 1) LIT101</td>
<td>164</td>
</tr>
<tr>
<td>5.11</td>
<td>The baseline and bit-flip affected (scenario 4) FTL</td>
<td>166</td>
</tr>
<tr>
<td>5.12</td>
<td>Slope histograms of the baseline and bit-flip affected (scenario 4) FTL</td>
<td>166</td>
</tr>
<tr>
<td>A.1</td>
<td>The ECG signal</td>
<td>192</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Targeted mistrust scenarios and countering security guarantees</td>
<td>80</td>
</tr>
<tr>
<td>3.2</td>
<td>The simulated unexpected or abnormal behaviours and the live verified responses of the wrappers</td>
<td>96</td>
</tr>
<tr>
<td>4.1</td>
<td>The implemented “DoS” modifications</td>
<td>112</td>
</tr>
<tr>
<td>4.2</td>
<td>The experimental setup</td>
<td>127</td>
</tr>
<tr>
<td>4.3</td>
<td>Number of transactions in the testing and evaluation datasets</td>
<td>127</td>
</tr>
<tr>
<td>5.1</td>
<td>The bit-flipping scenarios introduced to the ECG signal and the obtained match rates</td>
<td>160</td>
</tr>
<tr>
<td>5.2</td>
<td>The bit-flipping scenarios introduced to the SWAT level (LIT101) and flow (FIT301) sensor readings and the obtained match rates</td>
<td>163</td>
</tr>
<tr>
<td>5.3</td>
<td>The bit-flipping scenarios introduced to the AFDS FTL and CLF sensor readings and the obtained match rates</td>
<td>165</td>
</tr>
<tr>
<td>A.1</td>
<td>ECG signal variations</td>
<td>192</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

In as little as a couple of decades, electronic technology driven by advancements in the semiconductor industry has dominated our lives. In the very near past, concepts such as IoT, autonomous vehicles, robotic surgery, and even smart phones were the works of science fiction, whereas, nowadays, they are our daily reality. We are now immersed in electronic technology, it is around us, on us, and within us (i.e. implantable devices). The main enabler of this rapid development in the semiconductor industry is, from the technical point of view, the miniaturization of transistors that in turn enabled the encapsulation of millions of circuits in a single chip, leading to the integration of complex and advanced functionalities in small handheld or implantable devices. From the logistics and economy point of view, however, the wide spread and dominance of electronics was largely driven by the standardization in the IC (integrated circuits) design flow and the consequent globalization in its supply chain. This granted small IC design houses and technology companies from all over the world the ability to bring their technological vision into reality and manufacture their IC designs in the most advanced yet least financially draining way.

In spite of the apparently seamless and smooth advancement and spread of electronics, the backbone of this advancement, the semiconductor industry, is overshadowed by several challenges. The challenges range from the technical obstacles to further shrinking of transistors to the strain put on the IC supply chain due to increased demand, and from the geopolitics affecting the sourcing of the raw materials to the security and trust concerns surrounding the largely interconnected and globalized IC supply chain. Establishing trust in the semiconductor supply chain is one of the most pressing challenges as it is directly reflected in the security of the
produced ICs and electronic devices, which multiplies in importance the more electronics get integrated in our lives and societies. In our current day, we use electronics to safeguard our most confidential data and to automate and monitor our most critical systems, on both national and personal levels. This puts our very own lives and wellbeing at potentially unprecedented risk if the security of these electronics is ever compromised or if they are deemed unworthy of trust.

Historically, and due to the fact that access to the hardware (i.e. physical electronic components) of most devices was limited to parties directly involved in their design and manufacturing, it was assumed that hardware is inherently secure, a notion rooted in the now largely doubted concept of security by obscurity. As a result of this, academic researchers and technology companies alike focused on enforcing the security of the software running on those devices, given that it is more exposed to the outside world and hence more vulnerable to attacks from adversaries.

The convenience of ignoring the security of hardware was, however, short lived as hackers and researchers started showing that the obscurity of hardware is not a deterrent to hardware attacks. In a matter of a few years, several invasive (reverse engineering) [1-3], semi-invasive (probing) [4, 5], and non-invasive (side and covert channel attacks) [6-10] hardware attack techniques were developed and shown to effectively extract information about an IC’s design and operation leading to direct compromise of its security or paving the way for the introduction of malfunctions. Such attacks are initiated on the end-product of the IC supply chain, the manufactured chip/IC, and several design and run-time techniques were developed to counter them. For example, some of the early countermeasures (that are continuously improved) take the form of obfuscation and camouflaging techniques developed to make circuits indistinguishable to the reverse-engineering attacker [3, 11, 12]. Randomization and decorrelation techniques distorting the relation between the operation of the circuit and its leaked signals to deter side channel attacks [10, 13-15], and shielding and tamper sensors integrated in ICs to prevent probing and imaging techniques [16-19] are also proposed.

The introduction and integration of such countermeasures has indeed enhanced the security of ICs. Nevertheless, the threat to the security of ICs does not end at this point as ICs are also susceptible to attacks initiated in the design and manufacturing stages or what is referred to as the pre-silicon stage. Such attacks could originate from, among other sources, partners in the IC supply chain introducing malicious modifications (hardware Trojans) in ICs to serve a certain malicious intent, a
1.1 Motivation

In December 2019, news came of a novel virus previously unseen in the human population. The world did not pay much attention to it at the time and we continued our lives normally, until one day a couple of months later, its unprecedented and rapid spread forced us to follow isolation measures that none of us had experienced before. The direct consequences of the Coronavirus pandemic are undoubtedly related to human life, health, and wellness. Nevertheless, it is also undeniable that the financial, environmental, and social consequences of this long-lasting pandemic are rather extreme. Soon after the pandemic started, a semiconductor crisis in the form of a critical shortage in the supply of ICs was announced [20, 21]. This shortage was mainly driven by increased demand for consumer electronics amidst the isolation and work-from-home procedures adopted by most countries, and by lockdowns and regulations causing a serious disruption in the flow of components between different parties in the supply chain of ICs. This shortage led to an almost complete halt in the manufacturing of a number of products that are reliant on ICs such as automobiles, video game consoles, and handheld electronic devices, and consequently brought with it serious life disruptions and financial losses. At the time of writing this thesis, in the second half of 2022, this shortage has not yet been resolved. Reports show that many technology companies are resorting to desperate measures, from sourcing their ICs...
Chapter 1. Introduction

from grey markets to re-using lower grade ICs to shipping products with downgraded functionalities in order to curb the effects of this shortage, fulfill their orders, and maintain their position in the market [22–24]. Although these practices might provide short-term relief from the complications of the current semiconductor shortage, their long term impact on the security and trust of the end electronic devices is unknown and unpredictable. Additionally, the Covid-19 pandemic has revealed the extent of vulnerability and lack of resilience in the semiconductor industry, requiring a rethink of the structure of this industry in a timely manner to avoid serious technological disruptions in the event of future global (or even regional) events.

The importance of the semiconductor supply chain in today’s politics and economic landscape is further emphasized by the semiconductor tension between the US and China, that has been making headlines since 2017 [25, 26]. These tensions or what some refer to as “chip war” are arguably driven by the desire of each government to achieve technological supremacy. However, security and trust also play a central role, given the fact that one of the first moves by the US to regulate or ban the export of some technologies to China was justified by the possibility of China using these technologies in national security-threatening military applications [27].

As a consequence of these tensions, both the US and China announced plans to achieve semiconductor self-sufficiency and technological autonomy in the near future [28, 29]. Although these two countries may have the financial and human capital needed to achieve a high-level of semiconductor autonomy, a complete independence is not expected in the near future given the many challenges associated with localizing the design and manufacturing of ICs [30, 31]. Moreover, the road to semiconductor self-sufficiency for other nations around the world is faced by high barriers to entry in the semiconductor industry and low return for new entrants. Therefore, dependence on the highly interconnected global IC supply chain is the only option for the foreseeable future.

The Covid-19 pandemic and the US-China “chip war”, and their almost-simultaneous emergence, have put the semiconductor supply chain, an unknown and mostly out-of-sight process to the lay person, in the public spotlight. This has led the general public, for the first time, to realize how reliant we are on this supply chain, how globalized it is, how little control individual companies or countries have over it, and how even a temporary disruption in it could have a long-lasting impact on our economies, politics, and daily lives.

The crisis in the semiconductor supply and the US-China tensions are not the only
1.1. Motivation

concerns related to the semiconductor supply chain and its security. In 2021, The National Institute of Standards and Technology in the US (NIST) released a national initiative for improving cyber security in supply chains with a program for cyber security supply chain risk management (C-SCRM) at the core of this initiative \[32\]. The program focuses on managing and mitigating risks related to intentional (\textit{i.e.} counterfeit ICs and hardware Trojans) and unintentional (\textit{i.e.} environmental factors and poor quality) compromise to the security of the supply chain. NIST clearly states that “the factors that allow for low-cost, interoperability, rapid innovation, a variety of product features, and other benefits also increase the risk of a compromise to the supply chain, which may result in risks to the end user.” \[32\] emphasizing the seriousness of compromises and mistrust in the supply chain and indicating that the globalization in the IC supply chain, which has propelled massive advancements in electronic technology, is a double-edged sword.

In response to the unprecedented shortage in the supply of semiconductors and realizing the fact that this supply is linked to the security of many vital national sectors including national critical infrastructure, the business, environment and industrial strategy (BEIS) governmental committee in the UK issued a review on the situation of the semiconductor industry in the country. In this review, the issues and concerns surrounding the supply and industry of semiconductors are laid out and emphasis is put on the role of the government in ensuring the health and security of the semiconductor supply chain through, for example, the development of a national semiconductor strategy and through the support of research and innovation in the field \[33\].

The issue of trust and security in the semiconductor supply chain have also been recognized in academia, and researchers have been going to great lengths to study, analyze, and bring attention to a number of other concerns related to the heavy reliance on outsourcing and globalization in the IC industry. On top of these concerns is the extent to which technology companies trust their partners in the supply chain and the ICs that they contribute to producing. Carefully choosing and continuously auditing partners in the supply chain is a model followed in many industries to establish trust in the practices and operations of these partners and in ensuring the conformance of the end products to required specifications. However, with the prevalence of sub-contracting and the extent of specialization in the IC industry (where certain parts of the process could only be carried out by a single or few specialized companies in the whole world), following this model could become a
mammoth task that is not easily conducted on a regular basis. Evidently, semiconductor industry experts estimate that before an IC or computer chip reaches its final destination (i.e. the end customer or user), it could travel 25,000 miles around the world crossing international borders up to 70 times [34]. This realization prompts us to look at the concern of trust and security in the IC supply chain with fresh eyes and address it with a wider perspective, putting trust in the design and operation of the chip center stage rather than focusing solely on trust in the supply chain. This fresh perspective allows us to tackle mistrust in the IC supply chain, in the IC deployment environment, in the end user, and in the origin and licensing of the chip in case it is sourced in an encapsulated or fully fabricated form.

Mistrust or untrustworthiness in ICs is usually manifested in abnormal behaviour on either one or both of the operational and performance levels, and in unexpected, undesired, and potentially harmful actions and outputs. Several techniques were developed to tackle issues related to mistrust in ICs. The very first techniques introduced in literature focused on pre-deployment detection of untrustworthy behaviour by having ICs go through several tests to verify that their operation, performance and behaviour conform with expectations [35]. Later techniques resorted to enforcing design-level security and trust rules to prevent the exploitation of potential IC vulnerabilities [36]. Hardware security primitives such as Physically unclonable functions and true random number generators were also developed to fortify and establish trust in cryptographic circuits [37, 38], whereas hardware watermarking and stenography techniques were proposed to counter IP piracy and counterfeiting [39, 41]. Split fabrication and logic filling techniques were also introduced to prevent individual parties from having full visibility of the IC design [42, 43]. The question that remains after the development of all of these techniques and many others, however, is what if the attacker outsmarts the defender, and in spite of all of the pre-deployment fortification efforts, is still capable of introducing untrustworthy behaviour and security issues in ICs during their operation or at run-time? This is particularly possible with stealthily crafted and near-zero footprint hardware Trojans [44–47], environmentally induced fault injection attacks [48–50], and counterfeit ICs [51].

The realization that pre-deployment trust and security testing of ICs does not necessarily guarantee their trustworthy behaviour at run-time prompted researchers to the development of run-time monitoring and trustworthy computing on untrustworthy or untrusted components technique (TCUC) [52, 53]. Run-time monitoring
techniques focus on detecting anomalous and abnormal behavior or output at run-time whereas TCUC techniques go a step further by undertaking remedial and counter actions whenever such unexpected behaviour or output is detected. Both approaches are built on the notion that the electronic components and ICs in a particular system, although trustworthy at deployment, are not necessarily trustworthy throughout the lifetime of the system and hence continuous assessment and verification of their level of trust is necessary. The difficulty of guaranteeing life-long trustworthiness of ICs based on pre-deployment trustworthiness is related to the many physical, structural, operational, and environmental factors affecting the behaviour of ICs. Additionally, the current landscape of the semiconductor industry and the increasing dependence on electronics technology requires a greater amount of resilience engineering, which is defined as “the ability of systems to anticipate and adapt to the potential of surprise and failure” [54], in the process of designing ICs and electronic systems. The pressing need to bring these realizations to the design of real-life system-on-chips (SoCs) in an application-agnostic, lightweight, and adaptable manner form the main motivation of this thesis.

1.2 Aims and scope

Run-time monitoring and TCUC techniques are the last line of defence against trust-issues in ICs. Nevertheless, and in spite of the essential and unparalleled role of these techniques in run-time security and safety of electronic systems, research in this direction is quite limited and has not yet reached the high-level of maturity and universal-applicability of other types of IC security and trust measures. In the matter fact, run-time monitoring and TCUC techniques are yet to be developed (or honed) for fundamental and ubiquitous SoC components such as on-chip communication buses and interconnects, analog front ends, and even certain types of memory systems. This thesis, therefore, focuses on exploring and identifying the threat of untrustworthiness in those ubiquitous electronic components and consequently addresses the lack of run-time monitoring and TCUC techniques targeting them.

Conventionally, run-time monitoring and TCUC techniques were developed as countermeasures to hardware Trojans. However, in this thesis, we propose that by building techniques that revolve around the consequences and symptoms of run-time abnormal behaviour rather than its source and the particularities of its occurrence, we gain the ability to infer threats to the healthy and trustworthy operation of the system.
irrespective of their source; being internal, external, intentional, or accidental. This thesis, therefore, presents run-time monitoring and TCUC techniques that are, in principle, applicable to many types of circuit elements, yet are tested on ubiquitous elements that are deemed to be most vulnerable to them. In particular, the targeted mistrust scenarios are: the disrupting, shadowing, or diverting of the flow of transactions through an untrustworthy on-chip communication element (tested on AXI interconnect), the obscured prolonging of the processing time of transactions (tested on DRAM), and the introduction of new frequencies in the output of a signal transfer or conversion element (tested on analog front-ends). These scenarios and threat models were never addressed before, and independently, in run-time monitoring or TCUC techniques in spite of the serious security issues they might lead to and the fact that the targeted electronic components are fundamental and indispensable in most modern day SoCs.

In addition to targeting these mistrust issues for the first time, this thesis also aims to overcome the shortcomings associated with the limited-applicability and computational-complexity of conventional run-time monitoring and TCUC techniques. This is achieved by putting the concepts of adaptability, independence, autonomy, and computational lightweightness at the core of the proposed approaches and adhering to them throughout the development process:

- **Adaptability**: The ability to customize and adapt the developed techniques for different versions and implementations of the monitored element, and the ability to implement the proposed technique in SoCs and electronic systems irrespective of their size and application.

- **Independence**: Being independent of the architecture, design, and internal operational particularities of the monitored element.

- **Autonomy**: Being based on completely autonomous and self-reliant security elements and algorithms, that perform their monitoring and anomaly detection based on observing and analysing the external inputs, outputs, or actions of the monitored element without the need for any inputs from ‘inside’ the monitored element.

- **Computational lightweightness**: This is to be achieved through the adaptation of hybrid software-hardware implementation, lightweight hardware security
1.3 Contributions and thesis outline

The research presented in this thesis is concerned with facing the reality that the current IC design practices, the extent of globalization and counterfeiting in the industry, and the vulnerable physical nature of ICs, all contribute to the plausibility of SoCs and ICs exhibiting untrustworthy behaviour at run-time. By embracing and acknowledging this reality rather than turning a blind eye to it, the work in this thesis focuses on accepting that signs and symptoms of hardware mistrust or untrustworthy behaviour could appear any time during the operation of SoCs and ICs. Consequently, proper preparation is needed to avoid having such signs growing in impact and manifesting in potentially large-scale irreversible security issues. The core contribution of this thesis, therefore, lies in the development of three different schemes or frameworks for the run-time detection of different types of breach of trust or untrustworthy behaviour in SoCs and ICs. The run-time detection of the targeted untrustworthy behaviours, to the best of our knowledge, was never addressed before, especially, in the device and application-agnostic, adaptable, independent, and autonomous manner adopted in this thesis.

The first core contribution, presented in chapter 3 is a run-time monitoring and untrustworthy behaviour detection scheme that is based on the principles of component guarding, event verification, and data tagging. With experimental verification, using on-chip communication interconnects as the offending elements, we show that the implementation of these principles in hardware allows the detection of abnormal and unexpected behaviour that is out of the functional or operational boundaries of the monitored element. In the second core contribution, which is detailed in chapter 4, we highlight, with experimental evidence, the possibility of using the variability in the performance metrics of some electronic elements in masking untrustworthy behaviour. Consequently, a machine-learning based framework for the run-time detection of such behaviour is proposed and tested in detecting obscured latency extensions in on-chip memory systems. Chapter 5 presents the third core contribution of this thesis, which is a run-time machine-learning based framework for the detection of untrustworthy behaviour that is concerned with unnoticeable integrity degradation of intermediate SoC signals. A literature review
showing the possibility of the introduction of such behaviour is introduced, with particular focus on the susceptibility of the output of the analog part of SoCs (which is used in testing the framework) to such behaviour.

The targeted untrustworthy behaviours are chosen for three main reasons; the first is to show the breadth of the range of such behaviour and how it can manifest in many different ways; some of which are obvious violations of normal behaviour and some less so. The second reason is to show the extent of vulnerability to untrustworthy behaviour in conventional SoC and IC designs and emphasize on the fact that, up to this point, many SoCs and ICs are being designed and shipped without mechanisms to detect and counter such untrustworthy behaviour. The third reason is to shed light on the susceptibility of non-processing SoC elements to untrustworthy behaviour, an issue that has not gained sufficient attention in literature. The untrustworthy behaviour targeted by the first of the proposed schemes poses a direct violation to expected behaviour as in, for example, performing an “allowed” action at an unexpected point of time, by a fundamental yet non-processing part of the SoC. The second type of untrustworthy behaviour is obscure in nature and “hides” in the performance variability of some electronic elements. As for the third type of untrustworthy behaviour, it is of subtle low-profile nature that is hard to detect through simple observation of the offending on-chip element. All of the three types are not linked to a certain attack vector and could be the effect of a range of different hardware attacks or trust issues. By focusing on detecting untrustworthy behaviour at run-time, we therefore shift the focus from detecting the cause of mistrust to detecting its effect or early signs. This approach, considering the many different causes (i.e. trust issues) that could lead to the same effect (i.e. untrustworthy behaviour), streamlines and generalizes the run-time trust monitoring process and allows systems to focus on what should be a run-time priority; protecting SoCs and ICs from deviations in behaviour that may lead to serious security and safety consequences.

By focusing on run-time detection of untrustworthy behaviour, we aim to show that hardware trust and technological advancement can go hand in hand without the need to sacrifice on one for the sake of the other. The integration of run-time trust monitoring schemes (such as the ones proposed in this thesis) in SoCs is likely to reduce the amount of pre-deployment trust and security testing needed to the absolute necessary. This allows the satisfaction of time-to-market requirements and the maintenance of competitive market standing while ensuring the ability to detect and respond to unwanted behaviours if they ever arise during operation. Additionally,
through independent and autonomous run-time detection of untrustworthy behaviour, SoC integrators are allowed the opportunity of integrating outsourced state-of-art electronic components in their SoCs without being overly concerned with the potential trust and security consequences of this, sometimes inevitable, practice.

Although the proposed monitoring and detection schemes are primarily developed for trust and security purposes, they could also be utilized in performance monitoring or in long-term analysis of the impact of different environmental and physical factors on the behaviour of electronic components. The schemes also serve the additional benefit of run-time evaluation and validation of the effectiveness of design-for-trust techniques, and of pre-deployment hardware trust and security testing. For example, the run-time detection of trojan-related abnormal behaviour initiated from an on-chip element that was subject to pre-deployment trojan detection tests, indicates the failure of the implemented pre-deployment tests and prompts the need to improve them.

In addition to the technical contribution of this thesis, it also aims to present and discuss some of the gaps and shortcomings in the conventions and notions surrounding the concept of hardware trust. One such convention is the often interchangeable use of the terminologies “hardware trust” and “hardware security” in literature. Another convention is the linking of untrustworthy behaviour to globalization as its sole potential cause. The reasons why such conventions are prevalent in literature are likely historical, as concerns of lack of hardware trust started spreading mostly with the surfacing of the threat of hardware Trojans that are a consequence of globalization and that lead to security compromises. We discuss, however, that hardware trust should be viewed from a wider angle that is not necessarily tied to globalization in the IC supply chain, and from a perspective that is parallel to that used in viewing system or computing trust. Consequently, a definition is proposed for hardware trust along with a literature review of the many factors that may lead to trust issues or untrustworthy behaviour in hardware. With the introduction of device and application agnostic run-time monitoring and untrustworthy behaviour detection, we also aspire to pave the way for a fresh view on design-for-trust where generalization and universality are attempted as core goals, accommodating the large variation in SoC and IC designs, while also providing adaptability and customize-ability as feasible and available options, as per the specific needs of different systems.

Chapter 2 serves as the background for the remainder of this thesis by defining the concept of hardware trust and describing the issues surrounding it. It puts forward a
detailed explanation of the main causes of mistrust in hardware from globalization in the IC supply chain to IP piracy and from environmentally induced attacks to malicious software exploiting hardware trust and security vulnerabilities. The steps taken in published works towards achieving more trustworthy hardware are then presented and discussed, with particular focus on the last line of defence, run-time monitoring and TCUC techniques.

Chapter 3 presents the first core contribution of this thesis. The chapter starts with a survey of literature focused on run-time detection of anomalies in on-chip communication structures followed by a visualization of the scenarios of abnormal or untrustworthy behaviour that could be observed in a SoC constituting an untrustworthy communication interconnect and slave electronic components (i.e. peripherals) of mixed trust levels. The guiding principles of component guarding, event verification, and data tagging outlining the development of the proposed trustworthy computing on untrustworthy communication interconnects technique are explained, followed by the approach and architecture of the presented technique. This chapter describes the work published in:


In chapter 4, we propose the use of machine-learning models in characterizing the latency patterns of on-chip elements and detecting stealthy denial-of-service. The trust and security vulnerability posed by the inherent unpredictability in DRAM access latency is presented, and a hypothesis linking this perceived vulnerability with obscured latency-extending attacks is formed and tested using experimental simulations. To this end, the first run-time machine-learning technique to characterize and model the latency of DRAMs irrespective of its unpredictability is proposed as a viable approach to detect latency extending attacks. The underlining concept of DRAM transaction categorization is described, along with the approach and methodology developed to establish and verify its effectiveness. This chapter is concerned with the work presented in:

Following this, chapter 5 shows the susceptibility of analog and mixed signal circuit elements, and specifically, analog to digital converters (ADCs) to the introduction of malicious low-footprint bit-flipping. With this, we proceed to tackle the potential risk of injecting manipulated data into the SoC by an untrustworthy ADC and/or analog front end through the development of a machine-learning incorrectness-detection framework. The framework consists of an incorrectness detection phase and a trustworthiness assessment phase, and is capable of inferring incorrectness in the ADC output by solely analysing the temporal and arithmetic relations between its instances.

Chapter 6 concludes this thesis with a recap of the surveys presented, studies conducted, threat models targeted, and run-time monitoring and TCUC techniques developed. The chapter also presents a set of long-standing issues and challenges in the realm of hardware trust and discusses the lessons learned from the presented research.
Chapter 2

Background

This chapter provides the background needed to understand the issues surrounding the concept of hardware trust and the previous efforts made in enhancing our trust in hardware. We begin with the anatomy and layers of computing and hardware systems with a focus on the system on chip (SoC) layer, the heart of modern day electronic devices and the layer most affected by hardware trust issues. We then propose a definition for hardware trust, given the lack of a universal agreed-on definition. The main reasons behind mistrust in hardware are presented and a literature review on the measures developed to address lack of hardware trust is provided.

2.1 The Anatomy and security of computing systems

Modern day computing systems consist of multiple layers of abstraction as presented in Figure 2.1. From the upper layers of data and application, that are mainly concerned with application execution, data processing, and communication with other systems, to the middle operating system and firmware layers, that are concerned with the realization of the system’s functionalities through the utilization and facilitation of the bottom hardware layer. Evidently, the upper layers are more vulnerable to attacks due to their exposure to the outside world and their regular direct interaction with it. However, their existence at the top of the computing system allows the defender to address security or trust issues in a timely manner (i.e. security patches, updates) before they cause any serious damage to the healthy operation of the system or propagate to the lower layers. Therefore, it could be concluded that although the bottom most layer, the hardware layer, is the least susceptible to attacks due to its limited run-time visibility, it is, often for the same reason, the layer that is harder to
defend and the layer on which a single successful attack could propagate almost instantaneously to the upper layers leading to a complete system paralysis.

Figure 2.1: The abstraction layers of computing systems, the associated attack difficulty and impact, and the attack mitigation difficulty

The hardware itself consists of several layers starting from the system-level hardware that encapsulates all of the physical components of the system. Examples of system-level hardware include a smart phone, a gaming console, and a cyber physical system controlling a smart grid. Next comes the printed circuit board (PCB) layer, which has the main purpose of providing mechanical support and electrical connection to the elements providing the functionalities of the system, namely SoCs and microcontrollers. SoCs and microcontrollers consist of functional electronic elements that together form the computational and functional basis of any computing system. Examples of such elements are sensors, filters and amplifiers, analog to digital (and digital to analog) converters, digital and graphic processing units (processors), memory systems, and communication buses and interconnects, which perform, respectively, the tasks of signal acquisition, conditioning, transformation and conversion, processing, saving, and transfer. These functional electronic elements, are, in-turn, built of transistors, capacitors, diodes and other passive and active electronic components.

Figure 2.2 shows an example architecture of a modern day SoC. It consists of an analog part and a digital part. The analog part contains sensors, amplifiers, filters, and an analog to digital converter (ADC). The digital part of this SoC constitutes a CPU (central processing unit) and a DSP (digital signal processor), caches and memories
2.1. The Anatomy and security of computing systems

Figure 2.2: An example of a System on Chip (SoC)

with memory protection unit (MPU), input and output peripherals (USB and SPI),
encryption and decryption cores (Crypto) and a true random number generator
(TRNG) used for generating encryption keys, a circuitry of specific functionality to
this SoC application, and an on-chip communication fabric (i.e. communication
interconnects and buses). These SoC building elements are referred to as IPs
(intellectual properties), reflecting that the design is the intellectual property of the
design house, which could be the SoC integrator itself or an outside partner or
supplier. In the latter case, the IP is considered outsourced and is referred to as third
party IP (3PIP). The emergence of 3PIPs has introduced the concept of IP-based SoC
design, where technology companies (and SoC integrators) focus on the system-level
design and architecture of their SoCs while outsourcing most, if not all, of the SoC
building elements (IPs). Over the years, this has become a standard practice in the
semiconductor industry, especially in general purpose and consumer level electronic
devices, allowing technology companies of all sizes to catch up with the latest
technology trends while meeting stringent time-to-market constraints. This practice,
however, has also raised serious concerns regarding the trustworthiness and security
of the 3PIPs, and the subsequent trustworthiness and security of the SoCs they are
embedded in. The fact that 3PIPs are sold in a locked state or with limited visibility of

---

1Application-specific ICs (ASICs) are another type of ICs/SoCs that are customized to meet specific
functional specifications and performance metrics, and are mostly designed in-house. The design of
ASICs, however, is costly and resource-intensive. Hence, ASICs are often designed and owned by
certain companies and used in very specific applications.
the internal design and architecture, magnifies the security and trust issue further more.

Another concern regarding the security of SoCs is the relevant ease of access to the surface of the SoC by simple disassembly from the system-level hardware and desoldering from the relevant PCB, which opens the doors for probing and reverse-engineering attacks. An example of such attacks is the successful hardware mirroring performed on an iPhone 5c to bypass the allowed limit of pass-code retry attempts [4]. Due to their physical nature, SoCs also emit side signals that could reveal their internal state of operation and can be used by attackers in disclosing secret information (e.g. encryption keys) through what are known as side channel attacks, which proved to be successful even on modern “presumably secure” commercial processor SoCs [55]. The security and trust of SoCs could also be compromised when the fabrication and/or testing of their building IPs is outsourced and when IPs bought from the grey market are integrated in them. Given that SoCs and ICs are the functional hearts of electronic systems and devices, the terms hardware security and trust are often used to refer exclusively to the security and trust of SoCs and ICs (and their building IPs) instead of those of the upper layers of hardware; system-level and PCB. This, therefore, is the convention that shall be used in this thesis. Another common convention in literature is the interchangeable or non-clearly differentiated use of the terms hardware security and hardware trust. Nevertheless, based on the background established and studies conducted in this thesis, we are of the opinion that hardware trust is a broad concept that is not necessarily synonymous to hardware security. This is further explained in the next section.

2.2 Hardware trust: definition and issues

Although concepts such as design for hardware trust (DFTr), hardware root of trust, and trusted platform module (TPM), that stem from hardware trust, are well known, the concept of hardware trust itself is not well defined in literature. One attempt to differentiate between hardware security and hardware trust is presented in [56], which states that “Hardware security issues arise from its own vulnerability to attacks, as well as from lack of robust hardware support for software and system security”, whereas hardware trust issues “arise from involvement of untrusted entities in the lifecycle of a hardware”. Although this statement provides a reasonable differentiation between the terms of hardware security and hardware trust, framing hardware trust as an issue
arising solely from the globalization and outsourcing in the IC supply chain neglects the fact that hardware mistrust (in terms of deviation from normal operation) could also arise from other sources such as environment or software induced fault attacks and unintentional design faults. In order to come up with a comprehensive hardware trust definition, we apply the generic definition of trust introduced by the trusted computing group [57]: “the expectation that a device will behave in a particular manner for a specific purpose”, to hardware and extend it to cover hardware deviation from normal operation and performance, and the impact of hardware trust on security. Hence, the proposed hardware trust definition is, the expectation that a device will behave in a particular manner for a specific purpose, and will not perform actions that are out of the defined limits of its operation or that may lead to undermined system security. Lack of hardware trust, therefore, leads to, and could be identified by, a compromise in one or more of the following security properties [58]:

- **Dependability:**
  - Reliability: the ability to perform normally within the expected range of operation.
  - Availability: the ability to perform whenever expected and needed and within the expected time-frame.
  - Safety: the ability to prevent or avoid catastrophic failures that could lead to risks to the user or the environment.

- **Confidentiality**: the ability to guard against malicious obtainment or inferring of secret information through the observing of an output, an internal signal, or side and covert channels.

- **Integrity**: the ability to prevent the unauthorized and unexpected overwriting, modification, or destruction of data.

- **Isolation**: the ability to enforce rules of communication and interaction in systems with zones of various security levels.

- **Opacity**: the ability to perform computations and processing of data in invariant amount of time with respect to the input and/or the ability to obscure or de-link between patterns of inputs and their computation and processing time.
• Quantitative security properties: the ability to satisfy quantitative system-specific security properties such as the amount of randomness in the output of Crypto cores.

2.3 Why shouldn’t we trust our hardware systems?

The straightforward answer to the question posed in the title of this section is; because hardware is mostly not designed with trust in mind, assuming that it will always, under all (expected) circumstances, stay compliant and never cross its normal boundaries of operation. This, unfortunately, is a flawed notion that goes against the reality of the semiconductor industry and the very physical nature of electronics.

Since the early 2000s, light has been shed on the amount of globalization and outsourcing in the semiconductor industry and on the extent to which the industry is dependent on it. A new term, Hardware Trojan (HT), referring to the potential modifications introduced to ICs in their supply chain was introduced. This has, rightly so, led to a plethora of publications on the ways in which a HT could be injected and oriented in an IC, and on its potential malicious actions, in addition to the ways in which it could be detected, its impact could be mitigated, or its insertion could be prevented. The huge amount of research on the topic of HTs along with reports of suspected real-life HTs, moved the topic of HTs from the position of a research fantasy to that of a real and looming threat. Nevertheless, outsourcing and subcontracting in the IC supply chain and the consequent issue of HTs are not the sole threats to hardware trust. The relative ease of counterfeiting IPs and the lucrative business behind selling those counterfeit IPs, especially if offered and labeled as genuine ones, has also raised concerns regarding the possibility of the existence of counterfeit, low-quality, and untrustworthy IPs in the hardware of both common everyday electronic devices and safety-critical electronic systems. Additionally, the fact that the intentions of the end-user are, although assumed sincere, in reality unknown, poses another threat to the trustworthiness of hardware. An end-user (or administrator) with malicious intentions could abuse the physical properties of electronics and induce mistrust in hardware by putting it under adverse unexpected environmental or operational conditions. Attacks initiated from the software level that exploit hardware vulnerabilities and lead to abnormal behaviour in it such as the famous Spectre [59] and Meltdown attacks [60] and the recent Retbleed attack [61] also show the possibility of introducing mistrust in hardware without physically
2.3. Why shouldn’t we trust our hardware systems?

Accessing it. The main hardware mistrust-inducing factors are further explained in the remainder of this section.

2.3.1 Globalization in the supply chain and hardware Trojans

The significant rise in the complexity and consequently the cost of the design and manufacturing of ICs has led to a globalized IC supply chain, where a number of parties from all over the globe could be involved in the realization of a single IC. Outsourcing the IC fabrication to an external foundry (*i.e.* IC fabrication facility) has become commonplace in the industry, given the USD 10-20 billion needed to build a foundry, and is referred to as the fabless semiconductor model. Reliance on external parties is also common in other stages of the IC design process as it grants technology companies the ability to focus on core research and development activities while outsourcing the standardized ones. The rapid development of electronic technology, its use in multidisciplinary applications, and the continuous categorization and branching in it also lead to outsourcing being a need rather than a convenience. This is particularly the case for medium and small size technology companies that do not have the means to access top-notch cross-discipline technology without resorting to outsourcing.

Outsourcing, being a fundamental part of the current landscape of the semiconductor industry, brings to our attention unsolicited modifications that untrusted parties could introduce to ICs, or hardware Trojans. Hardware trojans (HTs) have been defined in literature as malicious, undesired, intentional modification of an integrated circuit that is introduced with the purpose of degrading the normal operation of the circuit, leaking secret information, or introducing a new malicious activity [62]. Figure 2.3 shows the main stages of the IC supply chain, the parties involved in each stage, and the IC resources available to them to realize their HTs.

The first step in the IC design process is the design specification, which is usually conducted independently with the in-house design team specifying the set of functional and performance requirements that the design should adhere to. This is followed by the translation of these specifications to an actual IC design, typically, using hardware description languages (HDL) such as Verilog and VHDL. The output of this stage is an RTL (register-transfer level) design that is synthesized to produce a netlist, which is a presentation of the design in the form of logic gates. The netlist then goes through a placement and routing process resulting in a physical layout file,
Figure 2.3: The IC supply chain, the parties involved in it, and the IC resources available for HT attack facilitation
that reflects the positioning of the different components (i.e. groups of logic gates, transistors, capacitors, etc) in the final chip. The physical layout file is referred to as GDSII, and it is the file used by the foundry to fabricate the IC. The design process from the RTL to the layout stage is conventionally conducted using electronic design automation (EDA) tools from external providers, thus the insertion of a backdoor or a HT by an EDA tool during the synthesis process is a plausible scenario [63, 64]. Additionally, rogue insiders in the technology company or IC design house itself or in a partner design house could insert malicious modifications in their own designs for sabotage or personal gain reasons [53, 65]. The design-level HT attack scenario that is most feared, however, is that of a HT-infected 3PIP, given its high chance of occurrence in real-life due to the significant reliance on 3PIPs in the industry and the fact that 3PIPs, whether soft (RTL) or hard (layout) are commonly treated as “trusted” block boxes that are directly integrated in the final designs of ICs or SoCs [53, 65].

The fabrication of ICs is commonly outsourced to specialized foundries that have full access to the IC design (through the GDSII file). Those foundries, could therefore easily alter not only the design of the circuit, but also the physical properties of the underlying silicon [66, 67]. Once the IC is fabricated, it goes through a testing and post-silicon validation process to verify that it is working to specification. So many technology companies outsource the testing process given that it is a systematic process. In early HT literature, testing facilities were assumed to have no potential malicious role in the IC supply chain, however, recent research showed the possibility of an untrusted testing facility breaking the camouflaging of the IC and extracting information that could potentially be used for future HT insertion [68]. Another potential test-stage attack is intentionally hiding or modifying the results of a HT detection test, in conspiracy with another untrusted party that inserted the HT in one of the design or fabrication stages [53, 69]. In the assembly, packaging, and distribution stages, the threat of HTs still stands, not through direct insertion, but rather through the packaging or distribution entity replacing HT-free ICs with HT-infected ones or with them reverse engineering the ICs for the purpose of piracy or future fabrication of similar ICs, but with HTs [53].

Ever since its first occurrence in literature in 2007 [70], the concept of HTs has always faced doubts regarding its feasibility (to the attacker or HT insertion party). The difficulty associated with crafting HTs, positioning them, and ensuring their activation at the right time could absolutely be a deterrent to successful real-life HT attacks, especially since, in some cases, similar malicious objectives could be
achieved with more mature types of attacks such as side-channel and fault injection attacks. Nevertheless, HTs, due to their nature of being embedded in the target IC, enjoy an unparalleled level of obscurity and freedom, allowing the realization of stealthy attack objectives that could never be achieved with other types of hardware attacks. Additionally, real life reports of HTs (or suspicions of them), although scarce and distant, are a clear indicator of the plausibility of a real-life HT attack. One of the earliest records of a suspected HT attack is related to the failure of the warning system in a reportedly state-of-art Syrian radar, back in 2007 [71]. A few years later, reports of alleged “kill switches” in European-manufactured microprocessors started surfacing. Those switches are arguably built into processors to totally disable them when desired by the attacker [71]. Reports claiming that spying microchips were embedded by malicious parties in the supply chain of the well-known and widely-used Supermicro servers emerged in 2018 [72]. As dozens of technology companies, including the technology giants, Apple and Amazon, use Supermicro servers, the authenticity and technical viability of the information presented in the report was heavily scrutinized and generally rejected [73]. Although those reports were never confirmed to be true, their negative impact on the market standing and reputation of Supermicro was undeniable with the company’s stocks dropping by more than 40% of their pre-report value on the day the report was published and not fully recovering, months later. Real-life reports of HTs or suspicions of them cast serious doubts on the opinion underestimating the threat of HTs. And in spite of the fact that reports of HTs are so few, we do not know if this reflects the reality of very few cases of real-life HTs, a result of the inability to detect all potential manifestations of HTs in electronic systems (given the complex nature of electronics), or a result of technology companies hiding and not publicly announcing HTs found in their devices in avoidance of consequent reputation and financial losses.

In the attempts to understand the risks posed by HTs and the dire consequences of their successful run-time activation, a large number of publications investigating the technical viability of HTs, their potential implementations, orientations, and activation mechanisms has appeared in literature. As a result of this, several researchers attempted to survey HTs present in literature and categorize them in taxonomies. The very first taxonomies differentiated between HTs based on the type of their trigger; the HT activation mechanism, and the type of the payload; the HT part responsible for executing the malicious action [74, 75]. Later taxonomies, however, expanded their analysis of HTs and included attributes such as the HT’s
physical features and location in their categorization [76]. The most comprehensive
HT taxonomy to date is the one presented in [65] and shown in Figure 2.4. In this
taxonomy, HTs are categorized based on the phase in the supply chain in which they
are inserted, their abstraction level, their activation mechanism, their location and
perhaps, most importantly, their effect in the circuit. The potential effects of
successful HT activation have been identified early in literature and they are usually
expected to be changing the functionality of the infected IC, changing its
specifications or affecting its performance, leaking (or assisting in leakage of) secret
information, or causing a denial of service. Simultaneously to HT taxonomy
development, benchmarks of HTs such as the Trusthub benchmark [65, 77] were
assembled to enhance our understanding of HTs and their effects, and assist in
building countermeasures to them. In spite of the fact that HTs and their
countermeasures are heavily researched, the risk of HTs is not yet fully mitigated and
our work against it is not yet finished. This is especially the case with the increasingly
distributed IC supply chain, the lack of extensive real life reports of HTs, and the
increasing complexity of electronics that, although may be useful in hindering HT
insertion, could also make the task of pre-deployment detection of HTs significantly
demanding.

It is also worth mentioning that backdoors inserted by the IC designer or SoC
integrator for genuine (e.g. debug or update) or malicious (e.g. spying) purposes are
sometimes considered hardware Trojans given that the end user does not know of their
existence. Nevertheless, this very specific type of backdoor is out of the scope of
this thesis as they are circuit modifications that are desired (by the SoC integrator or
technology company) and do not necessarily serve malicious intents and purposes.

2.3.2 IP piracy

There is no doubt that the semiconductor industry is one of the industries hit-hardest
by the Covid-19 pandemic. Reports published in 2022 indicate that the chip shortage,
caused by the Covid-19 pandemic and China-US trade tensions, had no clear end
in-sight, as had been the case since 2020 [78]. This seemingly extending shortage
placed technology companies in a dilemma of how to source the chips they need for
their electronic devices that the market had been eagerly waiting for. Desperate times
call for desperate measures, and one of the desperate measures that technology
companies resort to in such circumstances, and that security experts fear and warn
Chapter 2. Background

Figure 2.4: HT taxonomy [65]

from, is the integration of IPs or ICs bought from shady grey market suppliers in the light of the lack of genuine ones from authorized suppliers [23, 24]. The grey market of electronics offers all kinds of ICs, from recycled to overproduced and from cloned to out-of-specification, given that not all those types of ICs are produced, obtained, or sold in conceptually illegitimate ways; the market is referred to as grey instead of black. For example, in the case of a recycled or second-hand IC obtained from discarded electronics and sold on an e-commerce site such as Amazon or eBay, neither is the activity of selling second-hand electronics (if not being marketed as new) nor is the operation and activity of the distributor or supplier in the market (i.e. Amazon) considered illegal. Nevertheless, the fact that such ICs are sold without the explicit agreement and authorization of the original IP owner, through a non-authorized channel, and are potentially unfit for re-use (might have reached or are close to their end-of-life), they are still considered counterfeit grey market ICs that pose a threat to the security and trust of the electronic devices they are integrated in.

The issue of IP and IC counterfeiting has been identified in the industry since at least the early 2000s [79]. In 2011, the global value of the counterfeit chip market has been estimated to be around USD 75 billion [80, 81], a figure that is expected to be higher at the time of writing this thesis given that the current demand for electronic chips surpasses its legitimate supply.

In 2010, a report on counterfeit electronics and the extent of their spread was
published by the US department of commerce [82]. In this report, 42 electronic and microcircuit manufacturers (of the total 83 surveyed) stated encountering or becoming aware of counterfeit versions of their chips at least once in the period 2005-2008. The Electronics Resellers Association International (ERAI), a global organization concerned with monitoring, investigating, and reporting issues affecting the global electronics supply chain, has been analyzing and reporting the counterfeit electronics market since 2001 [83]. The very first report received by ERAI regarding a suspicion of counterfeit electronic components was in November 2001, where chips with wash marks and smears on their surface were distributed by a China-based distributor and arrived to purchasers in Samsung-branded tubes [83]. In the next couple of decades a large number of reports of counterfeit chips followed, many of which were destined for safety-critical or military applications. Examples are the uncovering of a California-based counterfeit IC distributor that provided ICs for military and navy contracts in the US, and the 2014 case of a US-residing China national selling military-grade, counterfeit, China-manufactured ICs as genuine US-manufactured ones [83]. The issue of IP counterfeiting is also strongly tied to the issue of a technology company mistrusting its partners in the IC supply chain, as evident from the lawsuit filed by the semiconductor giant, Xilinx inc, against the contract manufacturers Flextronics, where Xilinx accused Flextronics of reselling chips it brought from it and contributing in the counterfeiting and “grey marketing” of Xilinx chips [83]. In the aftermath of the many counterfeit IC incidents, governments moved to putting frameworks and regulations to detect, avoid, limit and mitigate the risks of counterfeit IPs, in addition to passing laws regarding the criminalization of involvement in counterfeiting electronics [83]. Nevertheless, and in spite of national and global efforts to combat IC counterfeiting, the trend of counterfeiting is continuously increasing and according to ERAI’s most recent statistics (2020) [84], 83.6% of the reports of counterfeit electronics it received in the year 2019 were never reported before, indicating a surge in the market and continued counterfeiting of new chips.

Counterfeiting of ICs takes many shapes and a counterfeit electronic component could be recycled, remarked, overproduced, defective, cloned, tampered, or provided with forged documentation [79] [85]. Irrespective of the type of counterfeiting, the intent behind it is usually financial gain and/or causing intentional damage to a certain system. Figure 2.5 shows the different types of counterfeits. A recycled IC is one that is obtained from a used device, removed from its original packaging, repackaged,
Chapter 2. Background

Figure 2.5: Types of counterfeit ICs

remarked and resold as new. Remarked chips are those that go under a process of grinding down old markings (showing country of origin, part number, etc), re-coating the chips to hide the grinding marks, and finally remarking them with new fake information reflecting for instance; a different country of origin or a higher grade (i.e. military grade for an originally commercial grade chip). Overproduction of ICs is mainly concerned with ICs produced outside of the contract limit between a foundry and an IC owner (i.e. design house or technology company), which poses a threat to the intellectual property of the IC owner and is specially expected when dealing with an untrustworthy foundry. A fundamental step in the manufacturing of ICs is their post-silicon validation testing, in which their functionality and performance are verified before they are packaged and distributed in the market. Due to normal variations in the manufacturing of ICs, some chips do not pass these tests and hence are rejected and deemed unsuitable for use. The selling of such defective and out of specification chips, either knowingly by the IC owner or by an external party that manages to get access to them, is another form of IC counterfeiting. A cloned IC is one that is produced from unlicensed and unauthorized copying of a certain IP or IC design. Cloning could be performed by an untrusted partner in the IC supply chain through, for example, illegal copying of the bitstream used to program field programmable gate arrays (i.e. a programmable type of ICs) or by external parties through reverse engineering chips and consequent extraction of their design. Tampered ICs are those that go through some intentional modification before reaching the end user, which could be in the form of IC-disabling hardware Trojan (commonly referred to as time bomb) inserted by one of the partners in the IC supply or a backdoor (e.g. for the purpose of spying on a certain end user or the SoC integrator ) inserted by the IC owners themselves. An IC sold with forged documentation is one where forgery is used to deceive the buyer about the compliance
of the IC to some standard, its passing of agreed-on tests, the changes (if any) it has gone through, or its origin and authenticity.

In addition to their clear violation of the intellectual property of the IC owner, the damage of counterfeit ICs goes beyond the sphere of business and finance to that of the security and safety of electronic devices and the humans who use them and interact with them. Given that counterfeit ICs are produced, distributed, or sold through unauthorized and potentially illegitimate means, they are expected to be defective, hence untrustworthy and potentially dangerous. This is particularly the case for recycled and out of specification ICs that constitute around 80% of the counterfeit IC market [86]. A detailed taxonomy of defects expected to be found in counterfeit ICs was introduced in [85] for the purpose of easing and facilitating the detection of counterfeit ICs. In the following, we present the main categories of this taxonomy in order to understand and assess the threat of counterfeit ICs on the security and trust of the systems they are deployed in. A counterfeit IC could suffer from exterior and/or interior physical defects [85]. The exterior defects are mainly concerned with the appearance of the IC and its non-conformity with conventional legitimately-obtained ICs, however, interior physical defects such as broken wires, die damage, and delamination (partial separation of IC layers) could introduce serious damage and failure, hence undermining the reliability, and consequently the hardware trust of the system. Electrical defects of counterfeit ICs are either parametric and hence related to the specification of the IC and its internal physical properties [85]. Parametric defects such as out-of-specification leakage current or incorrect temperature profiles could introduce to the system (in which a defective IC is integrated) unexpected vulnerabilities and trust issues such as the inability to operate in certain, otherwise normal, conditions. Additionally, parametric defects could also lead to rapid draining of system resources and could degrade its opacity making it more vulnerable to probing, fault injection, and side and covert channel attacks. Manufacturing defects are classified in three categories depending if they originate from the IC fabrication process, the materials used in the fabrication (i.e. impure silicon), or the packaging of the IC [85]. The existence of manufacturing defects in an IC introduces an unpredictability factor to its operation, given that manufacturing defects could be the cause of instantaneous or gradual failure of the affected IC, depending on factors such as the magnitude of the defect and its position. An unpredictable IC is therefore undependable and thus, untrustworthy.
Another issue related to counterfeit ICs is their life expectancy. A counterfeit IC coming from a used electronic device might not have any appearance defects and is expected to be internally non-defective (i.e. manufacturing and operational issues) given that it comes from a pre-used operational device. Nevertheless, this type of ICs suffer from an invisible issue related to their life expectancy and for how long more they can operate normally without reaching their end-of-life and causing serious performance degradation, or worse, complete unavailability or denial of service. Given the direct and obvious link between the defects of counterfeit ICs and their negative effect on the security properties of the system they are integrated in, one could conclude that a counterfeit IC is an untrustworthy one and that the sole existence of a counterfeit IC in a system deems it unworthy of trust. A similar conclusion is presented in the latest assessment of the critical supply chains supporting the U.S. information and communications technology industry, which states that the extended IC supply chain and threats from counterfeit parts “jeopardize the ability to provide secure devices and data” [87].

2.3.3 Fault injection attacks

Fault injection attacks are a well defined type of hardware attack that target the manufactured chip while in its operating environment, and aim to introduce faults to its operation and/or performance. This type of attacks originates from an untrustworthy end user of the chip (or the system or device it is embedded in) or an untrustworthy environment where some external malicious actor could gain close proximity to the targeted chip. Fault injection attacks are usually discussed in literature primarily as a hardware security issue, however, in this thesis, we view them also as weakeners of hardware trust as they are expected to cause deviation from the normal behaviour, actions, or performance of the victim chip at run-time. Although some types of fault injection attacks require physical access to the victim chip, many others are based on abusing the intrinsic physical characteristics of ICs through the introduction of temporary yet significant variations in their environment or inputs. This type of fault injection requires minimal direct physical intervention and often leaves no trace other than the abnormal behaviour induced in the chip, hence it is referred to as non-invasive fault injection. The two other types of fault injection attacks that require more direct interaction with the targeted chip are referred to as invasive and semi-invasive fault injection attacks.
2.3. Why shouldn’t we trust our hardware systems?

Fault injection attacks are hardware attacks that are concerned with using invasive, semi invasive, or non-invasive techniques to induce abnormal behaviour and actions in the targeted IC, cause a denial of service (DoS), bypass authentication and isolation rules, or facilitate the leakage of secret data (i.e. assist in side channel attacks) [88]. An example of an invasive fault injection is the use of focused ion beam (FIB) in direct modification of the design and internal connections of an IC. An FIB attack could be used to connect originally unconnected circuit nodes or to sever connections between originally connected ones. These modifications could lead to the introduction of permanent faults in the operation of the IC, the bypassing of certain authentication requirements, and perhaps more importantly, a paralysis in the fault detection mechanism in the circuit as FIB could be used to cut the connections between an on-chip fault detection circuit and the CPU [89]. Although a successful invasive fault injection attack would have dire hardware trust and security consequences due to its permanent and direct tampering of the silicon, this type of fault injection requires a great amount of skill and costly equipment, and is expected to leave visible tampering evidence (e.g. marks on IC packaging). The chance of it occurring in real-life is, therefore, low comparing to semi invasive and non invasive fault injection attacks.

Semi invasive fault injection attacks usually take the shape of optical faults, electromagnetic faults, and extreme temperature variations. Due to the fact that such attacks need access or close proximity to the silicon of the IC without physically tempering with it, they are considered semi or partially invasive. An electromagnetic (EM) fault injection attack uses a controlled and targeted EM field to induce behavioural and functional deviations in certain parts of the targeted IC. EM faults could take the shape of harmonic injection, characterised by continuous EM waves disrupting the healthy operation of analog circuits, or the shape of sudden EM pulses in the vicinity of digital ICs [90]. The equipment needed to introduce an EM fault injection attack to an IC is simple and readily available as a pulse generator connected to a magnetic antenna and directed at the target IC is sufficient for introducing processor faults in the form of changing one instruction to another, and memory faults in the form of modifying data retrieved from the affected memory systems [91]. Another type of semi invasive fault injection attack is an optical attack that was first introduced in [92]. Optical fault attacks use the illumination produced by laser or focused light beams in introducing a photoelectric effect in circuits and cause switching of transistors when they are not supposed to [93]. This sudden switching
Chapter 2. Background

could lead to abnormal circuit behaviour, bypassing of access rules, and targeted flipping of individual bits in, for example, on-chip memory. Although long identified in literature [94, 95], temperature-based fault injection attacks have gained increased popularity in recent years, mainly because of their ease of implementation. In temperature fault injection attacks, the ambient temperature of the victim IC is brought to extremes that are outwith the normal range of temperature for normal operation of the IC. This consequently leads to impaired switching and conducting ability in the IC leading to unpredictable behaviour, leakage of data, and potentially complete paralysis of the IC. Temperature fault injection attacks could be based on heating-induced faults [96, 97] or freezing-induced faults [49, 50].

Non-invasive fault injection attacks focus on introducing transient faults in circuits mainly through clock and voltage glitching. Clock glitching is characterized by the tampering of the external clock of the circuit by, for instance, shortening the length of a clock cycle or period through the insertion of additional rising clock edges. Given that the functionality of electronic circuits is dominated by strict timing and synchronization rules, the introduction of such glitches result in circuit timing violations and could lead to the propagation of wrong or repeated data in the circuit, the skipping of instructions in the processor, or the storing and retrieving of wrong data in the memory [88, 98]. In voltage glitching, the attacker introduces intentional variations, fluctuations, and deprivation in the power supply of the victim IC. In one of the first publications on voltage glitching, voltage spikes introduced to the power supply of a microcontroller performing RSA encryption led to the skipping of a subroutine call in the RSA computations [99]. Power deprivation or hunger faults (i.e. a type of voltage glitches) are based on pulling the IC feeding voltage below the nominal operating voltage in order to introduce timing errors and increase the propagation delay of signals [100]. In more recent voltage glitching attacks, a crowbar circuit was used to short the power supply and introduce voltage ringing (oscillation) to different types of ICs and consequently cause operational faults in them [101]. Non-invasive fault injection attacks are of particular trust and security concern due to their low footprint and the fact that by being non-invasive, they enjoy a higher chance of evading detection.

In addition to identifying the ways in which faults could be maliciously and intentionally introduced to ICs, researchers proposed taxonomies for categorizing the faults based on their effect on the IC [94, 95]. Faults injected into ICs could be of transient or permanent nature. Transient faults cause provisional and temporary
anomalous behaviour in the target IC and could often be remedied through resetting the circuit or closely monitoring and correcting any abnormal behaviour or action. Examples of transient faults are the skipping of a processor instruction, the fetching of a wrong value from on-chip memory, or the modification of a counter value. On the contrary, permanent faults cause significant irreversible damage to the victim IC and are irreparable. Fault attacks could also be specific to single bits or whole bytes, introduce random or specific modifications (e.g. random flipping of all bits versus flipping of specific bits), and target static (e.g. memory cells) or computational (e.g. processor computations) operations in the circuit. Fault injection attacks could also target either the control or the data signals in the victim IC. Given that control signals are fewer than data signals, injecting faults in them might be non-trivial, however, successfully doing so would result in more powerful malicious consequences.

2.3.4 System-level hardware attacks

Hardware Trojans, IP counterfeiting, and fault injection attacks are hardware-mistrust causes that are primarily realized through direct interaction with hardware and/or physical intervention in its supply chain, architecture, manufacturing, or operation. Nevertheless, research shows that hardware trust could also be weakened by attacks initiated from upper layers in a computing system in which vulnerable hardware is deployed. In this thesis we refer to this type of attack as system-level hardware attack, which is characterised by maliciously misusing or abusing the interaction between the hardware layer and the upper layers in computing systems in the exploitation of hardware vulnerabilities, whether inherent physical vulnerabilities or operational ones. The interaction between the hardware layer and the upper firmware and software layers is the basis of operation of all computing and electronic systems, and using it in inducing abnormal behaviour in hardware could be realized in a multitude of ways in proportion with the great diversity and variation of IC and SoC designs and the software interacting with them. In this thesis, for the purpose of generality, light is shed on the main types of system-level hardware attacks that are concerned with common, generic, and universal system hardware and architecture. Such types of system-level hardware attacks could be classified based on the targeted hardware process or circuit as DRAM attacks, cache attacks, and speculative execution attacks.

Dynamic random access memories (DRAMs) are staple components in the vast majority of modern day computing and electronic systems, mainly because of their
simple design and low cost. A DRAM memory is made up of memory cells where each cell constitutes a capacitor and a transistor. The capacitor is charged or discharged through the transistor in order to store a logic “0” or “1” depending on the value of the bit that needs to be stored. As the electrical charge stored in capacitors leak over time, DRAMs need regular refreshing, thus the terminology “Dynamic”. Due to this simple architecture, DRAMs allow high-capacity storage and hence are heavily used as the primary on-chip memory for processors. The essential role that DRAMs play in most computing and electronic systems led researchers to investigate the robustness of the charging and discharging process and if it could be exploited for malicious purposes. The cold boot DRAM attack introduced in [102] challenged the common assumption that the contents (i.e. charge) stored in DRAMs disappear almost instantaneously after disconnection from power, by demonstrating that in normal operating temperatures, total DRAM decay (i.e. loss of stored charge or data) could take up to 35 seconds, while in extremely reduced temperatures, it could take up to several hours or days. With this realization, the authors of [102] proposed multiple types of attacks where attackers of varying capabilities could gain access to retained data in DRAMs. The most basic form of these attacks require no physical access to the DRAM chip and could be carried out by rebooting (restarting the operating system or briefly cutting and restoring power) the target device and launching a small kernel to access the targeted data. Such attack, if initiated on any device, would considerably undermine its confidentiality and hence its hardware trust, as a system not capable of completely wiping confidential data when required is an untrustworthy system.

The Rowhammer attack is another DRAM attack that is based on the exploitation of an inherent physical vulnerability of DRAMs. Known physical phenomena in high density DRAMs are the electromagnetic coupling and cross-talk between adjacent or proximate cells. These phenomena, if not mitigated by isolation and screening techniques, could lead to undesirable interference between DRAM cells, that could in turn lead to leakage errors and malfunctions. Rowhammer attacks, first introduced in 2014, are based on the exploitation of this physical vulnerability, through magnifying its effect so that errors are induced even with the existence of basic cross-talk and electromagnetic interference mitigation techniques [103]. Rowhammer attacks are initiated from the software level and are based on a malicious software locating its data in DRAM cells adjacent (or within close proximity) to cells storing sensitive data or data that the attacker is interested in destroying. By repeatedly and rapidly accessing the memory cells in which the attacker’s data reside, a leakage of electrical
2.3. Why shouldn’t we trust our hardware systems?

In addition to the direct impact that the bit flipping induced by Rowhammer attacks has on the integrity of DRAMs, research shows that Rowhammer attacks could also be used in privilege escalation allowing the attacker to access memory locations they are otherwise prohibited from accessing [104, 105], and in deducing contents of DRAM cells that the attacker does not have access to [106]. Therefore, Rowhammer attacks do not only undermine DRAM integrity, but also its confidentiality and isolation.

Caches are temporary on-chip storage elements that are used for storing data and instructions most frequently used by the processor. The main purpose of the integration of caches is the efficiency achieved when avoiding wasting time fetching frequently needed data and instructions from the main on-chip memory. Many computing and electronic systems deploy multiple cache levels to cater for the different frequency of access of different data. In a multi-level cache architecture, the cache closest to the processor (and faster to access) would be the smallest in size whereas the one farthest from the process would have greater capacity but greater access time. Such an architecture allows the hierarchical storage of data based on the frequency of access needed by the processor [107]. The replacement of data and instructions and their transfer from one cache level to another (and from the main memory) is governed by defined policies. The fact that caches are shared resources that are used by all type of software running on the processor makes them vulnerable to attacks exploiting any traces left by previous sensitive or high-privilege use of the cache [58, 107]. Such cache attacks could be time-driven or access-driven, and an expected target are cryptographic circuits [108]. Both types of attacks are based on the process (i.e. code) of the attacker sharing a cache with a victim process that the attacker is interested in learning about through analyzing its interaction with the cache. The victim process is often a cryptographic operation. In time-driven attacks, information about an encryption key could be inferred by measuring the execution time that varies with the cache hits (i.e. when data requested is found in cache) and misses (i.e. when data requested is not found in cache) of the cryptographic process. In access-driven attacks, the attacker monitors cache components accessed by the victim process and infer, also from cache hits and misses, if a certain sub-process is being carried out [108]. For example, the S-box or look up table entries saved in the cache and accessed by an AES encryption process could be identified from the cache access patterns, and informed deductions could be made about the encryption key [58]. Cross-core attacks on caches where the attacker and victim process need not
be running on the same core were also introduced [109, 110].

The increasing demand for high performance computing and the race to include as many features as possible in computer and electronic systems has led to a continuous search for ways to enhance and optimize the architecture, performance, and efficiency of hardware. The use of caches is an example of such enhancements, and another example is out-of-order execution, in which processors avoid becoming idle by executing instructions based on the availability of input data and on-chip resources instead of the conventional in-order execution of instructions. Speculative execution is a form of out-of-order execution in which the processor speculates a need for some task and performs this task before it is prompted to, in order to avoid delays when this task or input is actually needed. Out-of-order and speculative execution were developed for computing efficiency and processor optimisation purposes, hence, as is common in the world of chip design, their trust and security were not heavily considered and scrutinized before their integration in computing systems. Researchers mostly started researching the vulnerabilities of speculative execution after the surfacing of the now-famous Spectre and Meltdown attacks in 2018 [59, 60]. Memory isolation, in which software programs and processes are assigned different parts of the memory, allows programs to run in parallel without interference and isolates privileged or confidential data from non-privileged data. Meltdown is based on overcoming this memory isolation through the exploitation of the out-of-order execution mechanism. In Meltdown, the attacker is an unprivileged user running an unprivileged code and has no physical access to the target machine. Through their code, the attacker prompts the memory to load data from some privileged memory address or location. As the attacker is not an authorized user of this memory address, a privilege check denies them access, however, at this point, and because of out-of-order execution, the data would have already been “temporarily” loaded to the cache. The attacker then uses a cache side channel to leak this secret information out of the chip, and by repeating this process they can leak out the entirety of the memory [60]. Branch prediction is one form of speculative execution in which a circuit, known as branch predictor, is used in guessing the branch (e.g. in an if-else statement) that will be executed next. In Spectre attacks, the attacker runs a code that prompts incorrect branch prediction, tricking the processor to speculatively execute a sequence of instructions that should not be executed under the current process. Although, the processor eventually realizes its incorrect decision and reverts the results of the incorrect branch prediction, traces of this prediction remain in the cache.
and hence are leaked out by the attacker through a cache side channel [59]. Spectre and Meltdown opened the eyes of researchers and technology developers to the trust issues posed by processor performance optimizations and they were the first in a series of similar attacks exposing the vulnerabilities of such optimizations [61, 111, 112].

In addition to system-level attacks on DRAMs, caches, and performance optimization measures, it was shown in [113] that the interaction of hardware with software or firmware could be abused to exploit apparently harmless RTL coding errors in instantiating hardware security bugs and triggering them at run-time. Such errors include incorrect connection or assignment of a certain variable.

Section 2.3 provided a literature review on the main hardware trust threats or undermining factors, namely, globalization in the IC supply chain and hardware Trojans, IP piracy and counterfeiting, fault injection attacks, and finally system-level hardware attacks. It is essential to realize, however, that although the most probable and proven, these are not the sole weakeners of hardware trust. By referring to the proposed definition of hardware trust, it can be deduced that unexpected deviations in the behaviour and performance of hardware could be caused by unintentional design faults and human errors as much as they could be caused by, for example, hardware Trojans. Additionally, the integration of open-source hardware is rapidly turning into a staple practice in the industry. This practice might seem advantageous to the trust and security of hardware as it eases and mainstreams the process of finding and reporting security vulnerabilities. However, it also brings with it the concerns of an “almost” fully transparent hardware and where hundreds, if not thousands, of parties and individuals contributed in developing, in which case, verifying and validating the trust and security of each of the contributions might be a challenging, if not an impossible task. Moreover, the continuous discovering of new types of hardware or system-level attacks is a clear indicator that hardware mistrust is influenced by a large number of factors; physical, operational, and environmental, and that new threats to hardware trust, that would thwart our current security and trust measures, might emerge any time in the future. The next section presents a literature review on the measures developed to counter each of the hardware trust issues presented in this section in addition to providing a survey of generic run-time hardware trust enforcing techniques that are not tied to a certain type of attack.
2.4 Towards more trustworthy hardware

In the previous section, the main reasons why hardware shouldn’t be blindly trusted were presented and explained. In this section, therefore, a literature review of the efforts made in the attempt to improve the trust of hardware is provided. Some of these efforts are specific to certain hardware trust issues while some others are more generic.

2.4.1 Countermeasures to hardware Trojans

Hardware Trojans are viewed as the malicious by-product of the globalization in the IC supply chain. It is therefore easy to conclude that to eliminate the issue of hardware Trojans, limits have to be put to this globalization and outsourcing. This is a solution that is generally of low feasibility as electronic technology has already reached a state of globalization-enabled development where it is almost impossible to go back to the pre-globalization days without halting or slowing down the wheel of technological development. The root cause (i.e. the existence of untrustworthy parties in the globalized IC supply chain) of hardware Trojans cannot be tackled easily without national and international coordination, agreements, and legislation aimed at enhancing the security of the globalized semiconductor supply chain. Consequently, the development of measures to prevent the insertion of HTs, detect their existence if attackers manage to insert them, and identify their actions at run-time is of utmost importance. Researchers have realized the importance of developing countermeasures to HTs ever since the first reports and publications of HTs started surfacing. A few years later, taxonomies of HT countermeasures were introduced in a number of publications, with particular focus on HT detection techniques as they were the first type of developed HT countermeasures [75, 114]. As the range of HT countermeasures started expanding with the introduction of HT insertion prevention and design for trust techniques, more inclusive taxonomies were presented [52, 115].

In addition to the typical categorization of HT countermeasures based on the mechanism and approach adopted to thwart HT attacks, some taxonomies also classified them based on the type of HTs they are capable of detecting [116] and based on the difficulty of evading them by the attacker or HT insertion party [53]. Figure 2.6 shows the taxonomy of HT countermeasures adopted in this thesis and that is largely based on the one presented in [52] with the addition of the machine learning based HT detection sub-category to reflect more recent advances in the field.

The first category and the most researched one is concerned with HT detection
either in the pre-silicon or post-silicon stages. Given the wide range of parameters and variables associated with HTs and their insertion circumstances, guaranteeing that a given IC is free of HTs, even if it was designed with preventive HT measures, is a non-trivial task. Therefore, putting ICs under HT detection tests to verify their clearance (up to the capabilities of available detection techniques) of HTs is a top priority in the fight against HTs, justifying hence the ever expanding research on HT detection techniques.

**HT detection: Post-silicon.** Destructive HT detection tests are based on reverse engineering the chip and imaging its individual layers to validate that the design was not tampered with [[17]]. By its nature, reverse engineering is an invasive and destructive process that leaves the chip unusable afterwards, and hence is only suitable for validating a small number of chips from a given batch. As HTs are not necessarily inserted in all chips of a given batch, this selective testing could lead to a high rate of false negatives. As reverse engineering is a resource demanding and time consuming process, it is often preferred to be used only in fields where a physical proof of clearance from HTs and tampering is needed, such as in military applications. Researchers also proposed the use of reverse engineering in obtaining the characteristics of a few ICs and using them in training machine learning models on distinguishing between normal and suspicious structures in ICs, hence, aiding in non-destructive HT detection [[18], [19]].

Non-destructive post-silicon HT detection techniques mainly fall into two
categories; functional testing and side channel analysis. As the name indicates, side channel analysis is based on analysing the side effects or parameters of IC operations, that is, power consumption, time variations, electromagnetic emissions, and thermal characteristics. Such analysis is performed to identify any deviation from the expectations, which in this case, is considered to be because of a HT. Early publications on this topic assume the existence of a “golden” HT-free IC that could be used as a reference for the normal side parameters of a given IC. As this is a challenging requirement, researchers proposed a number of ways to eliminate it from the side-channel HT detection process [120, 121]. Another identified issue is that the introduction of HTs does not necessarily cause detectable variation in side-channel signals as shown in [122], where the effect of a HT on the power consumption of a given IC was shown to be less than 1%. Some publications attempted to overcome this obstacle by segmenting ICs in order to localize the effect of the HT (if any) on the IC, and augment its influence on the IC’s side channel [123, 124].

Functional testing is another type of non-destructive HT detection that is mainly concerned with functionality-changing HTs. In functional testing, test patterns are applied to the chip of interest and its response is observed. If responses to all test patterns are identical to the expected responses, then the chip is declared HT-free. A major issue with functionality testing is that the probability of detecting HTs is tied to the number and variation in the applied test patterns and given the billions of nodes in modern-day ICs, the HT nodes might not be activated even after applying a large number of test patterns. Researchers tried to tackle this issue by identifying the nodes that are likely to be targeted by HT insertion (i.e. rare probability nodes that have low probability of activation in normal circuit operation) and deriving test patterns to activate them and observe the response of the IC [125]. As this approach needs iterative testing and hence is time consuming, techniques to accelerate the activation of rare-node HTs were also proposed [126]. For more rigorous testing, functional testing could also be combined with side-channel analysis [127]. Optical and thermal imaging of ICs are another post-silicon non-destructive HT detection techniques [128].

HT detection: Pre-silicon. Functional validation could be viewed as the pre-silicon simulation-based equivalent of functional testing, hence many of the functional validation techniques share the same advantages and disadvantages of the functional verification or testing ones. Early examples of techniques developed specifically for pre-silicon functional validation are Veritrust [129] and FANCI [130], which are based on testing circuit nodes that do not have a direct link to the circuit
output, as it is assumed that they are likely to be targeted by stealthy HTs.

Formal verification is an algorithmic test, mostly applied on the RTL level, and used to verify the functional properties of a given IC design. Property checking is one form of formal verification that has been used in pre-silicon HT detection. In property checking, functionalities of the IC design are translated into properties, which is then followed by the execution of code and functional coverage analysis to identify the parts of the circuit that do not correspond to known properties, these parts are then considered suspicious and potential HT parts [127, 131]. Formal verification methods could be followed by automatic test pattern generation (ATPG) aiming at activating the circuit parts that were identified as suspicious and observing the circuit response [114]. Another form of formal verification is based on the concepts of proof carrying code and security property assertion, in which agreed-on security properties (i.e. between the 3PIP vendor and the SoC integrator) are verified by the SoC integrator before integrating the 3PIP in the SoC [132, 133].

Feature analysis techniques are based on using knowledge obtained from HT benchmarks and latest research on HT designs in identifying potential characteristics of HTs. For instance, most HTs are expected to utilize circuit nodes with low-switching probability, additionally, time triggered HTs are known to use large counters whereas sequential HTs (i.e. activated by a sequence of certain events) are expected to be connected to state machines. Such characteristics are identified as potential HT features and are “searched for” in the circuit of interest [134, 135]. The main issue with feature analysis is its dependency on pre-defined HTs, deeming it unsuitable for detecting new and custom HTs as their features are not seen before by the feature analysis tool.

HT detection: Machine-learning based. In addition to the conventional HT detection techniques, a number of publications proposed using machine-learning in enhancing and improving HT detection.

One of the first publications employing machine-learning in HT detection is [136], in which the difficulty of having changes in a given signal penetrating to the circuit’s output, and the difficulty of setting a given signal to logic “0” or “1” are used as metrics to cluster the netlists in a given IC and infer the most suspicious ones. Support vector machines [137], random forest classifiers [138], multi-layer neural networks [139], and deep learning [140] were also proposed to improve feature analysis based HT detection and widen its scope and applicability. In the quest to craft HT detection techniques that are capable of detecting novel non-previously known types of HTs, the authors in [141]
and [142] used classification algorithms combined with feature, functional, and power analysis in distinguishing the power signature of genuine HT-free ICs from that of HT-infected ones. In addition to using machine learning techniques in pre-silicon detection of HTs, specifically those in 3PIPs [143, 144], machine learning was also applied in improving the post-silicon HT detection techniques of reverse engineering [118] and side-channel analysis [145]. In [118], a support vector machine (SVM) approach is used in replacing the time consuming net-list extraction (annotation and schematic creation) part of the reverse engineering process. Applying machine learning along with image processing in extracting features of ICs after scanning them was proposed in [146]. The authors of [69] took machine-learning based HT detection further by applying it in detecting the modifications performed by an untrustworthy IC testing party on the results of HT detection tests.

Design-for-trust is a concept adapted from the well-known concept of design-for-test [147], in which circuits are built with structures to facilitate and ease their post-silicon functional validation and testing. Design-for-trust is, therefore, concerned with designing circuits with built-in trust and security features. Conceptually, design-for-trust is a broad term that applies to trust-aware IC design that is capable of thwarting, preventing, or monitoring for all types of hardware attacks and trust issues. Nevertheless, in HT literature, design-for-trust is often viewed exclusively from the perspective of countering HT attacks.

**Design-for-trust: HT-detection facilitation.** Elements aiding in the post-silicon detection of HTs could be added to the design of ICs. Dummy elements such as flip flops are placed in the path of rarely activated nodes to accelerate their activation in post-silicon functional testing [148] whereas scan cells (originally used in design-for-test) are implemented for the purpose of localizing power consumption in the IC and hence aiding in power side-channel based HT detection [149]. Ring oscillators are used to assign unique “frequency” identities to ICs, that would be altered if a HT is found in HT detection tests [150], and also used as power monitors to detect abnormal power fluctuations in HT-infected ICs. The use of current sensors monitoring the current drawn from each block of the circuit [151] and thermal sensors monitoring the fluctuations in the IC’s temperature were also proposed to facilitate HT detection [152].

The multitude of factors affecting HT insertion and activation and our limited-knowledge of real-life HTs dictates the importance of monitoring for suspicious HT behaviour at run-time to assure hardware trust. Early approaches to
run-time monitoring focused on monitoring and detecting deviations in the power and thermal profile of a given IC using the same structures used in facilitating post-silicon HT detection; sensors and ring oscillators [153, 154]. In [155], on-chip sensors monitoring the electromagnetic radiation of ICs at run-time were also introduced. Researchers also proposed the use of performance counters in capturing micro architectural activities in microprocessors [156] and hardware property checkers in checking the behavioural properties of ICs at run-time [157].

Design-for-trust: Prevention of HT insertion. Prevention of HT insertion is realized primarily by adding a layer of obscurity or ambiguity to IC design or through analyzing and addressing IC vulnerabilities. Hardware obfuscation is a technique borrowed from the world of software security, where source code obfuscation is used to discourage unauthorized tampering. In the context of hardware, obfuscation is performed through the implementation of techniques that make the circuit functionalities and properties hidden or indistinguishable, deterring attackers from inserting HTs due to their confusion and inability to distinguish the different parts of the IC [158]. Obfuscation in the context of HTs, could be applied on the device (transistor) level, the logic level, the circuit level, and the system level. Device level obfuscation is concerned with the level of transistors and the connections between them. One method of device level obfuscation is the introduction of stuck-at-faults and delay-faults through adjusting the doping concentrations of transistors or through varying the thickness of vias between the IC layers [158]. The purpose behind the introduction of such faults is to obfuscate the short circuits and connections in the IC. The intentional introduction of cross-talk was also proposed as a way to conceal the circuit’s true functionality [11].

Logic level obfuscation is based on the utilization of device level obfuscation in creating logic gates of a behaviour that does not conform with their appearance, or in creating logic structures that are indistinguishable to potential adversaries. Obfuscated transistors could be used in creating flip-flops of normal appearance but that actually have some of their inputs stuck at a certain logic level, a technique useful in hiding certain inputs [158]. Obfuscated transistors and contacts could also be used in building logic gates based on a mix between real and fake (i.e. obfuscated) elements, hence hiding their actual functionality from adversaries [158, 159]. Circuit level obfuscation is concerned with obfuscation on the physical layout level, primarily, through camouflaging and logic cells. Camouflaging is physically hiding the architecture of the circuit through, for instance, substituting the layout of a given
circuit cell with the layout of a visually different but functionally identical circuit [160]. The use of programmable cells that are programmed after fabrication is another form of camouflaging [11]. The other type of circuit level obfuscation is the use of filler cells, which are logic cells, that could be functional [161] or non-functional [160], and used for filling empty spaces in an IC.

On the system or whole IC level, several obfuscation techniques were developed ranging from combining multiple device, logic, and circuit level techniques in crafting whole ICs with an appearance that does not conform with their functionality, to the use of programmable logic and logic encryption. The work in [162] proposes the replacement of logic gates with physical unclonable function (PUF) and programmable logic. Logic locking is another obfuscation technique that gained considerable attention due to its applicability not only in deterring HT insertion but also in preventing IP piracy [163]. In logic locking, additional logic gates are added to a given IC to act as a lock that if supplied with the wrong key would prompt the circuit to produce wrong outputs. Logic locking breaks the correlation between the inputs and outputs of circuits and prevents adversaries from deducing a circuit’s functionality from its input-output relation.

In a relatively new direction, researchers started focusing on identifying and addressing IC design vulnerabilities that allow the insertion of HTs in the first place. The framework introduced in [164] assigns each identified vulnerability a metric to quantify its extent in a given IC, and a rule, specifying the maximum allowed amount of this vulnerability for the IC to be considered secure. Examples of metrics adopted in this framework and that are applicable to the RTL level are the observability of signals (i.e. how much a change in the signal affects the output) and the hardness of statements (i.e. the rarity of their execution) [165]. Based on a certain threshold or rule, circuits with low observability signals and hard statements are considered vulnerable to HT insertion. Another identified RTL vulnerability is the existence of don’t care (x) statements in the code as they could be reassigned by the adversary [166]. A Don’t care statement also poses a vulnerability in the netlist level, given that in the synthesis process, the EDA tool, for some optimization purpose, could connect it to circuit assets that should normally be protected and inaccessible by this particular statement [164] [167]. Another netlist vulnerability could also arise from design-for-test and design-for-debug features. As those features could not be eliminated from ICs, they are considered an inherent vulnerability and their effect on the IC security should be analyzed and mitigated [164]. As the layout level resembles
2.4. Towards more trustworthy hardware

the physical chip, its vulnerabilities were analyzed with the aim of assessing how vulnerable is the physical structure of ICs to HT insertion, specifically, by an untrustworthy foundry. Of the identified and quantified layout-level vulnerabilities are the amount of unused routing channels and the amount of white space in the different regions of the IC [168, 169].

**Design-for-trust: Trustworthy computing on untrusted components.** The third category of design-for-trust HT countermeasures, trustworthy computing on untrustworthy components (TCUC), is mainly concerned with achieving resilience against attacks. In the context of HTs, TCUC is about designing ICs with a certain level of tolerance against HT attacks, that would allow them to withstand the consequences of HT activation and, optimally, recover from them. One approach to TCUC is the implementation of on-chip component redundancy or duplication (i.e. having multiple copies of on-chip components) and comparing the outputs of the duplicate chips to infer if any of them are untrustworthy [170]. Task scheduling was also proposed to prevent collusion between 3PIPs bought from the same vendor [171, 172]. Using techniques known for error detection in the sphere of TCUC was also proposed in [173] and [174], in which concurrent error detection and proofs of correct executions are utilized in assessing the trustworthiness of operation. The authors in [175] tackled the issue of run-time HT activation from the perspective of depriving the HT-infected component from resources needed to fully realize their attack, such as power and input data.

**Split manufacturing.** The concept of split manufacturing, although first patented in 2004 [176], mostly caught the attention of hardware security and trust researchers after it was introduced as the focus of the US IARPA’s (intelligence advanced research projects activity) trusted integrated chip program in 2011 [177]. In the basic form of split manufacturing, the circuit layout is divided into a front-end-of-line (FEOL) part and a back-end-of-line (BEOL) part. The FEOL constitutes the transistor blocks and one or two metal layers and is manufactured in a state-of-art untrustworthy foundry. After the fabrication of the FEOL, the chip is shipped to a trusted (with potentially less advanced technology) foundry for the addition of the remaining BEOL metal layers [178]. Given that split manufacturing conceals details about the “metalization” of the IC from the untrusted foundry, it could be considered a form of hardware obfuscation. Research shows that the basic form of split manufacturing (referred to as 2D) could be attacked by proximity and SAT (Boolean satisfiability problem solving) attacks, in which the untrusted foundry deduces information about the BEOL layers.
based on their knowledge of the floor-planning, placement, and routing processes conducted to translate netlists into physical layouts [179][180]. Several improvements were, therefore, proposed to overcome this issue, from randomizing the circuit netlist before the placement and routing process [181], to selectively choosing wires in BEOL and FEOL [182], and to locking the FEOL layer [183]. Combining split manufacturing with logic-level hardware obfuscation and self-testing circuitry was also proposed [184]. Additionally, new types of split manufacturing, namely 2.5D and 3D split manufacturing, started emerging. In 2.5D split manufacturing, the layout of a given circuit is split into two parts, with an “interposer” connecting them in a certain way, the two parts are fabricated in an untrusted foundry whereas the interposer is fabricated in-house (or in a trusted foundry) [185]. 3D split manufacturing was mainly proposed for circuits containing security elements and hence is based on fabricating the conventional computational part of the circuit in an untrusted foundry while fabricating the security parts in a trusted one or in-house [186].

2.4.2 Countermeasures to IP piracy

IP and IC piracy takes many shapes and a pirated or counterfeit IC might be recycled, remarked, overproduced, cloned, tampered with, or provided defective and/or with forged documentation. The variation in counterfeiting multiplies the difficulty of countering it as, for instance, we can put an IC under tests similar to HT detection tests to infer if it has been tampered with or provided defective, however, such tests would not be useful in detecting cloned or overproduced ICs as they are expected to be functionally intact. Researchers have realized this challenge and attempted to tackle it with developing a wide range of countermeasures, taking the factors affecting the types of IP counterfeiting and the symptoms and/or faults resulting from them into account. On a high-level, countermeasures to IP piracy or counterfeiting could be of detective or preventive nature. On one hand, detective countermeasures are designed to detect the abnormal physical and/or electrical features expected to be found in counterfeit ICs, specially those bought from unauthorized vendors or resellers, or provided through potentially untrustworthy means. On the other hand, preventive or counterfeiting avoidance measures aim to introduce elements and structures to the design of ICs that would make them less susceptible to counterfeiting and piracy. Evidently, preventive countermeasures to IC counterfeiting (i.e. on the IC design level) are generally more suitable for implementation in new and in-production ICs.
rather than obsolete ones with pre-defined and fixed designs.

The detection of counterfeiting marks or residues in ICs could be implemented via physical or electrical inspection. The physical inspection is based on detecting anomalies in the physical, chemical, and material characteristics of the IC under test and its packaging. Whereas, in electrical inspection, ICs undergo certain tests to assess their electrical characteristics, performance metrics, and functional correctness, and verify that they conform (or not) with the expectations of an authentic IC [85]. Figure 2.7 shows a taxonomy of the main physical and electrical inspection methods.

![Figure 2.7: A taxonomy of detective countermeasures to IC/IP piracy](image)

The first method of physical inspection is initial or incoming inspection, which is a preliminary and routine low-overhead visual inspection that incoming ICs (i.e., to the facility of the SoC integrator or technology company) should go through. In this inspection, the texture and color of the IC package, in addition to the marks on it are inspected using a low-power microscope [187] or infrared cameras [188] to verify their clear and clean appearance, and that the IDs on the IC packages are identical to those on the box they arrived in. The contacts or leads of the IC should also be inspected to detect the existence of any bending, bowing, insertion marks, or unusual shine, all of which are signs of recycled and remarked ICs [189]. X-ray imaging is another technique for incoming inspection, in which images of the IC are captured and compared with some “golden” reference [85].

Exterior physical tests are mainly concerned with thorough yet non-destructive inspection of ICs. This could be done through dimension analysis, in which the dimensions of the IC are precisely measured to infer any deviation in actual dimensions from those indicated in specifications. Additionally, as the types of materials used in IC packaging vary, so do the techniques and solvents used to dissolve them, therefore, ICs could undergo chemical tests to confirm if their original packaging was removed and if they were repackaged with, potentially, another material. Analyzing the material of the IC package could also be performed through
the collection of samples of this material, obtained from superfine non-damaging blasting of the package [85]. Scanning acoustic microscopy is a scanning and imaging method that uses the reflection of acoustic waves in imaging microscopic structures. In the context of physical inspection of ICs, scanning acoustic microscopy could be used in non-destructively scanning the IC for potential defects without the need to de-package it [190]. This could be specifically useful in detecting delamination, cracks, and inner physical defects in ICs that are defective and originally rejected and discarded by the manufacturer but later offered for sale by some external party.

Interior physical tests require decapsulating the IC in order to prepare it for internal inspection. This could be performed through well-known and commercially available chemical, mechanical, and laser based tools and techniques that are also used in the first steps of reverse engineering ICs. Once removed from its encapsulation, the structure of the IC becomes exposed, allowing the inspection of the integrity and age of the silicon die and the bonds, through tests such as the wire pull test. In wire pull tests, the tension between the die and the bonds is tested, and the results of this test are used to infer the age of the IC, since the die-bond tension normally degrades with time. In parallel with the steps of reverse engineering ICs (e.g. for HT detection), a scanning electron microscopy could also be used after the decapsulation to inspect the internal layers of the IC [85]. The use of machine learning techniques in aiding the processing of images obtained from exterior or interior physical inspection of ICs, and in improving and automating the process of distinguishing between images of counterfeit IC and those of genuine ones was also proposed [191]. The fourth type of physical inspection methods is material analysis, which is concerned with verifying the integrity of the materials used in the IC and their chemical composition. To implement IC material analysis, techniques could be borrowed from the field of food and drug quality checking; such techniques include ion chromatography and energy dispersive X-ray spectroscopy [85]. In [192], Diffuse Reluctance Infrared Fourier Transform Spectroscopy (DRIFT), which is a non-destructive technique based on measuring the diffracted signal intensity when an infrared signal is projected on the material, is used along with machine learning techniques in characterizing and identifying counterfeit IC material.

The first method of electrical detection of counterfeit ICs is parametric testing, where the DC and AC parameters of a chip are tested to verify their conformance with specifications and expectations. Those tests, are, in principle, consistent with conventional post-silicon validation tests of ICs. For instance, examples of DC tests
include power consumption test and threshold tests whereas examples of AC tests are rise and fall time tests and propagation delay tests. Parametric testing is particularly useful in detecting recycled ICs as degradation in the parameters of ICs is strongly tied to their aging [193]. Functional validation is a staple practice in the semiconductor industry that is used to functionally verify ICs before their deployment in the field. To this end, this validation could also be used to test the integrity and authenticity of the IC, and detect types of counterfeiting where the functionality of the IC is affected. This is mainly concerned with tampered or defective ICs but could also be applicable to other types of counterfeiting as, for example, a recycled IC might exhibit functional incorrectness due to aging, while an IC built with older technology but provided with documentation stating otherwise (i.e. forged documentation), is expected to have less efficient performance [85, 194].

Burn-in tests are a type of stress-test, where ICs are operated at high temperatures and the points (i.e. temperatures) where they start showing failures are observed [85]. For example, ICs built for military and space applications are designed to withstand more extreme temperatures compared to common commercial grade ICs. Therefore, if in burn-in tests, such ICs show early failure, it could be because they are in reality commercial grade ICs counterfeit as military or space grade. Structural tests are performed after reverse engineering the IC and are based on identifying faults such as stuck-at and path delay faults in ICs [85]. Such faults primarily indicate an out of specification or defective IC, but also incompetent or incomplete IC cloning, where the adversary couldn’t deduce and clone the exact circuitry of the original IC.

Although published counterfeiting detection tests are capable of detecting most types of counterfeiting, putting all incoming chips in a given batch under those tests is time consuming and could be expensive, specially when automated methods are not available. This realization prompted researchers to develop counterfeiting countermeasures that are integrated in the design of chips and that would aid in the post-silicon detection of counterfeit IPs or would hinder adversaries from counterfeiting. In literature, such countermeasures are referred to as counterfeit avoidance measures or design for anti-counterfeit [85, 194]. A taxonomy of counterfeit avoidance methods is presented in Figure 2.8. This taxonomy is loosely based on the one presented in [85] with additional categories for the most recent advance in the field, blockchain enabled supply chain, and for split manufacturing and obfuscation, which, in this context, are used in obscuring the full layout of chips from an untrusted foundry and preventing them from cloning, or overproducing..
Figure 2.8: A taxonomy of preventive countermeasures to IC/IP piracy
The first category of counterfeit avoidance methods is about embedding structures in ICs that would give them unique identities. Physically unclonable functions (PUFs) are elements that exploit the unpredictable inherent physical variations of ICs, and are used in generating unique IC identifiers or signatures in addition to their usage in the generation of cryptographic keys. When integrating a PUF in the design of a given IC, it could be used after fabrication to generate a unique ID for the IC, which could then be saved in a database for comparing the IDs of genuine ICs with those encountered in the market. This would particularly lead to detecting cloned or overproduced ICs [85, 194, 195]. Additionally IDs could be saved in one-time programmable on-chip memories to uniquely identify chips. Such IDs are referred to as electronic IDs.

Hardware metering is a method mainly used to enable IC designers from tracking the production of their ICs in a given foundry based on some pre-defined security protocols or rules. In hardware metering, ICs could be passively tagged with, for example, serial numbers printed on chips or unique identifiers saved in on-chip memories. As passive metering is prone to tampering, active metering techniques are considered more effective and secure, and are generally based on locking and/or unclonable functions [196, 197]. For example, an IC design house can generate IDs and insert locks in certain parts of the design of a given IC, the design will then go through the synthesis, placement and routing process and a GDSII file will be generated and sent to the foundry. After fabrication, the foundry will scan the IDs of the ICs and send them to the design house, which, in turn, will send the keys to the foundry. By locking the functionality of ICs and providing locks only when a valid ID is provided, foundries are hindered from overproducing ICs [163]. Secure split test (SST) takes active hardware metering a step further by integrating scan-locking in addition to functionality locking. As functionality-locking prevents a foundry from overproducing ICs, scan-locking is used in preventing the scanning of ICs, once their functionality has been unlocked, mainly to prevent cloning. Scan-locking is achieved by locking the test features in the IC so that correct testing results are only produced when the correct key is applied [198].

In order to combat the recycling of ICs, techniques exploiting the aging phenomenon of ICs and using them to identify new from recycled ICs have been proposed. In one approach, the use of on-chip “age” sensors based on ring oscillators is proposed. In this approach, two ring oscillators are integrated in the IC; one is designed to age slowly and is used as a reference while the other is designed to age at a faster rate, and is referred to as the stressed oscillator. As the circuit operates in the
field, the stressed oscillator starts aging rapidly, which gets reflected in a decrease in its oscillation frequency whereas the frequency of the reference oscillator remains largely static. By comparing the oscillation frequencies of the two oscillators, a SoC integrator could infer if they are being sold a recycled IC [199, 200]. In another approach, the frequency of the on-chip ring oscillators along with the operating conditions, such as, temperature and supply voltage, are saved in memory and used as digital “age” signatures of the IC [201]. Aging sensors that do not need dedicated circuitry were proposed in [202], in which the age of the IC is inferred by reading out the start-up state of on-chip SRAM. This approach takes advantage of the fact that an equal number of cells power up to logic “1” or “0” only in new unused SRAMs, while in older SRAMs, skews are observed in this distribution. The utilization of on-chip counters in counting the IC operating cycles or the times of switching in a certain part of the circuit was also proposed. The output of the counter in this approach is saved in anti-fuse (could only programmed once) memory to ensure their protection from wiping by adversaries attempting to recycle the IC [86].

Hardware watermarking and hardware steganography are other techniques to achieve IC-level protection against counterfeiting. A hardware watermark is a signature introduced by the IC designer and embedded deep within the IC design and structure. Watermarking is mostly performed on the high synthesis level or the logic synthesis level. In one of the earliest publications on hardware watermarking, the watermarking is realized through additional timing and design constraints that embed the signature of the IP or IC designer and are translated through the high level synthesis process into actual elements (e.g. registers) that form a part of the IC design [58, 203, 204]. An important requirement with watermarking techniques is that the IC designer should ensure that once translated to RTL, the added constraints, do not affect the correctness of IC functionality. Hardware steganography could be viewed as a more robust alternative to watermarking where, for instance, cryptography modules and security elements are used in generating the additional constraints [58, 205, 206]. Both hardware watermarking and steganography do not directly prevent counterfeiting as they are primarily used for detection of piracy (i.e. the seller of a cloned version of a watermarked IC doesn’t have knowledge of the watermark particularities) and proof of ownership of a certain IC design, aiding hence, in ownership conflicts. Nevertheless, they could also help in deterring adversaries from IP piracy as they would be aware of their inability to prove ownership.
2.4. Towards more trustworthy hardware

The implementation of chip-level counterfeit avoidance methods is expected to deter adversaries from the most common types of counterfeiting, given the inability of an average-skilled adversary to circumvent such methods. However, these methods are embedded in the design of ICs hence only suitable for implementation in new IC designs. ICs of standard or obsolete designs that are already being circulated in the market or no longer produced cannot be protected by chip-level countermeasures. This realization resulted in the development of package-level counterfeit avoidance methods that are based on adding a uniquely identifying and non-easily clonable (or reverse engineered) layer on the packages of ICs without the need to directly modify the IC or its packaging. DNA marking is one method of package-level counterfeit avoidance. In DNA marking, botanical DNA is used in creating unique genetic sequences, that get integrated with ink and used in coating IC packages. Such DNA marks cannot be copied, and as DNA is an internationally recognized forensic evidence, they can also be used in establishing an indisputable proof of ownership [207]. Another package-marking technique is based on patterns of nanorods. Nanorods are rod-shaped nanostructures created from, for instance, gold or carbon. Nanorods could be grown into arrays used to form patterns, that when repeated would create a distinct overall pattern, as with every repetition, the angle and length of the nanorods change. The final pattern is then printed on the surface of the IC package and used to uniquely identify it [208].

Split manufacturing for the purpose of counterfeit avoidance is based on the same concept of split manufacturing as a countermeasure to HTs, as the achieved obfuscation and obscurity would prevent an untrusted foundry from intercepting the full IC design hence, in this case, prevent cloning or overproduction. Camouflaging and obfuscation are HT countermeasures that could also work in preventing cloning or overproduction of ICs. The fact that none of the chip-level or package-level counterfeit avoidance techniques can prevent or deter all types of IP piracy is alarming and shows how vulnerable semiconductor chips are to counterfeiting. This is especially the case with commercial grade ICs where, IC designers, for overhead and time-to market-reasons, are not expected to implement countermeasures for each of the potential types of counterfeiting. In an attempt to address this issue, a blockchain-based framework for the authenticity and integrity of ICs was proposed in [209].

A blockchain is a vertical stack of blocks, where each block represents a single transaction and is connected to the previous block (i.e. its parent block) in the stack. The blocks in a blockchain are connected via cryptographic means as the identifying
header of each block contains, along with a timestamp of its creation, an encrypted version of the header of its parent block. As each block is (cryptographically) connected to the block before it, this forms a chain, in which data in any given block cannot be modified without causing the modification of data in all subsequent blocks. In addition to preventing the tampering of the information in each block, blockchain technology provides a decentralized method of authentication, eliminating the threat of a malicious central authentication party. In the context of the IC supply chain, blockchain technology provides decentralized authentication of ICs involving all parties that form a part of this supply chain. The implementation of the framework requires a database accessible by all parties (commonly referred to as entities) in the IC supply chain, and the unique identification of individual ICs through external marking and intrinsic IDs such as electronic IDs saved in memories [194, 209].

The framework starts with the IC designer sending a request containing the ID of a new IC to all entities in order to enroll this new IC in the blockchain. After mutual agreement and verification, where each entity verifies this request with the other entities in the chain, the ID of the new IC is enrolled along with a timestamp in a new block in the blockchain database. When the IC is ready to move to the next step in the supply chain, for example, the foundry, the IC designer announces a request for ownership release so that it can send over the IC to the foundry. After mutual verification by all entities in the chain, the foundry receives the IC and shares the received marking or identification with the other entities in the chain, which would (normally) verify that the information received is the same as the one saved in the blockchain. At this stage, full verification is achieved and the foundry can ask for ownership acquisition from the network which in turn provides the foundry with an ownership certificate and updates the “label” of the IC in the blockchain to indicate its current state of being in the fabrication stage. The steps are repeated until the IC leaves the supply chain to the market and finally to the end user, who could also verify the authenticity of the IC by sending a verification request to the blockchain [194, 209].

In addition to its advantage in preventing unauthorized external parties from getting involved in the supply chain of ICs, the labeling performed in this framework, in which, the stage of each IC is clearly indicated prevents recycling or remarking. For example, when provided with a “presumably” new IC, the SoC integrator could check if this is actually the case by announcing the ID of this IC in the blockchain and obtaining information about its stage. A pre-used IC would have already gone through system or SoC integration before and hence its label would indicate that.
Although implementing a blockchain framework in the IC supply chain would not technically prevent foundries from overproducing ICs, it would prevent them from moving them through this blockchain-enabled supply chain, making the process of putting them in the market more challenging. Cloning, however, would still be possible with a blockchain enabled supply chain. To this end, the authors of [209] propose the incorporation of PUFs in the blockchain enabled framework for the purpose of identifying and authenticating ICs. From the provided literature review on countermeasures to IP piracy, it could be concluded that so far, a universal countermeasure to IP piracy, although attempted by techniques such as the blockchain enabled supply chain, has not yet been developed.

### 2.4.3 Countermeasures to fault injection attacks

Fault injection attacks generally originate from sources that are neither part of the IC supply chain nor have any direct role in the design or selling of ICs. They are also initiated on ICs in their deployment environment, hence once they are physically out of the hands of their designers and manufacturers, and possibly, the hands of the original buyer. These realizations combined with the fact that in fault injection attacks, the attacker is expected to have at least access to the vicinity of the IC, if not the IC itself, multiply the threat of fault injection attacks and also multiply the difficulty of combating them. Given the unpredictability associated with the faults that a malicious user of a given IC could inject, IC designers, in the best case scenario could only employ countermeasures that would make fault injection considerably difficult for the attacker (comparing with attacking ICs with no countermeasures deployed) but not necessarily impossible. Countermeasures to fault injection attacks mostly fall under two categories; Intrusion detection and error or fault detection. Intrusion detection is about detecting and potentially preventing physical tampering attempts and hence are mainly concerned with invasive and semi-invasive fault injection attacks. Error or fault detection is about integrating elements in the design of the IC that would allow the detection of faults. This approach is specially useful in detecting non-invasive fault injection attacks as they do not leave any intrusion or tampering evidence.

Shielding is one of the first intrusion detection methods that were proposed to detect and counter probing and focused ion beam (FIB) fault injection attacks. Shielding is adding a metallic mesh on the top metal layers of the IC, hence, sandwiching it between the substrate (the chip “back-side” supporting material) and
Chapter 2. Background

the mesh (i.e. shield). Ideally, the mesh acts as a fence to the IC underneath preventing attackers from accessing it without causing some physical or electrical changes in the mesh hence leading to the detection of the attack. The very first type of proposed shielding was based on detecting changes in the capacitive properties of the shield in detecting, mainly reverse engineering, but also other types of invasive attacks [210]. Later, it was shown that this type of shielding could be circumvented by careful and accurate probing and FIB attacks that would not cause any out-of-bound changes in the mesh capacitance. A more common form of shielding is active shielding, which uses a wire mesh that carries a dynamic signal, used in transmitting a certain sequence of bits or a bit pattern. The received pattern is compared with the same pattern from the lower metal layer and any discrepancy is considered a result of an invasive fault injection attack. A constant challenge with active shielding is increasing the difficulty of breaking or bypassing the bit pattern while maintaining the complexity of the mesh within reasonable limits that would not pose unacceptable overhead on the IC. The use of pre-determined bit patterns [211] and linear equations [212] were some the first introduced methods, however, later research showed that they could be guessed by attackers [17]. Random active shielding with densely routed meshes was also proposed [213, 214]. Such shielding resembles a maze or a cage, as denoted by the authors of [214] as its intertwined geometry would be difficult, if not impossible, to navigate and comprehend by a potential attacker. Nevertheless, this type of shielding is not easily scalable due to its complexity. Additionally, a more sound implementation of it would require multiple additional mesh layers, which is generally viewed as undesired overhead.

Cryptographic active shields were introduced in [17, 215] as an attempt to overcome issues associated with other forms of active shielding. In cryptographic active shielding, two cryptographic blocks, one for the mesh and one for the layer below it, are used to generate the same bit pattern when provided with the same key and input vector. The generated bit patterns are exchanged between the two layers and compared. The more times this process is performed, the more likely it is to detect an invasive attack as the attacker is not expected to predict the bit values correctly in sequence given that they do not possess the information used to generate them (i.e. key and different input vectors). In more recent work, low-overhead internal shielding, where shields are integrated within IC layers is proposed [19]. In internal shielding, the functional parts of the IC are leveraged to create a shield protecting the sensitive or confidential parts of the circuitry. This approach leads to
2.4. Towards more trustworthy hardware

shields being distributed throughout the internal metal layers of the IC instead of having them implemented solely on the top layer as is the case with other shielding techniques. However, to achieve this, the entirety of the physical design flow need to be altered to incorporate the integration of these shields. As chips could be attacked from their back side, especially using laser beams that would easily penetrate through the substrate, researchers developed several backside shielding techniques. One technique is based on weakening the substrate with miniature cavities that would not have any negative effect on the IC in normal operation but that would physically break if milling (i.e. with FIB) is attempted [216]. Another technique involves a backside metal shield that is connected to the surface of the circuit with vias and coupled with current detectors [217]. The connection between the backside shield and the circuit allows current flow. This current is monitored by current detectors to detect any changes in the current, that are likely to be induced by invasive fault injection attacks. Regardless of the type of shielding, the detection of an intrusion in the IC is usually followed by the triggering of an alarm and a security action such as the erasure of secret information.

Another intrusion detection approach is concerned with using analog sensors in detecting changes in circuit properties, as caused by fault injection attacks. The frequency of ring oscillators is sensitive to their load capacitance, supply voltage, temperature, and process variations. Therefore, in [16], analog sensors exploiting ring oscillators in detecting capacitive changes caused by probing or FIB attacks are introduced. In this approach, a ring oscillator is used to excite multiple wires with a certain frequency. If a probing or FIB attack occur, the capacitance of the affected wire/s would change leading to a change in the frequency of oscillation of the affected wire and hence the detection of the attack. Another FIB sensor-based detection technique was proposed in [218], where charge sensors are used to detect the charge deposited in the initial navigational (i.e. when probes are approaching the target point) steps of a FIB attack. In [219], it was shown that probing attacks could be detected by measuring the delay difference when sending the same signal simultaneously on two symmetric bus lines. This is because when probes are connected to any on-chip wire or line, they act as additional parasitic capacitance, which is reflected in additional delay in signal transmission.

Sensor-based intrusion detection techniques were also developed to detect semi-invasive fault injection attacks. In [220] oscillators are implemented as sensors around a cryptographic element (i.e. likely target of an electromagnetic attack). The
proximity of the microprobe used in electromagnetic attacks to any of the oscillators will cause a change in its oscillation and a detectable difference in frequency in comparison with the other oscillators. Another approach to detecting electromagnetic fault injection uses multiple delay detectors, originally used to detect clock and power or voltage glitches, in detecting the glitches induced by an electromagnetic fault injection before the induction of actual faults in the operation [221]. A Phase locked loop (PLL) is a circuit with an output phase that is linked to the input phase, any disturbance in the input of the PLL “unlocks” it and causes it to go into several cycles of balancing and adjusting its output so that it is in-phase with the input. In [222], a PLL is connected to an internal clock as its input source, and whenever glitches induced by an electromagnetic fault attack affect the frequency or phase of the internal clock, they cause unlocking of the PLL and the detection of the attack.

As optical fault attacks cause the sudden switching of single transistors, some of the first countermeasures are based on eliminating the dependence on single transistors by designing circuits to take decisions only after referring to the logic values of multiple transistors, that in normal cases would not be at logic high or logic low at the same time [92]. The use of on-chip optical sensors to detect the illumination of laser beams, and software-based validation of the decisions made in hardware are also potential countermeasures to optical fault attacks [93]. As for temperature attacks, temperature sensors could be used to detect abnormal changes in temperature, although this approach is susceptible to circumventing by localized and fast temperature attacks [95].

In error or fault detection based countermeasures to fault injection attacks, additional elements are added to the circuit design to detect the errors and faults introduced by fault injection attacks. One approach is based on redundancy, in hardware, time, or information. Hardware redundancy is concerned with having multiple copies of the same on-chip element and comparing their outputs. The same approach when combined with vendor diversity, where each copy of the element is bought from a different vendor, could also be used in detecting hardware Trojans and counterfeit IPs. Time redundancy is about having the same process computed or performed by the same on-chip element multiple times. This type of redundancy allows temporal distancing between similar computations and hence the detection of any discrepancy in any of the computations. In information redundancy, error detection and correction techniques are used to transmit or process redundant information along with actual information. This is followed by verifying the integrity
of the redundant information to infer the integrity of the actual information. Clearly, all redundancy-based 
techniques suffer from overhead issues either in hardware or in 
performance [88].

Another approach to error detection is the deployment of on-chip glitch detectors. A basic form of voltage 
 glitch detector could be based on monitoring the supply lines of the IC, however, short voltage glitches 
might not be detected by this method and passed off as noise. Other approaches to voltage glitch detection 
are based on; monitoring faults induced in single bit registers [223], operational amplifiers that 
monitor the difference in output between voltage dividers connected to the circuit supply lines [224], and 
voltage-sensitive circuit structures that exhibit notable and easily detectable behaviour when a voltage glitch is 
introduced [225, 226]. As clock glitching attacks are characterized by varying the length of clock cycles, a number of 
measures countering them are based on on-chip delay detectors. When the delay generated by delay 
detectors is less than the clock period, no action is taken, whereas if the delay was found to be greater than the clock period, an alert is raised and the output of the combinational logic is discarded as it is deemed faulty [227]. In [228], a 
timing monitor based on ring oscillators is introduced. In this approach, a ring oscillator is deployed as an internal on-chip clock that runs faster than the external clock and is used in sampling it. The difference in frequency between the two clocks is saved in an on-chip memory, and compared with the expected reference difference, any deviation is flagged as a potential clock glitching attack. A technique utilizing ring oscillator PUFs in detecting clock glitches is proposed in [229], in which the outputs of the counters in the PUF, which reflects its oscillation frequency, are compared with some reference value to detect glitches, if any.

An emerging approach to countering fault injection attacks is based on reducing the vulnerability of circuits to such attacks by assessing their susceptibility to them in the design stages. Most publications on this topic are aimed at certain cryptographic implementations [230-232], however, a more recent publication proposed a universal and automated fault injection susceptibility framework. The framework is based on identifying the potential fault injection locations in a given IC through the gate-level simulation of fault injections with respect to desired security properties such as integrity and confidentiality. By identifying the parts of the circuit that are more susceptible to fault attacks, the framework allows the proper and lightweight placement of sensors and countermeasures to fault injection attacks [88].
2.4.4 Countermeasures to system-level hardware attacks

System-level attacks on hardware are initiated from the upper layers in computing systems, mostly from the firmware or software layers, but also remotely over the network. Consequently, trust and security measures designed to detect or counter these attacks are implemented across the different layers in computing systems. Given that the spectrum of system-level hardware attacks is broad and covers a wide range of both generic and application or device-specific attack, the range of countermeasures targeting them is equally wide and varied. For the purpose of showing the different approaches to system-level hardware attacks without focusing on particular systems or hardware implementation, this section presents some of the main countermeasures developed to mitigate the universal attacks presented in Section 2.3.4: DRAM attacks, cache attacks, and speculative execution or performance optimization attacks.

Countermeasures to system-level attacks on memory systems focus on enforcing memory isolation and access control. The aim is to strictly assign different regions of memory to processes or software of different criticality and security levels, and ensuring that secure memory areas are never accessed by a non-secure process or software. Additionally, dynamic policies are implemented to ensure, for example, that secret or confidential information such as cryptographic keys are written to the memory only once (i.e. in conventional cryptographic implementations) during boot-up and never accessed later. Conventionally, these measures in addition to other measures such as data encryption and hashing are performed by an on-chip memory management or protection unit [58]. Nevertheless, attacks on memories such as Rowhammer and cold boot attacks show that these countermeasures are not enough and that memory isolation could be circumvented and access control rules could be bypassed through the exploitation of the physical and operational properties of memories. Ever since their emergence in 2008, several countermeasures were developed to counter and prevent the disclosure and leaking of secret information via cold boot attacks. Some of the earliest countermeasures propose the storage of encryption keys out of the DRAM, primarily in CPU registers [233, 234]. This approach is capable of protecting encryption keys from DRAM cold boot attacks provided that access control measures are implemented on those CPU registers, that are otherwise, accessible by all processes running on the CPU. Nevertheless, this approach cannot be implemented when the amount of secret or sensitive information
exceeds the capacity of the small number of CPU registers. Full or partial memory encryption is another proposed countermeasure to cold boot attacks [235], however, this is more suitable for large-scale and high performance computing systems as lightweight or latency-sensitive systems cannot afford the performance and energy overhead associated with the need to encrypt or decrypt data on every memory access.

Memory scrambling introduced, mainly for performance purposes, in the newer DDR3 and DDR4 DRAM standards are also potential deterrents to cold boot attacks. In the simplest form of memory scrambling, an Xor operation is performed on data leaving the CPU with some sequence of pseudo-random numbers and then saved in the DRAM. When the data is retrieved from the DRAM, it goes through the same process to descramble it. In this context, memory scrambling serves as a data obfuscation mechanism preventing attackers from comprehending data obtained through cold boot attacks. Research published in 2013 [236] and 2017 [237] showed the possibility of performing cold boot attacks on DDR3 and DDR4 DRAMs, respectively, despite the implementation of memory scrambling. The attacks were carried on Intel’s DDR3 and DDR4 DRAM scramblers, and they exploited the fact that the scrambling and descrambling processes both use the same steps and the same sequence of numbers, and hence memory data acquitted through the cold boot attack could be brought back to its plain-text form by scrambling it again. The sequence of numbers used in scrambling was obtained either by deduction from the scrambling of memory locations with zeros (i.e. 0 xor key = key) or by exploiting the correlation between memory locations and the keys or sequence of numbers used to scramble them. The researchers in [237] proposed using cipher-based memory scrambling to thwart those attacks. A more proactive countermeasure to cold boot attacks was proposed in [238], in which circuitry is added to the DRAM to self-lock it and erase its contents, whenever an attempt to a cold-boot attack in the shape of sudden system boot or power on/off is sensed.

One of the first publications on countermeasures to Rowhammer attacks analyses the amount of leakage current that needs to be induced by row-hammering to cause errors and bit-flips. Based on this analysis, a row-hammering threshold is derived. Counter or probability based approaches are then used to detect if the number of activations of a particular row reaches the identified threshold, after which, a refresh command is issued for the neighbouring row to mitigate the affect of its hammering and retain its data [239]. Increasing the frequency of refresh was also proposed in a number of other publications that adopted different techniques to deduce when the
row-hammer threshold is reached \cite{240,242}. The use of error correcting codes to flip-back Rowhammer induced errors, and static remapping or retiring of DRAM cells that are found to be susceptible to Rowhammer attacks in the pre deployment tests are other types of countermeasures to Rowhammer attacks \cite{243}. In \cite{244}, a framework for identifying memory cells vulnerable to Rowhammer attacks based on analysing the features and behaviour of memory cells and their correlation with bitlines is introduced. The framework could be implemented in the manufacturing stage or in-field.

As is the case with memories, caches are storage elements where isolation of secure and non-secure accesses is adopted as a countermeasure to information-leaking attacks. This isolation requires partitioning of the cache to a part or space accessible by normal non-secure processes and another that could only be accessed by privileged and secure processes. Given the limited capacity of caches, a challenge is to achieve efficient partitioning where each process is given a number of cache lines based on its current needs instead of a pre-defined number. A potential way to overcome this challenge is through flexible partitioning. In one form of flexible partitioning, two tags are added to each cache line to indicate its owner (i.e. using process) and whether it is locked or not. Cache lines that are not locked are dealt with normally with new unlocked data replacing old unlocked data when a cache miss occurs, whereas, data in locked cache lines can only be replaced by new data belonging to the same process. This partitioning is referred to as partition-locked cache or PLcache \cite{245}. Although PLcache have been long considered secure, research presented in \cite{246} showed that due to the fact that a PLcache acts like a normal cache when a cache hit occurs and leaves the same traces (i.e. information about last accessed cache line), it could be exploited by adversaries. A mitigation for this vulnerability is proposed in \cite{246}, in which additional logic is added to the synthesis of the cache so that a cache hit does not change the state of hardware or leave trackable traces. Other cache partitioning schemes include hardware-based dynamic reservation of cache lines for active threads \cite{247} and the introduction of partitioning based on multiple levels of protection domains with dynamically allocated cache miss and hit restrictions and policies \cite{248}. Another way to counter system-level cache attacks is through randomization which serves as an obfuscation of the last evicted cache line and cache access patterns \cite{245}.

The disclosure of Spectre \cite{59} and Meltdown \cite{60} and subsequent attacks targeting speculative and out of order execution left the computing community in a dilemma over how to address this family of attacks while maintaining and preserving
the performance and microarchitectural benefits of out-of-order execution. The very first countermeasures to these attacks are based on software implementations \cite{249,251}, however, researchers quickly realized the performance, efficiency, and cross-implementation limitations arising from these countermeasures and started developing hardware-level countermeasures. Consequently, a large number of hardware-level countermeasure to speculative execution attacks emerged, many of which are based on the notion of obscuring the traces of speculative execution. A microarchitectural framework that eliminates speculative execution attacks based on delaying the forwarding of data needed for the execution of a speculated instruction until it is confirmed that this is not a mis-speculation was introduced in \cite{252}. The framework could be tuned to fit different performance and security trade-offs depending on the system at hand. The introduction of a speculative filter between the processor and the level 1 cache was proposed in \cite{253}. The speculative filter acts as a temporary storage element where all speculative execution state and traces are saved, whereas, the main caches, that are shared between “potential” attacker processes and “potential” victim processes, are used for non-speculative data. Similar techniques based on memory hierarchy and the storage of speculative data in temporary structures until they are committed were also introduced in \cite{254} and \cite{255}. Hybrid software and hardware machine-learning based techniques for detecting speculative execution attacks at run-time were also proposed \cite{256} \cite{257}.

### 2.5 The last line of defence

From the literature review presented in this chapter, it could be concluded that a plethora of countermeasures to hardware trust threats and issues have been developed. In spite of these efforts, we cannot assume that hardware trust issues, although partially tackled, have been completely addressed due to a number of reasons. First of all, the vast majority of countermeasures to hardware trust issues such as, hardware Trojans, have been developed on hypothetical grounds as technical and detailed reports of real-life Trojans are almost non-existent. Second, the research community and hackers alike continuously come up with new types of hardware attacks emphasising on the fact that we are still faraway from a comprehensive understanding of the range of threats surrounding hardware. Examples of new hardware attacks announced in 2020-2022 are re-programmable hardware Trojans \cite{258}, and new
variants of speculative execution attacks capable of breaking known mitigation to older types of speculative execution attacks [61]. Third, is the rapid advancement and miniaturization in the IC industry, which on one hand, makes it harder for the average attacker to insert Trojans or clone an IC, but on the other hand also makes it harder for the defender to anticipate where a Trojan could be hidden or how the circuit might behave under fault injection conditions (e.g. cross-talk between memory cells is expected to increase with smaller transistors). The possibility of the emergence of combined attacks is also another concern as such attacks combine multiple threats to hardware trust, a relatively recent example of such attacks are speculative execution attacks where two vulnerabilities are exploited; the speculative execution traces left in the system and the cache side or covert channel. The existence of such attacks dictate the need to implement a number of different countermeasures in circuits, which could lead to significant undesired overhead and eventually lead IC designers to compromise on security for the sake of performance, which is especially expected in IoT and other small-scale and lightweight systems. All of these factors lead to a single realization, that, in real-life settings, a system administrator does not know how, when, and even if their hardware would be attacked. This uncertainty leads to confusion and indecision on how to protect systems from an induced deviation in hardware behaviour and might result in under-protection where potential attacks are not mitigated against or over-protection where unneeded resource-consuming countermeasures are deployed. This is where the last line of defence comes into play. The last line of defence is a terminology used in this thesis to refer to run-time monitoring techniques that could be used in detecting abnormal behaviour at real-time. Once such behaviour is detected, the system could be designed to take certain decisions or perform pre-defined actions that would mostly start with informing the system administrator/ upper layers in the system of the hardware trust violation. This could be followed by actions such as rebooting the system or shutting it down, disconnecting the element that was found to be untrusted, or performing some remedial or corrective actions that would prevent the penetration of the abnormal behaviour. We refer to this as trustworthy computing on untrustworthy hardware (TCUH). The detection of the abnormal behaviour at run-time through run-time monitoring is an essential first step to achieving trustworthy computing on untrustworthy hardware as only once diverted behaviour has been observed, remedial action could be taken. Several publications on hardware trust state that trustworthy computing on untrustworthy hardware and the preliminary run-time monitoring are
key future research areas for achieving secure computing \[52, 53, 259\]. At its core, trustworthy computing on untrustworthy hardware is about systems being tolerant to hardware trust issues that might arise at run-time and being capable of dealing with them directly or alerting the system administrator in a timely manner. Besides the run time monitoring and trustworthy computing techniques surveyed in Section 2.4.1 which mainly target hardware Trojans, the field of run-time trust and security monitoring of hardware and TCUH is still in its infancy. In this section the main run-time monitoring and TCUH techniques proposed in literature are presented.

A parallel concept to trustworthy computing on untrustworthy hardware is fault tolerant hardware which is mainly concerned with systems capable of responding to faults or failures and recovering from them or going back to normal operation at run-time. In terms of impact on circuit and system operation, the faults could induce similar consequences to the healthy operation as those of HTs or system-level hardware attacks. They are also unpredictable and might pass verification tests undiscovered. This implies that the measures used in fault tolerant systems and hardware could also be used in run-time monitoring of hardware trust and security and in trustworthy computing. This was shown to be the case in a number of techniques that propose the use of redundancy and concurrent error detection in run-time HT detection \[170–174\], all of which are staple techniques in fault-tolerant systems. Nevertheless, there also exists a number of techniques that focus solely on run-time monitoring and trustworthy computing from the perspective of hardware security and trust.

One of the first dedicated trustworthy computing techniques was presented in \[260\]. This technique, referred to as Bluechip, addresses the issue of untrusted or malicious hardware by delegating its tasks to software. Bluechip is implemented in two steps, the first is in the pre-deployment stages and is about identifying the parts of the IC that were not activated during the verification tests, deactivating them, and inserting exception circuitry in their place that would alert the software whenever these circuits are expected to activate during operation. The second step is performed at run-time by the exception handling software that emulates the required hardware instructions (when needed) to allow the system to continue operation without disruption. Basing the activation of the hardware-substituting software on inactivated circuit parts infer that this is a technique built mainly for the threat of HTs and potentially, some types of counterfeit ICs.

The identification of security polices that IPs are expected to adhere to and the
run-time monitoring of this adherence was also proposed as a run-time security monitoring and TCUH technique. The security polices are related to security properties expected of a trusted operation such as access control and secure flow of information. An example of an access control policy is that the output of an on-chip crypto core cannot be observed by an IP other than its intended receiver. These security policies are implemented by a centralized microcontroller-based security element that uses information about the events observed and the activities conducted at the different on-chip IPs to infer suspicious or malicious behaviour. Accordingly, some active recovery technique such as partial or complete disabling of interactions with the malicious IP is performed. In this technique, the monitoring of IPs and the collection of information about events is conducted via security wrappers that base their monitoring on internal events in the monitored IPs such as the raising of certain flags or the accessing or updating of certain registers. This type of monitoring needs the integration of some parts of the security wrappers inside the monitored untrustworthy IPs by the IP providers, which raises concerns regarding the possibility of having the security wrappers maliciously modified by those untrusted providers. This issue was recognized by the authors, and the re-purposing of the trusted design-for-debug structures in verifying the actions of the security wrappers was proposed [261–263]. In [264], the use of this technique as a hardware patch was also proposed given that the security policies implemented by the microcontroller-based security element could be updated at run-time to accommodate for newly identified threats and attacks.

Several frameworks and techniques for the implementation of hardware patching through run-time monitoring, operational correctness verification, and error detection and recovery have been developed, mainly for the purpose of run-time verification of processors and memory systems [265–267]. In the context of hardware trust and security, however, hardware patching is a relatively new player. Trust and security-aware patching of hardware could be viewed as a form of TCUH that is mainly concerned with the capability of addressing new hardware threats or attacks as they become known. Due to their adaptable nature, however, hardware patches need to be based on programmable hardware elements, which could impose high overhead comparing with hard-wired solutions. In [268], a security-aware hardware patching framework is introduced. In this framework, programmable hardware blocks are deployed at the interface of the different on-chip IPs. These blocks monitor the inputs and outputs of the IPs to infer any abnormal behaviour in, for instance, the sequence
of requests at the interface or the expected values of the signals. The detection of an abnormal behaviour is then followed by the initiation of a corrective action such as amending the values of the control signals or replacing some incorrect data with correct data. In [269], a security hardware/software patch based on a programmable hardware monitor of processor cores supported by software is proposed. The proposed monitoring is based on collecting (from the monitored processor) traces of process execution, examining these traces and comparing them with known sequences of execution traces, and accordingly performing proper recovery action, if a discrepancy is detected.

Other run-time monitoring approaches include the use of on-chip hardware performance counters in monitoring the events occurring in processors and the number of their occurrences, and using this information in the verification of correctness of execution in relation to the executed programs [270]. The adaptation of formal verification and model checking techniques, commonly used in pre-deployment HT detection, to model either normal (mismatch-based detection) or abnormal (match-based detection) behaviour and designing run-time monitors aimed at monitoring the signals or states affected by this modelling have been also proposed [271, 272]. Another approach suggests the re-use of design for debug and design for test structures in security monitoring of on-chip IPs instead of disabling them after the verification tests, as is common in the industry [273, 274]. The utilization of circuit replication along with task scheduling in verifying the correctness and time properties of task execution has also been presented as a TCUH technique [275]. Tackling TCUH through a system-level approach have also been proposed in [276], in which security rule-based parallelism is deployed. In this approach, observations from the processor are distributed to a number of programmable security kernels that verify the security of processor actions and detect any violations to pre-defined security rules [276].
Chapter 3

Run-time monitoring of untrustworthy on-chip communications

3.1 Introduction

The literature review presented in chapter 2 shows the extent of hardware vulnerability to trust issues and the relative ease of driving SoC elements into erroneous, damaging, and potentially dangerous behaviour while in their operating environment. In spite of the many types of measures developed to counter and combat potential causes of hardware mistrust, it is largely still an open issue for a number of different reasons including:

- Pre-deployment HT detection tests, due to reasons such as the stealth and unpredictability of HTs and the hypothetical basis of some tests, do not guarantee complete clearance from HTs.

- Many design-for-trust techniques are based on assumptions that do not necessarily hold true in all cases. One such assumption is that IC designers are knowledgeable about hardware trust issues, are capable of deducing vulnerabilities in different SoC elements and of devising security rules and patching techniques for those vulnerabilities. The ability to request the addition of certain security monitoring elements inside IPs bought from untrustworthy providers is also another assumption. Although such techniques might be well within the means of large-scale technology companies, their feasibility for small-scale IC designers is questionable.

- The complexity and overhead associated with many hardware trust and security
measures and the time needed to realize them could hinder their realization in real-life systems. This is because SoC designers are unlikely to compromise on time-to-market and performance efficiency for the sake of protection from hardware trust issues that might never arise at run-time.

- Some types of HTs, fault injection attacks, and system-level attacks utilize the physical properties and features of semiconductors in realizing their attacks. The threat vector in this case is inherent to the nature of electronics and cannot be eliminated.

- System and architecture level attacks on hardware are continuously evolving and new attacks keep coming to the surface. This is expected given the many ways in which software interacts with hardware and the variations in system architecture. Mitigating all of those attacks with preventive design-level approaches or pre-deployment tests is infeasible and expected to pose significant overhead on resources.

These realizations about the feasibility and viability issues surrounding the development and implementation of conventional hardware trust measures led us to shift our focus from the cause and circumstances of hardware mistrust to its main sign; unexpected behaviour. We therefore discount the cause of hardware mistrust, be it a HT, fault, counterfeiting, or any other malicious or non-malicious cause, from our approach to achieving hardware trust, and concentrate our efforts on the mistrust-induced behaviour and its potential consequences. By isolating hardware mistrust from its cause, we simplify the hardware mistrust issue and model it in the shape of some non-anticipated behaviour that could be detected at run-time and potentially corrected, reversed, or prevented from escalation and penetration to the upper layers in the system.

This approach does not conflict with pre-deployment detection or design-for-trust techniques but rather complements them, providing a run-time assurance of trusted operation and of the success or failure of those pre-deployment techniques. We refer to this approach as the last line of defence, which is, in principle, run-time monitoring concerned particularly with verifying the behaviour and/or output of on-chip hardware elements, unlike some more conventional run-time monitoring techniques that are concerned with detecting side-channel signals of HT activation. Traits of what we consider the last line of defence are outlined in the following:
3.1.1 Traits of a ‘last line of defence’ technique

- Capable of performing continuous run-time monitoring of hardware elements based, preferably, on security elements implemented outside of the monitored element, especially, if the monitored element is a 3PIP.

- Capable of live instantaneous or periodic (i.e. depending on the nature of the unexpected behaviour, being of immediate serious effect on the healthy operation of the SoC or of a more prolonged effect) detection of abnormal and unexpected behaviour, output, and/or performance.

- The performed detection of abnormal behaviour is primarily based on clear and defined violations of normal behaviour, output, or performance and not on assumptions about potential attacks (although a combination of both approaches is possible).

- Capable of undertaking (or being extended to undertake) countering, remedial, and/or warning or alerting action once mistrust is detected. Therefore, actively achieving or assisting in achieving trustworthy computing on untrustworthy hardware.

- Capable of supporting claims of the detection of incorrect behaviour or action and of any preformed corrective action with explicit evidence such as logged information or comparisons with some baseline data.

These traits, in addition to the concepts of adaptability, independence, autonomy, and computational lightweightness, outlined in detail in chapter 1, form the conceptual and operational basis of the development of the three different last line of defence techniques presented in this thesis. The first technique, which is the focus of this chapter, is concerned with run-time assurance and verification of the trustworthiness of the actions and behaviour of on-chip communication interconnects, which are a fundamental part of the vast majority of SoCs.

3.1.2 Motivation

The growing architectural complexity of SoCs and the increasing number of elements integrated in them require the implementation of dedicated on-chip communication constructs to facilitate system-level communication beyond direct one-to-one
connection. Such constructs are referred to as communication busses and interconnects and their basic shape is that of a shared channel (set of wires and logic gates) connected to several on-chip elements for the purpose of transmitting requests and responses between them. This communication is typically dominated by certain communication protocols requiring the implementation of pre-defined steps in a certain order for the valid exchange and execution of transactions. In addition to communication buses and interconnects, other communication structures are network-on-chips (NOCs), which utilize the concept of router-based packet-switching in facilitating on-chip communication in complex multi-core SoCs.

The security and trust of NoCs have been well-discussed in literature, however, this is not the case for the structurally simpler, yet more ubiquitous, communication buses and interconnects. The relative simple operation and architecture of communication buses and interconnects might be the reason behind their security being rarely discussed in literature. As because of this “simplicity”, an IC designer may assume that pre-deployment functional validation, and deployment of on-chip protocol checkers are sufficient measures for ensuring the healthy and trusted operation of such simple elements. Nevertheless, it is a fact that communication buses and interconnects are standard circuit elements that are indispensable in all SoCs and ICs (with perhaps, the exemption of very lightweight SoCs where direct communication is adopted). This brings to our attention that, as is the case with most standard circuit elements, buses and interconnects are likely to be outsourced and bought from potentially untrustworthy IP vendors, given the remote probability that an IC design house would spend valuable resources on redesigning a very standard and common circuit element. In this case, having a HT-infected communication interconnect is a possible and realistic scenario. In addition to HTs, communication interconnects could also be bought from unauthorized sellers and hence could be counterfeited and suffering from performance issues that would potentially take the shape of run-time trust and security issues. Another threat is the possible utilization of communication interconnects as a vehicle in side-channel and system-level attacks, shown to be feasible in bus-assisted cache attacks [277], or in accessing and intercepting data-in-transit that cannot be easily accessed otherwise. All of these threat vectors indicate the pressing need to develop run-time monitoring for potentially untrustworthy on-chip interconnects. This is not to ensure that they adhere to the adopted communication protocol, which is already addressed by on-chip protocol checkers, but more importantly, to ensure that they do not perform any action
that is not expected or at an unexpected time. Particular actions of concern are those associated with or might lead to trust and security complications.

### 3.1.3 Contribution and chapter outline

The work presented in this chapter was published in IEEE European Test Symposium in 2021 as the following:


This chapter discusses the issue of untrustworthy on-chip communication interconnects and presents the developed run-time monitoring and trustworthy computing scheme. The remainder of the chapter starts with a literature review of related work, followed by the introduction of the targeted threat model and consequent abnormal behaviour scenarios. The guiding principles behind the proposed scheme, and its architecture, operation, and implementation are thereafter outlined. Finally, a discussion is provided on potential ways to improve the proposed technique and widen the scope of its application.

### 3.2 Related work

The run-time monitoring and detection of abnormal behaviour in on-chip communication buses and interconnects is rarely addressed exclusively in literature. The majority of relevant work is; generic to different elements in SoCs and only applicable to interconnects after some customization, concerned with the run-time security and trust of the larger NoCs, or designed specifically for the detection of invasive fault injection attacks. In the following, each of these approaches is briefly discussed, mainly to explore the parallels with our proposed scheme and show the targeted research gap.

The trustworthy computing on untrustworthy components or hardware techniques surveyed in chapter 2 are, in principle, implementable with untrustworthy communication interconnects. For example, the combination of vendor diversity, with component duplication, and task repetition and scheduling could be used for integrating two versions of the same communication bus from two different vendors.
and comparing the particularities of the transactions that go through them to infer any discrepancy and potentially detect abnormal behaviour by one of them. The technical feasibility of such techniques, however, does not compensate for the fact that they were not tailored for communication interconnects and hence their real-life implementation could incur large area and power overhead, and increased design complexity. The rule-based trustworthy computing technique introduced in \([261–263]\), was designed on a customisable and flexible basis, where the monitoring security wrappers are programmed to monitor and verify certain trust and security properties as pertinent to the monitored element. Nevertheless, the security wrappers in this technique are, out of necessity, designed with IP providers, which requires either complete trust in these providers or the implementation of additional measures to verify the trust of the security wrappers themselves.

When comparing with communication buses and interconnects, NoCs are new communication structures with a growing amount of literature focusing on their run-time trust. The run-time trust and integrity of transaction packets in NoCs was addressed in \([278]\), in which trusted processing and authentication units attached to the nodes in the NoC generate tags for packets and verify against them at each node. In \([279]\), a security-policy based approach similar to the generic scheme introduced in \([261–263]\) is proposed for run-time monitoring of NoC fabrics. In this approach, a centralized security engine is connected to the NOC for the global assertion and enforcement of security polices that are in turn implemented by individual security units connected to every router in the NoC.

In the sphere of dedicated run-time monitoring of communication buses and interconnects for the purpose of detecting or preventing hardware attacks, \([280]\) introduced a bus authentication and anti-probing architecture. This architecture addresses the realization that, most existing literature targets the protection of CPUs from tampering and probing attacks while leaving other SoC elements such as communication buses vulnerable to them. The architecture is based on the utilization of the unique impedance variations of transmission lines (\(i.e.\) buses and interconnects) in creating fingerprints for them. These fingerprints unpredictably vary based on the geometric and material characteristics of the different interconnects and buses and hence are unclonable and could be used in authenticating messages or transactions that pass through these buses and interconnects. Additionally, such fingerprints are sensitive to changes in electromagnetic fields and given that probing and fault injection attacks induce variations in this field, this technique could also be used in
detecting such attacks. In a similar approach, the work in [281] detects probing attacks aimed at the interception of data transmitted on buses through the detection of phase shifts exhibited by the outputs of these buses, as induced by impedance changes of buses, that are in turn caused by probing-induced changes in electromagnetic field. These techniques may deter fault injection attacks and detect their intrusion attempts, however, they are not applicable to other hardware trust issues that could also induce untrustworthy behaviour in communication interconnects.

Existing research is inadequate in addressing trust issues related to unexpected and abnormal behaviour of communication buses and interconnects at run-time. The scheme presented in this chapter targets this issue by proposing a run-time scheme for active detection of abnormal interconnect behaviour.

3.3 Background and threat model

In on-chip communication, some chip elements initiate transactions or data transfers and requests, and are known as source or master IPs. The elements receiving and responding to those requests are referred to as destination or slave IPs. For example, processors usually act as master IPs while peripherals are often slave IPs. Some IPs are also capable of both initiating transfers and responding to them and hence are considered hybrid master and slave IPs. An example of such IP is the direct memory access (DMA) element, which interacts with the processor as a slave and with the memory as a master. When using on-chip communication buses and interconnects in the conduction of on-chip communication between master and slave IPs, the interconnect receives read or write transaction requests from the output pins of the master IP and directs it to the input pins of the appropriate slave IP. The response or data is then transferred from the output pins of the slave via the interconnect to the input pins of the transaction-initiating master IP (or vice versa in the case of writing data to a slave).

Interconnects could be implemented for communication between one master and multiple slaves, multiple masters and one slave, or multiple masters and multiple slaves. In multiple-slave cases, decoders integrated inside the interconnect are used to decode addresses received from master IPs in order to send the associated data transfer request to the appropriate destination slave. In multi-master cases, arbiters are used to grant access to the interconnect to a certain master at a certain point of time based on pre-defined priority schemes. Common topologies for interconnects
facilitating multiple master to multiple slave communication are the crossbar topology, where data transfers between different masters and different slaves could occur concurrently and independently, and the shared access topology where only a single transaction or data transfer between any of the connected masters and slaves could be served at a time.

Transactions are typically performed in a number of steps or phases utilizing dedicated physical channels and wires. The particularities of these phases and those of the physical channels associated with them in addition to the temporal and spatial properties of the transactions and data transfers are specified by an agreed-on communication protocol used by the master IPs, slave IPs, and the communication interconnect. The Advanced eXtensible Interface (AXI) protocol is one of the most common and widespread communication protocols \[282\]. It was introduced by ARM and has been adopted by top technology companies and IP providers, such as Xilinx, as the main communication protocol in their SoCs. The fact that the AXI protocol is capable of supporting unaligned and out-of-order data transfers, in addition to allowing multiple outstanding transactions (i.e. from the same master) makes it suitable for implementation in numerous applications with different latency and bandwidth requirements.

The implementation of the AXI protocol requires 5 independent channels; read address channel, read data channel, write address channel, write data channel, and write response channel. Each of these channels consist of information-carrying signals, a VALID signal, and a READY signal. The VALID signal is asserted by the sender of the information to indicate the appearance of valid information on the information-carrying signal of the respective channel, whereas, the READY signal is asserted by the receiver of the information when it is ready to accept it. This two-way handshake mechanism is a requirement for the transfer of address, data, and control information in the AXI protocol. When using an AXI interconnect for facilitating communication, the interconnect acts as a slave for the master IPs, receiving requests from them and forwarding them to the slave IPs, which in turn view the interconnect as a master IP and send responses to it. These responses are then forwarded by the interconnect to the appropriate master. This convention, the associated architecture, and signals used in each of the five AXI channels are detailed in Figures 3.1 and 3.2. Currently two versions or specifications of the AXI protocol are in-use, the AXI3 specification and the AXI4 specification. The specifications are largely similar in data transfer mechanism, with the AXI4 specification capable of supporting larger data
3.3. Background and threat model

burst size\textsuperscript{1} and the transfer of additional information and control signals in the address channels. The work and discussions presented in this chapter are mainly concerned with the AXI3 specification, nevertheless, they are also applicable to the AXI4 specification with some minor modifications to accommodate primarily for the updated write data channel.

The threat model targeted in this work is that of a SoC that uses an untrustworthy AXI interconnect as the primary on-chip communication facilitator. This interconnect, therefore, connects multiple master IPs to multiple slave IPs, and allows multiple transactions to occur simultaneously and independently. The source of untrustworthiness of the interconnect is trivial to the proposed monitoring approach and could be a HT (the most likely source due to outsourcing or open-source interconnect), counterfeiting, system-level attack or even some unintentional source such as a hardware fault or degradation due to aging. Although the scheme presented in this chapter is mainly concerned with untrustworthy communication structures (interconnects and buses), we also consider a more realistic scenario where some other on-chip elements, due for example, to them being outsourced, may initiate some unexpected behaviour (\textit{i.e.} related to on-chip communication) on their own or in collusion with the untrustworthy interconnect. The proposed scheme therefore is concerned with the realistic scenario of SoCs consisting of components of mixed trust levels due to the majority of these components being outsourced, insufficiently tested for abnormal behaviour in the pre-deployment stages, or poorly protected against system-level and fault-injection attacks.

3.3.1 Targeted mistrust scenarios

The set of possible abnormal or unexpected scenarios within the specified threat model, and the corresponding security guarantees that should be provided to counter these scenarios, are presented in Table 3.1. The scenarios presented as abnormal behaviour by master and slave IPs are not meant to be a comprehensive list but rather as a demonstration of the fact that the trust of on-chip communication is multi-layered and cannot only be compromised by untrustworthy communication structures, but also by untrustworthy initiators (master IPs) and receivers (slave IPs) of transactions.

\textsuperscript{1}In AXI protocols, data transfers as a whole are referred to as bursts, the length of the burst, specified by the signals ARLEN or AWLEN, reflects the number of individual data transfers within the burst, which could be up to 16 beats in AXI3 or up to 256 beats in AXI4 (of a certain type). The signals ARSIZE and AWSIZE are used for specifying the number of bytes in each data beat.
Chapter 3. Run-time monitoring of untrustworthy on-chip communications

Figure 3.1: AXI3 write channels architecture (interconnect case) and signals

<table>
<thead>
<tr>
<th>Signal</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWID</td>
<td>Master</td>
<td>Transaction ID</td>
</tr>
<tr>
<td>AWADDR</td>
<td>Master</td>
<td>Address of first transfer in transaction</td>
</tr>
<tr>
<td>AWLEN, AWSIZE, AWBURST</td>
<td>Master</td>
<td>Number of transfers in transaction, number of bytes in each transfer for each transfer</td>
</tr>
<tr>
<td>AWLOCK, AWCACHE, AWPROT</td>
<td>Master</td>
<td>Lock, cache, and protection attributes of transaction</td>
</tr>
<tr>
<td>AWVALID</td>
<td>Master</td>
<td>Indicate valid address and control signals</td>
</tr>
<tr>
<td>AWREADY</td>
<td>Slave</td>
<td>Indicate ability to accept transaction</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Signal</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WID</td>
<td>Master</td>
<td>Transaction ID</td>
</tr>
<tr>
<td>WDATA</td>
<td>Master</td>
<td>Write data</td>
</tr>
<tr>
<td>WSTRB</td>
<td>Master</td>
<td>Strobe, indicates which byte lines hold valid data</td>
</tr>
<tr>
<td>WLAST</td>
<td>Master</td>
<td>Indicate last transfer in transaction</td>
</tr>
<tr>
<td>WVALID</td>
<td>Master</td>
<td>Indicate valid read data/ signals</td>
</tr>
<tr>
<td>WREADY</td>
<td>Slave</td>
<td>Indicate ability to accept write data</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Signal</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BID</td>
<td>Slave</td>
<td>Transaction ID</td>
</tr>
<tr>
<td>BRESP</td>
<td>Slave</td>
<td>Indicate status of transaction</td>
</tr>
<tr>
<td>BVALID</td>
<td>Slave</td>
<td>Indicate valid write response signals</td>
</tr>
<tr>
<td>BREADY</td>
<td>Master</td>
<td>Indicate ability to accept write response</td>
</tr>
</tbody>
</table>
3.3. Background and threat model

Figure 3.2: AXI3 read channels architecture (interconnect case) and signals

<table>
<thead>
<tr>
<th>Signal</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARID</td>
<td>Master</td>
<td>Transaction ID</td>
</tr>
<tr>
<td>ARADDR</td>
<td>Master</td>
<td>Address of first transfer in transaction</td>
</tr>
<tr>
<td>ARLEN, ARSIZE, ARBURST</td>
<td>Master</td>
<td>Number of transfers in transaction, number of bytes in each transfer, how address changes for each transfer</td>
</tr>
<tr>
<td>ARLOCK, ARCACHE, ARPROT</td>
<td>Master</td>
<td>Lock, cache, and protection attributes of transaction</td>
</tr>
<tr>
<td>ARVALID</td>
<td>Master</td>
<td>Indicate valid address and control signals</td>
</tr>
<tr>
<td>ARREADY</td>
<td>Slave</td>
<td>Indicate ability to accept transaction</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Signal</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RID</td>
<td>Slave</td>
<td>Transaction ID</td>
</tr>
<tr>
<td>RDATA</td>
<td>Slave</td>
<td>Read data</td>
</tr>
<tr>
<td>RRESP</td>
<td>Slave</td>
<td>Indicate status of transaction</td>
</tr>
<tr>
<td>RLAST</td>
<td>Slave</td>
<td>Indicate last transfer in transaction</td>
</tr>
<tr>
<td>RVALID</td>
<td>Slave</td>
<td>Indicate valid read data/ signals</td>
</tr>
<tr>
<td>RREADY</td>
<td>Master</td>
<td>Indicate ability to accept read data</td>
</tr>
</tbody>
</table>
## Chapter 3. Run-time monitoring of untrustworthy on-chip communications

### Threat model

<table>
<thead>
<tr>
<th>Threat model</th>
<th>Unexpected behaviour</th>
<th>Required security guarantee</th>
</tr>
</thead>
<tbody>
<tr>
<td>Untrustworthy AXI interconnect</td>
<td>Data diversion In a SoC where IP A and IP B communicate using an interconnect, it can be guaranteed that the interconnect cannot divert any packets between IP A and IP B to a third IP, C.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data modification and injection In a SoC where IP A and IP B communicate using an interconnect, it can be guaranteed that the data sent by IP A will be received as is by IP B, and that the interconnect cannot inject data to IP B presenting it as incoming from IP A.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Shadowing attack[2] In a SoC with an interconnect, it can be guaranteed that the interconnect cannot initiate transactions on its own.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Illegitimate access to resources In a SoC where IP A (master IP) and IP B (slave IP) communicate using an interconnect, it can be guaranteed that the only slave IP address/es accessed by the interconnect is/are the one requested by the master IP.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DoS[3] of flooding type In a SoC with an interconnect, it can be guaranteed that the interconnect cannot initiate transactions on its own.</td>
<td></td>
</tr>
<tr>
<td>Untrustworthy IP</td>
<td>Illegitimate access to resources (master IPs) A given IP in the SoC cannot use the interconnect to access a resource that it does not have an access right to (access control).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Shadowing attack (slave IPs) It can be guaranteed that a slave IP cannot send data without having received a request from one of the master IPs.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DoS of flooding type (slave IP) It can be guaranteed that a slave IP cannot send data without having received a request from one of the master IPs.</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.1: Targeted mistrust scenarios and countering security guarantees
3.4 Principles, approach, and architecture

It could be concluded, by inspecting Table 3.1, that the root cause behind any of the presented unexpected behaviour scenarios are the implicit capabilities of an AXI interconnect or of master and slave IPs to initiate unexpected transactions, modify data when in-transit, or access other on-chip IPs unrestrictedly. Evidently, these unrightful capabilities, or even privileges, from the perspective of an attacker, are a result of the absence of any limitations on the actions and interactions of those IPs. This is fundamentally a consequence of the common IC design and verification practice of focusing on what a given IP should do while not considering what it should not be capable of. We identify this absence of limitations and restrictions as a vulnerability that must be mitigated at run-time by imposing constraints that restrict the actions of a 3PIP to only those that satisfy its functionality and comply with its specification.

For example, an AXI interconnect is a communication bus architecture that, by definition, is a communication facilitator that should not be able to initiate transactions or process data (i.e. modify it in some way). Therefore, any “last line of defence” techniques must ensure that the interconnect remains within these boundaries at run-time. This imposition of boundaries must also prevent slave IPs from sending data without being involved in outstanding transactions and enforce access control on master IPs. A straightforward way of imposing such limitations might be to request their integration in 3PIPs by the IP vendors themselves, and then to verify the limitations in pre-deployment verification tests. However, this approach would place an unreasonable requirement on IP providers that may not be justified, especially when the IP buying party is a small or medium-scale design house. Additionally, the adoption of this approach would also defeat the purpose of allowing IC designers and SoC integrators the freedom of integrating state-of-art commercial-off-the-shelf (COTS) IPs in their systems.

In our approach to run-time monitoring of untrustworthy on-chip communication, we devise and implement an appropriate set of run-time constraints on the interconnect and the 3PIPs connected to it, that can provide the security guarantees defined in Table 3.1. This approach stems from the realization that in order to detect and/or prevent any unexpected behaviour at run-time, one does not look for this

\[2\text{In a shadowing attack, the interconnect keeps a copy of the data that it has transmitted at some point for a later time where it can leak it out of the system}\]

\[3\text{Denial-of-service}\]
unexpected behaviour directly, given the associated unpredictability, but rather for a violation of expected behaviour, which is well-known and defined. We therefore base our technique on evidence-backed verification of expected actions and behaviour at all times. Any deviation from this expectation is consequently identified and flagged as unexpected and hence untrustworthy behaviour regardless of its cause being malicious or non-malicious. The mechanism of applying this approach for run-time trust monitoring is based on combining the guiding principles of component guarding, event verification, and data tagging.

### 3.4.1 Guiding principles

The guiding principles underlying the development and implementation of the proposed run-time monitoring and trustworthy computing scheme are detailed in the following:

- **Component guarding**: monitoring, logging, and keeping track of the progress of every transaction that passes through the interface of the untrustworthy 3PIP.

- **Event verification**: ensuring that the events occurring at the interface of the untrustworthy 3PIP are within its functional and access boundaries and are not unexpected.

- **Data tagging**: using unique tags and IDs to identify the transmitted data and confirm its integrity.

In addition to these guiding principles, the proposed scheme adopts the known concept of foundry diversity, in which the main functional IPs in the SoC are fabricated in a separate foundry from the proposed trust and security monitoring elements. This practice minimizes the possibilities of malicious collusion between the monitored element and the monitoring elements. A particular collusion of concern is in the form of the monitoring element, due to modifications implemented in the foundry, defeating its own purpose and hiding or obscuring the unexpected behaviour of the monitored element. It is therefore advisable that the monitoring elements are entirely designed and fabricated in-house or in a local foundry that is ruled by local jurisdiction and could be closely monitored. The foundry diversity requirement is a very common approach for securing on-chip security elements, and is adopted and advised by a large number of publications on hardware security, and therefore, is not
considered a limiting factor. Moreover, the proposed security elements are built to be implemented in a “plug and play” manner by the basic IC designer or SoC integrator with minimal hardware security knowledge, in an attempt to achieve smooth and easy integration of trust-verifying components in SoCs of different sizes and applications.

### 3.4.2 Approach

The proposed communication interconnect run-time monitoring scheme consists of security elements that act as wrappers around the interconnect and the master and slave IPs connected to it. The wrappers are responsible of imposing the three guiding principles on on-chip communication, as needed to achieve the desired trust and security guarantees and as appropriate to the nature of the “wrapped” IP, being an interconnect, a slave IP, or a master IP. The design and implementation of the wrappers require no knowledge of the internal architecture of the “wrapped” IPs as they are designed to sit at the interface of these IPs, and base all of their monitoring and unexpected behaviour detection on the signals and events observed at this interface. This allows the integration of the proposed wrappers by the IC designer or SoC integrator independently at design or build-time without the intervention or involvement of untrustworthy IP providers. The mechanism in which each of the guiding principles is physically implemented largely depends on the associated behaviour. For example, to prevent data modification by the interconnect, data tagging for the purpose of authentication and integrity is implemented, where each of the wrappers (of master and slave IPs) is assigned a unique ID that is used to identify both the wrapper and the data issued by it (i.e. by the respective “wrapped” IP). Another example is the use of logs at both interfaces of the interconnect (i.e. the interface at the master IP’s side and the interface at the slave IP’s side) to keep track of transactions transmitted via the interconnect and to correlate the actions of the interconnect with the actions of the IPs connected to it. By doing so, we achieve component guarding and prevent the initiation of unexpected and unsolicited transactions. Therefore, as fitting to the trust and security considerations of the three different types of monitored elements (communication bus or interconnect, master IP, slave IP), three types of wrappers were designed. On the operational level, these wrappers work with each other to enforce the security guarantees presented in Table 3.1.

The following is a list of the main security functionalities that are designed into the
wrappers as needed to address the unexpected behaviour scenarios from Table 3.1.

- The functionalities of the wrapper of the interconnect:
  - Authentication and access control: prevent the interconnect from accessing any IPs or addresses that it has not been rightfully (i.e., by a legitimate master IP) asked to access.
  - Data integrity check: confirm that information and data received by the interconnect is transmitted as is.
  - Prevention of shadowing attacks: prevent the interconnect from initiating transactions and transmissions on its own.
  - Prevention of data diversion: prevent the interconnect from redirecting read or write requests or data to IPs other than their original sender or receiver.
  - Prevention of DoS attacks: verify the actions of the interconnect against the actions of the IPs connected to it.

- The functionalities of the wrappers of master IPs:
  - Authentication: identify the wrapped IP with a unique ID and use this ID to identify and tag information and data originating from this IP.
  - Access control: prevent the IP from accessing slave IPs or addresses that it is not supposed to access.

- The functionalities of the wrappers of slave IPs:
  - Authentication: identify the wrapped IP with a unique ID and use this ID to identify and tag information and data originating from this IP.
  - Prevention of DoS attacks and access control: prevent the IP from sending data without having received a request from one of the master IPs.

3.4.3 Architecture

The architecture of the proposed scheme consists of the security wrappers that are connected to the input and output signals (i.e., the signals used in the AXI communication) of the wrapped elements, and dedicated wrapper communication.

\[4\] In the current version of the scheme, this access control is static and based on a pre-defined list of slave IPs and addresses that a given master IP is allowed to access.
channels that run in parallel with the AXI protocol channels and are used for communication between the wrappers. An example of a SoC with \( n \) master IPs and \( m \) slave IPs is shown in Figure 3.3. In this figure, and in the remainder of this chapter, the wrapper of the AXI interconnect is denoted as W(AXI) whereas the wrappers of the master IP \( M_i \) and slave IP \( S_j \) are denoted W(M\(_i\)) and W(S\(_j\)), respectively.

![Figure 3.3: The deployment of the proposed scheme and wrappers in a SoC](image)

As can be seen in Figure 3.3, the wrapper for each IP receives all signals that appear on the AXI channels of that IP, allowing it to monitor and verify these signals before forwarding them to their destination. Therefore, a transaction initiated by a certain master is first received by the wrapper of this master, which sends it, after performing relevant checks, to the wrapper of the interconnect, which in turn performs further checks before sending the transaction to the interconnect. When the transaction leaves the interconnect, it goes to its wrapper again, which performs correlation and verification checks before forwarding the transaction to the wrapper of the slave IP and the slave IP itself. This progressive multi-level trust verification ensures the trust and security of signals at all stages of the communication. Additionally, as the wrappers do not require any additional direct communication with the wrapped elements, other than that of the AXI protocol, the wrapped elements are oblivious to the fact that their transactions are being checked. This is expected to prove useful in, for instance, detecting IPs infected with HTs of long-term malicious effect hence stealthy enough to evade regular HT detection tests.
3.4.3.1 Design of security wrappers

The wrapper of the AXI interconnect (W(AXI)) contains master and slave interface units U(Mi) and U(Sj) responsible for receiving, verifying and forwarding the AXI signals for each master and slave, respectively. Each interface unit implements state machines to monitor and implement the trust and security functionalities or guarantees concerned with the signals of the five AXI channels; Address read, Data read, Address write, Data write and Write response. W(AXI) also contains a collection of low-capacity storage elements or memories used in logging key information related to the transactions that pass through the interconnect, which are, in effect, the same transactions that are monitored by the master and slave units (U(Mi) and U(Sj)). The logs denoted TR_log are referred to as transaction logs and are used for tracking transactions and keeping record of their information, mainly, at the address exchange stage of transactions. The logs used to keep track of transmitted data are denoted DATA_log. The sizes of both types of logs are configurable at design time to support variations in the number of simultaneous transactions that a given interconnect is capable of handling or the number of concurrent transactions that the IC designer is concerned with monitoring. Furthermore, the logs are designed and arranged to allow simultaneous access from master and slave interface units, allowing transactions to proceed through the AXI network without reduction in throughput. This is achieved by designing each log as a group of sub-logs, where each sub-log is dedicated for transactions between each master-slave pair (or for the master-slave pairs that are allowed to communicate with each other).

For each transaction, the TR_log includes the transaction ID, the address, the transaction length, and status bits to indicate the state of the transaction; new transaction, at address-exchange stage, at data-transmission stage, or completed. Each log in DATA_log maintains a data tag for each data beat, along with its number or count (i.e. based on the order of this beat in the data transfer) and the transaction ID. The tag is used for integrity verification during transmission given that it is a function of the data value and the unique ID of the source of this data (the ID of the slave IP in the case of a read transaction and that of the master IP in the case of write transaction). The number or count logged for each data beat, however, is used to keep track of the number of beats sent by the source IP at any given point in time, and detect data injection by the interconnect. In general, logging occurs on the source side of the transaction or data beat, and is verified on the destination side, by the respective
interface units in W(AXI). The interface units use log management units, implemented for each log in W(AXI), to oversee and perform the logging and fetching of information from the logs whenever needed. This happens when one of the interface units has information to log or needs some information from the logs in order to verify a request or data that has been received. The structure and contents of each of the logs is shown in Figure 3.4, in which X, Y, W, and Z represent the configurable sizes of each of the logs.

![Figure 3.4: W(AXI) transaction and data logs](image)

An access control unit and a data tag generation unit are the main components of wrappers of master IPs. The access control unit is concerned with preventing the wrapped IP from issuing a read or write request to a slave address that it is not allowed to access, or issuing a write request to a slave that it is only allowed and expected to read from, or vice versa. The purpose of the data tag generation unit is generating tags for the data beats sent by the master IP when in write data mode. The same data tag generation units are also implemented in the wrappers of slave IPs for generating tags associated with read data. The tags are a function of the uniquely
assigned ID of the source of the data (i.e. hard-coded in the wrappers) and the data beat. The tags and counts of data beats are sent from the wrapper of the source IP (i.e. slave in the case of read, master in the case of write) to the W(AXI) via the dedicated wrapper communication channels that run in parallel with the AXI channels. In W(AXI), these tags and counts are saved in logs by respective interface and log management units to be later verified against at the data recipient side of the AXI wrapper. The wrappers respond to any inferred attack by blocking it and/or asserting an alert signal. The alert signal could be later used by a root of trust within the SoC, or by a higher level component in the system, for example to deduce when to isolate an offending 3PIP from the rest of the system (e.g. after a certain number of alerts from its wrapper). However, even if such high-level mitigation and recovery is not implemented, the wrappers themselves are capable of preventing a number of unexpected behaviours of concern.

3.4.4 The life cycle of transactions in the proposed architecture

Figure 3.5 and Figure 3.6 show the life cycle of a read transaction when the proposed scheme is implemented. Details about the exact steps of monitoring and verification shown in these figures are detailed below. Write transactions also follow the same life cycle with the addition of the write response checks that happen at the end of every write transaction, in the same way of logging on the source side and verifying on the destination side.

![Figure 3.5: The lifecycle of a transaction in the Address Read stage](image-url)
3.4. Principles, approach, and architecture

Figure 3.6: The lifecycle of a transaction in the Data Read stage

- Read address run-time monitoring and verification steps:

1. $M_i$ asserts the $ARVALID$ signal and places an address and transaction ID on $ARADDR$ and $ARID$ lines, respectively, along with the other transaction specification signals ($ARLEN$, $ARSIZE$, $ARLOCK$, etc).

2. $W(M_i)$, upon receipt of these signals, checks if the requested address is one that this particular master is allowed to access and perform the requested operation. In the case of a pass, the signals are forwarded to $W(AXI)$, otherwise the request is prevented from proceeding and an alert signal is asserted.

3. The signals are then received by $U(M_i)$ in $W(AXI)$. This unit decodes the address to identify the destination $S_j$ and saves (through the log management unit) the request information in the relevant transaction log. In the log, the status bits ‘00’ are assigned to this log entry to indicate that it is a new transaction.

4. At some point in the future, this request is expected to appear on one of the slave-side output pins of the interconnect. At this point, the respective $U(S_j)$ communicates with the log management units inquiring if there are any logged “new transactions” with the details it has received from the interconnect. If identical information is found, this means that the
interconnect forwarded the request to its rightful and original receiving \( S_j \), in which case, the request is forwarded normally. Otherwise, the request is prevented from proceeding and an alert signal is asserted.

5. The request is then received by \( W(S_j) \), however, no further checks are needed, hence the request passes through the wrapper to \( S_j \) directly. For \( W(S_j) \) to keep track of the open-transactions that \( S_j \) is involved in, it logs the transaction ID in an internal log.

6. As for the ARREADY signal, in the proposed architecture, it is tied to an internal signal in \( W(AXI) \) as in, whenever a certain slave IP asserts its ARREADY, the respective \( U(S_j) \) asserts the corresponding internal signal. These internal signals (\( i.e. \) one for each slave IP) are monitored by the interface units in the master-side of \( W(AXI) \) to verify if the slave IP of concern (for the current transaction or request) is actually ready whenever they receive an asserted ARREADY signal from the interconnect.

• Read data run-time monitoring and verification steps:

1. The mechanism for checking if a \( M_i \) is ready for receiving data is performed in the same way as checking if a \( S_j \) is ready for receiving requests, via internal signals in the interconnect wrapper connected to the RREADY signals of the master IPs. The state of these signals is checked by the slave interface units when a given unit receives RREADY from the interconnect.

2. When the \( S_j \) asserts the RVALID signals and puts the first data beat on its RDATA line, \( W(S_j) \) generates a tag for this data and assigns it a count (based on its order in the transfer). The tag and count are then sent to \( W(AXI) \) simultaneously with the other AXI data read signals. In the current implementation, the tag is generated from a basic arithmetic operation (\( i.e. \) addition and subtraction) involving bits 0 to 7 and bits 22 to 29 (assuming each beat is at least 4 bytes-long) of the data beat along with the unique 8-bit ID used by \( W(S_j) \) to identify data departing from \( S_j \).
3. The respective U(S_j) forwards the data channel signals to the interconnect and logs the data tag, count, and transaction ID in the respective log (via the log management unit). An additional bit in the log is dedicated for the “last-bit”, which holds “0” unless RLAST (from S_j) is asserted.

4. When the interconnect asserts RVALID, and places data with a certain ID on the interface of one of the master IPs, the respective U(M_i) searches for this ID in the logs, if not found, an alert is raised, otherwise, the information saved in logs is used to answer the following questions:

   (a) Is this an expected data beat belonging to an active transaction and was sent by the slave but not yet received by the master (vice versa in the case of a write transaction)? W(AXI) answers this question by inspecting the status bits, the count it had earlier received from W(S_j) and an internal count that it keeps for the beats that pass through it (for each active transaction) to the master IP side.

   (b) Is this genuine data sent by the legitimate slave? U(M_i), after checking the logs and identifying the source of the data, generates a tag for it with the same process used for tag generation in W(S_j). The tags are then compared, if found identical, it is inferred that the interconnect had not modified the data or copied it from another transaction.

   (c) In case RLAST is asserted, is this actually the last data beat as indicated by the respective slave? This question is answered by checking the value of the “last-bit” in the data logs. If it is not “1”, then the slave has not yet sent the last beat of data and therefore, the interconnect is trying to inject data. The “last-bit” could also be used in detecting protocol violation by the interconnect if it tries to end a transaction prematurely (useful in case protocol checkers are not implemented).

5. Alongside the progression of the transaction, the status bits pertaining to this transaction are updated to ‘01’ when the first beat of data is received
by $M_i$, and changed to ‘10’ to indicate that data transfer for this transaction is currently in progress. Finally, they are changed to ‘11’ when the last data beat is received to indicate that this transaction is complete.

The lifecycle of a write transaction in the proposed architecture is identical to the lifecycle of a read transaction, but takes into consideration that in write transactions the data source is the master IP. Therefore, the tag and count generation for data beats is performed at the master IP side and verified at the slave IP side. As for the write response stage, however, additional checks are performed by $W(AXI)$ to ensure that the response forwarded to $M_i$ from the interconnect is actually the same as the one sent by the relevant $S_j$.

### 3.5 Implementation

The steps of transaction monitoring and verification throughout the life-cycle of a transaction, as detailed in section 3.4.4 and as needed to achieve the desired security guarantees formed the basis of the RTL design of each of the wrappers and that of the interaction (i.e. communication channels and wires) between them. The design was written in Verilog and validated and simulated in Vivado Webpack 2020 [283](the algorithms are provided in appendix A.1). The wrappers were first designed and tested individually in test benches and then integrated together in a modelled SoC. This SoC included an open-source, multi-master, multi-slave AXI3 interconnect [284], Xilinx AXI traffic generators [285] to model master IPs and initiate transactions, and Xilinx block memory generators and block memory controllers [286, 287] to model slave IPs. A block diagram of the architecture used to implement and test the wrappers is shown in Figure 3.7.

#### 3.5.1 Performance and results

To prove the concept of the proposed communication interconnect monitoring, the block diagram in Figure 3.7 was tested only by simulation and it was simulated under both normal conditions and unexpected behaviour conditions where the scenarios presented in Table 3.1 were induced in the SoC. In the case of normal operation, the wrappers, as desired, do not obstruct the penetration of signals in the system, and all AXI signals leaving a given source reach their destination intact. However, due to the trust and security checks performed by the wrappers, a fixed amount of delay or
3.5. Implementation

Figure 3.7: Block diagram of the modelled SoC

Latency is added to all transactions. This added latency (i.e. additional number of cycles needed for information to go from source, to interconnect, to destination) is three clock cycles for each of address read, data read, address write, and data write, and two clock cycles for write response. As the SoC was simulated and synthesized with a 100MHZ clock frequency, the three clock cycle latency is, in effect, a latency of 30 nano-second for each information transfer. This additional latency may not be favorable in latency-critical systems. However, the level of security and protection that the wrappers provide, while being fully independent from any untrustworthy third party, is of high importance in security and safety-critical systems. Additionally, the wrappers are designed to, in the case of unexpected or abnormal behaviour, terminate transactions and/or raise certain alerts as soon as one of the checks fails, which might happen in the wrapper of the source IP or that of the interconnect. This allows the intervention of a root of trust or high-level components or software in the system in correcting any abnormal behaviour at the earliest possible chance. This eliminates unwanted delays that would occur if the wrappers were designed to take action only at the destination of information.

\[5\text{The targeted FPGA device is ZYNQ-7 ZC702 evaluation board (xc7z020clg484-1)}\]
Upon synthesizing the simulated SoC, it was found that the wrappers consumed 17% of the total power consumption of the SoC, which is non-trivial, however, poses less power overhead comparing to, for example, redundancy-based approaches where multiple copies of the same IP are integrated in the SoC. Although the obtained power figure of 17% is approximate given that it was obtained solely from simulation and without actual implementation in hardware, it provides an indication of the power cost of the adoption of the proposed wrappers in a given SoC. Additionally, it is expected that the power cost of the wrappers would be less significant in case of implementing the scheme in real-life SoCs with full-fledged master and slave IPs due to the fact that the architecture and the checks performed by the wrappers are consistent irrespective of the complexity and architecture of the SoC in which they are implemented. The multiplicity in the implementation of logs in W(AXI) allows simultaneous access to these logs, and hence rules out any negative serious effects on bandwidth and throughput. Moreover, as the wrappers monitor all 3PIP transactions, the opportunities for IPs to maliciously collude (using their AXI interfaces) are eliminated, obviating the need for IP vendor diversity and minimizing a design-time overhead.

Table 3.2 presents the unexpected behaviour scenarios that were induced in the modelled SoC, the way in which they were implemented or realized, and the respective simulation-time response of the wrappers. All of the scenarios were detected, blocked, and the dedicated alert signals were asserted by the wrappers. By blocking the serious scenarios where a violation of normal behaviour is evident, the scheme achieves dynamic and timely response to untrustworthy behaviours in addition to detecting their occurrence. This paves the way for achieving the ultimate goal of holistic and complete trustworthy computing on untrustworthy hardware.

\footnote{In actual implementation of the scenarios pertaining to slave IPs, as the modelled slave IPs are proprietary memory blocks that cannot be modified, small slave IP-emulating blocks were designed and used to induce the untrustworthy behaviour of concern, and consequently the response of the wrappers was observed.}
<table>
<thead>
<tr>
<th>Unexpected behaviour</th>
<th>Implementation</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data diversion (interconnect)</td>
<td>Switching the “select” signal of the data read or write multiplexers in the AXI interconnect</td>
<td>Detected by the interface units in W(AXI) due to incompatibility with logged information and blocked.</td>
</tr>
<tr>
<td>Data modification (interconnect)</td>
<td>Modifying interconnect output data as in output data = input data +1.</td>
<td>Detected by the destination interface unit in W(AXI) because of mismatch between data tags, and blocked.</td>
</tr>
<tr>
<td>Shadowing attack (interconnect)</td>
<td>Modifying the interconnect to issue a read or write request without having received one</td>
<td>Detected by the destination U(Sj) in W(AXI) due to non-existence of the request in the logs, and blocked.</td>
</tr>
<tr>
<td>Illegitimate access to resources (interconnect)</td>
<td>Modifying the interconnect to make it change the address of a request before forwarding it</td>
<td>Detected by the destination U(Sj) in W(AXI) due to incompatibility with logged information (i.e. address), and blocked.</td>
</tr>
<tr>
<td>Flooding DoS (interconnect)</td>
<td>Modifying the interconnect to repeat a shadowing or illegitimate access to resources attack</td>
<td>Detected in the same way as a shadowing or an illegitimate access attack.</td>
</tr>
<tr>
<td>Illegitimate access to resources (master IP)</td>
<td>Making a Mi send a request to read from or write to an address or Sj it is not allowed to access</td>
<td>Detected by W(Mi) as the requested address is out of the allowed range, and blocked.</td>
</tr>
</tbody>
</table>
Chapter 3. Run-time monitoring of untrustworthy on-chip communications

<table>
<thead>
<tr>
<th>Shadowing attack (slave IP)</th>
<th>Making a $S_j$ send data without having received any request</th>
<th>Detected by $W(S_j)$ because of the nonexistence of an active transaction with the specified ID in the logs, and blocked.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data flooding DoS (slave IP)</td>
<td>Making a $S_j$ send data (repeatedly) without having received any request</td>
<td>Detected by $W(S_j)$ if it is not involved in an active transaction with the specified ID, or detected by $W(AXI)$ if the ID belongs to a valid but already completed transaction, and blocked.</td>
</tr>
</tbody>
</table>

Table 3.2: The simulated unexpected or abnormal behaviours and the live verified responses of the wrappers

3.6 Evaluation

The run-time monitoring and abnormal behaviour detection scheme presented in this chapter addresses a serious hardware trust issue that has been so far overlooked in literature; an untrustworthy on-chip communication interconnect exhibiting unexpected and abnormal behaviour at run-time, compromising the security of the SoC and potentially leading to errors, faults, and malfunctions. The scheme satisfies the objectives of adaptability, independence, autonomy, and computational lightweightness outlined in chapter 1. It is adaptable due to the fact that its guiding principles of component guarding, event verification, and data tagging are universally applicable and not tied to a certain communication protocol or topology. The scheme, although tested on multi-master multi-slave interconnect, could be easily adapted to the shared access type of interconnects by changing the number of logs and interface units in the wrapper of the interconnect to fit a single-master, multi-slave or multi-master, single-slave case. Additionally, it could also be implemented in a partial manner, to perform its checks only occasionally, on certain types of transactions, or on transactions between certain master IPs and certain slave IPs. This is especially useful in the case of SoCs with stringent latency, power, or area constraints. The design of the wrappers is also unit-based and decentralized and hence is adaptable to perform additional trust and security checks, if needed, and to verify if the interconnect abides by certain pre-defined specifications such as the priority levels of
master IPs.

The design of the security wrappers could also be extended to perform protocol-adherence checks, eliminating hence, the need for protocol checkers. The proposed security wrappers, although dependent, in their design particularities, on the communication protocol and interconnect type and features, are independent of the internal architecture of the monitored elements (i.e. internal registers, wires, multiplexers, arbiters, and decoders) and their run-time monitoring and detection is fully based on the signals at the interface of the interconnect (and of the connected master and slave IPs). This type of monitoring, in addition to simplifying the integration of the monitoring elements, serves the fundamental objective of complete independence from the untrustworthy designers and providers of on-chip IPs, achieving hence autonomy in trust and security monitoring and verification. The scheme, due to its adoption of log-based monitoring and event correlation in the verification of behaviour, is computationally lightweight as the only actual computation is that needed for tag generation, which could be as simple as a single xor gate (though a more complex computation is recommended for increased resilience).

### 3.6.1 Limitations and potential improvements

The main limitation to the implementation of the scheme is that any changes whether to accommodate different protocols or perform further checks need to be decided beforehand and integrated in the design of the wrappers. The proposed scheme, is, in principle, designed to be adaptable to different communication protocols, however, the internal architecture of the wrappers requires modification if a protocol other than AXI3 is addressed. This is a consequence of the security wrappers being hard-wired and not based on programmable elements. The decision to implement the proposed scheme with hard-wired elements was taken to minimize their exposure to attacks initiated from the software-level. However, it comes at the cost of the wrappers needing design-level modifications in case a different communication protocol is adopted or in case further trust and security checks are needed. The changes, however, need not be radical and pose a significant design overhead. For example, in the case of AXI4 protocol, two main modifications to the current implementation of the scheme are needed. One is related to supporting transfers that are larger in size and carry more control information while the other is related to the fact that AXI4
does not support data write interleaving. As for the additional control signals in the protocol, the IC designer or SoC integrator may decide to ignore them and pass them normally to the interconnect or may decide to log them and verify against them at the destination side. The lack of support of data write interleaving in the AXI4 protocol could be addressed by modifying the wrappers to always assign data appearing on the write data channel (of the master IPs) to the first outstanding transaction instead of linking it to the transaction with the same ID, as is the case with the current AXI3-based implementation of the wrappers. The size of the data bursts is generally irrelevant in the current scheme since data is passed as is and as the tag generation and verification processes are performed per beat.

An assumption related to transaction and data IDs and that was adopted in the testing and simulation of the scheme, is that the interconnect forwards a given transaction ID (in the case of baseline healthy operation) as is without modification. This is a realistic assumption for many real-life interconnects. However, some interconnects are designed to add unique identifiers to the transaction (and data) IDs they receive from master IPs before forwarding them to slave IPs. If the interconnect of concern is of this type, and if the SoC integrator does not have prior knowledge about the identifiers, then the IC designer or SoC integrator would need to spend some time in the pre-deployment simulations mapping between the transaction IDs at the master IPs side and those appearing on the slave IPs side of the interconnect. This mapping would reveal the unique identifiers used by the interconnect, after which, the interconnect wrapper could be modified to take these identifiers into account when performing its logging and verification. As the unique identifiers assigned by this type of interconnects are usually not random, fixed (throughout the operating life of the SoC), and unique to each master IP, the mapping process should be straightforward.

The unique IDs assigned to the wrappers of master and slave IPs\footnote{These IDs are scheme-related and not to be confused with transaction and data IDs that are a part of the AXI protocol} and used to identify wrappers and generate data tags are, in the current design of wrappers, pre-defined and hard-coded. This approach was adopted as a proof-of-concept and is suitable for small lightweight SoCs such as those used in IoT and portable biomedical devices as it would allow the integration of the scheme without imposing large overhead associated with the generation of unique IDs. Nevertheless, such IDs are vulnerable to compromise and interception by attackers. To this end, one could follow on the footsteps of cryptography key generation by utilizing randomness in generating
the unique IDs of the wrappers. A relatively lightweight method to generate unclonable and unpredictable IDs is the use of physical unclonable functions (PUFs). One way to utilize PUFs in generating unique IDs for the wrappers is by using the response of a PUF to a certain challenge as a seed for ID generation. In this case, at every start-up of the device, the PUF is provided with a challenge and the response is broadcasted to all wrappers, each wrapper would then use some unique pre-defined function in generating its own ID, using the PUF response, and then would share it with the wrapper of the interconnect using the dedicated wrappers communication channels. For even more enhanced security, the IDs could be encrypted before sending them to the wrapper of the interconnect. Another PUF-based method could also base the ID of each wrapper on a different response of the PUF. The tag generation process, in the current proof-of-concept implementation of the scheme, is a basic arithmetic operation. For a higher level of security, a more complex or multi-level tag generation process could be adopted as appropriate to the needs and capabilities of the SoC of concern. Nevertheless, regardless of the adopted tag generation mechanism, unique tag generation practices such as the proposed use of PUFs, the use of long tags, and unique functions for tag generation (i.e. for each master and slave IP) should be sought in accordance with the computing capabilities and overhead allowance of the SoC of concern. This is mainly to avoid the risk of easy tag prediction (by malicious parties) and the risk of tag collision, where tags belonging to different data are calculated to have the same value, defeating hence the purpose of data tagging.

Potential improvements to the scheme that would widen its scope and enhance its effectiveness in detecting untrustworthy behaviour at run-time include integrating it with the electromagnetic field based detection of fault injection attacks on interconnects, reviewed in section 3.2, and with latency-based denial of service detection techniques (for slave IPs) such as the one introduced in the next chapter. Furthermore, error correcting techniques could be used in active and dynamic correction of errors, whether introduced intentionally or unintentionally by the interconnect, in the read or write data, once detected by the wrapper of the interconnect in the tag verification process. In systems where component redundancy is implemented, a certain threshold could be specified for the number of alerts asserted by a given wrapper in a predefined time-frame. Once this threshold is reached, the system would switch the operation to the duplicate component. The scheme could also be used in identifying untrusted IP providers and hence ceasing
any deals with them. As the proposed wrappers also target the detection of a number of abnormal scenarios by master and slave IPs, further improvements could focus on widening their scope and enhancing their features. For example, the access control units implemented in the wrappers of master IPs could be upgraded to be dynamic and updated regularly to reflect potential changes in access rules throughout the operation of the SoC.

3.7 Conclusion

The work presented in this chapter addressed, for the first time, the possibility of integrating untrustworthy on-chip communication interconnects in SoCs, and devised an architecture-level scheme for the run-time monitoring of interconnects and the consequent detection of unexpected, abnormal, and undesired behaviour. The scheme bases its monitoring on the signals appearing at the interface of the monitored element hence avoiding the involvement of the untrustworthy IP providers in the monitoring and trust verification process. The conceptual basis of the framework is that limitations and restrictions should be imposed on on-chip components to limit their actions and interactions in the SoC to only what is expected of them and what is needed to perform their specified functionality. As the main purpose of an on-chip communication interconnect is facilitating communication between the different IPs in the SoC, it should therefore be actively restricted to this functionality and never allowed to initiate transactions on its own or perform modifications on transactions and data transfers. This conceptual basis is implemented via the guiding principles of component guarding, in which the transactions passing through the interconnect (and the IPs connected to it) are continuously monitored and logged, event verification, where the actions of the interconnect are correlated with the actions of slave and master IPs connected to it, and data tagging, in which uniquely generated tags are used to identify transmitted data and verify its correctness. These guiding principles are implemented on the architecture level by security wrappers or elements that sit at the AXI interface of the monitored elements.

The proposed wrappers perform multi-level verification of transactions and monitor them throughout the whole transmission process, from the initiation of the read or write request to the receipt of the last data beat and response in the transaction by the recipient. The current implementation of the wrappers detects and responds (by blocking and/or raising alerts) to a different range of untrustworthy behaviours
including data diversion, data injection, and data modification. The wrappers are currently also capable of enforcing basic access control on master IPs and block slave IPs from initiating actions that could utilize the interconnect in leaking out information or might lead to a denial of service by flooding it with transactions. Nevertheless, due to the modular design of the wrappers, capabilities to perform additional trust and security checks and detect other behaviours or actions of concern, could be added if needed. The wrappers were tested in a modelled SoC with an AXI3 interconnect, traffic generators acting as master IPs, and memories acting as slave IPs. All of the targeted untrustworthy behaviour scenarios were successfully detected by the wrappers and blocked.

In this chapter, section 3.1 introduced the motivation behind targeting untrustworthy on-chip communication interconnects, and section 3.2 presented a review on published works that are concerned with run-time monitoring of untrustworthy on-chip communication structures. Section 3.3 presented a background on the operation of communication interconnects, the unexpected behaviour scenarios that could be induced by an untrustworthy interconnect, and the security guarantees that need to be implemented to stop the consequences of such behaviours in their tracks before they lead to any serious run-time trust and security issues. The proposed scheme for the monitoring of run-time trust of on-chip interconnects and the IPs connected to them and the associated approach, principles, and architecture are detailed in section 3.4. The experimental implementation and verification of the proposed scheme on a modelled SoC with an AXI3 communication interconnect is presented in section 3.5. A post-implementation evaluation of the scheme and a discussion of potential improvements are presented in section 3.6.
Chapter 4

Run-time detection of obscured latency extensions

4.1 Introduction

Denial-of-service (DoS) is a terminology used to describe the case when a system or part of it are, mostly intentionally and through malicious means, made inaccessible or unresponsive. In the world of information security, DoS attacks are mostly initiated over the network with the main target of overflooding the target system with requests and traffic, rendering it unavailable for dealing with actual legitimate traffic and hence, in effect, denying its service/s to its legitimate users. A DoS attack could lead to total paralysis of the system and ultimately its shut down leading in turn to serious and catastrophic consequences, especially, if the attacked system is used in critical infrastructure or in applications directly tied to human lives.

Network and software-level DoS attacks have been occurring since at least the late 1990s [288], and a countless number of real-life DoS attacks has been reported over the years. DoS is mostly viewed as a network security or information security issue and rarely viewed from the perspective of hardware security, especially on the SoC and IP level. The operation of the heart of modern electronics, SoCs and ICs, is based on communication and networking between the different components that reside on the SoC. Therefore, it is possible for attackers to resort to initiating a DoS attack on the system using a bottom-up approach, by disrupting or delaying certain on-chip processing and communication, or by flooding the communication channels of one or more of the SoC components, rather than going for the more obvious top-down approach of a network-based DoS attack.
In this chapter, we introduce a framework for tackling the issue of a DoS attack when carried out by one of the components in a SoC. Instead of focusing on the more commonly discussed, flooding leading to unavailability type of DoS attacks, we recognize and analyze a type of DoS that is based on slow and steady deprivation, in which an on-chip component delays the processing of transactions beyond the nominal, yet never beyond the maximum processing time. This type of DoS is not of sudden paralyzing nature and is rather meant to introduce either small possibly distant and intermittent delays to degrade the performance of the SoC over time, or small continuous delays to maliciously maximize the response time of SoC elements and disrupt their availability. Due to its nature, we deduce that this type of DoS may pass unnoticed, enjoying obscurity by other operations in the SoC, and hence is capable of posing great risk to the healthy operation of the SoC. Consequently, we experimentally verify these deductions and propose a framework for the detection of such obscured DoS at run-time.

### 4.1.1 Motivation

Availability has been long identified as one of the pillars of cyber security, along with confidentiality and integrity. It is also one of the security properties detailed in chapter 2 and that, in case it is compromised in a given system, the system is deemed untrustworthy. The importance of availability stems from the fact that the healthy and normal operation of a given system requires its availability to authorized users and processes whenever it is needed to perform and fulfill its purpose and functionality. Moreover, unavailability or DoS in hardware, which is based on flooding, where the victim component is overwhelmed with requests, or starvation, where on-chip components are deprived from the resources and inputs needed for their operation, or a combination of both, fall within the range of unexpected behaviour that is outwith the normal operation of an on-chip component. Therefore, a DoS, whether in the form of an arranged attack, for instance, via a HT or system-level attack, or in the form of an unintentional degradation in performance due, for example, to aging, is a compromise of the run-time trust and security of hardware. Therefore, its signs and symptoms should be monitored and mitigated at run-time, in addition to potential pre-deployment countermeasures.

A denial of service in hardware has been primarily discussed in literature from the perspective of a DoS HT. The utilization of a HT by malicious actors in initiating an
on-chip DoS attack has been theoretically discussed since the very first publications on HTs, and denial-of-service or operational delay has been identified in HT taxonomies as one of the main potential effects or payloads (i.e. malicious actions) of HTs [65, 74, 75, 289]. A more recent publication on DoS HTs show the possibility of implementing a HT to inject faults in packets traveling between certain nodes in an NoC. These faults in turn would trigger the error correction mechanism of the NoC resulting in repeated transmissions of packets and leading to deadlocks in the NoC and potentially, chip failure [290]. A HT that resides in a CAN (controller area network) controller (attached to one of the nodes connected to the CAN bus) and injects several errors in the CAN bus for the purpose of eventual disconnection of the effected node from the CAN architecture, denying hence the service of this node, is demonstrated in [291]. Several models of DoS HTs were also introduced in the Trust-Hub hardware Trojan benchmark [65, 77]; examples include a shift-register based HT that, through its rotation, increases the power consumption of the SoC leading to early exhaustion of system resources and consequent DoS, and a HT that, upon activation, induces a DoS by reducing the clock frequency of a certain part of the circuit [65, 77]. In [292], a DoS attack in which an FPGA is utilized in injecting faults leading to a DoS in a CPU residing on the same SoC is introduced. In [293] a micro-architectural level attack that exploits the shared cache in initiating a DoS through the issuing of concurrent cache-misses is presented. This wide range of potential realizations of DoS attacks show that, in hardware, the DoS attack vector is complex, capable of taking many shapes and forms, and could be traced back to a number of hardware trust issues including HTs, fault injection attacks, and architectural level attacks. Therefore, it could be concluded that a hardware DoS attack involves unpredictability related to the realization, manifestation, and mechanism of the attack.

In addition to the unpredictability related to the realization and nature of DoS attacks, the vulnerability of SoCs and electronic systems to DoS and availability attacks goes back to their physical, structural, and operational nature. Almost all SoCs are composed of sequential processes, where the output of one process or component is crucial for the operation of the next process and hence a delay in one process leads to availability issues in all dependent processes or components. Additionally, semiconductors are known to suffer from performance issues with aging, a trait that could be utilized by malicious actors in inducing early-aging and early-unavailability or failure of the system. Moreover, many on-chip components are
not consistent in the processing time of transactions, with the processing of some transactions taking longer than others. In this case, the other elements in the SoC do not have fixed expectations on when a response should be received from those components, a fact that may be used by an adversary in introducing unnoticeable and obscured delays to the operation of the SoC. The unpredictability of DoS attacks combined with the architectural and operational vulnerability of hardware to such attacks dictate the need to address hardware DoS attacks continuously throughout the life-time of the SoC, through run-time monitoring and live detection of DoS attempts.

4.1.2 Contribution and chapter outline

In this chapter, the hardware trust and security issue of a latency-extension or delay based DoS is presented and discussed. A latency-extending DoS could be caused by a HT, an aging or poorly cloned counterfeit IP, or an architectural or system-level attack. As this type of DoS aims to degrade the performance of the SoC by extending the latency of on-chip processes and responses without causing their total halt or disruption, it could, in principle, also result from non-malicious causes such as an invisible hardware fault, or the system reaching its end-of-life. In order to analyze the possibility of a latency-extending DoS passing unnoticed in real-life scenarios, we develop a hypothesis that such a type of DoS, if originating from an on-chip component that is originally, due to operational, architectural, or physical reasons, variable in operating latency, could be obscured by the normal and baseline operation of this component. The component that this hypothesis was tested on is the dynamic random access memory (DRAM). The remainder of this chapter introduces experimental evidence of the validity of the proposed hypothesis, and background on the variability associated with DRAM operation. This is followed by the introduction of the proposed run-time latency extension detection framework, its implementation, and its evaluation. The machine-learning based run-time framework for the detection of latency-extending DoS in hardware, that is presented in this chapter, was published in:

4.2 Background

Dynamic random access memory or DRAM is a ubiquitous type of volatile memory commonly used for temporary storage of code and data needed by processors to operate and function. DRAMs consist of DRAM storage cells, where each cell consists of a transistor and a capacitor and is capable of storing a single bit. The capacitor is charged or discharged through the switching of the transistor to store bit ‘1’ or ‘0’, respectively. As capacitors are known to lose or leak charge over time, DRAM cells are continuously refreshed or injected with charge to rewrite data in them and ensure data retention. DRAM memory cells are arranged in grid-like patterns, referred to as banks, where the address of each cell is defined by the intersection of its row and column. In DRAMs, all cells in a row are connected to a single read/write bus and accessing any of these cells, for either read or write, is performed on the basis of reading an entire row, but accessing a single cell. When a request to access a certain DRAM address (*i.e.* a certain cell defined by a certain row and column) is received, the address is decoded and the contents of all DRAM cells in the corresponding row are loaded in a dedicated row buffer or sense amplifier. Based on the decoded address, the value of the bit stored in the cell of concern is then placed on the output pins, in the case of a read, whereas, in the case of a write, the value of the cell of concern in updated in the row buffer. The process of loading row contents into the row buffer is a destructive process that erases the contents of the DRAM cells in the relevant row, and therefore, even if a write was not performed (*i.e.* no update to cell contents), the contents of the row buffer need to be re-loaded to the relevant row, in what is referred to as the pre-charging of DRAM memory cells. The most commonly used type of DRAMs is the Double Data Rate (DDR) DRAM, which deploys additional buffers for the pre-fetching of multiple words of data from memory cells allowing memory accesses on both the rising and falling edges of the clock. In spite of the fact that the relatively simple structure of DRAMs allows dense storage of large amount of data, this structure is governed by physical and operational limitations that reflect, rather negatively, on the predictability of DRAM access latency. As DRAM plays the crucial role of the main memory in many systems, and as its access latency is variable and unpredictable, it was chosen as the main focus for studying the possibility of obscuring delay-based DoS attacks by the naturally variable processing latency of some electronic elements, and for consequently developing and testing the proposed DoS detection framework.
4.2.1 Why is DRAM latency variable and unpredictable?

DRAMs suffer from the issue of variable and inconsistent response time to memory accesses. This could hinder the ability to determine the exact speed of a DRAM-deploying system and may result in performance issues. However, it is an inevitable issue originating from the physical nature and operational basis of DRAMs. The physical reasons for the variability in DRAM latency are related to the fact that DRAMs use capacitors for the storage of data. The natural time-related leakage of charge stored in capacitors is addressed by frequent refreshes, however, what complicates the issue furthermore is that the amount and rate of this charge leakage is neither consistent among all capacitors in a given a DRAM nor it is consistent for an individual capacitor throughout its lifetime. The former issue has left DRAM designers with two main options, either deciding on when to perform a refresh, or the refresh interval, based on the minimum data retention time of all the DRAM cells, or adopting a multi-rate refresh period where individual rows are refreshed independently. In multi-rate refresh approaches, a row is refreshed when, for example, signs of data retention failure are detected in the particular row, accounting hence for the variable retention time of the different cells and rows [294]. Each of these options has its drawbacks as the first could add unnecessary delay in the DRAM processing of requests while the latter multiplies the unpredictability in DRAM latency, given that the refresh mechanism is not performed in a consistent manner. The retention rate of a single DRAM capacitor being not fixed throughout its lifetime is identified by the variable retention rate (VRT) phenomenon. The VRT phenomenon might not have direct short-term effect on DRAM latency, however, on the long-term and due to this phenomenon, the need to update the adopted refresh mechanism and its frequency might arise while the system is in operation, in response to a change in surrounding environmental, electrical, and mechanical conditions [295, 296]. The down-scaling of transistors is also another factor, contributing, to an increase in fluctuations of the retention rate of DRAM cells [297].

The operational reasons behind the variability in DRAM latency come from the fact that a DRAM access is performed in a number of steps, ranging from one step to three steps, depending on the type of access and the state of the DRAM at the time of access. The three steps are the activation of the requested memory row, in which the data in the specified row is loaded into the row buffer or sense amplifier, the actual reading from or writing to the loaded row, and the pre-charging of the row, where its contents
are re-loaded (after updating them in the case of write) into it from the row buffer or sense amplifier. A memory access or transaction\[^1\] destined to the same row and bank as the previous transaction or access does not need the activation or pre-charge steps, and is hence implemented in one step. This type of memory access is referred to as open-row access. In closed-row access, the current memory access is to a pre-charged row in another bank, therefore, both of the activation and the read or write steps are needed. In conflict-row access, where the current memory access is destined for a different row in the same bank as the previous access, all three steps would be needed to close or pre-charge the row previously loaded into the buffer, load the contents of the new row in the buffer, and consequently perform the read or write. In addition to the variability introduced by the varying number of steps needed to perform a read or a write, switching from a read in one memory access cycle to a write in the next cycle, or vice versa, results in an additional delay of up to 18 cycles (\[^i.e.\] in DRR3 DRAMs), due to the DRAM buses being bidirectional with the same wires accommodating both read and write memory accesses \[^298\].

### 4.2.2 What has been done to address DRAM latency variability?

The crucial role that DRAMs play in the vast majority of electronic systems puts their performance metrics in the spotlight of research. The variability and unpredictability of DRAM latency has been long identified as a design and performance issue given the role of latency in determining the speed of any system and in satisfying stringent timing constraints of speed-critical systems. In applications such as military, medicine, and aviation, the slightest variation in the worst case execution time could lead to an irreversible failure and the most catastrophic consequences. As the physical reasons behind the variable retention rate of DRAM cells are tied to inherent properties of semiconductors, they cannot be completely eliminated. Therefore, the mechanisms developed to cope with the variable latency of DRAMs can be classified into two categories. The first category is concerned with design and architectural enhancements in DRAMs and their controllers with the aim of limiting the operational variability of DRAM access latency. The second category focuses on modelling the retention rate of DRAM cells in order to be able to anticipate when refreshes should be performed or when a data retention failure might be observed. The two categories are not necessarily complementary and might even contradict each other.

\[^1\]In this chapter, the terms ‘transaction’ and ‘memory access’ are used interchangeably.
other in the case where it is deduced, from the modelling of the retention rate of DRAM cells of a certain DRAM, that, for example, the refresh rate should not be consistent for all of the DRAM cells and throughout the operation of the DRAM, introducing hence further variability in DRAM latency.

In the first category of coping mechanisms, techniques based on bundling of memory accesses based on their type (read or write) to reduce the bus direction switching [299] [300], and techniques based on the partitioning and dynamic allocation of DRAM to reduce the number of closed and conflict-row accesses were proposed [301] [302]. Some publications also introduced new DRAM architectures with less architectural susceptibility to operational-based unpredictability, when comparing with the ubiquitous DDR DRAMs [298] [303] [304]. In the second category, a number of publications focused on characterizing and modelling the retention rate of DRAM cells to be able to propose suitable refresh mechanisms that are not, inefficiently, based on the worst case retention rate. One example is an experimental study performed over 200 DRAM chips from major DRAM manufacturers, which led to the realization that DRAMs could be divided into regions based on retention rate, and that different refresh intervals could be adopted for the different regions [305]. Another example is an analytical study on the amount of charge leakage in different DRAM cells and the introduction of the possibility of performing low-latency partial refreshes for the cells with slow rate of charge leakage [306].

The unpredictability of DRAM latency is an issue that has not yet been resolved, given that the different factors involved in it are, mostly, outwith the control of the IC designer. Limiting the operational vulnerability of DRAMs has been attempted by a large number of publications. However, the majority of the proposed approaches are based on new architectures and/or designs of DRAMs and their controllers, and require the re-write of software to match the proposed memory allocation mechanisms, hence could only be applied in new and future SoCs that are not yet in circulation. Additionally, many of the proposed DRAM access scheduling or bundling techniques improve predictability at the cost of area and therefore, are not suitable for implementation in systems with a tight area budget. Adapting with the reality of variable DRAM latency, and analyzing the potential implications of this variability on the run-time operation of DRAMs and SoCs is, therefore, of utmost importance. The performance implications of this variability have received notable coverage in literature and the techniques surveyed in this section provide an overview of how researchers and IC designers are dealing with the impact of unpredictable
DRAM latency and variable retention rate on DRAM performance. Nevertheless, to the best of our knowledge, the trust and security implications of this variability, and the possibility of utilizing it in realizing malicious intents, were never before analyzed and addressed.

4.2.3 The vulnerability to obscured latency extensions

In order to analyze the trust and security implications of DRAM variability and the possibility of it posing a threat to the healthy, trusted, and secure operation of DRAMs, we view this variability from a trust and security perspective. Given that the DRAM latency variability or unpredictability is inherent to its operational and physical characteristics, it cannot be considered a violation of trustworthy or expected behaviour as DRAMs are not, in the first place, expected to respond to memory accesses in a fixed non-variant amount of time. Nevertheless, a question that arises here is; what if a malicious actor introduces subtle and indistinct additional variations or extensions to the already-variable DRAM latency, moving this latency to the sphere of unexpected and untrustworthy behaviour while maintaining stealth, obscurity, and the impression of normal expected behaviour? For the purpose of investigating the possibility of the occurrence of such an attack in real-life scenarios, the following hypothesis was formed, and experimentally tested and verified; The unpredictability and variability in DRAM latency is a security vulnerability that could be used in masking or obscuring a latency-extending DoS or performance-degradation attack, in which the latency of some or all DRAM accesses is extended beyond the nominal latency, but not beyond upper-bound (maximum) latency.

To test this hypothesis, we simulated an open-source DDR3 DRAM with workloads from the PolyBench benchmark suite, and observed and recorded the individual DRAM access latency of around 2-million memory accesses. The latency is measured from the time an access request is received by the DRAM controller to the point in time where the corresponding response leaves the DRAM controller. The simulation was performed under normal baseline conditions where no modifications were introduced to the DRAM or its controller, and under DoS conditions, where the DRAM controller was modified to add an additional delay to the processing of transactions. The delay was introduced at the point of forwarding memory accesses or transactions from the arbiter to the scheduler (inside the DRAM controller), and/ or just before a transaction is released from the DRAM controller,
emulating potential scenarios of latency-extending DoS on either read or write memory accesses, or on both of them. The simulated latency-extending DoS scenarios are detailed in Table 4.1, while Figure 4.1 shows the values of the latencies obtained from both the baseline and the DoS simulations. In addition to showing the variability of DRAM latency, Figure 4.1 clearly shows that latency experienced by most transactions is less than 300 ns, regardless of the transaction coming from a normal operating scenario or a DoS scenario. Furthermore, the observed maximum baseline latency of 464.5 ns was, in around 2-million transactions, exceeded only twice by the DoS scenarios B and C, and never exceeded by the DoS scenarios A, D, and E. These experimental findings support the presented hypothesis and show that DRAM latency variability, in addition to being a performance nuisance, is a security vulnerability, that could be exploited by latency-extending DoS or performance-degradation attacks with effects hidden in plain sight.

Table 4.1: The implemented “DoS” modifications

<table>
<thead>
<tr>
<th>DoS scenario</th>
<th>Forward</th>
<th>Release</th>
<th>Read</th>
<th>Write</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>DoS scenario A</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>11.25 ns</td>
</tr>
<tr>
<td>DoS scenario B</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>11.25 ns</td>
</tr>
<tr>
<td>DoS scenario C</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>11.25 ns</td>
</tr>
<tr>
<td>DoS scenario D</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>5 ns</td>
</tr>
<tr>
<td>DoS scenario E</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>1.25 ns</td>
</tr>
</tbody>
</table>

Figure 4.1: The DRAM access latencies for the baseline and the DoS simulations
4.3 Approach and methodology

The experimental verification of the possibility of obscuring latency-extending actions in the inherently variable operation of some electronics elements requires the development of corresponding detection and countering mechanisms. In the case of DRAMs, the root causes of latency variability and unpredictability are, at least partially, unpreventable and cannot be simply eliminated at design-time. The development of run-time monitoring techniques to detect obscured latency-extending and performance-degrading actions is, therefore, of utmost importance and would act as the last line of defence against unexpected, undesired, and malicious latency extensions. In order to satisfy the objectives laid out in chapter [1] of adaptability, independence, autonomy, and computational lightweightness, we base our approach to run-time detection of latency-extensions on the characterization of the stream of memory access requests observed at the interface of the DRAM controller. In particular, our approach to run-time detection of obscured latency-extensions is based on the realization that although the exact latency of individual DRAM accesses cannot be predicted, our knowledge of the basis of variability in DRAM latency could be used in establishing predictive latency models for each of the different types of DRAM accesses.

One of the factors of DRAM latency variability, the read/write bus re-direction, is tied to the type of a given transaction or memory access and the type of the preceding transaction or memory access. The row and bank of a given memory access in comparison with those of the preceding memory access dictate whether the DRAM access will be an open-row, closed-row, or conflict-row access, hence giving an indication of the number of steps performed (inside the DRAM) for realizing this access. An additional latency-determining factor, that is universal to most electronic elements, is the load or number of outstanding transactions waiting to be processed by the element of concern. The larger the dynamic depth of the queue of outstanding transactions, the greater the processing delay and latency experienced by incoming requests or transactions. It could be concluded, therefore, that regardless of the implemented refresh mechanism and the internal architecture and design particularities of the DRAM and its controller, the latency of a DRAM access at a certain point of time is directly influenced by a set of factors. These factors are, namely, the DRAM workload, the type and attributes of the transaction to be performed, and the type and attributes of preceding transactions. The influence of
these factors, cannot be used in deterministic prediction of DRAM latency at a given point of time, however, it could be used in modelling the DRAM access as a discrete function of a small number of dynamic access features. This allows the categorization of memory accesses and the utilization of machine-learning in temporal and feature-based modelling of the latency ranges of the different categories.

Using machine learning techniques in modelling the behaviour of DRAMs was proposed in a number of previous works, albeit from the perspective of error detection and DRAM failure prediction at run-time. In [311], an ensemble machine-learning model is proposed for the purpose of predicting future errors in DRAMs. This model uses metrics, such as number of instructions per second, bandwidth and temperature, which are collected by sensors, and correlates them with historical information of DRAM failures to indicate potential future DRAM errors. The use of classifiers in predicting the time of occurrence of a future DRAM failure based on the time of occurrence of the first DRAM error and the rate of subsequent errors is proposed in [312]. A machine-learning based DRAM error predictor that takes the workload into consideration in addition to DRAM parameters and temperature was proposed in [313]. This predictor is based on analyzing more than 240 features of DRAMs and their environment, and using this analysis in constructing machine-learning based DRAM error behaviour models. Although these techniques are similar in their machine-learning based approach to the framework presented in this chapter, they were not designed with trust and security in mind. Additionally, they are largely based on knowledge about internal features of DRAMs and historical information about their behaviour, a type of knowledge that is very specific to the architecture and model of a given DRAM and is not expected to be readily available in the case of outsourced DRAMs and DRAM controllers.

### 4.3.1 DRAM transaction categorization

Grouping or categorizing DRAM accesses based on the type and attributes of current and preceding transactions is based on a well-defined notion that is directly tied to the multi-step operation of DRAMs. However, workload based variation in DRAM latency needs further investigation to establish knowledge on the impact of longer DRAM queues on DRAM latency. To this end, the load-latency relationship was verified by putting the simulated open-source DRAM under varying workload conditions, derived from different applications of the PolyBench benchmark. The heat map presented in
Figure 4.2 shows this relationship in load, presented as the number of outstanding transactions, versus latency experienced by the respective transactions. The general trend is clear; the longer the queue of memory access requests, the greater the latency of processing incoming requests. Nevertheless, as load is not the sole determinant of latency, some overlapping between the different latency ranges could be observed.

The DRAM access categorization convention adopted in this work is defined by the terminology of $T_cT_pRL$, where $T_c$ is the type of the current transaction, read or write, $T_p$ is the type of the preceding transaction, $R$ is the current row situation, whether the same as the last-accessed row or different, and $L$ is for the load or the number of outstanding transactions at the DRAM interface. With this terminology, a read transaction coming after another read transaction, targeting the same DRAM row, and received at a time when the number of outstanding transactions is 2, is denoted as $RRS2$. While a write transaction after a read transaction to different rows and received at a time with no other transactions waiting to be processed is denoted as $WRD0$.

In this work, the focus is on the best and worst cases of DRAM operational latency, which are, respectively, associated with open-row (one step access) and conflict-row (three step access) accesses. Therefore, closed-row accesses were not exclusively addressed and the performed simulations and tests are concerned with memory accesses to the same DRAM bank. Nevertheless, the proposed run-time monitoring and detection technique and the underlying categorization and modelling could be easily extended to model closed-row accesses by adding the bank to the list.
of DRAM access categorization features. The adopted categorization results in 32 categories or groups of memory accesses, as there are four different possibilities for the types of current and preceding transactions (RR, RW, WW, WR), two different options for the accessed row (same or different) and, in the simulated case, four different loading conditions (from 0 to 3 outstanding transactions). The type and row situation result in eight categories that are uniform across all types of DRAMs. The load, however, is an orthogonal feature, specific to the maximum allowed DRAM queue depth in a given system. Although occurrences of up to 6 outstanding transactions were observed in the simulated system, they were rare and distant, hence it was decided to limit the presented analysis and machine-learning modelling to 3 outstanding transactions, reflecting the most common observation, and without loss of generality.

The transactions obtained from the simulations were sorted into the 32 different categories, and naturally, some categories had more transactions than others, reflecting the variations in the memory access needs of the different workload applications. The latencies of the different transaction categories were then analyzed. Each of the different categories was found to exhibit a specific latency distribution with unique shape and distinctive peaks providing hence, experimental evidence of the possibility of modelling the latency of DRAM based on the proposed categorization. Additionally, most of the distributions were found to show two distinct peaks, a larger one occurring at a lower latency value, concluded to be the nominal latency for that category, and a smaller one occurring at higher latency values, deduced to be the result of a transaction (from the inspected category) coinciding with an internal refresh cycle.

4.3.2 The machine learning algorithm

The aim of the proposed run-time monitoring and latency-extension detection technique is detecting the occurrence of unusual latency variations and extensions rather than verifying or predicting the latency of individual transactions. Therefore, as noticeable differences were found when examining the latency distributions of the different categories of memory accesses, we base our run-time latency-extension detection technique on statistical and probability distribution machine-learning methods. To this end, trusted baseline latency distributions of the different memory access categories are obtained during offline trusted operation, either
before-deployment or after periodic verification tests, modelled, and used as a reference for comparing with run-time latency distributions, hence, leading to the detection of anomalies in the run-time latency distributions, if any. Using latency distribution as the basis of comparison and anomaly-detection rather than focusing on individual latency values, allows the detection of shifts and unexpected trends or patterns in the latency distribution in addition to the detection of unusual and unexpected extensions in latency values.

In order to accurately and distinctively model the latency distributions of the different memory access categories, a machine-learning algorithm that is capable of recognizing and modelling the defining characteristics and features, such as range, density, and location of peaks, of each of the categories had to be chosen. Kernel density estimation (KDE) was chosen given that it is a non-parametric estimation with the ability of providing probability distribution approximations for datasets with multiple peaks and non-defined distributions. KDE allows adaptive fitting and unique modelling to variant distributions, making it suitable for both the construction of reference models for baseline trusted latency distributions, and for the modelling of run-time untrustworthy latency distributions. Furthermore, the adoption of a versatile and flexible machine-learning algorithm such as KDE enhances the universality of the framework and ensures its suitability for adaptation in run-time latency monitoring of electronic elements with significant variations in processing latency.

For the purpose of verifying the viability of using KDE in modelling different latency distributions, the DRAM was simulated with more than 20 million transactions, derived from the PolyBench benchmark applications. The latency data of these transactions is referred to in the remaining of this chapter as the latency dataset. The latency dataset was divided into a training set and an evaluation set, with the former being around 80% of the dataset and the latter the remaining 20%. Both the training and evaluation datasets were sorted into the 32 memory access categories, and individually modelled using a Top-hat KDE algorithm with a bandwidth of 0.5. This bandwidth was chosen so that the resultant latency distribution estimation and modelling is neither too detailed and granular, where the minor features of the distribution are needlessly emphasized on, nor too simplified and smooth, where the distinct shape of the distribution is ambiguous and unrecognizable. The training dataset represents trusted baseline latency used for reference modelling, whereas, the evaluation dataset represents run-time latency data. Figures 4.3 to 4.6 show the KDE models of the training-data sets and those of the testing datasets for the RR and RW
memory access categories. Through visual inspection, it is evident that the KDEs of the training dataset and the evaluation dataset of the same memory access category adopt a similar distribution, even though they belong to temporally distant transactions. This finding supports the proposed modelling and comparison approach, and shows its feasibility in characterizing the variable latencies of DRAMs and in comparing two distributions belonging to the same DRAM access category but from different instances of operation.

Figure 4.3: Training KDEs of RR memory accesses

Figure 4.4: Evaluation KDEs of RR memory accesses

Figure 4.5: Training KDEs of RW memory accesses

Figure 4.6: Evaluation KDEs of RW memory accesses
4.4 The framework

The proposed run-time monitoring and latency-extension detection framework consists of a pre-deployment part and a post-deployment part. The first step in the framework is performed before deployment and is concerned with collecting DRAM access latency information under varying loading and access conditions. The obtained latencies are then categorized, modelled using KDE algorithms, and the resultant models are saved for future utilization in the run-time detection of anomalous latency trends and patterns. As these models are used as references of trusted baseline latency distributions, they should be based on latency measurements obtained from trusted operation of the DRAM, for example, from pre-deployment validation tests, from some relevant trusted historical latency data, or from periodic trusted tests of DRAM operation. Once the DRAM is activated in field, information about the types and addresses of DRAM accesses, the time taken to process them by the DRAM, and the dynamic DRAM load is collected and used in measuring the run-time latencies that correspond to the different access categories. On a pre-defined periodic basis, the run-time latencies are modelled using KDE and compared with the respective pre-saved baseline reference models.

Visual inspection could be used to identify the distinct latency distributions of the different categories and inspect similarities and differences between baseline and run-time KDEs. However, for an automated comparison between the baseline and the run-time latency distributions and systematic detection of extensions or anomalies in run-time latencies, quantitative comparison methods need to be adopted. As the comparison is performed between two density distributions or estimations, the Jensen-Shannon distance metric was chosen. The Jensen-Shannon (J-S) distance is a statistical metric that is the square-root of the Jensen-Shannon divergence, which in turn is derived from the Kullback-Leibler divergence \[314-316\]. The formulas of the Jensen-Shannon divergence and the Kullback-Leibler divergence are provided in formulas \[4.1\] and \[4.2\] respectively, with \(V\) denoting the first distribution, \(W\) denoting the other, and \(M\) denoting the vector mean of both. The Kullback-Leibler divergence quantifies the divergence of one probability distribution from another using the sum of probabilities in the former and the log of probabilities of both. When the log is taken to base 2, the range of the J-S distance is from 0 to 1, with 0 being the J-S distance for identical distributions, and 1 being the J-S distance for totally distinct ones. This somewhat limited range makes the J-S distance suitable for providing an
indication of the amount of discrepancy between two probability distributions, without being overly and needlessly sensitive to small variations in them. This is a desired feature in our framework as it is concerned with detecting anomalies in an originally variable metric.

\[
D_{KL}(V, W) = \sum_{x \in X} V(x) \log \frac{V(x)}{W(x)}
\]  

(4.1)

\[
D_{JS}(V, W) = \frac{1}{2} D_{KL}(V, M) + \frac{1}{2} D_{KL}(W, M)
\]  

(4.2)

For experimental assessment of the feasibility of using the J-S distance in measuring the difference between the baseline and the run-time KDE latency models, the J-S distance was calculated for each of the pairs of KDE models belonging to the training and evaluation datasets of the 20 most populated memory access categories (i.e. in the conducted simulations). The resultant J-S distances are shown in Figure 4.7, as can be seen, all distances are below 0.4 with the vast majority being less than 0.3 indicating that the compared distributions have similar attributes, reflecting the fact that both the training and evaluation datasets, in this case, were obtained from trusted operation of the DRAM. As one of the objectives of the framework is adaptability and as it is concerned with detecting anomalies in a naturally variant metric, it was designed to allow flexible interpretations of the range of J-S distances that are considered to reflect trusted operation. This mainly depends on the application, its security and performance needs, and the extent to which it could tolerate unsolicited latency variations. For instance, in a speed-critical system, a relatively small J-S distance of 0.3 could be adopted as a threshold that, if crossed during any of the periodic latency comparisons, alerts are raised and further investigations and tests are initiated. The use of such strict thresholds is especially feasible in systems employing component redundancy and duplication as once a component is found to show some untrustworthy behaviour, even if subtle, it could be disabled (and replaced) without affecting the operation of the whole system. Nevertheless, in another system where speed performance is not a priority and where, for example, intervening to test the DRAM or to investigate the reasons behind the abnormal latencies cannot be easily performed at run-time, alerts may only be raised when the J-S distances are found to repeatedly exist in the top-third of the J-S distance range.
4.4. The framework

Figure 4.7: The J-S distances between the training (baseline) KDEs and the evaluation (run-time) KDEs

4.4.1 Implementation

The framework was tested solely by simulation as outlined in Section 4.6 to prove the concept of the proposed monitoring and show its feasibility and effectiveness in detecting obscured latency extensions, however, Figure 4.8 (and the remainder of this section) shows the implementation proposed for deploying this monitoring in real-life electronic systems. After the post-silicon but pre-deployment collection, categorization, modelling, and saving of trusted latency data, the steps of data collection, sorting, modelling, and finally analysis and comparison are to be performed at run-time. The run-time data collection and sorting shall be performed continuously based on the memory accesses appearing at the DRAM interface, whereas the modelling and analysis steps are performed periodically as defined by the IC designer or SoC integrator. For efficiency and lightweightness purposes, we propose a hybrid software/hardware implementation of the framework. The live data collection and sorting is performed in hardware, preferably in an independent wrapper at the interface of the DRAM but possibly inside the DRAM controller, while the remaining machine-learning and computational steps are performed in a software module embedded within the operating system of the SoC.

The live data collection step is concerned with gathering and recording the information needed for the categorization of the DRAM accesses and for determining their respective latencies. This information is, namely, access start time, access end
time, target row, target bank, access type, and number of preceding outstanding accesses. The collection or recording of this information would be performed with the assistance of on-chip logs or small storage elements, a log of size \( N+1 \) would be needed for monitoring a DRAM with a maximum allowed queue depth of \( N \). The sorting of the recorded access information based on the defined memory access categories is performed with the assistance of a set of logic gates, mainly, to calculate latency (access end time - access start time) and conditional logic to sort the resultant latencies in the respective memory access category bin. For each of the memory access categories (i.e. RRS0, RRRS1, RRD0, WRS1, and so on), bins, realized as counters, are assigned to record the number of occurrences of latencies in specified latency ranges. This type of sorting and recording is based on the quantization of the full expected range of latency values into a number of sub-ranges and assigning a counter for each range, in addition to an additional counter for latencies falling outwith the defined range (i.e. latencies exceeding upper case latency). The adoption of this sorting convention allows the characterization and modelling of the different latency ranges without imposing large unnecessary overhead on hardware. An example is a certain memory access category with an expected latency range between 20 ns and 40 ns, depending on the system capabilities and overhead limits, this latency range could be counted into 10 bins or 100 bins (excluding the out-of-range bin). In the 10-bins case, the step size would be 2 ns with the first bin counting latency occurrences between 20 ns and 22 ns, the second counting those between 22 ns and 24 ns, and so on. While, in the 100-bins case, occurrences of latencies between 20 ns and 20.2 ns are counted in the first bin, latencies between 20.2 ns and 20.4 ns in the second bin, and so on. The 100-bins case would, therefore, result in more granular sorting of latencies, which would be reflected on the accuracy of the KDE modelling and on consequent comparison and detection of latency extensions. Nevertheless, this would be at the cost of relatively large hardware overhead. The described data collection and sorting is, in effect, ‘histogramming’ of the run-time latencies of the different memory access categories, with the number of bins or counters representing how condensed the histogramming is.

Every certain number of operation cycles, with this number depending on the expected frequency of DRAM accesses, the dynamic range of the counters or bins, and the trust and security requirements of the system (i.e. how often the latency should be verified), the contents of the counters are sampled by the dedicated software module and used to generate KDE models corresponding to live
untrustworthy latency data. This is followed by the analysis and comparison step, in which the J-S distances between the saved baseline KDEs and the run-time KDEs are calculated. The obtained J-S distances are then projected on pre-defined system or application-specific thresholds to determine if they are considered to indicate anomalous latency extensions or variations and if further security and trust tests or measures need to be performed or implemented. Generally speaking and regardless of the specific application, the greater the J-S distance, the greater the discrepancy between the baseline trusted latency and the run-time latency, and the greater the certainty of the occurrence of a latency-extending DoS in the previous operation cycle.

The implementation of the framework is designed to be fully independent of the refresh mechanism, the architecture, and the internal design of the DRAM and its controller. This implementation, in addition to achieving the desired independence from untrustworthy IP providers, allows the universal application of the framework regardless of the model and design particularities of the monitored DRAM. The flexibility in the categorization or ‘histogramming’ process also allows the implementation of the framework in systems of various hardware overhead allowance.

4.5 Experimental verification: DoS detection

The discussion and results presented in section 4.3 provide an experimental verification of the feasibility of using machine-learning methods in characterizing DRAM latencies after categorizing them based on the memory access type and the loading conditions. Furthermore, section 4.4 presents a tested statistical method for quantitatively comparing between the modelled trusted latencies and the modelled run-time latencies. These tests show the viability of implementing the proposed framework in real-life systems to monitor the processing latency of electronic elements with naturally variable and unpredictable latency. However, for a comprehensive assessment of the framework and its ability to detect abnormal latency extensions, it was tested under the cases of abnormal yet obscured latency extensions presented in section 4.2 and detailed in Table 4.1.

To test the framework under latency-extension scenarios, the latency data obtained from simulating the DoS scenarios in Table 4.1 was used as the anomalous run-time latency dataset. Therefore, this dataset was categorized according to the identified 32 memory access categories, and modelled using the same KDE algorithms that were
Chapter 4. Run-time detection of obscured latency extensions

Figure 4.8: The proposed latency-extension detection framework and its implementation in real-life systems
used in the modelling of the baseline training and evaluation datasets. The resultant models where then compared, using the J-S distance metric, with the baseline models belonging to the respective memory access categories. The obtained J-S distances are shown in Figure 4.9, evidently, the majority of distances are in the 0.5 to 0.8 range, indicating non-negligible discrepancy between the compared baseline and anomalous latency models, despite the fact that this discrepancy was mostly not evident when examining the individual latencies in Figure 4.1. Another noticeable observation from the obtained J-S distances is that the distances reflect the type of DoS action and its severity. For example, the J-S distances related to DoS scenario C are the ones with the greatest diversion from the baseline J-S distances (i.e. the J-S distances obtained when both the training and evaluation latency datasets belong to trusted DRAM operation), conforming with the fact that, among the simulated DoS scenarios, DoS scenario C is the one with the greatest latency-extending action. Another example, is the case of DoS scenario B, which, according to the obtained J-S distances, does not induce large discrepancies in the write-after-write (WW) categories. This reflects the fact that DoS scenario B was designed to target read transactions.

The results of this experimental verification indicate the effectiveness of the proposed framework in detecting divergence from baseline latency distributions, and anomalous trends, peaks, or latency extensions in run-time latencies. Additionally, it shows its effectiveness in pointing out the particularities of the detected anomalies and the extent of their impact on the different memory access types. This is specifically useful in detecting latency-extending DoS HTs, given that they may be designed to only target a specific type of transactions, related to, for example, the memory access patterns of certain critical or confidential on-chip processes (e.g. an on-chip encryption process could be indirectly attacked by extending the latency of a certain sequence of memory accesses known to be initiated by this process).

4.6 Experimental setup

All of the latency datasets used in testing the introduced hypothesis and in developing and testing the proposed framework were derived from the integration of two open-source simulators; the gem5 CPU simulator [317], simulating an out-of-order CPU, and the DRAMSys memory simulator [307–309], simulating a DDR3 DRAM and its controller. The specifications of the simulated CPU and DRAM are as detailed in Table 4.2. A subset from the PolyBench benchmark suite [310] was selected to run
on the simulated CPU to generate streams of memory access requests and direct them to the simulated DRAM. In order to generate memory access requests at a higher rate, accommodating the needs of the conducted experiments and the machine-learning based modelling, the CPU was simulated at a wide range of frequencies. Additionally, the source code of the DRAMSys simulator was slightly modified to enable the extraction of the needed memory access information, namely, the start time-stamp, the end time-stamp, the row, and the bank of each memory access. The simulators were run until around 20 million memory access traces were obtained and recorded for the analysis and modelling steps. The simulated PolyBench benchmark applications and the corresponding numbers of training and evaluation datasets are outlined in Table 4.3. The performed latency sorting was based on 1000-bin quantization of the latency ranges of the 32 different memory access categories. This relatively large number of bins was chosen because of the wide latency ranges obtained from the simulated system. For the KDE modelling, we used the KDE implementation provided by the Scikit-learn tool [318], while the J-S distance calculations were performed in Python.

The DoS latency-extension scenarios were introduced through small-modifications in the code of DRAMSys. In particular, an integer multiplier was used in varying the value of the “notification delay” parameter in DRAMSys. This parameter is used by
DRAMSys in calculating the time needed to forward transactions from the arbiter to the scheduler before processing them by the DRAM, and the time needed to release them from the DRAM controller once they are processed.

Table 4.2: The experimental setup

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>System simulation</td>
<td>Gem 5 simulator</td>
</tr>
<tr>
<td></td>
<td>System emulation (SE) mode</td>
</tr>
<tr>
<td></td>
<td>O3 CPU (out-of-order CPU)</td>
</tr>
<tr>
<td></td>
<td>A range of frequencies from 1 GHz to 8 GHz</td>
</tr>
<tr>
<td></td>
<td>32KB L1 cache and 8KB L2 cache</td>
</tr>
<tr>
<td>DRAM simulation</td>
<td>DRAMSys simulator</td>
</tr>
<tr>
<td></td>
<td>gem5 coupling mode</td>
</tr>
<tr>
<td></td>
<td>DDR3 standard</td>
</tr>
</tbody>
</table>

Table 4.3: Number of transactions in the testing and evaluation datasets

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Training</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>atax</td>
<td>2,420,781</td>
<td>1,000,000</td>
</tr>
<tr>
<td>jacobi-2d</td>
<td>7,030,047</td>
<td>2,000,000</td>
</tr>
<tr>
<td>adi</td>
<td>5,129,039</td>
<td>2,571,703</td>
</tr>
<tr>
<td>mvt</td>
<td>1,101,975</td>
<td>1,498,752</td>
</tr>
<tr>
<td>3mm</td>
<td>1,259,717</td>
<td>289,503</td>
</tr>
<tr>
<td>bicg</td>
<td>83,773</td>
<td>216,635</td>
</tr>
<tr>
<td>doitgen</td>
<td>1,286,567</td>
<td>879,942</td>
</tr>
<tr>
<td>2mm</td>
<td>1,002,676</td>
<td>39,197</td>
</tr>
</tbody>
</table>
4.7 Evaluation

In addition to presenting an adaptable, lightweight, and independent framework for run-time monitoring and detection of abnormal latency extensions, this chapter provides experimental evidence of potential trust and security implications of inherent physical and operational characteristics of some electronic elements. This experimental evidence is a clear indication of the depth and breadth of the issue of run-time hardware trust and that it is not necessarily confined to a set of defined threat models or limited to certain design-related vulnerabilities. A latency-extending attack such as the one described in this chapter does not conform with the conventional model of DoS attacks as it does not lead to a complete loss of availability nor does it extend latencies beyond the maximum expected latency. However, the strength of its impact lies in its prolonged and obscure nature that exploits delay-accumulation as the attack vector.

The presented run-time monitoring framework is, in principle, applicable with other types of electronic elements that suffer from the issue of latency unpredictability and variability and hence may fall victim to obscured latency attacks. A requirement, however, is that their latency could be categorized and modelled based on dynamic features that are deducible at run-time. The DRAM was chosen as the focus of the development and testing of this framework as it is a ubiquitous element used as the main memory in real-life systems of different sizes and applications. The framework was designed with the objectives of adaptability, independence, autonomy, and computational lightweightness in mind. Adaptability is achieved by basing the latency characterization and subsequent analysis steps solely on the features or characteristics of the transactions processed by the monitored element. The fact that the chosen transaction features could be easily inferred at the interface of the monitored element, allows the implementation of the framework in a device-agnostic manner, regardless of the application of the SoC or the internal particularities of the monitored element such as, the operating state and the refresh mechanism, in the case of monitoring DRAMs. Another aspect of the adaptability of the proposed framework is reflected in its flexible and customizable implementation that could be tailored based on a number of different factors related to the capabilities of the system and its run-time trust and security needs. Examples of the framework parts that could be customized include; the frequency of performing the latency modelling and verification tests, the number and depth of the bins used in quantizing the latency
ranges of the different memory accesses, the kernel and bandwidth used in KDE modelling, the choice of transaction features to be included in the characterization, and the choice of accepted ranges of J-S distances.

Integrating the logic needed by the framework for live data collection and sorting in a wrapper external to the monitored element, allows independence from the monitored element. This paves the way for the implementation of the framework in run-time latency monitoring of outsourced and COTS elements. For total independence from untrustworthy parties and for enhanced security, it is advised that this wrapper is fabricated in-house or in a trusted local foundry. As the framework does not rely on its monitoring and detection of any input or feedback from the monitored element or from any other potentially untrustworthy on-chip element (e.g. regarding the interaction between on-chip elements), the objective of autonomy is achieved. The proposed hybrid software/hardware implementation in which, a software module is used for the machine-learning and computational parts of the framework, significantly minimizes any computational overhead on hardware. Additionally, the periodic performance of the modelling and analysis steps is another factor to further computational and operational lightweightness. The hybrid implementation of the framework also allows it to be scalable and used in monitoring the latency of several elements or memories on the same SoC with minimal overhead in hardware. Only a set of additional counters and small conditional logic are needed to perform the live data collection and sorting for an additional element.

The ability of the framework to find latency patterns and distributions in the unpredictable DRAM latency, based on information that is easily gathered from the system and the DRAM interface, allows its utilization for other non-trust and security purposes, examples include, but are not limited to:

- Assisting in improved understanding of the latency behaviour of DRAMs under different loading conditions, different access patterns, and different refresh mechanisms leading to improved tolerance of unpredictable DRAM latencies in current and future DRAM architectures and SoCs.

- Implementing a statistical approach, based on continuous run-time monitoring, to analysing the long-term effects of aging, temperature, electrical and mechanical stress, and other physical and external influences on the latency performance of DRAMs deployed in different systems and environments.

- Early detection of DRAM failures and early-aging based on the detection of
gradual degradation in latency performance, in the shape of, for example, a shift in latency ranges or abnormal latency peaks and trends.

- Effective prediction of DRAM access latency at a given point of time in the future based on the latency models constructed in previous cycles and on analytical deductions of potential shifts in latency trends.

A security related application of the framework, other than its intended application in detecting abnormal latency-extensions, is using it in detecting suspicious faster than normal responses of the monitored element. Although not a DoS attack, a sudden unusual rise in the number of occurrences of small latency values could be an indicator of another type of attack such as data injection by the DRAM controller. In such an attack, the DRAM controller does not attempt to access the DRAM when it receives access requests (as it should) but rather it sends its own internally generated fake responses. As the time to fetch responses from the DRAM is eliminated, a spike in small latency values may be observed in the resultant latency distributions.

### 4.7.1 Limitations

In spite of the fact that the framework could be extended or modified to fit the needs and requirements of different SoCs, a large number of modifications can only be performed at design-time and hence need to be decided in the pre-silicon stages. Modifications related to the transaction information that is to be included in the characterization or the number and size of bins used in sorting transactions should be considered at design-time as the data collection and sorting process is performed in non-programmable hardware and cannot be amended once the SoC is fabricated. Nevertheless, modifications related to, for instance, the adoption of a different latency modelling method (*i.e.* other than KDE or KDE with different parameters) or a different statistical comparison method (*i.e.* other than J-S distance), or related to changing the frequency of the tests (provided that it is within the limits of the hardware bins or counters) could be performed at run-time.

Another consideration related to the implementation of the framework, is that the monitoring of a naturally variable metric might require periodic updating of the reference model to avoid false detection of anomalous behaviour that is originally just an accumulation of small natural shifts in the monitored metric. Therefore, it should be realized that it is not realistic to assume, given the various environmental, physical,
mechanical, and aging factors affecting semiconductors, that the baseline latency distributions obtained as a reference in the pre-deployment stages or in the early days of deployment would be valid even after years of operation. Therefore, it is advisable to update the baseline latency models, every certain number of operation cycles or years. The update could be based on fresh latency data, obtained from isolated and trusted testing of the monitored element. Another potential source is information about the expected amount of shift in latency deduced from some historical data or from analyzing the run-time latency models obtained from the same element over time (i.e. learning the natural amount of latency shift).

4.8 Conclusion

In this chapter, we shed light on a new type of untrustworthy behaviour that exploits the physically and/or operationally variable characteristics of electronic elements in inducing a slow and subtle, yet consistent and obscured denial of service or performance degradation attack. To this end, we introduced a hypothesis that suggests the possibility of utilizing the natural and inherent variability in the processing latency of some electronic elements in masking or obscuring anomalous latency extensions. The DRAM is an example of a ubiquitous and widely used electronic element that could be prone to obscured latency-extension attacks due to the physical and operational factors affecting its processing latency, and deeming this latency inconsistent, variable, and unpredictable. Through experimental verification we tested the introduced hypothesis on a simulated DDR3 DRAM under different workloads and types of DRAM accesses as derived from running benchmark applications on a simulated CPU. The experiments verified the possibility of using the variable nature of DRAM latency in obscuring latency-extension DoS or performance degradation attacks, realized through the introduction of delays in the internal processes of the DRAM. These delays could be designed to extend the latency of the DRAM beyond the nominal expected latency of a certain type of memory accesses but never beyond the maximum expected latency, hence hiding within the “trusted” range of operating latency and effectively evading detection. In real-life scenarios, such obscured latency-extensions might be initiated from a HT embedded inside the DRAM, a fault injection attack, a software or architectural level attack, or even unintentionally due to a hardware fault or DRAM aging.

In response to our realization of the vulnerability posed by the variable and
unpredictable DRAM latency, and its serious trust and security consequences, we developed an independent and adaptable framework for run-time modelling-based monitoring of DRAM latency and consequent detection of obscured latency extensions. The performed characterization is based on features of DRAM accesses that could be inferred from the interface of the DRAM, in addition to information about the dynamic DRAM load. In spite of the fact that these access features and the dynamic load do not provide a precise prediction of DRAM latency at a given point of time, they directly influence the internal DRAM operating state and hence could be used in determining the range and distribution of latencies. The latencies obtained from the DRAM simulations were, therefore, categorized based on the access features and DRAM load. As each category showed a distinctive distribution and range of latency values, the proposed framework is based on modelling the latencies obtained from trusted operation using machine-learning methods, particularly the kernel density estimation. The resultant models are used as references for comparing with run-time latency distributions after categorizing and sorting them based on the access features and loading conditions. The last step of the framework is concerned with quantitatively comparing between the reference latency models and the run-time latency model, where the occurrence of noticeable discrepancy is considered an indicator of obscured latency extensions.

The framework, although designed for addressing the trust and security issue of obscured latency extensions, could also be implemented for other performance-related purposes such as long term analysis of the impact of environmental conditions on DRAM latency. Additionally, the framework was designed to be implemented in a hybrid software/hardware manner with the computational and machine-learning parts implemented in software, allowing the adoption of the framework in systems with stringent limits on hardware area and power overhead. As the monitoring and detection process is based solely on information gathered from the interface of the DRAM, the framework is independent of the monitored element and its architecture, design, and application. This feature also allows the implementation of the framework in isolation from untrustworthy IP suppliers. This grants SoC integrators the ability to monitor and analyze the latency behaviour of the third party DRAMs employed in their SoCs in spite of them being oblivious to the internal specifications and architecture of those DRAMs.

The chapter started with the motivation in section 4.1 that presented some of the different ways in which a DoS could be carried out in hardware. The realization that a
4.8. Conclusion

Hardware DoS attack need not cause total disruption of system availability to be able to induce availability-related issues had motivated the work presented in this chapter, and led to the focus on the possibility of exploiting naturally variable latency in initiating DoS attacks. Given that DRAMs suffer from the issue of latency variability and unpredictability they were chosen for performing the latency analysis presented in this chapter, therefore, section 4.2 provided a background on the physical and operational reasons of DRAM latency variability in addition to a literature review on what has been done, so far, to address this issue. Stemming from the fact that the DRAM latency variability issue is not yet eliminated, section 4.2 presented a hypothesis on the possibility of abusing this variability in initiating obscured latency-extending attacks. As the hypothesis was verified, section 4.3 presented the categorization, modelling, and comparison convention that the proposed run-time latency-extension detection framework is based on. Section 4.4 provided details of the implementation of the framework whereas section 4.5 presented the experiments carried out to verify the effectiveness of the framework in characterizing DRAM latency at run-time, and consequently detecting obscured latency-extensions. The simulators, benchmark applications, and software tools used to develop and test the framework are detailed in section 4.6. Section 4.7 provided an evaluation of the developed framework in terms of its conformance with the desired objectives of adaptability, independence, autonomy, and computational lightweightness, and in terms of the limitations or considerations that need to be taken into account when deploying the framework in real-life SoCs.
Chapter 5

Run-time integrity monitoring of analog front-ends

5.1 Introduction

Portable electrocardiogram devices, smart water treatment systems, and airplane fuel distribution systems, at the first glance seem to have nothing in common except that they all incorporate electronic processing and control systems built from SoCs and ICs. These systems are very different in terms of purpose, architecture, specification, size, and operating limits and conditions. Nevertheless, one major unifying factor among these systems, and the majority of systems used in real-life applications, is that the signals they receive from the physical environment are capable of taking an infinite number of values, while the systems themselves are capable only of comprehending digital ‘1’s and ‘0’s. A bridge, therefore, is needed between the continuous real-life signals and the digital processing units that cannot deal with them in their raw continuous nature. This bridge is the process of converting the analog continuous signals to digital signals that are discrete in time and in value. This process is referred to as the analog to digital or A/D conversion process and is performed by analog to digital converters (ADCs). ADCs are crucial to the operation of electronic systems that deal with sensor-measured real-life signals and quantities such as biomedical signals, air temperature and pressure, level and flow of certain fluids, and voice and sound signals. Figure 5.1 highlights the role of sensors and ADCs in controlling physical processes such as those used in industrial control systems. The control decisions taken by the digital processing unit and performed by the actuators are all based on input received from ADCs and sensors.
ADCs do not work in isolation and the real-life signals that they receive for the purpose of conversion go through a set of analog electronic elements that prepare and condition them for the A/D conversion process before forwarding them to the ADC. This preparation includes amplifying weak signals or attenuating strong signals (i.e. in terms of amplitude) so that they fit within the operating range of the ADC, and filtering out minor impurities or noise in the signals received from the sensors. This preparation is performed by amplifiers or attenuators, filters, and potentially application-specific circuitry, that, along with the sensors used to sense and detect the signals from the physical environment, are referred to as the analog front-end. In spite of the vital role that analog front-ends and ADCs play in SoCs, their trust and security has, in practice, mostly been an afterthought, if considered at all. One likely reason is the almost non-existent interaction of this part of the SoC with the outside world, other than its direct surrounding physical environment, which may give the impression of security due to obscurity. Another reason is the small circuitry of analog electronic elements and their high sensitivity to design modifications, which might make it hard for an attacker to insert a HT or initiate a fault injection attack. Nevertheless, as literature shows that analog circuits are susceptible to hardware attacks and trust issues, ignoring the security and trust of analog front-ends is a serious shortcoming as the output of the analog front-end and ADC is what dictates...
and directs the control and decision making parts of the system. A non-trusted and non-secure analog front-end and/or ADC implies, therefore, a non-trusted and non-secure system, regardless of the security measures implemented in the digital processing part of the system. The threat of having the digital decision and control parts of the SoC attacked from inside the SoC through the malicious utilization of the analog front-end and ADC is real and serious. An untrustworthy behaviour initiated from the analog front-end or ADC is capable of propagating through the SoC and reaching the processing units in a manner that is hard to parallel with any other attack vector, making this a very desired and rewarding attack for adversaries, especially if the induced untrustworthy behaviour is of subtle unobtrusive nature.

In this chapter we study the technical possibility of inducing an untrustworthy behaviour inside the analog front-end or ADC. The untrustworthy behaviour or malicious action of concern is a low-level one that would have better chances of evading detection and realizing the malicious intent targeted in this work, which is degrading the integrity or correctness of the output of the front-end. This malicious objective or intent was chosen because of its ability to lead to concerning and possibly irreversible implications such as, impaired decision making in the control and processing units, solely through the introduction of subtle incorrectness or modification to the inputs of these units which are the outputs of the analog front-end. One way to realize low-level integrity-degrading malicious actions is through the introduction of, either literally or in an analogous manner, intermittent bit-flipping and/ or stuck-at-faults. Such faults would be imperceptible to the observer of individual ADC outputs and may not have any direct short-term security implications, however, over time they have the potential to cause significant disruption in the downstream digital units that rely on the ADC’s output. To address this issue, we propose a run-time machine-learning based framework for the run-time monitoring and detection of incorrectness or integrity-degrading action in the output of the analog front-end. Although the targeted low-level faults could originate from any of the elements in the analog front-end, we base our framework on monitoring the output of the ADC, given that it is, in effect, the output of the analog front-end in a digitized form. By doing so, we are capable of simplifying the monitoring and detection process by focusing on the signal (i.e. ADC output) that reflects the integrity of both of the front-end and the ADC in addition to avoiding the complications of developing individual analog monitoring elements to integrate in the front-end. The proposed framework was tested and verified on three different use-cases corresponding to three
different application areas where analog front-ends and ADCs are typically used; biomedical diagnostic or monitoring devices, industrial control systems, and avionics.

### 5.1.1 Motivation

In 2011 it was revealed that analog circuits are one of the most counterfeited electronic elements with 1 in every 4 analog circuits reported to be counterfeited or pirated [319, 320]. This is undoubtedly an alarming situation, and at the time of writing this thesis, more than a decade later, it is likely that the situation has worsened furthermore by the shortage of legitimate chip supply caused by the Covid-19 pandemic and the China-US trade wars. In addition to the copyright and ownership issues related to IP piracy, counterfeit or pirated IPs are likely to suffer from performance and functionality issues for a number of reasons. Counterfeit IPs, even those of the cloned type that is identical to the original legitimate IP, are expected to be made of low quality materials and their functionality and performance are not expected to be properly tested and validated (i.e. by the counterfeiting design or manufacturing facility) as is the case with IPs produced legitimately. This increases the possibility of having those IPs showing signs of abnormal or untrustworthy behaviour at run-time, even if such behaviour was not intended by the provider or seller of the counterfeit IP. Counterfeit IPs of the out-of-specification, defective, and recycled types are, by definition, defective and non-compliant with specifications, hence may introduce faulty operations or behaviours at run-time. Counterfeit IPs of the type provided with forged documentation may also induce unexpected, abnormal, and untrustworthy behaviour in the SoC, given that they are likely to be deployed in systems with operating conditions and environment that do not match the ones they were designed for. The reported large numbers of counterfeit analog IPs in the market increase the possibility of SoC integrators, possibly unknowingly, integrating such IPs in their SoCs. This, combined with the trust and security implications of potential unexpected and abnormal behaviour carried out, either intentionally or unintentionally by such IPs, show the importance of developing techniques for ensuring the security and trust of analog circuits, especially at run-time, where the counterfeit-related defects and faults are expected to surface.

Another threat to the trust and security of analog circuits and ADCs is that of hardware Trojans. As analog front-ends consist of analog circuits and mixed signal circuits (i.e. A/D converters), they compose of both analog circuit parts and digital
circuit parts, and hence are prone to HTs realized through digital triggers with analog payloads, analog triggers with analog payloads, and analog triggers with digital payloads. An example of a HT with a digital trigger but an analog payload is the one introduced in [321], in which digital circuitry is used to “steal” the AES encryption key bits from on-chip registers. The key bits are then leaked out of the chip using the analog payload, which is based on a PMOS transistor that is connected to the on-chip transmitter and that leaks bits through glitches in the transmitted signal. The glitches are realized by drawing additional small amounts of current (by the PMOS) that are within the range allowed for process variations, hence, this HT exploits process variation in realizing a covert channel attack. The A2 HT which consists of an analog trigger with a digital payload has gained considerable attention in the field due to its capability of causing a serious violation of processor access control and initiating a privilege escalation attack, with minimal additional circuitry [322]. The analog trigger in the A2 HT is a capacitor that is charged with the toggling of a victim wire connected to it. Once this capacitor is fully charged, it activates the digital payload circuitry which could be designed to perform a range of different malicious actions from the overwriting of some digital value saved in a certain register (e.g. used in privilege escalation attacks) to the initiation of additional delay in some on-chip processes. The A2 and the covert-channel HTs, although based on different attack mechanisms, are examples of HTs that use analog means in facilitating and/or enabling attacks targeting the security and confidentiality of processes conducted in the digital domain. This shows, therefore, the possibility of utilizing the analog front-end in compromising the security of the processor and the whole SoC.

In principle, a HT with an analog trigger and analog payload could be based on any type of analog trigger, such as the capacitor-based triggers of the A2 HT, that could be used in activating any type of analog-domain payload. Nevertheless, a notable type of analog trigger and analog payload in HTs is what is referred to as Trojan state. Trojan states do not satisfy the conventional model of HTs as they do not incorporate any modifications or additional circuitry in the targeted circuit and hence could be considered one of the stealthiest types of HTs. Trojan states utilize the fact that some analog circuits have multiple stable operating states, and hence these HTs use variations in initial conditions or temperature to force the circuit to switch its operation to an undesired state with consequently unexpected behaviour and output [320, 323]. The feasibility of implementing Trojan states and the possibility of using them in changing output characteristics of analog circuits was demonstrated.
Chapter 5. Run-time integrity monitoring of analog front-ends

with filters, oscillators, and operational amplifiers as the target \[324\] \[325\]. Undesired output characteristics in one of the elements in the analog front-end inevitably translate to untrustworthy behaviour in the SoC leading potentially to disastrous run-time consequences.

In addition to the trust and security threats posed by counterfeiting and HTs, side channel and frequency injection attacks could successfully induce untrustworthy behaviour in analog and mixed signal circuits. In \[326\], a side channel attack exploiting the noise and power signals of ADCs in intercepting “secret” data such as the encryption key was introduced. Whereas in \[327\], it was shown that an attacker could trick the ADC in an industrial control system (ICS) to accept and convert input signals injected with abnormal frequencies that are capable of causing undesired vibrations in the motor or turbine in the system, leading, in turn, to permanent irreversible damage and potentially serious health and environmental consequences.

The fact that the operation of analog circuits is influenced by a large number of parameters, such as their transfer function parameters, makes them sensitive to fluctuations or instabilities in these parameters and, therefore, prone to faulty operation (\textit{i.e.} parametric faults) due to maliciously-induced parameter variations. A fault injection attack in the shape of a voltage glitch, realized as additional noise in power supply lines, could lead to a change in the frequency of oscillation or jitter in an on-chip oscillator. This could pave the way for a range of malicious actions that would benefit from the altered oscillator frequency in for example, analyzing the side channels of the SoC and extracting information about its operation after slowing it down or disrupting the normal operation of the SoC due to increased oscillator speed \[328\].

The sensitivity of analog and mixed signals circuits to even slight variations in their parameters, their inclusion of both analog and digital parts or circuitry, and their relatively simple and limited functionality (\textit{i.e.} analog and mixed signal circuits often serve a single a purpose) all contribute to their vulnerability to HTs, side channel, and fault injection attacks. Additionally, the long life-time of these circuits and the fact that they are mostly fabricated using older technologies make them desirable counterfeiting targets. This is especially the case with the increasing demand for analog circuits in parallel with the emergence of IoT and autonomous devices, making the business of counterfeiting analog and mixed signal circuits a lucrative one. Another issue related to the trust and security of analog and mixed signal circuits is that the majority of techniques developed to counter counterfeiting or HTs are based on digital circuitry
5.2 Degrading the integrity of the output of the analog front-end

Integrity is one of the pillars of cyber security and is used to refer to the case where data and information in a given system is maintained and not modified or tampered with in any way or form by an unauthorized user. Integrity is sometimes viewed as a

and hence cannot be, without significant modifications, adapted to the case of analog circuits.

These realizations should open our eyes to the importance of deploying measures targeting the trust and security of analog front-ends and ensuring their conformance with trustworthy behaviour at run-time. Such techniques would not only secure the analog front-end but would effectively serve as protection for the digital processing and decision-making part of the circuit, shielding it from the potentially catastrophic impact of the propagation and escalation of untrustworthy analog front-end behaviour.

5.1.2 Chapter outline

In this chapter, we target a type of untrustworthy behaviour that could be induced by a HT, fault injection attack, counterfeit IPs, or even unintentionally by hardware faults and semiconductor aging. The untrustworthy behaviour of concern is the introduction of unwanted and undesired frequencies in the output of the analog front-end, degrading its integrity and potentially leading to errors and failures in the digital processing parts of the SoC. The introduction or alteration of signal frequency is analogous to the introduction of incorrectness in the shape of bit flips and/or stuck-at-faults to the digital representation of the signal, therefore, a literature review on potential ways to realize such faults is presented in addition to a review of measures developed to counter integrity-degrading attacks in analog front-ends. After that, the proposed mathematical and statistical approach to the monitoring of the integrity of the output of the analog front-end is introduced and discussed. The machine-learning based framework proposed to implement this approach in detecting incorrectness in the output of the front-end at run-time is then introduced and detailed along with its implementation and experimental verification. The chapter wraps up with an evaluation of the framework and a discussion of its limitations and potential areas of improvement.
system property in the sense that the output of the whole SoC has not been modified or tampered with, however, another aspect of integrity is that related to the “correctness” of the SoC’s internal signals and their conformance with trust and security expectations. Lack of integrity, especially on the level of SoCs and their constituting IPs, is not necessarily caused by some unauthorized access. For example, a HT could be embedded in an IP to preform some additional non-requested or expected processes on signals that normally pass through this IP and are processed by it. This would degrade the integrity of the affected signals, however, the cause of this degradation would not be an unauthorized access or user but rather an unexpected behaviour performed by an entity (i.e. the IP) authorized to access and process the affected signals.

Introducing incorrectness to the output of the front-end could take many shapes and could be based on complete deforming of this output. Nevertheless, analog circuits are small in size and noticeably sensitive to variations in their surrounding, making the realization of such large-scale high-impact type of attacks challenging. We therefore focus on the more realistic and stealthy case of subtle unnoticeable modifications in the output of the analog front-end, taking the shape of bit flips in the middle or lower bits of the ADC output. Such modifications or integrity-degrading actions are unlikely to be of direct damaging impact yet their strength lies in the utilization of the accumulation of errors or erroneous outputs in eventually causing confusion, misjudgment, miscomputation, or indecision in the digital part of the SoC.

5.2.1 The attack vector

A common practice in SoC and IC design is to blindly accept the output of the analog front-end, represented in the ADC output, as a true and a correct representation of its analog input signal/s, without deploying further tests to verify this assumption. Another common practice is the outsourcing of the front-end and the ADC, given the wide range of types and designs readily available in the market, the relatively-standard functionalities of these elements, and the preoccupation of most SoC designers with enhancing the features of the digital SoC elements that are at the forefront of technology advancement. These practices complicate the issues related to trust and security of the analog front-end furthermore and provide an easy way for adversaries to attack the elements in it and compromise their security.

The untrustworthy behaviour of concern, abnormal bit-flips and stuck-at-faults,
may originate from defects or age-related degradation in counterfeit IPs deployed in the front-end. As a matter of fact, stuck-at-faults are, as detailed in chapter 2, considered one of the identified counterfeiting defects that should be targeted in pre-deployment validation tests. Nevertheless, it is possible that these defects or faults are missed in the validation tests, ending up in the deployed SoC due to insufficient test vectors or, in case of physical structural tests, due to the inability to reverse engineer and inspect each IC individually. Another potential source for this type of untrustworthy behaviour is fault injection attacks that are based on altering the frequency of the output of the elements in the front-end or that of the ADC. When it comes to outsourced analog front-ends or ADCs or those fabricated and tested in untrustworthy facilities, concerns of HTs arise.

HTs whether implemented as Trojan states in the analog circuits in the front-end, or implemented as a combination of digital and analog parts and embedded, for example, in the ADC, are capable of inducing abnormal bit-flips. One way in which the untrustworthy behaviour of concern could be induced by HTs is through the utilization and enhancement of capacitive crosstalk. An untrustworthy foundry may intentionally and maliciously route several wires, where one of them is a victim wire targeted by the bit-flip and the others are offending wires used to perform the attack, close to each other so that at run-time, the toggling of the offending wires induces an amount of capacitive cross-talk sufficient to maliciously flip the victim wire. The work in [44] shows the possibility of forcing the victim wire to stay at logic high for several cycles only through single toggling or switching of the offending wires. Such HT payloads utilize existing wires in the IC and hence have zero-footprint. In case when the HT inserting party is not the foundry, then, HT payloads as simple as a single logic gate connected to the target signal could also be used in inducing bit-flips or stuck-at-faults. ADCs in particular are susceptible to HT insertion given their mixed signal nature and the fact that they comprise both analog and digital parts. For instance, the counters in some ADCs could be used by the adversary as a HT trigger, counting a certain number of conversions before activating the HT payload. Additionally, many ADCs deploy capacitors that could be used in capacitor-based analog triggers. This type of analog trigger has a good chance of evading detection as the literature [329] shows that extending the charging time of the capacitor by connecting it to sources of small current (e.g. the reverse saturation current of an

---

1 The same applies to digital to analog converters (DACs), however, our focus in this work is on the elements used in analog front-ends in conditioning and converting analog signals to a form that is comprehensible by the processors in the SoC.
on-chip diode and the gate-oxide leakage current of thin-oxide NMOS devices) could be used as a buffer stage in apparent de-linking of the HT trigger from its payload, avoid any suspicions of abnormal behaviour, for example, from the occurrence of an event known to be a potential HT trigger [329]. The advances in HT realization, such as the ones discussed in this section, show the feasibility of introducing bit-flips and stuck-at-faults to the output of the front-end with minimal footprint and a high chance of evading pre-deployment detection.

The fact that untrustworthy behaviour of this type could originate from different trust issues, in addition to the rapid advances in relevant HT and fault injection attacks, show that developing techniques to monitor the output of the analog front-end for signs of incorrectness (or integrity degradation) at run-time is of paramount importance. This is particularly the case as it is neither feasible nor efficient to integrate measures and perform tests against all potential attack vectors and mechanisms in the design and pre-deployment stages.

5.2.2 Related work

The untrustworthy behaviour of concern, to the best of our knowledge, has not been targeted exclusively before by run-time monitoring techniques that are independent of the design and architecture of the monitored front-end and are application-agnostic, as is the case with our proposed framework. However, run-time integrity monitoring of signals in SoCs with analog front-ends has been targeted in a number of publications.

In [330], the possibility of using HTs in injecting missing-code errors in the output of successive approximation register (SAR) ADCs was presented and discussed along with a bit-monitoring approach for the detection of such faults. The introduced bit monitoring convention is based on the realization that for signals of standard types (sine, ramp, sawtooth), the arithmetic difference between adjacent digital codes (i.e. ADC output codes) could only take one of three values; zero, +1LSB (least significant bit), or -1 LSB, therefore, any deviation from this convention is considered an anomaly. Although device and application-agnostic, the main issue with this bit monitoring convention is that it is only applicable to certain types of signals hence is more suited for occasional offline integrity testing rather than continuous online integrity monitoring. Another approach to run-time integrity monitoring as proposed in [330] is the deployment of component redundancy or duplication. In [330], it is assumed that the sample and hold circuit in a given ADC is
HT-infected, and that its sampling capacitors maliciously hold certain instances of the input analog signal longer than they should, leading to multiple A/D conversion of this instance and hence degraded integrity. Deploying a replica of the sample and hold circuit is proposed for detecting such behaviour as any discrepancy found between the run-time output of the main circuit and that of the replica circuit is considered an indicator of untrustworthy behaviour. The main issue with this approach is that it is generally not suitable for outsourced ADCs or those fabricated in untrustworthy foundries as a HT could be inserted in both sample and hold circuits or even in the comparison circuitry. Moreover, although the concept of component replication or redundancy could be implemented with most types of circuit components, and even with whole ADCs or whole analog front-ends, the associated power and area overhead on hardware is likely to pose an obstacle to its wide-spread use and adaptation. In [331], four potential HT attacks concerned with manipulating and degrading the integrity of the inputs or outputs of the modulator block in delta-sigma ADCs are introduced. The proposed countermeasure is based on filtering internal signals in the modulator and using curve-fitting models. Although the use of filters is a viable approach for smoothing-out minor manipulations in signals, the technique introduced in [331] is based on filters implemented inside the ADC and therefore, it is not an ideal approach for outsourced ADCs in addition to it being tailored for delta-sigma ADCs.

The work in [332, 333] introduced a framework for integrity monitoring of intermediate signals in SoCs based on the generation and multi-stage verification of three different types of hashes. The first hash is analog and generated from the output of the sensor in the analog front-end, the second hash is generated from the output of the ADC, whereas the third hash is generated from the output of the digital signal processing unit and is based on unique features of the signal of concern (e.g. physiological features in case of monitoring biomedical signals). The analog and digital hashes are compared with hashes generated from the output of the processing part while the third hash is used for verifying the conformance of the output of the processing unit with known and defined features of the signal of concern. As this framework uses independent elements for the hash generation and comparison, it is suitable for implementation in SoCs with outsourced elements. Nevertheless, the framework adopts the notion that the analog front-end is trusted and that reference hashes could be generated from its output or internal signals. Therefore, in case of implementing this framework for detecting integrity degradation in the output of the
front-end, the hash generation and comparison mechanism need to be modified as in, for example, generating an additional hash from the input analog signal and comparing it with the analog and digital hashes generated from the sensor and the ADC. Additionally, the proposed generation and multi-stage verification of hashes requires a non-negligible amount of additional circuitry and may lead to a non-desired hardware and performance overhead.

5.3 Approach

The process of A/D conversion is based on two processes: sampling which conventionally used capacitors in capturing instances (referred to as samples) of the analog input signals at defined points of time, based on the ADC frequency, and quantization which is concerned with mapping between continuous amplitude (i.e. voltage) levels and defined digital codes. The range of these codes mainly depend on the resolution of the ADC as an n-bit ADC is capable of producing $2^n$ digital output codes, where each code is n-bit long. The sampling and quantization process basically turn the analog signal, which is continuous in time and amplitude into a digital signal that is discrete in time and in amplitude. Although the purpose of these processes is well-defined, there are numerous ways to implement them in real-life ADCs based primarily on the speed, power, resolution, and area requirements of the specific application. The variation in ADC designs in addition to the fact that the architecture of the analog front-end is dependent on the type and characteristics of the analog input signals, and the consequent type of needed signal conditioning and preparation, highlight the need to develop run-time integrity monitoring techniques that are independent of these variations.

In our approach to independent device-agnostic integrity monitoring of the output of the analog front-end, we focus on one of the unifying factors among all types of analog front-ends. This factor is reflected in the fact that, in normal healthy operation, the output of the ADC is, and should always be, a true digital representation of the analog input signal. Therefore, the correctness and integrity of this digital representation could be verified in relation to identifying characteristics of the input signal. In order not to limit the approach to characteristics or features that are unique to certain types of input signals and applications, we base our monitoring on temporal and arithmetic relationships between the different instances of the signal. Such relationships could be extracted from any type of signal and the only input needed to
calculate them is a series of ADC output codes. Once extracted at run-time, they are compared using statistical and machine-learning methods with the same relationships when extracted from pre-deployment trusted operation of the front-end, to detect any abnormal trends, patterns, or behaviour. Although the relationships are extracted from the output of the ADC, they are effectively an indication of the trustworthiness of the whole analog front-end (and the ADC). This is because the output of the ADC would have passed through all elements in the front-end before reaching the ADC, hence any introduced incorrectness by any of the front-end elements would be reflected in the digital ADC output. This approach allows the integrity monitoring to be based solely on the output of the ADC (*i.e.* over certain intervals), allowing in turn the proposed framework to be independent of the type and design of the ADC, the instantaneous value of the analog input (*i.e.* as no direct input-output comparison is performed), and the design and specifications of other elements in the front-end. An additional benefit of implementing the monitoring at the output of the ADC is that in this case it could be performed with a simple digital circuitry avoiding the need to implement custom analog monitors at the output of each element in the front-end. This approach also allows the detection of incorrectness in the output of the ADC irrespective of its source, being malicious and intentional, or unintentional and fault-related. Therefore, identifying the source of the incorrectness or integrity degradation, in terms of the type of responsible attack or action, and in terms of the offending or untrustworthy element in the front-end, is out of the scope of this work. Nevertheless, once incorrectness is detected by the proposed framework, isolated security testing of the elements in the front-end and the ADC could be performed to infer the responsible element.

### 5.3.1 Temporal and mathematical analysis of ADC output

The proposed run-time integrity monitoring framework is based on the extraction of identifying and characterizing temporal and/or arithmetic relationships from the trusted output of the ADC and using them as a reference to compare against at run-time. In order to achieve independence of the type of application or the monitored signal, several generic relationships were studied and investigated for the purpose of identifying a single widely applicable relationship. The proposed framework uses machine-learning methods to model the distribution of extracted relationships. The best candidate relationship is, therefore, one that could be distinctively modelled.
Periodic signals with defined maximum levels or peaks could be identified by these peaks and the temporal distances between them, hence one relationship that could be adopted is the distance (in number of samples) between consecutive peaks. Figure 5.2a shows the distribution or histogram of the distances between absolute peaks (i.e. highest peaks in electrocardiogram signals; QRS peaks) obtained from the digital codes of a 80bpm electrocardiogram (ECG) signal. The distances between peaks reflect the input frequency and its ratio to the sampling frequency of the ADC, and hence is usable in detecting abnormal bit-flips that affect the value of the maximum output codes (for a given input) and/or their frequency of occurrence. Nevertheless, this relationship is strongly tied to the frequency of the input signal and could lead to errors in the monitoring and detection process due to slight variations in input frequency.

Another relationship is the distance between certain output codes, where the user (e.g. IC designer or SoC administrator) is concerned with the number of samples between the occurrences of specific digital codes. This relationship is especially useful in characterizing signals without distinct maximum and minimum values and that are somewhat stochastic and non-uniform in nature. Figure 5.2b shows a histogram of the distances between the occurrences of the digital code 01011000 in the 80bpm ECG signal. The distances fall within a certain range and hence could be used as a reference for correct conversion of an ADC. The main shortcoming of this relationship is related to the potential inconsistency in the frequency of occurrence of a certain digital code, given the non-ideal operation of real-life ADCs and the fact that, throughout the A/D conversion process, the same analog voltage level could be converted to different digital codes with +/- 1-2 LSBs variation from the ideal digital code. Therefore, when adopting this relationship in incorrectness detection, one should be aware of the expected variations or fluctuations in the A/D conversion process and take them into account in any subsequent analysis. This could be performed, for example, by looking into the distances between the occurrences of a number of consecutive codes instead of focusing on a single code.

As digital signals do not continuously vary in amplitude or value, meaningful information about the speed of change and variation in the analog input could be extracted from the frequency of change in the digital output. The rate or frequency of change in the digital output code is another investigated relationship, and is quantified by the number of samples between changes in the output code. A large number of samples between changes indicate a slowly-changing signal and vice versa. Some
signals, such as the ECG signal, show both small and large number of samples between changes as they consist of both fast and slow-changing components. Figure 5.2c shows a histogram of the rate of change obtained from the digitized 80bpm ECG signal. The rate of change is useful in detecting sudden bit-flipping that alters the normal intervals between changes in the digital code (for a given signal). The rate of change, however, would not be altered if the malicious bit-flips coincide with actual changes in output.

Investigating these relationships led to the realization that a universal relationship is one that reflects both the temporal and arithmetic change in the digital output codes of ADCs. The slope, or the ratio of the arithmetic change (difference in value) in the output code to the temporal change (number of samples) was therefore chosen for our approach as it effectively quantifies the frequency, factor, and direction of change in the output of an ADC. In our experiments, we found that slopes always fall within a certain range and have a certain distribution depending on the type and shape of the input signal. This distribution is noticeably sensitive to bit-flips or stuck-at-faults and exhibits recognizable new “histogram bins” whenever such modifications are introduced to the digital code. Figure 5.2d shows the histogram of the slope values obtained from correct conversion of the 80bpm ECG signal, whereas figure 5.3d shows the slope histogram for the same input but when bit 3 is stuck at level high. The histogram in Figure 5.3d shows a clear deviation in distribution (from the one in Figure 5.2d) and the introduction of new values, indicating the possibility of using the slope calculated from baseline trusted operation in run-time monitoring of the integrity and correctness of the output of the front-end. One important requirement, however, is that the type of the input signal is known (i.e. ECG, pressure sensor, flow sensor, etc) to accommodate the different ranges of slopes resulting from different shapes of signals. This is not a limiting requirement as the type of the input signal is usually pre-defined and directly related to the application of the system. Figures 5.2 and 5.3 show distributions of the four investigated relationships when extracted from the baseline digitized 80bpm, and from the stuck-at fault affected version of the same signal, respectively. Evidently, all relationships extracted from the stuck-at-fault affected signal exhibit some deviation from the baseline ones, however, the slope and the distances between peaks are the ones with the most visible and obvious deviations. As the distances between peak relationships is specific to periodic signals, the slope is considered the most suitable and universal candidate (out of the
investigated relationships) for the proposed framework.

Figure 5.2: Histograms of the temporal and arithmetic relationships calculated from the 80bpm ECG signal

Figure 5.3: Histograms of the temporal and arithmetic relationships calculated from the 80bpm ECG signal when bit 3 is stuck at logic ‘1’
5.4 The framework

The proposed framework for run-time integrity monitoring of the output of the analog front-end and consequent detection of any incorrectness in it adopts a black-box approach, assuming that the only knowledge available at run-time about the analog front-end is its output and the type of the input signal, which is pre-defined. Therefore, the primary action performed by the framework at run-time is concerned with the extraction of mathematical and temporal relations between the outputs of the monitored front-end (i.e. ADC outputs). Examples of potential relationships are the distances between peaks, distances between certain codes, rate of change, and slope. The IC designer or SoC integrator can decide to perform the monitoring using a number of relationships or just a single relationship, depending on the allowed amount of overhead and the trust and security requirements of the system. In the proposed implementation of the framework, we base our monitoring and detection on a single relationship, the slope, given its universality, and its quantification of the temporal, arithmetic, and directional change in signals. Basing the framework on a single relationship also allows its implementation in lightweight and small systems.

After the run-time extraction of the relations, and on a periodic basis, the relations are projected on machine-learning models that have been trained on the same arithmetic and temporal relations obtained from trusted pre-deployment operation of the front-end. Any inferred notable discrepancy between the projected run-time data and the trusted models is an indication of incorrectness and integrity degradation in the run-time output. The framework provides a metric for quantifying this discrepancy and consequently deducing the severity of incorrectness or lack of integrity in the monitored codes. The proposed framework, as shown in Figure 5.4, consists of two phases; a run-time monitoring and incorrect output detection phase and a trustworthiness assessment phase.

5.4.1 Run-time integrity monitoring and incorrect output detection

For the purpose of using the slope or any other relationship for constructing reference models that are to be used in live detection of incorrectness in the output of the front-end, correct and baseline values of this relationship should be obtained from trusted operation of the front-end. This is preferably performed in the pre-deployment testing and validation stages, where there is direct monitoring of the input-output relationship and hence high level of assurance of the correct operation of the
The slope histograms in Figures 5.2d and 5.3d show that the slopes obtained from the A/D conversion of real-life signals, whether trusted or injected with faults, have uniquely shaped distributions, possibly wide ranges of values, and a number of peaks and dips. Given the unique distribution, the slopes could be modelled using probability distribution or estimation machine-learning models, and given the non-defined and multi-peak shape of the distributions, non-parametric density estimation models are the best candidates. To this end, the kernel density estimation (KDE) was chosen in constructing reference slope models that are trained on the slope values obtained from the output of the front-end during trusted operation. These models are saved for later use in live monitoring and incorrectness detection.
At run-time, the arithmetic and temporal distances between changes in the output code are collected and used in slope calculation. Based on pre-defined intervals, the run-time slopes are queried in the saved KDE models to obtain an indication if these slope values or values close to them were “seen” by the KDE models in the training stage. If a certain run-time slope value was “seen” in the training stage, then the probability of the same value in the trusted slope distribution (i.e. the modelled distribution) is returned. The output of the query process is, therefore, a set of probabilities corresponding to the probability, if any, of each of the queried run-time slope values in the saved KDE model. For rigorous trust and integrity validation, the actual probabilities of the run-time slopes could be compared with the probabilities obtained from the query process (i.e. for the same run-time slope values) and used in further investigation of the distribution of the run-time slope values and their conformance to the expected distribution. For a more generic, fast, and lightweight approach, we adopt an incorrectness detection approach that is independent of the probabilities. In the proposed framework, the query process is followed by quantification of the extent of match between the run-time slope values and the slope values used in training the KDE models. To this end, we propose the match rate metric; which is the number of run-time slope values that return some probability when queried in the KDE model divided by the total number of run-time slope values. This metric allows the quantification of the deviation of run-time slope values from trusted slope values for a particular application as it effectively shows how many of the run-time slopes match or are close to those obtained during trusted operation. Therefore, the lower the match rate, the greater the number of new not seen-before run-time slopes, and the greater the extent of incorrectness or integrity degradation in the output of the front-end.

The bandwidth of the KDE modelling plays an important role in determining the accuracy of the framework in correctly detecting integrity degradation. On one hand, a loose KDE (high bandwidth) would identify a large range around each trained-on slope value as a valid value, defeating hence the purpose of the query process and the subsequent match rate calculation. On the other hand, a very constrained KDE would only identify exact trained-on slopes as valid slope values, which does not account for slight variations in run-time slope values as expected of real-life signals and the live non-ideal operation of the signal acquisition, conditioning, and A/D conversion processes.
5.4.2 Trustworthiness assessment

The second phase of the framework is designed to automatically and systematically distinguish trusted from untrusted operation of the front-end and deduce its level of trustworthiness, based on the match rates obtained in the first phase of the framework. An incorrectness severity metric is introduced for the purpose of performing the desired trustworthiness assessment. The severity metric is a quantified mapping between the calculated match rates and the extent of incorrectness in the run-time output codes. The extent of incorrectness in the output of the front-end or the severity of its deviation from the correct conversion of a given input signal, cannot be deduced directly from the run-time codes as the proposed framework does not deploy IP duplication nor perform live input-output comparison. However, during the extraction of baseline slope values from the trusted pre-deployment operation of the front-end, multiple bit-flipping and stuck-at-faults scenarios could be introduced to the series of output codes obtained from front-end to emulate the occurrence of these faults inside the front-end. The resultant modified codes are then used to obtain corresponding slope values, which are queried in the baseline trusted KDE models with the output of the query process used in match rate calculation. Performing this step allows the identification of the match rate ranges that correspond to different integrity degradation scenarios and, therefore, allows the deduction of the severity of integrity degradation and the assessment of the trustworthiness of the front-end at run-time.

In order to perform the proposed match rate vs. severity mapping, we identify the severity based on the position of the bit targeted by the fault and the frequency of occurrence of this fault. Consequently, we propose a weighted-sum formula, as shown in Formula 5.1, for calculating the severity metric and quantifying the introduced incorrectness. In this formula, the significance of the bit is the digit reflecting its significance or position in the output code as in, for example, a significance value of 2 for a fault-affecting bit 2. The frequency in this formula is defined as a percentage of the total number of output codes (in the inspected interval) that was affected by the fault hence for a fault targeting 50% of output codes, the corresponding frequency value is 0.5. In the performed experiments, it was found that the more significant the bit targeted by the introduced faults, the lower the corresponding match rate, even if the frequency of the fault remains unchanged. This is expected as targeting any of the bits by a bit-flip or stuck-at-fault would always introduce new slope values and/or change the normal distribution of slopes, whereas, the frequency of the fault may
5.4. The framework

coincide, intentionally or not, with the frequency or rate of actual change in the signal. Therefore, as our framework is more concerned with the occurrence of new slope values rather than the probability densities of these values, a weight of ‘1’ is assigned for the bit significance and a weight of ‘0.5’ for the frequency. Figures 5.5 and 5.6 show the mapping of the calculated severity to the targeted bit and the frequency of the bit-flipping, when applying different bit-flipping scenarios to the 80 bpm ECG signal, and the mapping of the match rates (corresponding to the same ECG signal and the same bit-flipping scenarios) to the severity, respectively.

\[ \text{Severity} = \text{significance}_{\text{bit}} \times \text{weight}_{\text{bit}} + \text{frequency} \times \text{weight}_{\text{frequency}} \] (5.1)

The mapping of the match rates obtained from the implementation of the different fault scenarios to the severity calculated from the same fault scenarios is performed via regression-based machine-learning models. At run-time, whenever a new match-rate is obtained from the implementation of the first phase of the framework, this match rate is provided as a predictor to the regression match rate vs. severity model, which would return the corresponding severity level. The obtained
incorrectness severity is a clear indicator of the bit where incorrectness or abnormal behaviour was observed and the frequency of occurrence of this behaviour in the previous monitoring cycle. The range of severity levels for an 8-bit ADC would normally be up to 9, corresponding to the case when bit 8 is flipped all the time, however, as bit-flips in the lower and middle bits are harder to detect, we focus on them in this work. For the sake of automating the decision on any remedial or counter actions that are to be taken once incorrectness and lack of integrity in the front-end output is found, levels of trustworthiness could be assigned to the different severity values while mapping each of the trustworthiness levels to certain alerts and appropriate remedial actions. Potential remedial or counter actions include offline isolated trust and security testing and verification of the individual elements in the analog front-end, and filtering the ADC output to smooth out any minor incorrectness or noise caused by low-severity integrity degradation. In systems with front-end or ADC duplication, the realization of a certain severity or trustworthiness level could be used as an indicator of the point of time where the additional front-end or ADC need to take over the operation of the main untrustworthy one.

5.4.3 Experimental setup

The experimental setup used to obtain the series of digital codes that were used in slope extraction, modelling, and the subsequent steps in the framework is based on
the A/D conversion of different types of signals performed by the open-source 8-bit arithmetic tracking successive approximation register (SAR) ADC simulator from [334]. To achieve diversity and comprehensiveness, the framework was tested and verified with a number of different types of signals pertaining to three different applications, in particular; the ECG signal from the ECGSYN tool-set [335, 336], readings of flow and level sensors from the secure water treatment system developed by iTrust in the centre for Research in Cyber Security at Singapore University of Technology and Design [337], and level and flow readings of airplane fuel tank sensors [338]. In order to be able to condition the signals and prepare them for A/D conversion, they were amplified or attenuated, as needed to fit the amplitude range of the ADC, and interpolated to obtain a more continuous presentation and a large number of samples as needed for the machine-learning based modelling. Interpolation is important in this context in order to produce signals that are as continuous (in time and amplitude) as possible, resembling the real-life characteristics of the analog signals that are fed to analog front-ends. Another reason for interpolating the signals before feeding them to the ADC is that this particular ADC simulator performs conversion on the basis of a sampling frequency that is equivalent to the input signal frequency, in case the signal is not of a sinusoidal type. Therefore, in order to satisfy the general rule of A/D conversion that for correct conversion, the ADC sampling frequency has to be at least twice that of the signal frequency, the preformed interpolation is also effectively, sampling of the signals. In the ECG case, a sampling frequency of 10 Ksps (samples per second) was specified in the ECGSYN tool. The sampling frequency of the sensors in the water treatment system is one sample per second. We interpolated those sensor readings at a rate of 4 samples per second before converting them to digital. As for the airplane fuel sensor readings, they were interpolated at a rate of 6.5K, given that less than 200 readings are provided in the dataset. The signal conditioning (amplification/ attenuation and interpolation), the ADC simulations used in converting the signals to digital codes, and the slope calculation were all performed in Matlab.

The KDE implementation provided by the Scikit-learn tool [318] was used for the KDE modelling of baseline trusted slope values. To ensure that the generated KDE models are neither too loose nor too constrained and that they are moderately detailed around the trained-on slope values, a Top-hat kernel and 0.1 bandwidth were chosen. The query process was performed by the (score.samples) function, which is part of the Scikit-learn KDE package that, when presented with a certain array of values (i.e.
run-time slope values in this case) returns the probability densities of these values if they were found in the queried KDE model, otherwise it returns zero. The match rate is, therefore, calculated based on the number of queried or “scored” slope values that return a non-zero probability. The fine-tree regression model provided by Matlab regression learner was used in modelling the relationship between the severity metric and the match rate metric. Upon training the fine-tree regression model on the severity and match rates of the different fault scenarios that were introduced to the ECG signal, a root mean square error (RMSE) of 0.11865 was obtained showing the feasibility and effectiveness of using this approach in indicating the severity of integrity degradation based on the obtained match rates.

The introduced bit-flipping scenarios were introduced to the output of the ADC either in a continuous manner or in a random manner. In the continuous manner, the flipping action is performed on a large number of consecutive digital codes, emulating a permanent fault or attack such as; an always-on HT. In the random case, however, the flipping was introduced to the codes occasionally on a random basis without selectively targeting a specific code as could be expected of a poorly counterfeit front-end or ADC that demonstrates abnormal behaviour only occasionally.

5.5 Implementation

The experiments carried out to verify the concept and effectiveness of the framework in run-time integrity monitoring and in detecting bit-flips were simulation-based as indicated in Section 5.4.3. In order to implement the framework in real-life systems, a hybrid hardware/ software implementation is recommended. In this implementation, the computational and machine-learning parts of both phases of the framework are performed in software whereas the extraction and collection of the temporal and arithmetic distances needed to calculate run-time slopes are implemented in hardware. This approach allows implementation in lightweight systems and limits hardware overhead.

The steps related to KDE modeling, match rate calculation, regression model training, and all of the other steps in the second phase of the framework can be implemented in a dedicated software module. Given the fact that the only input needed to perform the proposed monitoring is the series of ADC output codes over a defined interval, the hardware part of the framework is implemented independently of the front-end in, potentially, a separate wrapper sitting at the output interface of the
ADC or the analog/digital boundary of the SoC. As the hardware part is concerned with live tracking and logging of the amount and time of change in the monitored output codes, it primarily consists of a comparator used to detect changes in the output code, and a subtractor used for determining the amount of this change. Simultaneously, a counter is used to count the number of samples between changes in the output code as indicated by the comparator. The outputs of the counter and the subtractor are saved in logs, implemented as small storage elements or sets of registers, and sent to the dedicated software module for the query process and the subsequent match rate and severity computation.

5.6 Functional verification and evaluation

For the purpose of verifying the effectiveness of the proposed framework in the run-time integrity monitoring of the output of the analog front-end regardless of the application, it was implemented with a number of sensor signals obtained from applications related to biomedical systems, industrial control systems, and avionics.

5.6.1 Use case 1: The electrocardiogram

The Electrocardiogram (ECG) signal is the recording of the electrical activity of the heart (i.e. in voltage vs. time), as obtained by electrodes placed on the skin of the examined person. The electrocardiography test, from which the ECG signal is obtained, is one of the most common and adapted cardiac examinations and is used in diagnosing many heart conditions and breathing disorders. As ECG signals are analog in nature, analog front-ends and ADCs are used to condition them and convert them to digital for the extraction of diagnostic information. Malicious parties could, therefore, cause errors and failures in the diagnostic process through compromising the security and trust of the front-end. One attack scenario would severely modify the ADC output as to imply a certain heart condition for a healthy person or vice versa. This scenario is out of the scope of this work as it cannot be realized by the attack vector of concern; subtle and unnoticeable integrity-degrading actions taking the shape of frequency injection or bit-flips and stuck-at faults. Such high-severity attack scenarios are also not easily implemented as HTs given the non-negligible amount of malicious circuitry needed to change the features and shape of the ECG signal. Another less severe attack scenario would cause errors and confusion in the diagnosis...
by adding unexplainable noise and abnormal additional peaks and/or dips in the ECG signal, which could be easily achieved by malicious bit-flipping, and hence is addressed by the proposed framework. Figure 5.7a shows part of a 80bpm ECG signal without modifications whereas Figure 5.7b shows the same part when random flips of bit 4 (Scenario 2 in Table 5.1) are introduced. Figure 5.8 shows the histogram of the slopes calculated from the bit-flipping affected ECG signal and as can be seen the range of slope values does not conform with that of the correctly converted signal (as shown in Figure 5.2d).

Table 5.1: The bit-flipping scenarios introduced to the ECG signal and the obtained match rates

<table>
<thead>
<tr>
<th>Bit-flipping scenario</th>
<th>Frequency</th>
<th>Affected bit</th>
<th>Match rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 1</td>
<td>50% of code (random)</td>
<td>bit 5</td>
<td>26.7%</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>50% of code (random)</td>
<td>bit 4</td>
<td>34.67%</td>
</tr>
<tr>
<td>Scenario 3</td>
<td>16% of code (random)</td>
<td>bit 3</td>
<td>70.4%</td>
</tr>
<tr>
<td>Scenario 4</td>
<td>3% of code (random)</td>
<td>bit 2</td>
<td>88.6%</td>
</tr>
<tr>
<td>Scenario 5</td>
<td>50% of code (random)</td>
<td>bit 1</td>
<td>88.79%</td>
</tr>
<tr>
<td>Scenario 6</td>
<td>0.3% of code (random)</td>
<td>bit 3</td>
<td>79.9%</td>
</tr>
</tbody>
</table>
5.6. Functional verification and evaluation

(a) Baseline ECG signal

(b) ECG signal with abnormal bit-flips

Figure 5.7: Baseline and bit-flip (scenario 2) affected ECG signal

Figure 5.8: Histogram of the slopes of the bit-flipping affected (scenario 2) ECG signal

The ECG signal varies naturally, within clinically identified limits, from one healthy person to another. For the KDE model to distinguish those natural variations from anomalies, slopes calculated from 50 different variations of the ECG signal were used in training the KDE model. Those variations reflect ECG signals of individuals of different resting heart rates, between 60bpm and 100bpm, and different duration of ventricular depolarization and repolarisation. Further details about these variations are provided in appendix A.2. In order to test the feasibility of the framework in real-life electrocardiography systems, the KDE model was queried with slopes obtained from single variations of the ECG signal (reflecting the ECG of an individual) when affected by a number of different bit-flipping scenarios. Overall, 1 million samples or digital codes were obtained from the performed ECG signal A/D conversions, 700K of them were used as pre-deployment trusted codes with the remaining used as run-time untrustworthy codes. The fault scenarios and the corresponding match rates are shown in Table 5.1. The framework was implemented with a number of different variations of the ECG signal. The presented match rates are the average of match rates obtained from querying those variations, when affected
by the respective bit-flip scenarios, in the baseline trusted KDE model. When referring to Figure 5.6, which shows the severity versus match rate mapping related to the ECG signal, it could be seen that, for example, the first bit-flipping scenario with a match rate of around 26.7% implies a severity greater than 5, indicating a non-negligible amount of integrity degradation. Additionally, the severity versus match-rate mapping provides a clear indication of the significance of the targeted bit as when examining, for instance, the match rates of the fourth and fifth bit-flipping scenarios, it is evident that they belong to low-severity integrity degradation.

5.6.2 Use case 2: The water treatment system

The vital infrastructure applications of power transmission, oil refining, and water treatment and distribution, among many others, all employ industrial control systems (ICS) to oversee and control the physical processes carried out in these applications. A pre-requisite for the controlling process, which provides it with required inputs, is the sensing, measuring, and detecting of the instantaneous changes and quantities in parameters belonging to the specific application. As the quantities measured by ICS reflect real-life continuous quantities such as temperature, pressure, and the level and flow of fluids, ICS deploys sensors, analog front-ends, and ADCs for conducting this sensing and measuring. The output of the front-end or ADC in ICS is used in dictating the decisions made by the digital processing units and consequently the actions performed by the actuators in controlling the physical process of concern. Therefore, an attractive attack surface in ICS is the analog front-end and/or ADC as an attacker might use it in causing catastrophic potentially irreversible damage and destruction in an ICS while ensuring a small hard-to-detect attack footprint. An example of such attack is the one demonstrated in [327] where abnormal motor vibrations in an ICS system were induced by tricking the ADC in the system into converting abnormal signal frequencies.

To assess the effectiveness of our framework in detecting abnormal frequencies in the output of ICS ADCs, it was implemented on sensor readings from a secure water treatment system (SWAT) [337], after they were converted to digital and injected with various bit-flipping scenarios. The tested-on sensor readings are those of the LIT101 sensor, a level sensor for measuring and reporting the level of water in the raw water tank of the SWAT system, and those of the FIT301 sensor, a flow sensor for measuring the flow of water. Introducing bit-flips to these sensor readings emulates the case when
an attacker aims to cause a tank overflow or underflow, which is a malicious action capable of damaging the actuators; valves and pumps, connected to the tank. The SWAT dataset is based on seven days of normal operation, data from the first four days was considered trustworthy and used for the “pre-deployment” KDE modelling. Data from the latter three days was considered untrustworthy “run-time” data and was injected with bit-flips. The pre-deployment data consist of around 1.4M digital codes while the run-time data consist of 600k codes, for each of the sensors.

When querying the slopes obtained from the run-time bit-flipping effected LIT101 and the FIT301 sensor readings in the respective KDE models, similar match rates were obtained for both signals for the same bit-flipping scenario. Therefore, the match rates in Table 5.2 are the average of the rates obtained from both signals. Figures 5.9a and 5.9b show the same part of the LIT101 sensor readings in its baseline state and when it is affected with the first bit-flipping scenario (from Table 5.2), respectively, while Figures 5.10a and 5.10b show the respective ranges of slopes. The differences between the two slope distributions are visibly clear, which is reflected in the resultant match rates. These results are clear indicators of the effectiveness of implementing the proposed framework in detecting incorrectness and integrity degradation in the output of the analog font-end in ICS.

<table>
<thead>
<tr>
<th>Bit-flip scenario</th>
<th>Frequency</th>
<th>Affected bit</th>
<th>Match rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 1</td>
<td>50% of code (random)</td>
<td>bit 5</td>
<td>23.09%</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>50% of code (random)</td>
<td>bit 4</td>
<td>32.64%</td>
</tr>
<tr>
<td>Scenario 3</td>
<td>28% of code (random)</td>
<td>bit 3</td>
<td>63.52%</td>
</tr>
<tr>
<td>Scenario 4</td>
<td>14% of code (random)</td>
<td>bit 2</td>
<td>81.77%</td>
</tr>
<tr>
<td>Scenario 5</td>
<td>20% of code (random)</td>
<td>bit 1</td>
<td>84.96%</td>
</tr>
<tr>
<td>Scenario 6</td>
<td>1.4% of code (random)</td>
<td>bit 3</td>
<td>78.34%</td>
</tr>
<tr>
<td>Scenario 7</td>
<td>1.4% of code (random)</td>
<td>bit 6</td>
<td>38.09%</td>
</tr>
</tbody>
</table>
Figure 5.9: The baseline and bit-flip affected (scenario 1) LIT101

Figure 5.10: Slope histograms of the baseline and bit-flip affected (scenario 1) LIT101

5.6.3 Use case 3: The airplane fuel distribution system

Electronic systems deployed in aviation are some of the most safety-critical systems in our world as any operational failure in them, regardless of it originating from malicious or non-malicious means, would have dire immediate impact on human lives. One of the vital systems in airplanes is the aircraft fuel distribution system (AFDS), which is concerned with the storage and distribution of fuel in the aircraft. Typically, an AFDS consists of two engines, several fuel-storing tanks, valves to control the flow and direction of fuel transmission in the system, and jettison points used in releasing fuel from the system in emergency situations such as emergency landing. The AFDS monitors the state of fuel in the tanks via level and temperature sensors, and monitors the rate of fuel flow in the pipes using flow meters. An attacker could utilize the dependency of the control part of the AFDS on the quantities measured by the sensors in disrupting the fuel distribution and delivery process. Degrading the output of the analog front-end through, for instance, the introduction of abnormal bit-flips could give the impression of a lower or higher instantaneous fuel level or that of an untypical rate of fuel flow. This could in turn lead to misjudgment and impaired decision making.
The framework was tested with readings of AFDS level and flow sensors obtained from the dataset in [338]. This dataset is composed of sensor readings of normal trusted operation of the AFDS and the same sensor readings when obtained from certain attack scenarios. As we are concerned with a certain type of attacks or untrustworthy behaviour (i.e. bit-flipping scenarios), the sensor readings of the normal operation were used in the framework testing. The tests were performed on the readings of the level sensors of the front and the central tanks and the readings of the central tank to front tank flow sensor, denoted as FTL, CTL, and CLF, respectively. The interpolated sensor readings were converted to digital, and around 1.3M digital codes were obtained. The slopes were then calculated and used in the construction of the KDE models. The bit-flipping scenarios presented in Table 5.3 were introduced to the readings, which was followed by the calculation of the respective slopes and the KDE query process. The obtained match rates for the three sensors for each of the bit-flipping scenarios are as detailed in Table 5.3, and as each of the sensors exhibited different match rates for the same scenarios, they are presented separately. The baseline and bit-flipping affected (scenario 4) readings of the FTL sensor are presented in Figures 5.11a and 5.11b while Figures 5.12a and 5.12b show the respective slope histograms. In spite of the fact that both histograms peak at the same value, their ranges are significantly different as it is from -1 to 1 for the baseline signal while it goes beyond 50 for the bit-flipping case.

Table 5.3: The bit-flipping scenarios introduced to the AFDS FTL and CLF sensor readings and the obtained match rates

<table>
<thead>
<tr>
<th>Bit-flipping scenario</th>
<th>Frequency</th>
<th>Affected bit</th>
<th>Match rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 1</td>
<td>50% of code (continuous)</td>
<td>bit 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>FTL</td>
<td>22.22%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CLF</td>
<td>33.33%</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>50% of code (continuous)</td>
<td>bit 5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>FTL</td>
<td>28.57%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CLF</td>
<td>50%</td>
</tr>
<tr>
<td>Scenario 3</td>
<td>6% of code (random)</td>
<td>bit 3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>FTL</td>
<td>18%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CLF</td>
<td>30%</td>
</tr>
<tr>
<td>Scenario 4</td>
<td>25% of code (continuous)</td>
<td>bit 6</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>FTL</td>
<td>25%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CLF</td>
<td>60%</td>
</tr>
</tbody>
</table>
Figure 5.11: The baseline and bit-flip affected (scenario 4) FTL

Figure 5.12: Slope histograms of the baseline and bit-flip affected (scenario 4) FTL

5.7 Evaluation

Integrity degradation observed in the output of the analog front-end is a topic that has not gained sufficient attention in literature in spite of its potentially serious environmental, physical, and personal consequences. Integrity degradation does not need to be of significant immediate impact as we recognised a way that is realised through flipping one of the lower or middle bits of the signal at the analog-digital boundary of the SoC. Such type of integrity degradation is of subtle largely invisible nature, could be introduced with minimal footprint, and has similar impact to more complex frequency injection attacks. The run-time integrity monitoring and incorrectness detection framework proposed in this chapter was tested specifically with the untrustworthy behaviour of bit-flipping, however, the framework is in principle applicable with any type of integrity degradation that would introduce new slope values. In case of detecting attacks or faults that would alter the distribution of the slopes without adding new values, the framework could be adjusted to model the run-time slopes and statistically compare them with the baseline slope models.
Therefore, in addition to its primary objective of detecting incorrectness, the framework could also be adapted for performance-related testing and for the detection of errors, if any, in the A/D conversion process.

As the proposed approach is based solely on the output of the ADC or the front-end, and as the hardware part of the framework is implemented independently of the monitored front-end, independence from the architecture and design of the front-end is achieved. In addition to allowing the implementation of the framework in a device and application agnostic manner, this independence allows the adoption of this framework in integrity monitoring of all types of front-ends, whether implemented on-chip, or off-chip, outsourced, or designed and fabricated in-house. The framework, being oblivious to the internal characteristic of the front-end, could be easily adapted to different monitoring requirements. For example, the frequency of performing the run-time parts of the framework could be adjusted as needed, provided that the size of the storage elements is taken into account. Another aspect of adaptability is that decisions related to the trustworthiness levels and the corresponding alerting and countering mechanisms, are left to the discretion of the SoC integrator. In fact, the entirety of the second phase of the framework could be left out, if needed, for example, for some performance reasons, without affecting the monitoring and detection process. Moreover, the framework is easily adaptable to perform the monitoring with multiple relationships instead of focusing solely on a single relationship.

Computational lightweightness was achieved mainly by having the machine-learning and computational steps performed in software, but also by adopting a query-based approach that is of low computational intensity compared with, for example, the probability comparison approach. The only input needed for the monitoring and incorrectness detection process is the output of the front-end hence the proposed monitoring is autonomous provided that the type of the input signal (i.e. the signal sensed by the sensors) is pre-defined. Although the framework was developed with the main purpose of detecting malicious integrity degradation in the output of the analog front-end, it could be utilized in monitoring the impact of aging and environmental conditions on the operation of the front-end as reflected in its output integrity. Additionally, in case of systems with stringent power and area constraints, the proposed monitoring mechanism could be performed on an occasional basis with all steps performed in software. Moreover, the fact that the framework is implemented in a fully digital manner widens its scope and allows its usage in
integrity monitoring in any of the digital parts of the SoC where the monitored signal is expected to retain its temporal and arithmetic features such as; during on-chip transmission and communication.

5.7.1 Limitations

The proposed implementation of the framework is based on using hard-wired elements for the collection and saving of the temporal and arithmetic distances needed to calculate the slopes. Therefore, the implementation of changes in the frequency of the query process beyond the capacity of the storage elements, or changes in the type of extracted relationship need to be decided at design-time. Moreover, given that the monitoring is very specific to the type and shape of input signal, implementing it with ADCs with multiple input sources would require the integration of a mechanism that would inform the framework of the input source currently in-use, allowing the query process to be performed using the corresponding KDE models.

In principle, the framework could be used in detecting incorrectness in any type of signal, however, the presented relationships, including the slope, may not be suitable for signals of stochastic nature and non-defined shapes. In which case it is advisable that the SoC integrator, based on their knowledge of this signal, chooses a suitable characterizable relationship that could be extracted from the signal at run-time.

5.8 Conclusion

Sensors, ADCs, and analog signal conditioning elements are the bridge between our physical world and the world of computers. Without these elements, collectively referred to as the analog front-end, computers and processors cannot comprehend real-life quantities and phenomena, deeming them unusable in most, if not all, real-life applications in the fields of medicine, space and aviation, critical infrastructure, consumer electronics, and manufacturing, among many others. The research presented in this chapter was motivated by the realization that an untrustworthy or insecure analog front-end is an untrustworthy and insecure SoC, regardless of the trust and security levels of the digital part of the SoC. To this end, we explored the possibility of introducing subtle and distant integrity degradation to the output of the front-end compromising its security and deeming it untrustworthy. It was found that introducing such undesired behaviour is possible through a number of
identified trust issues such as hardware Trojans, fault injection attacks, and IP counterfeiting. The unintentional introduction of this behaviour is also possible by a hardware fault in, for instance, a poorly tested ADC. One way of subtle integrity degradation is the introduction of new frequencies or bit-flips to the output of the front-end leading to potentially significant confusion and disruption in the downstream digital systems.

Realizing the seriousness of incorrectness in the signals at the analog-digital interface of the SoC, we developed a run-time framework for the detection of such incorrectness. This framework shall serve as the last line of defence against integrity degradation by detecting its occurrence at run-time and escalating the issue to the higher levels in the system, that could in turn perform the appropriate mitigation actions such as isolated testing of the front-end elements. We base the framework on the notion that, when analog signals are converted to digital they retain their shape and defining characteristics, therefore, the correctness of their digital representation could be evaluated based on the extent to which these characteristics are preserved. Some of the defining characteristics of signals are based on the temporal and arithmetic relations between different instances or samples of these signals. The proposed framework extracts these relations from the output of the ADC in trusted pre-deployment operation and uses them in training KDE models that are used as references of correct conversion (and pre-conversion conditioning) of the signal of interest. During live operation of the SoC, the same relations are extracted from the output of the ADC and periodically queried in the KDE models. Based on the output of the query process, the rate at which the run-time relation values match those used in training the KDE models is calculated. This is the first phase of the framework, which is concerned with the detection of incorrectness. The second phase of the framework is concerned with assessing the severity of the integrity degradation, if any, and evaluating the trustworthiness of the analog front-end. Examples of characterizing relationships are distances between peaks, distances between certain codes, rate of change in the signal, and the slope. In the proposed implementation, the slope was chosen because of its universality and the fact it carries information about both the temporal and arithmetic changes in the signal.

A hybrid software/hardware implementation is adopted allowing the framework to be implemented in small lightweight systems as well as in industrial and infrastructure related applications. The framework is realized independently of the monitored front-end, hence it is suitable for monitoring outsourced and COTS
front-ends. As it is reliant only on analyzing the output of the front-end, the framework is device, application, and architecture agnostic, and is autonomous in its operation. Adaptability was taken into account when developing the framework as the frequency of performing the detection, the relationship that this detection is based on, and the trustworthiness evaluation process are largely left to the discretion of the IC designer or SoC integrator.
Chapter 6

Conclusions

The goal of this thesis is to realistically and effectively respond to the issue of untrustworthy hardware, not by eliminating the sources of untrustworthiness as they are often unavoidable, but rather by live monitoring and detection of actions or behaviours identified as a breach of trust. We consider this monitoring the last line of defence against untrustworthy hardware behaviour as it provides the final assurance of the level of trust in the operations and communications carried out in an SoC.

In chapter 2, we provided a literature review on major hardware trust issues and the countermeasures developed to address them, primarily at design and pre-deployment stages as this is the focus of the majority of literature in the field. We introduced a definition for hardware trust presenting it in terms of the conformance of actual observed hardware behaviour to the expected and specified hardware behaviour. A mismatch between the run-time behavior and the expected behaviour is consequently recognized as untrustworthy behaviour, which could also be defined as any behaviour that deviates from the expected behaviour of an IP or IC. Untrustworthy behaviour is not necessarily of serious and direct security implication but given its deviation from normal behaviour, it could affect the system’s security properties (detailed in chapter 2), and may be indicative of a hardware attack such as HTs or fault injection attacks, a hardware trust issue such as counterfeiting, a hardware fault, or a performance issue.

Different types of untrustworthy behaviour were targeted (by analysis and consequent developing of run-time monitoring and detection techniques) in this thesis ranging from direct and obvious violations to normal and expected behaviour (chapter 3) to obscured physical and operational vulnerability-exploiting behaviour (chapter 4) to subtle hard-to-notice behaviour (chapter 5). The targeted behaviours
were mainly chosen to show the wide range of potential realizations of untrustworthy
behaviour, and that even apparently trusted and “harmless” outputs or actions could in
reality be malicious and unworthy of trust. The fact that the presented behaviours
could all be caused by a range of trust issues shows the seriousness of the issue of
untrustworthy behaviour. This seriousness partially lies in the fact that the elimination
of some relevant causes does not automatically guarantee the elimination of the
possibility of such behaviour. Another factor to the seriousness of the presented
untrustworthy behaviours is their potential manifestation and induction in
fundamental and ubiquitous on-chip elements such as; communication structures,
memory systems, and analog front-ends, as demonstrated in this thesis. The research
presented in chapters 3 to 5 showed with theoretical and experimental evidence the
plausibility of using the targeted untrustworthy behaviours in undermining and
compromising the dependability, availability, and integrity of the targeted ubiquitous
SoC elements. This thesis, therefore, demonstrates the vulnerability of conventional
IC designs to untrustworthy behaviour and the realization that this is an issue that is
yet to be granted the research attention it, and its potentially serious impact, deserve.

The range of potential untrustworthy behaviours, the multitude of ways in which
they could be induced, and their potentially serious long-term cumulative
consequences (in case the short-term consequences are not significant) require us to
start viewing untrustworthy behaviour as a serious event, that on its own, is worthy of
detection and mitigation, regardless of what could have caused it. Having reached this
conclusion, the development and deployment of run-time monitoring presents itself as
a substantial pre-requisite to the trust and security of hardware. Although run-time
monitoring of hardware, from the trust and security perspective, have been discussed
in published literature, its applicability and adaptability is limited to certain
applications and/or is dependent on impractical assumptions. Recognizing this issue,
this thesis aims to bridge some of the gaps between the theory and application of
run-time trust and security monitoring by introducing three different approaches to
run-time monitoring and untrustworthy behaviour detection.

By developing three different schemes for the run-time monitoring and detection
of untrustworthy behaviour, we aim to present a step forward in the road towards the
ultimate goal of being able to, with high degree of confidence, guarantee the
trustworthiness of computations and operations irrespective of the trustworthiness of
the underlying silicon. The developed schemes, being capable of detecting a range of
untrustworthy behaviours at run-time, but also being independent of untrustworthy IP
providers, device and application agnostic, and adaptable to different situations and implementations, show the feasibility and effectiveness of approaching hardware trust from the angle of run-time monitoring and detection. As the schemes are not based on certain attack vectors and as they are concerned with detecting abnormal or anomalous behaviours, rather than their causes, the schemes could also be utilized for performance or fault monitoring purposes. Although each of the schemes was developed and implemented in a distinct manner, they all stem from the realization of the extent of vulnerability of SoCs and ICs to the run-time induction of untrustworthy behaviour. This is especially the case when the offending element (i.e. source of untrustworthy behaviour) is a non-processor and non-crypto SoC element, as in conventional IC and SoC designs the run-time trust and security of the actions and behaviour of such elements are rarely monitored and verified.

To the best of our knowledge, the schemes presented in this thesis are the first to target the run-time detection of the presented untrustworthy behaviours, and to experimentally show the feasibility of utilizing run-time monitoring in the detection of such behaviours when initiated from fundamental and ubiquitous on-chip elements such as; communication interconnects, memory systems, and analog front-ends. The first scheme, which is presented in chapter 3, effectively identifies and blocks attempts of data or transaction injection, shadowing, redirection, or modification when attempted by on-chip communication structures, preventing consequent confidentiality, integrity, or availability compromise. By detecting obscured latency extensions, the second scheme, presented in chapter 4, allows the early detection of prolonged denial-of-service attacks and/or early degradation in the performance of the monitored element. This allows the SoC integrator to intervene and react to the obscured availability compromise (as appropriate) in a timely manner, before more serious availability or performance implications are observed. By tackling untrustworthy behaviours leading to integrity degradation or incorrectness in the output of the analog front-end, the third scheme, presented in chapter 5, addresses a serious trust issue that, if successfully realized, is capable of inducing catastrophic and potentially life-threatening consequences. As the scheme is based solely on analyzing and verifying the features of the front-end output, it provides an effective lightweight approach to the run-time detection of incorrectness in this output without the need to deploy individual, potentially complex, monitors at the output of each of the elements in the front-end.

Our first focus (chapter 3) is explicit and direct breach or violation of the normal
Chapter 6. Conclusions

operation and behaviour of a given on-chip element. The targeted elements are on-chip communication constructs initiating untrustworthy behaviour of the potential shapes of illegitimate redirection or re-transmission of packets, modification of transferred data, or illegitimate access to on-chip resources. Hardware-based wrappers are proposed, as the first core contribution of this thesis, to monitor for the occurrence of such untrustworthy behaviour through progressive transaction logging and verification, or what we identify as the principles of component guarding, data tagging, and event verification. The tested-on on-chip communication construct is the ubiquitous AXI interconnect.

Chapter 4 is concerned with exposing the potential vulnerability posed by inherent physical, architectural, or operational characteristics of some electronic elements. To this end, we focused on the well-known issue of DRAM latency variability and unpredictability investigating whether it presents a trust and security issue in addition to it being an identified performance issue. Our experiments showed the possibility of obscuring abnormal latency extensions within the variability of DRAM latency. Therefore, a machine-learning framework for the detection of such latency extensions at run-time is introduced as the second core contribution of this thesis. The framework adopts the notion of latency characterization and modeling based on the workload and the memory access type during trusted operation, and using the resultant models as references in statistical comparisons with run-time latency models.

In chapter 5 another type of hidden and subtle untrustworthy behaviour is studied. This behaviour is represented by abnormal and unexpected bit-flips or stuck-at-faults appearing in the middle and lower bits of the output of the analog front-end. Our approach to the run-time detection of these faults, which is the third core thesis contribution, is based on deducing the integrity and correctness of the front-end output by extracting temporal and arithmetic relations between its samples. Density estimation machine-learning is used to model these relations when obtained from trusted output. This is followed by the extraction of the same relations at run-time and projecting them on the trusted models to infer the extent to which expected signal features are retained.

All of the three schemes satisfy the objectives of independence, autonomy, adaptability, and computational lightweightness that were laid out in chapter 1. Independence was achieved by the adoption of generic monitoring mechanisms that are not tied to a certain design, model, or architecture of the monitored elements. The purpose of including autonomy as an objective was to avoid the need to interact with
potentially untrustworthy IPs in the monitoring process, leading us to be suspicious of the monitoring process itself. Autonomy was achieved by basing the introduced techniques and frameworks on the inputs and outputs of the monitored elements, and on deductions made solely through the careful and continuous observation of the interfaces of these elements. In the first monitoring technique that is concerned with communication interconnects (chapter 3), computational lightweightness was sought by primarily basing the technique on non-computational checks such as logging and correlation of events. In the other two frameworks (chapters 4 and 5), computational lightweightness was achieved by proposing a hybrid software/hardware implementation where the computationally-intensive machine-learning and statistical comparison methods are performed in software, and the lightweight data collection performed in hardware. Although the schemes and frameworks introduced in this thesis were designed to be of individual low implementation cost, their collective integration in a given SoC may incur undesired amount of power, area, or performance overhead. This could especially be the case for integration in Small lightweight SoCs, such as those used in IoT devices. It is therefore advisable that the decision to integrate such run-time monitoring techniques is taken based on informed assessment of the possible trust and security risks associated with the specific application of the SoC of concern. For example, in safety critical fields, such as industrial control systems and avionics, implementing run-time measures to ensure that each on-chip element operates within its normal and trusted boundaries at all times is likely a priority, that if ignored, could lead to serious consequences. Another aspect to consider is the likelihood of the occurrence of a certain untrustworthy behaviour based on factors such as the extent of which untrustworthy actors are involved in the development and fabrication of a given SoC and the environmental conditions in which the SoC is deployed. For example, the greater the number of 3PIPs in a SoC or the more exposed it is to external actors in its deployment environment, the greater the possibility of the occurrence of untrustworthy behaviour at run-time and the greater the need to deploy run-time security and trust monitoring techniques. Moreover, a trade-off between the cost of implementing run-time monitoring and the risk of not doing so could be sought by, for instance, performing the monitoring only on a partial basis or on certain types of transactions and operations as discussed in Sections 3.6, 4.7, and 5.7. By presenting different approaches to run-time trust monitoring, we also aim to show that this type of monitoring could be tackled from different angles depending on the untrustworthy
behaviour of concern and its potential consequences (i.e. short term consequences requiring immediate detection and counter-action or long term consequences that could be detected on a periodic bases), the offending element, the trust and security needs of the SoC, and the area and performance budget allowed for the monitoring technique.

6.1 Lessons learned and future directions

In addition to introducing run-time monitoring and trustworthy computing (on untrustworthy components or hardware) as the last line of defence against untrustworthy behaviour observed at run-time, through this thesis, we aim to bring to the attention of the research community a number of realizations that have motivated or were deduced from the presented research:

- Hardware security should not be defined and evaluated solely by processor security as all on-chip elements are vulnerable and are prone to various shapes of trust and security exploitation, as shown in chapters 2 to 5. And even though, subtle untrustworthy behaviour initiated from an on-chip non-processing element may not seem of significant impact on the short-term healthy operation of the SoC, its long-term cumulative impact could lead to a serious compromise of security. This is specifically evident when examining the bit-flipping affected ADC outputs presented for the three use-cases in chapter 5, in which intermittent individually-unnoticeable bit-flips resulted in significant and potentially dangerous distortion in the affected signals.

- Pre-deployment attack-specific trust and security measures and tests are not necessarily sufficient and more research is needed in the realm of run-time monitoring. In the case of hardware, where post-deployment patching is, in most cases, challenging if not impossible, systems need to be capable of live detection of untrustworthy undesired behaviour regardless of its potential cause. We also need to be aware of the best possible way, given certain circumstances of a SoC application, to act upon detected untrustworthy behaviours with preferably pre-defined mitigation and triage plans and processes. The framework proposed in chapter 5 presents an example of a post-detection trustworthiness assessment plan and consequent devising of appropriate mitigation or countering mechanisms.
6.1. Lessons learned and future directions

- More generic adaptable hardware trust and security measures should be developed as many IC design-houses do not have the resources nor the experience and knowledge to develop trust and security measures from scratch. Developing “plug-and-play” types of measures with minimal design-effort required of the design-house is also of utmost importance. This realization was taken into account in the development of the three proposed schemes and was the main motivation behind developing them in an adaptable and generic manner.

- Security should be an enabler and facilitator of technology advancement and not an obstacle in its way, therefore, run-time monitoring and trustworthy computing techniques are now needed more than ever to allow worry-less integration of state-of-the-art outsourced IPs.

- Developers of hardware trust and security mechanisms need to be aware of (and avoid) the trap of implicitly trusting the untrustworthy. This could happen, for example, in the case of adopting trust and security measures that can only be implemented inside untrustworthy IPs or are reliant on their internal states as is the case with some of the measures surveyed in chapter 2.

- A breach of trust is not necessarily related to untrustworthy IP providers or foundries as is commonly presented in literature. It could, in the matter fact, originate from IPs designed and fabricated in-house but are exposed during operation to, for example, potential sources of fault injection attacks or adverse environmental conditions, leading these IPs to show violation of normal behaviour (as shown in the literature review in chapter 2). It is, therefore, fundamental to not consider such “isolated” IPs immune to untrustworthy behaviour and its potential consequences.

- Run-time monitoring techniques developed for trust and security purposes could also be utilized in performance assurance and monitoring purposes, and vice versa. Therefore, IC designers should look for potential ways to merge the two, minimizing overhead wherever possible.

The area of run-time monitoring and detection of untrustworthy behaviour, although has been discussed in literature for a number of years, it is from the practice and application perspective, still in its infancy. Run-time monitoring techniques for
the majority of on-chip elements are yet to be developed, and there are probably many potential trust issues waiting to be discovered, investigated, studied, analyzed, and eventually addressed, examples include:

- The possibility of utilizing known physical semiconductor phenomena, by adversaries, in initiating untrustworthy behaviour.

- The potential vulnerability posed by operational and architectural features of non-processing on-chip elements and peripherals.

- Subtle low footprint untrustworthy behaviour that could be introduced in a nearly invisible manner.
Appendix A

Supplementary material

A.1 Algorithms

In the following are the algorithms performed by the wrappers proposed in chapter 3 in transaction and event logging and verification. The shown algorithms are those concerned with read transactions, however, write transactions go through the same process with the switching of the read data algorithms as the source of the data in this case is the master IP. Write transactions also include a write response stage where W(AXI) checks if the response forwarded to M_i from the interconnect is actually the same as the one sent by the relevant S_j. Additionally, the algorithms for the role of W(S_m) in address read and that of W(M_n) in data read are not included as they are concerned mainly with signal forwarding.
Algorithm 1: W(Mi) address read

1 Input: $m_i$arvalid; // arvalid from $M_i$
2 Input: $m_i$araddr; // address from $M_i$
3 Input: $m_i$ar_read; // other address read signals from $M_i$
4 Input: waxi$m_i$arready; // arready from W(AXI)
5 Output: W($m_i$)arready; // arready to $M_i$
6 Output: W($m_i$)arvalid; // arvalid to W(AXI)
7 Output: W($m_i$)araddr; // address to W(AXI)
8 Output: W($m_i$)ar_read; // other address read signals to W(AXI)
9 Output: W($m_i$)invalid_transaction; // invalid transaction (alert)

IC designer input (Parameter) min allowed address;
IC designer input (Parameter) max allowed address;
reg [AR SIZE-1:0] allowed_ar_address; // AR SIZE = 2 * address size
reg valid_address; // address within allowed range
allowed_ar_address = { min allowed address , max allowed address };

begin Read address: synchronous state machine

if $m_i$arvalid = 1 then
    if allowed_ar_address [31:0] $m_i$araddr = allowed_ar_address [63:32]
    then
        W($m_i$)arvalid = 1;
        W($m_i$)araddr = $m_i$araddr;
        W($m_i$)ar_read = $m_i$ar_read;
        valid_address = 1;
    else
        W($m_i$)invalid_transaction = 1;
end

if waxi$m_i$arready then
    if valid_address = 1; then
        W($m_i$)araddr =1;
    end
end
end
Algorithm 2: W(AXI)$_i$ unit and relevant part of transaction log management unit - address read

1. **Input:** W($m_i$)$_i$.arvalid; // arvalid from W($M_i$)
2. **Input:** W($m_i$)$_i$.araddr; // address from W($M_i$)
3. **Input:** W($m_i$)$_i$.arid; // transaction id from W($M_i$)
4. **Input:** W($m_i$)$_i$.arlen; // transaction length from W($M_i$)
5. **Input:** W($m_i$)$_i$.ar.read; // other address read signals from W($M_i$)
6. **Input:** axi.$m_i$.arready; // arready from AXI interconnect
7. **Input:** slave.$j$.ready; // from W(AXI)-slave$_j$ unit
8. **Output:** waxi.$m_i$.arready; // arready to W($M_i$)
9. **Output:** W(AXI)$_i$.arvalid; // arvalid to AXI interconnect
10. **Output:** W(AXI)$_i$.araddr; // address to AXI interconnect
11. **Output:** W(AXI)$_i$.arid; // transaction id to AXI interconnect
12. **Output:** W(AXI)$_i$.arlen; // transaction length to AXI interconnect
13. **Output:** W(AXI)$_i$.ar.read; // other address read signals to AXI interconnect
14. **Output:** W(AXI)$_i$.m$_i$.invalid.transaction; // invalid transaction

(alert)

reg slave.id; // ID of slave IP
reg length; // actual transaction length (number of beats)
reg [TR_SIZE-1:0] artr_log [0: MAX_TR_NO]; // transaction logging memory of "TR_SIZE" width and "MAX_TR_NO" length
reg [TR_SIZE-1:0] artr_log.entry; // transaction log entry
reg artr_log.update; // transaction log update signal
reg status_bits; // status of transaction

parameter TR_SIZE, MAX_TR_NO; integer k=0;
begin Read address: synchronous state machine

if \( W(m_i)_{arvalid} = 1 \) then

\[ \begin{align*}
&W(AXI)_{arvalid} = 1; \\
&W(AXI)_{araddr} = W(m_i)_{araddr}; \\
&W(AXI)_{arid} = W(m_i)_{arid}; \\
&W(AXI)_{arlen} = W(m_i)_{arlen}; \\
\textbf{Decode} W(m_i)_{araddr} \text{ to slave id; } // \text{ to determine which} \\
&\text{length} = W(m_i)_{arlen} + 1; \\
&\text{status_bits} = 00; // \text{new transaction} \\
&\text{artr_log_update} = 1; \\
&\text{artr_log_entry} = \{W(m_i)_{arid}, W(m_i)_{araddr}, \text{length}, \text{status_bits}\}
\end{align*} \]

end

if \( \text{artr_log_update} = 1 \) then

\[ \begin{align*}
&\text{if } i < \text{MAX_TR_NO} \text{ then} \\
&\text{artr_log}[k] = \text{artr_log_entry}; \\
&k = k+1;
&\text{else} \\
&k = 0;
&\text{end}
\end{align*} \]

end

if \( \text{axi_m_iarready} = 1 \) then

\[ \begin{align*}
&\text{if } \text{slave_j_ready} = 1 \text{ then} \\
&\text{waxi_m_iarready} = 1; \\
&\text{else} \\
&W(AXI)_{m_i_{invalid_transaction}} = 1;
&\text{end}
\end{align*} \]

else

\end
Algorithm 3: W(AXI)S$_j$ unit - address read

1. **Input:** AXI_arvalid; // arvalid from AXI interconnect
2. **Input:** AXI_araddr; // address from AXI interconnect
3. **Input:** AXI_arid; // transaction id from AXI interconnect
4. **Input:** AXI_arlen; // transaction length from AXI interconnect
5. **Input:** AXI_ar_read; // other address read signals from AXI interconnect
6. **Input:** W(S$_j$)_arready; // arready from W(S$_j$)
7. **Input:** logged_address; // address from atrt_log
8. **Output:** W(AXI)$_s_j$arready; // arready to W(S$_j$)
9. **Output:** W(AXI)$_s_j$arvalid; // arvalid to W(S$_j$)
10. **Output:** W(AXI)$_s_j$araddr; // address to W(S$_j$)
11. **Output:** W(AXI)$_s_j$arid; // transaction id to W(S$_j$)
12. **Output:** W(AXI)$_s_j$arlen; // transaction length to W(S$_j$)
13. **Output:** W(AXI)$_s_j$ar_read; // other address read signals to W(S$_j$)
14. **Output:** slave$_j$_ready; // to W(AXI) master units
15. **Output:** W(AXI)$_s_j$invalid_transaction; // invalid transaction (alert)

    reg valid_address; // logged address = received address
begin Read address: synchronous state machine
| if AXI_arvalid = 1 then
|      Send info request to artr_log management unit with AXI_arid;
|      if AXI_arid found in artr_log then
|          if AXI_araddr = logged_address then
|              W(AXI)_sj_arvalid = 1;
|              W(AXI)_sj_araddr = AXI_araddr;
|              W(AXI)_sj_arid = AXI_arid;
|              W(AXI)_sj_ar_read = AXI_ar_read;
|              valid_address = 1;
|          else
|          end
|      else
|      end
| end
if W(Sj).arready = 1 then
    if valid_address = 1;
    then
        W(AXI).sj_arready = 1;
        slave_j_ready = 1;
    end
end
Algorithm 4: \( W(S_j) \) - data read

1. **Input:** \( W(\text{AXI})_j \_\text{rready}; // \text{rready from } W(\text{AXI}) \\
2. **Input:** \( S_j\_\text{rvalid}; // \text{rvalid from } S_j \\
3. **Input:** \( S_j\_\text{rid}; // \text{rid from } S_j \\
4. **Input:** \( S_j\_\text{rdata}; // \text{rdata from } S_j \\
5. **Input:** \( S_j\_\text{rlast}; // \text{rlast from } S_j \\
6. **Input:** \( S_j\_\text{rresp}; // \text{rresp from } S_j \\
7. **Output:** \( W(S_j)\_\text{rready}; // \text{rready to } S_j \\
8. **Output:** \( W(S_j)\_\text{rvalid}; // \text{rvalid to } W(\text{AXI}) \\
9. **Output:** \( W(S_j)\_\text{rdata}; // \text{rdata to } W(\text{AXI}) \\
10. **Output:** \( W(S_j)\_\text{rlast}; // \text{rlast to } W(\text{AXI}) \\
11. **Output:** \( W(S_j)\_\text{data\_read}; // \text{other data read signals to } W(\text{AXI}) \\
12. **Output:** \( \text{data\_tag}; // \text{data tag to } W(\text{AXI}) \\
13. **Output:** \( \text{data\_count}; // \text{data count to } W(\text{AXI}) \\
14. **Output:** \( W(S_j)\_\text{invalid\_data}; // \text{invalid data (alert)} \\

\begin{align*}
\text{reg data\_count\_current; // data count from data\_count\_log}
\end{align*}
begin Read data: synchronous state machine
    if $W(AXI)_rready = 1$ then
        $W(S_j)_rready = 1$;
    end

    if $S_j_rvalid = 1$ then
        $W(S_j)_rvalid = 1$;
        $W(S_j)_rdata = 1$;
        $W(S_j)_data_read = S_j.data_read$;
        Search for $S_j_rid$ in data_count_log$^1$
        if $S_j_rid$ found then
            Calculate data_tag;
            data_count = data_count_current;
        else
            $W(S_j)_invalid_transaction = 1$;
        end
    end

    if $S_j_rlast = 1$ then
        $W(S_j)_rlast = 1$;
    end
end

$^1$ data_count_log is an array in $W(S_j)$ where the ids of open transactions and the number (count) of beats sent so far for a given transaction are saved.
Algorithm 5: W(AXI)S\textsubscript{j} unit - data read

1. **Input:** AXI\_S\textsubscript{j}.rready; // rready from AXI interconnect
2. **Input:** master\_i.rready; // rready from W(AXI) master units
3. **Input:** W(S\textsubscript{j}).rvalid; // rvalid from W(S\textsubscript{j})
4. **Input:** W(S\textsubscript{j}).rid; // rid from W(S\textsubscript{j})
5. **Input:** W(S\textsubscript{j}).rdata; // rdata from W(S\textsubscript{j})
6. **Input:** W(S\textsubscript{j}).rresp; // response from W(S\textsubscript{j})
7. **Input:** W(S\textsubscript{j}).rlast; // rlast from W(S\textsubscript{j})
8. **Input:** data\_tag; // data tag from W(S\textsubscript{j})
9. **Input:** data\_count; // data count from W(S\textsubscript{j})
10. **Output:** W(AXI)\_S\textsubscript{j}.rvalid; // rvalid to AXI interconnect
11. **Output:** W(AXI)\_S\textsubscript{j}.rlast; // rlast to AXI interconnect
12. **Output:** W(AXI)\_S\textsubscript{j}.rid; // rid to AXI interconnect
13. **Output:** W(AXI)\_S\textsubscript{j}.rdata; // rdata to AXI interconnect
14. **Output:** W(AXI)\_S\textsubscript{j}.rresp; // response to AXI interconnect
15. **Output:** W(AXI)\_S\textsubscript{j}.rready; // rready to W(S\textsubscript{j})
16. **Output:** W(AXI)\_S\textsubscript{j}.invalid\_data; // invalid data (alert)

reg [DATA\_SIZE-1:0] rdata\_log [0: MAX\_DATA\_NO]; // Data logging
memory of "DATA\_SIZE" width and "MAX\_DATA\_NO" length

reg [DATA\_SIZE-1:0] rdata\_log\_entry; // data log entry
reg rdata\_log\_update; // data log update signal
reg last\_bit; // 1 if last beat in transfer
begin Read data: synchronous state machine

if $AXI_{S_j}.rready = 1$ then
  if $master_{i}.rready = 1$ then
    $W(AXI)_{S_j}.rready = 1$;
  else
    $W(AXI)_{S_j}.invalid\_data = 1$;
  end
end

if $W(S_j).rvalid = 1$ then
  $W(AXI)_{S_j}.rvalid = 1$;
  $W(AXI)_{S_j}.rid = W(S_j).rid$;
  $W(AXI)_{S_j}.rdata = W(S_j).rdata$;
  $W(AXI)_{S_j}.rresp = W(S_j).rresp$;
  $W(AXI)_{S_j}.rlast = W(S_j).rlast$;
  if $W(S_j).rlast = 1$ then
    last\_bit = 1
  end
  rdata\_log\_entry = \{ W(S_j).rid, last\_bit, data\_count, data\_tag, W(S_j).rresp \};
  rdata\_log\_update = 1;
end

if rdata\_log\_update = 1 then
  if $l < MAX\_DATA\_NO$ then
    rdata\_log[l] = rdata\_log\_entry;
    $l = l+1$;
  end
  $l=0$;
end
Algorithm 6: W(AXI)Mₙ unit - data read

1. Input: W(Mₙ).rready; // rready from W(Mₙ)
2. Input: AXI(Mₙ).rvalid; // rvalid from AXI interconnect
3. Input: AXI(Mₙ).rid; // rid from AXI interconnect
4. Input: AXI(Mₙ).rdata; // rdata from AXI interconnect
5. Input: AXI(Mₙ).rresp; // response from AXI interconnect
6. Input: AXI(Mₙ).rlast; // rlast from AXI interconnect
7. Input: data_count; // count from data_log
8. Input: calculated_data_tag; // data_tag calculated from AXI(Mₙ).rdata
9. Input: status_bits; // status_bits from artr_log
10. Output: masterₙ.rready; // rready to W(AXI) slave units
11. Output: W(AXI).Mₙ.rready; // rready to AXI interconnect
12. Output: W(AXI).Mₙ.rvalid; // rvalid to W(Mₙ)
13. Output: W(AXI).Mₙ.rid; // rid to W(Mₙ)
15. Output: W(AXI).Mₙ.rresp; // response to W(Mₙ)
16. Output: W(AXI).Mₙ.rlast; // rlast to W(Mₙ)
17. Output: expected_count; // expected number/ count of current data transfer to W(AXI) logs management units
18. Output: W(AXI).Mₙ.invalid_data; // invalid data (alert)

reg last_count; // number/ count of last data transmission from data_count_log
reg valid_data_request; // request for valid data
reg data_output; // valid data
begin Read data: synchronous state machine
    if \( W(M_i).rready = 1 \) then
        \( W(AXI).M_i.rready = 1; \)
        master_i.rready = 1;
    end
    if \( AXI.M_i.rvalid = 1 \) then
        Search for \( AXI.M_i.rid \) in data_count_log;
        expected_count = last_count + 1;
        Send request to artr_log and data_log management units with
        \( AXI.M_i.rid \) and expected_count;
        if \( AXI.M_i.rid \) & expected_count found in logs then
            valid_data_request = 1;
        else
            \( W(AXI).M_iinvalid.data = 1; \)
        end
    end
    if valid_data_request = 1 then
        Obtain transaction information from artr_log and data_log;
        if status_bits = 00 data_count = 1 then
            status_bits = 01; // first data transfer for this
            data_output = 1;
        end
        else if status_bits = 01 data_count > 1 then
            data_output = 1;
        end
    end
end
if data_output = 1 then
    Calculate data tag from AXI_M_i_rdata;
    if calculated_data_tag = data_tag;
        then
            if W(AXI)_Mi_rresp = data_response then
                W(AXI)_Mi_rvalid = 1;
                W(AXI)_Mi_rid = AXI_M_i_rid;
                W(AXI)_Mi_rdata = AXI_M_i_rdata;
                W(AXI)_Mi_rresp = AXI_M_i_rresp;
            else
                W(AXI)_Mi_invalid_data = 1;
            end
        else
            W(AXI)_Mi_invalid_data = 1;
        end
    else
        W(AXI)_Mi_invalid_data = 1;
    end
else
    W(AXI)_M_i_rlast = 1;
if AXI_M_i_rlast = 1 then
    if last_bit = 1 then
        W(AXI)_M_i_rlast = 1;
        status_bits = 11;
    end
end
end
A.2 Data

Figure A.1 shows the ECG signal and its parts. The following table presents the different ECG signal variations that were used in the ECG use-case presented in chapter 5 [339]. Two of the variations are not included in the table as one is based on lower QRS amplitude (obesity case) and the other is based on raised elevation of the signal (early re-polarization case). Each of the presented variations was simulated assuming 5 different potential heart rates between 60bpm and 100bpm, which resulted in 50 variations overall.

![ECG Signal Diagram]

Figure A.1: The ECG signal

<table>
<thead>
<tr>
<th>ECG variation</th>
<th>condition/ case</th>
<th>PR interval</th>
<th>QRS interval</th>
<th>QT interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>low normal</td>
<td>0.12s</td>
<td>0.06s</td>
<td>0.35s</td>
</tr>
<tr>
<td>2</td>
<td>mid normal</td>
<td>0.16s</td>
<td>0.09s</td>
<td>0.38s</td>
</tr>
<tr>
<td>3</td>
<td>high normal-male</td>
<td>0.20s</td>
<td>0.12s</td>
<td>0.40s</td>
</tr>
<tr>
<td>4</td>
<td>high normal-female</td>
<td>0.20s</td>
<td>0.12s</td>
<td>0.44s</td>
</tr>
<tr>
<td>5</td>
<td>short QT syndrome</td>
<td>0.16s</td>
<td>0.10s</td>
<td>0.30s</td>
</tr>
<tr>
<td>6</td>
<td>long QT syndrome</td>
<td>0.16s</td>
<td>0.10s</td>
<td>0.48s</td>
</tr>
<tr>
<td>7</td>
<td>QRS widening</td>
<td>0.14s</td>
<td>0.14s</td>
<td>0.36s</td>
</tr>
<tr>
<td>8</td>
<td>first-degree AV block</td>
<td>0.22s</td>
<td>0.08s</td>
<td>0.38s</td>
</tr>
</tbody>
</table>
Bibliography


circuit part to be protected of an integrated circuit,” 2010, patent No.

T. Porteboeuf, “Random Active Shield,” in 2012 Workshop on Fault Diagnosis

A. Milchior, D. Naccache, and T. Porteboeuf, “3D Hardware Canaries,” in
Cryptographic Hardware and Embedded Systems – CHES 2012, E. Prouff and
1–22.

S. Bhasin, “Cryptographically Secure Shield for Security IPs Protection,” IEEE

Backside Protection Against Physical Attacks on Secure Chips or SiP,” in 2018
IEEE 68th Electronic Components and Technology Conference (ECTC), 2018,
pp. 515–520.

[217] T. Miki, M. Nagata, H. Sonoda, N. Miura, T. Okidono, Y. Araga, N. Watanabe,
H. Shimamoto, and K. Kikuchi, “Si-Backside Protection Circuits Against

detection,” in 2012 IEEE International Symposium on Hardware-Oriented

Area Probing Detector as a Countermeasure Against Invasive Attacks,” IEEE

[220] N. Miura, D. Fujimoto, D. Tanaka, Y.-i. Hayashi, N. Homma, T. Aoki, and
M. Nagata, “A local EM-analysis attack resistant cryptographic engine with
fully-digital oscillator-based tamper-access sensor,” in 2014 Symposium on

Sage, J. Clediere, and A. Tria, “Efficiency of a glitch detector against
electromagnetic fault injection,” in 2014 Design, Automation & Test in Europe
Conference & Exhibition (DATE), 2014, pp. 1–6.


