Politecnico di Milano
Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica

TROJAN-FREE FPGA CIRCUITS USING ECC-BASED FUNCTIONAL TRUST-CHECKING

Relatore: Prof. Marco Domenico Santambrogio

Tesi di Laurea di:
Marco Maggioni
Matricola n. 706986

Anno Accademico 2009-2010

Dedicato alla mia famiglia
per il passato, il presente ed il futuro

La science est le tronc d'un baobab
qu'une seule personne ne peut embrasser
Proverbio, Togo

Sommario

Le *Field Programmable Gate Array* (FPGA) combinano la programmabilità di un processore a prestazioni sempre più vicine a circuiti ASIC customizzati. Nell' ultimo decennio, le FPGA hanno raggiunto un livello di costo e prestazione tale da rappresentare una soluzione attraente per delicate applicazioni militari o commerciali caratterizzate da un basso volume di produzione [1]. Questo genera un nuovo problema chiave relativo all' attendibilità del circuito FPGA presente nel campo operativo poichè il convenzionale processo di design è vulnerabile ad inserimenti di circuiti malevoli comunemente conosciuti come Hardware Trojan, i quali possono, sotto specifiche condizioni, portare a cambiamenti funzionali e/o fallimenti catastrofici per la delicata applicazione associata.

Al fine di ridurre il pericolo di alterazioni circuitali o inserimento di Trojan, risulta necessaria l'introduzione dell'idea di *trusted FPGA design* secondo cui un circuito deve svolgere solo la funzionalità per cui esso è stato originariamente progettato, niente di più e niente di meno. Questa idea deve essere fatta rispettare durante tutta la vita del circuito, dalla sua progettazione alla suo uso applicativo. Intuitivamente, le FPGA sono intrinsecamente progettate per cambiare la loro funzionalità e ovviamente questa malleabilità introduce delle vulnerabilità. L'integrazione on-chip di crittografia o altri sistemi di sicurezza può proteggere contro copie non autorizzate del sottostante design circuitale ma risulta insufficiente per immunizzarsi da inserimenti di circuiti malevoli. Inoltre, altre tecniche alternative di sicurezza presenti in letteratura sono relativamente inefficienti poichè nessuna di esse copre tutte le possibili vulnerabilità presenti nel ciclo di vita di un circuito basato su FPGA.

Da queste premesse deriva la necessità di una tecnica per il rilevamento di Trojan più comprensiva in modo tale che copra tutto il ciclo di vita del circuito. Da questa necessità scaturisce poi la principale motivazione per questo lavoro di tesi. Un trust-checking esplicito per circuiti basati su FPGA può garantire con probabilità molto alta che la funzionalità sia quella per cui i circuiti sono stati originariamente progettati. Inoltre, la tecnica utilizzata deve essere basata sulla funzionalità e deve operare on-chip in modo tale da poter rilevare ogni alterazione intenzionale o inserimento di Trojan. Questa tesi si preoccupa quindi del progetto di trusted FPGA circuits adottando un meccanismo di trust-checking, un area emergente di ricerca secondo DARPA [2].

Il contributo principale di questo lavoro di tesi è lo sviluppo di un tecnica di trust-checking chiamata *improved Fully Integrated Embedding* (iFIE) in cui un meccanismo di parità 2D ECC (basato sull'idee proposte in [3]) risulta placed-and-routed accanto al circuito originale. L'obiettivo sottostante è di distribuire un circuito FPGA monolitico che sia capace di auto-rilevare alterazioni o Trojan senza l'uso di riconfigurazione dinamica parziale, una caratteristica che se usata esclude la possibilità di bitstream encryption. Diversamente dalla tecnica originale introdotta in [3] dove la riconfigurazione dinamica parziale è necessaria, il nostro approccio funzionale on-chip iFIE combina trust-checking e bitstream encryption, una combinazione desiderabile in delicate applicazioni militari o commerciali. Possiamo poi ottenere un' efficiente implementazione per l'architettura iFIE di trust-checking grazie a:

- Una modifica dello schema strutturale di parità 2D ECC in modo tale da evitare l'uso di alcuni componenti originariamente richiesti in [3].
- Una nuova struttura iFIE basata sull'idea di coni capace di ridurre l' hardware overhead associato con l'architettura di trust-checking.
- Un algoritmo euristico per la generazione e la selezione di coni in modo tale da minimizzare l'hardware overhead.
- Un secondo algoritmo euristico per la generazione e la selezione di coni con considerazioni riguardanti le prestazioni.

Un altro contributo di questo lavoro è una nuovo protocollo richiesta-risposta di trust-checking che supera il protocollo base presentato in [3] in termini di probabilità garantita. Viene quindi proposto uno schema di parità *Reconfigurable Error Correcting Code* (RecECC) che cambia la sua composizione ad ogni richiesta, generando un numero astronomico di combinazioni e nascondendo la struttura iFIE da occhi malevoli. Inoltre, questo nuovo protocollo non richiede comunicazioni cifrate così l'area usata per implementare le funzioni di encryption/decryption può venire tranquillamente riutilizzata per l'hardware overhead associato all'architettura iFIE.

Questa tesi è composta da sei capitoli. Il **Capitolo 1** fornisce un introduzione al problema dei circuiti attendibili, presentando una classificazione dettagliata dei Trojans. Il problema viene poi contestualizzato alle FPGAs, introducendo l'importante idea di *trusted FPGA design*. Inoltre, il capitolo riporta una dettagliata analisi delle tecniche di sicurezza per FPGA sottolineando la loro parziale inefficienza contro alterazioni circuitali e Trojans.

Una dettagliata visione d'insieme sulla tecnica introdotta in [3] è proposta nel **Capitolo 2**. La tecnica presentata consiste in un'applicazione strutturale di un codice di parità all'array FPGA in modo tale da rilevare ogni cambio di funzionalità associato con alterazioni circuitali o iniezione di Trojans. Sono introdotti due livelli di randomizzazione in modo tale da evitare semplici masking e attacchi ai componenti di trust-checking. Uno studio analitico fornisce una valutazione teorica sulla robustezza contro i masking. La rimanente parte del capitolo è quindi dedicata alla presentazione delle modalità con cui è possibile integrare la tecnica nel flusso convenzionale di progettazione EDA. Infine, sono presentati tre differenti approcci per integrare i componenti di trust-checking nel circuito FPGA originale. Il mio lavoro di tesi si focalizzerà sull'approccio FIE proponendo una versione estesa e migliorata nota come iFIE.

Dunque, il **Capitolo 3** è completamente dedicato all'approccio iFIE. Prima di tutto, vengono introdotti alcuni stimolanti problemi di design relativi al mantenimento dell'hardware overhead e del performance overhead all'interno di limiti accettabili. Vengono analizzate le varie sorgenti di overhead che rendono l'implementazione FIE di base molto inefficiente. Poi viene presentata una prima soluzione adottata dall'approccio iFIE in modo da ridurre drasticamente l'hardware overhead relativo ai multiplexers. Successivamente, viene presentato un altro miglioramento architetturale che permette di evitare l'uso di funzioni di parità. Il capitolo continua introducendo la più importante innovazione relativa all'approccio iFIE. Come vedremo, il meccanismo di trust-checking è applicato a coni sottocircuitali in modo da evitare multiplexers sulle connessioni interne ai coni. Ultimo ma non meno importante, viene introdotto un nuovo schema di parità RecECC, confrontandolo poi in termini di vantaggi e robustezza con un convenzionale schema di parità 2D ECC.

Il **Capitolo 4** elabora ulteriormente le strutture a cono, discutendo alcuni approcci euristici per la generazione e la selezione di coni con overhead minimizzato. Prima di tutto, il problema è formalizzato in termini algoritmici come cone covering. Viene quindi introdotta una metrica di benefit basata sui concetti di covering e di cutting. Questa metrica viene poi utilizzata da un algoritmo di generazione in modo tale da costruire "buoni" coni secondo la prospettiva di minimizzazione dell'hardware overhead. Il capitolo continua proponendo un algoritmo di cone covering che sceglie in base alla citata metrica di benefit. L'ultima parte del capitolo introduce un approccio algoritmico alternativo che genera e sceglie i coni tenendo conto sia dell'hardware overhead che del performance overhead.

Il **Capitolo 5** fornisce una serie di risultati sperimentali che mostrano una riduzione drastica nel contributo di hardware overhead relativo ai multiplexers. Vengono inoltre proposte una simulazione di tipo behavioral ed una simulazione di tipo post-P&R le quali hanno la funzione di validare l'idea di trust-checking applicata ai coni. La prima parte del capitolo si preoccupa dei risultati relativi alle tecniche architetturali ed algoritmiche di minimizzazione dell'overhead. Siccome il nostro lavoro è, al meglio della nostra conoscenza, il primo di questo genere, confronteremo i nostri algoritmi con alcune variazioni interne in modo da stabilire se le metriche scelte sono vantaggiose o meno. La seconda parte del capitolo è quindi dedicata alle simulazioni le quali verificano la capacità del meccanismo ECC di rilevare ogni iniezione di Trojan o ogni alterazione malevola nella logica, nei componenti sequenziali o nelle interconnessioni.

Il **Capitolo 6** elenca infine gli obiettivi raggiunti in questo lavoro di tesi e propone alcuni spunti per lavori futuri riguardanti l'architettura iFIE di trust-checking.

Summary

Field Programmable Gate Arrays FPGAs combine the programmability of processors with performance closer and closer to custom ASIC. In the last decade, they have reached a sufficient level of cost and performance to represent an attractive solution for low-volume sensitive military or commercial application [1]. This opens a new key issue related with the “trustworthiness” of the deployed FPGA circuit since the conventional design process is vulnerable to malicious insertions commonly referred as Hardware Trojan that could, under specific conditions, result in functional changes and/or catastrophic failure of the sensitive application.

In order to reduce the threat of tampering or Trojan injection, it is necessary to introduce the idea of *trusted FPGA design* for which circuits must perform only the functionality for which they were originally designed, no more and no less. This idea needs to be enforced during all the circuit life, from its design to its deployment on the application field. Intuitively, FPGAs are designed to change their functionality so this malleability introduces uniques vulnerabilities. The availability of on-chip security techniques such as cryptography can protect the underlying design against unauthorized copy but are not sufficient to immunize from malicious insertions. Moreover, other alternative security approaches available in literature are relatively inefficient since none of them covers all the possible vulnerabilities of the FPGA design/deployment.

From these premises it follows that we need a more comprehensive trojan detection technique that covers all the circuit life. This need has provided the main motivation of this thesis work. An explicit trust-checking for FPGA circuits can guarantee with very high probability that the functionality is the one for which the

circuit was originally designed, no more and no less. Moreover, the used technique must be functionality-based and must operate on-chip in order to detect any intentional tampering or Trojan insertion. This thesis is thus concerned with the design of trusted FPGA circuits adopting trust-checking mechanisms, an emerging area of research according to DARPA [2].

The main contribution of this thesis work is the developing of a trust-checking technique called *improved Fully Integrated Embedding* (iFIE) where a 2D ECC parity mechanism (based on the ideas proposed in [3]) is placed and routed along with the original design. The underlying goal is to deploy a monolithic FPGA circuit which is capable of self-detecting tampers or Trojans without using partial dynamic reconfiguration, a feature which excludes bitstream encryption. Differently from the original technique introduced in [3] where partial dynamic reconfiguration is required, our on-chip functional-based iFIE approach combines trust-checking with bitstream encryption, an highly desirable combination for sensitive military or commercial application. We can obtain an efficient iFIE trust-checking logic implementation by the means of:

- A modification of the structural 2D ECC parity scheme in order to avoid the use of some trust-checking components originally required in [3].
- An iFIE structure based on the idea of cones which reduces the hardware overhead related with ECC-based functional trust-checking.
- An heuristic algorithm for generating and selecting cones in order to minimize the hardware overhead related with trust-checking architecture.
- A second heuristic algorithm for generating and selecting cones according to delay performance considerations.

Another contribution of this work is a novel challenge-response trust-checking protocol which overperforms the basic protocol presented in [3] in terms of guaranteed probability. We propose a *Reconfigurable Error Correcting Code* (ReECC) parity scheme that changes its composition at any challenge, generating an astronomically large number of combinations and hiding the iFIE structure from malicious eyes. Moreover, this novel protocol does not require ciphered communications so the released FPGA area used for encryption/decryption can be used for the required hardware overhead.

This thesis is composed of six chapters. **Chapter 1** provides an introduction to the problem of circuit trustworthiness, presenting a detailed classification of Hardware Trojans. The problem is then contextualized to FPGAs, introducing the underlying idea of *trusted FPGA design*. Moreover, the chapter reports a detailed analysis of the available FPGA security techniques highlighting their partial ineffectiveness against tampering and Trojans.

A detailed technical overview of the ECC-based technique introduced in [3] is proposed in **Chapter 2**. The presented technique consists of a structural application of a parity code to the base FPGA array in order to detect any functionality change associated with tampering or Trojan injections. Two level of randomization are introduced in order to avoid trivial masking and attacks to the trust-checking components. An analytical study provides a theoretical evaluation of robustness against masking. The remaining part of the chapter is then dedicated to present the integration of the ECC-based technique with the conventional EDA flow. At last, three different approaches for embedding the trust-checking components are presented. Our thesis work will focus on the FIE approach devising its extended and improved version known as iFIE.

Thus , **Chapter 3** is completely dedicated to the iFIE approach. First of all, we introduce some challenging design issues related with keeping the hardware and the performance overheads within acceptable limits. We analyze the sources of overheads which make the basic FIE implementation very inefficient. Then, we present a first solution adopted by the iFIE approach in order to drastically decrease the hardware overhead related with multiplexers. Subsequently, we present another architectural improvement that permits to avoid the use of parity functions. The chapter continues introducing the most important innovation of the iFIE approach. As we will see, the trust-checking mechanism is applied to more coarse-grained cone subcircuits in order to avoid multiplexers on the internal connections. Last but not least, a novel RecECC parity scheme is introduced, making a comparison of its advantages and its robustness against the conventional 2D ECC parity scheme.

Chapter 4 further elaborate on cones structures, discussing some heuristic approaches for generating and selecting cones with minimized overheads. First of all, the algorithmic problem is formalized as a cone covering. Then, a benefit metric based on the concepts of covering and cutting is introduced. This metric is then used by a cone generation algorithm in order to construct "good" cones with the perspective of minimizing the hardware overhead. The chapter continues proposing a cone covering algorithm for overhead minimization which selects cones using the aforementioned metric. The last part of the chapter introduces an alternative algorithmic approach that generates and selects cones taking account of both hardware and performance overheads minimization.

Chapter 5 provides a set of experimental results in order to show a drastic reduction in the multiplexer overhead contribution. Moreover, it proposes a behavioral and a post-P&R simulations which validate the presented idea of trust-checking based on cone structures. The first part of the chapter is concerned with the results of proposed architectural and algorithmic overhead minimization techniques. Since our work is, to the best of our knowledge, the first of its kind, we compare our algorithms to some internal variations that establish that our chosen metric is advantageous. The second part of the chapter is then dedicated to simulations which test the capability of the ECC-based mechanism of detecting any Trojan injection or any malicious modification such as logic, sequential or interconnection tampering.

Finally, **Chapter 6** summarizes the goals achieved in this thesis work, also proposing some hints for future works concerning the iFIE trust-checking architecture.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Field Programmable Gate Arrays (FPGAs) combine the programmability of processors with performance closer and closer to custom *Application Specific Integrated Circuits* (ASICs). In the last decade, FPGA technology has gained popularity due to its cheapness in low to medium-volume production compared to hardwired solutions and due to the added advantage of reconfigurability that allows multiple application circuits to be mapped into the FPGA chip at different times, as needed by the overall system. Moreover, modern FPGAs have reached a sufficient level of performance to represent an attractive solution for low-volume sensitive applications such as avionics, communications, military, industrial and so on. This trend introduces a new key issue in the FPGA design flow related with the "trustworthiness" of the deployed hardware, according to the intuitive notion recently given by authors in [4].

This thesis work is mainly concerned with the design of trusted FPGA circuits. This problem is part of a more general trust issue in semiconductor *Integrated Circuits* (ICs), a novel research area recently supported by DARPA [2]. The outsourcing of the manufacturing process to external foundries spread around the world have made ICs vulnerable to malicious alterations that could, under specific conditions, result in functional changes and/or catastrophic failure of the sensitive application in which they are embedded. In section 1.1 we introduce a general model for these malicious circuits commonly referred as Hardware Trojans. Despite literature [5,6] tailors the presented Trojan model with IC layouts in mind, we can easily transpose it to FPGA circuits, the matter our concern.

In order to avoid any tampering or Trojan injection, it is necessary to introduce the idea of *trusted design* for which circuits must perform only the functionality for which they were originally designed, no more and no less. FPGA design flow depicts a different scenario compared to ASIC where tampering and Trojans are directly injected in the silicon layout. FPGAs provide an important separation between device manufacturing and application design for which the problem of *trusted FPGA design* is restricted to a fixed array of configurable elements already fabricated in silicon. In section 1.2 we discuss this separation by presenting the entire manufacturing flow of a FPGA-based application.

The idea of *trusted FPGA design* needs to be enforced during all the circuit life, from its design to its deployment on the application field. Intuitively, FPGAs are designed to change their functionality so this malleability introduces uniques vulnerabilities. In section 1.3 we show how most of the available FPGA security techniques are not sufficient to immunize from malicious insertions. Thus, we need a trojan detection technique that works on-chip and that is functionality-based. This need has provided the main motivation of this thesis work.

At last, section 1.4 points out our innovative contributions in the design of trusted FPGA circuits and trust-checking mechanisms capable of detecting any intentional or unintentional tampering.

1.1 Hardware Trojans

A Trojan circuit is a malicious circuit alteration capable of performing unexpected actions that can compromise the system application. The outsourcing of the IC fabrication to “untrusted” fabrication facilities has made ICs vulnerable to this new threat. An adversary can introduce a Trojan designed to disable/destroy an important functionality, to degrade signal integrity or to covertly leak confidential information stored in the circuit memory. Moreover, every software security mechanism used in a computing system relies on the underlying hardware and can be easily bypassed by malicious tampers.

A description of a Trojan circuit is presented in [5]. The basic model is composed by two major components that are the triggering logic monitor and the payload activation logic. Intuitively, the triggering circuit describes when a Trojan activates itself whereas the payload circuit describes how a Trojan carries its destructive action. Figure 1.1 shows the aforementioned structure. A set of q-external inputs (q-trigger) activates the payload at the proper event which occurs only under very rare conditions in order to minimize detection. The payload action is carried through a XOR-gate which basically inverts the logic value of a legitimate wire connection when the Trojan is triggered.

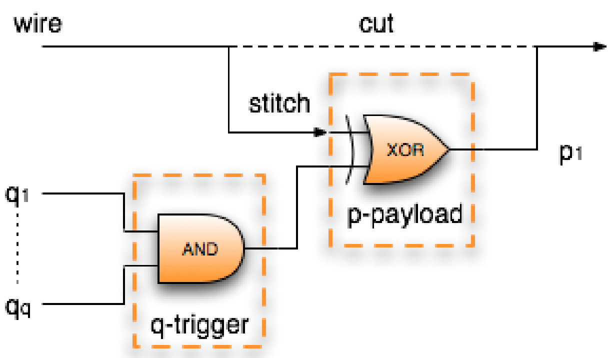


Figure 1.1: A Trojan model

In order to evade during chip testing, a Trojan uses nodes with low controllability as q-trigger conditions and affects nodes with low observability as p-payload effects. Moreover, a Trojan can be designed as a time bomb which activates after a certain number of triggering. This behavior makes the Trojan detection harder and can be obtained by adding a k-bit counter in the triggering logic monitor.

In general, Trojan circuits can have complex and clever behaviors. In [5] it has been proposed a taxonomy based on the nature of their triggering and payload mechanisms, as shown in Figure 1.2. The presented classification considers different activation modalities which are purely combinatorial (rare values) or even sequential (k-bit delay and rare sequences). The activation event can be synchronous or asynchronous depending by its timing. A malicious circuit can

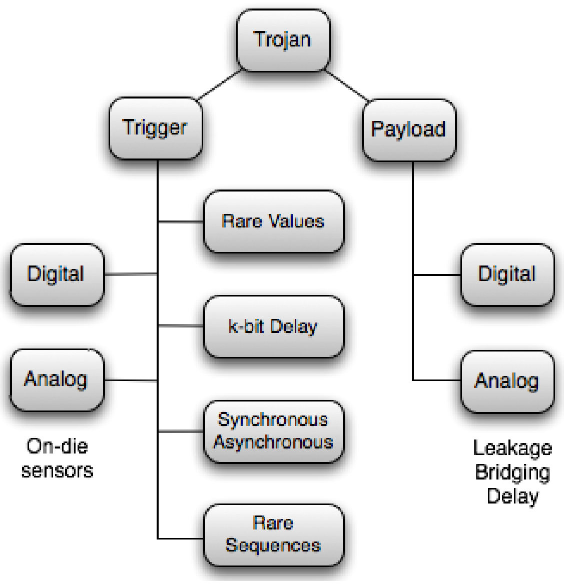


Figure 1.2: Trojan taxonomy

be digital or analog based. An adversary can integrate an on-die sensor in the silicon layout. This sensor works as a trigger mechanism activating the Trojan on an external stimulus. For example, embedding a radio receiver we can virtually activate a Trojan using a radio signal. The payload mechanism can also be analog based. A Trojan can increase the toggling activity as well as insert a current leakage in order to drain the battery energy. Another destructive behavior consists of increasing the signal delay or inserting bridging faults.

In this thesis work we only focus on digital triggering and payload. This represents a reasonable restriction since an adversary cannot freely integrate IC sensors on the FPGA area. Moreover, analog Trojan detection is a difficult task since we need to monitor different physical properties such as power, thermal emission or electromagnetic profiles. Thus, the aforementioned simplification restricts the problem on a functional perspective which fits with the purposes of a reconfigurable hardware architecture. However, the detection of tamper and Trojan inclusions in IC still remains a difficult problem. A brute force approach involves physical inspection and destructive reverse engineering of a chip. Unfortunately, nanometer IC size and complexity make this process difficult, slow and especially costly to be extensively applied on large scale. An adversary designs a Trojan circuit to activate under very specific conditions, which makes it difficult to detect using classical testing techniques. One might consider exhaustive testing by applying 2^{n+m} test vectors on the original circuit with n-inputs and m-flipflops as assumed in [5]. It is easy to argue that this approach has exponential complexity and so it is applicable only to small circuits. Even in this scenario, a Trojan designed as a time bomb makes the detection problem undecidable since we need 2^{n+m+k} test vectors but k is unknown (basically, we cannot certify a circuit as Trojan-free since there is no k at which the detection algorithm stops itself). This simple reasoning gives an intuitive hint on the underlying complexity of the detection problem.

In [6] it has been provided a more detailed explanation about the inefficiency of functional and (*Automatic Test Pattern Generator*) ATPG-based fault-detection techniques. The main concern with testing is that we focus on verifying a pool of known functionalities but we don't deal with the unwanted ones. A malicious opponent can easily craft a Trojan in order to evade the set of test vectors tailored for verifying existing functionalities. Both [5, 6] also discuss about another approach known as side-channel analysis or *Differential Power Analysis* (DPA) [7]. Briefly, this technique analyzes a measured physical parameter, the so called side-channels such as power, temperature and electromagnetic profile. A tampered IC can be potentially detected by variation of these parameters during circuit activity. Unfortunately, in modern nano-scale ICs the amount of parameter variation introduced by the fabrication process can be $\pm 7.5\%$ so side-channel analysis has important limitation, especially when Trojan are small circuits.

A possible solution to the Trojan detection problem is finally proposed in [5]. The approach reasonably assumes that a Trojan is maliciously placed in order to be driven by nodes with low controllability and in order to affect nodes with low observability. For this reason, the presented technique constructs test vectors that are able to stimulate these nodes. Simulation results show that such a technique can be effective to detect most (but not all) small combinatorial Trojans. Another possible solution based on graph coloring is proposed in [8]. The presented technique deals with an abstract graph representation of the circuit and embeds clique-like traps which make difficult to insert additional nodes (basically a Trojan) without increasing the number of colors needed for solving the graph coloring problem. Given a circuit graph, an increased number of colors respect to a previously determined parameter corresponds to a malicious insertion. However, the presented technique represents a partial protection for the design flow. In other words, we cannot have a *trusted IC design* since a malicious fabrication foundry can take the protected design, insert a Trojan and produce malicious chips without control of the original design owner. Other security mechanisms are proposed in [9, 10]. The proposed techniques are more concerned with IC design piracy (unauthorized copies) than Trojan detection. *Digital Right Management* (DRM) of *Intellectual Property* (IP) cores is done using asymmetric cryptography and combinatorial locks. More in detail, a foundry can activate a manufactured IP core only with a key provided by the design owner. Every chip needs a different key since it embeds an different chip identifier computed using a *Physical Unclonable Function* (PUF) [11].

The previous literature analysis shows how the *trusted IC design* issue represents an emerging area or research, with a concrete and underlying application that motivates this thesis work. From now on, we focus on the strictly related issue of *trusted FPGA design*. In both the cases, we need to assure that circuits perform only the functionality for which they were originally designed, no more and no less. However, there is a subtle difference between ASICs and FPGAs. In the former case, Trojans are directly inserted in the silicon layout of the chip. In the latter case, we have a fixed array reconfigurable elements already fabricated in silicon and Trojans only affect the FPGA configuration which determines the implemented circuit functionality. Since many low-volume critical applications are mapped onto an FPGA, the *trusted FPGA design* research is well-motivated. The presented problem can be solved by using a functional-based technique capable of detecting any type of malicious injection in the FPGA circuit configuration, from a tamper in the reconfigurable elements mapping the FPGA circuit to a Trojan injected in the unused FPGA area.

1.2 Trust issue in FPGA manufacturing

Field Programmable Gate Array are semiconductor devices structured as a matrix of *Configurable Logic Blocks* (CLBs) connected through a programmable network. On its perimeter, an FPGA provides a set of pins known as *Input Output Blocks* (IOBs) which are typically used to access I/O resources. The general structure of a typical FPGA is shown in Figure 1.4. A CLB is the elementary computational unit which is used to build combinatorial or sequential logic functions, whose internal structure varies according to the specific technology. For instance, Xilinx Virtex-4 family [12] implements a CLB using four similar slices which can efficiently cooperate within the same logic block. Each slice can generate boolean functions by means of two *Look-Up Tables* (LUTs), two registers, a carry chain and a multiplexer network. Communication among CLBs is guaranteed by the programmable interconnection network, which also provides routing routes to reach the I/O pins. In addition, an FPGA chip can integrate one or more microprocessors useful for implementing embedded systems.

Figure 1.3: FPGAs from various vendors

Both the configuration of a single CLB and the routing scheme can be computed by automatic tools, which translate a high-level specification into an appropriate configuration code called bitstream, which specifies how each programmable element of the device should be configured. The FPGA can be physically programmed by sending a bitstream thought a reconfiguration interface, which is able to access the device configuration in both writing and reading mode. A bitstream is employed to program the entire device area, and hence the entire logic on the FPGA is interrupted and reconfigured during the process, even if part of it is left unchanged.

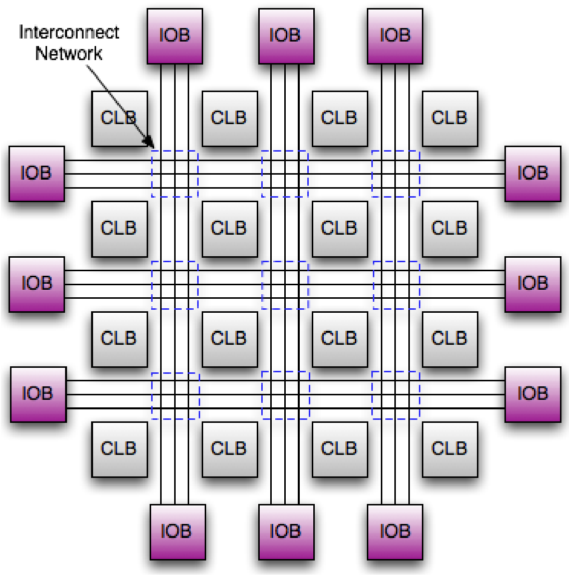


Figure 1.4: FPGA structure

FPGA technology is highly attractive for implementing system applications since represents a good trade-off between software flexibility and ASIC performance. In the former case, we may take advantage of usability, upgradability, portability and low development cost, having as down side moderate speed and high power consumption. In the latter case, we may take advantage of low unit price, high performance and lower power dissipation, having as down side higher development cost and lack of flexibility with respect to application updates. Summarizing, FPGA technology captures the best of both worlds since offers a sufficient level of performance with the flexibility allowed by reconfiguration. As consequence, the popularity of such technology has increased in the last few year, especially for highly specific low-volume applications such as sensitive military systems. According to an estimate in [13], 110000 different FPGA design project are expected to begin in 2010.

FPGAs offer a different perspective in terms of "trustworthiness" if compared with ASICs. An intuitive weakness of FPGAs is related with reconfigurability which virtually permits to change their functionality at any time. On the other side, ASICs are hard-wired so no Trojan can be inserted after their fabrication. Despite this negative aspect, we can identify an important separation between design process and manufacturing flow. In [1] it has beed explained how this separation dramatically simplifies the process of assuring the aforementioned *trusted FPGA design*. Considering the traditional IC manufacturing flow, external foundries are a serious concern since expose the design to attacks. For instance, an adversary may exist in the mask-making company, the wafer fabrication, the packaging company or in any of the shipping facilities in between them. In FPGAs, the sensitive IP is not loaded onto the device until after it has been manufactured and delivered, making it harder for adversaries to target a specific application or user.

The FPGA manufacturing flow is shown in Figure 1.5. We have security concern only during manufacture of the base FPGA array and during the deployment of the fielded system. The physical FPGA device is manufactured using the traditional IC flow so it is exposed to the mentioned threats (non-secure manufacturing). An adversary can still insert a Trojan in the CLBs array in order to affect the final application that will be loaded as a bitstream. However, this kind of attack is very ineffective since it is based on a small probability of success. In fact, an adversary cannot directly attack the actual design so it tries to insert a Trojan in a random position of the base FPGA array. For this reason there is only a small possibility that a sensitive part of the application is placed on the tampered CLB. Moreover, it is not possible to decide a-priori the application that will be implemented on a certain manufactured FPGA device. In other word, there is a even smaller probability that a Trojan will exactly meet the target for which it was originally meant. In the most probable case, the tampered FPGA device will be used in a non-critical application and then returned to the IC manufactured since it exposes the Trojan as a malfunction. Any returned device is carefully inspected for defects so deliberate tampering can be detected. The attacker can intuitively increase the number of tampers in order to increase its low probability of success.

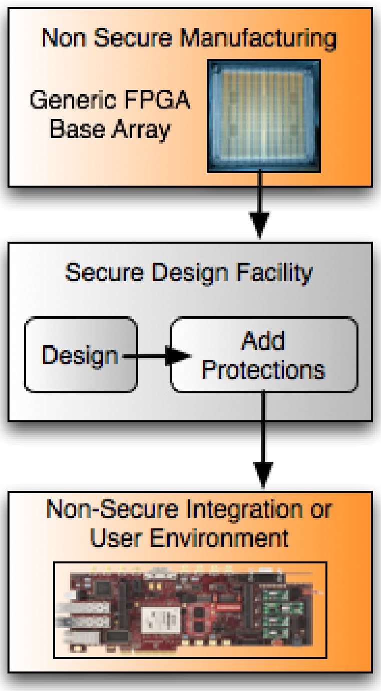


Figure 1.5: FPGA manufacturing flow

However this strategy has a drawback since a device with more Trojan has also more probability of being caught as defective during testing. On the other side, a design can further decrease the probability of success implementing critical parts of the design with *Triple Modular Redundancy* (TMR) [14]. Since each part is placed on a different CLB, there is a lower probability that two of the three modules are affected by a Trojan. At last, an adversary can craft a more-fruitful attack to the FPGA's security features used to protect the bitstream [15, 16]. As we will see later in this chapter, these security features are used to protect the system in the field. In this case, the only defense is a complete verification of these functions during testing. This may sound difficult, but the verification task must cover only a very small fraction of the overall design. Summarizing, the separation between the generic FPGA device manufacturing and the application design assures with a certain degree of confidence that no effective attack is possible during the base FPGA array manufacturing.

The second step of the FPGA manufacturing flow is concerned with the design process. The depicted scenario is beneficial since the FPGA bitstream can be developed and loaded in a secure design facility after the base array is manufactured and tested. Supposing an adequate level of protection for the secure site, there is no way for a malicious adversary to insert Trojans or just steal the design, at least until the FPGA-based application is released into a possible hostile environment (non-secure environment). Modern FPGAs include bitstream security features suitable for protecting the fielded design. Since an *Static Random Access Memory* (SRAM) FPGA's bitstream is an electronic message, methods of information security such as encryption can be applied to assure the integrity and confidentiality of the bitstream. This represents an effective mechanism against unauthorized copy of a design, theft of the design and reverse engineering. FPGA devices may integrate a cryptographic protection based on hardwired algorithms like *Data Encryption Standard* (DES), triple DES or *Advanced Encryption Standard* AES (usually, there are implemented by a dedicate on-chip decryptor and a dedicated key storage memory). Moreover, after loading an encrypted bitstream readback is disabled and every attempt of reading/writing the key will clear all keys and configuration data. Roughly speaking, bitstream encryption is an effective technique for IP design integrity and confidentiality. We can assume that the average adversary cannot overcome this protection. In fact, the only viable attack requires to steal the encryption key stored in the FPGA. More in detail, it is necessary to keep power to the key memory and to destructively probe the silicon layout by milling away many levels of metal. It is reasonable to assume that this attack is beyond the capabilities of most adversaries. Bitstream encryption mode introduces some restrictions related with reconfiguration. In fact, only single full-chip configuration is permitted since partial and dynamic reconfiguration are disallowed. This may represent a serious limitation for such application that explicitly require to dynamically load IP cores during their running.

The idea of *trusted FPGA design* requires that the deployed FPGA circuits perform only the functionalities for which they were originally designed, no more and no less. We have seen how base FPGA array fabrication and design in a secure facility can prevent Trojan insertions. Despite bitstream encryption is effective against unauthorized copy of a design, it is not sufficient to prevent Tamper insertions in the field using remote attacks capable of modifying bitstream bits. Moreover, a secure design facility represents a strong assumption especially for current system integration approaches based on *Commercial Off-The-Shelf* (COTS) devices to save and money. For these reasons, the separation approach proposed in [1] is not sufficient to completely obtain a *trusted FPGA design*. Thus, we require a deeper analysis of the design/deployment flow in order to point out all the vulnerabilities where tampers and Trojans can be injected.

1.3 Trusted FPGA design

The problem of assuring a *trusted FPGA design* cannot be transformed into a generic security problem in embedded systems [17] solvable with cryptography. As we have seen, we need a technique capable of detecting tampers and Trojan insertions during the entire FPGA-based application life cycle, from its design to its deployment in the field. Before analyzing some ideas and proposing our technique, we need to clarify the scenarios in which a malicious tamper can happen. We focus our attention only on the FPGA design/deployment flow since we assume with a certain degree of confidence that the base FPGA array is trusted. In any case, the technique proposed in this thesis is capable of detecting any functional modification, even caused by a base CLB tampering.

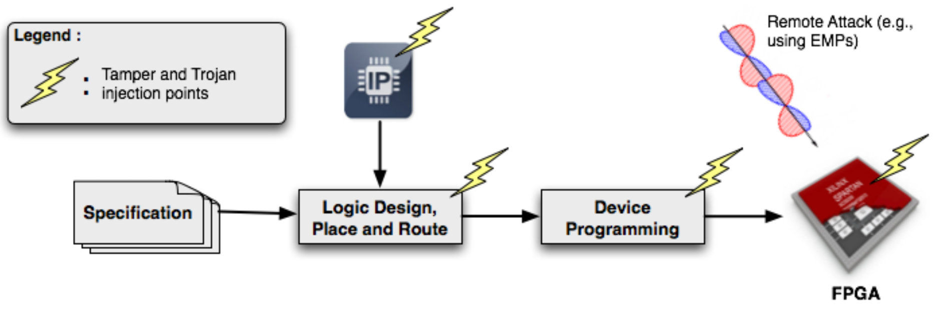

Legend :
Tamper and Trojan injection points
IP
Remote Attack (e.g., using EMPs)
Specification
Logic Design, Place and Route
Device Programming
FPGA

Figure 1.6: Trojan injection points

Figure 1.6 shows four vulnerabilities that affect the FPGA design/deployment flow. During the entire *Electronic Design Automation* (EDA) flow, a high-level specification of the circuit which is assumed Trojan-free goes through logic synthesis, technology mapping, placing and routing phases. An output bitstream will represent the original circuit ready to be loaded on the FPGA device. In any of the presented step, a malicious adversary can inject Trojans by means of tampered EDA tools. Since most of the time these tools are developed outside a company or an organization, the hypothesis of a secure design facility assumed in [1] becomes weak. Another vulnerability is concerned with integration where different IP cores, often obtained from third parties, are cobbled together in order to compose the overall FPGA system. These cores can be subverted by tampering the tools or by tampering the cores themselves. The integration can also happen at Printed Circuit Board (PCB) level where COTS chips provide latest suitable technologies with lowest cost. Again, the hypothesis of a secure design facility seems weak in this scenario where different companies and organization need to collaborate. Device programming represents the loading of a bitstream onto the FPGA device. This phase is vulnerable to Trojan insertions since an adversary can tamper the bitstream tools or the loading device in order to modify the designed circuit functionality. At last, the fielded FPGA device can be affected by remote attacks which modify the bitstream (and so the circuit functionality), even in case of bitstream encryption and disabled reconfiguration. For SRAM FPGA devices, single configuration bits are stored as electronic charges. Using high-energy *ElectroMagnetic Pulses* (EMPs) is possible to flip the bit status without using traditional reconfiguration interfaces in order to inject tampers or Trojans.

The trustworthiness of all the EDA tools involved in the complex FPGA design flow is not a trivial issue. Industry already deals with this problem in operating system. security kernels, applications and compilers. In [18] it has been proposed an holistic approach in order to build a custom set of trusted tools for security-critical hardware as a subset of the commercial tool chain's optimization functions. This approach can prevent tampers with the tools used to translate the design to the FPGA bitstream but not with the design itself since a Trojan can be added in the high level circuit specification or even in the bitstream. *Trusted FPGA* *design* can be guaranteed only by immunizing the entire design and deployment flow from the aforementioned vulnerabilities.

In [1] it has been proposed a Layout-Versus-Schematic (LVS) comparison between two netlists, one extracted from the initial untampered design and another extracted from the deployed bitstream. This technique can potentially highlight any difference introduced by a tamper or a Trojan and it is effective only for vulnerabilities inserted before device programming. We can extend its range to the remaining vulnerabilities by reading back the deployed bitstream from the FPGA device and by applying the LVS comparison. However, this technique does not seem applicable for some reasons. First of all, readback mode may be disabled for protecting IP confidentiality. In another case, the programming device may be tampered in order to remove the Trojan during readback, cheating the LVS comparison. At last, this technique is an off-chip approach so it is not effective for a fielded FPGA design.

Fault-tolerant techniques of configuration memory cannot solve the *trusted FPGA design* problem since they do not assume malicious opponent. In fact, the knowledge of the underlying mechanism can be exploited to craft the circuit functionality with-out raising alarms. Consider, for example, the TMR technique used in [19]. A circuit is replicated and its output is compared with backup copies outputs. In this scenario, a malicious attacker can apply a smart attack in which the same tamper is inserted in every copy or in which the output checker is crafted in order to not raise alarms. In [19] it has also been proposed configuration scrubbing, a technique in which the configuration is continuously reloaded in order to repair disrupted bits. Scrubbing can virtually delete any tamper or Trojan injected by remote attacks. However, the technique is ineffective against pre-configuration tampering. In [20] it has been presented another approach concerned with fault-tolerance of configuration memory that can deal with a limited number of errors. The proposed technique includes a duplicate-and-compare implementation of the design and uses partial reconfiguration in order to reload the detected erroneous memory frames. Again, the technique is ineffective against pre-configuration tampering as any other fault-tolerant approach.

Another possible trust-design approach is signature computation and checking of the bitstream. In this approach, a signature $\mathcal{S}$ of the bitstream is computed on the Trojan-free FPGA bitstream. During system integration or just before device programming, an off-chip integrity computation checks if current signature $\hat{\mathcal{S}}$ is equal to the expected one, detecting possible tampers or Trojans in the bitstream. As mentioned, this is an off-chip checking approach, and thus it cannot protect against tampers introduced by the device programming unit or by remote attacks on the field. Authors in [21] introduce a reconfigurable trustworthy computing platform based on asymmetric encryption. The proposed architecture offers a set of trusted reconfigurable modules used for security functions such as symmetric encryption or hashing. A *Bitstream Trust Engine* (BTE) is capable of decrypting and verifying the authenticity and integrity of loaded bitstreams. However, the proposed technique explicitly excludes non-invasive tampering so it is still virtually vulnerable to remote attacks on the field. Regarding device programming, a *Secure Update Mechanism* (SUM) hardware architecture is presented in [22]. This mechanism is available for remote updating of FPGA-based systems. SUM encrypts the bitstream during the upload phase and verifies its signature in order to prevent man-in-the-middle attacks where an adversary replaces a configuration update by one of his choice or performs a downgrade in order to exploit previous system flaws. Again, this mechanism does not address remote attacks meant as modification of the bitstream on the field.

In conclusion, all the above alternative approaches to the *trusted FPGA design* are relatively inefficient since none of them covers all the aforementioned vulnerabilities of the FPGA design/deployment flow where tampers and Trojans can be inserted. Thus, it is necessary to employ an explicit trust-checking technique for FPGA circuit that guarantees with very high probability that the functionality is the one for which the circuit was originally designed, no more and no less. This trust-checking technique must be functionality-based and must operate on-chip in order to cover all the possible vulnerabilities. In fact, we can virtually catch any tamper or Trojan insertion by comparing the functionality currently implemented on-chip with the functionality of the original untampered design. A background idea of trust-checking is introduced in [3]. We will provide a detailed explanation of the proposed ECC-based trust-checking technique in Chapter 2. In this thesis work we propose a significative extension of the ideas and methodology proposed in [3] in order to provide an efficient *Placed-and-Routed* (P&R) implementation and a novel challenge-response trust-checking protocol with an astronomically smaller tamper/Trojan insertion probability. Moreover, to the best of our knowledge there is no other available literature regarding the *trusted FPGA design* problem.

1.4 Innovative contributions

This thesis work is concerned with the design of trusted FPGA circuits and trust-checking mechanisms capable of detecting any intentional tampering or Trojan insertion at any point of the FPGA design/deployment flow, an emerging area of research according to DARPA [2]. The main contribution of this work is the developing of a trust-checking technique called *improved Fully Integrated Embedding* (iFIE) where a 2D ECC parity mechanism (based on the ideas proposed by [3]) is placed and routed along with the original design. The underlying goal is to deploy a monolithic FPGA circuit which is capable of self-detecting tampers or Trojans without using partial dynamic reconfiguration, a feature which excludes bitstream encryption. Differently from the original technique introduced in [3] where partial dynamic reconfiguration is required, our on-chip functional-based iFIE approach combines trust-checking with bitstream encryption, an highly desirable combination for sensitive military or commercial application. We can obtain an efficient iFIE trust-checking logic implementation by the means of:

- A modification of the structural 2D ECC parity scheme in order to avoid the use of some trust-checking components originally required in [3].
- An iFIE structure based on the idea of cones which reduces the hardware overhead related with ECC-based functional trust-checking.

- An heuristic algorithm for generating and selecting cones in order to minimize the hardware overhead related with trust-checking architecture.
- A second heuristic algorithm for generating and selecting cones according to delay performance considerations.

Another contribution of this work is a novel challenge-response trust-checking protocol which overperforms the basic protocol presented in [3] in terms of guaranteed probability. We propose a *Reconfigurable Error Correcting Code* (RecECC) parity scheme that changes its composition at any challenge, generating an astronomically large number of combinations and hiding the iFIE structure from malicious eyes. Moreover, this novel protocol does not require ciphered communications so the released FPGA area used for encryption/decryption can be used for the required hardware overhead.

Chapter 2

State of the Art

In this chapter we give a technical overview of the novel ECC-based technique introduced in [3] and capable of detecting tampering of a FPGA circuit with very high probability. The rationale behind presented technique is a structural application of a parity code to the base FPGA array in order to detect any functionality change associated with tampering or Trojan injections. A genuine FPGA circuit will be distinguished by the correct parity output during the so-called trust-checking phase. As mentioned, to the best of out knowledge [3] represents the only work explicitly concerned with the *trusted FPGA design* problem so we consider the introduced ideas as the state-of-the-art in this particular field.

The chapter is organized in the following way. Section 2.1 introduces the basic notion of *Error Correcting Codes* (ECCs) presenting a simple *two-Dimensional* (2D) parity scheme which represents the theoretical foundation for the trust-checking mechanism in [3]. Section 2.2 presents the trust-checking technique itself which is basically a structural 2D ECC parity scheme implemented by means of available reconfigurable logic. A first randomization level is introduced in section 2.3 in order to avoid a trivial masking implementable by knowing the underlying 2D scheme. An analytical study of robustness against masking is then provided in section 2.4. A second randomization level is introduced in section 2.5 in order to protect the trust-checking units from malicious tampering. Section 2.6 introduces a new FPGA design flow in which conventional EDA phases are mixed with *trusted FPGA design* phases in order to enforce the presented randomized 2D ECC parity scheme. Finally, section 2.7 presents three different approaches for embedding the trust-checking components into the FPGA circuit since an on-chip implementation is a necessary requirement for addressing all the design vulnerabilities.

2.1 Overview on Error Correcting Codes

Coding techniques are greatly used in binary data transmissions to detect and eventually correct errors caused by noise or other impairments. Error correcting codes are based on using a redundancy check composed by extra data added to the initial information in order to achieve the error detection goal. One of the most classical ECCs is based on a 2D parity scheme [25]. This technique arranges the binary information as a bit matrix. For instance, Figure 2.1 shows two bytes (16 bits) arranged as a 4×4 bit matrix.

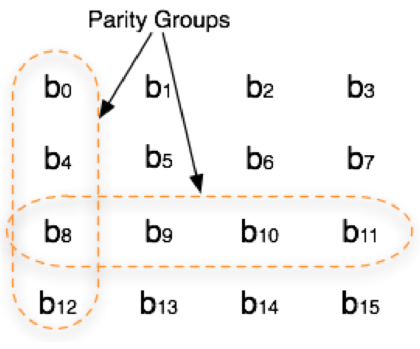


Figure 2.1: 2D ECC parity scheme

We compose the so-called *Parity Groups* (PGs) by aggregating bits in a row-wise and column-wise fashion. Each PG has a cardinality which depends by the considered matrix size. Suppose that a PG group is composed by k bits, $x_0, \ldots, x_{k-1}$. We associate to it a so-called even parity bit c calculated as

$$c = x_0 \oplus x_1 \ldots \oplus x_{k-1} = XOR_{i=0}^{k-1} x_i$$

Alternatively, we can use the opposite XNOR operator (odd parity) with identical properties in terms of error detection.

A parity bit c is capable of detecting an odd number of errors in a transmitted PG. More in detail, the receiver can compute the parity c^r of the received PG $x_0^r, \ldots, x_{k-1}^r$ and then check if it corresponds to expected parity bit c which is part of the transmitted message. This binary comparison is implemented by means of a XOR operation.

$$c \oplus c^r = \begin{cases} 0 & \text{no error } \textit{or} \text{ even errors} \\ 1 & \text{odd errors} \end{cases}$$

According to the logic table, a XOR between two identical logic values will always generate a zero value (or an even parity, using another terminology). For this reason, we expect $c \oplus c^r$ to be even in case of an error-free PG. However, this simple ECC cannot detect a flipping between two bits (or in general an even number of errors) inside a single PG since $x_i \oplus x_j = \bar{x_i} \oplus \bar{x_j}$ where x_i, x_j are two arbitrary bits inside the PG. This phenomenon is called error masking. Moreover, a parity bit c can detect a single error inside a PG but doesn't have enough information in order to identify (and correct) the exact position where the error occurred.

Using a 2D scheme, it is possible to improve the code reliability. Referring again to Figure 2.1, each row PG overlaps with all the column PGs (one for each bit) and vice versa each column PG overlaps with all the row PGs. This scenario gives an added property to the ECC parity. When a single bit is flipped, the correspondent row PG and the correspondent column PG detect an error. These simultaneous detections permit to identify the error position and apply a correction, avoiding an expensive data retransmission. Moreover, the masking phenomenon becomes harder since one or more errors remain undetected if and only if they cause an even number of flipping in all the PGs. The simplest combination which respects the above condition is a set of four errors intentionally placed in a 2×2 submatrix. In general, this combination is very unlikely in data transmission so the 2D ECC parity is generally consider reliable.

2.2 2D ECC parity scheme on FPGA

An FPGA circuit is implemented by means of a reconfigurable matrix composed by several CLBs and an interconnection network. In other words, we have a fixed IC layout (FPGA device) whereas the current circuit functionality depends by the loaded application bitstream. In order to detect tampering or Trojan insertions, we need to apply a functionality-based checking which verifies the configuration of each CLB. Moreover, trust-checking must be apply on-chip in order to address all the vulnerabilities highlighted in section 1.3. The technique presented in [3] doesn't explicitly deal with routing checking. Despite this weakness, it seems hard to introduce a malicious Trojan without modifying functionality of some CLBs. More likely, a routing modification can simply be detected by using classical testing. As partial solution, a sketch of routing checking is given by the future work section of [3].

Before continuing with our overview, we need to clarify the notions of tampering and Trojan insertion in a FPGA circuit. A tampering consists of a modification of a CLB configuration such as the function implemented by a LUT. A Trojan insertion is concerned with using an empty CLB to implement an hidden malicious functionality. However, there is a subtle distinction between the two notions since a Trojan can even be placed by tampering a configured CLB.

The core idea of functional trust-checking is concerned with applying the presented 2D ECC parity scheme to the CLB logic outputs. This arrangement is quite straightforward since FPGAs are logically organized in a matrix architecture, so we can identify a direct mapping between CLB outputs and bit matrix used in the presented ECC scheme. For the sake of simplicity, we can assume each CLB as a single output function driven by u inputs. We can exhaustively test this function by inputting all the 2^u different input combination, assuming that u is reasonably smaller. Considering the entire FPGA, we can feed all the CLBs with the same input combination and obtain the equivalent bit matrix composed by CLB outputs. At last, we can impose the 2D ECC parity scheme in order to assure that each CLB has the expected output given a certain input. By repeating this procedure over 2^u combinations, we can check the entire functionality of each CLB in the base array. In other words, we can verify that a FPGA circuit is tamper-free and also Trojan-free.

The proposed 2D ECC parity scheme is embedded into a structural implementation. PGs are still selected in a row-wise and column-wise fashion as previously done for the basic ECC technique. An additional hardware unit called *Test Pattern Generator* (TPG) generates the exhaustive sequence of inputs also known as *Test Vectors* (TVs). TPG is connected to each CLB in a PG in order to perform an exhaustive checking over each functional output. Moreover, the parity bit c related with a PG is not stored as redundant information but is structurally mapped to the FPGA by means of an arbitrary parity function. More specifically, for each PG we require 2^u parity bits, one for each possible input combinations. This parity sequence is generated by using an additional CLB connected to the TPG and opportunely configured in order to implement the so-called parity function. In other words, a PG is completed with a CLB capable of calculating its parity bit c for each input combination. At last, another hardware unit called *Output Response Analyzer* (ORA) completes the presented trust-checking schema. ORA implements an associative XOR function (even parity) between PG outputs and the CLB parity function output in order to detect an odd number of tampers (or bit flipping) for any TV. According to the ECC properties, any result different from even parity (zero value) is interpreted as an intentional modification (the technique also covers unintentional modification). Moreover, all the presented trust-checking units can easily be implemented onto the FPGA device using the unused area, giving to the FPGA circuit self-checking capabilities.

Figure 2.2 summarizes the described 2D ECC parity scheme applied for trust-checking purposes. Suppose that gray CLBs contain the application circuit. The TPG unit exhaustively stimulates the highlighted PG including its parity function (blue CLB) whereas the ORA unit calculates the parity, producing a set of ordered parity values called *Parity Vector* (PV). Considering an even parity calculation (obtained using XOR function), the trust-checking structure recognizes a functional modification if the ORA doesn't produce a zero vector $PV = [0,0,\dots,0]$. In the opposite case, the considered PG is tamper-free with very high probability.

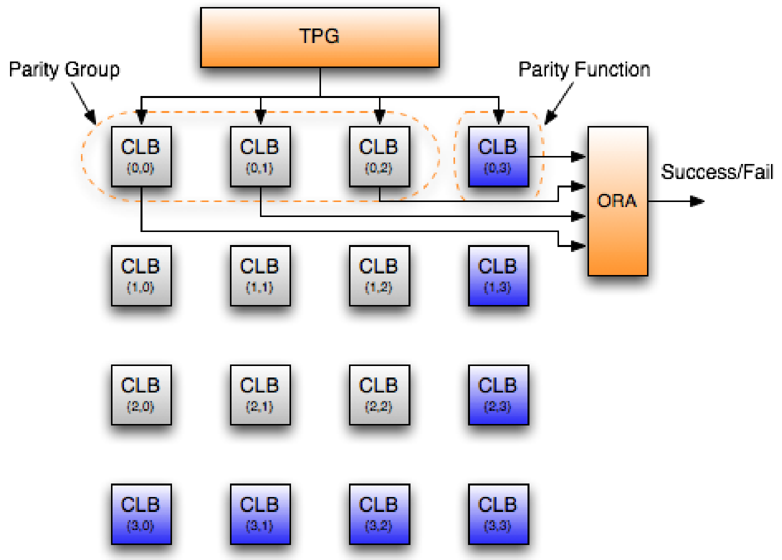


Figure 2.2: 2D ECC parity scheme applied to CLBs

Moreover, the trust-checking needs to be extended to all the PGs in order to cover the entire FPGA circuit assuring trustworthiness with very high probability. Summarizing, the result of a trust-checking phase is a PV which must be observed by outside in order to decide between a success or a fail.

Referring again to Figure 2.2, we consider a single PG at time. The presented trust-checking technique can also be applied in parallel in order to verify multiple PGs at time. A single TPG can be connected to additional CLBs in different PGs by means of additional routing routes. Supposing we have k PGs, we forcedly require k separate ORAs in order to produce k different PV. Depending by the available FPGA resources and by the ORA overhead, we can potentially check the entire FPGA circuit in an unique trust-checking phase.

At last, we point out that the presented 2D ECC parity technique can also prevent Trojan insertions into the unused FPGA area. We can extend the ECC detection capability by mapping PGs over the entire FPGA, forcing the unused CLBs to implement a zero function. With this technical solution, whatever logic insertion will be easily detected since a Trojan represents a functional modification of a zero function.

2.3 Randomization of parity groups

The utilization scenario for the original 2D ECC parity scheme is concerned with bit flipping due to casual occurring faults. Thus, PGs selected in a row-wise and column-wise fashion are sufficient to assure reliability. Assuming a malicious opponent, this selection strategy cannot anymore be considered robust. With an intentional placement of four tampers onto a 2×2 submatrix, the presented scheme fails detection since a masking phenomenon occurs.

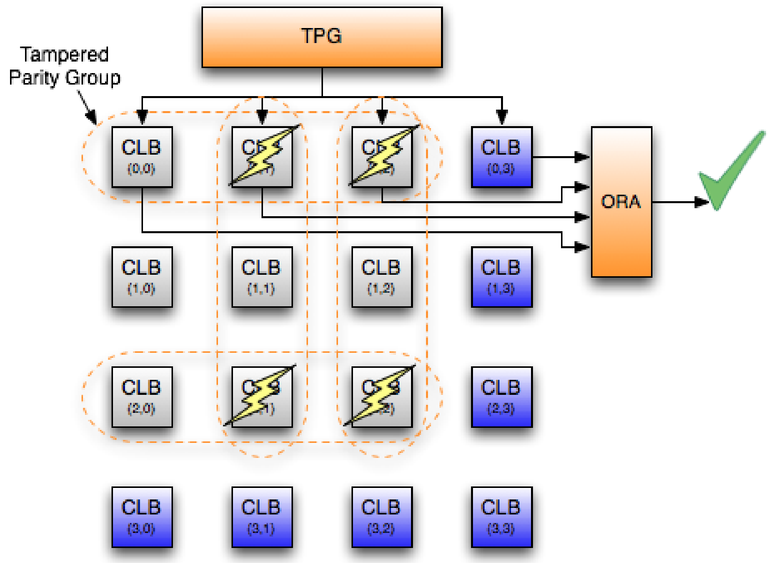


Figure 2.3: Masking

Figure 2.3 shows an example of this naive masking scenario. As we can see, the trust-checking of the first row has success despite the PG is tampered. The same false negative happens the remaining three tampered PGs, leading to a masking phenomenon.

The way around to this drawback involves the use of a randomized PG mapping in order that no adversary can know or easily guess what is the exact placement of the PGs and, consequently, of a 2×2 matrix which causes masking. Given a $m \times n$ matrix composed by CLBs, we define a random mapping r as one-to-one mapping $r : H \times V \to H \times V$ between two sets composed as cartesian product of

$H = \{0, \ldots, m-1\}$ rows and $V = \{0, \ldots, n-1\}$ columns. The randomized PGs are constructed considering row and column PGs and substituting all their elements with the correspondent random mapping r. For instance, consider a row i with PG composed by $[CLB_{\{i,0\}}, CLB_{\{i,1\}}, \ldots, CLB_{\{i,n-1\}}]$. The correspondent randomized PG is $[CLB_{\{r(i,0)\}}, CLB_{\{r(i,1)\}}, \ldots, CLB_{\{r(i,n-1)\}}]$ where each element is obtained using random mapping r.

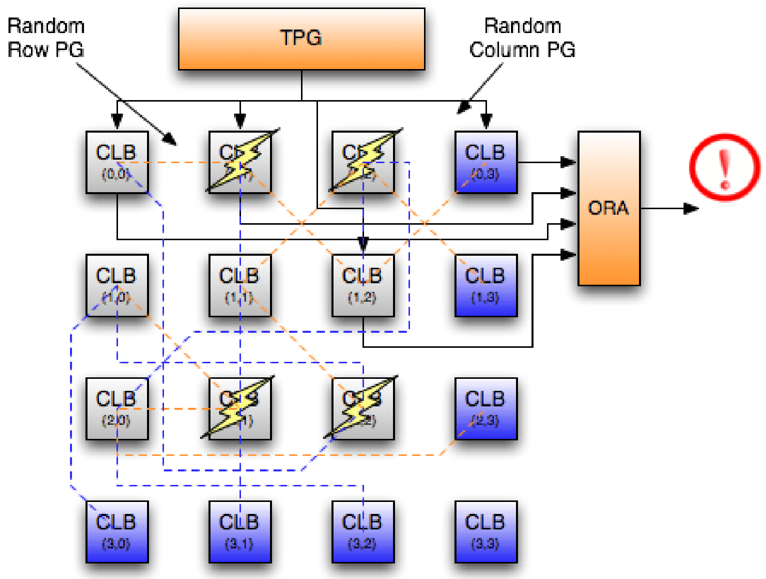


Figure 2.4: Randomized parity groups

Figure 2.4 shows as previous masking is no more effective. In fact, there is a randomized PG (first row) composed by $[CLB_{\{0,0\}}, CLB_{\{0,1\}}, CLB_{\{1,2\}}]$. Since the tamper is unique ($CLB_{\{0,1\}}$), the ORA unit can detect it producing a non-zero PV. In this example, randomization doesn't involve CLBs implementing parity functions. This solution is not robust since it exposes details about the random mapping. More in detail, a malicious adversary can reverse engineering the mapping r by inserting an unique tamper in a CLB and observing what parity functions are involved in the error detection. For instance, if we insert a tamper in $CLB_{\{0,2\}}$ then we can observe a detection with PG involving parity functions $CLB_{\{1,3\}}$ and $CLB_{\{3,2\}}$. We can deduce a correspondence with row 1 and column 2 (in other words, $r(1,2) = \{0,2\}$). A simple way around to this drawback involves the shuffling of the parity functions placement. We can extend the random mapping in order to include not only the PGs but also the parity functions. Subsequently, we need to replace each CLB according to the random mapping r disrupting the optimal FPGA circuit placement calculated during P&R phase.

We can analytically determine what it is the robustness of the presented random mapping technique. Intuitively, it is virtually impossible for an adversary to determine the randomized embedding since he needs to exhaustively analyze an huge set of combinations. Moreover, we can decide to map our random function r using a different domain $H' \times V'$ of cardinality $m' \times n'$ (this represents a different 2D ECC parity scheme) where $m'n' \geq mn$ (in order to cover all the original base FPGA array). For this reason it is necessary to explore all the possible subsets of CLB, verifying that a CLB is the parity function of the others. This means a time complexity of $O(2^{mn})$ so reverse engineering is virtually impossible considering the current FPGA devices size. Taking as reference architecture the Xilinx Virtex 4 [12], we have a CLB array of size 64×24 that corresponds to 2^{1536} possible subsets of CLBs.

2.4 Tamper masking analysis

We have previously depicted a masking scenario for which four errors/tampers inserted in 2×2 submatrix remain undetected by the 2D ECC parity scheme. We can generalize the definition of masking by referring to all the set of similar cases in which multiple bit errors are not detected. Considering a *parity group* PG composed by CLBs, its parity is calculated and verified over an entire set of *test vectors* TVs producing an unique *parity vector* PV. We denote this computation as $XOR_{i=0}^{t-1} f_i$, where $f_0, f_1, \ldots, f_{t-1}$ are t output functions composing the PG. Suppose now to insert tampered functions $\hat{f}_0, \hat{f}_1, \ldots, \hat{f}_{t-1}$ inside the PG. We can define the masking as

$$XOR_{i=0}^{t-1} f_i = XOR_{i=0}^{t-1} \hat{f}_i$$

meaning that both the original and the tampered PVs correspond.

As we can imagine, it is not trivial to tamper a function in order to produce a masking (it is necessary to consider all the input combinations). Intuitively, we can restrict our search space only to certain tamper insertion patterns that may be advantageous for masking goal. Two possible approaches are :

- After we have discovered a set with even cardinality s of equivalent functions $f_0, \ldots, f_{s-1}$ inside the PG, we substitute them with copies of a new tampered functions $\hat{f}$, remembering that $a \oplus a = 0$. Thus we obtain $XOR_{i=0}^{s-1} f_i = XOR_{i=0}^{s-1} \hat{f} = 0$.
- We simply substitute an even number s of output functions inside the PG with their inverses, according with boolean property $a \oplus b = \bar{a} \oplus \bar{b}$.

This latter masking pattern suggests a simple approach to mystify the ECC parity by just complementing the entire set of CLB outputs. This works only with the assumption of a PG composed by even elements since ECC parity is capable of detecting odd errors (according with boolean property $a \oplus b \oplus c \neq \bar{a} \oplus \bar{b} \oplus \bar{c}$). This simple approach can be circumvented by embedding a $m' \times n'$ 2D ECC parity scheme with at least one odd dimension. It might be argued that an adversary can still arbitrarily insert an even number of complemented functions or substitute an even number of equivalent functions for each intersecting PG. However, the randomized 2D ECC parity scheme makes this task very difficult since each PG intersects each other in complex ways and the adversary has no knowledge of the embedded random mapping.

Consider now a scenario in which two generic tampers are inserted in two different functions f_1 and f_2 inside the same PG. It is reasonable to assume that it is extremely difficult to arbitrarily tamper $\hat{f}_1$ and $\hat{f}_2$ in order that $\hat{f}_1(I) \oplus \hat{f}_2(I) = f_1(I_r) \oplus f_2(I_r)$ for each input vector I, at least without using the previous tampering patterns. Thus, it appears the only viable way to induce masking is to randomly insert an even number of the aforementioned two types of tampers, hoping to guess a valid masking placement.

It is clear how an even number of tampers is a necessary condition for masking. A sufficient condition is instead represented by an even number (also including the case of zero) of tampers inserted in each PG. From this consideration, we can roughly quantify which is the probability of masking using a random insertion strategy. According to what previously seen, it might be argued that four is the exact number of tampers for which we have more probability of masking. Intuitively, any number different from a multiple of four means a certain detection (so zero probability of masking) whereas any multiple of fours leads to a lower probability since it involves a greater number of random insertions

We denote the probability of masking as p_{mask}. In simple words, we need to calculate the probability that four tampers are placed in a 2×2 submatrix which corresponds to $\binom{m}{2} \times \binom{n}{2} / \binom{mn}{4} = O((m^2n^2)/(mn)^4) = O(\frac{1}{(mn)^2})$, where the numerator represents the exact number of 2×2 submatrix over the all possible ways in which four tampers can be randomly distributed all over the base FPGA array.

Referring again to the Xilinx Virtex 4 architecture [12], each CLB may have t used outputs (more than a single output considered in our explanation for the sake of simplicity) on which we apply a 2D ECC parity scheme. We choose a square $\sqrt{tmn} \times \sqrt{tmn}$ mapping in order to minimize the p_{mask}. Thus we obtain

$$p_{mask} = \binom{\sqrt{tmn}}{2}^2 / \binom{tmn}{4}.$$

Considering that each CLB has 28 outputs and assuming roughly 70% of the output are used ($t \approx 20$), a 160×160 scheme is mapped to the outputs and $p_{mask} \approx 1.8 \times 10^{-8}$. This conclusively shows how a randomized 2D ECC parity schema is robust against tamper/Trojan insertions.

2.5 Random parity polarities

We have just seen as a randomized 2D ECC parity schema is capable of protecting against tampers inserted in the PGs as well as any malicious alteration of the parity functions. Despite this property, there is still a weak chain link in the overall trust-checking technique regarding the surrounding trust-checking hardware circuits. Suppose that an adversary has knowledge of the design placement (in particular of the ORA or the TPG). In this scenario, he can easily address his attack to these trust-checking circuits, breaking the entire technique reliability. There are two simple ways in which an attack can be implemented. The first one targets the TPG unit in order to skip a certain test vector I which stimulates and exposes the inserted tamper to the 2D ECC parity schema. The second one is concerned with the ORA unit in order to reconfigure its functional output as a zero function and produce an immutable zero vector which hides each tampering. As we can observe, the trust-checking technique fails in both the cases.

The way around to the previous attacks consists of randomly changing the polarity (even or odd) of the parity function used to calculate the expected parity vector. Until now, we have exclusively used an even parity function which produces a zero parity vector $PV = [0,0,\dots,0]$ as ORA output supposing a tamper-free FPGA circuit. We shuffle the expected PV by assigning a random parity function to each different test vector I_i in the exhaustive TVs sequence $S = [I_0, I_1, \dots, I_{2^n-1}]$ where n is the number of bits in a TV. For instance, an arbitrary assignment can correspond to an expected PV similar to $PV = (1,0,0,1\dots,0,1)$. This second level of randomization doesn't imply additional overhead since the random parity function simply substitutes the even parity function. Consider an arbitrary parity group PG_i composed by t outputs. We describe the random polarity assignment related with PG_i using a function P_{even} that is true for those TVs chosen to have even parity and false otherwise. Moreover, we require both the even-parity function O_{even} $(XOR_{i=0}^{t-1} f_i)$ and the odd-parity function O_{odd} $(XNOR_{i=0}^{t-1} f_i)$. Finally, the random parity function O_{random} is expressed as

$$O_{random} = P_{even} O_{even} + \overline{P_{even}} O_{odd}$$

This multiplexed expression permits to obtain a random PV rather than a zero vector. The remaining trust-checking hardware remains unchanged. The ORA still calculates an even parity but the inclusion of the presented random parity function generates an expected PV with random polarities.

We can proof that this technique is effective against both TPG and ORA tamper insertions, resulting in a negligible masking probability. We define $p_{mask}^{tpg-clb}$ as the probability that a tampered TPG can hide a tamper inserted in the CLB array. As mentioned, this attack requires a tampered TPG which produces a modified TVs sequence S' in order to skip TVs which stimulates CLB tampers. Using a random approach, the probability of guessing the correct parity (odd or even) for an arbitrary TV is obviously $1/2$. Suppose that sequence S' differs from the original S for a certain number d of TVs. Given a parity group PG_i, we have $\frac{1}{2^d}$ as probability that d changes will be undetected by the ORA. Moreover, the tampered TPG will stimulate all the parity groups $\{PG_0, PG_1, \dots, PG_{g-1}\}$ in the 2D ECC parity scheme so we can assume that masking probability is

$$p_{mask}^{tpg-clb} = \prod_{i=0}^{g-1} p_{mask}^{tpg-clb}(PG_i) = \frac{1}{2^{g \cdot d}} \le \frac{1}{2^g}.$$

Considering again the Xilinx Virtex 4 architecture [12], we have a 64×24 CLB array that implies $p_{mask}^{tpg-clb} \le \frac{1}{2^{88}}$, an astronomical minuscule value.

Suppose now to replace the ORA (even parity calculation) with an arbitrary function in order to produce the correct PV and hide a CLB tamper insertion. We define this masking probability as $p_{mask}^{ora-clb}(PG_i)$. A CLB tamper can be masked only if we guess the right random polarity (even or odd) associated with one or more particular TVs that normally lead to tamper detection during the trust-checking. Moreover, the tampered ORA still need to verify the remaining TVs. During this checking, it should behave as normal, not producing erroneous results (interpretable as tamper detection). From these considerations, a differential equation arises and its solution is extended in order to consider all the PGs

$$p_{mask}^{ora-clb}(PG_i) \le \frac{1}{2^{g \cdot 2^n}}$$

where n is the number of TPG bits and g is the number of PG. Again, we obtain a negligible probability for the Xilinx Virtex 4 architecture.

2.6 Trusted FPGA design flow

The conventional FPGA design flow needs to be redefined in order to protect the FPGA circuit with the presented randomized 2D ECC parity technique and in order to finally implement a *trusted FPGA design* flow. Some underlying assumptions are necessary. First of all, we assume that the base FPGA array is trusted thanks to the aforementioned separation principle. We assume to have a genuine circuit description expressed using an *Hardware Description Language* (HDL) such as VHDL [26] or Verilog [27]. Moreover, we assume that every EDA tools involved in the *trusted FPGA design* flow is specifically designed for security-critical hardware as suggested in [18]. This assumption is necessary since flawed tools at early design stages can introduce tampers or Trojans before the presented ECC-based technique is applied. Since the trust-checking mechanism is based on a comparison with an expected PV representing the genuine circuit functionality, we require that conventional EDA steps (logic synthesis, technology mapping, placing and routing) produce an early genuine FPGA design on which we can subsequently add trust-checking structures. In other words, this assumption fixes a trusted functionality comparison that can be used in order to detect tampering or Trojan injections during system integration, during device programming or during FPGA circuit operation on the field.

Figure 2.5 presents a *trusted FPGA design* flow as an enrichment of the conventional FPGA design flow. We can observe three distinct macro steps. In the first one, security-critical EDA tools take an high level design and synthesize an FPGA circuit composed by CLBs and routing routes. This step is necessary since the presented randomized 2D ECC parity technique is specifically tailored for FPGA circuits. The integration of trust-checking structures is done by a second step named *trusted FPGA design* phase. We begin with random mapping of the PGs over the functional outputs of the reconfigurable elements. In [3] slices are considered as basic configurable blocks since each CLB is a composition of them. In any case, it is preferable to consider a more general approach based on generic configurable functional outputs in order to easily adapt the technique to any FPGA architecture. Moreover, unused FPGA area is configured as zero-functions and included in the PGs in order to prevent Trojan injections. After random PG mapping, we introduce a second level of randomization by calculating a random parity function for each PG. These functions along with the other trust-checking structures (TPG and ORA) are then synthesized using available reconfigurable area. Finally, the trust-checking mechanism is embedded in the original FPGA circuit according to different approaches highlighted in section 2.7.

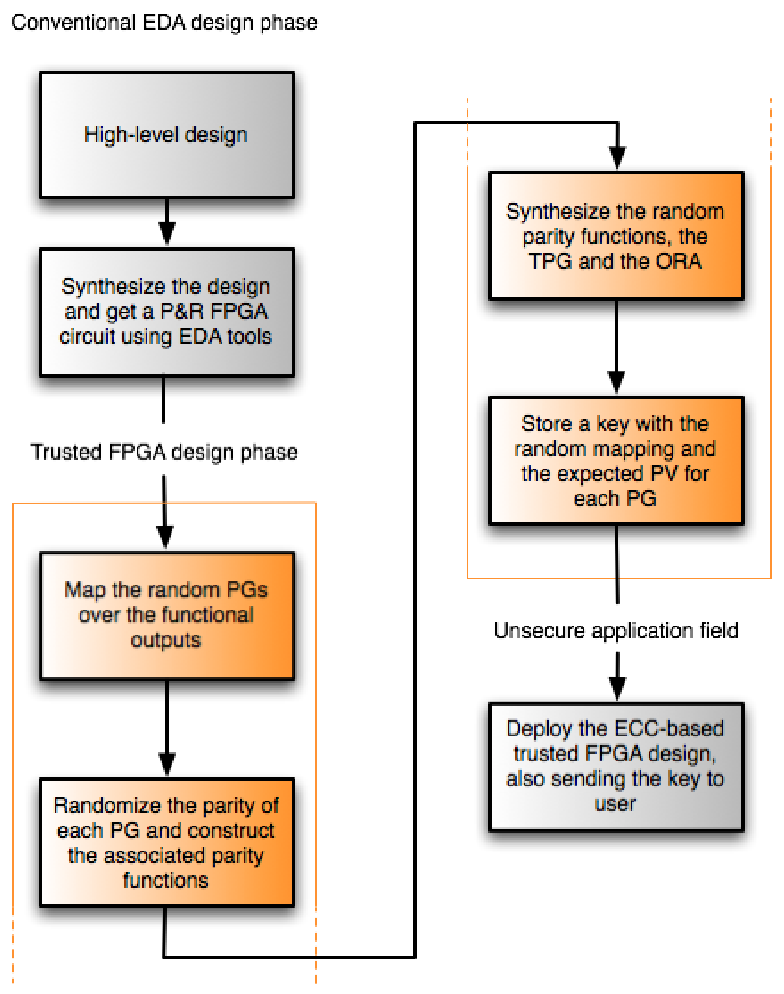

Conventional EDA design phase
High-level design
Synthesize the design and get a P&R FPGA circuit using EDA tools
Trusted FPGA design phase
Map the random PGs over the functional outputs
Randomize the parity of each PG and construct the associated parity functions
Synthesize the random parity functions, the TPG and the ORA
Store a key with the random mapping and the expected PV for each PG
Unsecure application field
Deploy the ECC-based trusted FPGA design, also sending the key to user

Figure 2.5: Trusted FPGA design flow

At last, we have a third phase in which the FPGA circuit is deployed in an unsecure application field. Assuming that trust-checking structure are correctly integrated, the deployed FPGA circuit has the on-chip capability of self-checking its functionality in order to detect any tampering or Trojan injection with very high probability according to the analytical results previously shown. A trust-checking phase is periodically triggered from an external user which is then responsible of comparing the expected PVs (one for each PG) with the PVs produced by the trust-checking hardware and representing the functionality currently implemented by the FPGA circuit. We assume that the expected PVs are calculated as random polarities during the *trusted FPGA design* phase and secretly stored by the user along with the PGs mapping. Moreover, we assume a secure channel with strong cryptography used to communicate the produced PVs. According to these assumptions, an adversary doesn't know the expected PVs and cannot embed them in a fake circuit with different functionality. In other words, only a genuine FPGA circuit can produce the expected PVs.

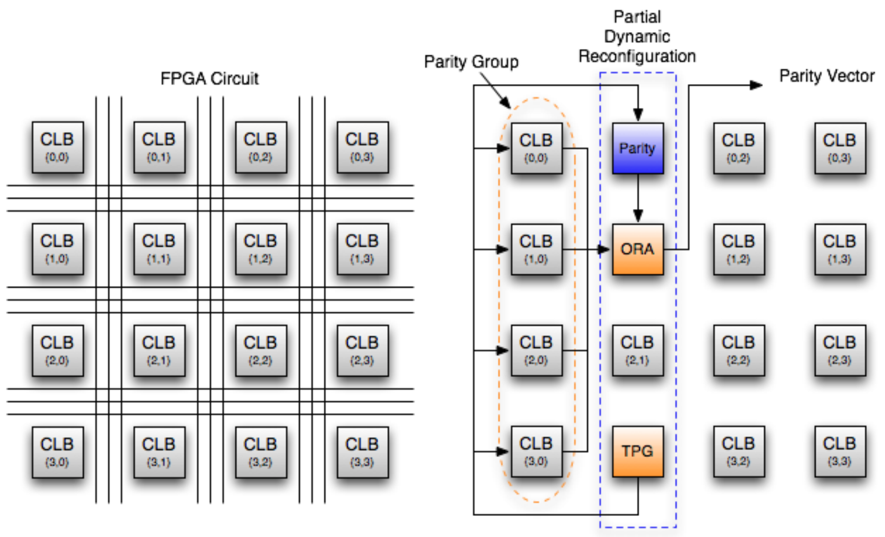
The trusted FPGA circuit is deployed as a monolithic bitstream containing both the original FPGA circuit and the trust-checking components (parity functions, ORA and TPG). This bitstream can easily be used in order to provide a safe system integration. In this scenario, we may have multiple FPGA devices and multiple bitstreams which are dynamically loaded on them. On each of these bitstreams we can trigger an independent trust-checking phase in order to detect any tampering or Trojan injection in the correspondent system component.

2.7 Embedding of trust-checking components

Before application deployment, we need to embed the trust-checking components in the original FPGA circuit in order to implement the presented randomized 2D ECC parity technique. Trust-checking hardware overhead has the advantage of covering some of the unused CLBs, the sames that require to be configured as zero-functions and included in the PGs in order to avoid Trojan insertions. For this reason, FPGA resources may represent a constraint on the applicability of the technique only in case of small FPGA devices or large trusted FPGA circuits. However, dynamic and partial reconfigurability can overcome this constraint by loading trust-checking components only when needed. Thus, we can identify three different embedding approaches that expands the applicability of the trust-checking technique by making use of the available FPGA device features.

2.7.1 Non Integrated Embedding

This approach known as *Non Integrated Embedding* (NIE) implies that no trust-checking component is statically integrated on the device. In fact, only the original circuit is placed and routed onto the FPGA device whereas the parity functions, the ORA unit and the TPG unit are dynamically configured only when a trust-checking phase is needed. Depending by the available area, it is possible to verify one PG at a time or multiple PGs simultaneously, speeding up the entire trust-checking phase. In that case, we can share the TPG unit between all the PGs whereas we need a separate parity function and ORA unit for each PG. Moreover, the FIE approach permits to reconfigure part of the original FPGA circuit (the other PGs not under trust-checking) without requiring additional free CLBs outside the FPGA circuit.



(a) FPGA circuit (b) Trust-checking phase

Figure 2.6: Non Integrated Embedding

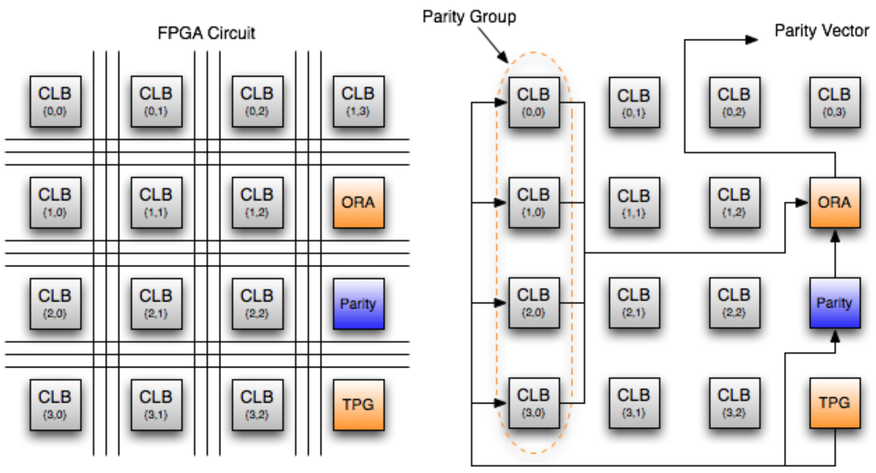
Figure 2.6 depicts the NIE scenario during trust-checking phase. A partial dynamic reconfiguration permits to instantiate the trust-checking components necessary to verify a single PG. Reconfiguring used CLBs we can virtually apply the randomized 2D ECC parity technique to any FPGA device sufficiently large to contain the original circuit. The only regard is to consider one or few PGs at a time until all the FPGA is covered and to restore the original circuit when the trust-checking phase ends.

The great advantage of the FIE approach is that no area overhead is required during the normal circuit working. In other words, there are no timing delays since the trust-checking components are instantiate only when needed. Moreover, the routing of the trust-checking is not statically embedded in the FPGA circuit so an adversary have no possibility to steal the design and reverse engineering the PGs composition. On the other hand, the NIE approach requires partial dynamic reconfiguration, a feature that does not come for free since it is not offered by all the FPGA families. When available, reconfigurability requires to disable bitstream encryption opening a possible attack window related to active reading/writing of the FPGA device. At last, the FIE approach slows down the trust-checking phase since the added reconfiguration phase is intrinsically slow.

2.7.2 Partially Integrated Embedding

This approach known as *Partially Integrated Embedding* (PIE) is similar to the NIE approach but supposes an FPGA device capable of reconfiguring just the routing. The trust-checking components are synthesized and statically placed into the unused CLBs along with the FPGA circuit. More in detail, it is necessary to place all the parity functions, a TPG unit and a certain number of ORA units which determines the number of PGs on which we can simultaneously apply the trust-checking. In the normal working scenario, the programmable interconnection network implements the original circuit whereas trust-checking components are disconnected. When a trust-checking phase is requested, the connections around one or more PGs are rerouted in order to implement the parity scheme. This rerouting procedure is then iterated until all the PGs are checked for tampering.

Figure 2.7 depicts the PIE scenario. In Figure 2.7a, we see the FPGA circuit along with trust-checking components placed but not routed on the right column (for the sake of simplicity, we just consider the components necessary for checking a single PG). The trust-checking phase is shown in Figure 2.7b. A routing reconfiguration connects the TPG unit to the entire PG along with the correspondent random parity function. Moreover, the ORA units calculates parities in order to produce a PV and detect possible tampering in the current PG.



(a) FPGA circuit (b) Trust-checking phase

Figure 2.7: Partially Integrated Embedding

The PIE approach shares some strengths and weaknesses with the NIE approach. However, it is only implementable with an FPGA device sufficiently large to contain the original circuit along with the trust-checking components (as mentioned, they are statically placed). Moreover, the PIE approach offers a speed advantage compared to the NIE approach since reconfiguration only involves rerouting and not CLB functionalities.

2.7.3 Fully Integrated Embedding

This approach known as *Fully Integrated Embedding* (FIE) embeds all the trust-checking components along with the original FPGA design. Considering the previous approaches, the switching between normal circuit working and trust-checking phase is done using routing reconfiguration. In the FIE scenario, the trusted FPGA circuit is fully placed and routed so we implement a structural switching by the means of multiplexers. More in detail, each CLB input has a 2:1 multiplexer in order to disconnect the original circuit network and connect the TPG. As the other trust-checking components, this 2:1 multiplexer is implemented by means of the available FPGA area. On the other side, each CLB output has two additional fanouts in order to be connect to two ORA units corresponding to its row PG and its column PG. The FIE approach offers hardware parallelism to the trust-checking phase by the means of multiple placed and routed ORAs. The alternative solution multiplexes one or few ORAs between all the PGs. However, this solution involves the use of large multiplexers so its hardware overhead is comparable with the completely parallel solution, which is obviously better in terms of speed.

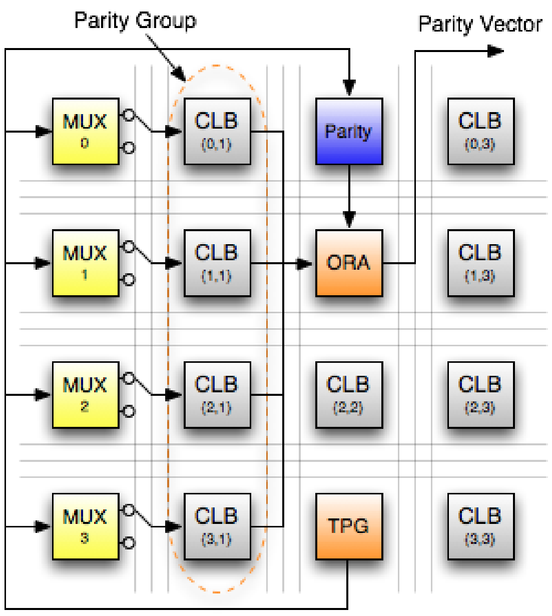


Figure 2.8: Fully Integrated Embedding

Figure 2.8 shows a simplified FIE scenario where some CLBs implement multiplexers for switching between normal circuit working and trust-checking phase (for the sake of simplicity, Figure 2.8 just shows the TPG connections). Moreover, all the trusted components are already placed and routed on the FPGA device in order to avoid reconfiguration.

The FIE approach has a considerable area overhead due to multiple ORAs and to switching multiplexers. Moreover, there is an added timing delay since multiplexers are interposed in the circuit critical path composed by CLBs. Another drawback is related with the embedding of PGs routing that can potentially be reversed, leading to a security issue for the entire ECC-based technique. However, the FIE approach has the main advantage of not requiring reconfiguration, thus permitting implementation of the randomized 2D ECC parity technique over a larger set of FPGA families. Reconfiguration is slow so the FIE approach has an averagely faster trust-checking time, also thank to multiple ORA units. In addition, it is still possible to use bitstream encryption in order to avoid active reading/writing of the FPGA device. Remote attacks can be undetected in the NIE approach since the on-chip configuration bits may be overwritten during trust-checking, similarly to scrubbing technique in [19]. This does not happen in the FIE approach where detection may alert the system integrator in order to adopt additional countermeasures such as shielding the system. Least but not least, the FIE approach allows very easy trust-checking since the trusted FPGA circuit is deployed as monolithic application and the user simply provides a triggering signal, thereby avoiding the more onerous multi-reconfiguration trust-checking process of the other approaches.

In this thesis we explicitly consider the FIE approach since it is the only one that can be combined with bitstream encryption offering a superior level of protection and assuring a *trusted FPGA design*. There are several challenging design issues for the FIE approach, including keeping hardware and performance overheads within acceptable limits or hiding PGs compositions. These issues are tackled in the next chapters proposing structural and methodological innovations that improve the ideas originally proposed in [3].

Chapter 3

An improved Fully Integrated Embedding

The FIE approach represents the best solution from the perspective of *trusted FPGA design* since it is the only embedding approach that can combine the effectiveness of the randomized 2D ECC parity technique with the protection assured by bitstream encryption. In fact, no sensitive military or commercial FPGA-based applications is deployed without encryption, at least for protecting the underlying IPs related with the circuit design. By the way, the basic FIE structure proposed in [3] is still slightly raw for being efficiently implemented on a realistic FPGA device. There are several challenging design issues, including keeping hardware and performance overheads within acceptable limits. The heaviest contribution to these overheads come from switching multiplexers which have an expensive k-LUT implementation. Moreover, the circuit critical path doubles its length due interposed multiplexers. Aims of this chapter is to address these problems by proposing an *improved Fully Integrated Embedding* (iFIE) approach based on structural and methodological innovations. Another worrying issue of the FIE approach is related with the monolithic trusted FPGA circuit deployed on the field. In fact, it still embeds all the secret information on which the randomized 2D ECC parity technique relies. Despite reconfiguration, the NIE and PIE approaches do not offer a greater advantage since PGs composition can be inferred from routing reconfiguration, an information which is stored as unencrypted bitstream in the FPGA device memory. We propose a *Reconfigurable Error Correcting Code* (RecECC) parity scheme capable of changing its composition at any trust-checking phase, implying no static embedding in the deployed FPGA circuit. This novel parity scheme is also robust against any kind of reply attacks involving PV theft.

The chapter is organized in the following way. Section 3.1 analyzes all the sources of hardware and performance overheads which make the basic FIE implementation very inefficient. Section 3.2 presents a first solution adopted by the iFIE approach in order to drastically decrease the hardware overhead related with multiplexers. In section 3.3 we introduce a slight modification of the structural 2D ECC parity technique in order to avoid the use of parity functions and the consequential hardware overhead. Section 3.4 presents the most important innovation of the iFIE approach. Instead of considering single reconfigurable elements, we apply functional trust-checking to more coarse-grained cone subcircuits in order to avoid multiplexers on the internal connections. For this reason, a new cone generation step is added in the *trusted FPGA design* flow. Finally, section 3.5 introduces the novel RecECC parity scheme and analyzes its advantages and robustness compared to the conventional 2D ECC parity scheme.

3.1 Hardware and performance overheads

The FIE approach cannot make use of reconfiguration so it implements the switching between normal circuit working and trust-checking phase by the means of 2:1 multiplexers. These hardware components, along with the other expected trust-checking units, should be synthesized, placed and routed using the available reconfigurable FPGA area. Looking at the practical result, a single PG composed by some CLBs will need several times its area for implementing the related trust-checking mechanism, leading to a tremendous problem in terms of hardware overhead. In other words, the basic FIE approach is not usable in practice since it is inconceivable for the circuit complexity to be restricted to only a small fraction of the available FPGA device area.

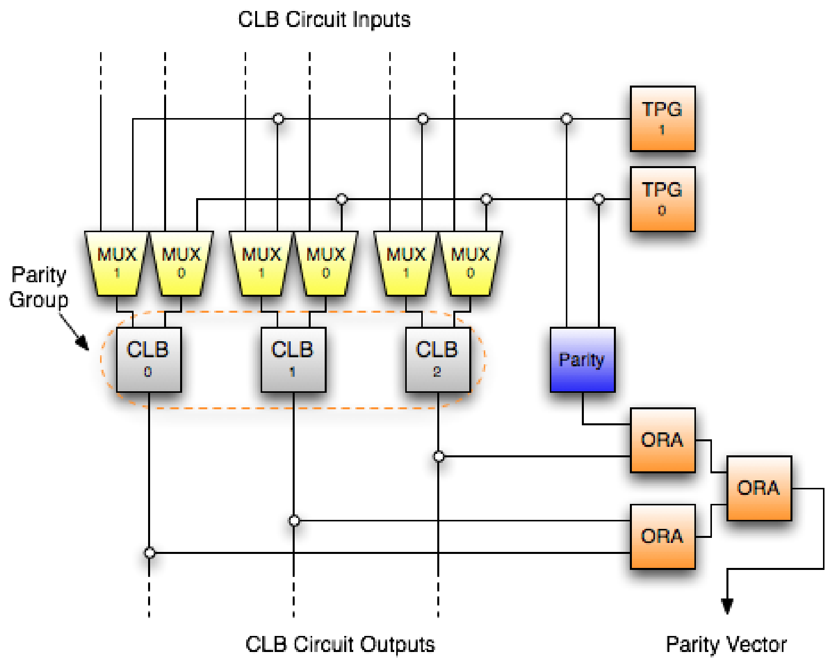


Figure 3.1: FIE structural overhead

Figure 3.1 gives us a rough idea of the overhead related with the FIE trust-checking of a PG. For the sake of simplicity, we assume a very simple FPGA architecture with 2-input CLBs capable of implementing any sequential or combinatorial trust-checking component (obviously, a 2:1 multiplexer should require an additional input for selection). In the presented example, the trust-checking logic needs so much as four times the CLBs composing the PG. Moreover, it might be argued that the multiplexers represent the heaviest contribution due their expensive k-LUT implementation whereas other components can take benefit from sharing. Differently from what happens in other approaches based on reconfiguration, the FIE structure also affects the circuit delay in its normal working. In fact, the circuit critical path may traverse additional CLBs or some CLBs may have additional output capacitance due to an increased fanout. In this section we propose a detailed analysis regarding both hardware and performance overheads introduced by the different trust-checking components.

3.1.1 Test Pattern Generator

The TPG unit is used for producing an exhaustive sequence of TVs in order to verify the functionality of reconfigurable elements. In other words, the TPG is composed by p lines and generates binary numbers from 0 to $2^p - 1$. For this reason, its implementation simply consists of a sequential p-bit counter which occupies a proportional number of CLBs. This hardware overhead may become neglegible when an unique TPG is shared between all the PGs. Intuitively, during a trust-checking phase we need that all the functional outputs inside a PG are stimulated with the same PV. Considering two different row parity groups PG_x and PG_y, there is a logic independency between the two trust-checkings. In other words, we can stimulate PG_x and PG_y with different exhaustive TVs sequences S_x and S_y. It might be argued that there is no counter-indication if we use the same TVs sequence S for both the PGs. For this reason, we can simultaneously connect an unique TPG unit to all the row PGs. Consequently, the overlapping column PGs will be stimulated with the same TV sequence S. Again, there is no counter-indication since the column PGs are logically independent. This finally proves that it is possible to share an unique TPG between all the PGs.

The TPG unit does not introduce any performance overhead since it is normally disconnected from the original circuit by the means of multiplexers. On the other hand, the TPG placement can produce a less optimized FPGA layout compared to the scenario with only the original circuit. Moreover, the routing of p lines from an unique TPG to every CLB may congest the programmable interconnection network. This scenario can be prevented by using multiple TPGs distributed over the entire FPGA layout. Their placement can follow an H-tree [28] which is commonly used in *Very Large Scale Integration* (VLSI) design as a clock distribution network and assures an uniform covering of the entire layout.

3.1.2 Output Response Analyzer

The ORA unit is used for calculating the even parity considering t functional outputs $f_0, f_1, \ldots, f_{t-1}$ inside a PG and the correspondent random parity function

$f_t = O_{random}$. This calculation is done according to the formula

$$XNOR_{i=0}^{t} f_i$$

and produces the PV during the trust-checking phase. An even parity function is essentially a combinatorial logic function of $(t+1)$ variables easily implementable by using k-LUTs. Depending by the FPGA architecture, each CLB contains one or more of these programmable logic functions. Composing enough k-LUTs, we can obtain an arbitrary $(t+1)$-input function such as the even parity function. The hardware overhead in terms of k-LUTs can be calculated as

$$ORA_{ovhd} = \left\lceil \frac{t}{k-1} \right\rceil (LUTs)$$

This calculation has a simple derivation. Consider a single k-LUT implementing a k-input function. In order to implement a larger function, we need another k-LUT that will be connected to one of the available k inputs. This addition increases the number of available inputs by $k-1$ since k inputs are added but one is consumed. From this reasoning, we can infer the aforementioned formula. Moreover, the k-LUT circuit should be composed using a hierarchical structure in order to minimize the ORA latency measured in traversed k-LUTs.

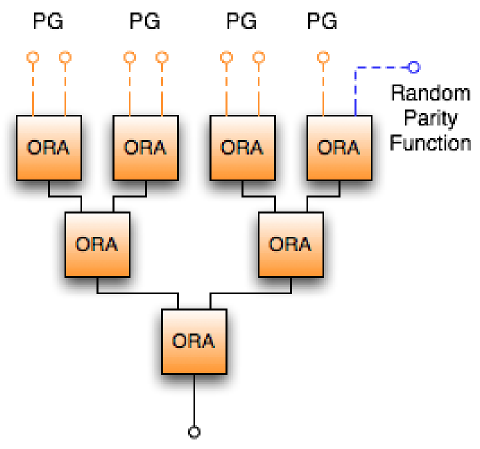


Figure 3.2: ORA unit implemented by k-LUTs

Figure 3.2 shows an example of ORA unit considering 2-LUTs and a PG with

$t = 7$. As we can see from the picture, the hardware overhead corresponds to $ORA_{ovhd} = \lceil 7/1 \rceil = 7$. Moreover, we can calculate the ORA latency in terms of k-LUTs as

$$ORA_{latency} = \left\lceil \log_k(t+1) \right\rceil (LUTs)$$

which corresponds to $ORA_{latency} = 3$ for the considered example. Obviously, this latency does not affect the normal circuit working but only the trust-checking phase where the logic signal traverses the ORA unit in order to produce the PV.

Each PG needs an ORA unit in order to perform the trust-checking. As mentioned, the best solution consists of a parallel implementation where multiple dedicate ORAs (one for each PG) are used. An alternative solution involves one or few ORAs which are serially multiplexed between the PGs. We can show that the parallel solution has a better hardware overhead. According to a k-LUT implementation, the total ORA overhead is proportional to

$$ORAs_{ovhd} \propto g \cdot t$$

where g is the number of PGs and t is the number of functional outputs inside a PG. Regarding the serial solution, suppose to use an unique large multiplexer for switching t lines between g different PGs. In other words, we have a t combinatorial functions with hardware overhead proportional to

$$LargeMUX_{ovhd} \propto (g + \log_2 g) \cdot t$$

where the logarithmic term is due to the selection inputs. Moreover, we need to consider a shared ORA unit connected to the multiplexer outputs. These considerations leads to the following disequation

$$g \cdot t \ \leq (g + \log_2 g) \cdot t + t$$

which proves that the parallel solution is generally better in terms of hardware overhead. There are other reasons for which we prefer a parallel implementation. First of all, it is possible to speed up the entire trust-checking phase thanks to the hardware parallelism. Moreover, the serial architecture has a longer critical path since the logic signal should traverse an additional large multiplexer. At last, dedicate ORAs can easily fit a 2D ECC parity scheme with $m \neq n$ whereas the serial implementation has an intrinsic waste since half of the PGs will necessarily use a shared ORA unit which is larger than necessary. For instance, in a CLB array of size 64×24 the column parity is calculated using a 65-input shared ORA whereas a 25-input ORA is sufficient.

Multiple ORAs introduce a marginal performance overhead in the original FPGA circuit. More in detail, each functional output drives two additional fanout corresponding to the parity calculation for a row PG and for a column PG. Again, the ORA placement can produce a less optimized FPGA layout compared to the scenario with only the original circuit. In order to produce and simultaneously communicate multiple PVs, we need a certain number of available IOBs on the FPGA device. If this is not possible, we can share a smaller number of IOBs by using a memory buffer.

3.1.3 Parity functions

A random parity function O_{random} is necessary for each PG in order to complete the trust-checking architecture. We have seen that any combinatorial logic function can be implemented by the means of k-LUTs. A random parity function has an input size of p variables where p represents the size in bits of the TVs which exhaustively stimulate the functional outputs inside the PG. For this reason, we can calculate the hardware overhead in terms of k-LUTs as

$$Parity_{ovhd} = \left\lceil \frac{p-1}{k-1} \right\rceil (LUTs)$$

This quantity needs to be multiplied by g in order to estimate the total hardware overhead related with random parity functions.

The previous calculation assumes homogenous reconfigurable elements in terms of active inputs. In the common scenario, a CLB can be configured in different ways and each of them corresponds to different number of active inputs. For instance, suppose that an FPGA circuit has half of its CLBs configured as 8-bit functions and half configured as 16-bit functions. It might be argued that it is possible to compose a PG with only 8-bit functions. Moreover, we can reduce the hardware overhead for this PG by using an 8-bit random parity function. According to this reasoning, in [3] it has been suggested to partition the functional outputs in terms of their complexity and to apply different 2D ECC parity scheme in order to reduce the hardware overhead associated with parity functions. For instance, in the given example we have a bipartition of 8-bit functional outputs and 16-bit functional outputs. We apply a first 2D ECC parity scheme where we only consider 8-bit PGs. The TPG unit will stimulate these functional outputs with 8-bit TVs (we connect 8 out of 16 available lines). A second 2D ECC parity scheme is then applied to 16-bit PGs. Depending by the distribution of reconfigurable elements into different functional classes, this approach may reduce the hardware overhead. On the other hand, this functional classification decreases the masking probability as shown in [3]. In fact, we have an added constraint for which four tampers must be contained in the same functional class in order to generate a masking scenario.

The random parity functions do not directly introduce any performance overhead in the original circuit since they are only connected with other trust-checking components. By the way, the parity functions placement may produce a less optimized FPGA layout compared to the scenario with only the original circuit.

3.1.4 Switching multiplexers

The switching multiplexers represent the most penalizing factor in terms of hardware overhead. There are two underlying reasons for their heavy contribution. The first reason is related with the k-LUT implementation of the switching multiplexers. As we have seen, a k-LUT is designed to implement an arbitrary k-bit function and it can obviously be used for a 2:1 multiplexer supposing that $k \geq 3$ (usually $k \approx 4,5$). Considering that each CLB has a limited number of k-LUTs, a multiplexer is expensive in terms of FPGA reconfigurable logic (much more than ASICs). The second reason for an heavy multiplexer overhead is concerned with the FIE structure which requires a 2:1 multiplexer in front of each functional input. Considering a simple k-LUT function, we need so much as k additional 2:1 multiplexers in order to switch from normal circuit working and trust-checking phase.

Referring to the Xilinx Virtex 4 architecture [12], a typical circuit is composed by slices containing two 4-LUTs and using an average of $k \approx 7$ active inputs. In order to implement the FIE structure, we require three and a half slices which directly leads to an unacceptable hardware overhead of 350%. This simple reasoning can be validated by analyzing some circuits taken from ITC99 benchmarks [29].

Benchmark	Circuit	Size (slices)	Multiplexer Overhead
b11	Scramble string	75	373.33%
b12	Guess a sequence	220	340.91%
b14	Viper processor	1275	358.98%
b15	80386 processor	1349	345.14%
b17	Three copies of b15	4106	344.40%
b20	Two copies of b14	2384	367.37%

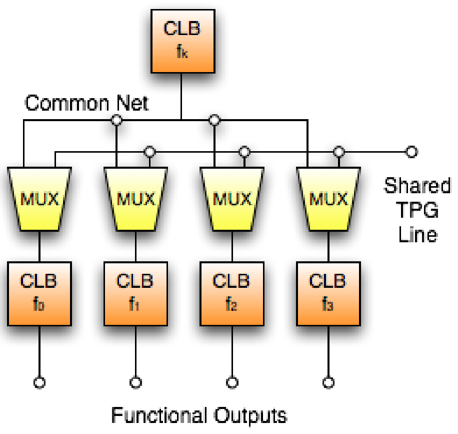
Table 3.1: Multiplexer overhead

Table 3.1 shows how the average multiplexer hardware overhead is close to the predicted one. For this reason, the basic FIE approach is limited to small circuits which fit on a portion of the FPGA device. Another drawback of switching multiplexers is concerned with the performance overhead. Each CLBs has a set of multiplexers on its inputs so each circuit path lenght, included the critical ones, is virtually doubled. In other words, we have have a performance overhead of 100%. Summarizing, the multiplexer overhead represents the most penalizing contribution on which the proposed iFIE approach will focus the optimizations

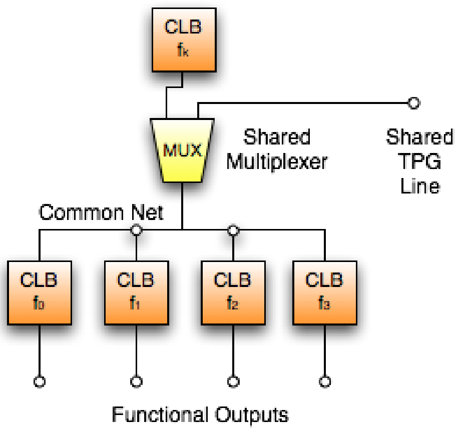
3.2 Multiplexer sharing over common nets

The basic FIE approach counts for a 2:1 multiplexer placed in front of each functional input leading to an huge hardware overhead. The iFIE approach introduces a structural improvement in order to reduce the number of required multiplexers. This optimization is based on some considerations about the assignment of the TPG lines. Consider a functional outputs f inside a PG. The goal of the trust-checking phase is to exhaustively verify all the $2^p - 1$ truth table combinations of f by the means of a structural 2D ECC parity scheme. It might be argued that these truth table combinations can be verified in an arbitrary order. The TPG units generates a TVs sequence from 0 to $2^p - 1$. However, the current truth table combination stimulated by the TPG depends by the assignment between TPG lines and functional inputs $[i_0, i_1, \ldots, i_{p-1}]$ for f. In other words, there exists $t!$ possible permutations and each of them corresponds to a different order in which the TVs sequence exhaustively stimulates the truth table. Each permutation is logically equivalent from the point of view of the trust-checking despite it may correspond to a different sequence of logic values produced by f. Considering an entire PG composed by t functional outputs, we have $(p!)^t$ possible TPG assignments and thus $(p!)^t$ possible parity functions O_{even} and O_{odd}. However, when the connections from the TPG to the functional inputs are permanently routed we end up with precise parity functions O_{even} and O_{odd}. These are then embedded in a random parity function O_{random} in order to implement the trust-checking mechanism.

Given two functional outputs f_i and f_j, it might be argued that any TPG lines assignment is acceptable. Suppose that a functional input i_i for f_i and a functional input i_j for f_j lie on the same net n. In other words, in the original FPGA circuit a third functional output f_k has i_i and i_j in its fanout by the means of a shared net n. As we have seen, we can assign the same TPG line to i_i and i_j. This choice may be beneficial in terms of multiplexer hardware overhead. According to the FIE structure, we should place a multiplexer in front of i_i and a multiplexer in front of i_j. Since the two functional inputs i_i and i_j are driven by the same TPG line and since both the outputs f_i and f_j are simultaneously verified during the trust-checking phase, we can also share an unique multiplexer by placing it upstream on the net n. Generalizing, we can share a multiplexer over any common net by substituting the multiplexers placed in front of any functional input with an unique multiplexer placed upstream on the net. An example of multiplexer sharing is shown by Figure 3.3. For the sake of simplicity, we consider CLB functional outputs with an unique input.



(a) Input multiplexers



(b) Net multiplexer

Figure 3.3: Benefit of sharing a common multiplexer

As we can see, f_0, f_1, f_2 and f_3 are driven by the functional output f_k so they have an input net in common. By placing upstream on the net a shared multiplexer, we can reduce the hardware overhead by 75% (three multiplexers). Referring to the Xilinx Virtex 4 architecture [12], a typical circuit have nets with approximately an average fanout of 3.8. Applying the net multiplexer sharing, the iFIE approach can reduce the multiplexers to $(1/3.8) \approx 26\%$. In other words, the hardware overhead drastically decreases from 350% to 91%. On the other hand, there are no improvements concerning the performance overhead since the same number of multiplexers is placed on every path, including the critical ones.

Suppose now that two functional inputs i_i and i_j for the same output f share an input net n. Apparently this scenario may cause problem to the net multiplexer technique since the two inputs are short-circuited to the same TPG line reducing the range of input combinations. In other words, we are not anymore able to exhaustively verify the configuration of output f since we skip all the combinations where $i_i \neq i_j$. However, this does not represent an issue from a functional point of view. More in detail, the short-circuit between i_i and i_j is originally embedded in the FPGA circuit network so no input combination can have $i_i \neq i_j$. A configuration tamper for input combinations where $i_i \neq i_j$ is not reachable in any case so there is no effective functional modification in the FPGA circuit.

Sharing a TPG line between functional inputs introduces some constraints for the overall TPG routing. Consider a set of distinct functional inputs $[i_0, i_1, \ldots, i_{p-1}]$ for an output f. In order to exhaustively stimulate f, we need a biunivocal mapping between p inputs and p TPG lines. It might be argued that we can easily choose any biunivocal assignment for those inputs which are not included in a shared net fanout. However, the remaining inputs have an added constraint for which inputs in the same net fanout must share the same TPG line assignment. This added constraint may lead to an unfeasible biunivocal TPG assignment, at least using only p lines. An example of this scenario is shown by Figure 3.4.

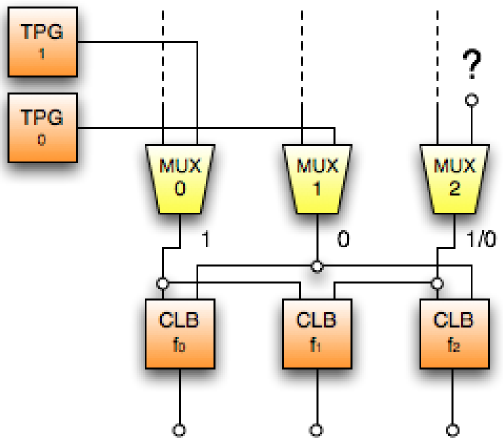


Figure 3.4: Conflict in the TPG assignment

As we can see, there is a conflict in the TPG line assignment regarding the last multiplexer. In fact, we cannot assign neither line 0 nor line 1 since both are not a biunivocal assignment for f_2 and f_1. This conflict can only be solved by increasing the TPG size. In other words, we insert an additional line 2 that will be connected to the last multiplexer in order to solve the assignment conflict.

We can generalize the TPG assignment problem in terms of graph coloring problem. We construct a graph where each node represents a net in the FPGA circuit. We add an edge for each pair of inputs inside the same functional output in order to represent the biunivocal constraint. Moreover, we add an edge for each pair of inputs inside the same shared net in order to represent the sharing constraint. Coloring this graph, we obtain a solution for the initial TPG assignment problem since each distinct color represents a distinct TPG line. It might be argued that the coloring solution requires at least p colors since each functional output has p inputs. In case of more colors, we only require a larger TPG units despite this increases the hardware overhead and the duration of the trust-checking phase.

3.3 Trust-checking without parity functions

The basic FIE approach proposes a structural 2D ECC parity scheme where each parity group calculates its current parity. This value is then compared with the expected parity value produced by a random parity function in order to produce zero (even polarity) or one (odd polarity) in the output PV. Finally, we can detect tampering or Trojan insertions by an off-chip comparison with the expected PV. Intuitively, there is an intrinsic inefficiency in this approach since two similar comparisons are performed at different stages.

The iFIE approach introduces an architectural improvement for which we substitute the off-chip polarity comparison with a parity comparison. In this way it is not necessary to embed random parity functions in the trusted FPGA circuit. We modify the trust-checking mechanism in order that the sequence of even parities for a PG is contained by the correspondent PV. Thus, we perform an off-chip comparison with the expected PV representing the genuine parity sequence. In other words, each parity function is fully described by an off-chip expected PV obtained by algebraic manipulations or by simulation so there is no reason for a parity function to be implemented inside the trusted FPGA circuit. It might be argued that this iFIE approach has robustness similar to random polarities. Assuming arbitrary CLB functions and random PGs, the expected PV can be considered as a pseudorandom binary vector similar to a random polarity vector. For instance, PV will be a vector of all 0's (thus becoming equivalent to a non randomized scheme) with very low probability. This is sufficient for saying that the trust-checking mechanism remains robust against TPG and ORA tampering.

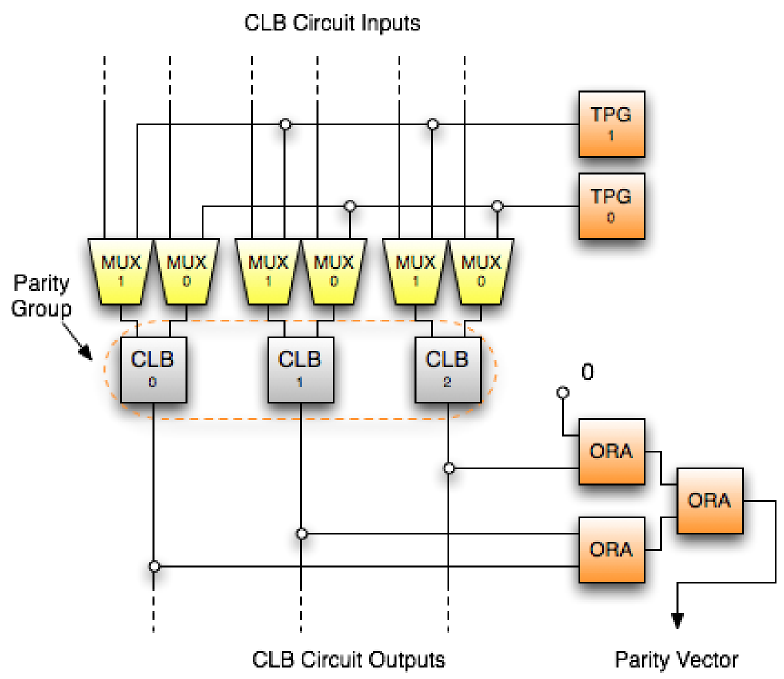


Figure 3.5: iFIE structural overhead

Figure 3.5 shows the immediate benefit in term of hardware overhead of the iFIE structure which does not use embedded parity functions. More in detail, each removed parity function corresponds to a saving of $\left\lceil \frac{p-1}{k-1} \right\rceil$ k-LUTs where p is the number of TPG lines. The total overhead benefit is then obtained by considering all the PGs. Moreover, each ORA unit needs one less input so even their hardware overhead slightly decreases to

$$ORA_{ovhd} = \left\lceil \frac{t-1}{k-1} \right\rceil (LUTs)$$

where t is the number of functional outputs in a PGs.

Without parity functions it is easier to apply the ECC-based technique since we can avoid the functional classification outlined in section 3.1. Given n functional outputs, we construct a squared 2D ECC parity scheme with size $\sqrt{n} \times \sqrt{n}$. We can analytically show that a squared scheme minimizes the mask probability p_{mask}. Suppose to have a parameter a that determines a 2D scheme of size $a \times \frac{n}{a}$ over n functional outputs. According to section 2.4, we can calculate the masking probability as

$$p_{mask} = \frac{\binom{a}{2} \cdot \binom{n/a}{2}}{\binom{n}{4}} = \frac{\frac{a(a-1)}{2} \cdot \frac{(n/a)(n/a-1)}{2}}{\binom{n}{4}} = \frac{n}{4\binom{n}{4}} \Big[(a-1)(n/a-1)\Big]$$

In order to calculate the p_{mask} minimum, we consider the derivative

$$\frac{dp_{mask}}{da} = \frac{n}{4\binom{n}{4}} \Big[(n/a-1)(a-1)(n/a^2)\Big] = \frac{n}{4\binom{n}{4}} \left[\frac{an - a^2 - an + n}{a^2}\right]$$

Setting $dp_{mask}/da = 0$, we obtain $a = \sqrt{n}$. This proves that a squared 2D ECC parity scheme of size $\sqrt{n} \times \sqrt{n}$ minimizes the masking probability p_{mask}. The iFIE approach will always use a squared mapping in conjunction with PGs randomization. We can estimate the total hardware overhead associated with the ORA units. It might be argued that given a $\sqrt{n} \times \sqrt{n}$ scheme the PG size in terms of functional outputs is $t = \sqrt{n}$ whereas the number of PGs is $g = 2\sqrt{n}$. Substituting these parameters in the formula proposed in section 3.1, we obtain the total hardware overhead as

$$ORAs_{ovhd} = g \left\lceil \frac{t-1}{k-1} \right\rceil = 2\sqrt{n} \left\lceil \frac{\sqrt{n}-1}{k-1} \right\rceil (LUTs)$$

This shows a linear dependency $O(n)$ between circuits size n in terms of functional outputs and the total hardware overhead related with the ORAs units. In a real implementation of the iFIE approach we have a slightly large hardware overhead since the unused CLBs and their functional outputs are included in the 2D ECC parity scheme in order to avoid Trojan insertions. It might be argued that this inclusion increases the scheme size as well as the ORA overhead. On the other hand, these outputs configured as zero functions do not require multiplexer overhead since they are not involved with the switching between normal circuit working and trust-checking phase. At last, we should say that hardware overhead increasing related with unused CLBs can be considered negligible since it intrinsically helps to cover those unused functional outputs otherwise target for Trojan injection.

3.4 Cone-based iFIE approach

We have seen that the switching multiplexers have the heaviest contribution in the hardware overhead associated with the basic FIE approach. The iFIE approach introduces the idea of net multiplexer sharing which approximatively reduces the overhead to 91%. However, this result is still not very reasonable since there is substantial room for improvement. For this reason, we propose a novel efficient iFIE structure to realize an ECC-based functional trust-checking architecture that considers more coarse-grained functionalities than simple CLB outputs. The intuitive rationale is to compose CLBs and avoid multiplexers on internal connections since the switching between normal circuit working and trust-checking phase is already done at the upstream inputs. This idea may be beneficial not only for the hardware but even for the performance overhead since less multiplexers are interposed on the circuit paths.

It might be argued that an exhaustive verification applied to a large function f can be as effective (in terms of trust-checking) as verifying the several fine-grained subfunctions from which f is composed. Suppose that there is a tamper in a subfunction f_i. This tamper may or may not be detected from an exhaustive trust-checking over f. While the first scenario is desirable, the second one seems to be problematic. However, if the truth table for the overall function f is unchanged and no functional modification is detected by the exhaustive trust-checking then we can simply ignore that tamper. In other words, the modification in the subfunction f_i produces some changes in the intermediate logic values but these are not propagated to the functional output f. For instance, suppose to have a logic function $f = (f_1 \cdot \bar{f}_1) + f_2 = f_2$. We can insert a temper in f_1 but the overall logic function remains $f = (\hat{f}_1 \cdot \bar{\hat{f}}_1) + f_2 = f_2$. A successful functional trust-checking at a certain granularity level is sufficient for assuring with very high probability that no modification affects the higher level functionalities. This underlying property permits to consider larger functionality in our iFIE approach without loosing the capability of providing a trusted FPGA circuit. On the other hand, a failed trust-checking phase is a necessary but not sufficient condition for having a functional modification in the overall circuit. For the sake of security, we consider any circuit which does not pass a trust-checking phase as not trustworthy since a tampering (successful or not) has occurred. In this scenario, a fine-grained approach can be useful for detecting a wider set of attempted attacks. This advantage is heavily counterbalanced by a large hardware overhead associated with the switching multiplexers. For this reason, the iFIE approach reasonably chooses to use more coarse-grained functionalities.

3.4.1 A more general CLB functional model

CLBs are the reconfigurable hardware units used to build combinatorial or sequential logic functions. In some FPGA architectures, a CLB can further be decomposed in more elementary units known as slices containing LUTs, registers, carry chains or large multiplexers. Figure 3.6 shows an example referring to the Xilinx Virtex 4 [12] architecture where a CLB is composed by four slices and two of them can even be used to implement a distributed RAM or a shift register.

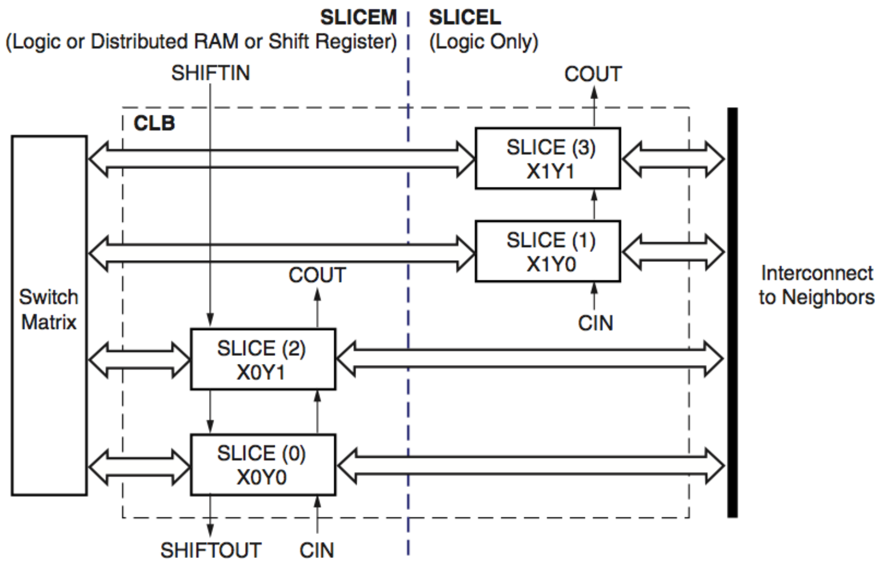
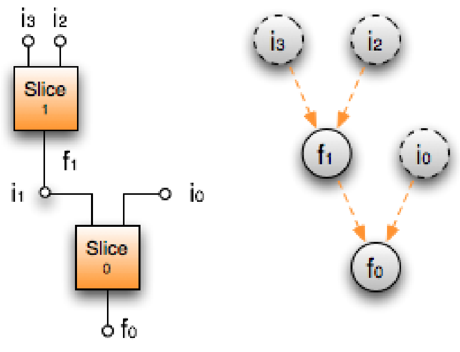


Figure 3.6: Arrangement of slices within a CLB

Intuitively, a slice can be modeled as a set of inputs $i_0, i_1, \dots, i_{\alpha-1}$ and a set of functional outputs $f_0, f_1, \dots, f_{\beta-1}$. Depending on the slice configuration, each output implements a certain function of a certain subset of inputs. Usually a dependency can be described in terms of k-LUTs so it is sufficiently general in order to model predetermined functions such as a carry chain or a multiplexer. Moreover, a functional output f can be sequential when the correspondent output flip-flop is configured as active.

This functional model generalizes the idea of slice in order to avoid dependency on any specific FPGA architecture. Moreover, it depicts a very intuitive way for composing large functionalities as needed by the iFIE approach. Given a functional output f, we can consider another functional output f_i which is connected to one of its inputs by the means of a net. Thus, we can compose a large functionality still corresponding to output f but having as inputs the union of both f and f_i input sets. An example of composition is presented by Figure 3.7.



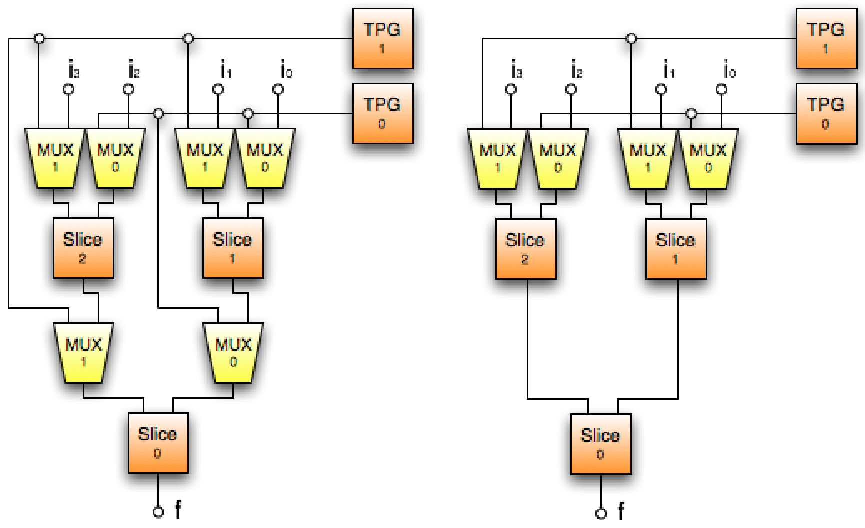
(a) Slice composition (b) Functional graph

Figure 3.7: Functional model of slices

We can describe every FPGA circuit in terms of basic functional elements or technology-mapped slices. In Figure 4.3a it is presented a simple circuit composed by two slices where functional output f_0 has i_0 and i_1 as inputs and functional output f_1 has i_2 and i_3 as inputs. According to the circuit connections, we can compose a larger functionality f_0 depicted by the functional graph in Figure 4.3b. As we can see, each node in the graph corresponds to a net (input or output) in the circuit whereas each edge represents a functional dependency between two of these nets by the means of a conveniently configured slice. Moreover, the dashed nodes represent the input nets for the overall functional output f. It might be argued that f_0 and f_1 may even belong to the same slice assuming slice with multiple functional outputs (for instance, multiple k-LUTs). However, the presented functional model is sufficiently general in order to deal with complex scenarios since each functional output f is independently considered along with its inputs and connections with other functional outputs in the circuit. At last, the randomized 2D ECC parity scheme can trivially be applied to this model since we basically focus on a set of functional outputs.

3.4.2 Cone structures

The concept of functionality trust-checking is substantially independent form the granularity at which we decide to implement it. Considering a large combinatorial circuit, we can directly apply the trust-checking to its *Primary Outputs* (PO) placing switching multiplexers on the circuit *Primary Inputs* (PI). Alternatively, we can consider slice-grained functional outputs. However, this scenario is very expensive in terms of hardware overhead since each interconnection virtually needs a switching multiplexer. This intuitive idea is further depicted by Figure 3.8.



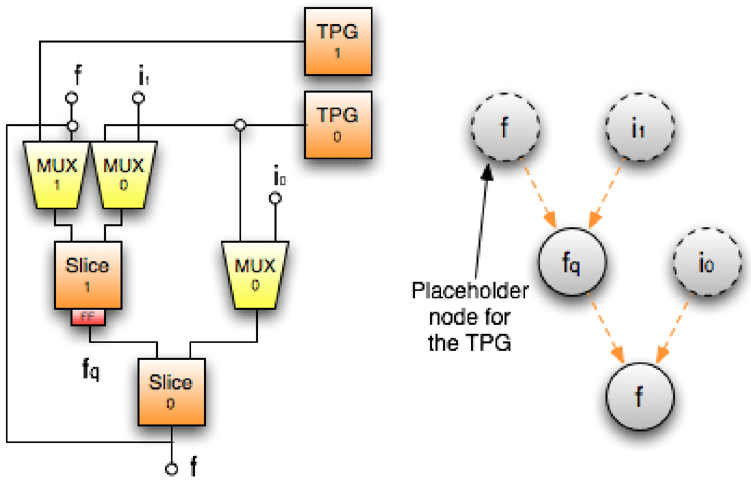
(a) Slice-grained (b) Grained-grained

Figure 3.8: Granularity versus Multiplexers

As we can see, we can adopt a slice-grained approach by considering three separate functional output (Figure 3.8a). However, a coarse-grained approach (Figure 3.8b) is preferable since it only uses four switching multiplexers instead of six.

According to these considerations, we introduce a novel iFIE architecture where coarse-grained functionalities are constructed on the raw idea of cone structures. A cone is basically a functional tree associated with a slice output f that describes its dependency from other intermediate functional output selected according to the circuit interconnections. Looking again to Figure 3.8b, we can perceive a cone structure where a set of inputs merges into in an unique output f passing through some intermediate functions. When an intermediate net has its entire fanout within the cone we don't need a multiplexer on it. In other words, the switching multiplexers are only placed on the cone input set according to the described dependencies. Cone structures are well-known for their application in many EDA problems ranging from FPGA technology mapping [30, 31] to circuit partitioning [32] and packaging [33]. In this thesis work we use the cone idea in order to minimize hardware and performance overheads related with the ECC-based iFIE trust-checking architecture.

A cone structure should be able to be exhaustively verified during a trust-checking phase in order to implement a randomized 2D ECC parity scheme. Suppose to have a combinatorial circuit. For each functional output f (also known as cone seed), we can construct a cone rooted in f. According to the functional model, we can expand the initial cone by including a functional output connected to one of its current inputs. This expansion guarantees that the associated functional graph is a *Direct Acyclic Graph* (DAG) where given the set of cone inputs we can exhaustively verify the functional output f. Consider now a sequential circuit in which one or more functional outputs have their flip-flops activated. In this scenario, the proposed expansion policy does not guarantee to obtain a DAG since we may have a closed loop over the sequential component. For this reason, we should compose a cone in order to enforce the DAG condition on its functional graph. It might be argued that a switching multiplexer provides a separation during the trust-checking. More in detail, the cone inputs are connected to the TPG lines so every circuit loop passing through a switching multiplexer is cut. In order to represent this property and maintain a DAG structure in the functional graph, we duplicate the node representing the functional output that causes the loop. This copy is then used to represent an input placeholder for the TPG unit during the trust-checking. Figure 3.9 shows an example of this technique.



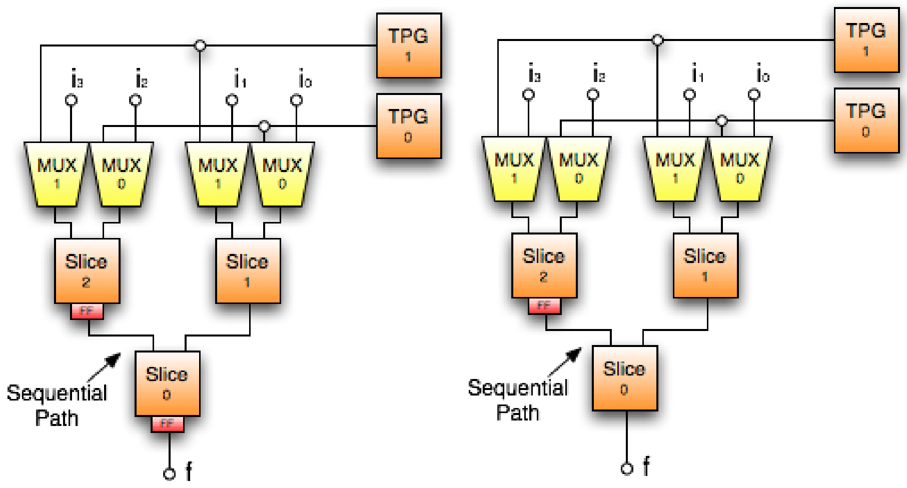
(a) Cone (b) Functional graph

Figure 3.9: Loop cutting during trust-checking

As we can see, we have a circuit with a loop involving f_q and f. However, we can still construct a cone since the path from functional output f to the correspondent cone input is cut during the trust-checking phase. An hypothetical functional graph should include an edge from node f to node f_q but the logical separation permits to duplicate node f which is then added as cone input. This cutting strategy must be applied in order to preserve the DAG structure of the functional graph associated with the cone. Every time we expand a cone using a new functional output, we check the loop condition. If the logical separation between an output fanout and the correspondent input multiplexer can maintain the DAG structure we accept the expansion otherwise we simply avoid that possible cone.

The presented DAG structure is necessary but not sufficient in order to verify a cone containing sequential elements. Our goal is to virtually emulate a combinatorial stimulus of the functional truth table. In other words, for each TV applied to the cone inputs we want to obtain the correspondent logic output which is subsequently embedded in a parity calculation. This goal requires a correct synchronization within the single clock cycle. A flip-flop can be considered as a buffer which stores an intermediate logic value f_{in} into the functional output f_q. Obviously, f_{in} should be stable at the flip-flop input before the clock edge in order to load the current f_q and then propagate to the cone functional output f. Supposing that the TPG unit is able to provide a TV during the opposite clock edge respect to the flip-flop triggering and that the propagation delays to f_{in} are sufficiently small, we can calculate the cone functional output f within a single clock cycle. It might be argued that this consideration is valid only if there is no path with more than one flip-flop within the cone. More generally, we can assume that a pseudo-combinatorial testing requires one clock cycle for the TV generation and several clock cycles for the buffering depending by the longest sequential path in terms of flip-flops within the cone.

The presented technique permits to verify the combinatorial functionality of the cone. However, it is not effective against flip-flop insertion (activation) or deletion (deactivation) within the cone. For this reason, we add an additional cone constraint for which each path should have at most one flip-flop. Moreover, we include the set and reset signals of each flip-flop in the set of the cone inputs that are exhaustively verified. Suppose that a flip-flop is inserted in a sequential path. It might be argued that this insertion add one delay cycle in the signal propagation disrupting the output value for some TVs which should be processed in two clock cycles (including TV generation). Similarly, if we delete a flip-flop basically we have an intermediate f_q which is not directly controlled by a set or reset signal. During trust-checking, we certainly incur in an input combination for which f_q has an arbitrary value instead of being set (or reset). Regarding the addition of a flip-flop on a combinatorial cone path, we need an additional control signal for which is possible to disable all the flip-flop in the FPGA device. A disabled f_q with a fixed arbitrary value certainly disrupts some output values in the exhaustive trust-checking which are subsequently detected by the randomized 2D ECC parity scheme.



(a) Forbidden sequential path (b) Accepted sequential path

Figure 3.10: Sequential cones

Figure 3.10a shows an example of incorrect cone where there exists a sequential path traversing two flip-flops (slice 2 and slice 0). Potentially, we can insert a flip-flop in slice 1 without detection during the trust-checking phase. On the other hand, Figure 3.10b shows a correct cone where an unique sequential path has a single flip-flop (slice 2). As we mentioned, this scenario permits to detect every type of flip-flop insertion or deletion within the cone. Summarizing, during cone expansion we must enforce the presented sequential path constraint along with the other DAG constraint in order to provide an exhaustively testable structure. In other words, if an expansion creates a forbidden sequential path as Figure 3.10a then we discard that possibility selecting other alternatives (for instance, we limit our cone to slice 0 and slice 1).

At last, the presented cone-based iFIE approach offers a partial protection against interconnection tampering. More in detail, the cone functionality also depends by its internal interconnections. For this reason, we are able of detecting any routing tamper within the cone during trust-checking, an added capability compared to a fine-grained approach as the basic FIE.

3.4.3 Cone-based trusted FPGA design

A coarse-grained approach introduces certain advantages in terms of hardware and performance overheads. In this novel scenario, the random PGs are directly composed by considering cone functional outputs. We have discussed about the rationale behind the cone idea and the constraints necessary for assuring cone testability. Here, we propose a detailed analysis of the cone circuit design.

An important design issue is related with finding an optimal cone size. Intuitively, a large cone has the advantage of having a large number of interconnections which do not need switching multiplexers. Just to clarify, suppose to have a combinatorial circuit with single fanout nets and a single output f. In this simplified case, there exists a cone that covers the entire circuit in order that all the internal connections do not need switching multiplexers. We only require a mandatory set of multiplexers on the primary inputs in order to connect the TPG unit. It is straightforward to prove that this scenario is optimal in terms of hardware overheads. Suppose that the combinatorial circuit has p primary inputs and consider a solution composed by c cones. In addition to the p switching multiplexer on the primary inputs, this solution has $c-1$ intermediate multiplexer placed at the cone outputs in order to implement the separation between cones during trust-checking. It might be argued that

$$p \leq p+c-1$$

so the solution with an unique cone is optimal in terms of multiplexer overhead. Another advantage of large cones is related with ORA overhead. Intuitively, coarse-grained functionalities decrease the number of functional outputs to verify by the means of the trust-checking components. Supposing that the initial circuit has n functional outputs and that each cone covers an average of l outputs, we should apply the ECC-based technique to n/l functional output reducing the 2D scheme size to $\sqrt{n/l} \times \sqrt{n/l}$. According to the previous considerations about trust-checking without parity functions, the total ORA overhead is calculated as

$$ORAs_{ovhd} = 2\sqrt{n/l}\left\lceil \frac{\sqrt{n/l}-1}{k-1} \right\rceil (LUTs)$$

In others words, the trust-checking hardware overhead scales down by a factor l representing the cone size.

On the other hand, large cones also have drawbacks. First of all, decreasing the number of functional outputs in the 2D ECC parity scheme increases the masking probability p_{mask}. We have seen that p_{mask} has a magnitude in the order of $O(1/n^2)$ where n is the number of functional outputs. Referring to cones with average size l, we have a quadratic scale-up factor that increases the masking probability to $O(l^2/n^2)$. Intuitively, if $l \ll n$ then p_{mask} is sufficiently small. Another important drawback is concerned with the TPG unit. Intuitively, large cones have a lot of inputs and consequently a large TPG unit is required in order to exhaustively verify all the input combinations. More in detail, the largest cone in the circuit determines a lower bound on the number of necessary TPG lines (we have seen that the actual TPG size is decided after graph coloring). An unique TPG unit can be shared between all the PGs so its hardware overhead has a little overall impact. On the other hand, the number of TPG lines affects the routing and the trust-checking duration. Regarding the routing, an higher number of TPG lines may saturate the programmable interconnection network used to implement the trusted FPGA circuit. Regarding the trust-checking duration, generating all the possible combinations requires an exponential $O(2^p)$ amount of time where p is the TPG size in bits. Supposing that an FPGA device has a clock of 200MHz, we can estimate the trust-checking phase duration for different p sizes as reported here in Table 3.2.

TPG size	Trust-Checking Duration
16	655 μs
20	10.5 ms
24	167 ms
28	2.68 s
32	42.9 s

Table 3.2: Trust-checking phase duration

Depending by the real-time application constraints, an FPGA circuit can suspend its normal working for a limited amount of time. It seems reasonable to consider an upper bound comparable with partial dynamic reconfiguration times in order to maintain the speed advantage of the iFIE approach over the NIE approach. An investigation on runtime partial reconfiguration in [34] reports an average times on the millisecond scale. For this reason, we should not consider a TPG unit larger than 24 bits. At last, the ECC-based trust-checking mechanism requires to store an expected PV for each PGs in order to perform an off-chip comparison with the PV produced by the trusted FPGA circuit. Considering 24 bits as maximum TPG size, the expected PVs only require 64 KBytes for each of the $2\sqrt{n/l}$ PGs.

As we have seen, there are different design issues related with the cone size in terms of inputs and in terms of covered interconnections. Moreover, we have two constraints related with the DAG structure and the sequential paths which practically limit the cone size. In this complex scenario, it might be argued that it is not possible to provide an analytically optimal trade-off between all the aforementioned design parameters. A reasonable approach consists of introducing a maximum cone size in terms of inputs (or TPG size) and constructing the biggest possible cone in terms of covered interconnections, always respecting the remaining constraints. Heuristic approaches for cone generation and selection will be presented in chapter 4.

The cone-based iFIE approach introduces some modifications into the *trusted FPGA design* flow. Differently from what was proposed in [3], the conventional EDA flow is not completed with placing and routing but rather we stop with a technology mapped FPGA circuit. In fact, these final design phases are postponed after the embedding of the trust-checking components in order to provide a monolithic trusted FPGA circuit. Moreover, the first step of the *trusted FPGA design* phase consists of generating and selecting the cones in order to minimize hardware and performance overheads. The identified functional outputs are then used to compose random PGs. As mentioned, the iFIE approach does not provide a structural implementation for the parity functions since the parity comparison is done off-chip. For this reason, the parity function truth tables are stored as expected PVs. All the mentioned modifications are summarized in the *trusted FPGA design* flow presented by Figure 3.11. As we can notice, the deployment phase remains almost unchanged. More precisely, we still perform a “blind” comparison between a PV produced by the trusted FPGA circuit and an expected PV stored off-chip. However, this comparison is no more focused on random polarities but rather on pseudo-random parity sequences determined by the cone functionalities and by the PGs compositions.

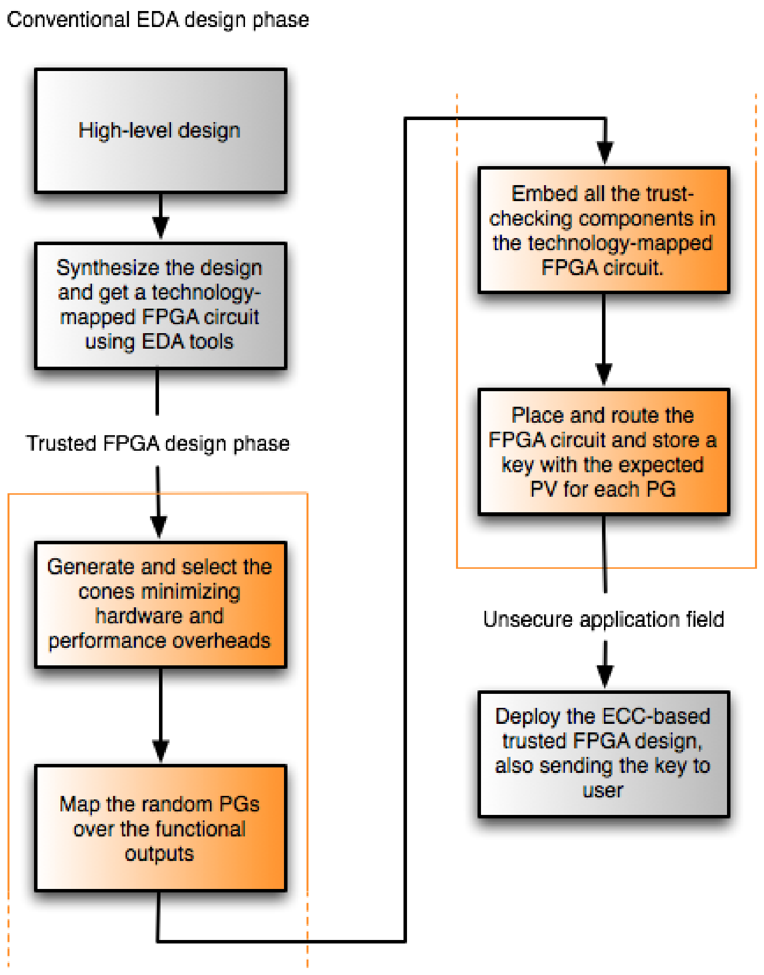

Conventional EDA design phase
High-level design
Synthesize the design and get a technology-mapped FPGA circuit using EDA tools
Trusted FPGA design phase
Generate and select the cones minimizing hardware and performance overheads
Map the random PGs over the functional outputs
Embed all the trust-checking components in the technology-mapped FPGA circuit.
Place and route the FPGA circuit and store a key with the expected PV for each PG
Unsecure application field
Deploy the ECC-based trusted FPGA design, also sending the key to user

Figure 3.11: iFIE trusted FPGA design flow

3.5 Reconfigurable Error Correcting Code

The presented iFIE approach provides a more efficient implementation of the ECC-based trust-checking mechanism. However, it still inherits some intrinsic drawbacks of the basic FIE approach. First of all, the trusted FPGA circuit statically embeds both the cones composition and the random PGs mapping. Despite bitstream encryption, a malicious adversary with a sufficient level of resources can destructively inspect the FPGA device and stole the encryption key. Thus, he can still reverse-engineering the circuit routing in order to discover the genuine expected PVs. Another underlying drawback of statical embedding is concerned with the necessity of a secure channel between the FPGA circuit and the user in order to complete the trust-checking. It might be argued that this secure channel can still be bypassed through reverse-engineering in order to disclose the expected PVs. These considerations highlight a vulnerability to reply attacks for the ECC-based trust-checking mechanism . Suppose that the adversary knows the expected PVs. A malicious FPGA circuit can circumvent the ECC-based trust-checking mechanism by configuring part of the FPGA device as combinatorial circuits capable of producing genuine PVs. In other words, we maliciously implement the correct parity functions in order to produce the expected output for a trust-checking phase. A visual example of replay attack is given by Figure 3.12.

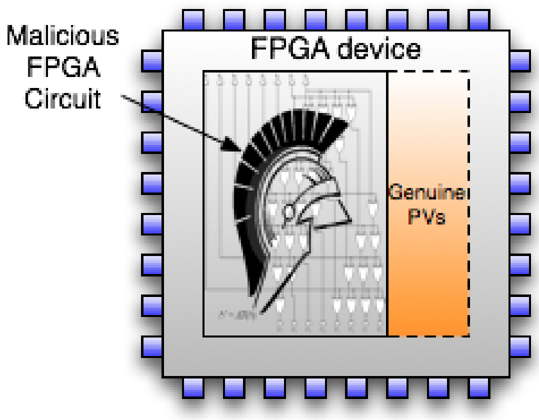


Figure 3.12: Replay attack

As we can see, part of the device is occupied by an FPGA circuit compromised by a Trojan whereas the other is used for implementing the parity functions (alternatively, we can implement an unit which reads the expected PVs from an external memory). It might be argued that any circuit capable of producing genuine PVs is consider "trustworthy". For this reason, the ECC-based trust-checking mechanism fails against reply attacks. Using random polarities cannot resolve the problem since we can still embed PVs corresponding to random polarity vectors. We may assume that encryption alone is sufficient for both bitstream protection and channel transmission. However, an adversary can still eavesdrop the encrypted channel. It might be argued that an encrypted message can be captured and reused later for further deceptive trust-checking, at least with an encryption protocol that does not use random keys. In other words, a replay attack can directly embed the encrypted message in the FPGA circuit since this is always the same. As mentioned, a more complex encryption protocol can partially solve the replay attack issue. However, it is still virtually possible to eavesdrop the expected PVs on the communication line between the trusted FPGA circuit and the encryption unit, especially in a PCB system where different chips are used. At last, it might be argued that the added hardware overhead is not an issue for replay attacks. In fact, we can eliminate all the ORA units and substitute the released FPGA area with the needed parity functions since these latter have a comparable hardware overhead estimable as

$$Replay_{ovhd} = 2\sqrt{n/l}\left\lceil \frac{p-1}{k-1} \right\rceil (LUTs)$$

where n is the number of functional outputs, l is the average cone size and p is the TPG size.

According to the previous analysis, replay attacks are successful because the routing of cones and random PGs is intrinsically embedded in the trusted FPGA circuit. We propose a novel challenge-response trust-checking protocol known as *Reconfigurable Error Correcting Code* (RecECC) which immunizes the system from replay attack and also offers astronomically small masking probability p_{mask}. The underlying idea is concerned with not embedding any fixed PG routing but rather using a different random PGs composition at each trust-checking phase.

This idea has the clear advantage of being robust against the reverse-engineering of routing. In fact, an adversary can only retrieve the cones composition which is not sufficient to disclose the PVs. On the other hand, a variable PGs composition make useless the embedding of a fixed set of parity functions. In fact, a reply attack always produces the same PVs whereas the RecECC technique continuously changes PVs at each trust-checking phase. Moreover, since the number of possible PGs mappings is astronomically large there is no way of embedding all the possible PVs. For these reasons, we can assume that the RecECC technique immunizes the trusted FPGA circuit from reply attacks.

The RecECC technique can be integrated in the iFIE trust-checking by the means of a structural modification of the ORA unit. Considering a logic value involved in a parity calculation, we can exclude it by using a masking AND gate. More in detail, given t functional outputs $f_0, f_1, \cdots, f_{t-1}$ we mask a functional output f_i from a parity calculation by connecting f_i to an AND gate. It is easy to prove that

$$XOR_{j \neq i}\left[f_j\right] + f_i \cdot 1 = XOR_{j=0}^{t-1}\left[f_j\right]$$

and that

$$XOR_{j \neq i}\left[f_j\right] + f_i \cdot 0 = XOR_{j \neq i}\left[f_j\right]$$

In other words, when the masking is disabled (logic value one) the parity calculation includes all the functional outputs whereas when the masking is enabled (logic value zero) f_i is excluded. Extending this reasoning to all the n functional outputs, the RecECC technique introduces a masking array composed by n AND gates, one for each functional output. Moreover, a n-bit shift register is used to store the masking configurations for all the n AND gates. At last, a large ORA unit with n inputs calculates the parity functions over all the masking gate outputs. Loading a bit sequence in the shift register can virtually create an arbitrary PG composed by the enabled functional outputs (logic value one). Thus, we can obtain the correspondent PV by exhaustively testing all the input combinations through the large ORA unit. During this trust-checking phase, the iFIE architecture stimulates all the available functional outputs thanks to the provided hardware parallelism. However, only the not masked outputs (in other words, the PG) will contribute in the PV calculation. The proposed RecECC architecture is shown by Figure 3.13.

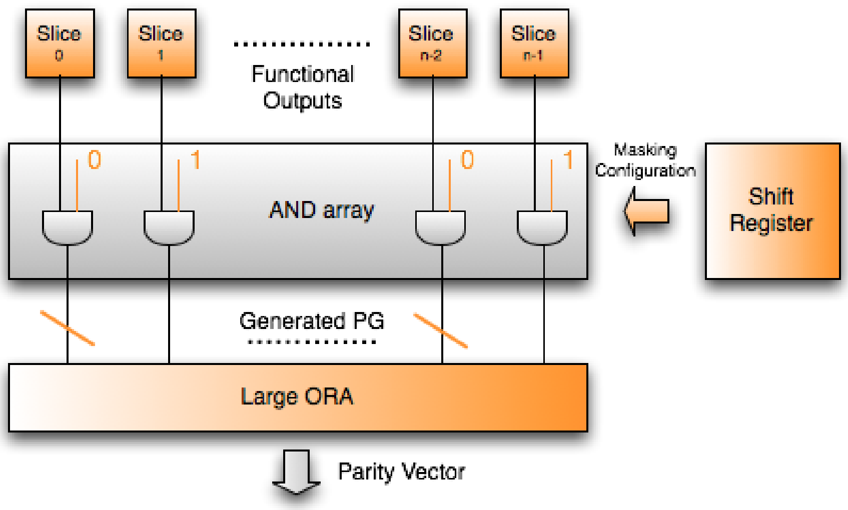


Figure 3.13: RecECC architecture

As we can see, the shift register bits correspond to a masking configuration for the AND array in order to decide the PG composition in terms of functional outputs. The RecECC technique permits to configure an arbitrary PG and to calculate its parity. This feature is then exploited in order to apply a 2D ECC parity scheme with variable mapping. We introduce a request-response trust-checking protocol for which the user loads a bit sequence for a specific PG in the shift register and then obtains the correspondent PV. Supposing to have an unique large ORA unit, this procedure is serially repeated until all the PGs are verified. It might be argued that each trust-checking phase can have a different PG mapping depending by the loaded bit sequences. For this reason, the RecECC technique is immune to replay attacks. The off-chip PVs comparison is slightly more complex than the case of fixed PGs mapping where we simply store the expected PVs. In fact, this approach is not feasible in case of reconfigurable PGs since we have an astronomically large number of possible mappings (and consequently PVs). For this reason, we should calculate the PV at run-time by the means of a simulation. More in detail, we have a description for all the functional outputs (cones) so we can dynamically compose a PG and calculate its expected PV in order to perform an off-chip comparison with the response PV provided by the trusted FPGA device. The proposed RecECC request-response protocol is shown by Figure 3.14.

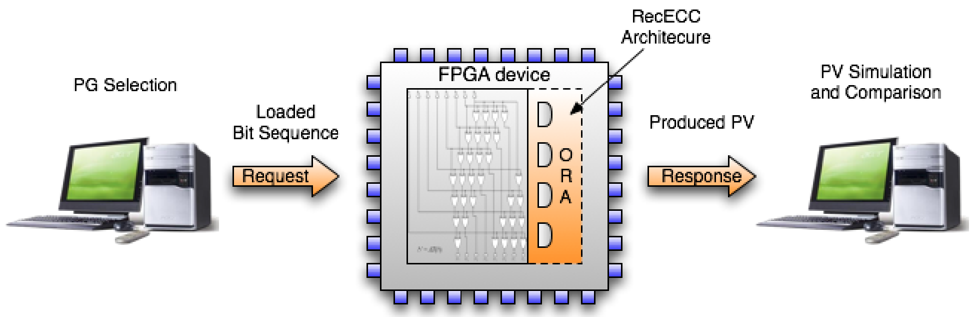


Figure 3.14: RecECC protocol

Alternatively, we can still store a considerable number of precomputed PGs mappings (for instance 100). During the trust-checking, we limit the possible requests to these mapping and we simply perform a comparison with the produced PVs. This approach can avoid reply attacks provided that the amount of different PGs mapping is sufficiently large in order that no FPGA device can implement all the different possibilities.

The hardware overhead associated with the RecECC technique is surprisingly comparable with a fixed mapping scenario. Considering cone-based functionalities, we have a total of n/l functional outputs where n is the number of slice-grained functional outputs of the original FPGA circuit and l is the average size of a cone. Moreover, we have n/l additional bits related a the masking shift register. We can consider the AND array and the ORA unit as an unique large combinatorial function with $2n/l$ inputs. According to k-LUT implementation, we can calculate the hardware overhead of the RecECC technique as

$$RecECC_{ovhd} = \left\lceil \frac{2(n/l)-1}{k-1} \right\rceil (LUTs)$$

which is totally comparable with the overall ORA overhead calculated in section

3.4 for the cone-based iFIE approach

$$ORAs_{ovhd} = 2\sqrt{n/l}\left\lceil \frac{\sqrt{n/l}-1}{k-1} \right\rceil (LUTs)$$

As we can see, both are proportional to $2n/l$. Intuitively, the RecECC technique should have more hardware overhead due to its architectural complexity. However, we should consider that the RecECC trust-checking phase uses a serial approach for which the PGs are verified one at a time (this slows down the trust-checking by a factor $2n/l$). In other words, these considerations compensate with one another so we approximately have the same hardware overhead initially associated with the trust-checking units. On the other hand, the RecECC architecture requires an additional (n/l)-bit shift register. However, the RecECC technique has the added advantage of not requiring any encryption neither for request nor for response. In other words, the released FPGA area can be used to implement the shift register. From another perspective, we can consider the same RecECC technique as an hashing code for which an input sequence (the PG composition) of n/l bits is processed in order to produce another output sequence (the produced PV) of 2^p bits. It might be argued that the RecECC architecture is very flexible. In fact, we can modify the ORA unit in order to implement an arbitrary hashing code over the entire set of functional outputs. Moreover, the shift register may still be used as random padding in order to avoid reply attacks.

In addition to protection against reply attacks, the RecECC technique offers an astronomically small masking probability p_{mask}. Considering a 2D ECC parity scheme of size $\sqrt{n/l} \times \sqrt{n/l}$, we have

$$\frac{(n/l)!}{(\sqrt{n/l})! \cdot (\sqrt{n/l})!} = \frac{(n/l)!}{\left[(\sqrt{n/l})!\right]^2}$$

possible combinations where $(n/l)!$ represents all the possible random mappings whereas $(\sqrt{n/l})!$ represents the row and the column permutations between random mappings that generates the same PGs. Thus, the masking probability p_{mask} achieved in [3] is scaled down by a large factor corresponding to these possible combinations. We can further expand the capability of the RecECC technique by composing PGs with arbitrary 2D schemes. In other words, we have $\sqrt{n/l}$ different possible scheme sizes for which the total number of possible combinations approximately increases to

$$\sum_{i=0}^{\sqrt{n/l}} \frac{(n/l)!}{i! \cdot \left[(n/l)-i\right]!}$$

We have mentioned that the cone-based iFIE approach increases the masking probability by a factor l^2 since it reduces the number of available functional outputs to n/l. However, the RecECC technique is capable of overcompensating this increasing by scaling down p_{mask} by a large factor determined by the aforementioned possible combinations.

Chapter 4

Heuristics for cone generation and selection

The iFIE approach improves the original trust-checking mechanism originally proposed in [3]. The underlying optimization focuses in particular on the heavy hardware and performance overheads associated with multiplexers and is based on a novel trust-checking architecture that considers more coarse-grained functionalities known as cones. In the previous chapter we presented the general idea of cones and the associated constraints necessary for applying the randomized 2D ECC parity scheme. This chapter further elaborates on cone structures, discussing some heuristic approaches for generating and selecting cones with minimized overheads.

The chapter is organized in the following way. Section 4.1 formalizes the problem of generating and selecting cones as a covering problem concerned with the functional outputs of the FPGA circuit. Section 4.2 proposes a benefit metric based on covering and cutting, two concepts related with the amount of fanouts that lie within the cone and the cost of placing a multiplexers in front of cone inputs. This metric is then used for generating "good" cones with the perspective of minimizing the hardware overhead. Section 4.3 proposes a cone covering algorithm for overhead minimization which selects between cones generated according to the aforementioned benefit metric. At last, section 4.4 concludes the chapter by introducing an alternative approach to generate and select cones which takes account of both hardware and performance overheads minimization.

4.1 A covering problem

The goal of a trust-checking phase is to verify the functionality of the entire FPGA circuit in order to detect tampering or Trojan injections. This obviously involves a functional verification for all the slice-grained functional outputs. Omitting some of these outputs basically corresponds to an incorrect application of the ECC-based mechanism. For this reason, we always require a capillary functional verification even when we deal with more coarse grained functionalities as cone structures. Intuitively, we can introduce a raw concept of covering referring to a functional verification for a cone c which is also sufficient for checking a slice-grained functional output f. In other words, we say that c covers f if we can check function f by testing cone c. Moreover, a correct application of the ECC-based mechanism requires that all the slice-grained functional outputs are covered, a condition also known a complete covering.

The proposed scenario perfectly fits with the well-known computer science covering problem. Referring to its classic formulation, we have a set A of n elements $\{a_0, a_1, \dots, a_{n-1}\}$ and a collection of k subsets $B_0, B_1, \dots, B_{k-1} \subseteq A$ with associated cost $c(B_i)$. The covering problem solution S consists of a composition of the available subsets in order to provide a complete covering of the set $A = \bigcup_{B_i \in S} B_i$ and in order to minimize the total cost $\sum_{B_i \in S} c(B_i)$ associated with the solution. Transposing the covering problem to an FPGA circuit, we have a set F composed by all the slice-grained functional outputs and a collection of cones each of which covers a subset of F. In this scenario, the covering solution is a set of cones C_{sol} that covers all the set F of slice-grained functionalities with a minimized hardware overhead. It might be argued that this problem is intrinsically difficult since the original covering problem is in the class of NP-complete problems [35, 36]. Moreover, we don't have an initial collection of cones as required by the covering problem formulation. For this reason, we requires an additional cone generation phase which creates a collection of candidate cones G_{gen} from the initial set F of slice-grained functional outputs. This further expands the problem search space. For the sake of simplicity, consider a purely combinatorial circuit with single fanout output nets and implemented with 2-LUTs. Moreover, suppose to impose a constraint p related with the maximum cone input size. Each cone is approximately a binary tree with p elements. Given a functional output $f \in F$, we can root $\binom{2p}{p}/(p+1)$ different cones (combinations of possible binary trees). Taking into account of all the n functional outputs, we can generate a collection C_{gen} of cardinality

$$|C_{gen}| = \frac{\binom{2p}{p} \cdot n}{p+1}$$

Considering a brute force approach for the covering problem, we have an exponential time complexity $O(2^{|C_{gen}|})$ where the exponential term $\binom{2p}{p}$ contributes to an huge number of possible combinations. However, the presented analysis only represents an upper bound. Considering a generic FPGA circuit, we have additional generation constraints related with the DAG structure of the cone and with its internal sequential paths. For this reason, we end up with a reduced search space.

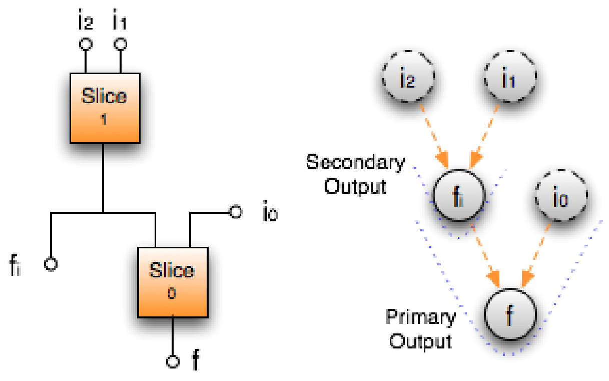
We have introduced an intuitive definition of covering. However, it is necessary to clarify the conditions for which we can verify a function f by a trust-checking on a cone c. Considering the functional graph associated with the cone (see section 3.4), any slice-grained output f corresponds to a circuit net n necessary for propagating the logic output value to another slice or to a primary output. For the sake of clarity, there is a biunivocal mapping between functional outputs and nets so we can interchangeably use the two terms. Given a cone c, its functional graph identifies a set of nets with different roles. First of all, we have an unique root/seed net on which the cone is constructed according to the metric that will be presented in section 4.2. A root net represents the primary output of cone c. During trust-checking, the cone functionality is verified by propagating an exhaustive TVs sequence from the cone inputs to this primary output. For this reason, the slice-grained function f correspondent to the root net is intrinsically covered by cone c .

The functional graph also contains a set of input nets on which the cone functionality depends. The switching multiplexers in the trusted FPGA circuit are placed on the input nets in order to exhaustively verify the cone functionality. Despite input nets are included in the functional graph, the associated slice-grained function are not covered by the cone c. More in detail, an input net may correspond to a circuit primary input or to another functional outputs. In the former case there is no meaning for functional verification whereas in the latter case the functional output is obviously covered by another cone.

At last, we can identify a third type of nets associated with intermediate nodes of the functional graph. During trust-checking, these internal nets provides a propagation path towards the primary output by at least one of their fanouts. Moreover, the associated slice-grained functional output f_i contributes to the overall cone functionality. Given an internal net, it might be argued that if all its fanouts lie within the cone c then the associated function f_i is covered by cone c. In other words, f_i expresses its functionality only through the cone primary output and no other part of the circuit is influenced. For this reason, a functional trust-checking on the primary output is sufficient in order to detect effective tampering.

The scenario is different when an internal net has some fanouts outside the cone. In that case, f_i directly expresses its functionality by the means of connections to primary outputs or to other slices. For this reason, a functional modification in f_i may lead to a functional modification for the overall FPGA circuit. Thus, it is necessary to consider an internal net with exposed fanouts as a secondary output for the cone c. For a logical point of view, this output identifies a subcone c_i with primary input f_i and depending by a certain subset of cone inputs. Moreover, cone c_i can be stimulated by applying a TVs sequence on cone c since they share the same input nets (and consequently the same TPG line). For this reason, the functional output f_i can easily be included in the 2D ECC parity scheme without contraindications. The logical separation between primary and secondary cone outputs is shown in Figure 4.1. We can identify a secondary cone output f_i related with an internal net with two fanouts (only one lies in the cone). This secondary output also corresponds to a subcone with input subset $\{i_1, i_2\}$.

During trust checking, an exhaustive TVs sequence is applied to the cone inputs thus all the possible combination of $\{i_1, i_2\}$ are stimulated (more than once, since they represents a bit subset of the original TVs).



(a) Slice composition (b) Functional graph

Figure 4.1: Primary and secondary cone outputs

The presented relationship between cone and subcone represents the first example of cone overlapping. Intuitively, two cones are overlapping if they cover the same slice-grained functional output f. However, the overlapping should respect a constraint in order to be compatible with the trust-checking phase. More in detail, an input net for a cone c_1 cannot be an internal net for another cone c_2. The motivation for this constraint is related with the placement of a switching multiplexer on the input net. During trust-checking, this multiplexer basically disconnects the net from its original functional output and then connects it to a TPG line. In other words, we disrupt the cone c_2 functionality by cutting some internal paths towards the cone primary output. We can avoid this problem by modifying the net sharing technique introduced in section 3.2. More in detail, we can partition the fanouts between the ones related with internal nets and the ones related with inputs nets. The multiplexer is then placed in order to be bypassed by the fanouts in the first group. In other words, the internal fanouts have a permanent connection which is not switched during the trust-checking phase. This technique for solving the multiplexer conflict is shown by Figure 4.2.

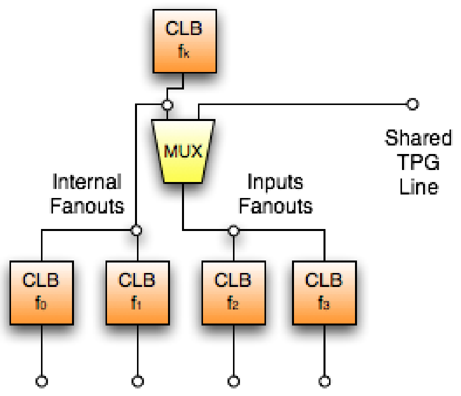
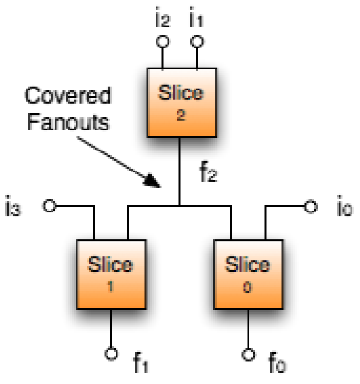


Figure 4.2: Partitioned fanouts for solving the multiplexer conflict

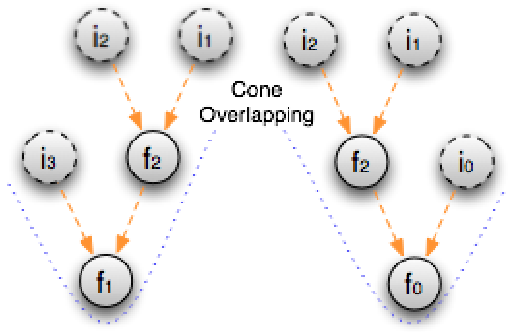
As we can see, the internal fanouts connected to f_0 and f_1 bypass the switching multiplexer so they preserves the structure of the cones in which f_k is an internal net. On the other hand, the input fanouts connected to f_2 and f_3 can still be driven by a TPG line during the trust-checking.

The presented technique based on a fanout partitioning solves the overlapping problem. However, we adopt a simpler solution based on a trivial consideration that will be still useful during covering. More in detail, when a net is internal for some cones and input for some others, the correspondent multiplexer must be placed anyway. The fanout partitioning technique is not able to reduce the hardware overhead. On the other hand, we can split a cone c rooted in f by considering the intermediate net f_i where a multiplexer is already placed and obtaining two cones. The first one is rooted in f_i and composed by all the functional paths from the inputs of cone c to the net f_i. The other one is still rooted in f and composed by all the paths which do not traverse f_i plus the subpaths from f_i (which becomes an input net) to f. It might be argued that the total hardware overhead of the two cones is equivalent to the original scenario. In fact, the number of multiplexers remains the same since the new input net f_i has a multiplexer already placed on it. Moreover, the number of functional outputs does not change since the primary output of the cone rooted on f_i basically replaces a secondary output of the original cone c. Summarizing, when we have two overlapping cones that do not respect the aforementioned constraint we simply apply the splitting procedure obtaining three valid cones with an equivalent total hardware overhead.

Considering the aforementioned constraint, the only possible overlapping between cones requires the complete sharing of a common subcone c_i in order that no input net overlaps with an internal net. Under certain conditions, we can remove the secondary outputs associated with subcone c_i and the possible multiplexer overhead. Given a internal net f_i, if all its fanouts lie under a set of shared cones then there is no need for a switching multiplexer since during trust-checking we only have internal propagation paths that traverse f_i. Moreover, the functional output f_i is not exposed so we can verify its functionality by applying trust-checking on the cones in which it is included. On the other hand, we can have a fanout that includes some connections to circuit primary outputs. In that case, we can still avoid the use of a multiplexer if all the fanout concerned with slice connections lie under a set of shared cones. However, we require to include f_i in the 2D ECC parity scheme since it has an exposed functionality. An example of sharing with fanout covering is presented by Figure 4.3.



(a) Slice composition



(b) Functional graph

Figure 4.3: Cone sharing with fanout covering

As we can see, we have two cones rooted in f_0 and f_1. They share a common subcone rooted in f_2. Regarding the functional output f_2, it has two fanouts and both of them are included in a cone. Since there are not additional fanouts connected to some slices, we do not require a multiplexer. The functionality of net f_2 is exposed considering a cone by itself but it is totally covered considering the overlapping. For this reason, we can avoid to include f_2 in the 2D ECC parity scheme. It might be argued that sharing is a good minimization approach in order to cover those interconnections which are not possible to cover within a single cone.

4.2 Metrics for cone generation

In the previous section, we have got an intuitive sense about the approaches for constructing cones tailored for the final hardware minimization goal. A first idea concerns covering all the fanouts of an internal net using a single cone or even a composition of overlapping cones. The rationale behind this idea is to avoid the use of a multiplexer on fanouts completely covered. A second idea is instead related to the reuse of multiplexers already placed by the means of input sharing among cones. It might be argued that these two ideas are correlated since cones with shared inputs are easily inclined to cover the same nets and vice versa.

We need to devise a metric capable of capturing the two key ideas mentioned above. We propose a formalization of those concepts broken in two main parts, the covering metric and the cutting cost. Given a net n_i and cone c, we define the covering ratio as

$$cov\_ratio(n_i, c) = \frac{|cov(n_i, c)|}{fanout(n_i)}$$

where $cov(n_i, c)$ is the subset of fanouts of net n_i that are internal interconnections for cone c (or covered by c) and $fanout(n_i)$ is simply the fanout of net n_i. In other words, $cov\_ratio(n_i, c)$ represents a quantitative measure of the degree of fanout covering of net n_i. It is easy see that $cov\_ratio(n_i, c) = 1$ implies that there is no need for a multiplexer on net n_i since there is no exposed functionality and the relative trust-checking is done through the cone. We can extend this evaluation to the entire set of internal nets of cone c introducing the covering metric

$$cov(c) = \frac{\sum_{n_i \in Int(c)} cov\_ratio(n_i, c)}{|Int(c)|}$$

where $Int(c)$ is the set of all the internal nets of cone c. This formula calculates a average covering within cone c, taking account of the various internal nets in an equal manner.

The presented metric just considers a single cone so it does not incorporate the concept of shared fanout covering. We need a further improvement that accounts for overlapping between selected cones. More specifically, the covering ratio depends not only on the cone c but also by the cones in a (partial) solution C_{sol} because all of them contribute to the covering of the n_i's fanouts. For this reason, we generalize the covering ratio as

$$cov\_ratio(n_i, c, C_{sol}) = \frac{|cov(n_i, c) \cup cov(n_i, C_{sol})|}{fanout(n_i)}$$

where $cov(n_i, C_{sol})$ is the subset of fanouts of net n_i covered by cones in solution C_{sol}. It might be argued that this generalized covering ratio will be integrated in a covering algorithm in order to consider a set of cones already selected to be in the solution C_{sol}. Thus, we also propose a new formulation for the generalized covering metric

$$cov(c, C_{sol}) = \frac{\sum_{n_i \in Int(c)} cov\_ratio(n_i, c, C_{sol})}{|Int(c)|}$$

The second concept that we need to formalize is concerned with the cutting cost of a cone. We use the term "cut" to refer to the cone inputs. Intuitively, when we select a cone to be in the solution C_{sol} we basically introduce a set of multiplexers in front of the cone inputs depicting a separation (or a cut) which is then used during trust-checking. Given an input net n_i of cone c, we can define its cut cost as

$$cut\_cost(n_i) = \begin{cases} \frac{1}{fanout(n_i)} & n_i \text{ is partially covered} \\ 0 & n_i \text{ is a primary input} \end{cases}$$

Essentially, this metric is related to the potential avoidability of placing a multiplexer on net n_i. Intuitively, a net with small fanout will be covered with high probability; so its cut cost should be high since cutting it and placing a multiplexer on it obviates the high likelihood event of the net being completely covered, and thus not needing a multiplexer. Moreover, the iFIE architecture always imposes multiplexers on the circuit's primary inputs and so there is no cost associated with these cuts. Similarly to what done for the covering ratio, we can extend this measure over the entire set of input nets of cone c. For this reason, we define the cutting cost of a cone as

$$cut\_cost(c) = \frac{\sum_{n_i \in Inp(c)} cut\_cost(n_i)}{|Inp(c)|}$$

where $Inp(c)$ is the set of inputs nets of cone c. In this case too, we need a further improvement that accounts for overlapping cones. More specifically, the cones in solution C_{sol} introduce some cuts and some covered fanouts that change the cut cost as follows

$$cut\_cost(n_i, C_{sol}) = \begin{cases} 1 & n_i \text{ is completely covered} \\ \frac{|cov(n_i, C_{sol})|+1}{fanout(n_i)} & n_i \text{ is partially covered} \\ 0 & n_i \text{ is already cut} \\ 0 & n_i \text{ is a primary inputs} \end{cases}$$

Intuitively, the cost of a cut depends also on how many fanouts are covered by C_{sol}. Nets with few not-covered fanouts have an high likelihood to be completely covered and thus an high cost for their cutting. The +1 factor in the covering ratio is used in order to distinguish between completely uncovered net with different fanouts advantaging small nets. For instance, a net with two fanouts has a cut cost of $\frac{1}{2}$ whereas a net with ten fanouts has a cut cost of $\frac{1}{10}$ since has less probability of being completely covered. On the other hand, there is no cost if we cut a net that already has a multiplexer such as a circuit primary input (where multiplexers are mandatory) or an input net already in the solution C_{sol}. The generalized cut cost $cut\_cost(c, C_{sol})$ is obtained by substituting $cut\_cost(n_i)$ with $cut\_cost(n_i, C_{sol})$ in the mean calculation

$$cut\_cost(c) = \frac{\sum_{n_i \in Inp(c)} cut\_cost(n_i, C_{sol})}{|Inp(c)|}$$

Finally, we can combine the two key concepts in an unique benefit metric. It might be argued that both covering metric and cut cost have value in the range

$[0,1]$. Moreover, a "good" cone should maximize the covering metric and minimize the cut cost. For this reason, we define the benefit metric for a cone c as

$$benefit(c, C_{sol}) = \frac{cov(c, C_{sol})}{cut\_cost(c, C_{sol})}$$

This formulation implies a maximization goal. Moreover, it encourages sharing of internal nets across multiple cones (in order to completely cover them) and inputs nets (in order to share multiplexers). The benefit metric has value in the range $[0,\infty]$. In fact, it can virtually assume value infinity if the correspondent cut cost is zero. We define ∞-cones as cones with no *cut cost*. It might be argued that selecting ∞-cones does not introduce any additional hardware overhead since all the needed input multiplexers are already placed in the trusted FPGA circuit. In section 4.3 we will see how the covering algorithm will take advantage of these ∞-cones by directly including them in the solution C_{sol}.

After having devised the benefit metric, we propose a cone generation algorithm responsible for producing a set of cones then available for covering. These cones are constructed according to a maximal benefit metric in order to increase the possibilities for hardware overhead minimization. Each generation is done considering a single slice-grained functional output (seed net) and continuing with iterative expansion steps based on a local benefit maximization approach. Each iteration expands an intermediate cone $temp\_cone$ by including a functional output f_i corresponding to one of its input net. This expansion should always satisfy the following constraints.

[a] TPG size : We consider the number of available TPG lines p as an a-priori parameter determined by upper bounds on metrics such as trust-checking time and TV size. Thus, we cannot construct a cone with more than p inputs.

[b] DAG constraint : We cannot include any net that generates a loop in the cone. This is done in order to have a cone that can be exhaustively tested as an "almost-combinatorial" circuit without requiring to sequence through all the states of the circuit.

[c] Sequential paths : We cannot construct a cone with internal paths that include more than a sequential element. This is done in order to be able to detect any flip-flop inclusion or deletion within the cone.

[d] Cut net : As we have seen earlier, there is no aim in covering a net if there is already a multiplexer on it. For this reason, we avoid expansion along such path.

We omitted the overlapping constraint described in section 4.1. In fact, the set of generated cones are no more than candidate for being inserted in the covering solution C_{sol}. Thus, we assume to use the splitting technique on C_{sol} in order to solve possible overlapping conflicts.

Algorithm 1 describes in greater detail the cone generation process. During the generation phase, we iteratively choose an intermediate *temp_cone* that maximizes the benefit metric, and that respects the given cone constraints. According to the general cutting and covering metric formulations, the generation algorithm may produce different cones depending on the particular partial solution C_{sol}. Given a seed f, the generation algorithm returns two alternative cones rooted in f: one is the maximum-benefit cone over all the intermediate steps, and the second is a maximal-size maximum-benefit cone that is generated in the last iteration when no further expansion is possible. While more alternative cones could be generated starting from seed f, this generally causes a combinatorial explosion in the number of generated cones. Moreover, we reduce the search space by using the set of input nets I_{forb} in order to prune those expansions which violates the constraints. A-priori pruning is also possible for those nets with large fanout which have small probability to be completely covered. The time complexity of the algorithm is related with the number of iterations and the number of possible expansion at each iteration. For the sake of simplicity, suppose a purely combinatorial circuit with single fanout nets. In this scenario, the time complexity is $O(p^2)$ where p is a-priori TPG size parameter. It might be argued that in an arbitrary circuit the enforced constraint may reduce the average case time complexity.

Algorithm 1 Cone Generation algorithm

```
Input : A functional output $f$ (cone seed)
Output : The maximum-benefit cone and the maximal-size maximum benefit cones
Notations :
$temp\_cone$ := the expanded cone at each iteration
$simple\_cone(f)$ := a cone only composed by a functional output $f$ (root net) and its input nets
$C_{int}$ := the set of maximum-benefit cones at each intermediate step
$I_{forb}$ := the set of input nets associated with forbidden expansions
$Inp(c)$ := the set of input nets for cone $c$
$C_{exp}$ := the set of generated expansion cones for each iteration
$cone(f_i, c)$ := the cone obtained including $simple\_cone(f_i)$ into cone $c$
$C_{sol}$ := the set of cones selected to be in the final solution
$benefit(c, C_{sol})$ := metric evaluation for cone $c$ given the solution $C_{sol}$
Algorithm :
// Create the initial cone from a functional output
$temp\_cone \leftarrow simple\_cone(f)$;
$C_{int} \leftarrow \{temp\_cone\}$;
$I_{forb} \leftarrow \emptyset$;
// Iterate only if there are possible expansions for the cone
while $Inp(temp\_cone) \not\subseteq I_{forb}$ do
    $C_{exp} \leftarrow \emptyset$;
    // Try an expansion at each of current cone's inputs
    for each $f_i \in Inp(temp\_cone)$ do
        // Construct the cone related to the selected expansion and verify the cone constraints
        $c' = cone(f_i, temp\_cone)$;
        if ([a] and [b] and [c]) then
            $C_{exp} \leftarrow C_{exp} \cup c'$;
        else
            $I_{forb} \leftarrow I_{forb} \cup f_i$;
        end if
    end for
    // Save the maximum-benefit cone for this iteration and continue the expansion
    Select $c'' \in C_{exp}$ with highest $benefit(c'', C_{sol})$;
    $temp\_cone \leftarrow \{c''\}$;
    $C_{int} \leftarrow C_{int} \cup \{temp\_cone\}$;
end while
// Return maximum-benefit cone $c$ and the maximal-size maximum-benefit cones $temp\_cone$
Select $c \in C_{int}$ with highest $benefit(c, C_{sol})$;
return $\{c, temp\_cone\}$;
```

4.3 Cone covering algorithm for overhead minimization

After cone generation, we can implement the cone-based iFIE architecture by selecting an appropriate composition of functional cones. Thus, we propose a covering algorithm which represents the key step for minimizing the multiplexer hardware overhead. The optimization strategy basically relies on the devised benefit metric and tries to compose a solution where cones share as many inputs as possible (thus sharing multiplexers) and cover as many net fanouts as possible (thus obviating the need for multiplexers for the completely covered net).

A set of candidate cones C_{gen} is generated by applying Algorithm 1 on each slice-grained function outputs of the FPGA circuit. An iterative process is then applied in order to compose the covering solution C_{sol}. At each step, the algorithm selects and includes a cone c which maximizes the benefit metric. The iterations only stop when all the coarse-grained functional outputs are covered by cones in C_{sol}. As mentioned, we have defined the benefit metric in order to take account of the current solution C_{sol}. For this reason, the selection of a cone c may modify the covering metric and the cut cost for some candidate cones, especially if overlapping with c. It might be argued that a new cone generation phase may obtain better results than the previous one based on an old C_{sol}. For this reason, we introduce a regeneration step for each iteration in order to always have the best possible set of candidate cones according to the updated benefit metric. All the presented steps concerning cone covering are summarized by Algorithm 2.

Another interesting aspect of the covering is related to the cuts imposed by C_{sol}. In other words, each cone $c \in C_{sol}$ requires a set of switching multiplexers in front of its inputs nets and this added structure identifies a cut (basically, the cone is cut on these nets and connected to TPG during trust-checking). By iteratively selecting cones to be included in the solution, the covering algorithm continues to insert cuts in the circuit. Often, some of the candidate cones may have all their inputs on cuts. It might be argued that the addition of these cones has no overhead since all the needed multiplexers are already placed into the circuit. For

Algorithm 2 Cone Covering algorithm

```
Input : All the circuit functional outputs $f \in F$
Output : A covering solution $C_{sol}$ with minimized overhead
Notations :
$C_{gen}$ := the set of candidate cones for covering
$cone\_generation(f)$ := the set of cones generated by Algorithm 1 using seed $f$
$seed(c)$ := the root net of cone $c$
$covered(f, C_{sol})$ := a flag indicating if functional output $f$ is covered by some cones in $C_{sol}$
$overlap(c, C)$ := a flag indicating if cone $c$ overlaps with any cone in the cone set $C$
For other notations, see Algorithm 1
Algorithm :
//Generate candidate cones using all the functional outputs in the circuit
$C_{gen} \leftarrow \cup_{f \in F} cone\_generation(f)$;
$C_{sol} \leftarrow \emptyset$;
// Iterate while there are uncovered functional outputs
while $\exists f \in F$ : $!covered(f, C_{sol})$ do
    // Add to the solution the cone with maximum-benefit
    select $c \in C_{gen}$ with highest $benefit(c, C_{sol})$;
    $C_{sol} \leftarrow C_{sol} \cup c$;
    // If the cut induced by cone $c$ creates some $\infty$−cones then add them to the solution
    while exist $c' \in C_{gen}$ with $benefit(c', C_{sol}) = \infty$ do
        $C_{sol} \leftarrow C_{sol} \cup c'$;
    end while
    // Update the set of candidate cones according to updated solution
    for each cone $c' \in C_{gen}$ do
        //Delete cones with root already covered by the solution
        if $covered(seed(c'), C_{sol})$ then
            $C_{gen} \leftarrow C_{gen} - c'$;
        end if
        //Regenerate cones that overlap with covered functional outputs in order to take account of the updated solution
        if $overlap(c', C_{sol})$ then
            $C_{gen} \leftarrow C_{gen} - c'$;
            $C_{gen} \leftarrow C_{gen} \cup cone\_generation(seed(c'))$;
        end if
    end for
end while
return $C_{sol}$;
```

this reason, the covering algorithm immediately includes ∞-cones (defined with zero cut cost) in the solution C_{sol}, covering additional slice-grained functional outputs and reducing the set of candidate cones C_{gen}. An example of ∞-cone is presented by Figure 4.4.

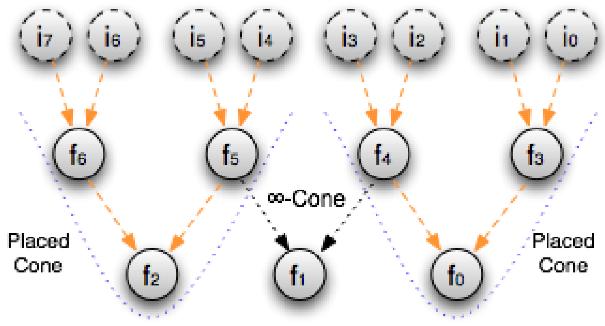


Figure 4.4: Example of ∞-cone

As we can see, there are two placed cones with primary output f_0 and f_2 that impose a cut on their input sets $\{i_0, i_1, i_2, i_3\}$ and $\{i_4, i_5, i_6, i_7\}$. Moreover, we can identify a ∞-cone rooted in f_1 with an input set $\{i_2, i_3, i_4, i_5\}$. Including this new ∞-cone in the solution C_{sol} we have the benefit of covering the coarse-grained functional output f_1 without additional hardware overhead.

Regarding the time complexity, the algorithm is based on an iteration which can be repeated at most n times where n is the size of the FPGA circuit in terms of functional outputs. This worst case scenario is associated with the basic FIE approach where we have simple cones only composed by a slice-grained functional output. Moreover, each iteration involves a linear search $O(n)$ for the cone selection and a cone regeneration with time complexity $O(p^2)$ repeated for a set of candidate cones C_{gen} which decreases iteration after iteration. Thus, we have an overall time complexity

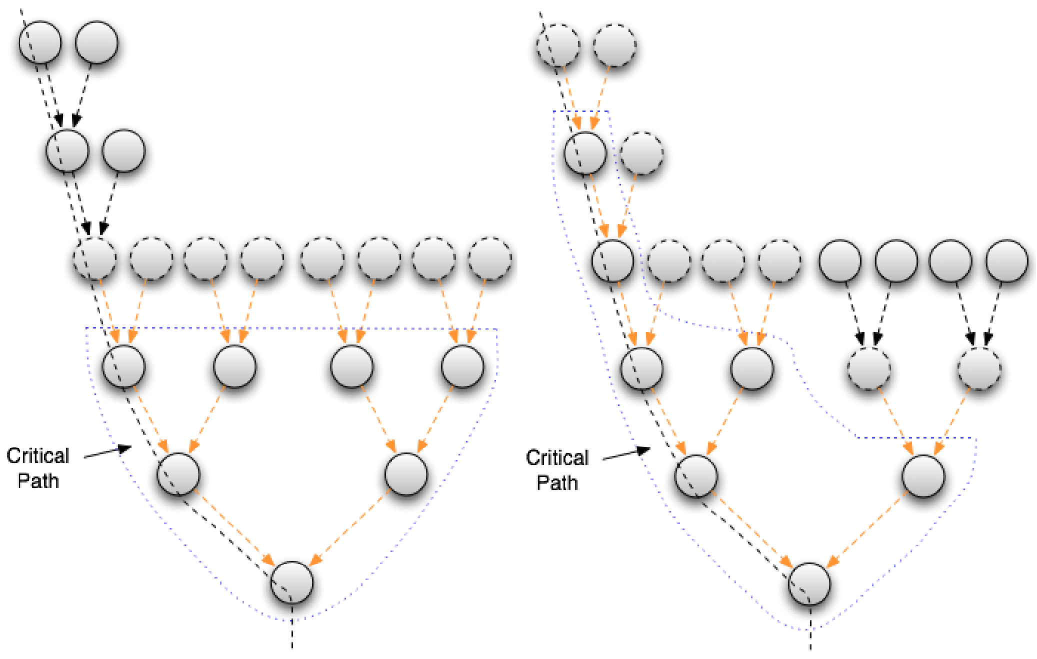
$$\sum_{i=1}^{n}\sum_{j=i}^{n} O(p^2) = O(p^2 n^2)$$

where p is a parameter related with the TPG size and the other cone constraints. In practice, we have a faster average case because we may have the immediate inclusion of ∞-cones (this reduces the number of external loops) and because the regeneration is only limited to a subset of overlapping cones (this reduces the number of internal loops).

4.4 Performance-aware cone covering algorithm

The presented covering algorithm is focused on reducing the hardware overhead related with multiplexers but it does not account for degradation in circuit performance. More in detail, the covering solution introduces cuts on some circuit nets. Each of these cuts add delay on all the paths from which is traversed since we have an added switching multiplexer necessary to implement the trust-checking structure. Measuring the delay in terms of traversed slices, each cut trivially increases the delay by one since the correspondent switching multiplexer is implemented by the means of a slice k-LUT.

An area-optimized covering algorithm may introduce too many cuts over a single critical path, leading to a significant performance degradation for the trusted FPGA circuit. Referring to the basic FIE approach where we have a cut on each net, the corresponding critical path is virtually doubled. It might be argued that the presented benefit metric is not tailored for minimizing the number of cut over critical paths. For this reason, we introduce a different approach of generating cones called deep cone generation. Intuitively, we can reduce the number of cuts on a critical path p by constructing a deep cone c in which most of the critical path length lies within the cone. Consider a circuit with an unique critical path p from its primary input i to a primary output f. We can virtually construct a cone rooted in f and composed by all the slice-grained functional outputs along the path p. In this way, the critical path delay is only degraded by a switching multiplexer (mandatory, since i is a primary input). More formally, we can define a critical cone as a cone c that contains a path p that will exceed a delay threshold D_{max} after the insertion of a cut. Applying the cone generation approach based on the benefit metric and presented in Algorithm 1, we may cover few nets belonging to path p leading inevitably to other cuts introduced by several "shallow" cones, and then a large delay increment. We thus change the expansion policy so that each critical cone is tailored in order to covers most of such critical path resulting in a deeper cone with less cuts on path p. Typically, a cone based on benefit metric has a more broad "shape" whereas a deep cone presents more levels along the critical path p. Figure 4.5 shows an example of the two cone types.



(a) Normal cone (b) Deep cone

Figure 4.5: Different approaches for cone generation

As we can see, the two cones have the same number of covered slice-grained functional outputs. However, a deep cone is capable of covering the entire critical path in order that a unique switching multiplexer is required in the final covering solution. In other words, the trusted FPGA circuit will only have a small performance degradation.

The proposed deep cone generation approach is described in detail by Algorithm 3. If during generation the cone is not critical, the algorithm exactly behaves as the conventional cone generation approach by expanding those inputs that maximize the benefit metric and by returning the maximum-benefit cone and

Algorithm 3 Deep-cone generation algorithm

```
Input : A functional output $f$ (cone seed) and the delay threshold $D_{max}$ corresponding to critical paths
Output : A deep-cone which minimizes the added delay on critical paths, otherwise same output of Algorithm 1
Notations:
$path_{del}(f, C_{sol})$ := the longest delay path traversing functional output $f$ (considered the current solution $C_{sol}$)
$deep\_mode$ := a flag used to switch between benefit metric expansion and critical path expansion
$I_{cri}$ := the set of input nets traversed by critical paths
For other notations, see Algorithm 1
Algorithm :
$temp\_cone \leftarrow simple\_cone(f)$; $C_{int} \leftarrow \{temp\_cone\}$; $I_{forb} \leftarrow \emptyset$;
$deep\_mode \leftarrow false$;
// Iterate only if there are possible expansions for the cone
while $Inp(temp\_cone) \not\subseteq I_{forb}$ do
    $C_{exp} \leftarrow \emptyset$; $I_{cri} \leftarrow \emptyset$;
    // As Algorithm 1, try an expansion at each of current cone's inputs
    for each $f_i \in Inp(temp\_cone)$ do
        $c' = cone(f_i, temp\_cone)$;
        if ([a] and [b] and [c] and [d]) then
            $C_{exp} \leftarrow C_{exp} \cup c'$;
            // If the current input net is on a critical path then activate a deep-cone expansion on that path
            if $(path_{del}(f_i, C_{sol}) + 1) > D_{max}$ then
                $deep\_mode \leftarrow true$;
                $I_{cri} \leftarrow I_{cri} \cup f_i$;
            end if
        else
            $I_{forb} \leftarrow I_{forb} \cup f_i$;
        end if
    end for
    // Finalize the cone expansion depending by the scenario
    if $deep\_mode$ and $I_{cri} \neq \emptyset$ then
        // Between all the critical paths, select the least expanded in order to balance cut length
        Select $f_c \in I_{cri}$ closest to root $f$;
        $temp\_cone \leftarrow cone(f_i, temp\_cone)$;
    else
        Select $c'' \in C_{exp}$ with the highest $benefit(c'', C)$;
        $exp \leftarrow c''$; $C_{int} \leftarrow C_{int} \cup temp\_cone$;
        $deep\_mode \leftarrow false$;
    end if
end while
// If critical path expansion activated then return deep-cone otherwise as Algorithm 1
if $deep\_mode$ then
    return $(temp\_cone)$;
else
    Select $c \in C_{int}$ with highest $benefit(c, C_{sol})$;
    return $\{c, temp\_cone\}$;
end if
```

the maximal-size maximum-benefit cone. However, if during the expansion process we encounter an input net over a critical path then the algorithm switch to a deep mode for which the expansion is only done through critical input nets. This strategy permits to build deep cones which cover most of the critical path. Suppose now that more than one input net is critical. In this scenario the algorithm selects for the expansion the one which is closest to its root. This strategy is oriented to balance the cuts over the various critical paths since an unbalanced scenario trivially corresponds to the delay of the path with more cuts. At last, the parameter D_{max} may simply specify the critical path length or may represent an acceptable performance degradation within which we can continue to enforce the benefit metric and the hardware overhead minimization. Moreover, all the delay considerations are done by the means of *Static Time Analysis* (STA) [37] applied to the technology-mapped FPGA circuit.

After having introduced deep cones, we devise a performance-aware covering algorithm based on two different covering strategies depending on whether a candidate cone is on a critical path or not. In order to enforce this distinction, we partition the set of generated cones C_{gen} in two subsets C_{cri} and C_{non_cri}. The former is composed of cones with critical input nets whereas the latter is not. We also introduce a new path covering metric related with the covering of critical cones. The rationale behind this metric is that we should place the smallest possible number of cones (and thus cuts) over critical paths. For this reason, we should prefer a cone that covers more uncovered nets traversed by critical paths. Given a cone c and a (partial) covering solution C_{sol}. We define the set of critical nets as

$$N_{cri}(c, C_{sol}) = \{f \in Int(c) : f \text{ is critical and not covered by } C_{sol}\}$$

where a net f is critical if $(path_{del}(f, C_{sol}) + 1) > D_{max}$ and "f is not covered by C_{sol}" means that no cones already included in the solution has f as root or internal net. The path covering metric is then defined as

$$path\_cov(c, C_{sol}) = |N_{cri}(c, C_{sol})|$$

This simple formulation permits to discriminate between two critical cones by counting how many critical nets they add to the covering solution C_{sol}. Ties in the

Algorithm 4 Performance-aware cone covering algorithm

Input : All the circuit functional outputs $f \in F$ and the delay threshold D_{max}
Output : A covering solution C_{sol} with minimized hardware and performance overheads
Notations :
$deep\_cone\_generation(f)$:= the set of cones generated by Algorithm 3 from seed f
$path\_cov(c, C_{sol})$:= path covering metric for cone c considering the solution C_{sol}
C_{cri} := the set of critical candidate cones that increase the circuit delay over the threshold
C_{non_cri} := the set of non-critical candidate cones that do not increase the circuit delay over the threshold
$to\_switch(c, C_{sol})$:= a flag indicating if cone c should switch from C_{cri} to C_{non_cri} or vice versa after C_{sol} updating
For other notations, see Algorithm 2 and 3
Algorithm :

```
//Generate candidate cones using all the functional outputs and partition them into critical and non-critical cones
$C_{gen} \leftarrow \cup_{f \in F} deep\_cone\_generation(f, D_{max})$;
for each $c \in C_{gen}$ do
    if $\exists f \in Inp(c) : (path(f, C_{sol}) + 1) > D_{max}$ then
        $C_{cri} \leftarrow c$
    else
        $C_{non\_cri} \leftarrow c$
    end if
end for
// Iterate while there are uncovered functional outputs
while $\exists f \in F : !\ covered(f, C_{sol})$ do
    if $C_{non\_cri} \neq \emptyset$ then
        //If possible, select between the available non-critical cones in order to not increase the delay
        select $c \in C_{non\_cri}$ with higher $benefit(c, C_{sol})$
    else
        // Otherwise select a critical cone which covers most functional outputs on critical paths
        select $c \in C_{cri}$ with higher $path\_cover(c, C_{sol})$ using $benefit(c, C_{sol})$ as tie-breaker
        $D_{max} \leftarrow D_{max} + 1$
    end if
    //As Algorithm 2, add the selected cone and potential $\infty$-cones to the solution
    $C_{sol} \leftarrow C_{sol} \cup c$;
    while exist $c' \in C_{gen}$ with $benefit(c', C_{sol}) = \infty$ do
        $C_{sol} \leftarrow C_{sol} \cup c'$;
    end while
    //As Algorithm 2, update candidate cones according to current solution
    for each cone $c' \in C_{cri} \cup C_{non\_cri}$ do
        if $covered(seed(c'), C_{sol})$ then
            $C_{cri} \leftarrow C_{cri} - c'$; $C_{non\_cri} \leftarrow C_{non\_cri} - c'$;
        end if
        //Some cones may need to be regenerated from their seed
        if $overlap(c', C_{sol})$ or $to\_switch(c', C_{sol})$ then
            $C_{cri} \leftarrow C_{cri} - c'$; $C_{non\_cri} \leftarrow C_{non\_cri} - c'$;
            Use $deep\_cone\_generation(seed(c'), D_{max})$ and partition the result cones into $C_{cri}$ and $C_{non\_cri}$
        end if
    end for
end while
```

path covering metric are broken by using the benefit metric. In other words, the algorithm maintains a secondary hardware overhead minimization goal. While $C_{non\_cri} \neq \emptyset$, the performance-aware covering algorithm optimizes multiplexer hardware overhead by selecting cones in C_{non_cri} along the lines of Algorithm 2. When $C_{non\_cri} = \emptyset$ and $C_{cri} \neq \emptyset$, the algorithm applies another performance-oriented strategy and selects cones based on the highest value of the path covering metric. At any critical cone inclusion, the delay threshold is clearly increased to $D_{max} + 1$ since the new critical path will contain an additional multiplexer. Moreover, it might be argued that after each cone selection some parts of the overall circuit timing are changed. Consequently, some cone should be moved from C_{non_cri} to C_{cri} (or vice versa) since may (or may not) be traversed from a critical path according to the new circuit timing. In order to always have the best possible cones (normal or deep), we regenerate all the cones involved in the switching between C_{non_cri} ans C_{cri} as well as all the cones overlapping with the last selected cone. Algorithm 4 describes in greater detail the performance-aware cone covering. We can notice a strong similarity with the previous cone covering algortithm. For instance, all the two algorithms stops when no more slice-grained functional outputs are available. Moreover, when $C_{non\_cri} \neq \emptyset$ they exactly behaves in the same way. In terms of time complexity, the performance-aware cone covering remains $O(p^2n^2)$ since the algorithmic structure of Algorithm 4 is maintained. However, it might be argued that average case of the performance-aware algorithm is faster since it separately deals with two smaller candidate cone sets C_{non_cri} to C_{cri} instead of an unique candidate cone set C_{gen}.

Chapter 5

Results ans simulations

The presented iFIE approach for ECC-based trust-checking deals with many different minimization techniques, which include architectural improvement such as multiplexer sharing over common nets or trust-checking without parity functions and algorithmic approaches for overhead minimization and simultaneous overhead/delay minimization. Aim of this chapter is to provide a set of experimental results in order to show a drastic reduction in the multiplexer overhead contribution. We note that our work is, to the best of our knowledge, the first of its kind, and there is thus no previous work to compare to if not the basic FIE approach suggested in [3]. Moreover, we propose a behavioral and a post-P&R simulations which validate the presented idea of trust-checking based on cone structures.

The chapter is organized in the following way. Section 5.1 introduces the experimental setup. Section 5.2 presents the experimental results related with the proposed architectural and algorithmic overhead minimization techniques. The effectiveness of the underlying ideas is verified showing the achieved improvement steps related with the use of each single technique. Thus, we compare our algorithms to some internal variations that establish that our chosen metrics are advantageous. Section 5.3 is then completely dedicated to simulations which construct a circuit representation of a cone PG with the aim of testing the capability of the ECC-based mechanism of detecting any Trojan injection or any malicious modification such as logic, sequential or interconnection tampering.

5.1 Experimental setup

In this section we report the experimental results concerned with the overhead minimization techniques introduced by the iFIE approach. All the architectural calculations as well as the covering algorithms have been implemented in C++ using Xcode developer tools [38]. The simulations were done using Xilinx ISE development tool [39]. Moreover, all the experiments were run on an Intel Core 2 Duo@2.66 GHz [40] under Max OS X [41] operating system. We consider a set of benchmark circuits from the ITC99 benchmark set [29]. In our experiments we specifically target the Xilinx Virtex 4 architecture [12]. For this reason, an initial high level description of the benchmark circuits is synthesized and then technology-mapped to the specified FPGA architecture producing a parsable *Xilinx Design Language* (XDL) file [42]. In others words, the benchmark set corresponds to a set of technology-mapped XDL circuits on which the proposed iFIE ECC-based trust-checking mechanism is then applied. We selects six benchmark circuits of different sizes and composed by slices with both combinatorial and sequential configurations.

5.2 Experimental results

The first experiment results are related with the effectiveness of multiplexer sharing over common nets. As mentioned in Section 3.2, this architectural technique is adopted by the iFIE approach and places an unique multiplexer upstream on each net instead of covering the fanout. The results are reported in Table 5.1.
As we can see, we compare the overhead of the basic FIE approach (a multiplexer on each fanout) with the one resulting after the application of the multiplexer sharing technique. The hardware overhead is then given as a percentage of the total size of the original circuit. The average results are instead obtained by weighting the circuits according to their sizes. Moreover, we report a column expressing the relative overhead reduction introduced by the multiplexer sharing technique. It might be argued that the effectiveness of the technique mainly depends by the

Benchmark	Size	Average	Multiplexer Overhead		Overhead
Name	(slices)	Fanout	Basic FIE	Sharing Technique	Reduction
b11	75	3,72	373.33%	101.33%	72.86%
b12	220	3,90	340.91%	89.55%	73.73%
b14	1275	4,06	358.98%	88.71%	75.29%
b15	1349	3,50	345.14%	105.93%	69.31%
b17	4106	3,54	344.40%	103.90%	69.83%
b20	2384	4,15	367.37%	88.63%	75.87%
Average	2384	3,81	352.45%	97.91%	72.22%

Table 5.1: Multiplexer overhead of sharing over common nets

average net fanouts. For instance, circuits b15 and b17 have a smaller overhead reduction due to their smaller average fanout. The analytical consideration introduced in Section 3.2 have predicted a multiplexer overhead around 91%. As we can see, the obtained experimental results are along that line. At last, considering an average reduction of 72.22%, we can positively evaluate the idea of multiplexer sharing over common nets.

The second set of experimental results is related with the another architectural improvement introduced by the iFIE approach. As explained in Section 3.3, we can avoid the use of structural parity functions in order to reduce the hardware overhead associated with the trust-checking architecture. The impact of this technique is shown by Table 5.2. As we can notice, the hardware overhead related with the ECC-based trust-checking mechanism, comparing the basic FIE approach with the technique without parity functions. Again, the overhead is given as percentage of the original circuit size and its average (given in the bottom row) is weighted. Moreover, we report the circuit size in terms of slice-grained functional outputs that will be included into the ECC-based trust-checking mechanism. This information is then used for calculating the size of the squared 2D ECC parity scheme which gives a clue about the amount of hardware associated with the trust-checking. In order to highlight the various benefits related with the iFIE ap-

Benchmark	Functional	2D ECC	Trust-checking Overhead		Overhead
Name	Outputs	Scheme	Basic FIE	w/o Parity Function	Reduction
b11	144	12 x 12	118.67%	70.67%	40.45%
b12	420	21 x 21	107.73%	69.55%	35.44%
b14	2262	48 x 48	75.84%	60.78%	19.86%
b15	2823	54 x 54	88.51%	72.50%	18.09%
b17	8529	93 x 93	79.42%	70.36%	11.41%
b20	4214	65 x 65	71.18%	60.28%	15.32%
Average	3065	49 x 49	79.13%	66.80%	15.58%

Table 5.2: Trust-checking overhead with and without parity functions

proach, we keep separated the multiplexer overhead from the remaining overhead associated with the TPG unit, the multiple ORA units and the parity functions. A reported average reduction of 15.58% clearly points out the effectiveness of the proposed architectural improvement. Moreover, we can notice that the technique without parity functions is more effective when we are dealing with smaller circuits. Intuitively, in this scenario the overhead contribution of parity functions is more relevant so its deletion has a greater benefit. On the other hand, if the circuit increases its size then the single parity function overhead remains the same (it basically requires a slice-grained functional outputs) whereas the number of PGs (and so parity functions) grows with complexity $O(\sqrt{n})$ assuming a squared 2D ECC parity scheme. For this reason, the relative overhead contribution decreases with the circuit size. At last, it might be argued that with coarse-grained functionalities the overhead contribution is heavier since the parity functions itself is implemented by the means of a coarse-grained functionality. However, this does not change the decreasing behavior related with circuit size.

The next set of experimental results proposes an interesting comparison between a generation and covering approach based on the presented benefit metric and an alternative metric which takes account of the covering but not of the cut cost. In this way, we can point out the effectiveness of the underlying algorithmic ideas. Moreover, the cones are generated under increasing TPG size constraints in order to evaluate the benefit of having large cones. The obtained multiplexer overhead is presented in Table 5.3 and Table 5.4.

	Multiplexer Overhead						
Bench	Range	TPG 12 bit			TPG 16 bit		
Name	[Min-Max]	Covering	Benefit	Reduction	Covering	Benefit	Reduction
b11	[5.33-78.91]%	61.33%	57.33%	28.17%	52.00%	44.00%	44.87%
b12	[2.27-101.33]%	44.09%	35.45%	65,02%	33.18%	30.00%	70.39%
b14	[1.33-89.54]%	51.92%	43.29%	51.65%	33.80%	33.02%	63.12%
b15	[2.89-105.93]%	52.85%	52.85%	50.11%	48.11%	42.55%	59.83%
b17	[1.97-103.90]%	53.51%	52.68%	49.30%	47.59%	41.94%	59.63%
b20	[0.71-88.63]%	54.49%	49.37%	44.30%	36.87%	35.53%	59.91%
Avg.	[2.42-97.91]%	53.29%	50.23%	47.05%	42.78%	38.93%	58.96%

Table 5.3: Multiplexer overheads using different metrics (TPG 12-16 bit)

	Multiplexer Overhead						
Bench	Range	TPG 20 bit			TPG 24 bit		
Name	[Min-Max]	Covering	Benefit	Reduction	Covering	Benefit	Reduction
b11	[5.33-78.91]%	45.33%	37.33%	53,23%	41.33%	32.00%	59.90%
b12	[2.27-101.33]%	32.73%	27.27%	73,09%	28.18%	25.91%	74,43%
b14	[1.33-89.54]%	28.63%	27.45%	69,34%	24.86%	25.10%	71,97%
b15	[2.89-105.93]%	45.00%	36.84%	65.22%	40.47%	33.36%	68,51%
b17	[1.97-103.90]%	43.42%	36.95%	64.44%	38.85%	32.81%	68,42%
b20	[0.71-88.63]%	30.41%	31.25%	64.74%	26.38%	26.80%	69,76%
Avg.	[2.42-97.91]%	38.11%	33.98%	64,18%	33.80%	30.15%	68,22%

Table 5.4: Multiplexer overheads using different metrics (TPG 20-24 bit)

For each benchmark circuit we have an upper and a lower overhead bounds related with the iFIE trust-checking architecture. Assuming the use of the multiplexer sharing technique, the upper bound represents a scenario in which each net has a multiplexer (this overhead was already calculated in Table 5.1). The lower bound instead represents the only switching multiplexers necessary for the circuit primary inputs. It might be argued that this overhead is mandatory in order to implement the ECC-based trust-checking mechanism. Moreover, the tables are divided in sections which correspond to different TPG sizes. Within each section we have two different overhead results still expressed in terms of a percentage of the original circuit. One column corresponds to the covering metric and the other corresponds to the more advantageous benefit metric. A third column expresses the relative overhead improvement achievable starting from the upper bound and using the proposed covering algorithm.

Taking a look at the results, it might be argued that the proposed cone generation and covering algorithms obtain a reasonable iFIE cone-based solution far away from the tremendous multiplexer overhead of the basic FIE approach proposed in [3]. Considering a large TPG constraint, we can achieve an average multiplexer overhead corresponding to 30.15% of the original circuit with a considerable relative reduction of 68,22% respect to the upper bound. As expected, the benefit metric represents the best solution since includes both the concepts of covering metric and cutting cost. Moreover, it is not surprisingly that increasing the TPG size results in lower overhead, since we can select larger and deeper cones to cover the circuit.

A more comprehensive analysis of the cone-based iFIE approach is possible from Table 5.5. Considering the favorable scenario of a large TPG constraint, we can evaluate what is the real implementability of the proposed trust-checking mechanism. In the first part of Table 5.5 we have both the multiplexer and the trust-checking overheads. More in detail, this latter overhead is determined by a TPG unit and by multiple ORA units, assuming of not having structural parity functions. Referring to the results in Table 5.2, it might be argued that the cone-based iFIE approach does not only decrease the multiplexer overhead but also the one related with trust-checking. The underlying reason is that coarse-grained functionalities decrease the number of element to be included in the 2D ECC parity schema and, consequently, the size of the trust-checking components. More in detail, we have shown in Section 3.4 that the cone-based iFIE approach scales down by a factor l the overhead associated with multiple ORA units, where l is the average cone size in terms of covered coarse-grained functional outputs. From the sum of the two contributes we obtain the total overhead associated with a trusted

Bench	TPG 24 bit				
	Overhead			Masking	Running
Name	Multiplexer	Trust-checking	Total	Probability	Time
b11	32.00%	30.67%	62.67%	$625 \cdot 10^{-6}$	61 s
b12	25.91%	27.27%	53.18%	$51.0 \cdot 10^{-6}$	153 s
b14	25.10%	17.57%	42.67%	$2.43 \cdot 10^{-6}$	2102 s
b15	33.36%	22.46%	55.82%	$1.32 \cdot 10^{-6}$	5089 s
b17	32.81%	21.92%	54.72%	$0.135 \cdot 10^{-6}$	31519
b20	26.80%	18.62%	45.43%	$0.629 \cdot 10^{-6}$	9086 s
Avg.	30.15%	20.77%	50.92%	$6.91 \cdot 10^{-6}$	8002 s

Table 5.5: Overall cone-based iFIE overhead (TPG 24 bit)

FPGA circuit. There are several reasons for which an average total overhead of 50.92% can be considered reasonable. In absolute terms, there is no objective comparison for which we can positively or negatively judge the overhead associated with our checking technique. We may consider a duplicate-and-compare architecture in which the trust-checking is based on output comparison between two copies of the same circuit. Despite the underlying security weaknesses, it might be argued that this duplicate-and-compare architecture offers trust-checking with at least 100% of hardware overhead, a lot more than our cone-based iFIE approach. Referring to the NIE approach introduced in [3], we have the same underlying ECC-based mechanism with very low overhead due to partial dynamic reconfiguration and due to single PG trust-checking. However, the iFIE approach has the main advantage of being combined with bitstream encryption, a feature not currently compatible with partial dynamic reconfiguration.

In general, it might be argued that an average hardware overhead of 50.92% does not really represent an implementation issue for the proposed iFIE approach. Typically, a significant portion of an FPGA is left unused by an application circuit so the iFIE approach is immediately implementable. On the other hand, a slightly larger FPGA chip should be available on the market considering the level of nanoscale integration provided by the modern IC industry. We believe this is well worth the cost, especially considering the sensitive military or commercial application in which a trusted FPGA circuit is used. From another perspective, it might be argued that overhead is not completely a disadvantage since the associated trust-checking components will cover the unused part of the FPGA device from tampering or Trojans. No matter how, a trusted FPGA circuit must occupy 100% of the FPGA device in order to extend the trust-checking mechanism to all the available CLBs. In fact, we have seen that the remaining area should be configured as zero functions and included in the 2D ECC parity scheme. For this reason, the introduced overhead may be considered as an alternative “filler”.

Coming back to Table 5.5, we also have a column related with the masking probability p_{mask}. Intuitively, this parameter represents the robustness of the underlying ECC-based trust-checking mechanism. Not surprisingly, a 2D ECC parity schemes with more functional outputs also have smaller p_{mask}. According to the theoretical analysis presented in [3], the obtained masking probability is the order of $O(1/n^2)$ where n represents the number of slice-grained functional outputs in the circuit. In addition, we have shown in Section 3.4 that the cone-based iFIE approach increases the masking probability by a factor l^2. Despite this disadvantage, the obtained results show reasonably small probabilities, especially for larger circuit. However, it is still possible to apply the RecECC technique which makes the masking probability astronomically small without increasing the hardware overhead associated with the iFIE approach. The last column of Table 5.5 presents the algorithm running times. In short words, it might be observed that the algorithm duration is coherent with the circuit size increasing.

Last but not least, Table 5.6 provides results comparing the algorithms with and without performance considerations. Beside the multiplexer hardware overhead, we report the original FPGA circuit critical path length in terms of slices and the added delay in the covering solution expressed as a percentage of the critical path length. As we can see, the performance-aware cone covering algorithm in collaboration with the deep cone generation approach yields a small area increasing of about 5%, but significantly reduces the trusted FPGA circuit performance overhead of 33.71% (from 65.85% to 44.38%). It thus offers a good minimization tradeoff between area and delay overheads. It might be argued that differently from hardware overhead the introduced delay has no bright side from the perspective of the ECC-based trust-checking mechanism. In other words, the performance degradation intrinsically affects the trusted FPGA circuit which may thus require a slower clock cycle. However, we can consider the circuit trustworthiness as an additional FPGA circuit design parameter. The designer is then responsible for balancing the various design aspects such as area, delay, power and security in order to identify the right trade-off which satisfies the application requirements. In this perspective, a delay increasing is counterbalanced by the added protection against tampering and Trojan injections, a very sensitive feature indispensable for some military or commercial applications. At last, we notice that the performance-aware cone covering algorithm is faster. Despite the two proposed approaches have the same quadratic time complexity, the performance-aware algorithm works on reduced sets of candidate cones (C_{cri} and C_{non_cri}) instead of a large one (C_{gen}) leading to a better average-case time complexity.

TPG 24 bit		w/o Performance Considerations			w/ Performance Considerations			Performance
Bench	Critical	Multiplexer Overhead		Running	Multiplexer Overhead		Running	Overhead
Name	Path	Hardware	Performance	Time	Hardware	Performance	Time	Reduction
b11	8	32.00%	25.00%	61 s	37.33%	12.50%	13 s	-50.00%
b12	6	25.91%	50.00%	153 s	35.91%	66.67%	28 s	+33.33%
b14	38	25.10%	73.68%	2102 s	29.65%	44.74%	1373 s	-39.29%
b15	21	33.36%	66.67%	5089 s	41.81%	33.33%	3229 s	-50.00%
b17	25	32.81%	68.00%	31519 s	35.97%	44.00%	26670 s	-35.29%
b20	38	26.80%	65.79%	9086 s	33.56%	50.00%	3883 s	-24.00%
Avg.	22.67	30.15%	65.44%	8002 s	35.35%	43.38%	5866 s	-33.71%

Table 5.6: Covering algorithms with and without performance considerations

5.3 Validation simulations

In this section, we validate the cone-based iFIE approach by simulating the proposed architecture during a trust-checking phase. The aim is to provide a proof-of-concept for the ECC-based trust-checking mechanism applied to coarse-grained functionalities. In [3] it has been proposed a simulation for the ECC-based mechanism applied to the NIE scenario. In our thesis work, we propose two different simulation approaches more focused on the cone structures. The first simulation approach is concerned with a behavioral model in which the iFIE trust-checking architecture is described as a circuit of high-level functional components whereas the second one is concerned with a post-P&R simulation where the iFIE trust-checking architecture is described in terms of slices mapped on a FPGA device, a scenario closer to a real implementation of a trusted FPGA circuit.

Our aim is to describe the trust-checking architecture composed by a cone PG, by a TPG and by an ORA unit. Given an exhaustive TVs sequence, we verify the ability of detecting tampering or Trojan insertions. More in detail, we intentionally introduce functional modification into the cone PG and we observe if the trust-checking architecture raises an alarm. For this reason, we define the tamper detection probability P_D as the percentage of times that an inserted tamper is detected by the 2D ECC parity scheme and the false alarm probability P_{FA} as percentage of times that a tamper-insertion alarm is raised without intentional or unintentional modifications in the circuit. We are interested in testing different kinds of tampering such as modification of the logic functions implemented by the slice-grained functional outputs, insertion or deletion of flip-flops, interconnection rerouting or Trojan injection in the zero-configured slices. Moreover, we are not only interested in tampering the cones but also the related trust-checking components.

The first behavioral simulation is manually composed by considering a simple PG with five cones outputs (each cone with average size of three slice-grained functional outputs) randomly selected from benchmark b12. Each of those cones has at most 16 input nets (TPG constraint) and is represented in the behavioral model as a composition of k-LUTs and flip-flops which agrees with the slice configurations and interconnections. Given these five cones, we are then able of constructing an unique functional component representing the entire cone PG. It might be argued that this is the component on which we will target the tampering in order in order to verify the ECC-based mechanism. In the trust-checking architecture adopted for this behavioral simulation we suppose to simply impose a fixed even parity scheme by the means of a structural parity function for the PG. We implement this latter combinatorial function from an untampered copy of the PG component in order to be able of producing the correct parity sequence associate with the cones. Moreover, the TPG unit is implemented as a 16-bit sequential counter whereas the ORA unit is implemented by XOR gates (in fact, we suppose even polarity). The final behavioral model is then obtained by assembling these trust-checking components with the cone PG component as shown by Figure 5.1.

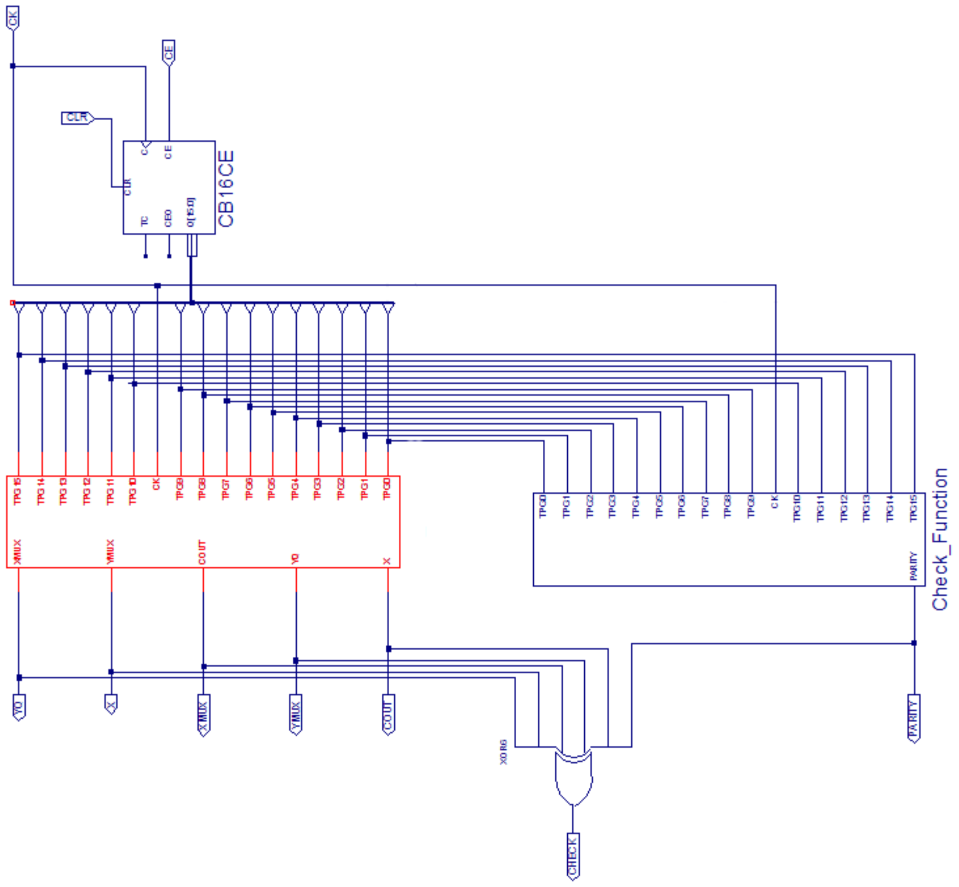


Figure 5.1: Simulation schematic constructed with ISE

Looking at the picture, it is possible to recognize the TPG unit (a counter on the top), the ORA unit (a 6-input XOR gate on the bottom), the parity function (the blue component connected to the TPG on the right) and the cone PG (the other red component on the left). The presented schematic model essentially represents a cone PG during trust-checking. Thus, the behavioral simulation will consist of generating all the possible TVs expecting a zero PV on the ORA output (we are implementing an even parity scheme). With a 16-bit counter and a 10 *ns* clock, the simulation should last at least 65.536 μs. A tamper is trivially detected by observing any value different from zero on the ORA output. An example of a positive simulation without tampering detection is shown by Figure 5.2.

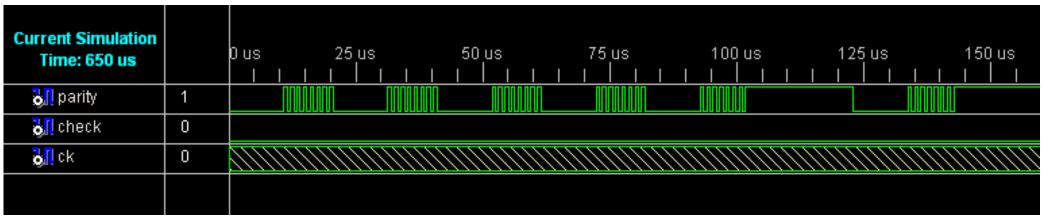


Figure 5.2: Tamper-free behavioral simulation

As we can see, the ORA output (second logic signal) is stable at zero. In other words, the ORA is producing a zero parity vector corresponding to a correct trust-checking phase. In order to observe the opposite scenario, we introduce a functional modification in the schematic component representing the cone PG. Running again the simulation, we obtain a result similar to Figure 5.3.

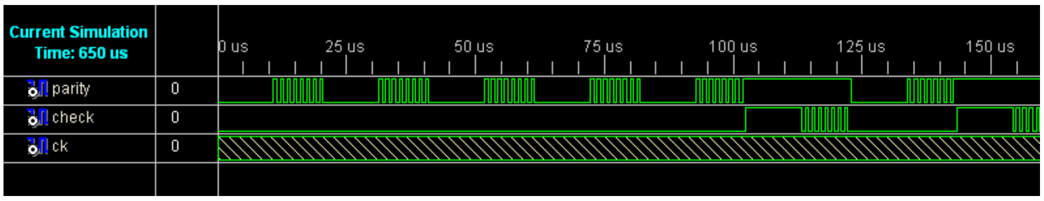


Figure 5.3: Behavioral simulation with tamper

Not surprisingly, this time the ORA does not produce a zero vector. This kind of output represents a detection signal and an alarm for the ECC-based mechanism. The behavioral simulation tests the trust-checking architecture under a set of different tampers (25 logic tampers, 10 sequential tampers, and 15 internal cone interconnection tampers) inserted within the cone PG, keeping track of the obtained detections and false alarms. We assume to insert each tamper individually since we are just interested into simulating a single PG trust-checking architecture and not the robustness against masking of the entire 2D ECC parity scheme. As we have seen, two tampers in the same PG can mask each other. However, considering the two PGs in the opposite 2D dimension we can still assume a single tampering scenario. According to the simulation results reported by Table 5.7, the ECC-based trust-checking mechanism maintains its effectiveness when applied to coarse-grained functionalities. In fact, every tampering is detected with a certain detection probability P_D and without raising any false alarm in case of untampered PG (thus, implying a zero false alarm probability P_{FA}).

Tamper Type	P_D	P_{FA}
Logic	100%	0%
Sequential	100%	0%
Interconnection	100%	0%
Total	100%	0%

Table 5.7: Behavioral simulation results in terms of P_D and P_{F_A}

A second simulation approach based on a post-P&R scenario is used in order to validate the cone-based iFIE trust-checking architecture at an FPGA implementation level. More in detail, we describe the cone PG and the related trust-checking components as a set of slices placed and routed on the FPGA device. Again, the aim is to analyze the reaction against tamper insertions, evaluating the effectiveness of the ECC-based mechanism on coarse-grained functionalities. A post-P&R model can be composed only dealing with the low level details of XDL files [42]. In other words, we parse an FPGA technology-mapped circuit file composing again a cone PG and the related trust-checking architecture. The cones are directly extracted by a technology mapped file representing the benchmark b12. On the other hand, the TPG unit is still a 16-bit counter and the ORA unit is still composed by XOR gates. However, their models are not functional but directly correspond to FPGA technology-mapped circuits (basically, we have a set of slices configured as flip-flops or k-LUTs). Differently from the behavioral simulation, we also include some zero-configured slices in the technology-mapped cone PG in order to simulate Trojan insertions. Moreover, the trust-checking mechanism will be based on an off-chip parity comparison in order to avoid a structural implementation of the parity function. When the trust-checking architecture model is completed, we apply placing and routing in order to obtain the final model ready for the simulation. This FPGA post-P&R model is shown by Figure 5.4.

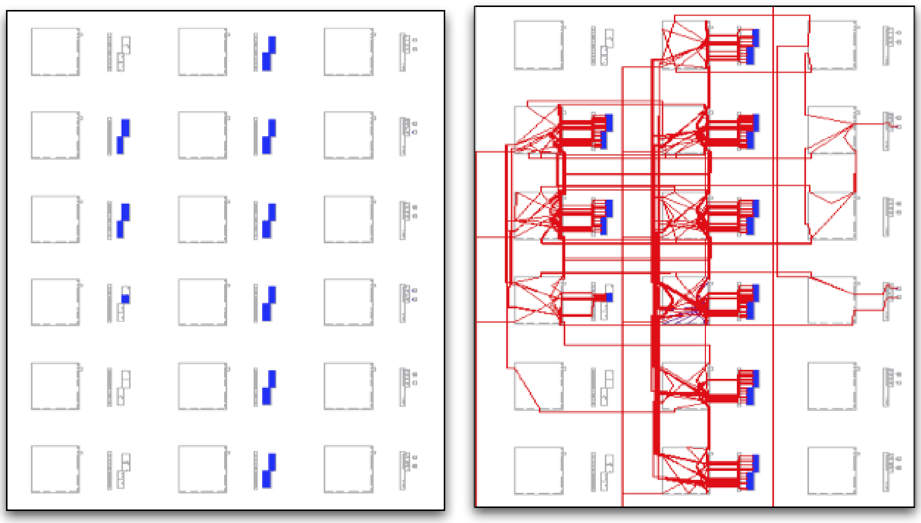
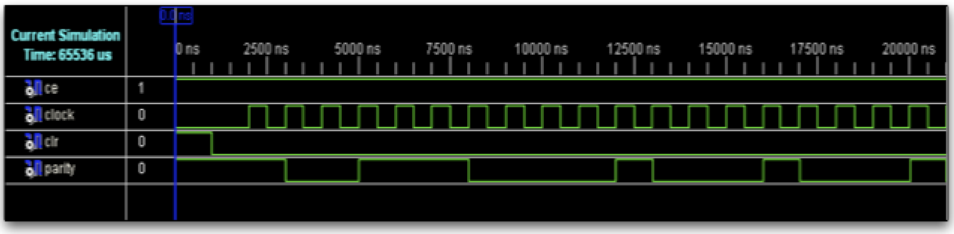

(a) Placed (b) Placed and routed

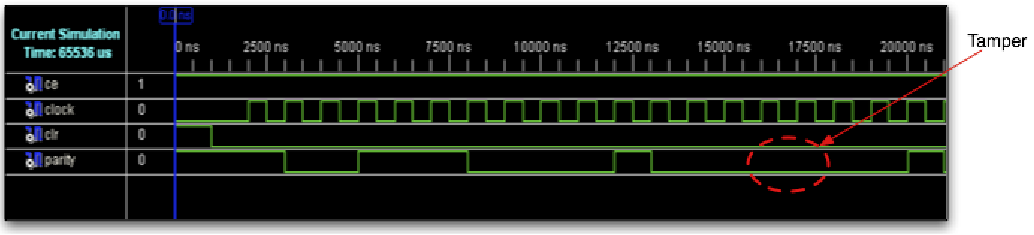
Figure 5.4: Post-P&R simulation model

As we can see, we have a magnified view of the FPGA device where the simulation model is placed and routed. The blue-colored rectangles represent the slices composing the cone PG and the related trust-checking architecture. Moreover, the routing is presented in red. As mentioned, we implement the trust-checking mechanism by the means of an off-chip comparison. For this reason, we store a bit vector containing the expected PV. During the simulation, we obviously compare the PV produced by the ORA with the the expected one. Figure 5.5 clarifies this procedure by showing a screenshot of the expected ORA output. As we can see, Figure 5.5a presents a correct PV produced by a genuine cone PG. On the other hand, Figure 5.5b shows an unsuccessful trust-checking simulation for a tampered model in which the output PV is different from the expected parity sequence.

Thus, it is still possible to validate the effectiveness of the cone-based iFIE ar-



(a) Correct PV



(b) Tampered model

Figure 5.5: Post-P&R trust-checking simulation

chitecture by inserting different kinds of tampers in the post-P&R model and observing if the ECC-based trust-checking mechanism is capable of detecting these malicious modifications. In order to provide a more comprehensive validation against tampering and Trojans insertions, we target not only the cone PG but also the empty slices, the TPG unit and the ORA unit. As for the behavioral simulation, the results reported by Table 5.8 correspond to a certain detection probability P_D and to a zero probability P_{FA} of raising false alarms.

Tamper Type	P_D	P_{FA}
Logic	100%	0%
Sequential	100%	0%
Interconnection	100%	0%
Empty slices	100%	0%
TPG & ORA	100%	0%
Total	100%	0%

Table 5.8: Post-P&R simulation results in terms of P_D and P_{F_A}

In conclusion, this post-P&R simulation shows not only the effectiveness of the cone-based iFIE trust-checking but also its implementability on a real FPGA device.

Chapter 6

Conclusions

In this thesis I have presented an on-chip functionality-based trust-checking mechanism capable of detecting with very high probability any malicious tampering or Trojan insertion in a FPGA circuit. The underlying goal is concerned with the more general idea of *trusted FPGA design* for which FPGA circuits, especially if used in sensitive application, must perform only the functionality for which they were originally designed. Introducing a significative extension of the ideas and methodology proposed in [3], I have provided an hardware and delay efficient implementation of the iFIE trust-checking architecture which protects the FPGA circuit at any stage of its life and, differently from the NIE approach, can be combined with bitstream encryption, an highly desirable feature in sensitive applications. The proposed approach is also general enough that it can be adapted to any FPGA device family with little effort. Moreover, it can be applied for fault-detection purposes since the underlying 2D ECC parity scheme is able of detecting any functional modification in the deployed trusted circuit.

The obtained multiplexer overhead minimization is the result of some architectural and algorithmic techniques applied to the basic FIE approach. Firstly, I have introduced multiplexer sharing over common nets, a technique which drastically reduces the hardware overhead without affecting the robustness of the ECC-based mechanism. Then, I have removed the structural implementation of parity function by substituting the off-chip random polarity comparison with an off-chip parity comparison. Finally, I have introduced a more important architectural innovation related with cone structures in order to reduce the number of switching multiplexers on the internal connections. This cone-based iFIE architecture is supported by algorithmic approaches available for optimal cone generation and selection in order to provide overhead minimization or even simultaneous overhead-delay minimization. The proposed heuristics take advantage of a benefit metric based on the concepts of covering and cut cost. Moreover, an alternative deep cone generation strategy is available for critical path covering. Last but not least, I have introduced the RecECC technique, a novel challenge-response trust-checking protocol which is robust against replay attacks and that reduces the masking probability to an astronomically small value.

The presented iFIE architecture is supported by theoretical considerations which assess the mechanism robustness against tampering and justify the overhead reduction. Other experimental results show that the proposed techniques can reduce the average multiplexer hardware overhead to 30.15% of the original FPGA circuit, a satisfactory result considering an overhead of about 350% related with the basic FIE technique. Moreover, the performance-aware cone covering can limit the performance degradation to 43.38% which may be reasonable considering a scenario where delay and security represent two opposite design aspects of the FPGA circuit. On the whole, the iFIE architecture can be implemented with an average total overhead of 50.92% which practically does not represent an issue. In fact, we should consider that typically a significant portion of an FPGA is left unused by an application circuit. Alternatively, it is still possible to use a slightly larger FPGA chip with an affordable additional cost. At last, a behavioral and a post-P&R simulations validate the trust-checking architecture related with a cone PG showing an optimal detection probability P_D=100%. Combining this result with the theoretical consideration about the 2D ECC parity scheme, it possible to assess the iFIE architecture robustness. In addition, this consideration is also supported by an average masking probability p_{mask} of $6.91 \cdot 10^{-6}$. In case this probability value is not satisfactory, we can still apply the RecECC technique which further reduces p_{mask} without overhead increasing.

The presented thesis work contributes to a novel research area recently supported by DARPA [2] and concerned with the design of trusted FPGA circuits and trust-checking mechanisms suitable for detecting FPGA circuit tampering or Trojan injection. The proposed iFIE approach may be considered quite mature from a theoretical point of view. However, it still needs development work in order to provide a set of EDA tools capable of implementing the presented *trusted FPGA design* flow. A more ambitious project may involve a redesign of the available FPGA devices in order to naturally integrate the ECC-based trust-checking mechanism in their chip layout. For instance, we can add a set of switching multiplexer to each CLB. Intuitively, the added performance overhead will be small since multiplexers are implemented as transistor circuits instead of k-LUTs. The PGs composition cannot be decided at chip fabrication time. For this reason, it seems logical to consider the RecECC technique with its capability of configuring PGs at runtime.

A drawback of the iFIE trust-checking mechanism is concerned with the routing between cones. Considering the internal interconnections, these are covered by the trust-checking since their modification also affects the cone functionality. On the other hand, this is not true for connections between cones. It might be argued that it is difficult to craft meaningful tampers by only changing the inter-cone routing. Despite this favorable consideration, it may be useful to introduce an extension of the trust-checking mechanism in order to protect inter-cone routing. Last but not least, it may be interesting to investigate different ECCs or hashing codes in order to substitute the underlying trust-checking mechanism and possibly reduce the hardware and the performance overheads.

Appendix A

List of Abbreviations

2D	Two Dimensional
AES	Advanced Encryption Standard
ASIC	Application Specific Integrated Circuit
ATPG	Automatic Test Pattern Generation
BTE	Bitstream Trust Engine
CLB	Configuration Logic Block
COTS	Commercial Off-The-Shelf
DAG	Direct Acyclic Graph
DES	Data Encryption Standard
DMR	Digital Right Management
DPA	Differential Power Analysis
ECC	Error Correcting Code
EMP	ElectroMagnetic Pulse
FIE	Fully Integrated Embedding

FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
IC	Integrated Circuit
iFIE	improved Fully Integrated Embedding
IOB	Input Output Block
IP	Intellectual Property
LVS	Layout-Versus-Schematic.
LUT	Look-Up Table
NIE	Non Integrated Embedding
ORA	Output Response Analyzer
PCB	Printed Circuit Board
PG	Parity Group
PI	Primary Input
PIE	Partially Integrated Embedding
PO	Primary Outputs
PV	Parity Vector
PUF	Physical Unclonable Function
RecECC	Reconfigurable Error Correcting Code
SRAM	Static Random Access Memory
STA	Static Time Analysis
SUM	Secure Update Mechanism

TMR	Triple Modular Redundancy
TPG	Test Pattern Generator
TV	Test Vector
VLSI	Very Large Scale Integration
XDL	Xilins Design Language

Bibliography

[1] Steve Trimberger. Trusted fpga design in fpgas. *Proceedings of the 44th annual Design Automation Conference*, pages 5–8, June 2007.

[2] DARPA. Darpa program for trust in integrated circuits (trust).

[3] Shantanu Dutt and Li Li. Trust-based design and check of fpga circuits using two-level randomized ecc structures. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 2(1):1–36, March 2009.

[4] Cynthia E. Irvine and Karl Levitt. Trusted hardware: Can it be trustworthy? *Proceedings of the 44th annual Design Automation Conference*, pages 1–4, June 2007.

[5] Francis Wolff, Chris Papachristou, Swarup Bhunia, and Rajat S. Chakraborty. Towards trojan-free trusted ics: problem analysis and detection scheme. *Proceedings of the conference on Design, automation and test in Europe*, pages 1362–1365, March 2008.

[6] Xiaoxiao Wang, Mohammad Tehranipoor, and Jim Plusquellic. Detecting malicious inclusions in secure hardware: Challenges and solutions. *Proceedings of the 2008 IEEE International Workshop on Hardware-Oriented Security and Trust*, pages 15–19, June 2008.

[7] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptolog*, pages 388–397, August 1999.

[8] Junjun Gu, Gang Qu, and Qiang Zhou. Information hiding for trusted system desig. *Proceedings of the 46th Annual Design Automation Conference*, pages 698–701, July 2009.

[9] Jarrod A. Roy, Farinaz Koushanfar, and Igor L. Markov. Epic: ending piracy of integrated circuits. *Proceedings of the conference on Design, automation and test in Europe*, pages 1069–1074, March 2008.

[10] Yousra Alkabani and Farinaz Koushanfar. Active control and digital rights management of integrated circuit ip cores. *Proceedings of the 2008 international conference on Compilers, architectures and synthesis for embedded systems*, pages 227–234, October 2008.

[11] G. Edward Suh and Srinivas Devadas. Physical unclonable functions for device authentication and secret key generation. *Proceedings of the 44th annual Design Automation Conference*, pages 9–14, June 2007.

[12] Xilinx. *Virtex-4 FPGA User Guide*. Xilinx, http://www.xilinx.com/support/documentation/user_guides/ug070.pdf, December 2008.

[13] Dylan McGrath. Gartner dataquest analyst gives asic, fpga markets clean bill of health, June 2005.

[14] Brian Dipert. Cunnung circuit confound crooks, October 2000.

[15] Thomas Wollinger and Christof Paar. How secure are fpgas in cryptograpic applications? *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, pages 91–100, September 2003.

[16] Thomas Wollinger, Jorge Guajardo, and Christof Paar. Security on fpgas: State-of-the-art implementations and attacks. *ACM Transactions on Embedded Computing Systems (TECS)*, 3(3):534–574, August 2004.

[17] Srivaths Ravi, Anand Raghunathan, Paul Kocher, and Sunil Hattangady. Security in embedded systems: Design challenges. *ACM Transactions on Embedded Computing Systems (TECS)*, 3(3):461–491, August 2004.

[18] Ted Huffmire, Brett Brotherton, Timothy Sherwood, Ryan Kastner, Timothy Levin, Thuy D. Nguyen, and Cynthia Irvine. Managing security in fpga-based embedded systems. *IEEE Design and Test of Computers*, 25(6):590–598, November 2008.

[19] Philippe Adell and Greg Allen. Assessing and mitigating radiation effects in xilinx fpgas. Technical report, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California, February 2008.

[20] Cristiana Bolchini, Davide Quarta, and Marco D. Santambrogio. Seu mitigation for sram-based fpgas through dynamic partial reconfiguration. *Proceedings of the 17th ACM Great Lakes symposium on VLSI*, pages 55–60, March 2007.

[21] Thomas Eisenbarth, Tim Güneysu, Christof Paar, Ahmad-Reza Sadeghi, Dries Schellekens, and Marko Wolf. Reconfigurable trusted computing in hardware. *Proceedings of the 2007 ACM workshop on Scalable trusted computing*, pages 15–20, November 2007.

[22] Benoit Badrignans, Reouven Elbaz, and Lionel Torres. Secure update mechanism for remote update of fpga-based system. *International Symposium on Industrial Embedded Systems*, pages 221–224, June 2008.

[23] Oliver Kömmerling and Markus G. Kuhn. Design principles for tamper-resistant smartcard processors. *Proceedings of the USENIX Workshop on Smartcard Technology*, pages 2–2, May 1999.

[24] Marco Maggioni. Techniques for fully-integrated embedding of design and verification logic for trusted fpga circuits. Master's thesis, University of Illinois at Chicago, May 2009.

[25] Shu Lin and Daniel J. Costello. *Error Control Coding: Fundamentals and Applications*. Computer Applications in Electrical Engineering. Prentice-Hall, 1983.

[26] Peter J. Ashenden. *The Designer's Guide to VHDL*. Morgan Kaufmann, 2nd edition, June 2001.

[27] Donald Thomas and Philip Moorby. *The Verilog® Hardware Description Language*. Springer, 5th edition, October 2002.

[28] Joe Burkis. Clock tree synthesis for high performance asics. *Proceedings of the 4th ASIC Conference and Exhibit*, pages 9.8.1–9.8.3, September 1991.

[29] University of Texas at Austin, http://www.cerc.utexas.edu/itc99-benchmarks/bench.html. *ITC99 Benchmark Home Page*, 1999.

[30] James Cong and Yuzheng Ding. Flowmap: An optimal technology mapping algorithm for delay optimization in lookup-table based fpga designs. *IEEE Transaction on Computer-Aided Design,*, 13(1):1–12, January 1994.

[31] James Cong and Yuzheng Ding. On area/depth trade-off in lut-based fpga technology mapping. *IEEE Transaction on VLSI Systems*, 2(2):137–148, June 1994.

[32] Gabriele Saucier, Daniel Brasen, and J.P. Hiol. Partitioning with cone structures. *Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design*, pages 236–239, November 1993.

[33] Daniel Brasen and Gabriele Saucier. Using cone structures for circuit partitioning into fpga packages. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(7):592–600, July 1998.

[34] Zhonghai Lu Axel Jantsch Ming Liu, Wolfgang Kuehn. Runtime partial reconfiguration speed investigation and architectural design space exploration. *International Conference on Field Programmable Logic and Applications*, pages 498–502, August 2009.

[35] Jon Kleinberg and Eva Tardos. *Algorithm Design*. Addison Wesley, 2006.

[36] Thomas H. Cormen, Ronald L. Rivest Charles E. Leiserson, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, third edition, September 2009.

[37] J. Bhasker and Rakesh Chadha. *Static Timing Analysis for Nanometer Designs: A Practical Approach*. Springer, first edition, April 2009.

[38] Apple Computer, http://developer.apple.com/technologies/tools/xcode.html. *Xcode Developers Tool*.

[39] Xilinx, http://www.xilinx.com/tools/logic.htm. *Xilinx ISE*.

[40] Intel, http://www.intel.com/itcenter/products/core/core2/index.htm. *Intel Core 2 Processors*.

[41] Apple Computer, http://www.apple.com/macosx/. *Mac OS X Snow Leopard*.

[42] N.J.Steiner. A standalone wire database for routing and tracing in xilinx virtex, virtex-e, and virtex-ii fpgas. Master's thesis, Electrical Engineering Virginia Polytechnic Institute and State University, August 2002.

Ringraziamenti

Giunti al termine di questa esperienza universitaria, arriva il momento di fermarsi un attimo per guardare indietro. Parlando con onestà, il lavoro di questa tesi mi ha insegnato che nella vita bisogna prendere i momenti difficili così come vengono, e lavorarci su per tirare fuori un qualcosa di buono. Si deve magari giocare un po' in difesa ma al momento giusto si deve avere la forza per un contropiede, per un cambiamento che cambia le tue prospettive.

Vorrei comunque ringraziare le molte persone incontrate lungo il cammino. Il primo pensiero va sempre alla mia famiglia che mi vuole bene e che è sempre alle mie spalle nei casi di difficoltà. Grazie per avermi cresciuto con i giusti valori e avermi dato l'opportunità di costruirmi un futuro con lo studio. Ringrazio in particolare mamma Orietta e papà Giovanni per avermi fatto così, sempre con un sorriso per tutti. Un bacio va alla mia Benedetta che, nonostante la mia lontananza per la maggior parte dell'anno, vive pensando tutti giorni a me. D'altronde ogni rosa ha le sue spine e i sacrifici che si fanno oggi saranno la base per una luminosa vita insieme domani.

Un ringraziamento di tutto il cuore va al "Santa" che piú di un relatore é un amico su cui posso sempre contare. In tutta sincerità, è grazie ai suoi consigli se mi trovo dove sono ora. Ringrazio anche la mia zia americana Leila, senza la quale non saprei come la mia esperienza di dottorato a Chicago potrebbe continuare. Come si dice, la vita è tutto un "What goes around comes around" quindi spero anch'io un giorno di fare per qualcun altro quello che ho avuto la fortuna di ricevere. Un altro ringraziamento va ai miei attuali advisor alla University of Illinois, Prof. Tanya Berger Wolf e Prof. Jie Liang, che mi hanno lasciato questa estate libera per finalmente completare questa laurea. Una menzione anche per il Prof. Shantanu Dutt, per la collaborazione nella ricerca nell'area *trusted FPGA design*.

Per concludere, un ringraziamento generico per tutti gli amici e le persone che mi sono vicine. A volte, quando sono un po' triste, mi basta pensare alle avventure passare insieme che mi torna subito il sorriso. Anche se per gli impegni c'é sempre meno per stare insieme, vi porto sempre nel cuore.

Milano, Giugno 2010.

Marco