FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Learning System for IC Security

Francisco Carvalho Viana Pimentel Torres

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Vítor Manuel Grade Tavares

Second Supervisor: Ronald DeShawn Blanton (Carnegie Mellon University)

July 23, 2015

U.PORTO

FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

MIEEC - MESTRADO INTEGRADO EM ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES | **2014/2015**

A Dissertação intitulada

"Learning System for IC security"

foi aprovada em provas realizadas em 23-07-2015

o júri

Presidente Professor Doutor José Carlos dos Santos Alves
Professor Associado do Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

Professor Doutor Paulo Francisco Silva Cardoso
Professor Auxiliar do Departamento Eletrónica Industrial da Universidade do Minho

Professor Doutor Vitor Manuel Grade Tavares
Professor Auxiliar do Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

O autor declara que a presente dissertação (ou relatório de projeto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, parágrafos, ou outros extratos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são corretamente citados.

Autor - Francisco Carvalho Viana Pimentel Torres

Faculdade de Engenharia da Universidade do Porto

Resumo

Todos os circuitos integrados, de certo modo, são vulneráveis a diversos ataques para diferentes fins, tais como a obtenção de um acesso privilegiado a recursos não licenciados ou para *reverse engineering*. Muitas vezes, as portas de teste e *debug* têm de ser mantidas operacionais, proporcionando consequentemente uma maneira de aceder ao chip. Essas portas são projetadas para executar diferentes tarefas antes do circuito integrado chegar ao mercado, para maximizar a qualidade através da análise de respostas particulares a vetores de teste aplicados a blocos de circuitos especificados. Assim, as operações de teste durante as diferentes fases de fabricação do circuito integrado implicam o acesso a funcionalidades profundas. Estas funcionalidades estão normalmente escondidas de um utilizador normal do sistema, e para que o *hacker* consiga ter acesso às instruções não documentadas, ele ou ela necessita de atuar nas portas de uma forma diferente de um utilizador legítimo, de maneira a tentar descobrir as funções escondidas e em seguida, ter acesso a todos os recursos do circuito integrado se bem sucedido.

Para a segurança de circuitos integrados, diversos métodos e algoritmos foram já propostos, por exemplo, o acesso por palavra-chave ou encriptação de dados. No entanto, também têm sido bem sucedidas estratégias complexas de ataque que são capazes de superar esses mecanismos de defesa. Novos e robustos mecanismos de segurança são, por isso, sempre necessários.

Este trabalho segue essas diretrizes e é focado na implementação e modificação de um sistema de segurança em hardware (FPGA), que assinalará se um ataque está a ser realizado. A ideia principal é construir um classificador que monitoriza o sistema, segue a atividade no circuito integrado e decide, baseado no comportamento, se uma determinada utilização é legítima ou não. Desse modo, um ataque segue um procedimento de erro que vai criar um padrão de actividade diferente dos padrões normais, o que por sua vez permite a detecção.

Assim, a previsão é efetuada por um classificador constituído por uma floresta de árvores de decisão que são treinadas utilizando técnicas de *bagging*. O sistema de deteção é composto por 4 blocos principais e está em constante interação com o JTAG. O sistema final está implementado numa *Xilinx Zynq-7000 ZC706*.

Abstract

All integrated circuits (IC), in some extent, are vulnerable to numerous attacks for different purposes, such as getting privileged access to non licensed features or for reverse engineering. Often the testing and debugging ports are kept operational, providing a way to access the chip. These ports are designed to perform several different tasks before the IC reaches the market, to maximize quality through the analysis of particular responses to testing vectors applied to specified circuit blocks. Thus, testing operations during the different phases of the IC fabrication implies accessing deep core functionalities. These functionalities are normally hidden from a normal user of the system, and in order for the hacker to get insight of undocumented access instructions and functionalities, he or she needs to actuate on the ports differently from a legitimate user, to try and to discover those hidden functions, then gaining access to all the resources of the IC if successful.

For the security of integrated circuits, several methods and algorithms are already in place, for instance, password-based access or data encryption. However, there also have been successful complex attacking strategies that are capable of overcoming such defense mechanisms. New and robust methods of security are then always on demand.

This work follows these guidelines and is focused on the implementation and modification of a security scheme in hardware (FPGA) that will flag if an attack is being performed. The main idea is to build a classifier that monitors the system and tracks the activity in the IC and then decides, based on its behavior, whether an utilization in a given moment is legitimate or not. Thereby, an attack follows an error procedure that will create an activity pattern distinct from the normal patterns, which then allows the detection.

Thus, the prediction is made by a random forest classifier that is trained using bagging techniques. The detection system is composed by 4 main blocks and it is in constant interaction with the JTAG. The final system is implemented in a *Xilinx Zynq-7000 ZC706*.

Agradecimentos

Esta dissertação representa o final de cinco anos repletos de trabalho e aprendizagem, mas também cheio de grandes amizades, diversão e momentos incríveis que os tornaram nos melhores anos da minha vida. Os cinco anos em que tive a oportunidade de viver no estrangeiro duas vezes, conhecer outras culturas e muitas pessoas. Tudo isto só foi possível devido às condições proporcionadas pela minha família durante toda a minha vida.

Quero agradecer, por isso, a toda a minha família, em especial aos meus pais, por tudo o que me proporcionaram. Educação, motivação, felicidade, inspiração, carinho, conselhos, vontade de ser sempre melhor, e muitas outras coisas que não caberiam nesta página. Muito obrigado por toda a paciência que tiveram comigo. Tudo o que sou deve-se ao que me ensinaram e mostraram ao longo da vida... tive e tenho o melhor que poderia ter.

Aos orientadores desta dissertação, Professor Vítor Tavares, Professor Shawn Blanton e Xuanle Ren. Obrigado por todos os conselhos, disponibilidade e troca de experiências. Este trabalho não seria possível sem o vosso apoio. Ao Professor Vítor, que segue o meu percurso desde o meu segundo ano e é um dos responsáveis pelo meu gosto especial pela Eletrónica. Ao Professor Shawn, muito obrigado pelos três meses inesquecíveis que vivi nos Estados Unidos da América, não só dentro de todo o ambiente da Carnegie Mellon University, mas também por todas as trocas de experiências durante os nossos almoços informais. Ao Xuanle, muito obrigado por toda a disponibilidade e colaboração durante todo o trabalho.

À minha irmã Rita, que evidentemente está incluída em tudo que disse no segundo parágrafo, mas também por todos os momentos que tivemos nestes últimos três anos que vivemos juntos no Porto. Muito obrigado. (Ah... e obrigado também por me fazeres o jantar de vez em quando.)

À minha namorada Mar, muito obrigado por toda a paciência que tiveste durante toda esta tese, por ficares acordada até bastante tarde devido à diferença horária de cinco horas, apenas para me ligares e dares-me motivação quando tudo parecia não funcionar. Sempre quis fazer melhor e tu foste responsável por isso. Por todos os momentos incríveis e inesquecíveis que vivemos e por me tornares uma pessoa melhor, *muchas gracias guapa*. Tenho a certeza que teremos muitos mais juntos.

A todos os Professores que tive a oportunidade de conhecer e que contribuíram para a minha aprendizagem e formação como Engenheiro.

E por último mas não menos importante, a todos os meus amigos que partilharam estes cinco anos comigo. Isto não teria sido o mesmo sem vocês. Muito obrigado por estarem sempre presentes e pelos momentos incontáveis e inesquecíveis que vivemos. Felizmente, estão muitas pessoas incluídas aqui, desde todos os meus amigos de ERASMUS até aos craques da equipa de Andebol da AEFEUP, mas queria deixar uma palavra especial ao Carlitos, Bernie, Mário, Fisgas, Bife, Tico, Pereira, Moutinho, César, Mafalda, Joana, Jarro, Pete D, Laurinha, Pantera, Xico, Quim, Diogo e Vinícius.

Esta dissertação é, por isso, dedicada a todos os presentes neste texto. Muito obrigado!

Francisco Carvalho Viana Pimentel Torres

Acknowledgments

This document represents the end of five years full of work and learning, but also full of friendships, fun and incredible moments which make them the best five years of my life. The five years where I had the opportunity of living abroad two times, experience new cultures and meet a lot of people. All of this was only possible due to the conditions that my family provided me during my entire life.

I want to thank to all my family, specially my parents, for everything they gave me. Education, motivation, happiness, inspiration, affection, advises, desire to be better and better, and many others that would not fit in this page. Thank you for all the patience that you had with me. All that I am today, is because of what you taught me and showed me... I had and I have all the best that I could have thanks to you.

To my supervisors of this work, Professor Vítor Tavares, Professor Shawn Blanton and Xuanle Ren. Thank you for all your advises, availability and share of experiences. This work would not be possible without all your support. To Professor Vítor, that follow my work since my second year, and is one of the Professors that are responsible for my special interest in the Electronics area. To Professor Shawn, thank you very much for providing me an unforgettable three months in the United States of America, not only inside the Carnegie Mellon University research environment, but also for all the share of experiences during our informal lunches. To Xuanle Ren, thank you very much for all your availability and collaboration during the entire work.

To my sister Rita, which is included in everything that I said in second paragraph, but also for all the moments that we had during these last three years that we lived together in Porto. Thank you very much. (Ah... and thanks also for making me the dinner sometimes.)

To my girlfriend Mar, many thanks for all the patience that you had during this entire work, staying wake until late due to the 5 hour difference just to call me giving me motivation, when everything seemed to not work. I always wanted to do better and you were responsible for that. For all the incredible and unforgettable moments that we had and for making me a better person, *muchas gracias guapa*. I am sure we will have many more together.

To all the professors that I had the opportunity to meet and contribute for my learning and formation as Engineer.

And last but not least, to all my friends that shared these five years with me. This would not have been the same without all of you. Thank you for being always there and for the uncountable and unforgettable moments that we had together. Fortunately, many people are included here, starting from my ERASMUS friends until the star players of AEFEUP's Handball Team, but a special word to Carlitos, Bernie, Mário, Fisgas, Bife, Tico, Pereira, Moutinho, César, Mafalda, Joana, Jarro, Pete D, Laurinha, Pantera, Xico, Quim, Diogo and Vinícius.

This thesis is, therefore, dedicated to all of you present in this text. Thank you very much!

Francisco Carvalho Viana Pimentel Torres

"Obstacles don't have to stop you. If you run into a wall, don't turn around or give up. Figure out how to climb it, go through it, or work around it."

Michael Jordan

Contents

List of Figures

List of Tables

Abbreviations and Symbols

AES	Advanced Encryption Standard
AMBA	Advanced Microcontroller Bus Architecture
ASIC	Application-Specific Integrated Circuit
AXI	Advanced eXtensible Interface
BIST	Built-In Self Test
BSC	Boundary Scan Cells
BSR	Boundary Scan Register
CART	Classification And Regression Tree
CHAID	CHi-squared Automatic Interaction Detector
CLB	Configurable Logic Blocks
CREG	Control Registers
DES	Data Encryption Standard
DFT	Design for Testability
DR	Data Register
DT	Decision Tree
FF	Flip-Flops
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
IC	Integrated Circuit
ID3	Iterative Dichotomiser 3
IEEE	Institute of Electrical and Electronics Engineers
IP	Intellectual Property
IR	Instruction Register
I2C	Inter-Integrated Circuit
I/O	Input/Output
JTAG	Joint Test Action Group
LSB	Least Significant Bit
LUT	Look-Up Table
MCM	Multi Chip Module
MLUT	Memory Look-Up Table
MMCM	Mixed-Mode Clock Managers
NVM	Non-volatile Memory
OS	Operating System
PCA	Principal Component Analysis
PCB	Printed Circuit Board
PUF	Physically Unclonable Function
RTI	Run-Test/Idle
SLIC	Statistical Learning In Chip

TAP	Test Access Port
TCK	Test Clock
TCU	Test Control Unit
TDI	Test Data In
TDO	Test Data Out
TLR	Test-Logic-Reset
TMS	Test Mode Select
TRST	Test Reset
SAM	Security Authentication Module
SMpL	Single Module per Level
SMpL-P	Single Module per Level Parallel
SoC	System-On-Chip
SODIMM	Small Outline Dual In-Line Memory Module
SPI	Serial Peripheral Interface
SSH	Secure Shell
UART	Universal Asynchronous Receiver/Transmitter
UN	Universal Node
UN-P	Universal Node Parallel

Chapter 1

Introduction

This chapter presents the motivation, challenges and objectives of this work. It also describes the structure of the document.

1.1 Motivation

The Joint Test Action Group (JTAG), also named IEEE 1149.1, is the standard for test access port and boundary scan architecture. It is widely used for the testing, debugging and failure analysis of integrated circuits (ICs), since it dramatically improves the testability and reduces the testing time compared to other testing methods [1]. However, the high testability provided by the JTAG may cause security problems in the chip. Since it needs to be left operational in order to be used, it inevitably provides a backdoor that can be used for reverse engineering purposes or to dump critical data. These backdoors are undocumented functions that grant access to areas of the chip normally not directly readable by the end-user. Thereby, it is possible to perform several different attacks through scan-based side-channel attacks to retrieve secret keys, for example, of cryptographic chips (DES and AES). Besides this, the user-defined functions introduce even more backdoors, and since that reverse engineering techniques are becoming more common and powerful, the chip stays even more susceptible to attacks.

There are several methods and mechanisms to protect the JTAG, and some of them will be presented in this document, but each one of them has its own set of drawbacks. For instance, the password of an authentication scheme can leak, or the original JTAG can be modified, requiring additional hardware to support lock and unlocks mechanisms or even the network availability that is needed for a server-based authentication. However, since the attacks consist in different actions and patterns compared to the normal actions that occur in the integrated circuit, Machine Learning concepts such as Decision Trees or Random Forests, can be used for the detection of these attacks. Therefore, in this work, a security system is defined and implemented in an FPGA, which is able to monitor the activity of the JTAG, track user behavior, detect illegitimate accesses and learn the patterns in order to become more accurate and efficient [2].

1.2 Objectives

The work consists in an hardware implementation and modification of the security system proposed in [2], replacing the decision tree with a random forest which is an ensemble of decision trees. The final goal is to have the complete system working properly in a *Xilinx Zynq-7000 All Programmable SoC ZC706*, with the JTAG included, taking into account the restricted timing constraints. Thereby, to achieve this, the work was divided into four big steps, each one containing several tasks.

First, it was necessary to study all the necessary background for the implementation and to understand the concept behind the algorithm. The concept of JTAG, which is the port that can provide undesired access to the integrated circuit, was carefully studied as well as some concepts of Machine Learning that were used in this work, since it is based on a learning system that makes predictions based on previous patterns. It was also necessary to define the tools to use and to be familiarized with them.

The second step consisted in the simulation of random forests using MATLAB, in order to train them using the appropriate data set for the algorithm, which contains information extracted from JTAG programs. This part also included the determination of the optimal number of trees for the forest.

The third step is related to the hardware design and implementation of the system. It is focused on an on-chip classifier, which started from a single Decision Tree followed by the Random Forest. The random forest classifier was designed following two different approaches and both of them were implemented on the FPGA. The other three modules of the system were also synthesized individually in order to be integrated in the complete system.

Finally, the JTAG TAP Controller was integrated with the rest of the system. The entire system was simulated, synthesized, implemented in the FGPA and optimized achieving the final goal of this work.

1.3 Challenges

The system that is being implemented, presented in March 2015, is a recent and innovative work. It is, to the best of the author's knowledge, the first work that uses an on-chip learning-based approach to ensure the security of the JTAG, without modifying the IEEE 1149.1. It uses the OpenSPARC T2 [3] as a platform to describe a typical JTAG behavior and it achieves high accuracy in detecting the illegitimate accesses to the chip.

Besides being an extremely interesting system, implementing it in a real hardware such as an FPGA makes it even more challenging and innovative, since there are only a few defense mechanisms for JTAG and integrated circuits that are implemented in a real board.

One of the biggest challenges of this work is meeting the timing requirements of the JTAG. Since this is a system that monitors the JTAG behavior in real time, it cannot miss any input bit or any instruction. New instructions are always being triggered and the on-chip classifier needs

to classify each one of them in time. Therefore, the implementation should be able to run under a clock frequency faster than the normal JTAG frequency. Accordingly, two different clocks need to be synchronized properly.

Another challenge is to find the optimal number of decision trees for the random forest. On one hand, several trees perform better and the error of classification is reduced. On the other hand, the trees need to be stored on chip, and therefore a large number of trees requires much more memory space.

In summary, the challenges of this work mainly involve the speed/frequency of the classifier, synchronization between the main system and the JTAG TAP Controller and, area overhead and power consumption that are commonly considered in hardware implementation.

1.4 Structure

The following chapters are organized as below.

Chapter 2 contains a brief description of the background of JTAG and Machine Learning. The JTAG section starts with an overview and description of main concepts, followed by the JTAG interface signals, registers and TAP Controller. The Machine Learning section also has an overview of what is machine learning and which typical prediction approaches exist, and then a more specific subsections focused on the Decision Trees and Random Forests. This subsection of Decision Trees also contains some hardware implementations. Finally, this chapter concludes with the state of the art about Integrated Circuit Security, presenting some possible attacks, defense mechanisms and solutions developed so far.

Chapter 3 has two main parts, although it contains four sections. The first part contains a detailed description of the system that is being implemented, referring its normal operation, how the attacks are detected, its complete architecture, with a brief description of each module, and finally a comparison with other existing defense mechanisms. The second part, corresponding to section 3.4, presents the training of the trees and how the optimal number of trees for the random forest was found.

Chapter 4 starts with a brief description of the FPGA that is used in this work, presenting its components and resources. However, this chapter is focused on the hardware implementation of the modules of the system, presenting how each module works, how they are implemented and which resources they utilize. It has a main focus on the classifier, which is the essential part of the project and was implemented following two different approaches.

Chapter 5 presents the synthesis of the JTAG TAP Controller module, the integration of all the modules and the synthesis of the complete system. It also describes the interface between the processing system and the programmable logic of the FPGA, how the inputs are applied and the outputs are read. Moreover, it contains the communication with the computer, the results of the final implementation and a discussion of the results.

Finally, chapter 6 presents the conclusions of this work and future steps that may be taken to improve the current system and integrate it in real life.

Chapter 2

Background and Related Work

Since this work is related with JTAG, by controlling the illegitimate access to this port using Machine Learning, a brief background on both of these topics is given in sections 2.1 and 2.2. Also, in section 2.3, it is provided a summary of the work done in the area of attacks and defense mechanisms for integrated circuits.

2.1 Joint Test Action Group (JTAG)

2.1.1 Overview

The in-circuit testing was done with a mechanism that probes the backs of the boards with nails, called *bed-of-nails*. However, with the constant miniaturization of device packing and the development of surface-mounted packaging, this mechanism became obsolete. Thus, IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture [4], most known as JTAG (Joint Test Action Group), appeared in 1990 as a new solution to the in-circuit testing. Essentially, the best advantage offered by the boundary-scan technology is the ability to set and read values of the pins without the need of a direct physical access.

A JTAG enabled device, which is shown in Figure 2.1, must have a serial scan path, known as the Boundary-Scan Register (BSR), which consists of a parallel-in, parallel-out shift register between the core logic of the device and the pins. This Boundary-Scan Register is an internal hardware that provides a register at each pin position, which is made of a sequence of cells, known as Boundary-Scan Cells (BSC). It is possible to load the boundary-scan cell for a particular input, shifting a pattern into the boundary-scan register, and using the value at that pin to drive the system circuitry [1].

Besides the debugging and testing part, JTAG also allows a programmer device to transfer data into the internal non-volatile device memory. Hence, some of these device programmers have both purposes – debugging and programming.

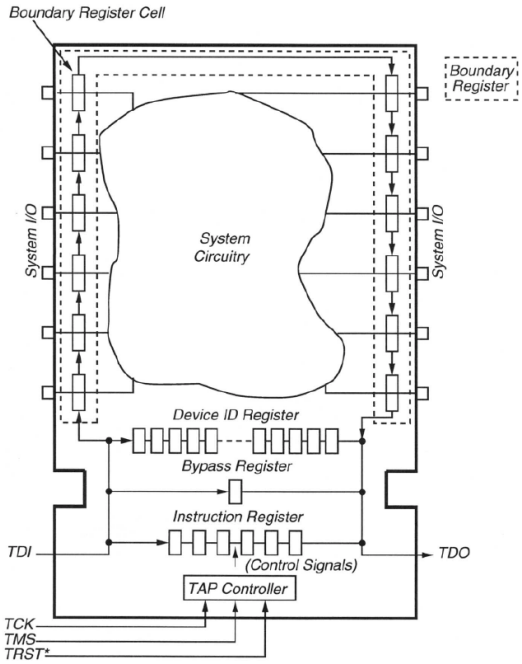


Figure 2.1: JTAG Device Structure from [1]

2.1.2 Interface Signals

The TAP Controller, that will be presented in section 2.1.4, has the following signals to support the operation of the boundary scan: [5]

- **Test Clock (TCK)**: which synchronizes the internal state machine operations and must be capable of operating at an independent clock rate from the system clock rate, asynchronously from the system circuitry;
- **Test Data In (TDI)**: which is the serial input to the boundary register. It is sampled at the rising edge of the TCK when the internal state machine is in the correct state;
- **Test Data Out (TDO)**: which is the serial output from the boundary register and is valid on the falling edge of the TCK when the internal state machine is in the correct state;
- **Test Mode Select (TMS)**: which causes the testing hardware to enter various testing modes;
- **Test Reset (TRST)**: which is optional and, when available, can reset the state machine of the TAP controller. It is generally an active-low signal.

2.1.3 Registers

Boundary scan has two types of associated registers: instruction registers and data registers (two or more). The Instruction Register (IR) holds the current instruction and its content determines to which of the data registers the signals should be passed.

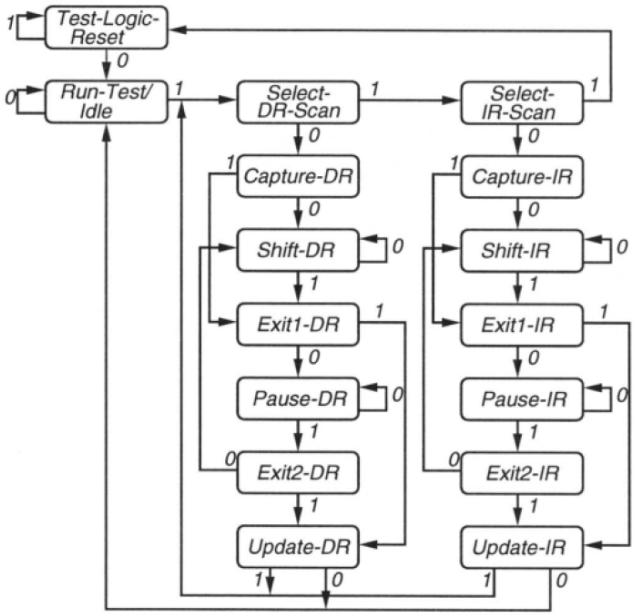


Figure 2.2: TAP Controller State Diagram from [1]

The Data Registers (DR) are the Boundary Scan Register, the BYPASS register and the IDCODES register. The first, as already explained, is the main testing data register and is used to move data to and from the I/O (Input/Output) pins of a device. The BYPASS is a single-bit register that passes directly the information from the TDI to the TDO, which can be extremely useful when many boundary registers of all components on the PCB are chained together into one long shift register and we want to ignore some of them that are not involved in the current test [1]. Finally, the IDCODES contains the ID code and revision number for the device.

2.1.4 Test Access Port (TAP) Controller

The Test Access Port (TAP) Controller is the JTAG Interface and implements a 16-state finite state machine that recognizes the boundary scan communication protocol and controls the operation of the boundary scan test hardware through internal signals. Its transitions are controlled by the TMS signal and it controls the behavior of the JTAG. The path on the left sets or retrieves information from a data register, while the path on the right is responsible for the instruction registers. The TAP Controller state diagram is shown in Figure 2.2.

Some of these states play an extremely important role in this work since they need to be analyzed to extract some features, as will be further explained. Thereby, a short description of them is given in this section [6].

The Test-Logic-Reset enables the normal operation of the IC, disabling the test logic. The Run-Test-Idle enables the test logic in the IC only if specific instructions are present. The Select DR-Scan state controls if the system enters in the left path (of the data register) or if it goes to the Select IR-Scan while the Select IR-Scan controls if the system enters in the path of the instruction

register. If not, the system goes back to the Test-Logic-Reset state. Capture-IR loads a pattern of specific values when a rising edge of the TCK is detected and the Shift-IR connects the instruction register between the TDI and the TDO. The pattern loaded in the Capture-IR state starts to be shifted in this state. Exit1-IR simply controls if the system should enter in the Pause-IR state or move to the Update-IR state and the Pause-IR state, as the name indicates, stops temporarily the shifting of the instruction register. The Exit2-IR decides if the system goes back to the Shift-IR state or to the Update-IR. In the Update-IR state the current instruction is already defined. This state is one of the most important for the whole implementation. Capture-DR loads data into the data registers that are chosen by the current instruction. Finally, the other 5 states do not need further explanation since they are very similar to the states of the IR that were already defined.

2.1.5 Instructions

For a device to be considered compliant with the IEEE 1149.1 Standard, a set of instructions [1] must be available, such as:

- **BYPASS**: This instruction is used to bypass the boundary scan chain with a one-bit bypass register. It allows the testing of other devices in the JTAG chain without any unnecessary overhead;

- **EXTEST**: The EXTEST instruction is used mainly to test off-chip circuits and board-level interconnections independently of the chip. To do this, the signals coming into the chip in the boundary scan register are captured and the signals coming out from the chip from the boundary scan register are driven;

- **SAMPLE/PRELOAD**: The SAMPLE/PRELOAD instruction is used to obtain a snapshot of the normal component input and output signals and store them in the first of the two master-slave flip-flops in the boundary scan ring;

Other common instructions available can be INTEST, IDCODE, RUNBIST, CLAMP, USERCODE or HIGHZ, but these will not be described in this brief background chapter.

2.1.6 Summary

Since nowadays it is no longer possible to test the majority of the circuits exclusively with the *bed-of-nails* mechanism, boundary scan is becoming absolutely essential for the electronic testing. It supports external testing with an automatic test equipment, boundary scan chain reconfiguration and response compressor for built-in-self-test (BIST) [1]. JTAG is considered the most widely used device test mechanism [6] and allows the reduction of costs, improves the product quality and increases the speed of the development. Despite being an extremely powerful tool which incorporates testing and debugging capabilities, as already referred in this document, JTAG's features can be used and explored by hackers to gain access to the chip [7].

2.2 Machine Learning

2.2.1 Overview

Machine Learning draws on ideas from a large set of disciplines such as artificial intelligence, probability and statistics, information theory, control theory, computational complexity, among others, and addresses the question of how to build computer programs that improve their performance at some task through experience. There are several machine learning algorithms, with numerous different applications. These algorithms learn from data and make predictions and decisions without following only specific instructions previously programmed [8].

The main idea behind Machine Learning is to learn from data, using a predictor and having a prediction in the end. This predictor is deterministic, it is a function and basically it receives the inputs and computes the predicted output values of those inputs. It is the "secret" of the Machine Learning, and it should be built with training examples and learning algorithms. The training examples/inputs should be pairs of x, which is the input and $f^*(x)$, which is what should be seen as an output for that x input.

There are different types of prediction tasks, such as: classification, regression and clustering [9]. Classification is a way of classifying different objects/inputs based on some labels assigned to a few other inputs. Starting from a collection of objects with some labels already assigned, the predictor in this classification task divides the space into two (or more) parts, defining an imaginary line splitting these two parts, called decision boundary. That line is the function that predicts the results, based on which side the input falls. The most important thing about the classification is that the decision boundary depends on the labels. If the same set of inputs is taken, with different labels, a distinct decision boundary will be specified. Figure 2.3 shows an example of classification, where only some "M" and "F" labels are assigned as initialization, and then the algorithm assigns the others, whether they fall in the M or F side of the decision boundary [10].

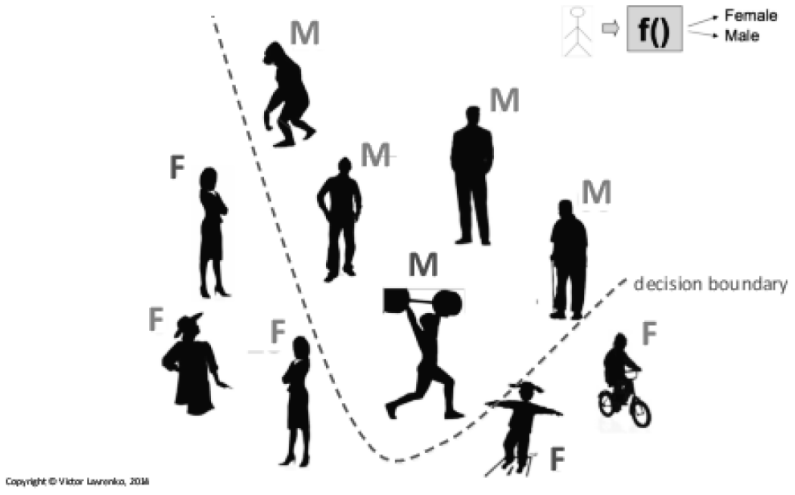


Figure 2.3: Example of classification task, taken from [10]

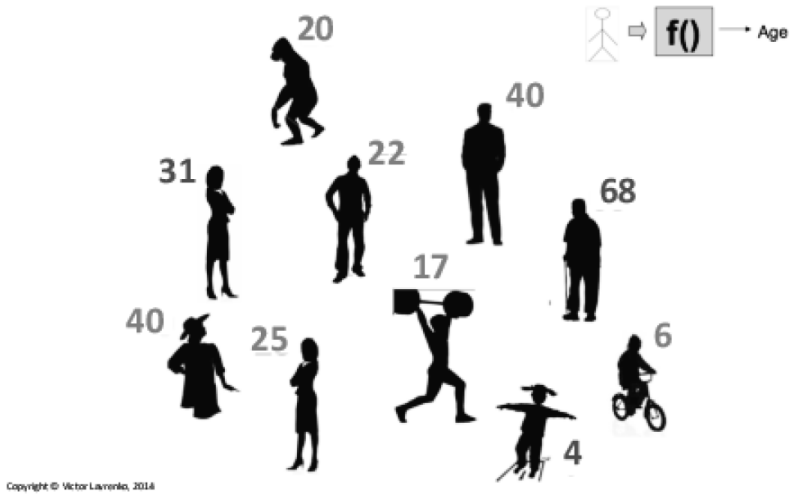


Figure 2.4: Example of regression task, taken from [10]

Regression is a way of predicting outputs based on different objects with numbers assigned to some of them. The prediction needs to assign numbers to the unassigned objects, by learning with the numbers already assigned. Figure 2.4 shows an example of regression, where only some numbers were assigned initially, and then the algorithm assigns the others based on the given ones.

Both of the methods, classification and regression, are methods of the so called supervised learning because something very specific (such as examples of desired results) is given, that allows the algorithm to learn.

On the other hand, Clustering is an unsupervised learning task, since no examples are given, only the data set. The goal is to discover subpopulations inside the population. Clustering is very useful to detect objects that are not usual but for some reason are inside the population. By dividing the population into subgroups, if any object is an outlier, it will stand alone, without association to any group. This can be seen in Figure 2.5 that shows a clustering example with an outlier.

Although there are several algorithms for machine learning, for instance support vector machines, k-nearest neighbors, etc., this document and the next subsection will focus on Decision Trees and Random Forests since this work is based on trees' concepts.

2.2.2 Decision Trees

Decision tree learning is one of the most widely used and practical predictive models. These trees can also be represented by several if-else structures. Since the decision trees (DT) are included in the supervised learning, they can have two main types: classification trees or regression trees. There are many decision-tree algorithms, but the most important ones are ID3, C4.5 and CART.

The concept of CART, which stands for Classification and Regression Trees, was introduced by Breiman to refer to both of the trees, classification and regression [11].

Decision trees allow the classification of examples by sorting them down the tree, from the root to a leaf node which provides the final classification [12]. Each node represents a test of one or more attributes of the target function, and each branch descending from that node is one of the

Figure 2.5: Example of clustering task, taken from [10]

possible values for the attribute. Therefore, the classification of an instance starts with the testing of the attribute at the root node of the tree and then moving down with the possible values for that attribute [8].

The tests represented by the nodes, can be written, generally, as a function f of n attributes:

$$f(A_1, A_2, \ldots, A_n) = 0 \tag{2.1}$$

The nodes are continuously split until a pure set is reached. A pure set is a set of attributes in which the final result of its branches is the same. For example, Figure 2.6 shows a simple decision tree to decide whether a person plays tennis, or not, according to the weather. Looking at the figure, one can see that if the weather is "Overcast", that person will definitely play, so this represents a pure set. On the other hand, if the weather is "Sunny", the result depends on the Humidity, so "Sunny" is not a pure set. However, "Sunny" and "Humidity Normal" is already a pure set.

In order to know which attribute should be split, the "purity" of the split should be evaluated. So, the goal is to have more certain answers (Yes or No in the example presented) after the split. However, it is not possible to use probabilities to measure this, even with conditional probability (probability of getting a certain value, knowing one subset already), because it has to be symmetric, which means that a subset full of "Yes" answers is as pure as a subset full of "No" answers. Hence, there is a way to measure the uncertainty of a class in a subset of examples, called Entropy, given by:

$$Entropy(S) = -p\log_2 p - n\log_2 n \tag{2.2}$$

where p is the percentage of positive examples in the subset S, and n is the percentage of negative examples.

Equation 2.2 should be interpreted in terms of a "fractional" number of bits. So, the main

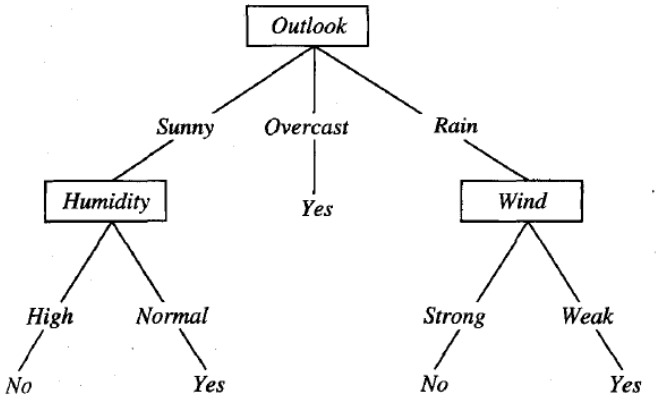


Figure 2.6: Decision Tree for the decision of playing tennis from [8]

question is, how many bits are necessary to tell if a certain item that belongs to the subset is positive or negative? If the subset is pure, the number of bits needed is zero. On the other hand, if the subset is 100 % uncertain (number of positives equal to the number of negatives) the number of bits should be one, since it is either positive or negative. Finally, for cases with less certainty than a pure set, but higher certainty than a 100 % uncertain subset, the result of the entropy will be a number between zero and one, and that is why this can be interpreted in a "fractional" number of bits.

Figure 2.7 shows the function of entropy as the proportion of positive examples (represented by p_+ in the x-axis) varies between 0 and 1. Equation 2.2 stands only for the case when the classification is boolean. If the attribute can take n different values, then the entropy would be computed as:

$$Entropy(S) = \sum_{i=1}^{n} -p_i \log_2 p_i \tag{2.3}$$

where p_i is the proportion of the subset S that belongs to the class i.

Since the entropy tells how pure a subset is, then it is necessary to aggregate the information from various subsets. In the weather example there are three different subsets, each one with its own purity value. Thus, the information gain of an attribute A relative to a collection of examples S is given by [8]:

$$Gain(S,A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v) \tag{2.4}$$

where S_v is the subset of S where the attribute A has the value v, and *Values(A)* is the set of all possible values for the attribute A. So, the value of *Gain(S,A)* is the expected drop in entropy after split, which in other words means the number of bits that it is possible to gain, if the attribute A is chosen as the attribute to split on. Therefore, to have the most certainty possible, the attribute with highest gain should be selected as the attribute to split.

There are also some techniques that allows the construction of more than one tree:

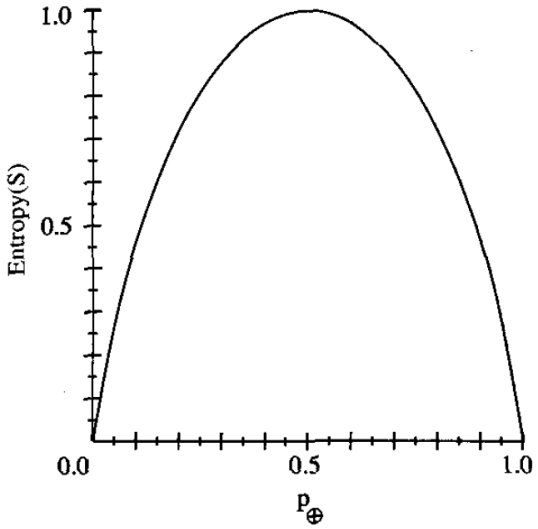


Figure 2.7: Entropy function varying the proportion of positive examples from 0 to 1 from [8]

- **Bagging Trees** take bootstrap samples of examples and trains a classifier on each sample, and the classifier votes are combined by majority voting; [13]
- **Random Forests**, which uses a defined number of decision trees, in order to improve the classification rate; [14]
- **Boosted Trees** can be used for both classification and regression types and the idea is to boost the performance of a "weak" classifier by using it within an ensemble structure; [15]
- **Rotation Forest** consists in splitting the feature set into N subsets, apply principal component analysis (PCA) on each subset and then reassembling a new extracte feature set. [16]

2.2.2.1 Hardware Implementation

A straightforward approach for an hardware implementation of a decision tree can be the implementation of each node as an independent module as proposed in [17]. Despite being a simple implementation, it has the disadvantage of waiting until a classification of an instance is completed, before applying a new instance.

Based on the equivalences between DTs and threshold networks, Bermak and Martinez proposed a faster implementation, since the propagation of the signals only goes through two levels, independently of the number of levels of the decision tree [18].

Since these two realizations proposed above require a considerable number of hardware resources, for instance, the number of node modules to be realized needs to be equal to the number of nodes in the decision tree, Struharik tried to find some architectures that besides the reduction of the hardware complexity, could also achieve high throughput [19]. In this approach, to classify

an instance, only one node per level needs to be evaluated. Hence, the number of modules for the realization is equal to the depth of the tree, instead of the total number of nodes of the decision tree. This way, it is enough to have only one universal node module per each level, and that is why this architecture is called Single Module per Level (SMpL). Thus, this architecture consists of N pipeline stages, being N the depth of the decision tree being realized. Also, it is possible to reduce the computational time by doing a parallelization in the evaluation of the test in each node. With this parallelization, the throughput of the SMpL architecture is one instance per clock cycle, instead of one instance per $n+1$ clock cycles, if the evaluation is sequential. The author called this architecture Single Module per Level Parallel (SMpL-P). This SMpL-P architecture increases a bit the hardware complexity. On the other hand, the author defined another architecture called Universal Node (UN) for when the classification speed is not a problem, being even more possible the reduction of the complexity of the hardware, by assigning a single universal node that can evaluate the whole tree (by evaluating each tree node). This architecture consists of only one programmable node where one of the modules is the control unit for the perfect operation of the system. It is necessary not only to load the attribute values of the next instance to classify, but also to classify the current instance. As the Single Module per Level, it is also possible to do a parallelization in the evaluation test in the universal node, reducing the time of computation from $n+1$ clock cycles to one clock cycle, which increases the classification speed by a factor of $n+1$. This architecture was called Universal Node Parallel (UN-P).

2.2.3 Random Forests

A random decision forest is an ensemble of randomly trained decision trees. Its model is characterized by a specific number of components.

The accuracy of the classification was significantly improved by this growing of an ensemble of trees and letting them vote for the most popular class. In order to train each tree, it is necessary to generate random vectors that can supervise the growth of each one of them. One example, proposed in 1996 by Breiman, is bagging, where examples are chosen, randomly, from the training set. [20]

Later, in 1998, Dietterich proposed the random split selection [21], where at each node the split is selected randomly among the K best splits. In 1990, Breiman randomized the outputs in the original training set to generate new training sets. Also, another different approach is the selection of the training set from a random set of weights on the examples in the training set.

In 1997, Amit and Geman defined a number of geometric features and search in a random set of these for the best split at each node [22]. In 1998, Ho described the random subspace method that selects randomly a subset of features that is used for the growth of each tree. [23]

2.3 Attacks and Defense Mechanisms - The current State of the Art

2.3.1 JTAG Based Attacks

As already referred in this document, the JTAG is very useful in some environments, but in other more hostile environments it can lead to undesirable forms of exposure.

It is possible for a system to have multiple potential attackers, but some of the attacker's capabilities might be blocked by defenses.

Kurt Rosenfeld and Ramesh Karri from the Polytechnic Institute of New York University described briefly five JTAG-based attacks [24] such as sniff secret data, read-out secret, obtain test vectors and responses, modify state of authentic part or even to return false responses to test. In these different attacks, the attacker possesses a limited set of capabilities, for example the capability of modify the TDI/TDO signals, or control the TMS and TCK signals or even access the keys used by the testers.

The sniffing attack has the goal of learning a secret message that is being sent to a specific chip via JTAG. This attack consists in a false bypass done by the attacker, to let the message reach the pre-determined chip without any modification. The key point to perform this attack is to capture a copy of the message while it is being sent to the chip.

The read-out secret attack has the goal of learning a secret message from a device that already has it. In this attack, two attackers' chips are necessary, both in the same JTAG chain of the chip under attack: one placed "before" the victim (attacker A) and the other one placed "after" it (attacker B). To perform this attack correctly, the attacker A needs to control the TMS and TCK lines, in order to act as the JTAG bus master and to try to do a scan operation to access the secrets of the chip that is being attacked. Then, the attacker B will acquire the information as it is expelled from the TDO of the chip.

The attack to obtain test vectors and responses, as the name indicates, is for the attacker to obtain a copy of the test vectors and normal responses of a chip. In order to do this properly, the tester sends test vectors to the device, which responds normally to these test vectors. The attacker, placed after the victim, acquires the responses as they propagate from the device under test back to the tester, which receives unmodified test responses.

The modify state of authentic part attack is used to modify the state of the chip that is being attacked. The attacker chip is placed before the device under test/attack and it puts the JTAG TAP of the chip into a state where it is able to transfer data into the chip. Hence, it is possible for the attacker to set the state of the registers of the victim's chip and affect the normal operation of the device.

Finally, the return false responses to test attack has the goal of misleading the tester about the real state of the chip being attacked. In order to test the state of a specific chip, the tester tries to put all the others into BYPASS mode, but the attacker ignores this request and it introduces fake responses that are transmitted back to the tester, putting all the other chips into BYPASS mode.

2.3.2 Scan-based side-channel attacks

The scan chains are connected to an external JTAG interface and scan-based attacks can be used as a side channel to retrieve secret keys by analyzing the scan data obtained from the scan chains.

2.3.2.1 Scan-based attacks against DES

The Data Encryption Standard (DES) is one of the common-key-encryption methods, developed by IBM. It is a symmetric algorithm which encrypts 64-bit data controlled by a 56-bit secret key. Eight bits are only used for parity check [25]. There are 16 stages of processing, all of them identical, and also an initial and final permutation, where each one undoes the action of the other one, and none of them has any cryptographic meaning. A scan-based attack against Data Encryption Standard has two major problems: the encryption timing, because attackers do not know when the encryption is done, and the register connection order since they do not know the correspondence between scan data and registers inside the scan chain.

In 2004, Yang, Wu and Karri proposed a mechanism to recover secret keys from a hardware implementation of the Data Encryption Standard [26]. This method assumes the attackers knows the DES algorithm and also can input any plaintexts into the target DES cryptosystem. Besides this, it does not include registers capable of storing the secret and round keys. To solve the problem of the encryption timing, Yang, Wu and Karri assumed that the attackers can obtain scan data at any timing and to solve the register connection order problem. They also assumed that the content of the plaintexts introduced into the cryptosystem differ one bit between each other and the scan chain is composed only by the Round Register, Input Register and Output Register, with no other registers included in the chain.

Later, Kodera, Yanagisawa and Togawa improved the mechanism proposed by Yang et al., using scan signatures [27]. They claim that the assumptions made to solve the problems of the encryption timing and the register connection order are impossible to be applied to real and practical cases. Hence, they improved the mechanism by assuming that the scan chain includes at least a 32-bit Round Register and also that the attackers may obtain scan data over several cycles. Furthermore, they were interested in ensure that a scan signature was generated, focusing only on a particular 1-bit column of comparison data. This means that the attacker does not need to know the correspondence between scan data and registers of the chain. This scan signature indicates a particular 1-bit register change when the plaintexts are inputted into the DES cryptosystem. Thus, one of the most important features of this method is the number of plaintexts required to retrieve secret keys correctly.

2.3.2.2 Scan-based attacks against AES

Advanced Encryption Standard (AES) is a 128-bit symmetric key block cipher that can have three different lengths for its key: 128, 192 or 256 bits. A major difference comparing with its predecessor DES, is that AES does not use a Feistel Network, which is a symmetric structure used in the construction of the block ciphers that allows the operations of encryption and decryption to

be very similar, reversing the key schedule [28]. AES is divided into several identical cycles that convert the input (plaintext) into the final output (ciphertext). The number of cycles varies with the length of the key: for 128-bit keys the algorithm has 10 rounds, 12 rounds for 192-bit keys and finally, 14 cycles for 256-bit keys [29].

There are several proposed attacks against AES and all of them operate in two steps: First, the chip is in normal mode with its input plaintext applied to the primary outputs for only one cycle of AES; then, the response of the round operation that is being stored in the scan chains is shifted out. The attack is done using one of the differential properties of the round operation of the AES, which tells that if one-bit difference is given in the least significant bit (LSB) of the 16 bytes, the output difference can have 18 possible hamming distances. From these hamming distances, four can be generated by a unique pair of S-box inputs. Thus, the attacker should follow these two steps for all possible pairs of plaintexts and tries to get any one such hamming distances [30].

One of the existing attacks is the architecture proposed by Yang, Wu and Karri that maintains the high test quality of the traditional scan DFT without compromising the security [31]. To show that it is possible to compromise the secret key with the traditional DFT scheme, the authors used an hardware implementation of the advanced encryption standard. All of the attacks can even work in the presence of DFT structures such as partial scan [32], X-masking [33] and X-tolerant architectures [34, 35].

Ali, Sinanoglu, Saeed and Karri found that all of the attacks described above have one common disadvantage: their incapability of switching the device between the normal and the test mode, preserving the data in the scan cells. This happens because all these attacks assume a perfect access to functional inputs. Hence, they proposed a scan-based attack that only uses the test mode [30]. Thus, the proposed method only requires applying the plaintext from the scan-in pins rather than the functional inputs, which makes the attack more complex because the attacker is blinded from the input side and from the output side. The challenging part is then the identification of the scan cell that corresponds to the desired input bit. To do this, the attack is split in four different steps: the first one is to determine the scan cells that correspond to the words; the second is to determine the scan cells that correspond to the bytes; the third one determines the order of the bytes in a word and, finally, the fourth step determines the order of the scan cells in a byte and thus the corresponding key.

2.3.3 Reverse Engineering Techniques

Reverse engineering consists in extracting knowledge from anything man-made. It is normally used to obtain missing knowledge or ideas when the information is not available or belongs to someone that does not want to share. Usually the idea is to discover some secrets about a design, in order to do a better one [36].

Domke published in 2009 a paper showing that the test modes of the JTAG can be reverse-engineered by looking at the JTAG inputs and outputs [37]. To do this it is necessary to find the JTAG pins. If no documentation is available for that specific IC, there are some tools that can be used for this part. After finding the pins, it is necessary to determine the length of the instruction

register, which means determining the number of accessible registers. Finding the length of the IR is quite simple, after scan in several ones in order to fill the whole register, a single zero is scanned in. Therefore, the number of clock cycles that is necessary to wait until the zero reaches the TDO, is the length of the register.

It is important to be aware of some implementation specific behaviors that need to be determined, for example the possibility of having a scratch register that is transferred into the final register when a certain control bit is toggled, or for example the possibility of having an additional bit in the DR, to determine if the register is going to be updated or not.

2.3.4 Defense Mechanisms

2.3.4.1 Basic protocols

The simplest way to block malicious use of JTAG port is to restrict its functionality. This approach is limited to certain situations where JTAG becomes unnecessary after initial testing. With this approach a portion of the functionalities can be denied by modifying the JTAG port or by removing part of the input/output using fuses [38].

2.3.4.2 Two-entity security protocol

Other defense methods for integrated circuits and JTAG, are based on user authentication or test data encryption.

Novak and Biasizzo proposed a security extension for IEEE Std. 1149.1 [39] which provides a locking mechanism that prevents unauthorized users from interfering, via test bus, with the normal operation of the system. It is basically an access control solution in which a key must be presented to the chip before the chip's JTAG test access port can be controlled. The instruction set of the TAP includes two additional instructions: lock and unlock. In the lock instruction, the TAP control logic maps all the instructions, except the unlock, to a bypass instruction until the unlock instruction with valid key code is applied, which secures the device operation. The structure of the locking mechanism is presented in Figure 2.8.

As one can see in Figure 2.8, the comparator block compares the contents of the Lock Register and the Key Register. If the contents are different, the locked signal fed to the Instruction Decoder is activated, effectively locking the access to the system. To unlock, it compares the contents again, and if they are equal, it deactivates the locked signal and the next instruction entered via TDI can be executed [39]. This security mechanism proposed by Novak and Biasizzo requires a small hardware overhead but it does not slow down the conventional boundary-scan tests.

Another method is to permit the use of JTAG port to all users, but encrypting or encoding the input/output data using a secret key as described by Rosenfeld and Karri [24]. They employed three standard security primitives in their scheme: a hash function, a stream cipher and a message authentication code. Instead of using a PUF (Physically Unclonable Function) as Suh and Devadas used [40], they used fuses and a hash, but the idea is similar. In Rosenfeld and Karri's work, the uniqueness comes from fuse bits that are programmed at the factory while in the work proposed by

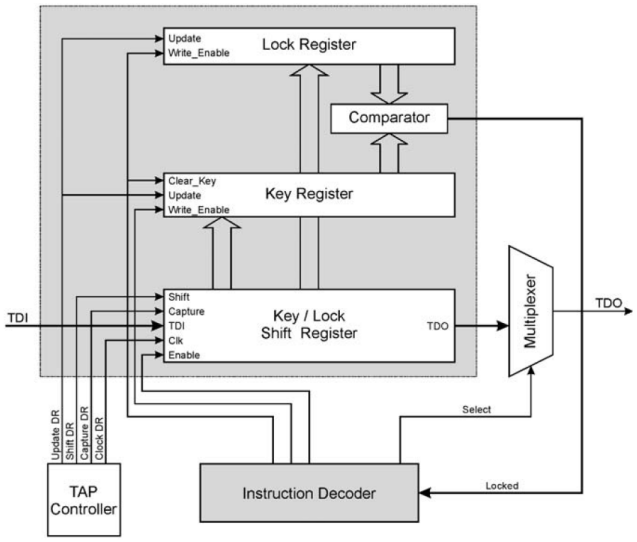


Figure 2.8: Structure of the locking mechanism proposed in [39]

Suh and Devadas, the uniqueness comes from subtle physical differences between chips. Both of these works were extremely useful in what concerns JTAG defenses: on one hand, PUFs have the advantage of being intrinsically unique which means that they do not need to be programmed with the desired uniqueness; on the other hand, fuses have the advantages of using less area and also being a reliable technology against temperature, aging and power variations. As already said, they also used a stream cipher for encrypting JTAG communication and a message authentication code scheme to protect against unauthentic JTAG messages, which in other words means the verification that the message was sent by the authentic sender without being modified. The security scheme proposed by Rosenfeld and Karri has four levels of protection that are going to be described with detail:

- **Level 0**: it does not assure anything. The communication is exactly the same as what is defined in IEEE 1149.1;
- **Level 1**: it is assured authenticity, in which the JTAG operations start after the tester extracts from storage a code read protection (CRP) randomly, then send a challenge to the chip and finally compare the chip's response with the CRP;
- **Level 2**: besides the authenticity provided by the level 1, this level also provides secrecy of the JTAG signals, which involves sending and receiving encrypted data to and from the data register. It has two steps: setup and communication. In the first one, the goal is to establish a shared secret between the tester and the chip;
- **Level 3**: this level is the last one in this security scheme and completes level 1 and 2 by adding integrity, protecting the JTAG link against active attacks that attempt to insert or modify messages.

The disadvantage of this approach is the delay introduced by encryption or encoding processes. These key-based security methods are truly dependent on the key management, because all security features do not work if the key is disclosed. There could be several issues regarding this key management and distribution mechanisms. For instance, delegating this management to human causes insecurity; on other hand, if all devices shared the same key, the risk is even bigger; the opposite, which would be assigning a unique key for each device leads to a cost problem.

2.3.4.3 Three-entity security protocol

In order to avoid the problems of distribution described in the two-entity security protocols, and also ensure more security and efficiency, the scan-chains or the keys used for authentication must be managed by a trusted and independent identity.

Pierce and Tragoudas presented the first detailed hardware implementation of a flexible multilevel security access system [41, 42]. In this mechanism, each scan chain is monitored by an independent and separate security mechanism, which can restrict the types of data that can be loaded into each scan chain and thus, it is possible to have different access levels per scan chain.

The architecture uses a hierarchical multi-level permission structure, where a user has an associated privilege level P_i and groups of instructions have an access level A_i. Only a user with a privilege level $P_i > A_i$, can execute those instructions. Since this is a flexible system, the number of permission levels is set by the designer of the IC.

As can be seen in the Figure 2.9, the proposed hardware consists in two components: the Security Authentication Module (SAM) and the Access Monitor (AM). The SAM provides an unlocking communication protocol, sets the user level and passes the privilege level to the Access Monitor, which changes the accessible features of each scan chain and thus prevents harmful data from being load into them.

This mechanism, besides the unlocking protocol at the beginning of a JTAG session, does not introduce any additional timing overhead and its security strength is only dependent on the initial authentication protocol, because the only component that can modify the memory in the Access Monitor is the Security Authentication Module.

Buskey and Frosik proposed a solution that introduces different levels of access, based on user's permissions [43]. This mechanism controls the protection level of the device limiting the interaction through the JTAG port. They implemented three protection levels, PL0, PL1 and PL2 and four access modes: AM0, AM1, AM2 and AM3.

Hence, the most secure protection level, PL0, limits user access through the JTAG port to external functions; the following level, PL1, besides allowing the same functionalities of the previous one, also allows the programming of the flash memory and the writing and reading to registers. This protection level is similar to the one provided by the standard JTAG; the last protection level, PL2, allows full access to the device, including the secure registers or secure sections of memory.

The four access modes were created according to the normal life cycle of the products, that require different access levels. This way, there is a non protected access mode (AM3), needed for example during the product's development phase, in which the access to the device through JTAG

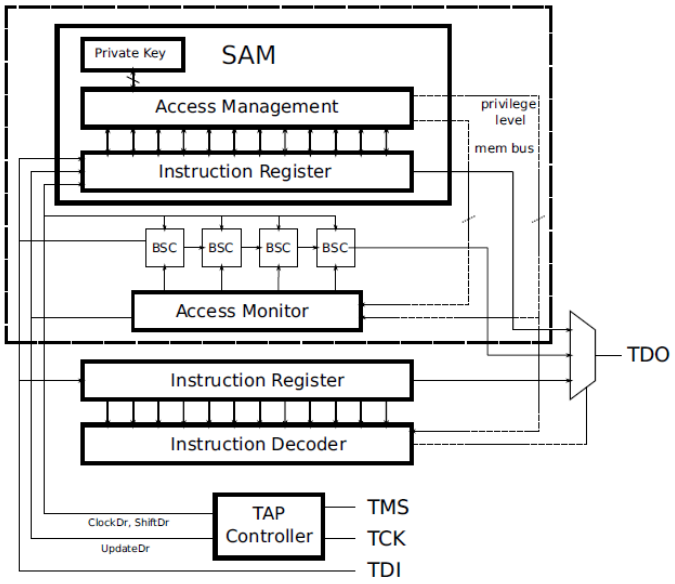


Figure 2.9: JTAG Block Diagram of the hardware proposed in [41] and [42]

port is unrestricted. For this access level, the device should be configured with the protection level PL2 described previously. The next level, in which the protection mode should be set to PL1, or sometimes to PL2 (only available to authorized users), is the low protection access mode (AM2). A device configured with this low protection access mode allows standard debugging functions, writing to flash memory, and viewing or modifying the processor's registers and ROM. Increasing the security, the high protection access mode (AM1) provides only the circuitry debugging functions and the protection level should be set to PL0, but it can be temporarily reduced to PL1 or PL2 by an authenticated user. Finally, the last access mode is the maximum protection access mode (AM0) and is intended for the products that contain very sensitive information. The protection level is, therefore, set to PL0 and downgrading is not supported.

Regarding the user authorization, this solution requires a secure database containing the credentials. This database is checked during an authorization process, and its functions are delegated to another trusted identity, a secure server. The authorization process is based on asymmetric cryptography, in which the device owns a public key and the secure server holds the private key.

A mechanism which complements some of the weaknesses found in the two-entity protocols and in this previous three-entity protocol from Frosik and Buskey and also enhances its strengths, is a JTAG security mechanism proposed by Park, Yoo, Kim and Kim which consists in a system based on credentials [7].

Their approach has a lower implementation cost than encryption/decryption-based solutions because its authentication protocol only uses hash and XOR calculations, instead of complex asymmetric cryptography like the one presented before. Also, it eliminates the possibility of key leaks because, instead of secret keys, their method authenticates and allows access to users using credentials and passwords issued by a remote server. Another improvement of this mechanism compared with the Frosik and Buskey's method, is the availability. Once the credentials are

issued, the host does not need further communications with the authentication servers.

The authentication scheme proposed by Park et al. grants also different access levels to users, based on the permission of the credential. This scheme consists in two phases: the credential issue phase and the user authentication phase. The first one is the stage when the user receives the credential from the server. The server verifies the identity and the correspondent access level and, if the user is authenticated, it provides the credential. The second phase is for the authentication of a user that already has a valid credential for the Secure JTAG, as the authors called it. In this phase, the JTAG allows access to all features defined in the credential and authenticates the user based on the submitted credential and password. The main advantages provided by this mechanism are the impossibility to use a valid credential without its password and, even having knowledge of the password, the user cannot request other access levels.

Besides the normal control logic present in all JTAG devices, this Secure JTAG has also an authentication logic, controlled by the TAP controller and the instruction decoder of the control logic. This authentication logic consists in a random number generator, a hash engine and a non-volatile memory, and is responsible for the execution of all authentication steps.

Finally, in this credential-based mechanism, is also possible to assign a limit to the number of times a credential can be used and also share a group credential, instead of assigning one credential for each developer, with the appropriate configuration, when several developers are testing a single device.

In summary, some of the security mechanisms proposed in this chapter are extremely safe and accurate. However, as previously introduced, it is possible to learn the patterns that correspond to illegitimate actions and therefore define a system based on machine learning. This learning approach consists of the main difference of the proposed system when compared with others in the literature. Furthermore, to the best of the author's knowledge, this is the first security system to use this kind of approach. Combining this system with some other mechanism can result in an extremely high security level, as it will be shown in section 3.3, where a comparison between the different methods is done.

Chapter 3

Description of the System and Training of the Trees

This chapter is divided into two important parts. Sections 3.1, 3.2 and 3.3 consist in a detailed description of the system, its architecture, its main features and all the modules it has. Section 3.4 presents some important aspects related to the random forest, such as the training and searching for the optimal number of trees.

3.1 System Overview

As already mentioned, this work is based on the security scheme proposed in [2], which monitors in real time the behavior of JTAG using an on-chip machine learning approach, while protecting backdoors from misusing and consequently detecting illegal accesses to the chip.

In order to be able to perform attacks, the attackers need to behave differently than the normal users, because they need to discover and characterize the undocumented functions without being aware of the implemented functions and how they work. Thereby, it is possible to have an on-chip classifier that decides the legitimacy of the access.

The authors propose an on-chip classifier based on a single decision tree; instead, in this work the on-chip classifier is implemented with a random forest, an ensemble of decision trees.

To detect the attacks, the method uses a scheme containing two steps: offline learning and online prediction. In the first step, it is necessary to train the decision trees of the forest by extracting features from JTAG programs and to store them in a memory. The extraction of the features is assisted with a look-up table (LUT) that contains information related to the transitions between instructions. Then, the second step, the online prediction stage, consists in supplying the forest with the same set of extracted features. This allows for the possibility of making predictions about whether an instruction is normal or suspicious. In order to improve the accuracy of the next predictions, the LUT containing the transitions between instructions is modified at the end of this process.

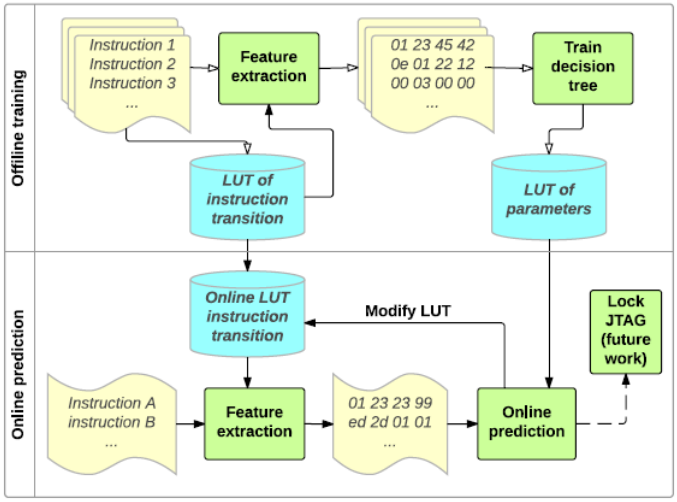


Figure 3.1: Two phases of the mechanism proposed in [2]

To illustrate the detection method, figure 3.1 shows, in detail, both of these two steps: offline learning and online prediction.

To characterize the difference of behavior between normal use and illegitimate use, the authors propose to use eight features, seven of them related to intra-instruction statistics and the last one to the inter-instruction transition:

1. Four most significant bits (MSB) of the instruction
2. Four least significant bits (LSB) of the instruction
3. Number of shift Data Register cycles
4. Number of cycles in Run-Test/Idle (RTI) state
5. Number of cycles in Test-Logic-Reset (TLR) state
6. Number of test-mode select (TMS) transitions
7. Instruction defined
8. Transition miss

The first and second features represent jointly the 8-bit opcode of the instruction. The third one is related to the number of the Data Register (DR) shift cycles, because without prior knowledge of the length of the DR, it may be hard to be able to shift the register the number of desired cycles. The fourth feature defines the number of cycles in Run-Test/Idle (RTI) state and the fifth feature defines the number of cycles in the Test-Logic-Reset (TLR) state. Feature number six consists in the number of transitions of the TMS, and feature seven checks whether the opcode loaded into the instruction register is valid or not. Finally, feature eight, checks the legitimacy of the transition from one instruction to another, based on the information read from the LUT. All these features

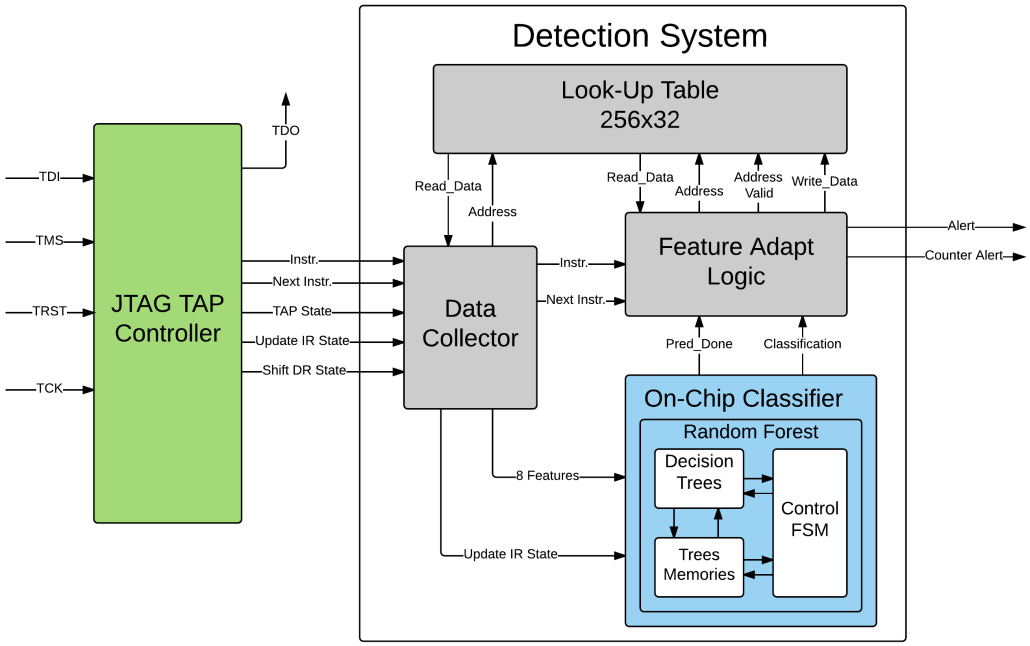


Figure 3.2: System Architecture

are captured when the Instruction Register (IR) is updated with a new opcode and are then fed into the forest.

3.2 System Architecture

The architecture of the system is shown in Figure 3.2.

The detection system is composed by four main modules: an on-chip classifier, a look-up table, a data collector and a feature adapt logic. The classifier is represented in blue since it is the main focus of this work. The figure also shows the JTAG TAP Controller which will be integrated and implemented in the FPGA together with the entire system.

3.2.1 On-chip Classifier

The on-chip classifier is based on a random forest, it includes the module of the decision trees, a memory that stores the structure of all the trees and a control Finite State Machine (FSM) to manage the predictions between the trees. The control FSM includes a Majority Vote Unit that makes the final prediction based on the predictions of all trees. As stated before, the on-chip classifier is capable of making predictions whether an instruction is legitimate or not. It receives the eight features that represent the current instruction from the Data Collector and then it starts evaluating them using the forest. In addition, the classifier also receives an Update Instruction Register signal which is active when the JTAG TAP Controller is in the Update IR State and thus

is responsible for the starting of the classifications. Its operation will be detailed in Chapter 4, since it is implemented with two different approaches.

3.2.2 Data Collector

The Data Collector is responsible for the interaction with the JTAG, extracting the features. Although it is not shown in the figure, it receives the JTAG input bits, TDI and TMS, the current instruction, the next instruction, the TAP Controller state, the update Instruction Register signal, and the Shift Data Register signal, which is active when the TAP Controller is in that same state. These last five inputs are sent by the TAP Controller. The Data Collector transforms the input bits and instructions into features that are then supplied to the on-chip classifier. These features are updated when the update Instruction Register signal is triggered.

3.2.3 Look-up Table

The Look-up Table (LUT) stores information about the transitions between instructions. It interacts with the data collector and with the feature adapt logic. Hence, it has a top module to control this interaction with both modules, for instance, choosing the address and enabling the reading or writing. More details and how the information is stored will be explained in Chapter 4.

3.2.4 Feature Adapt Logic

Finally, the proposed mechanism also has a feature adapt logic that modifies the LUT of the transitions between instructions and sets the final outputs. It delays the labeling of the JTAG operation until it gathers a significant evidence. Specifically, it waits for 4 consecutive predictions before taking any action. It receives the current and next instructions from the data collector, the final classification and a flag that indicates this classification is complete, from the classifier, and the data read from the LUT. The output called Address Valid, is a 1-bit signal that helps the LUT defining the address, as will be further explained.

3.2.5 JTAG TAP Controller

The JTAG TAP Controller represented in green in Figure 3.2 is a module of the OpenSPARC T2 Processor from *Sun Microsystems* [3], which is an open-source processor design and has been used in many projects. It simulates the behavior of the normal TAP Controller from a JTAG device. The JTAG TAP Controller is basically the 16-state FSM represented in Figure 2.2, and it converts the JTAG input bits into instructions that are supplied to the data collector and to the feature adapt logic. It also sends to the data collector the current state of the TAP Controller and two one-bit signals (Update Instruction Register and Shift Data Register) that are set to 1 when the TAP Controller is these respective states.

3.3 Comparison of the system with the current state of the art

Table 3.1, extracted from [2] compares some of the protection schemes with the proposed mechanism. As shown in the table, the proposed mechanism only works for the undocumented backdoors because the attacker can not be aware of the correct operations of the JTAG. Each scheme has its own disadvantages and risks. For instance, in a password-based mechanism, if the password is leaked, the security is immediately compromised. In a protocol-based method, the failure of the network is undoubtedly a potential risk. Therefore, combining the proposed mechanism with one of the already described, for instance, the password-based, can be an excellent option, because it makes the attacks harder. In that case, it would be necessary to know how to operate the JTAG correctly and also the password in order to perform an attack successfully.

It is important to note that the proposed algorithm has a higher security level, compared with passwords protocols or on-chip compression or compaction.

Table 3.1: Comparison between protection schemes, from [2]

JTAG Protection Scheme	**Protected Target**	**Knowledge of JTAG Operation**	**Hardware Overhead**	**Potential Risks**	**Security Level**
Disable JTAG	All functions	Maybe	Small	Invasive attack	High
On-chip compression/ comparison	Boundary Scan	Maybe	Medium	Differential attack	Low
Password-based authentication	All functions	Maybe	Medium	Password leakage	Medium
Protocol-based authentication	All functions	Maybe	Large	Eavesdropping, Network fails	High
Proposed Learning System	Undocumented backdoors	No	Medium	Attacker with previous knowledge	High

3.4 Random Forest Training

As referred in Section 2.2.3, there are several algorithms to train the trees of a random forest. In this work, bagging or bootstrap aggregation was used, and the training was performed using MATLAB.

Bootstrap aggregation consists in generating independent training sets from the original set and training each tree using different training sets. Hence, each tree of the forest grows on an independently replica of the original training data. Observations that are not included in the replica

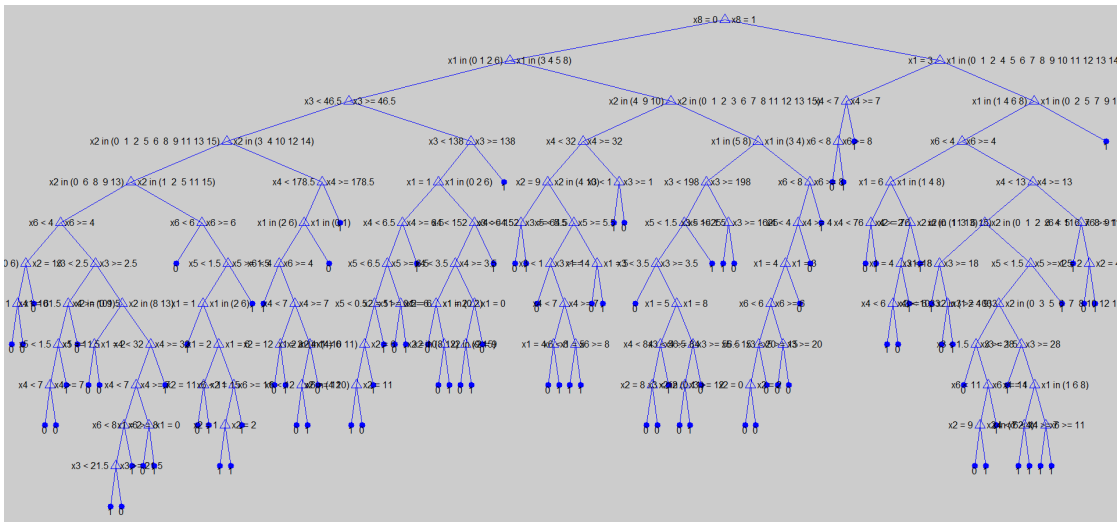


Figure 3.3: Example of a tree of the forest classifier

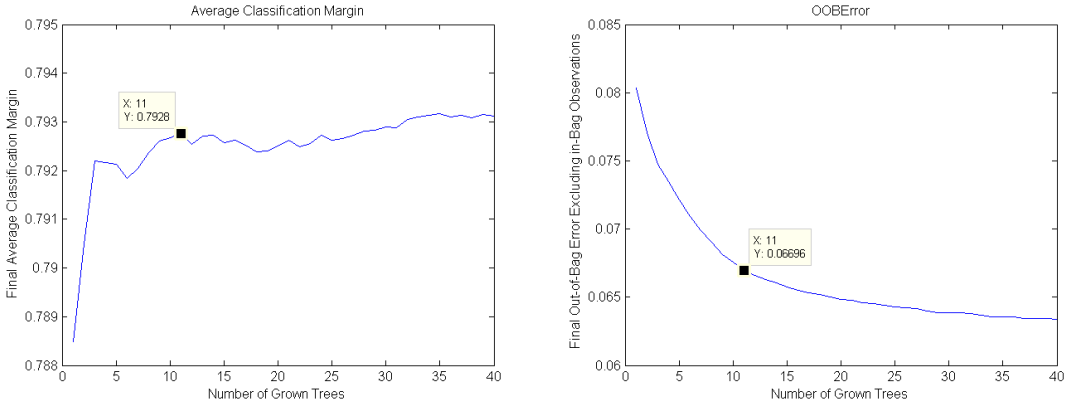
are called out-of-bag, and can be used to perform the out-of-bag error which evaluates the performance of the random forest.

During the training phase, a training set was provided by the authors [2]. The data set has 14124 observations: 7674 for training and 6450 for testing.

The first step of the training is to define if the features are categorical or numerical. Categorical features are discrete ones that can just take one specific value from a number of fixed values. On contrary, numerical features are continuous and can take any value within an interval. Hence, the split values in each node of the trees, to decide whether the evaluation should go left or right, are different based on the features. For the categorical features, the node checks if the feature is included in a set of values while for the numerical features the node has a threshold and it evaluates if the value of the feature is above or below the threshold, further determining to split to right or left, respectively.

Figure 3.3 shows an example of a tree trained by the data set provided. It is important to note that the categorical features are the features number 1, 2, 7 and 8. As one can see, in this example the first node evaluates the feature 8 that checks if the next instruction is defined in the LUT or not. Other trees of the forest may start by evaluating other features, and not the feature number 8 as this one.

To train the trees, the class *TreeBagger* in MATLAB was used. It includes the constructor function and several methods to add trees, estimate errors, margins, etc. The function creates an ensemble of bagged decision trees after receiving the train data, the train label and the number of the trees. Since the trees need to be stored on chip, the number of trees in a forest leads to a trade-off with memory space. To solve this problem, different numbers, from 2 to 40, were tried to search for the optimal number of trees. In order to be more accurate and precise, the ensembles were generated 100 times, averaging the out-of-bag error and the classification margin over these 100 executions. The out-of-bag error computes the misclassification probability for out-of-bag observations in the training data. The classification margin is the difference between



(a) Average Classification Margin (b) Out-of-bag Error

Figure 3.4: Training results averaged over 100 executions

the classification score for the true class and maximal classification score for the false classes. It is an important metric of representation of the success of the training, illustrating the benefit of adding more trees to the ensemble.

Figure 3.4 shows the results of the training of the trees. Figure 3.4a represents the classification margin while figure 3.4b shows the out-of-bag error in terms of number of trees. The point marked in the figure indicates the optimal number of trees chosen for the forest. As one can see, the forest comprises 11 trees. This number gives a good classification margin and the error is also very satisfactory. Indeed, with more trees, for instance 35 or 40, both perform slightly better, which however requires much more memory on chip.

It is important to note that the out-of-bag error plot is consistent with the results obtained by the authors. Their classifier was simulated with a single decision tree with an error of about 8.17 % which is very close to the result obtained in the second plot, with just one grown tree. Thereby, with an 11-tree forest, the error is reduced from 8.17 % to approximately 6.70 %, which consequently means an increase in the accuracy from 91.83 % to 93.30 % (Table 3.2).

Table 3.2: Comparison between a single decision tree and an 11-tree random forest

Implementation	**Error**
Decision Tree	8.17 %
11-Tree Forest	6.70 %

Once the random forest is trained, the description of the trees in text format is generated in MATLAB, and saved as .txt files. This file contains as many lines as nodes in each tree and each line/node follows the syntax represented below, where *x1* is the feature under evaluation:

if x1 in {0 1 2 6} then node 4 elseif x1 in {3 4 5 8} then node 5 else 0

These .txt files are then converted to .hex files, in order to be stored in memory. The structure of these .hex files will be detailed in chapter 4.

Chapter 4

Hardware Implementation of the Modules

This chapter is related with the hardware implementation of all the modules of the system. It describes how they are designed, their operation and synthesis results, specially concerning area and time. However, it starts with a brief presentation of the board that was used.

4.1 The Field-Programmable Gate Array (FPGA)

This section gives a brief introductory description of the FPGA used in this work, which is present in the *Zynq-7000 ZC706 Evaluation Kit*, from *Xilinx*.

The board, shown in Figure 4.1 taken from the Xilinx website, is extremely powerful and contains numerous tools and components, for instance a DDR3 component memory of 1 Gigabyte and a DDR3 SODIMM memory of 1 Gigabyte as well. It also has several different ports, connectors and controllers that allow different types of communication, such as USB, JTAG, Ethernet, PCI Express, HDMI, SD Card, I^2C, SPI and UART. The FPGA chip present on the board is the *Zynq-7000 All Programmable SoC Z-7045* which is composed by a Dual *ARM Cortex-A9* core processor. Since the System-On-Chip (SoC) allows the creation and customization of co-processors that interact with the ARM processor, this complete system will be implemented as a co-processor.

There are two different ways of designing an implementation for the *Zynq-7000 ZC706* board. The first is using a bare-metal application which runs directly on the SoC. In this option, the programmable logic is programmed via JTAG which requires the reset of the board and makes harder the load of data into the system. The second is to install a Linux based operating system (OS) in the SD Card, which can be done by loading the binaries of the kernel and the bootloader that are provided by *Xilinx*. The board can be accessed via Ethernet using a secure shell (SSH) connection and the bitstream generated on *Vivado* is loaded into the board. This option has the advantage of allowing the configuration without the need of a reset, while the operating system is running.

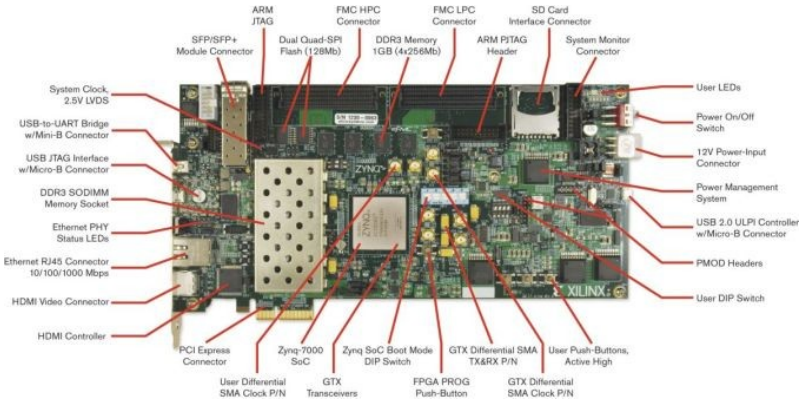


Figure 4.1: Xilinx Zynq-7000 ZC706 Evaluation Kit

The number of configurable logic blocks (CLB) of the FPGA is impressive. It contains 437200 flip-flops (FF), 218600 look-up tables (LUT), 70400 Memory LUTs, 32 global buffers for clock lines (BUFG), 8 Mixed-Mode Clock Managers (MMCM), between many others. BUFG are global buffers that are most commonly used for clock nets to provide the least possible amount of skew between registers that are physically located large distances apart.

Further details about the communication between the processing system and the system's co-processor will be given in Chapter 5. The next sections describe the implemented hardware, which was simulated and synthesized using *Vivado Design Suite*, also from *Xilinx*.

4.2 The Classifier

The implementation of the classifier followed two different approaches: one sequential and one parallel. Both of them were implemented in the FPGA and are described in this section along with a comparison between the two.

4.2.1 Storage of Decision Trees Structure

As referred in section 3.4 the .txt files containing the trees are converted to .hex files through a C program, that evaluates each line and each value of the text. These .hex files are then stored in memories, where each tree occupies 512 positions of 40 bits. Categorical nodes are stored from the position 256 to 511 while numerical nodes are in the positions 0 - 255. As also stated, the categorical nodes need to compare a set of discrete values with the current value of the feature, therefore each categorical node is represented by two lines of the .hex file. Hence, with the 512 positions, it is possible to store 384 nodes: 256 numerical nodes and 128 categorical nodes. The structure of each position of the memory is represented in Figure 4.2.

As one can see, the first line is common to either categorical or numerical nodes. The bits 0 to 23 store the addresses of each one of the son nodes of the current node. Since the tree has 512 positions, only 9 bits are necessary to represent an address, therefore 3 bits in each address are not being used. However, 12 bits are reserved for each address for two reasons: the first is because

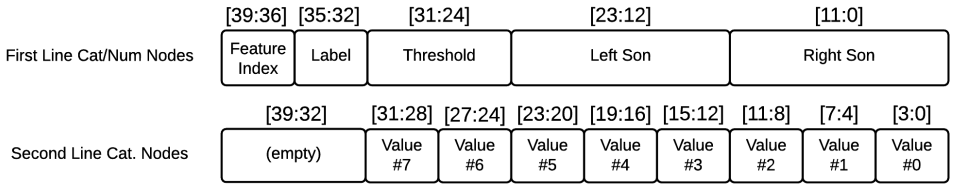


Figure 4.2: Structure of positions of memory

the C program that converts the trees from .txt to .hex writes in hexadecimal format and thus, it might be only possible to write either 8 bits (which are not enough) or 12; the second reason is the possibility of increasing the size of the trees, for example by training them with a different data set. Then, it is necessary 8 bits to store the threshold value for the numerical nodes (bits 24 to 31) and 2 bits to store the labels, although 4 are defined. Label[0] (bit 32) represents the prediction, where 0 means normal action and 1 means illegitimate action, and label[1] (bit 33) represents whether the current node is a leaf node or not. If it is a leaf node, this bit is 1 and the label[0] is valid. On the other hand, if it is not a leaf node, the value of the label[0] is not valid, since only leaf nodes have the final classifications. The last 4 bits are the index of the feature that is being evaluated in that node.

The second line is only for the categorical nodes and it stores 8 discrete values to compare with the current feature. If the value of the current feature is within those 8 values, the tree goes to the left son; if it is not, it follows the son to the right. The discrete values are 4 bits each, since the features number 1 and 2, that are categorical, have also 4 bits each. Features 7 and 8 are also categorical, however they are composed by just 1 bit.

It is important to note that the threshold bits of the first line are also used in the categorical nodes to define the discrete values of the second line that are valid. The value of the categorical feature will only be compared with valid discrete values. For instance, if the threshold of a categorical node is 8'b00101100, only the discrete values number 2, 3 and 5 (positions where the bit is 1) are valid and will be compared with the current feature.

The storage of the trees is slightly different in the two implementations of the classifier, that will be described in this section. In the sequential implementation, the trees are stored all together, in a memory of 5632 positions, which corresponds to *512 positions* $\times$ *11 trees*. On the other hand, in the parallel implementation, the trees are stored in 11 separate memories, of 512 positions each. Both of these memories are asynchronous, which means the clock is not necessary. The data from the addressed location is available on the output bus after the access time. Hence, these memories are created from LUT cells of the board and therefore they are called Distributed Memories, which as the term indicates, are memories distributed throughout the chip inside the logic blocks.

The resources used by these two memories are shown in the table 4.1, however it is important to note that the smaller memories are replicated 11 times in the design. A better comparison between the two implementations will be given in section 4.2.5.

The number of Memory LUT used is exactly the same, since 320 $\times$ 11 = 3 520, however the

Table 4.1: Trees' Memory Resources Usage

Resource	Utilization		Available	Utilization (%)	
	All-trees	1-Tree		All-trees	1-Tree
LUT	3 822	362	218 600	1.75	0.17
Memory LUT	3 520	320	70 400	5.00	0.45
BUFG	1	1	32	3.12	3.12

number of LUTs is slightly different. Multiplying the number of LUTs of the smaller memory by the 11 memories, $362 \times 11 = 3982$, it is possible to see an increase of 160 LUT compared with the memory that has all the trees together.

4.2.2 The Decision Tree

The decision tree module is almost the same in both implementations of the classifier, either sequential or parallel. It is implemented using the universal node proposed in [19] where the nodes in the current path are evaluated sequentially, from the root node to the leaf node of the decision tree.

The inputs of the module are the 8 input features described in section 3.1 as well as the clock, reset and prediction start. The prediction start is the update instruction register signal that comes from the JTAG and passes through the Data Collector, which means the JTAG TAP Controller is in the Update IR state and a new classification should be started. The outputs are just two bits, one to flag when the final classification is complete, and the other with the classification itself, where 0 represents a normal action and 1 represents an illegal action. In the sequential implementation, this module has one extra input that represents the number of the current tree to evaluate.

The decision tree module has two simple finite state machines (FSM) of only two states each. The first one controls when the module is active or idle. When the update instruction register signal is active, meaning that the module should start a new classification, the machine passes from the state idle to the state active, starting the prediction. When the prediction is complete, the tree goes back to the idle state, to wait for the next update instruction register signal.

The second FSM controls the reading from the memory that stores the tree, which is only done in the active state of the FSM described in the last paragraph. As mentioned, the categorical nodes occupy two lines of 40 bits each. Therefore, two lines are always read and this FSM controls the reading of these lines, ensuring that the first line is read before moving the address to the second line. These two lines are assigned to three temporary 40-bit variables, called *current_categorical_data*, *current_numerical_data* and *current_categorical_candidates*. The first line that was read is assigned to the first two variables, while the second line is assigned to the *current_categorical_candidates*, since it represents the discrete values to compare with the current value, when the node is categorical. Then, the division of the bits to different variables is done in the *current_categorical_data* and *current_numerical_data*, as shown in Figure 4.2.

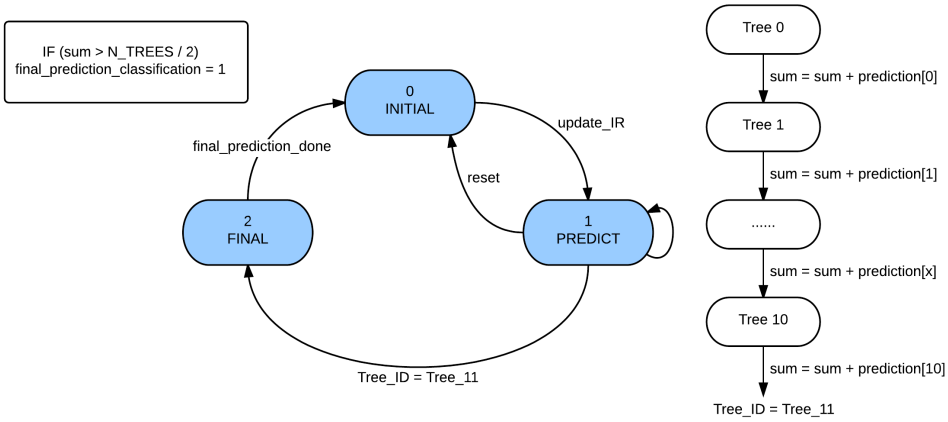


Figure 4.3: Operation of the Sequential Classifier

The last step is just the decision of going left or right in the tree. First, the current feature needs to be defined by analyzing bits 36-39 that represent the Feature Index. According to the index, the value of one of the eight features is assigned to a variable called *current_node_feature*. Then, if the node is numerical, it is only necessary to do a comparison between that *current_node_feature* and the current threshold. If the current feature is smaller than the threshold, then the next node should be the left son. Otherwise, the next node is on the right path. In contrast, if the node is categorical, the four LSB of the current feature (since the categorical features do not have more than 4 bits) are compared with the discrete values of the second line, that are present in the variable *current_categorical_candidates*. As mentioned in the previous section, the threshold bits of the first line are also used in this part, to define the discrete values that are valid. Only the valid values are compared with the 4-LSB of the *current_node_feature*. If the current feature is equal to one of the valid discrete values, the following node is the left son. On the contrary, if it is not within the values, the tree should take the right path.

Finally, the bit 33 (Label[1]) is analyzed. If it is one, it means a leaf node was reached and the final classification is defined, along with the flag indicating it is complete. If it is not one, the address is updated with the correct son node address, and the evaluation of the nodes continues.

4.2.3 The Sequential Classifier

The sequential classifier does the predictions one at each time, from the tree number 0 to the tree number 10. As previously mentioned, in this implementation, all the trees are stored together. Hence, an extra variable is used, representing the number of the tree (Tree_ID) that is being evaluated, in order to load the right node from the memory. This variable will be used as input for the decision tree module, as described in the previous section. The inputs of the sequential classifier module are exactly the same as the decision tree module, while the outputs represent the final classification instead of the classification of just one tree. The predictions of the 11 trees are controlled by an FSM, illustrated in figure 4.3, with 3 states: initial, predict and final.

The initial state waits until a new update instruction register signal is active. When this signal is triggered, the classifier goes to the state predict where it starts the prediction of the tree number 0. When the system detects the rising edge of the flag that indicates the prediction of the tree is complete, the number of the tree is incremented and its prediction (0 or 1) is accumulated in a variable *sum*. The prediction start signal of the decision tree module is set to 1 again, and the next tree starts predicting. When the number of trees reaches the last tree, the FSM goes to the final state that represents the majority vote unit, where the value of the variable *sum* is compared with the number of trees over 2. If it is bigger, then it means that the majority of the predictions were 1 and the final classification output is set to 1. In contrast, if it is smaller, it means a normal action and the final classification is 0. Also, this state sets to 1 the flag that indicates the final classification is complete.

4.2.4 The Parallel Classifier

On the opposite, in the parallel version, the universal tree node is replicated 11 times, each one with its own memory of 512 positions, in order to allow 11 accesses at the same time. There is no need to use a variable that represents the number of the current tree. However, two vectors are created, with 11 positions each: one to store the predictions of all the trees (named *class*) and the other to store the flags of prediction complete (named *done*). The operation of the parallel classifier is similar to the sequential version, since it is also controlled by an FSM with 3 states: initial, trees and final.

Likewise the sequential, the initial state waits until a new update instruction register signal is active. Once it is active, the FSM goes to the state trees, where all the 11 decision trees start their classifications at the same time. Hence, when a tree has its prediction complete, it stores it in the corresponding position of the vector *class* setting also the flag that indicates it is complete, in the vector *done*. While in this state, the variable sum is constantly being updated with the sum of the 11 positions of the vector *class*. Once the vector done is totally full with ones, meaning that all the trees completed their prediction, the system moves to the final state. This state defines the final classification and the global flag exactly in the same way as in the sequential implementation.

The operation of the parallel classifier can be better visualized in the figure 4.4.

4.2.5 Comparison of both Sequential and Parallel

Before the integration with the other modules of the system and with the JTAG, that will be described in Chapter 5, the two versions of the classifier were implemented in the FPGA. To test the classifiers, several sets of eight input features were applied, representing both normal and illegitimate actions. The way how the inputs are applied and how the output results are shown in the terminal of the computer will be explained later, with the complete implementation, since the principle is the same.

The results of these implementations concerning area and time are presented in this section.

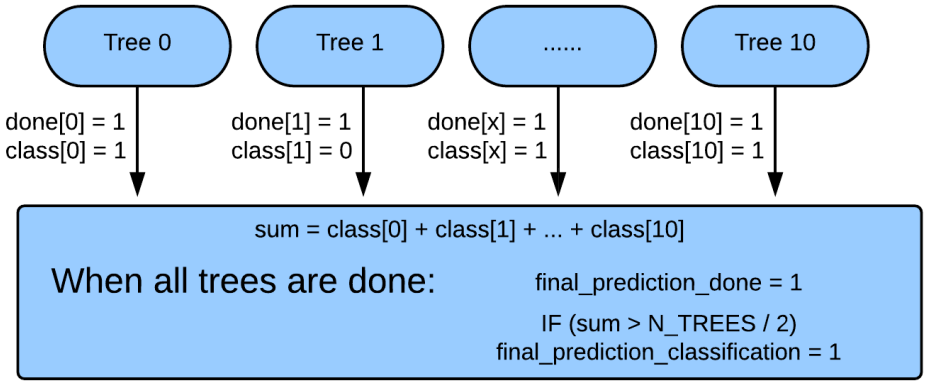


Figure 4.4: Operation of the Paralell Classifier

4.2.5.1 Resources Usage

The resources usage of both implementations is shown in Table 4.2. These results include the entire interface to communicate between the processing system of the FPGA and the programmable logic, that will be explained later as referred.

Table 4.2: Sequential and Parallel Classifiers Resources Usage

Resource	**Utilization**		**Available**	**Utilization (%)**	
	Sequential	Parallel		Sequential	Parallel
Flip-Flops	840	1 292	437 200	0.19	0.30
LUT	4 243	4 945	218 600	1.94	2.26
Memory LUT	3 408	3 144	70 400	4.84	4.47
BUFG	1	1	32	3.12	3.12

As one can see, the parallel implementation is slightly bigger than the sequential one, which was expected since the parallel implementation contains 11 universal nodes instead of just one. The difference in the number of Memory LUTs might be less expected than the other resources, since the stored data (the structure of the trees) is the same in both implementations. As it is possible to see in the resources usage of the trees memory in Table 4.1, the number of memory LUTs were the same when the memory blocks were synthesized individually, without any system or module integrated. However, the memory LUTs consist of blocks of a variable number of bits with a specific width and depth. The way how the memories are instantiated in the top modules and in the system determines the optimizations that the software does to synthesize them. The memory LUT blocks are generally instantiated with depths on the order of a power of two since the synthesizer allocates these memories to the blocks of physical memories that exist in the board. Since the parallel implementation uses 11 memories of 512 positions, which is a power of two, it is easier to allocate them. Finally, it is possible to conclude that both of the implementations are extremely small compared with the entire board.

4.2.5.2 Timing Performance

The major difference between the implementations is the timing performance, as would be expected. To compare both, as mentioned, several sets of input features were applied to the classifiers and the number of clock cycles needed to compute the final classification was determined. Table 4.3 shows the number of clock cycles with two specific sets of features, set 0 and set 1. Set 0 represents an illegitimate action while set 1 consists in a normal action.

Table 4.3: Number of clock cycles to classify two specific sets of input features

Implementation	Set 0	Set 1
Sequential	187 cycles	215 cycles
Parallel	22 cycles	28 cycles

Table 4.4 shows the time needed to classify the same two sets of Table 4.3. This time is calculated with a clock frequency of 50 MHz (period of 20 *ns*), which is the value that was used since it is the frequency of the interface between the processing system and the coprocessors of the FPGA, that will be further explained.

Table 4.4: Time spent to classify two specific sets of input features

Implementation	Set 0	Set 1
Sequential	3 740 *ns*	4 300 *ns*
Parallel	440 *ns*	560 *ns*

As it is easily seen, the parallel implementation performs much faster than the sequential version. The increase may vary between 7 and 11 times, which would happen if all the trees took the same time for the classification. But since the trees are independently trained and different in number of nodes and depth (number of levels), it would not be correct to affirm that the time is always reduced 11 times.

4.3 The Look-up Table (LUT)

The Look-up Table is a memory with 256 positions of 32 bits each. Each position is accessed through an 8-bit address that consists in the opcode of a JTAG instruction, public or private, defined in the OpenSPARC T2 and it contains 4 opcodes of 8 bits each, which represent 4 possible instructions that may be executed after the current instruction. This module is used by the Data Collector to assist in the extraction of features, as will be detailed in the next section.

If all the 4 opcodes are 8'b11111111 it means that the instruction is undefined, which is also extremely important for the extraction of the features.

Besides the Data Collector, the LUT can also be accessed by the Feature Adapt Logic. The Data Collector can only read from the LUT, to help in the extraction of features as mentioned,

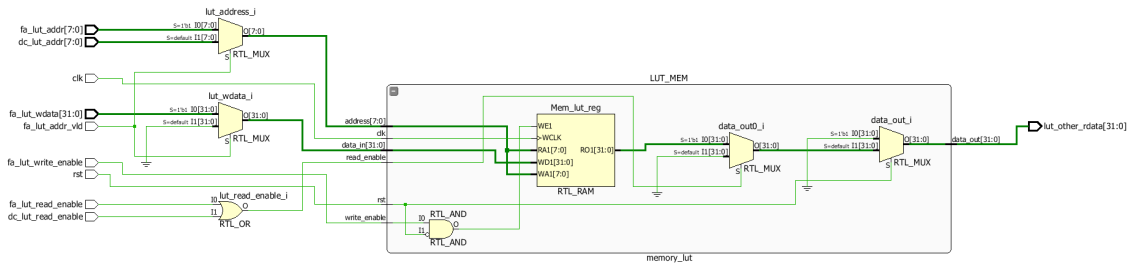


Figure 4.5: RTL Schematic of the LUT

while the Feature Adapt Logic can read or write, since it can modify the LUT based on the predictions of the classifier. This will be further detailed in the section 4.5. Both the reading and the writing mechanisms are controlled by their correspondent enable.

The memory of 256x32 is placed inside a top module that interacts with the other modules, in order to define the address to read or write to/from the LUT, since the address can be defined in the data collector or in the feature adapt logic. To decide which one should be used, the top module of the LUT also has an extra input bit, called *address_valid* that comes from the feature adapt logic. This bit controls a multiplexer that decides the address. If this bit is 1, then the address should be defined as the address sent by the feature adapt logic. If it is 0, then the address used should be the one from the data collector. Also, the LUT top module decides when the read enable is active, based on the read enables from the data collector and from the feature adapt logic. The RTL Schematic of the LUT is shown in Figure 4.5. The memory itself is within the block called *LUTMEM* while the other two multiplexers and the OR gate are in the top module, to define the address, enables and data to write, as described.

The Look-up Table was synthesized individually and the resources usage is shown in Table 4.5.

Table 4.5: Look-up Table Resources Usage

Resource	Utilization	Available	Utilization (%)
LUT	175	218 600	0.08
Memory LUT	128	70 400	0.18
BUFG	1	32	3.12

4.4 The Data Collector

The Data Collector converts the current instruction into features which are then supplied to the on-chip classifier.

Features 1 and 2 consists in reporting the instruction itself, that it receives from the TAP controller dividing it into 4 most significant bits and least significant bits, respectively.

Feature 3 counts the number of shift cycles of the data register. The data collector module has 1-bit input that indicates when the TAP Controller is in the shift data register state. Every time the value of this bit is 1, the data collector will add 1 to the counter. This counting is done between two consecutive update instruction register signals. If a new one comes, the counter goes back to zero and starts again.

To define the feature 4, the data collector also analyzes the current state of the TAP controller with the TAP controller state input, referred in section 3.2.2. Every time the TAP Controller is in the Run-Test/Idle (RTI) state, a counter is incremented until a new update IR signal arrives.

Feature 5 is also very similar to features 3 and 4 since it consists in counting the number of cycles in the Test-Logic-Reset state, using the same input as feature 4, which represents the current state of the TAP controller.

Feature 6 counts the number of transitions of the test-mode select (TMS). As mentioned, the data collector module receives the TDI and TMS. The previous and the current TMS are stored and compared. When the current is different than the previous, then it means a flip occurred, and the value of the feature 6 is incremented.

Finally, features 7 and 8 are slightly different as they require an interaction with the LUT. Feature 7 consists in evaluating if the current instruction is defined. To read from the LUT, the address, which is 8-bit long, is defined as the current instruction and the read enable is assigned every time the update IR signal is 1. As referred in section 4.3, each position of the LUT has 32 bits, which represent four instructions of 8 bits each. If in the position of the current instruction the next four possible instructions are all 8'hFF, then it means the current instruction is undefined and feature 7 is set to 1. Feature 8 is quite similar in the sense it also evaluates the content of the LUT in the position of the current instruction. However, the data collector module also receives the 8 bits of the next instruction. A transition miss is reported, and consequently feature 8 is 1, when the next instruction is different from all the four instructions defined in the LUT.

The data collector was synthesized individually and the results are shown in the Table 4.6. One can see that the module is extremely small. In fact, the RTL Schematic is shown in Figure 4.6. It is not possible to see all the components in detail, but it is sufficient to show that this module is composed mainly by comparators, AND gates and multiplexers. The registers on the right store the eight features that are then forward to the classifier.

Table 4.6: Data Collector Resources Usage

Resource	**Utilization**	**Available**	**Utilization (%)**
Flip-Flops	85	437 200	0.02
LUT	58	218 600	0.03
BUFG	1	32	3.12

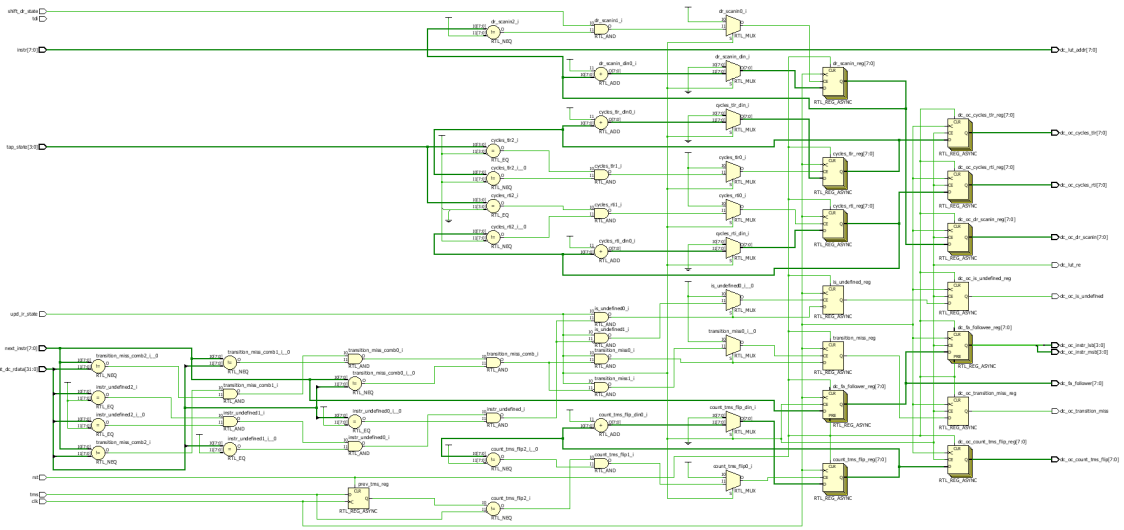


Figure 4.6: RTL Schematic of the Data Collector

4.5 The Feature Adapt Logic

The Feature Adapt Logic or feature revision mechanism was proposed in the initial paper due to the variance that naturally occurs within both legitimate and illegitimate uses of the JTAG. Thus, this module delays the labeling of the JTAG operations until it gathers ample information, as will be described.

The inputs of the feature adapt module, besides the clock and reset, are the current instruction and the next instruction, that come from the data collector, the classification and the flag that indicates the classification is complete from the random forest classifier and the data that is read from the LUT. The outputs are the address, read and write enables, data to write and the extra bit referred in the section 4.3 that makes the address valid. All of these are for the interaction with the LUT, since this module can modify the LUT based on the predictions. However, the most important outputs are an alert flag and a counter that determines the number of alerts a specific set of instructions had. These will be also outputs of the complete system, as will be described in Chapter 5.

The feature adapt module is responsible for two main actions: trigger the security alert and adapt the LUT. Since the alert is triggered every 4 instructions, the design has a period counter and a flag that is assigned to 1 from 4 to 4 instructions. It is also necessary a counter for the number of illegitimate predictions that are accumulated in that period of 4 instructions. There are also two important variables that must be explained, called remove request and insert request. The remove request is one if the number of illegitimate predictions is equal or bigger than a high threshold that is defined, while the insert request is one if there are less or equal illegitimate predictions than a low threshold. These threshold were defined as 3 and 1, respectively for the high and low. This means that if the classifier sent 3 or 4 illegitimate predictions in the same period of 4

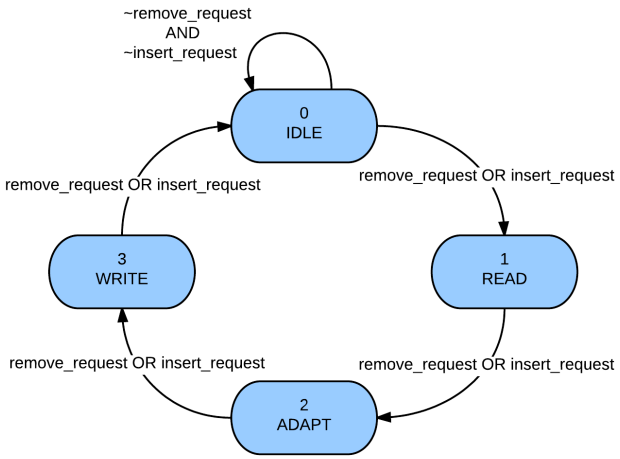


Figure 4.7: Control FSM of the Feature Adapt Logic module

instructions, the remove request is set to one. Following the same thought, if the predictions sent by the classifier are all legal or just one illegitimate, the insert request is set to one.

Hence, the security alert is defined based on the value of these two variables of removing and inserting requests. If the remove request is one, then the alert is set to 1 as well and if the insert request is one, the alert is 0, since no illegal actions are taking place. It is important to reinforce the idea that this alert signal is updated every four instructions, so during the first 3 following instructions, it keeps the value that resulted from the previous 4 instructions.

The second feature of this module is to adapt the LUT as referred. This is controlled by a finite state machine (FSM) with 4 states, represented in the Figure 4.7. The states are controlled only by the variables insert and remove request.

As it is possible to see in the figure, the system is always in the idle state, until either the remove request or the insert request is active. When one of them is active, the system goes through all the other 3 states (read, adapt and write with this order), and stays for just one clock cycle in each one. In order to work properly, the insert and remove requests need to last for a long period, but since they are only updated from 4 to 4 instructions, this situation is perfectly controlled. In the read state, as the name indicates, the feature adapt module just reads the content (32 bits) from the LUT, in the position of the memory correspondent to the current instruction. The adapt state creates the 32-bit data to write in the LUT in the following state. The first step is to determine which variable (insert or remove) is set to 1. If the remove request is one, then it means that a specific opcode will be removed. The opcode that needs to be removed is the opcode of the next instruction, which is an input of the feature adapt module, sent by the data collector. After determining which byte of the read data from the LUT is equal to the opcode of the next instruction, that same byte is replaced by zeros. The adapt state defines the content of the 32 bits that are going to be written in the LUT, and the write state enables the write and performs the replacement. On the other hand, if the insert request is one instead of the remove request, then the new opcode of the next instruction

will be added to the LUT, if it is still not there.

The Feature Adapt Logic module was also synthesized individually and the resources it uses are shown in the Table 4.7.

Table 4.7: Feature Adapt Logic Resources Usage

Resource	**Utilization**	**Available**	**Utilization (%)**
Flip-Flops	173	437 200	0.04
LUT	124	218 600	0.06
BUFG	2	32	6.25

Chapter 5

System Integration and Overall Results

This chapter presents the integration of the final system, with the JTAG TAP Controller included. It characterizes how the system was implemented in the FPGA, describing the interface between the processing system and the programmable logic as well as the experimental setup and communication with the computer. Finally, it concludes with the results of the complete system followed by a brief discussion.

5.1 JTAG Synthesis

The final step before the integration of the complete system was the synthesis of the JTAG TAP Controller. As previously mentioned, the OpenSPARC T2 was used to describe the typical JTAG behavior.

The JTAG TAP Controller module consists of a 16-state finite state machine, shown in Figure 2.2. It contains signals that represent specific states, for instance, the update instruction register and the shift data register. Both of these signals are sent to the detection system, through the Data Collector. Furthermore, the TAP Controller has the responsibility of converting the JTAG input bits (TDI, TMS and TRST) into instructions that are also provided to the Data Collector. Moreover, it generates the TDO, which is not relevant for the purpose of this work.

Likewise the other modules, the TAP Controller was synthesized individually and the resources usage is shown in Table 5.1.

Table 5.1: JTAG TAP Controller Resources Usage

Resource	**Utilization**	**Available**	**Utilization (%)**
Flip-Flops	22	437 200	0.01
LUT	21	218 600	0.01
BUFG	1	32	3.12

One can see that the module is extremely small, when compared with the other modules and with the complete board. Indeed, the block is composed only by some registers and standard D flip-flops.

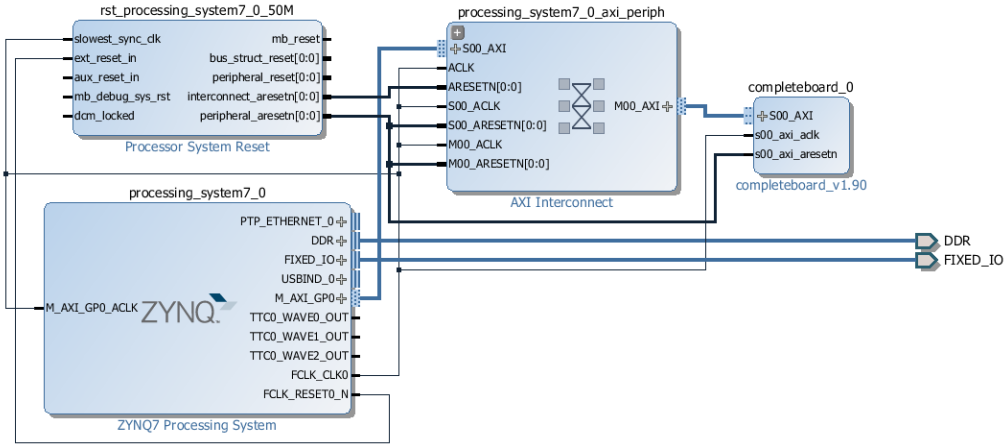


Figure 5.1: Communication System Architecture

5.2 Interface and Communication System

As already pointed out in section 4.1, the processor of the FPGA runs Linux and it is necessary to establish the communication with the system, which is implemented as a co-processor.

The protocol used to connect the processor with the co-processors was the AMBA AXI, introduced by ARM, whose Intellectual Property (IP) is included in *Vivado Design Suite*. AMBA, which stands for Advanced Microcontroller Bus Architecture, is an on-chip interconnect specification focused on the connection and management of functional blocks inside a System-On-Chip (SoC). There are 5 AMBA protocol specifications and the AXI (Advanced eXtensible Interface), which is defined in the third specification, targets high performance and is widely used in ARM Cortex-A processors.

Figure 5.1 shows the architecture of the communication system, composed by four blocks. The bottom left block represents the ARM processor, which runs with a frequency of 666 MHz, while the top left block contains the reset distribution logic, which is needed to propagate the signal through all the blocks. The module in the middle of the architecture is the AXI Interconnect responsible by connecting the registers of the processor with the registers of the slave core. The frequency of the AXI Interconnect is 50 MHz. Finally, the block on the right, called *completeboard* is the implemented system, which includes the entire detection system and the JTAG TAP Controller. In order to be able to communicate with the processor through the AXI interface, after the HDL implementation of the modules, the system was packaged as an AXI slave core, whose interface is a set of user-defined 32-bit registers. Accordingly, it was necessary to ensure that the inputs could be read from those registers and the final outputs written in the same registers. These reading and writing mechanisms will be explained in the next section.

5.3 Experimental Setup and Synchronization

As mentioned before, the inputs and the outputs need to be read and written in the slave registers. Since the registers are addressable on the processor side, it was developed a C program responsible for this interaction with the registers. In fact, two different C programs were elaborated: one for the implementation of the sequential and parallel classifiers and another for the final implementation of the complete system.

Nevertheless, it is important to note that the Linux of the ARM processor is not a real time operating system. As a result, the synchronization between the processor and the system's slave core needs to be done using flags.

5.3.1 Classifier Implementation

The AXI Interface for the FPGA implementation of the classifier uses three registers of the set of user-defined 32-bit registers mentioned in section 5.2.

From these three registers, two are used by the inputs of the classifier, which correspond to the 8 features (that represent a total of 42 bits), the Update Instruction Register signal that is used to start a classification and an extra flag that was created to address the mentioned synchronization problem, called *read_done*, which indicates that the final outputs were successfully read.

The other register is used for the final outputs, which are the classification and the flag that indicates the classification is complete. A new output was defined in other to compare both implementations of the classifier, the sequential and parallel. This output counts the number of clock cycles that a specific classification took.

Hence, the C program developed for interaction with the slave registers for the implementation of the classifier has a pointer to the base address of the first register. From this first register, the other registers are accessible with some pointer arithmetic and the 8 features are written in the correspondent positions. Then, the Update Instruction Register signal is set to one and the program enters in a loop until it reads a 1 in the flag, indicating that the classification is completed. Afterwards, it just reads the value of the final classification and the number of clock cycles and prints them on the screen. Finally, the C program sends the acknowledge of the outputs and the classifier waits for new features to classify.

5.3.2 System Implementation

Likewise in the implementation of the classifier, the AXI Interface for the FPGA implementation of the complete system also uses three registers of the set of user-defined 32-bit registers. Nevertheless, it includes other components that are going to be described.

However, in this implementation two of the registers are used for the outputs and just one for the inputs. The inputs are the reset, the *start* and the *finish*. These last two are special flags needed for the synchronization. The reset is either the reset from the AXI Interface or a reset that is sent by the C program. The outputs are the final alert, the counter of the alerts and two extra flags for

synchronization. One is called *got_reset* and it acknowledges the reception of the reset sent by the C program while the other is called *out* and indicates the end of the set of instructions under evaluation. In fact, the system has more inputs and outputs, that are going to be described in the next section, but the ones mentioned in this section are the inputs and outputs used in the AXI Interface.

Hence, the C program starts by sending a reset signal, and waits for the response of the system through the *got_reset* flag. Then, the *start* is set to one and the system begins the classification of the entire set of instructions, until the *out* flag indicates the set is over. Afterwards, the *finish* is set to one and the value of the counter of the alerts is read. The execution time is also determined in the C program and shown in the end.

One component present in the interface is a Mixed-Mode Clock Managers (MMCM) that works as a clock multiplier and divider. The AXI clock was used as the base clock to avoid clock skew between the two or more clocks. The clock frequencies generated by this MMCM will be presented along with the complete system results.

5.4 Complete System Results

After the individual synthesis of each module, they were placed all together within a top module composed by two sub-modules, one for the JTAG TAP Controller and another as a top module for the whole detection system, which includes the classifier, the LUT, the feature adapt logic and the data collector.

The classifier that was included in the final system was undoubtedly the parallel classifier due to its much better performance concerning time. The sequential classifier was a first approach which did not take into account the timing constraints of the entire system and consequently the frequency of the Update Instruction Register signal, which represents a new classification should be started.

In order to test properly the implementation, several sets of instructions, normal and illegitimate, were provided by the authors. The instructions were initially in files .instr and two Python programs were used in order to convert them to testbenches (.v) and to .bit files. The testbenches were used in *Vivado Design Suite* to simulate the system while the .bit files were created for the FPGA implementation. There are 3 .bit files per instruction, one for each input (TDI, TMS and TRST). These 3 inputs were stored in 3 external memories, one for each input, and were applied to the JTAG TAP Controller module. Hence, the resources usage of the final implementation depends on the size of the set of instructions as well as the execution time.

Besides the TDI, TMS and TRST that are stored in memories, the top module has the reset, *start*, *finish* and two different clocks as inputs. The clocks, as mentioned, are generated by the Mixed-Mode Clock Managers that uses the AXI Interface clock of 50 MHz as the base clock. A clock of 30 MHz is generated for the JTAG TAP Controller, which is a normal JTAG frequency, while the detection system works with a clock frequency of 150 MHz. The fact that the detection system is running with a frequency 5 times higher than the JTAG ensures the meeting of the timing

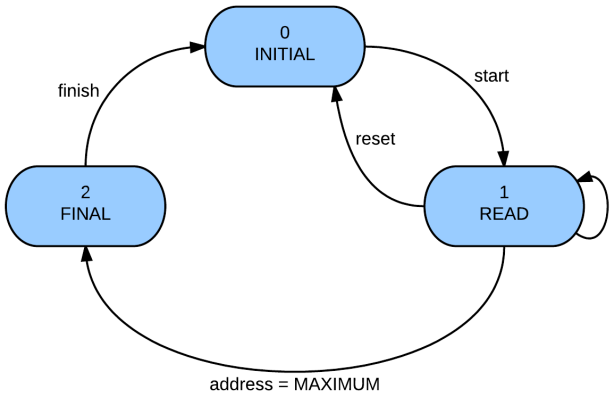


Figure 5.2: Global Control Finite State Machine

constraints, imposed by the Update Instruction Register signal. It is not possible to determine the number of clock cycles needed for one classification since it depends on the instruction, but as shown in section 4.2.5 it can take 20/30 clock cycles. Although it is not common, the worst case scenario regarding the Update Instruction Register could be 10/12 clock cycles, which means that a faster clock for the classifier is definitely needed.

The outputs of the system's top module, as mentioned in the previous section, are the *out* and *got_reset* flags, the alert and the counter of alerts. It is important to note that this top module is instantiated within the AXI Interface, in order to read the inputs from the registers, apply them to the module and write the final outputs in the registers.

The final system has a global finite state machine (FSM) with 3 states, shown in Figure 5.2 to control the reading of the inputs from the external memories and to assign the *out* flag.

The system stays in the initial state until the *start* is sent by the C program. Once the start is active, it starts reading the TDI, TMS and TRST from the correspondent memories, with the clock frequency of the JTAG TAP Controller. While in this state, both the detection system and TAP Controller are working. When the end of the instruction set is reached (after reading the last bit of TDI, TMS and TRST), which depends on the size of the set of instructions, the FSM moves to the final state and the *out* flag is set to one. In this state, the C program can already read the flag and end the execution. Once it ends the loop, it sends the *finish* flag, and the FSM goes to the initial state to wait for a new set of instructions.

5.4.1 Resources Usage

The resources usage of the complete system, including the AXI Interface and the MMCM, is shown in table 5.2. One can see that the final implementation is extremely small compared to the entire board. In order to give a better visualization of the space, Figure 5.3 shows the FPGA board with the implemented design represented by the blue dots. However, neither the figure nor the table include the memories that store the instruction bits, since they depend from set to set.

Table 5.2: Complete System Resources Usage

Resource	Utilization	Available	Utilization (%)
Flip-Flops	1 492	437 200	0.34
LUT	5 297	218 600	2.42
Memory LUT	3 272	70 400	4.65
BUFG	3	32	9.38
MMCM	1	8	12.50

5.4.2 Execution Time

The execution time depends on the size of the set of instructions. However, in section 4.2.5 two different sets of input features were applied to both classifiers. In these sets, the parallel classifier took 22 and 28 clock cycles for the classification. With the clock frequency of 150 MHz, which is the frequency that the classifier is running in the final system, these two classifications would take approximately 147 and 187 nanoseconds, respectively. In fact, there are instructions that can take more or less than the number of clock cycles presented, since these are just examples.

Nevertheless, in the final system, the execution time depends on the frequency of the JTAG, since it is the responsible of triggering new instructions. Hence, it is not related with the frequency of the detection system, assuming that all the classifications are done within the time.

Four different clock frequencies (12.5, 25, 30 and 42 MHz) for the JTAG TAP Controller module were tested for four sets of instructions, two illegitimate and two normal, and the execution times were determined. Table 5.3 shows these times, in microseconds. These times include the sending of a reset pulse in the beginning of the execution, and also the waiting time of the response to the reset pulse, in order to know that the reset was read and executed.

Table 5.3: Execution Times in microseconds for different clock frequencies

Instruction Sets	JTAG Frequencies (MHz)			
	12.5	25	30	42
Set 1 - Normal - Boundary Scan	2 003	1 006	840	603
Set 2 - Normal - CREG Read	488	248	209	152
Set 3 - Illegitimate - eFUSE	471	240	202	147
Set 4 - Illegitimate - Clock Manipulation	554	282	236	172

The four sets that were used were provided by the authors and are based in the Test Control Unit (TCU) of the OpenSPARC T2 [3]. The first set represents the normal Boundary Scan operation that is used in manufacturing test, while the set 2 indicates read operations to Control Registers (CREG). Control Registers are internal registers of the computer system that can control the computer configurations and settings. The set number 3 and 4 represent two possible attacks. Set 3 is related with eFUSE, which is a technology that allows dynamic real-time reprogramming of computer chips whose interface for read and write is provided by JTAG. Finally, set number 4 is about clock manipulations. The JTAG provides an interface capable of manipulating the clock

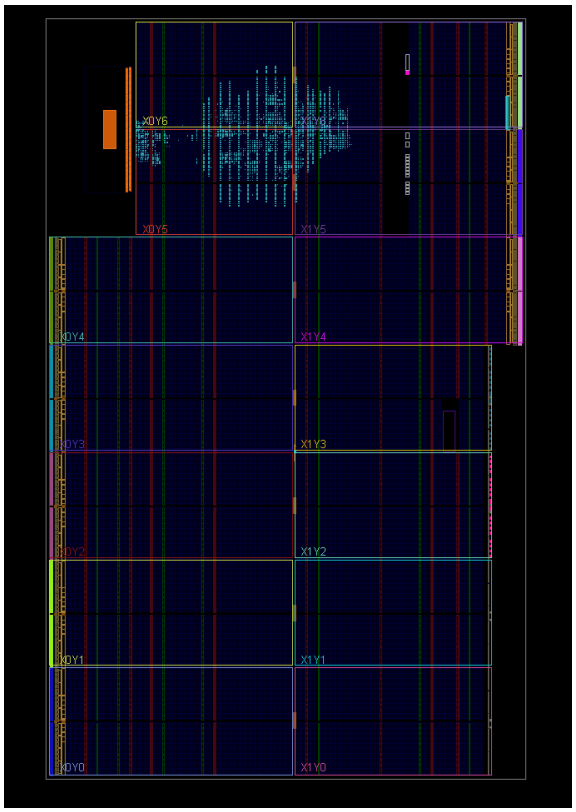


Figure 5.3: FPGA with Implemented Design

of a chip, including for instance, stop the clock, restart it, count the clock in debug mode, and many other operations. As stated, the execution time depends on the size of each instruction set. Thus, the sizes in number of instructions and bits per input (TDI, TMS and TRST) are shown in Table 5.4. The number of bits per input is shown because the number of instructions does not give an idea on the size of each set. For instance, the Boundary Scan set is the biggest set, in spite of having just one instruction (Boundary Scan is selected by EXTEST instruction).

Table 5.4: Number of instructions and bits per input for each instruction set

Instruction Sets	**Number of Instructions**	**Number of Bits Per Input**
Set 1 - Normal - Boundary Scan	1	24 924
Set 2 - Normal - CREG Read	66	5 980
Set 3 - Illegitimate - eFUSE	89	5 770
Set 4 - Illegitimate - Clock Manipulation	129	6 813

Analyzing both tables 5.3 and 5.4, it is possible to confirm that indeed, the execution times are very small concerning the total number of bits, and the system has an excellent timing performance.

5.4.3 Accuracy and Escape Rate

The accuracy and the escape rate for each set of the four that were tested are shown in Table 5.5. This method has the drawback of not dealing with imbalanced data sets. Thus, it is necessary to assume that in the sets that represent an attack, all the predictions were supposed to be 1, and in contrast, in sets that represent normal actions all the predictions were supposed to be 0. Hence, the True Positive (TP), True Negative (TN), False Negative (FN) and False Positive (FP) were counted. TP represent the number of illegitimate actions that were detected successfully and TN is the number of normal actions that were also detected. On the contrary, FN is the number of illegitimate actions that were not seen and FP represent the number of normal actions that were classified as illegitimate actions. Moreover, using these numbers, the Accuracy and Escape Rate were determined using equations 5.1 and 5.2. The accuracy is the fraction of correct predictions out of all predictions while the escape rate is the fraction of attacks that escape detection out of all attacks.

$$Accuracy = \frac{TP+TN}{TP+TN+FN+FP} \tag{5.1}$$

$$Escape\ Rate = \frac{FN}{TP+FN} \tag{5.2}$$

Table 5.5: Accuracy and Escape Rate

Instruction Sets	TP	TN	FN	FP	Accuracy	Escape Rate
Set 1 - Normal - Boundary Scan	-	36	-	0	100 %	-
Set 2 - Normal - CREG Read	-	65	-	1	98.48 %	-
Set 3 - Illegitimate - eFUSE	88	-	2	-	97.78 %	2.22%
Set 4 - Illegitimate - Clock Manipulation	241	-	5	-	97.97 %	2.03 %
Total	329	101	7	1	98.17 %	2.08 %

One can see that these results are extremely good, however they just give an idea of the performance of the system, and can not be assumed as the final accuracy. These accuracy and escape rate values are for these four specific sets of instructions.

5.4.4 Power Consumption

The total power consumption of the system was estimated using the *Vivado Design Suite* post-implementation analysis and is shown in Table 5.6. Four different power results were obtained for four different clock frequencies of the detection system: 60, 90, 120 and 150 MHz. The frequency of the JTAG TAP Controller was kept at 30 MHz in all these 4 implementations. For a better visualization, these values are also plotted in Figure 5.4.

Furthermore, it is shown in Figure 5.5 the distribution of the power. It can be seen that the dynamic power increases slightly with the increase of the clock frequency. This increase can be noticed in the Clocks and MMCM components, while the Logic and Signals stay constant.

Table 5.6: Power consumption for 4 different clock frequencies

Detection System Frequencies (MHz)	Total Power (W)
60	1.929
90	1.936
120	1.956
150	1.957

In fact, the power of the implementation is assumed to be just the four components that are shown inside the dynamic power: clocks, signals, logic and MMCM. These components represent the programmable logic, and if the four values are summed and this sum subtracted to the value of the dynamic power it is possible to conclude that the resultant value is always constant between the four implementations, approximately 1.567 W, which is assumed to be the dynamic power consumption of the processing system of the board.

Moreover, the system's IP Core was synthesized without the other components of the design (Processing System, AXI Interface) and the value obtained was always 0.245 W. Thus, this value can represent the static power consumption of the programmable logic of the *Zynq-7000 ZC706* board.

Hence, Table 5.7 shows the power that comes from the Processing System and Programmable Logic. Following what was mentioned before, this last value is composed by the 0.245 W and the sum of the four components.

Table 5.7: Processing System and Programmable Logic Powers (W) for different frequencies

Detection System Frequencies (MHz)	Processing System Power (W)	Programmable Logic Power (W)
60	1.567	0.362
90	1.566	0.370
120	1.566	0.390
150	1.567	0.390

5.5 Discussion

Analyzing all the results presented in this section, it is possible to admit that the complete system is fast and small in area. In fact, as mentioned, the speed only depends on the JTAG frequency, but besides that, a frequency of 150 MHz for the detection system could also be achieved, allowing the possibility of not only make all the predictions within the time, but also increase the clock frequency of the JTAG, if desired, consequently decreasing the execution time.

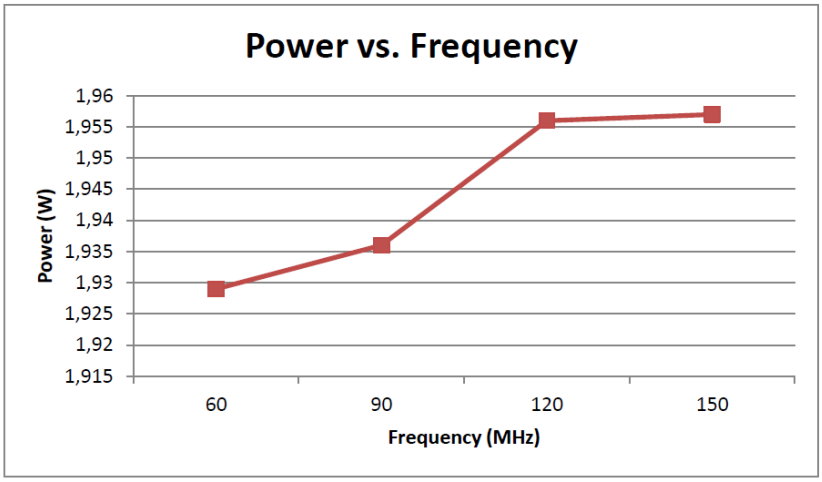
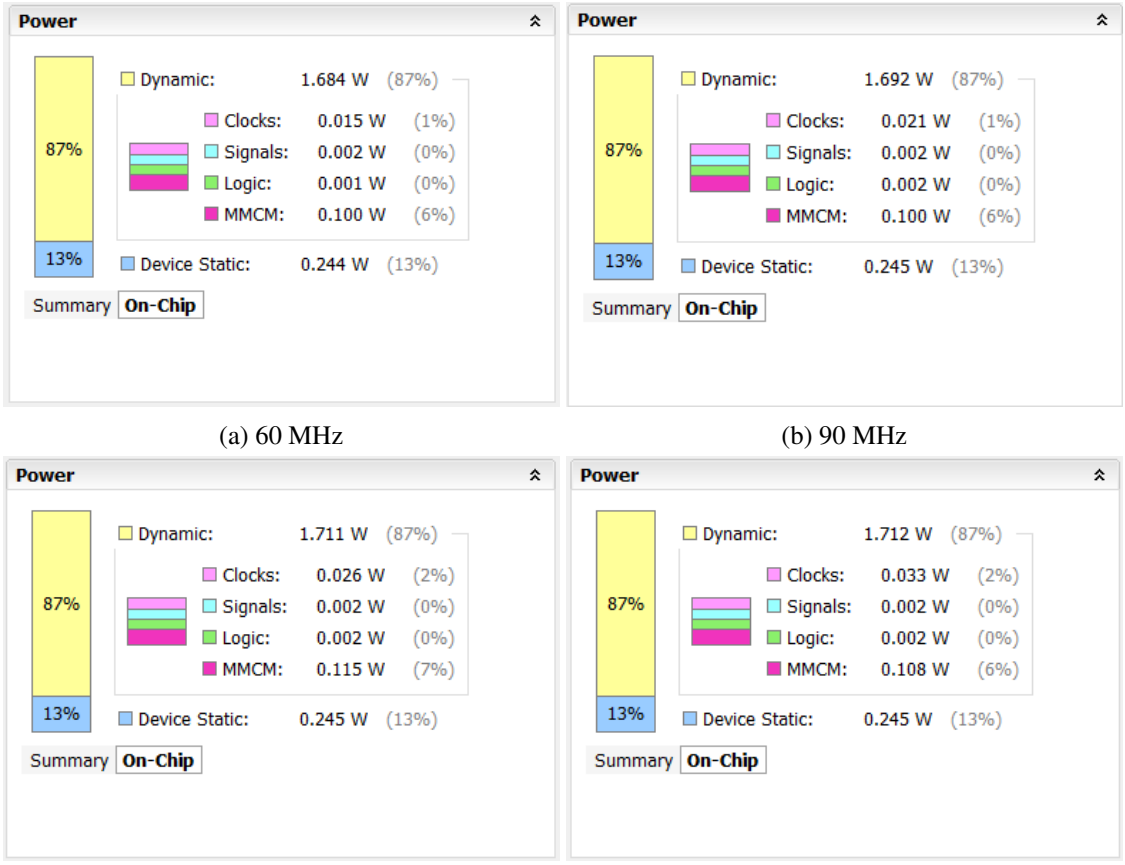


Figure 5.4: Power vs. Frequency

The accuracy and escape rate results give an idea of the performance of the system but as referred, they can not be assumed as final results. The sets are assumed to be balanced and in reality they are not. However, the results obtained show that the system is extremely accurate.

Moreover, the system consumes a very reduced power, since the Programmable Logic power, in all those four frequencies tested can achieve a maximum of 390 mW, where 245 mW is always present since it corresponds to static power.



(a) 60 MHz (b) 90 MHz

(c) 120 MHz (d) 150 MHz

Figure 5.5: Power Consumption and Distribution for 60, 90, 120 and 150 MHz

Chapter 6

Conclusions and Future Work

This chapter presents the final conclusions of this work, how the proposed objectives were achieved and some drawbacks of the current implementation as well. Moreover, it describes what can be done in the future to improve the system and integrate it in real systems.

6.1 Conclusions

The final goal of this work was to have the complete security system working properly in a *Xilinx Zynq-7000 All Programmable SoC ZC706*. This complete system includes the random forest classifier which consisted in the main focus of the work, and three more important modules: the data collector, the feature adapt logic and a memory that stores information about the transitions between the instructions. Moreover, the system would need to be capable of interacting with the JTAG, since it is a security system that tracks the activity in this important and widely used testing and debugging port. Thus, a JTAG TAP Controller module of the OpenSPARC T2 was used and included in the project to describe the typical behavior of the JTAG.

Being motivated by the fact that there are no hardware implementations of security systems that are based on Machine Learning, the main challenge of the work would be the need of a fast system that could be able to perform all the classifications within the time.

The objectives proposed were accomplished successfully and the entire system was simulated, synthesized and implemented on the board. A clock frequency of 150 MHz for the detection system could be achieved, and several frequencies for the JTAG TAP Controller were tested. The area and resources usage of the final implementation are extremely reduced as well as the power consumption which is just 390 mW for the Programmable Logic part at the highest frequency. The execution times obtained depend on the size of the instruction set that is being classified, and different instructions have different lengths. However, analyzing the sizes of the sets in number of bits, it is possible to conclude that the execution times are small, since 30 MHz is a normal JTAG frequency.

Therefore, this work represents, as far as the author knows, the first hardware implementation of a security system based on Machine Learning.

6.2 Future Work

Although this work is working properly in the FPGA and achieved all the proposed objectives, it was designed as a proof of concept since it is, as referred, the first hardware implementation of a machine-learning based security system.

Therefore, it has drawbacks. The major drawback of the implementation is the fact that the instruction sets are stored in memories and not being supplied externally. However, that can be a future step to improve this work, which is perfectly feasible but requires an extremely well done synchronization between the clock frequency of the inputs (JTAG) and the frequency of the detection system.

Moreover, the decision trees of the forest can always be improved, in terms of training or hardware implementation. Either by training them with a better training set, or by pipelining each tree, having for instance, a node for each level of the tree, instead of an universal node as is being used. This last suggestion would bring more hardware complexity but the speed of the detection system could be increased.

Finally, future work can also be the implementation of this system in an ASIC (Application-Specific Integrated Circuit). ASICs gives design flexibility which consequently gives opportunity for speed optimizations. Also, they can be optimized for required low power, since there are several techniques that allow the achievement of the target power, such as clock gating, power gating, etc. Furthermore, ASICs also have advantages in cost.

References

[1] Michael L Bushnell and Vishwani D Agrawal. *Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits*. Kluwer Academic Publishers, Hingham, MA, USA, 2000.

[2] Xuanle Ren, Vítor Grade Tavares, and R.D. (Shawn) Blanton. Detection of Illegitimate Access to JTAG via Statistical Learning in Chip. In *2015 Design, Automation and Test in Europe (DATE)*, 2015.

[3] Sun Microsystems. Test Control Unit (TCU). In *OpenSPARC T2 SoC Microarchitecture Specification*, chapter 4. 2008.

[4] IEEE Standards. IEEE Standard Test Access Port and Boundary Scan Architecture, 1990.

[5] XJTAG. Technical Guide to JTAG.

[6] IEEE Standards Association. IEEE Standard Test Access Port and Boundary Scan Architecture, 2001.

[7] Keunyoung Park, Sang Guun Yoo, Taejun Kim, and Juho Kim. JTAG Security System Based on Credentials. *Journal of Electronic Testing*, 26(5):549–557, September 2010.

[8] T.M. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[9] Stuart Russell and Peter Norvig. Artificial Intelligence: A Modern Approach. *Prentice-Hall, Egnlewood Cliffs*, 25, 1995.

[10] V. Lavrenko and N. Goddard. Introductory Applied Machine Learning. In *Introductory Applied Machine Learning*. University of Edinburgh, 2014.

[11] Leo Breiman. Classification and Regression Trees, 1998.

[12] L Rokach and O Maimon. Top-down Induction of Decision Trees Classifiers - A Survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 35(4):476–487, 2005.

[13] Xiaoyuan Su, Taghi M Khoshgoftarr, and Xingquan Zhu. VoB Predictors: Voting on Bagging Classifications. *2008 19th International Conference on Pattern Recognition (ICPR)*, pages 1–4, 2008.

[14] S Bernard, S Adam, and L Heutte. Using Random Forests for Handwritten Digit Recognition. *2007 9th International Conference on Document Analysis and Recognition (ICDAR)*, 2:1043–1047, 2007.

[15] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. The elements of statistical learning data mining, inference, and prediction, 2001.

[16] J J Rodriguez, L I Kuncheva, and C J Alonso. Rotation Forest: A New Classifier Ensemble Method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1619–1630, 2006.

[17] S Lopez-Estrada and R Cumplido. Decision Tree Based FPGA-Architecture for Texture Sea State Classification. *2006 IEEE International Conference on Reconfigurable Computing and FPGA's (ReConFig)*, pages 1–7, 2006.

[18] A Bermak and D Martinez. A compact 3D VLSI classifier using bagging threshold network ensembles. *IEEE Transactions on Neural Networks*, 14(5):1097–1109, 2003.

[19] J R Struharik. Implementing Decision Trees in Hardware. In *2011 IEEE 9th International Symposium on Intelligent Systems and Informatics (SISY)*, pages 41–46, 2011.

[20] Leo Breiman. Bagging Predictors. *Machine Learning*, 24(2):123–140, 1996.

[21] Thomas G. Dietterich. Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization. *Machine Learning*, 40(2):139–157, 2000.

[22] Yali Amit and Donald Geman. Shape Quantization and Recognition with Randomized Trees. *Neural Computation*, 9(7):1545–1588, 1997.

[23] Tin Kam Ho. The Random Subspace Method for Constructing Decision Forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.

[24] Kurt Rosenfeld and Ramesh Karri. Attacks and Defenses for JTAG. *IEEE Design and Test of Computers*, 27(1):36–47, 2010.

[25] National Institute of Standards and Technology (NIST). Data Encryption Standard (DES). In *Federal Information Processing Standards Publication 46*, 1977.

[26] Bo Yang, Kaijie Wu, and Ramesh Karri. Scan Based Side Channel Attack on Dedicated Hardware Implementations of Data Encryption Standard. In *Proceedings - International Test Conference*, pages 339–344, 2004.

[27] Hirokazu Kodera, Masao Yanagisawa, and Nozomu Togawa. Scan-Based Attack Against DES cryptosystems Using Scan Signatures. In *2012 IEEE Asia Pacific Conference on Circuits and Systems*, pages 599–602. IEEE, December 2012.

[28] C. Adams. SHADE cipher: An efficient hash-function-based Feistel network. In *Canadian Conference on Electrical and Computer Engineering*, volume 1, pages 338–342. IEEE, 1997.

[29] National Institute of Standards and Technology (NIST). Advanced Encryption Standard (AES). In *Federal Information Processing Standards Publication 197*, 2001.

[30] Sk Subidh Ali, Ozgur Sinanoglu, Samah Mohamed Saeed, and Ramesh Karri. New Scan-Based Attack Using Only the Test Mode. In *2013 IFIP/IEEE 21st International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 234–239. IEEE, October 2013.

[31] B. Yang, K. Wu, and R. Karri. Secure Scan: A Design-for-Test Architecture for Crypto Chips. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(10):2287–2293, October 2006.

[32] Rohit Kapur. Security vs. Test Quality: Are they Mutually Exclusive? In *Proceedings - International Test Conference*, page 1414, 2004.

[33] Jean Da Rolt, Giorgio Di Natale, Marie-Lise Flottes, and Bruno Rouzeyre. Are advanced DfT structures sufficient for preventing scan-attacks? In *2012 IEEE 30th VLSI Test Symposium (VTS)*, pages 246–251. IEEE, April 2012.

[34] Jean Da Rolt, Giorgio Di Natale, Marie-Lise Flottes, and Bruno Rouzeyre. Scan Attacks and Countermeasures in Presence of Scan Response Compactors. In *2011 Sixteenth IEEE European Test Symposium*, pages 19–24. IEEE, May 2011.

[35] Baris Ege, Amitabh Das, Santosh Gosh, and Ingrid Verbauwhede. Differential Scan Attack on AES with X-tolerant and X-masked Test Response Compactor. In *2012 15th Euromicro Conference on Digital System Design*, pages 545–552. IEEE, September 2012.

[36] Eldad Eilam. *Reversing: Secrets of Reverse Engineering*. John Wiley & Sons, 2011.

[37] Felix Domke. Blackbox JTAG Reverse Engineering. pages 1–5, 2009.

[38] A Ashkenazi, D Akselrod, and Y Amon. Platform Independent Overall Security Architecture in Multi-Processor System-on-Chip ICs for Use in Mobile Phones and Handheld Devices. *2006 World Automation Congress (WAC)*, pages 1–8, 2006.

[39] Franc Novak and Anton Biasizzo. Security Extension for IEEE Std 1149.1. *Journal of Electronic Testing*, 22(3):301–303, June 2006.

[40] G. Edward Suh and Srinivas Devadas. Physical Unclonable Functions for Device Authentication and Secret Key Generation. In *2007 44th ACM/IEEE Design Automation Conference (DAC)*, pages 9–14. IEEE, June 2007.

[41] L Pierce and S Tragoudas. Multi-level secure JTAG architecture. *2011 IEEE 17th International On-Line Testing Symposium (IOLTS)*, pages 208–209, 2011.

[42] L Pierce and S Tragoudas. Enhanced Secure Architecture for Joint Action Test Group Systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(7):1342–1345, 2013.

[43] R.F. Buskey and B.B. Frosik. Protected JTAG. *2006 International Conference on Parallel Processing Workshops (ICPPW)*, pages 405–414, 2006.