Ca' Foscari University of Venice

Department of Environmental Sciences, Informatics and Statistics

Master's Degree programme
Computer Science, Software Dependability and Cyber Security
Second Cycle (D.M. 270/2004)

Master Thesis

Efficient Black-box JTAG Discovery

Supervisor:
Prof. Riccardo Focardi

Assistant supervisor:
Dott. Francesco Palmarini

Graduand:
Riccardo Francescato
Matriculation Number 857609

Academic Year
2016 - 2017

Riccardo Francescato
Matriculation Number 857609
Efficient Black-box JTAG Discovery, Master Thesis

Abstract

Embedded devices represent the most widespread form of computing device in the world. Almost every consumer product manufactured in the last decades contains an embedded system, *e.g.,* refrigerators, smart bulbs, activity trackers, smart watches and washing machines. These computing devices are also used in safety and security-critical systems, *e.g.,* autonomous driving cars, cryptographic tokens, avionics, alarm systems. Often, manufacturers do not take much into consideration the attack surface offered by low-level interfaces such as JTAG. In the last decade, JTAG port has been used by the research community as an entry point for a number of attacks and reverse engineering techniques. Therefore, finding and identifying the JTAG port of a device or a de-soldered integrated circuit (IC) can be the first step required for performing a successful attack. In this work, we analyse the design of JTAG port and develop methods and algorithms aimed at searching the JTAG port. More specifically we will provide an introduction to the problem and the related attacks already documented in the literature. Moreover, we will provide to the reader the basics necessary to understand this work (background on JTAG and basic electronic terminology). Successively we provide the analysis of the problem and a naive solution. After having ascertained the poor performance of the basic solution, we will introduce an efficient set of algorithms that exploit the electronic properties of the components. Moreover, we will present a randomized approach to the problem that exploits the electronic properties to further reduce the search time. We will also provide suggestions end techniques to discover JTAG port directly on printed circuit boards. For concluding we will provide to the reader some suggestions for a proficient use of the technique presented in this work.

Acknowledgements

Foremost, I would like to express my sincere gratitude to my thesis advisor Professor Riccardo Focardi for the continuous support during my studies and the writing of this work. I would also like to thank Dr. Francesco Palmarini for his invaluable aid during the research and the writing of this work, for his patience, for the detailed explanations, the supplied solutions, the competence and the kindness, without whom this work would not have been possible. I would also like to thank all the professors met during the Bachelor and Master degrees for their passionate and inspiring influence.
Last but not least, I must express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study. This accomplishment would not have been possible without them.
Thank you.

Riccardo Francescato

List of Figures

Contents

Chapter 1

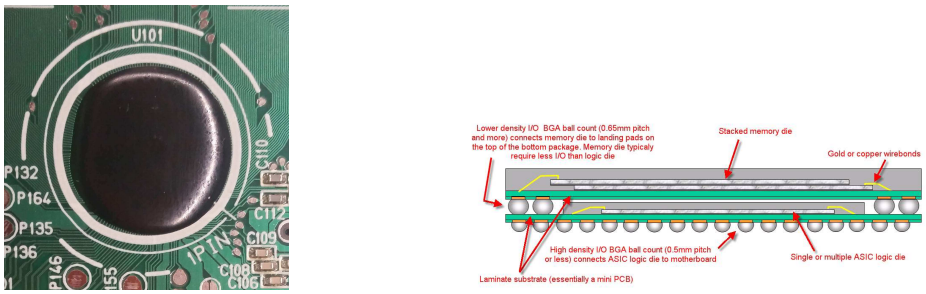
Introduction

Embedded electronic systems probably represent the most widespread form of computing device in the world. Embedded devices are contained in almost every consumer product manufactured in the last decades, *e.g.,* refrigerators, smart bulbs, activity trackers, smart watches and washing machines. These computing devices are also used in safety and security-critical systems such as autonomous driving cars, cryptographic tokens, avionics, alarm systems, etc.. Typically, the core of an embedded system is an integrated circuit (IC) or microcontroller. Joint Test Action Group (JTAG) is an IEEE 1149.1 standard [1] for in-circuit test. This interface is designed to aid engineers and manufactures during the design and production phases. They need tools to test PCBs[1] and to program and debug the integrated circuits. The JTAG standard illustrates a basic set of features that allows boundary scan[2] for ICs, and also leave space for additional features for programming and debugging purposes. The research interest on JTAG port is given by the countless and valuable information that it can expose. It can disclose manufacturer's name, IC model and revision, data contained in RAM, the flash memory content and the internal processor registers. For instance, when dealing with an unknown, unlabelled or undocumented IC (see Figure 1.1a), this information can be extremely useful for finding the datasheets or other kinds of information. In particular, there are countless scenarios where manufacturers scrape off the label of electronic components to prevent their identification or to hamper the reverse engineering process. Moreover, a microcontroller can be stacked[3] under another IC (typically memory) in such a way that is not possible to clearly identify it (see Figure 1.1b).
JTAG standard lays down an interesting feature, the ability to share a single port on the PCB and share it with several chained JTAG compliant ICs. This feature allows for reverse engineer the whole PCB by identifying the ICs mounted on the board and, through boundary scan functions, determining all the interconnections from a single access point. A JTAG port can also expose, either due to a misconfiguration or an attack, the firmware contained in the flash memory of the device. An attacker that manages to gain access to the firmware can identify and exploit software vulnerabilities or infringe the intellectual property. JTAG port, once identified, constitute a main entry point for several attacks. In the last two decades, several

[1] PCB, or printed circuit board, consists of a non-conductive board laminated with copper that both offers mechanical support and electrical connection to all the mounted electronic components.

[2] Boundary scan is a method for testing interconnections between components soldered onto a PCBs.

[3] Also known as package on package (PoP)



(a) Typical case of chip on board (b) Package onaiven Package [8]

Figure 1.1: Examples of hidden or undocumented IC

attacks have been published. The first one is [3], the authors showed how to obtain a one-to-one dump of the system memory using the JTAG interface. Another related research is represented by the work carried out by Rosenfeld and Karri [26] where they describe several attack patterns (*e.g.,* sniff and modify the TDI/TDO signals) and countermeasures. In [18] Moein, Gebali and Traore described the covert JTAG port attack that allows for sniff secret data from the JTAG port. The authors of [15] presented a privilege escalation attack over JTAG; they exploited this low-level interface for modifying addresses of kernel system calls. A widely known and used attack in the world of video-games console is the JTAG hack on Xbox 360 [10]. It breaks DRM system on Xbox console, allowing users to play non-original contents. In [11] Domke presented a black-box fuzzing attack of the JTAG registers that can lead to the discovery of undocumented features of the target IC.

1.1 Structure of the thesis

In this work, we will provide the fundamental concepts of JTAG, a set of automatic procedure, methods, and tools that aim to find the pins related with the JTAG port in the case of an unknown or undocumented IC or PCB.

Background (Chapter 2) provides the reader a list of definitions and acronyms about software and hardware terminology. Successively it introduces the basic description about the hardware architecture, embedded systems, and microcontrollers. Finally, it describes the fundamental concepts of the JTAG standard useful to fully understand this work.

Analysis of the problem and naive solution (Chapter 3) is divided in four sections. The first part of the chapter is dedicated to the introduction of the problem addressed by this work. In the second and third part, are presented the two brute force solutions and evaluated the performances of the two proposed algorithms. Finally, will be presented the hardware problem derived by a non standard implementation of JTAG and it's solution.

An efficient algorithm based on 4-state GPIO (Chapter 4) is divided in six sections. The first part of the chapter is intended to give the basic electronic background necessary to understand the electrical properties exploited by the new algorithms presented in the successive sections. The third and fourth part describe the two new algorithms that exploit the electrical properties in order

to reduce the time required to accomplish the task; will be also discussed the performance, problems, and limitations of these solutions. Finally, the last part describes the randomized algorithm that exploits the electrical properties described in the first section.

Comparison (Chapter 5), this chapter aims to discuss and compare the performances of the algorithms in terms of time of the various algorithms presented in Chapter 3 and Chapter 4

Performing attacks on PCBs (Chapter 6) will introduce the technique used to find the possible JTAG port on a PCB and the problems that an attacker can encounter during this process.

1.2 Contributions

- IEEE 1149.1 standard [1] may not be easy to understand and can require a good amount of time for a complete comprehension. In this work we provide a summary of the fundamental features and technological background of JTAG standard, evicting the specific electronic design implementation that is not useful for JTAG users.

- In order to solve the problem of automatically find a JTAG port on an unlabeled or undocumented integrated circuit or printed circuit board, we provide a detailed analysis of the problem by studying the IEEE 1149.1 standard and the possible methods of attack. Moreover is developed a brute-force naive solution that exploits the standard features of the JTAG port add guarantee the maximum compatibility with all integrated circuits that implement the JTAG standard.

- In order to reduce the time complexity of the naive algorithm, we provide an analysis of the electrical properties that can be exploited to reduce the attack time. Moreover, we propose an algorithm that exploits the aforementioned properties to automatically find the JTAG port on a desoldered integrated circuit. Furthermore, we provide a Monte Carlo randomized algorithm which exploits the new electrical properties and solves the problem with a quasi-linear time complexity.

- We provide a set of software, electronic principles and best practices to find out the JTAG port directly on printed circuit boards. In particular, we introduce the most common scenarios and problems that an attacker can face when performing the attacks on PCBs. Specifically, we analyze the limitations caused by the components present on the board, we provide suggestions to reduce the number of pins to test and analyze the case in which on the board is soldered more than one JTAG enabled integrated circuit.

Chapter 2

Background

In this chapter, we introduce the basic concepts necessary to fully understand the work presented in this thesis. In Section 2.1 we give a list of acronyms and definitions related to hardware and software terminology. Following Section 2.2 will provide an overview of the equipment used during development and test phases. Finally, Section 2.3 will present a brief overview about embedded systems.
Topics treated in this work are broad fields of studies, thus we have identified a list of prerequisites necessary to fully understand this work.

- Basic concepts of digital and analog electronics.
- Minimal knowledge of signal theory.
- Minimal knowledge of embedded systems internal design.

2.1 Acronyms and Definitions

2.1.1 Hardware Terminology

This section aims to present the meaning and description of hardware-related terms and acronyms used in this thesis:

Embedded system: also known as embedded device is a computing system designed to accomplish specific functions or tasks (*e.g.,* ADSL modem/router, calculator, satellite receiver, etc.). These devices are designed to meet physical constraints where general-purpose systems are not suitable (*e.g.,* low power consumption, real-time computing, low per-unit cost).

PCB: *Printed Circuit Board* consists of a non-conductive board laminated with copper that both offers mechanical support and electrical connection to all the mounted electronic components. PCBs can be single sided (one copper layer), double-sided (copper layers on both faces of dielectric board) or multilayer (copper layers alternated with non-conductive substrate).

Integrated Circuit also abbreviated with *IC*, is usually the core component of embedded system, a chip or a microchip. Is a set of electronic circuits built on a layer of semiconductor material, commonly silicon. The integrated circuit definition includes a large range of components *e.g.,* memory chip, microprocessors, logic gates etc..

PoP: is an integrated circuit package method that allows manufacturers to vertically combine two or more ICs. Packages are stacked one over the other allowing a higher density of components in devices such as smartphones.

GPIO: *General-Purpose Input/Output*, as the name suggests, is a generic type of pin that is built into most ICs or single board computers. The behavior of this component is controllable by the programmer via software. Programmers can customize the direction (input/output), also switching from input to output (and vice versa) during the execution of the program. The pull up / down resistors can be set and changed at software level. Also, the signal type analogical or digital can be selected.

Pin: Most integrated circuits are enclosed in a plastic (or ceramic) enclosure called *package* which, in turn, exposes the electrical connections called pins. Pins are thin pieces of metal that connect the die[1] to the external environment.

Microcontroller: or commonly abbreviated MCU, consists of a single integrated circuit on which are built a CPU (*i.e.,* central processing unit), memory such as flash or ROM, usually a small amount of RAM and programmable input/output peripherals.

Clock: is a particular type of signal that oscillates between to two voltage levels and is utilized for synchronizing the actions of digital circuits. Usually, the signal used for clock is in the form of a square wave with a duty cycle of 50% and a fixed constant frequency. Usually, this kind of signal is generated by a crystal oscillator (or resonator) or by a circuit that generates a clock signal.

RAM: *Random Access Memory* is a volatile (when power is disconnected all data are cleared) form of computer data memory that temporarily stores data and instructions.

ROM: is a non-volatile, read-only memory type; commonly used for storing basic programs like bootloaders.

EEPROM: *Electrically Erasable Programmable Read-Only Memory* is a non-volatile kind of memory, used to store small amounts of data (*e.g.,* device configuration, calibration tables) that must be saved when power is removed.

Flash memory: is a non-volatile read and write memory, that can be electrically erased and reprogrammed. Commonly used for storing from few kilo-bytes to some gigabytes of data and programs.

Logic level: logic levels represent the two state that a bit can assume 0 or 1, which corresponds to the low and high logic level. This two logical states, in digital electronic, are represented by two different voltage levels. Two thresholds are designed to distinguish the low and high level. Usually, these thresholds are calculated via some mathematical formulas on the voltage provided by the power supply. Usually, the logic low or 0 is represented by the ground or 0V, while the high or 1 is represented by 1.8V, 3.3V, 5V and 12V depending on the power supply voltage.

[1]DIE, is the small block of semiconductor material on which the IC circuitry is fabricated.

JTAG: *Joint Test Action Group* is an IEEE 1149.1 standard [1] for in-circuit test. Is designed to aid engineers and manufactures during the design and production phases. They need tools to test PCBs and to program and debug the integrated circuits.

BSDL: *Boundary Scan Description Language* is a hardware description language, part of the IEEE 1149.1 standard [1]. It describes the public instructions and design implemented in a particular IC.

Datasheet: is a document provided by the manufacturer of electronic devices that summarized the features and characteristics of a specific electronic component. These documents contain information such as the pin mapping, supply voltage, functional diagrams, etc..

2.1.2 Software Terminology

This section aims to present the meaning and description of software-related terms used in this thesis:

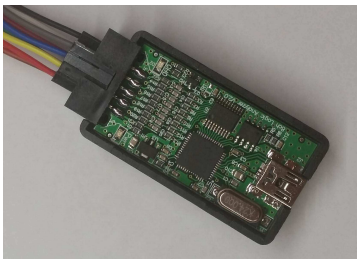
Firmware: is the combination of data and program code stored in the persistent memory of an IC. In embedded devices firmware is the main program running on the device and usually is self-contained and does not require any external software component. Usually, on embedded devices, it is stored in the non-volatile memory such as flash or EEPROM or others.

Boot-loader: is the first piece of software that is loaded when an embedded component starts up. It provides a minimum set of functionalities such as firmware update and is in charge to load the firmware on start-up. It is usually stored in a virtually separated area of the flash or in a ROM.

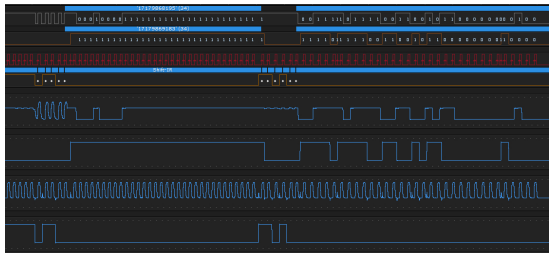
Debugging: is the set of methodical process that in software engineering is performed with the purpose of finding and reducing the number of errors (bugs). While in reverse engineering context denominates the set of instruments and methods used in order to help reverse engineering processes.

2.2 Equipment

Logic Analyser: is electronic equipment that allows capture and register multiple signals (digital and analog) from digital systems. Is usually a combination of hardware Figure 2.1a and computer software Figure 2.1b, to record, display and analyse digital traces. It can decode communication protocols, state machine traces that can correlate software events to digital signals. Usually the limit of these devices is the maximum signal frequency (from 3 MHz to 100 MHz), the sampling rates (from 6 MS/s to 500 MS/s) and the input voltage range that goes from 0 V to a maximum of 5 V (in some devices such as Saleae Logic Pro that implements analog traces voltage range is extended from −10 V to 10 V).



(a) 24 MHz 8 signal logic analysers with cables connected



(b) Saleae Logic software with traces and decoding of JTAG protocol

Figure 2.1: Hardware and software components of logic analyser used in this work

Digital Multimeter: is a electronic appliance that combines several measurement functions. Typically, multimeters include voltage, current and resistance measurement functions. This instrument is useful to identify power sully voltages on PCBs. Moreover, the resistance measurement function can be used to identify PCB traces that connects ICs to test pads or to connectors.

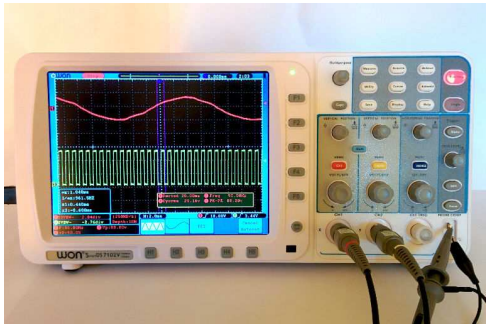


Figure 2.2: On the left Iso-tech digital multimeter, On the right Owon DS7102V digital storage oscilloscope

Power supply: is the electronic device that supplies electric current to all other electronic components. This device has been used in this work to supply power to the *TargetIC* with specific voltages.

ST Nucleo-F446ZE: is a development board produced by ST semiconductors, it integrates an MCU with a 32-bit ARM processor core, 6 GPIO (see Section 2.1.1) ports with 16 pins each. In this work the Nucleo board is used as *AttackerIC*. It provides 99 free GPIO pins that can be connected to the *TargetIC* and used by the algorithms, that will be described below, to find the JTAG port on the *TargetIC*.

Digital Storage Oscilloscope: is a electronic test instrument that allows observation of analogue an digital signals in the domain of voltage and time. Digital storage oscilloscope allows sample and record signals for a live or offline analysis. In this work these devices is used to analyse the characteristics of the JTAG clock signal, such as max. and min. voltage, rising time, falling time, etc.. Differently from logic analyser this device has a more wide range of voltage input and a higher sampling frequency (1 GS/s).

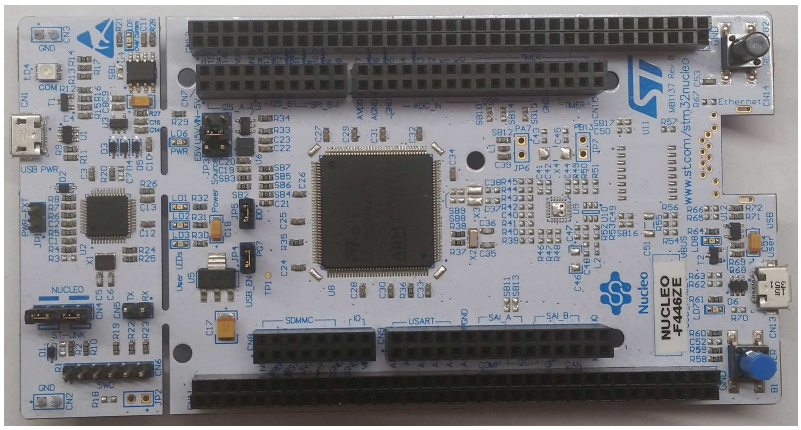


Figure 2.3: ST Nucleo-F446ZE board used as *AttackerIC*

2.3 Embedded Architectures

Embedded devices or embedded systems are computer systems *i.e.,* a combination of hardware and software built on top of a printed circuit board to achieve specific tasks. They differs from general-purpose computer for lower power consumption, small size, low costs, real time operations, etc.. Typically, embedded systems can be represented schematically as in Figure 2.4. Usually embedded systems core device is the microcontroller unit (MCU see 2.3.1), this component is directly connected with volatile and non-volatile memory that supports the MCU operations. Frequently, embedded systems are equipped with external flash or EEPROM memory that allows storing bigger amount of data with respect to MCU internal non-volatile memory.

2.3.1 Microcontrollers Architecture

As written in 2.3, the core component of an embedded system is the microcontroller. This section will give an overview of the main components (see Figure 2.5) and functionalities of a microcontroller (also called IC). Microcontroller (or MCU) is not only the central processing unit of the embedded system but a full computer on a single integrated chip (is similar, but less sophisticated than a SoC or system on chip). Generally, a microcontroller contains a CPU (processing core) that can range from 4-bit architecture to a 32-bit or 64-bit architecture on high power devices. The CPU is responsible for running the firmware code and operating all other peripherals. Alongside with the CPU is provided an integrated memory, both volatile and non-volatile, such as CPU RAM, flash and an optionally EEPROM. Typically, flash and RAM memories are directly mapped to the CPU address space, while EEPROM is accessed from the peripherals data bus. Microcontroller also integrates peripherals such as timers, analog to digital converters (A/D converters) communications devices such as USB, UART, SPI, I^2C, etc.. GPIO ports are raw digital I/O controllers which are used to control every single pin programmatically. Microcontroller also usually integrate debug ports such as JTAG, this interface is the core of this thesis work, it allows complete low-level access on almost all components being part of the microcontroller.

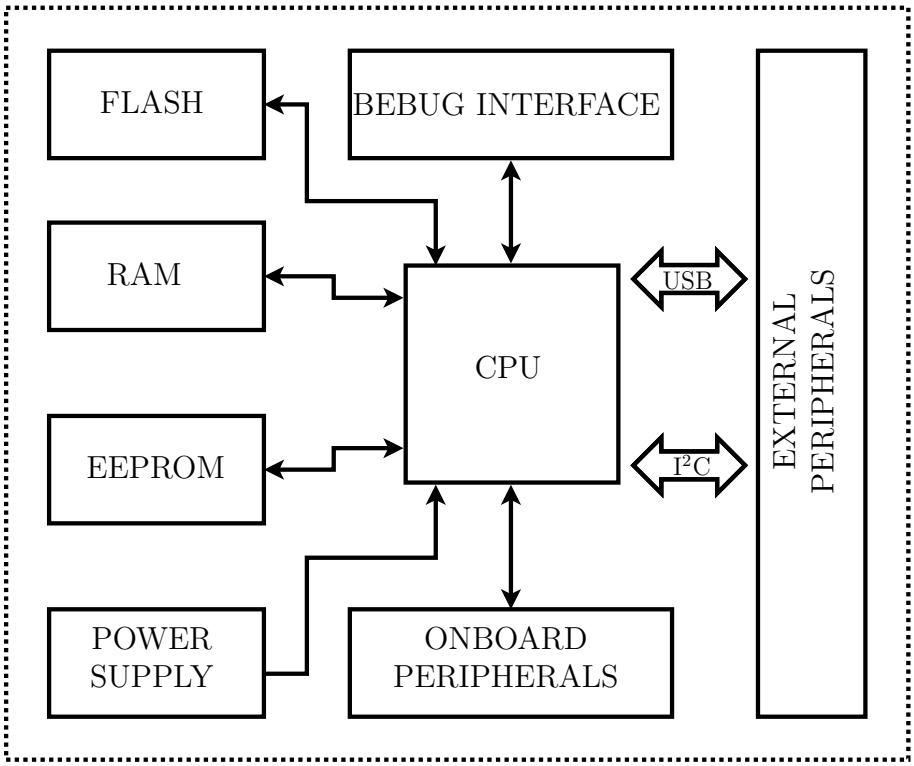


Figure 2.4: Typical logic block diagram of an embedded system

2.4 JTAG Standard

With high-density PCBs, standard debug methods such as *bed of nails*[2] are becoming challenging and expensive due to the concentration of components in the PCB or due to the high number of test pads. To solve this problem in the early 90s a group of industries developed JTAG and after few years IEEE standardized this debug port with IEEE 1149.1 standard. The IEEE 1149.1 standard illustrates the Test Access Port (TAP) and its operations along with a mechanism to provide additional features. JTAG is designed to achieve three main tasks:

1. testing the interconnections between the various integrated circuits that are already assembled on the board;

2. testing the integrated circuit itself;

3. examine or alter the behavior of the circuits during normal functioning.

Test access port is the core of JTAG; it can be subdivided in the more electric and physical part, the TAP, and a more abstract part that is in charge to assist the hardware operations, the TAP controller.

[2]The bed of nails is a classical tool for testing PCBs. It is composed of many pins that establish an electrical connection with the test pads on the PCB. This tool is an automatic and reliable testing procedure during manufacturing.

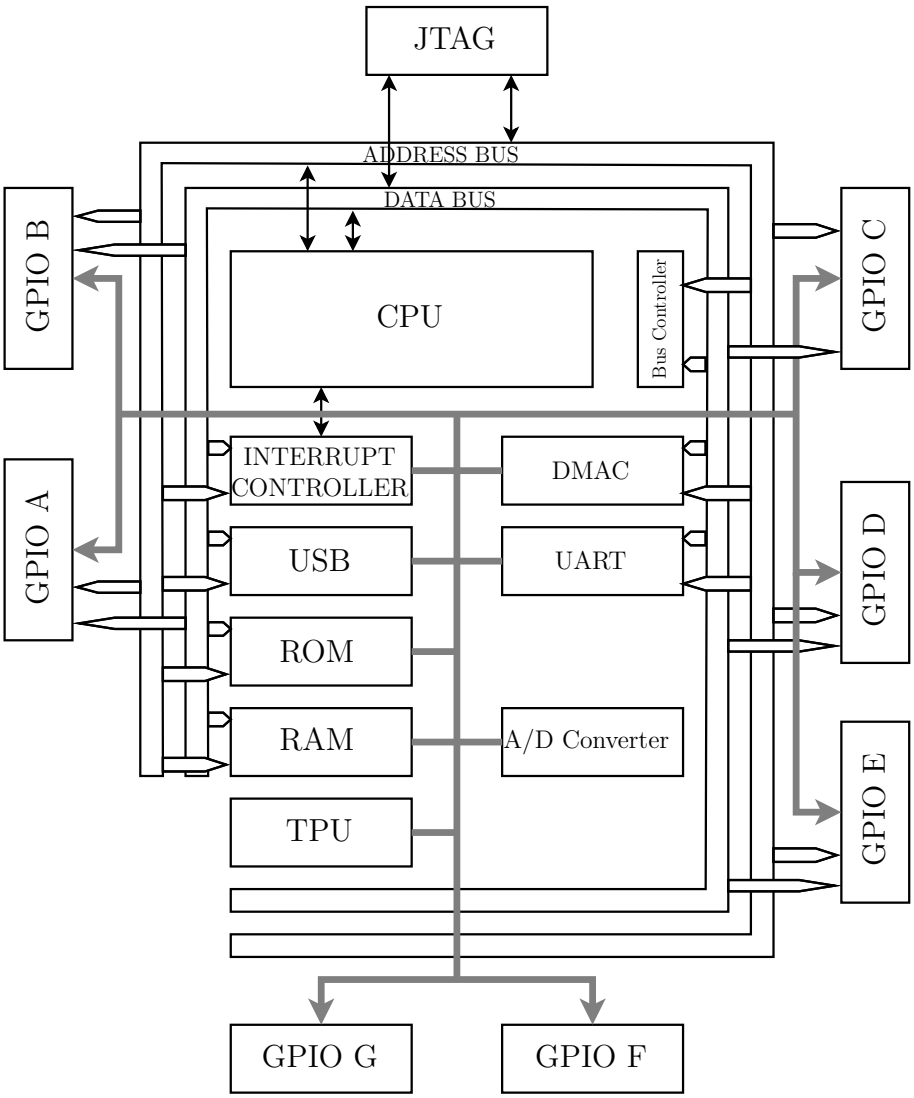


Figure 2.5: Typical reduced internal logic block diagram of a microcontroller. Are shown the main components such as CPU, memory, buses, GPIO ports, JTAG port, etc.

2.4.1 Test Access Port (TAP)

TAP is the more physical and electric part, it is integrated into the silicon and provides the hardware support to the TAP controller. TAP includes the support for several test functions, including the test logic delineated by JTAG standard. It is composed by the electrical connections (pins) that are necessary for the communication with the driver. There is a minimum of three inputs, one output and one optional input for asynchronous reset of the TAP. The characteristics and the functions of each signal are described below.

TCK provides the clock for the test access port. This independent clock allows JTAG to operate the tests independently from the main system clock. These features grant that test data can be shifted in and out from the test logic independently from the system logic and also when the core logic is in its

normal operation (useful for debugging). Change of state and read/write operations are performed during the rising / falling edge of the TCK signal and these operations must be performed in a precise time interval. The timing is independent of the clock and set out by the manufacturer.

TMS is a control pin that allows navigating through the TAP state machine, selecting the data or instruction registers (IR). Signals on TMS must be shifted on the rising edge of TCK.

TDI is a data pin for the serial communication, in particular, it is used to shift data into the JTAG registers. By standard, the data shifted in from TDI must appear, unchanged, as output on TDO after a number of clock cycles.[3] Data on TDI is shifted into the chosen register on the rising edge of the clock signal.

TDO is the serial output pin of the TAP controller. The content of the selected register is shifted out in TDO every falling edge of the TCK signal.

TRST is an optional control signal that provides an asynchronous reset / initialization of the TAP state machine, *i.e.,* it brings the state machine to the *Test-Logic-Reset* when TRST is put at logic level one (if $\overline{\text{TRST}}$ is negated, the state machine is moved to *Test-Logic-Reset* when $\overline{\text{TRST}}$ is put at logic level zero). In this work we assume that TRST is active high namely, while the TRST signal is driven to logic level high the TAP controller is forced in *Test-Logic-Reset* state.

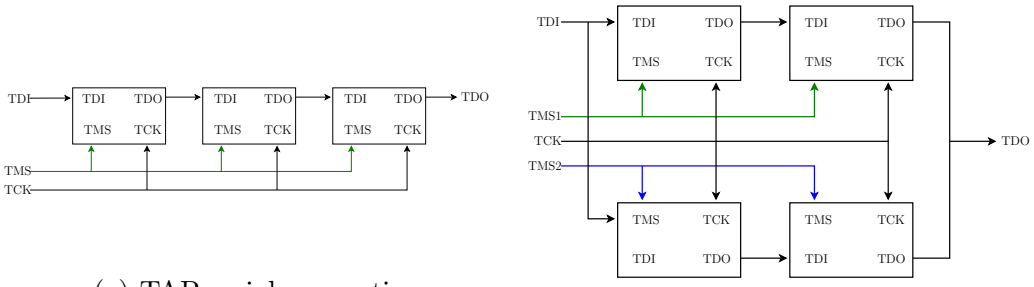
On the same PCB can be mounted several components that are compliant with the JTAG standard autoeg processor, flash memory, FPGA or ASIC and special-purpose hardware such as image DSP or even deep learning accelerators [5, 6]. To optimize the PCB layout and minimize the number of traces and exposed test connectors, all the TAP controllers on a PCB can be connected in series or in parallel. In the serial configuration Figure 2.6a TCK and TMS are common to all ICs, while each output pin (TDO) is connected to the input data pin (TDI) of the subsequent IC of the chain. This setting allows conducting a set of tests aimed at verifying that all the interconnections between ICs are correct. The parallel chain Figure 2.6b allows partitioning the test domain: two or more chains of ICs are controlled via a common TCK signal and a dedicated TMS lines.

2.4.2 TAP Controller

The TAP controller constitutes the more abstract part of the JTAG port. Its core component is the state machine (see Figure 2.7) that responds to the TMS, TCK signals and supervise the sequence of operations in the JTAG circuitry. For the sake of brevity, in the section below are described only the set of actions performed in the states (of the JTAG state machine) that are relevant for this work:

Test-Logic-Reset In this state, the test logic disabled and the operation of the on-chip system logic can be executed regularly. The instruction register is initialized with the `IDCODE` instruction, or if it is not set out by the manufacturer

[3] This allows for daisy-chaining of multiple devices on the same JTAG bus



(a) TAP serial connection

(b) TAP parallel connection two chains

Figure 2.6: TAP interconnection sachems

of the chip, (`IDCODE` is an optional instruction see Section 2.4.3) it is initialized with the `BYPASS` instruction (see Section 2.4.3). The state machine will be brought in this state when the TMS pin is held high for five rising edges of TCK. If the optional asynchronous reset pin TRST is implemented, the state machine, the controller can be brought to this state by applying a high logic level (or low logic level if $\overline{\text{TRST}}$).

Shift-DR In this state the data register selected by the instruction shifted into the IR is selected and connected between TDI and TDO. When TMS is held low and at TCK is applied a rising edge a bit is shifted into the DR and when the falling edge of the clock occur a bit from the DR is shifted out and can be read on TDO.

Shift-IR In this state, the shift-register that is responsible for containing the instructions is connected between TDI and TDO. When TMS is held low and at TCK is applied a rising edge a bit is shifted into the IR.

2.4.3 The instruction register

The instruction register allows shifting an instruction into the design. Each instruction is used to identify a particular test to be performed or the test data register to be accessed or both. There are a number of mandatory instruction that every design must implement to be compliant with the IEEE 1149.1 standard. JTAG allows also the possibility to implement additional instructions to control design specific properties. JTAG instructions are divided into two classes, public and private. Public instructions should be available to the end-user instead, private instructions are intended solely for the component manufacturer use. The mandatory instructions that are part of the public domain are: `BYPASS`, `SAMPLE`, `PRELOAD` and `EXTEST`. In the section below are described only the instructions that are relevant to this work:

Public

The instruction that is public must be available to the end user of the component. Test features illustrated by each instruction must be independent of the variant of the component, except variant specific instructions like `IDCODE`.

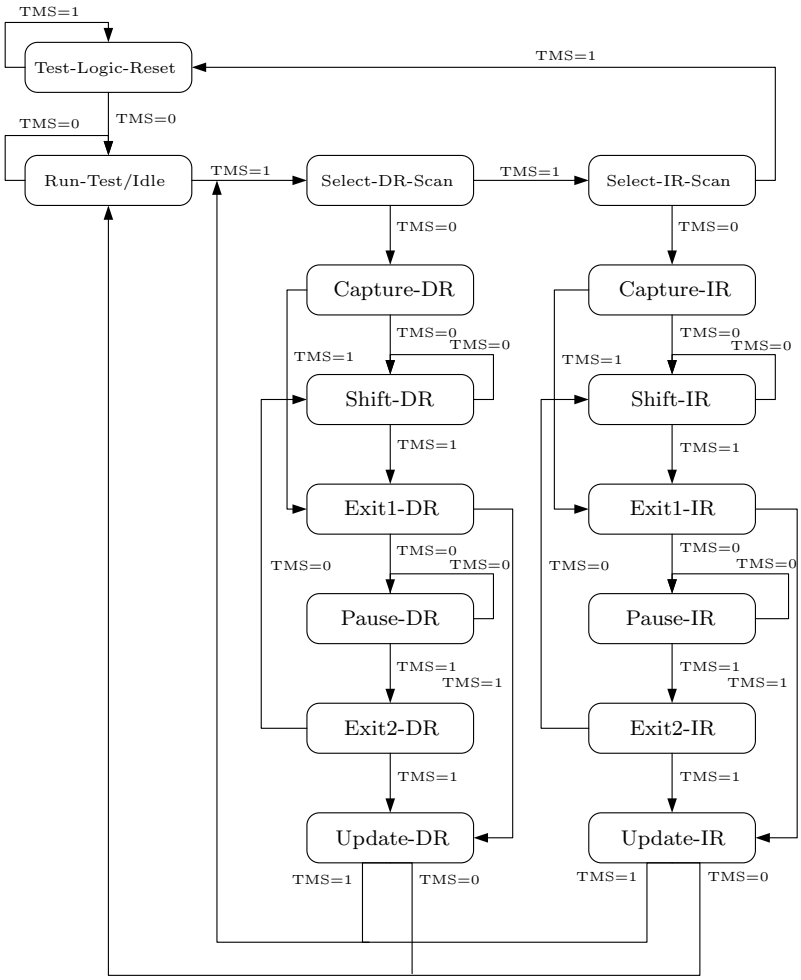


Figure 2.7: TAP state machine schema

`BYPASS` The `BYPASS` instruction, once shifted into the instruction register, allows selecting the `BYPASS` register as data register. The standard binary code for this instruction should be all ones 111...1 This instruction does not interfere with the normal operation of the on-chip system logic and operations. The `BYPASS` instruction may have additional binary codes that must be documented. If no `IDCODE` instruction is available this register is selected by default in the *Test-Logic-Reset* state and connected to TDI and TDO.

`IDCODE` The `IDCODE` instruction, once shifted into the instruction register, allows selecting the device identification register as data register. When the manufacturer decides to implement the identification register in the design, the component must provide a `IDCODE` instruction. When the microchip is provided with the identification register, the default instruction loaded in the JTAG instruction register in the *Test-Logic-Reset* state is the `IDCODE`. This allows the user to perform a blind interrogation of the components soldered on a board. The IDCODE when selected, does not affect the normal operation of the on-chip system logic.

`USERCODE` Where the identification register is provided and the component is user programmable, the manufacturer should provide a `USERCODE` instruction if the firmware version cannot be determined in other ways by use of public

instructions. The `USERCODE` instruction when selected, do not affect the normal operation of the on-chip system logic.

Private

This instruction set is intended solely for the component manufacturer use. Instructions which are part of this set may be not documented on the data-sheet or in the BSLD (see Section 2.1.1). This instruction set allows gaining access to test features embedded in the design such like: design verification or fault diagnosis. Private instructions may be hazardous since are not documented and can cause damages to the component and the vendor need to identify the instruction binary codes that can cause hazardous operations.

2.4.4 Test data registers

JTAG standard prescribes a minimum set of mandatory data registers, below we will describe the two register useful for our purposes, the bypass and the boundary-scan registers. In addition to the two just mentioned registers, the standard allows manufacturer to implement also non mandatory or non standard registers, a optional test data register important for our purposes is the device identification register. The standard permit the extension of the base architecture with other registers to allow access to test particular functions of the IC. Additional registers may not be publicly documented by the manufacturer since they are restricted to public use. Each register has a fixed length and can be concatenated with other registers that are implemented in the design.

Bypass register The bypass register contains a single bit shift-register, providing a minimum length path from TDI to TDO. This register is generally used when the tester needs to perform tests in a chain of JTAG, allowing speeds access to test data registers in other components.

Boundary-scan register This register allows testing of the circuitry external to a component. Is used to test interconnections with other components that are not compliant with the standard or to test board connections. This registers also allows performing tests on the on-chip system logic simulating connections with external components.

Device identification register This optional device identification register can be selected by shifting into the JTAG instruction register the `IDCODE` instruction. This register, if implemented, contains the manufacturer, part number, and version of the IC allow determining through JTAG this information. One interesting application of this register is to identify the components mounted on a board via a single interface. The manufacturer identity is codified with 11 bits, the part number is codified with 16 bits and the version code is codified with 4 bits.

Chapter 3

Analysis of the problem and naive solution

In this chapter we will introduce the naive solutions of problem addressed by this work and the analysis of the two solutions. In Section 3.1 we give the problem definition. Following in Section 3.2 we give the brute-force naive solution and discuss the performance problems. The Section 3.3 will give an enhanced version of the previous algorithm and discuss the performance of the new version. Finally, in Section 3.4 we will present some hardware problems and their solution.

3.1 Problem definition

As introduced in Chapter 1, the objective of this work is to find, in an automatic manner, the set of pins associated with the JTAG signals (TCK, TMS, TDI, TDO and the optional TRST) in an unknown, unlabelled or undocumented IC. To address this problem we have identified two scenarios: in the first, the IC to analyze is de-soldered from the board, while in the latter we need to find the traces of the JTAG interface directly from a functional PCB. The first scenario generally allows the attacker to obtain better result since all nuisance factors of the PCB case are taken apart. As a matter of fact, the second scenario, several other components are soldered onto the board; the electrical characteristics of these components can disturb the search process. Furthermore, in the PCB case, the JTAG interface could not be exposed or directly accessible in the form of a connector. On PCBs, manufacturers, in order to limit the possibilities of reverse engineering attempts, they are used to removing plugs and other components (*e.g.,* resistors, jumpers, solder bridges, etc.) from the production mask of the PCB, thus also reducing production costs. The hardware configuration required for this attack consists of a *TargetIC* that is the IC on which we must find the port and an *AttackerIC* that is the one who leads the attack.

3.2 Naive solution $O(n^5)$

To address the problem already presented and allow a complete black-box approach, the objective can be achieved by exploiting the standard features of the JTAG port. The `BYPASS` instruction is a well-standardized instruction that, as the standard

prescribes, must be implemented, along with the bypass register, in every JTAG compliant device. JTAG standard defines the mandatory instruction code for the `BYPASS` instruction as all ones (11...11), and it defines also the length of the register selected by the `BYPASS` instruction. Bypass register provides a single bit shift register, this means that every bit shifted into this register is shifted out with a delay of one clock cycle. The fact just presented allows us to identify the JTAG port by selecting the bypass register and trying to shift into it a known sequence of bits. Since every bit shifted into this register will appear as output with a delay of one clock cycle, JTAG port can be found by shifting a sequence of bits into the TDI pin and if the same sequence is read on the output pin with a delay, this means that the pins used for the communication compose the JTAG port. To reduce the possibility of false positives, the sequence of bits shifted into TDI must be a random sequence. The randomness allows a certain grade of precision in the case in which the bits shown on the candidate TDO pin are not the bits shifted into TDI. For instance sequences like all ones, all zeros or alternated can lead to false positives when the *TargetIC* is running some code or sequences like all 1 or 0 are typical when the alleged set of candidate pins is wrong. The first and more trivial approach is a complete brute-force on all the five JTAG pins. Defining n as the number of pins of the *TargetIC*, the brute-force attack comprises all the combinations of five different pins. More formally the solution space of the naive problem is given by:$(TCK, TMS, TDI, TDO, TRST) \forall\, TCK, TMS, TDI, TDO, TRST \in PINS \mid TCK \neq TMS \neq TDI \neq TDO \neq TRST$ where $PINS$ is the set of all the pins of the *TargetIC*. The number of different solutions to the problem is given by $n(n-1)(n-2)(n-3)(n-4)$ This can be translated into an algorithm (Algorithm 1) with $O(n^5)$ time complexity. The total time required to complete the brute-force attack can be derived by calculating the time required to test a single candidate solution, that is, the time required to perform the following operations:

1. Reset the TAP controller: 5 TCK cycles, TDI sequence `11111`;
2. Put the state machine in Shift-IR state: 5 TCK cycles, TDI sequence `00110`;
3. Shift into the instruction register the correct (see below) number of ones for selecting the `BYPASS` register: x TCK cycles where x is the number of shifted bits;
4. Put the state machine in *Shift-DR* state: 6 TCK cycles, TDI sequence `001011`;
5. Shift into the instruction register the correct (see below) number of random bits for checking the response: y TCK cycles where y is the number of shifted bits.

The resulting formula to calculate the number of clock cycles is: $5+5+x+6+y = 16+x+y$ where x is given by the alleged length of the IR (typically less than 64 bits) and y is given by the alleged length of the TAPs chain (ICs can have an internal TAP chain which usually does not exceed the three units). When performing the search over PCBs we can estimate the number of TAP by counting the number of ICs additionally, the y value should be appropriate to ensure test precision and interference rejection. Suppose that we set $x = y = 32$, then the required number of clock cycles (K) is 80. With a TCK clock frequency (F) of 300 KHz the time T required to test a candidate solution is $T = K \times \frac{1}{F} = 80 \times \frac{1}{300\,\mathrm{KHz}} = 2.6 \times 10^{-4}\,\mathrm{s} =$

0.26 ms As an example, we can estimate the total time (T_{tot}) for testing an IC with 200 pins: the brute-force time required is $T_{tot} = T \times n^5 = 0.26\,\text{ms} \times 200^5 = 83\,200\,000\,\text{s} \approx 963\ days$. Clearly, this solution is close to unfeasible even for devices with low pin count.

3.3 Exploiting the directions $O(n^4)$

As shown before, a time complexity of $O(n^5)$ is close to unfeasible. In order to lower the time complexity, we can take advantage of the pins directions. The TDO pin is the only JTAG output, it can be exploited in order to reduce the solution space. TDO pin can be excluded by simultaneously read all the unused pins, reducing the solution space to $(TCK, TMS, TDI, TRST)$. This improvement can be achieved by setting all the GPIOs of the *AttackerIC*, that are not testing any other JTAG signal, in input mode. This allows attacker to simultaneously (*i.e.,* in a single TCK cycle) read the value of all the unused pins in order to spot any data output from TDO. By evicting the TDO pin, the resulting solution space is given by:$(TCK, TMS, TDI, TRST)\ \forall\ TCK, TMS, TDI, TRST \in PINS \mid TCK \neq TMS \neq TDI \neq TRST$ where $PINS$ is the set of all the pins of the *TargetIC*. The number of possible combinations is reduced and given by $n(n-1)(n-2)(n-3)$. This method brings the time complexity of the brute-force algorithm (see Algorithm 2) to $O(n^4)$ and, consequently, the time required to brute-force all the JTAG signals combinations is considerably decreased. The total time required for a 200 pin brute-force attack can be estimated by the following formula: $T_{tot} = T \times n^4 = 0.26\,\text{ms} \times 200^4 = 416\,000\,\text{s} \approx 4.8\ days$. This improvement enables a feasible, yet slow, attack on medium-small devices while it is still not applicable to larger ICs.

3.4 Tests and RESET Problem

To prove the correctness of the algorithms presented, we have performed a set of tests using the hardware in Figure 4.4. The test has been conducted on three main scenarios: an empty IC with JTAG and a programmed IC that sends as output on all the GPIO, except the JTAG pins, random signals in order to disturb the search process and a programmed IC with all the pins used as output with random signals. With the first test case has proven that the basic search works correctly and return the JTAG port pins. The second test has proven that the technique already presented is not biased by random signals coming from the *TargetIC*. The third test has proven that in some cases manufacturers do not design the JTAG port in full compliance with the standard. Standard prescribes that all the JTAG signal must have a dedicated pin on the package. Most manufacturers violate this rule and in order to save pins, they tend to share JTAG pins with other general purposes I/O functions. These design choices can raise the difficulty of automatically finding the JTAG pins, for instance requiring the IC to be put in reset state in order to halt the core and expose the JTAG port. Halting the core or bringing the IC into reset state, is required when the manufacturer re-targeted the JTAG pins to other purposes. This implies that only when the core is in halt or reset state the JTAG port is accessible.

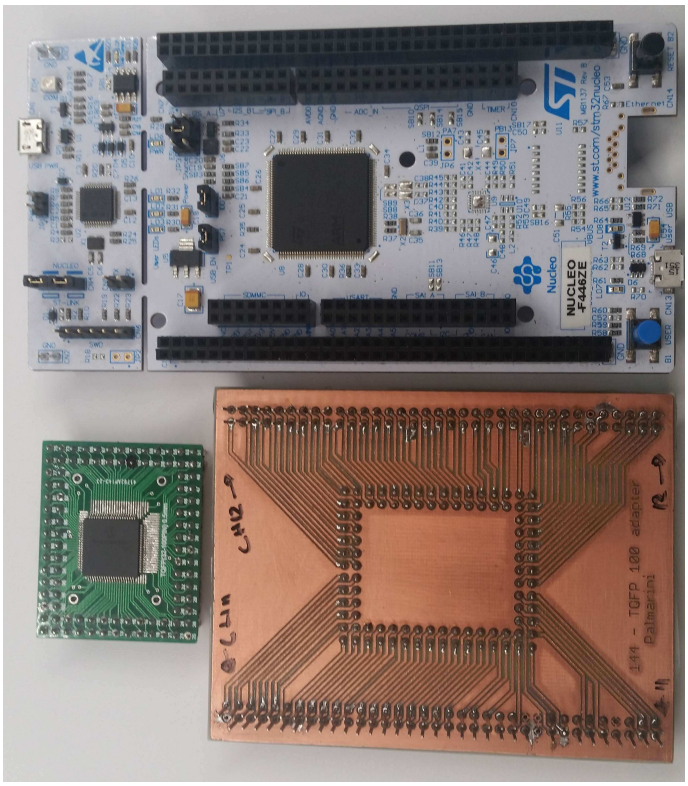


Figure 3.1: Test configuration, on the right the Nucleo board; on the top left the target IC soldered on the adapter board; on the bottom left custom adapter to match the Nucleo and target adapter.

Solution This problem can be solved with a simple shrewdness, without adding the RESET signal to the solution space and consequently rise the time complexity. Usually ICs, during the power on, bring the core in reset state for a small amount of time (typically 1 or 2 milliseconds), before loading the bootloader and subsequently the firmware. This short time window can be exploited in order to test a single possible solution. By powering on and off and on the IC before every test, the time complexity remains unchanged and only a small delay is added.

```
Let PINS[0...n-1] the array of all pins;
Let Sequence[0...y-1] the sequence of bit to shift into DR;
foreach TCK ∈ PINS do
    foreach TMS ∈ PINS ∧ TMS ≠ TCK do
        foreach TDI ∈ PINS ∧ TDI ≠ TMS ≠ TCK do
            foreach TDO ∈ PINS ∧ TDO ≠ TMS ≠ TCK ≠ TDI do
                foreach TRST ∈ PINS ∧ TRST ≠ TCK ≠ TMS ≠ TDI ≠ TDO
                do
                    Set TCK, TMS, TDI and TRST as output;
                    Set TDO as input;
                    Set TRST low or high ;     // low if TRST is active high,
                     high if TRST is active low
                    Set TMS high ;    // Start reset the JTAG state machine
                    for i = 0; i < 5; i++ do
                        Set TCK to high;
                        Set TCK to low;
                    end
                    Set TMS low ;        // End reset the JTAG state machine
                    Move the JTAG state machine to Shift-IR;
                    Set TDI high ;                // Start shifting ones in IR
                    for i = 0; i < x; i++ do
                        Set TCK to high;
                        Set TCK to low;
                    end
                    Set TDI low;                    // End shifting ones in IR
                    Move the JTAG state machine to Shift-DR;
                    /* Start shifting DR                                    */
                    foreach bit ∈ Sequence do
                        Set TDI to the level of bit ;
                        Set TCK to high;
                        Set TCK to low;
                        Read TDO, compare the value with prevbit, TDO if TDO
                         read equal to prevbit mark as TDO else mark as non TDO;
                        prevbit = bit;
                    end
                    /* End shifting DR                                      */
                end
            end
        end
    end
end
```

Algorithm 1: Pseudo-code of the naive $O(n^5)$ JTAG brute-force algorithm

```
Let PINS[0...n-1] the array of all pins;
Let Sequence[0...y-1] the sequence of bit to shift into DR;
foreach TCK ∈ PINS do
    foreach TMS ∈ PINS ∧ TMS ≠ TCK do
        foreach TDI ∈ PINS ∧ TDI ≠ TMS ≠ TCK do
            foreach TRST ∈ PINS ∧ TRST ≠ TCK ≠ TMS ≠ TDI do
                Set all pin ∈ PINS as input with pull-down;
                Set TCK, TMS, TDI and TRST as output;
                Set TRST low or high ;  // low if TRST is active high, high
                 if TRST is active low
                Set TMS high ;        // Start reset the JTAG state machine
                for i = 0; i < 5; i++ do
                    Set TCK to high;
                    Set TCK to low;
                end
                Set TMS low ;           // End reset the JTAG state machine
                Move the JTAG state machine to Shift-IR;
                Set TDI high ;                    // Start shifting ones in IR
                for i = 0; i < x; i++ do
                    Set TCK to high;
                    Set TCK to low;
                end
                Set TDI low;                        // End shifting ones in IR
                Move the JTAG state machine to Shift-DR;
                /* Start shifting DR                                        */
                foreach bit ∈ Sequence do
                     Set TDI to the level of bit ;
                    Set TCK to high;
                    Set TCK to low;
                    Read TDO, compare the value with prevbit, mark pin as
                      candidate TDO if TDO read equal to prevbit;
                    prevbit = bit;
                end
                /* End shifting DR                                          */
            end
        end
    end
end
```

Algorithm 2: Pseudo-code of the naive $O(n^4)$ JTAG brute-force algorithm

Chapter 4

Efficient algorithms based on 4-state GPIO

In this chapter, we will present three new algorithms that exploit electrical properties to reduce the time complexity. In Section 4.2 we give the basic electronic background necessary to understand the electrical properties exploited by the new algorithms. Following in Section 4.3 is present the new algorithm that exploits the electrical properties in order to reduce the time required to accomplish the task. In Section 4.4 we give an enhanced version of the previous algorithm and discussed performances, problems and limitations of these solutions. Finally in Section 4.6 is presented a randomized algorithm that exploits the electrical properties described in Section 4.2.

4.1 Problem definition and base idea

In the previous section (Chapter 3), has been described and tested two algorithms that solve the problem. Nevertheless, the naive algorithms already presented, have a high time complexity. In this section will be presented a novel and efficient algorithm for the automated discovery of a mapping between JTAG signals and the matching pins of an unknown IC package. In order to accomplish the result, in this section will be presented a set of algorithms that exploit the physical nature of the problem by augmenting the number of states that an output pin can assume. In digital electronics, bits are represented by voltage levels. Usually, the bit 0 is represented by the logic level low that is typically `GND`; the bit one is represented by the logic level high that is `VDD` (*e.g.,* 1.8V,3.3V,5V). Usually, IC pins can be set only as input or output and a pin set as input cannot be used as output and vice versa. The new solution will exploit electrical properties by varying the electric impedance of a pin we can simultaneously "set" and "read" its value. This implies that a pin can drive the line with a weak high / low and at the same time read the same logic level or a different one. The use of this configuration in the *AttackerIC* enables us to collect more information about the pins of the device under test, reducing the solution space. On *AttackerIC* the configuration presented above can be achieved by using the pull-up/down resistors built into every GPIO pin. In the next section will be briefly described the electronic principle on which the method already introduced is based.

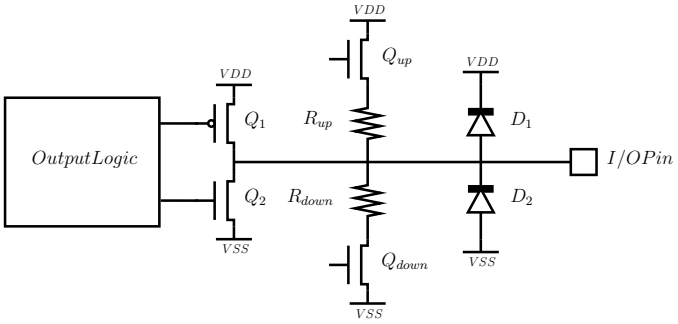


Figure 4.1: Schematic of a GPIO output drive circuitry. On the left the output transistors, at the centre the internal pull-up/down logic and on the right two protection diodes.

4.2 Electronic background

In order to fully understand the novel technique, in this chapter, we will present the basic electronic background. Firstly it is described the internal design of a GPIO port. In Figure 4.1 is depicted the schematic of the output stage of a GPIO pin. In particular, we are interested in:

Output logic: this is responsible for controlling the two transistors Q_1, Q_2 that control output mode and direction of the GPIO. Driving only Q_1 puts the pin in *open-drain* configuration while driving both enables the *push-pull* output mode (*cf.* [12]). When Q_1 and Q_2 are non-conducting the pin is *floating* (input mode).

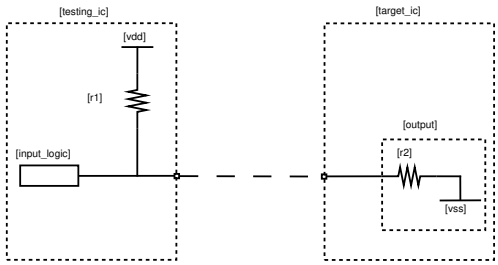
R_{up}: when the GPIO is in floating state and Q_{up} is conducting, R_{up} can *weakly* drive the line logic level one (`VDD`).

R_{down}: on the contrary, when Q_{down} is conducting, R_{down} can *weakly* drive the line logic level zero (`VSS` that is typically `GND`).

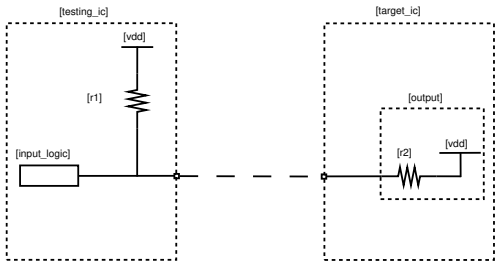
Thus, the impedance of a GPIO pin is software (firmware) controllable in three steps based on the output drive configuration: very low in open-drain or push-pull output mode, very high in floating input mode and a midrange value when the pin is in input mode and either R_{up} or R_{down} are enabled. The third mode can also be achieved, *e.g.,* using a similar configuration of external resistors but it would require a complementary external control logic, raising the circuit complexity. The value of the integrated pull-up/down resistors can be found in the component datasheet. We can safely assume that it varies from few $k\Omega$ to thousands $k\Omega$ as in the case of our *AttackerIC* where it is about 50 kΩ (*cf.* [29]).

4.2.1 Real case application

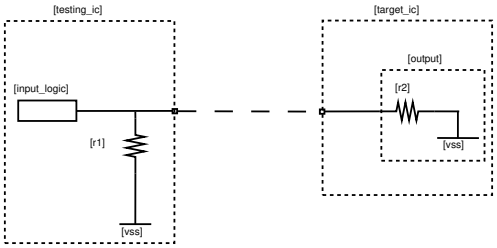
The basic principle of the method that we will describe is based on big difference of impedance between output pins and input pins. Usually, input pins have a bigger impedance than the output ones, and we can exploit this difference to reduce the number of solutions to our problem. For better understanding, this principle we can simplify the real environment in two main cases: target IC pin output mode and



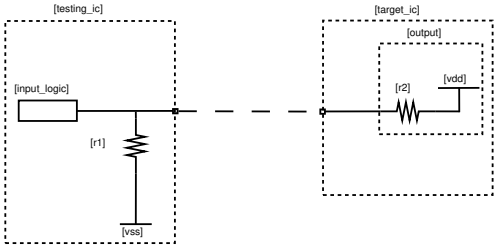
(a) *AttackerIC* pin with pull-up resistor enabled, *TargetIC* pin in output mode at logic level low



(b) *AttackerIC* pin with pull-up resistor enabled, *TargetIC* pin in output mode at logic level high



(c) Test with pull-down resistor and output low



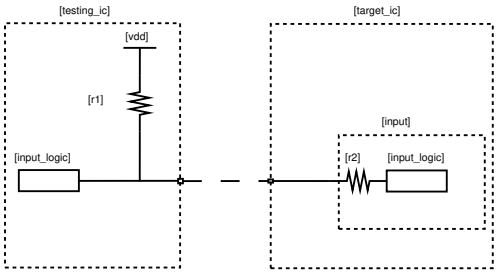
(d) Test with pull-down resistor and output high

Figure 4.2: Interaction between the *AttackerIC* pull-up mode and the two logic levels of a *TargetIC* pin in output mode

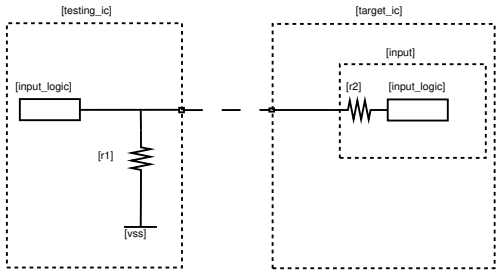
target IC pin input mode. Each case can be subdivided in sub case as can be seen from Figure 4.2 and Figure 4.3.

Target IC pin output mode

Now, each case where $TargetIC$ pin is in output mode can be discussed. Starting from case A, supposing a real scenario where the weak internal pull up resistor R_1 is setted. A typical value for R_1 is $40K\Omega$ (STM32F446). The target output pin in this case can be modelled with a resistor (R_2); a typical value of impedance for an output pin is $R_2 = 100\Omega$. The normal operating voltage of IC is $3.3V$. Having all the necessary parameters, the voltage that the $Inpultlogic$ see on the line can be calculated. As the first step must be calculated the equivalent resistance for the series of resistors, given by the sum of all impedances. $R_{TOT} = R_1 + R_2 = 40K\Omega + 100\Omega = 40100\Omega$; having the total resistance can be calculated the current flowing through the two resistors that is given by the Ohms law. $\Delta V = R \times I$ this implies that $I = \frac{\Delta V}{R} = \frac{3.3V}{40100\Omega} = 82\mu A$. Having the current can now be calculated the voltage drop given by R_2 and the value that $Inputlogic$ can read $\Delta V = R \times I = 82\mu A \times 100\Omega = 8.23mV$.
For the case B as can be seen from the Figure 4.2b both pins are connected to the $3.3V$ line so there is no voltage difference and no current flows between the two pins. Since there is no current, resistors do not affect the voltage level and the *inputlogic* read a stable value of $3.3V$. The case C is similar to case B, as can be seen from the Figure 4.2c both pins are connected to VSS (or GND) line so there is no voltage difference and no current flows between the two pins. Since there is no current, resistors do not affect the voltage level and the *inputlogic* read a stable value of $0V$.



(a) *AttackerIC* pin with pull-up resistor enabled, *TargetIC* pin in output mode at logic level low



(b) *AttackerIC* pin with pull-up resistor enabled, *TargetIC* pin in output mode at logic level high

Figure 4.3: Interaction between the *AttackerIC* pull-up mode and the two logic levels of a *TargetIC* pin in output mode

For case D, where the weak internal pull down resistor R_1 is setted and as in case A, a typical value for R_1 is $40K\Omega$. The target output pin ir represented by the resistor R_2; a typical value of impedance for an output pin is $R_2 = 100\Omega$. The normal operating voltage of IC is $3.3V$. Having all the necessary parameters, the voltage that the *Inpultlogic* see on the line can be calculated. $V_{\texttt{line}} = \frac{R_{down}}{R_{down}+R_{out}} \cdot \texttt{VDD} = 3.29\,\text{V}$.

Target IC pin input mode

Now we can discuss each case for $TargetIC$ pin in input mode. Starting from case A, supposing a real scenario where the weak internal pull up resistor R_1 is set. A typical value for R_1 is $40K\Omega$ (STM32F446). The input resistance is not specified in data-sheet and can be modeled by an infinite impedance resistor ($R_2 = \infty\Omega$). The normal operating voltage of IC is $3.3V$. We can now calculate the voltage that the *Inpultlogic* see on the line, first we calculate the equivalent impedance for the series of resistors that is given by the sum of all impedances. $R_{TOT} = R_1 + R_2 = 40K\Omega + \infty\Omega = \infty\Omega$; having the total resistance we can calculate the current flowing through the two resistors that is given by the Ohms law. $\Delta V = R \times I$ this implies that $I = \dfrac{\Delta V}{R} = \dfrac{3.3V}{\infty\Omega} = 0A$. Since there is no current flowing in the circuit there is no voltage drop on the resistors.
The other three cases B, C and D are similar to the case A that we have just seen. After seeing that there is no current flowing from the $TargetIC$ to the $TestingIC$, we can conclude that the voltage seen by the two *Inpultlogic* units is the voltage set by the pull-up/down resistor of the $TestingIC$. As can be noticed from the examples, there are cases in which the *AttackerIC* can read a value different from the one set by the pull-up/down resistors. This fact derives from the conversion of the voltage level of a signal when they are converted to logic levels 0 or 1. The hardware component that performs the conversions from voltage to logic level works with voltage thresholds that for $3.3V$ IC are:

- For the logic level 0 is $0V \rightarrow 0.8V$;
- For the logic level 1 is $2V \rightarrow 3.3V$.

For the output case A we can notice that the *Inputlogic* is reading a logic low ($8.23mV < 0.8V$) while the pull-up resistor is enabled. We can notice the same

thing in case C where the pull-down resistor is set to low and *Inputlogic* is reading a logic high. This property can now be exploited by the algorithms presented below in order to reduce the solution space and/or the time complexity. In the sections below will be presented algorithmic improvements for the naive algorithm and also randomized algorithms that exploit the just described properties.

4.3 Base implementation $O(n^3)$

The internal weak-pull mechanism of a GPIO pin can be exploited to achieve the aforementioned four-states configuration. Setting a GPIO pin of the *AttackerIC* to input mode with pull-up/down resistors enabled simultaneously, allows the attacker to (weakly) set the logic level of the line and to read back its actual value. Indeed, the moderately high resistance of the internal pull-up/down resistors sets the logic level of the line only when connected to an input pin on the *TargetIC*. On the contrary, when a weak-pull *AttackerIC* pin is connected to a strong push-pull output *TargetIC* pin, the former has almost no influence on the line voltage as seen in Section 4.2. With this premise, in this improved solution, we generate all the signals by enabling and disabling the pull-up/down resistors while operating every *AttackerIC* pin in input mode. As a result, we can remove TCK from the solution space entirely. Once selected a 3-tuple solution (TDI, TMS, TRST), all the remaining *AttackerIC* GPIOs are driven (and acts) as if they were all TCK clock signals. Within the same clock cycle, a TDO signal is searched among all the clock-acting pins. A difference between the set and read value of a pin is caused by the corresponding pin of the *TargetIC* that is transmitting data out, altering the logic level of the line. The pseudocode for this technique is presented in Algorithm 3. Being limited to TDI, TMS, and TRST, the time complexity of our improved solution is $O(n^3)$, that is at least one order of magnitude lower than the trivial brute-force attack.

4.4 Further optimization $O(n^2)$

Under the assumption that the *TargetIC* does not expose the JTAG TRST line, this algorithm can be further optimized. In fact, we noticed that several ICs do not include the optional TRST pin. This design choice could either be driven by cost and space reduction as we found in low-end microcontrollers or, as in the case of high-performance and high pin-count ICs such as FPGAs and Cortex-A ARM cores, since the JTAG state machine can also be reset using other mechanisms (*cf.* [1]). Thus, if TRST is evicted from the solution space the time complexity of the attack is as low as $O(n^2)$ or two to three orders of magnitude lower than the trivial brute-force of Chapter 3. For example, when attacking a 200 pins IC the total time required for scanning all the possible combinations is $T_{tot} = T \times n^2 = 1.9\,\text{ms} \times 200^2 = 76\,\text{s}$. Interestingly, since the time required to complete this reduced attack is negligible an attacker can always perform it first. If no solutions are found, then the full attack (with TRST) can be carried out.

```
Let PINS[0...n-1] the array of all pins;
Let Sequence[0...y-1] the sequence of bit to shift into DR;
Set all pin ∈ PINS as input with pull-down;
foreach TMS ∈ PINS do
    foreach TDI ∈ PINS ∧ TDI ≠ TMS do
        foreach TRST ∈ PINS ∧ TRST ≠ TMS ∧ TRST ≠ TDI do
            Set TMS, TRST and TDI as output;
            Set TRST low or high ;  // low if TRST is active high, high if
             TRST is active low
            Set TMS high ;          // Start reset the JTAG state machine
            for i = 0; i < 5; i++ do
                Set all pin ∈ PINS to pull-up;
                Set all pin ∈ PINS to pull-down;
            end
            Set TMS low ;           // End reset the JTAG state machine
            Move the JTAG state machine to Shift-IR;
            Set TDI high ;          // Start shifting ones in IR
            for i = 0; i < x; i++ do
                Set all pin ∈ PINS to pull-up;
                Set all pin ∈ PINS to pull-down;
            end
            Set TDI low;            // End shifting ones in IR
            Move the JTAG state machine to Shift-DR;
            /* Start shifting DR                                    */
            foreach bit ∈ Sequence do
                Set TDI to the level of bit ;
                Set all pin ∈ PINS to pull-up;
                Set all pin ∈ PINS to pull-down;
                Read all PINS, compare the value with bit, mark pin as candidate
                 TDO if it differs;
            end
            /* End shifting DR                                      */
        end
    end
end
```

Algorithm 3: Pseudo-code of the improved JTAG brute-force algorithm $O(n^3)$

4.5 Real test case

Our novel technique reduces the solutions space to (TDI, TMS, TRST) and reduces the time complexity to $O(n^3)$ or $O(n^2)$. Although, we performed a set of tests using the hardware in Figure 4.4, Figure 4.5, and we have identified minor drawbacks and limitations of this technique. In particular, we faced a problem related to the GPIO input stage voltage thresholds. Since the voltage is continuous and logic levels are discrete, the GPIO has two threshold voltage that sets the lower/upper bounds for low/high logic levels. The value of the integrated pull-up/down resistors in *AttackerIC* does not allow voltage to reach the higher bound. Since the threshold is dependent on the power supply voltage, we circumvent this issue by lowering `VDD` to about 2 V.

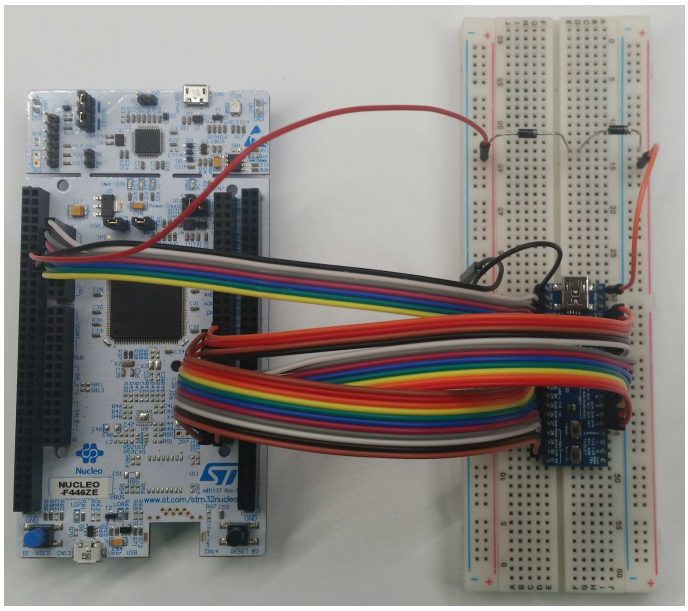


Figure 4.4: Test configuration with nucleo board connected to STM32F103 powered through two diodes for lowering the power voltage level.

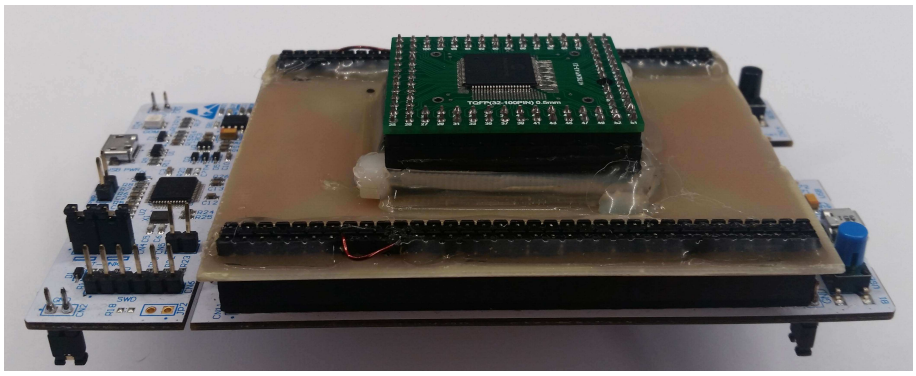
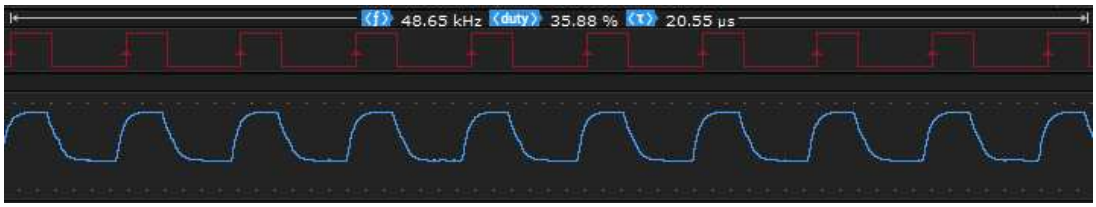


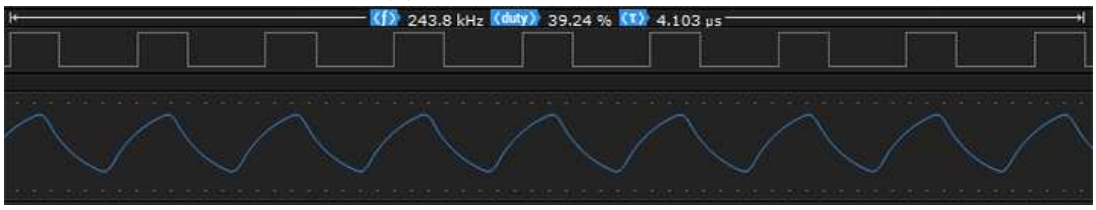
Figure 4.5: Test configuration with *TargetIC* stacked onto the custom adapter and nucleo.

4.5.1 Clock deformation problem

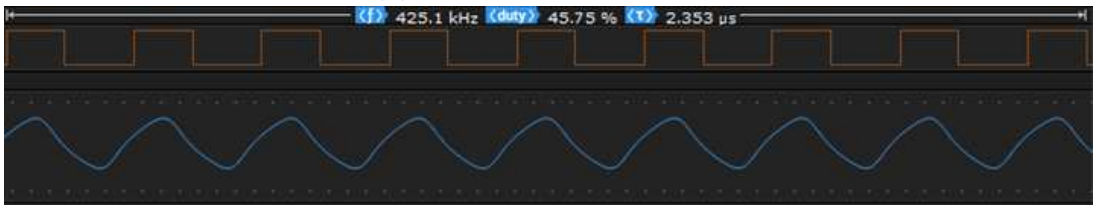
An additional problem caused by the integrated pull-up/down resistor values is related to the shape of the generated square wave. In particular, we noticed that increasing the TCK clock speed can significantly alter the shape of the signal. In fact, fast digital signals require a low impedance output stage in order to keep the rise and fall times of the square wave within an acceptable range. In Figure 4.6a and Figure 4.6b we can see the effect of raising the TCK frequency from 50 kHz to 200 kHz to the edges of the square wave. In Figure 4.7 can be seen a more detailed representation of the TCK waveform used during tests at 50 MHz. We can notice the high rise time caused by the increased impedance.



(a) 50 kHz TCK waveform



(b) 200 kHz TCK waveform

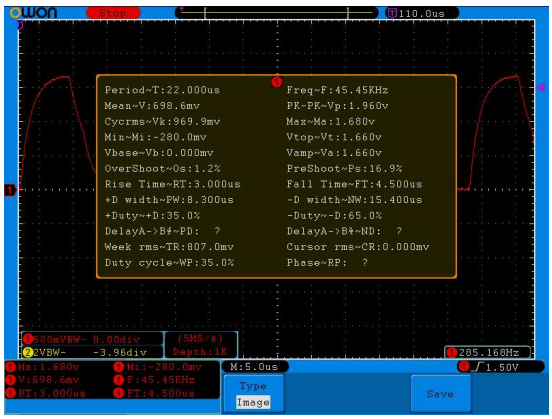


(c) 400 kHz TCK waveform

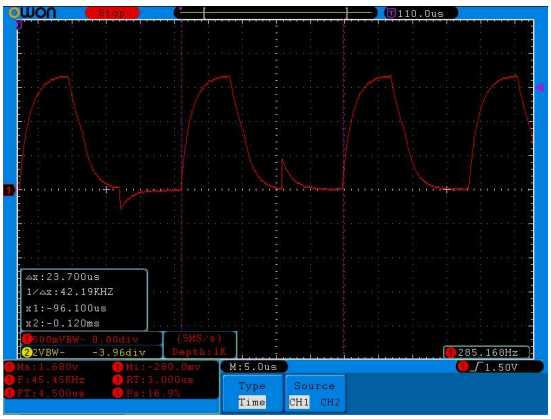
Figure 4.6: Screen-shot from logic analyzer showing square wave deformation, effect of the increase in frequency for TCK signals generated with GPIO operating in pull-up/down output mode

4.6 Randomized algorithms

In some occasions, especially with ICs with a high number of pins, *e.g.,* 2000, the brute force technique with all the optimization described hitherto, can be too expensive in terms of time. Randomized algorithms combined with the properties described in Section 4.1, can be used in order to reduce the time complexity partially releasing it from the number of pins. The most suitable class of randomized algorithms that allows the attacker to solve the problem with an improvement in time complexity is Monte Carlo algorithms. This class of algorithms, always guarantee an answer to the problem (the answer is correct with a probability p), run in a specified time and is the most advisable in the case in which there is no certainty that the *TargetIC* has a JTAG port. The problem requires finding five pins associated to signals $(TCK, TMS, TDI, TDO, TRST)$. By exploiting the 4-state technique the set of pins can be divided into four subsets, one for each signal excluded TDO that can be found by reading all the GPIO simultaneously as described before. The randomized algorithm will generate four random subset A,B,C,D from the set of all the *TargetIC* pins $PINS$ such as $A \cup B \cup C \cup D = PINS \wedge A \cap B \cap C \cap D = \varnothing$. Each set will be associated to a signal (*e.g.,* set A will transmit the TCK signal, set B will transmit the TDI signal, etc.) and each signal will be transmitted over all the pins that belong to the set. The algorithm will succeed when the *TargetIC* pins that belong to the JTAG ports are each one in a different set, in all the other cases the JTAG port would not respond. This can be translated into a problem where there are 4 boxes and 4 elements that must be distributed among the five boxes. There are no constraints on the number of elements in each box meaning that a box can



(a) Oscilloscope wave parameters screen-shot



(b) Detailed waveform deformation seen from oscilloscope

Figure 4.7: Screen-shots from Owon DS7102 oscilloscope showing wave deformation and wave parameters

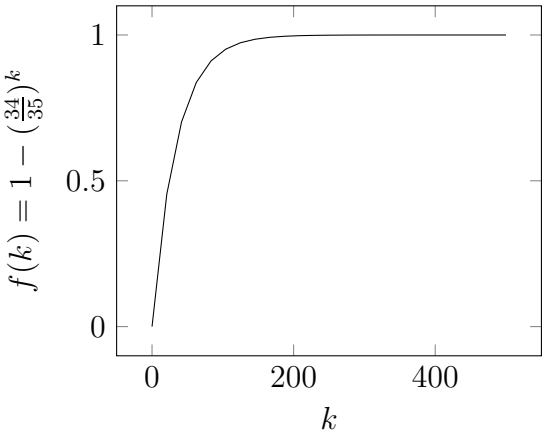
contain all the 5 elements or no elements or the other combinations. This problem can be seen as a combinatorial problem where the total number of possible combinations is given by $\binom{n+k-1}{n-1} = \binom{n+k-1}{k} = \frac{(n+k-1)!}{(n-1)!k!}$ where $n = k = 4$ is the number of different signals (*i.e.,* TCK, TMS, TDI, TRST), the resulting number of combinations is 35. Having the total number of possible combinations and knowing that the favorable case is the one in which the four signals are each one in a different set. The probability of the event already described is given by $Pr[allindifferentsets] = \frac{1}{35}$. The resulting Monte Carlo algorithm, that can be seen in Algorithm 4, has a probability of success that is dependent by the number k of iterations, the probability of finding the JTAG port is given by $Pr[findJTAG] = 1 - (\frac{34}{35})^k$. Time complexity depends not only from the parameter k but also on the number n of pins of the *TargetIC*. To amortize the time complexity $O(n)$ given by the generation of the four random sets, sets A,B,C,D can be assigned in turn to all the different signals, permit reducing the number of iterations k. The resulting time complexity is superiorly limited by $O((4! + n) \times k)$. The algorithm presented above, does not return the precise pins of the JTAG port but four sets associated to the four signals. A further step is required in order to find the exact combination of pins. To identify the pins associated with each signal can be used a binary search over each set. The key idea (Algorithm 5) is to select one set at a time and recursively split it into two subsets and drive the associated signal on only on one subset a time. By checking the response on the output the algorithm can decide which subset must be further split until the subsets are composed of only one element. This second part of the algorithm has a time complexity of $O(4 \log_2 n)$ that joined with the first part brings the total time complexity of this solution to $O(((4! + n) \times k) + 4 \log_2 n)$

Estimation of k The Monte Carlo algorithm already described, will always give an answer for the problem, but not always the correct one. The precision is based on the probability of correctness, the algorithm is p-correct when it gives a correct answer with probability p. In order to obtain a sufficient grade of reliability and simultaneously keep the time complexity as low as possible, the parameter k must be

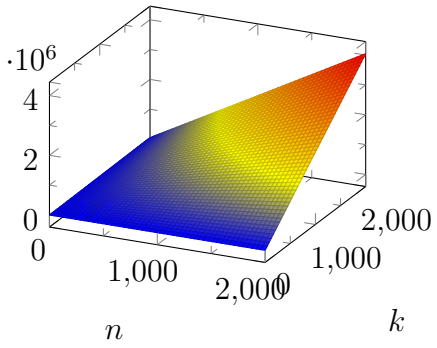
```
Let PINS[0...n-1] the array of all pins;
Let Sequence[0...y-1] the sequence of bit to shift into DR;
Set all pin ∈ PINS as input with pull-down;
Set k = 200 ;      // 250 cycles allow ≈99.9% propability of finding JTAG
 port if exists
while k >0 and JTAG not found do
    Random Generate sets A, B, C, D from PINS;
    foreach TCK ∈ A,B,C,D do
        foreach TMS ∈ A,B,C,D | TDI ≠ TMS ≠ TCK do
            foreach TDI ∈ A,B,C,D | TDO ≠ TMS ≠ TCK ≠ TDI do
                foreach TRST ∈ A,B,C,D | TRST ≠ TCK ≠ TMS ≠ TDI do
                    Set all pin ∈ TRST low or high ;  // low if TRST is active
                     high, high if TRST is active low
                    Set all pin ∈ TMS high ;    // Start reset the JTAG state
                     machine
                    for i = 0; i < 5; i++ do
                        Set all pin ∈ TCK to pull-up;
                        Set all pin ∈ TCK to pull-down;
                    end
                    Set all pin ∈ TMS to pull-up ; // End reset the JTAG state
                     machine
                    Move the JTAG state machine to Shift-IR;
                    Set all pin ∈ TDI high ;       // Start shifting ones in IR
                    for i = 0; i < x; i++ do
                        Set all pin ∈ TCK to pull-up;
                        Set all pin ∈ TCK to pull-down;
                    end
                    Set all pin ∈ TDI to pull-down; // End shifting ones in IR
                    Move the JTAG state machine to Shift-DR;
                    /* Start shifting DR                                    */
                    foreach bit ∈ Sequence do
                        Set all pin ∈ TDI to the level of bit ;
                        Set all pin ∈ TCK to pull-up;
                        Set all pin ∈ TCK to pull-down;
                        Read all PINS, compare the value with bit, mark pin as
                         candidate TDO if it differs;
                    end
                    /* End shifting DR                                      */
                end
            end
        end
    end
end
```

Algorithm 4: Pseudo-code of the Monte Carlo JTAG algorithm

estimated. The mathematical function that correlates the accuracy of the algorithm with k is the $Pr[findJTAG] = 1 - (\frac{34}{35})^k$. As can be evicted from the plot in Figure 4.8a, for low values of k (from 0 to 150) the precision is low; although for higher values of k the precision reaches almost one. To keep time complexity low, a trade-off between precision and the value k must be evaluated. Since from values



(a) Graph of the probability of finding JTAG port in function ok k



(b) Time complexity graph of randomized algorithm in function of k and n

of k higher than 200 the increase of precision is negligible, the maximum number of tests can be fixed to 200.

Performance evaluation The performance of the algorithm presented hitherto given by the parameters k and n and described by $O(((4! + n) \times k) + 4 \log_2 n)$ (see Figure 4.8b). After setting the parameter k, the time required to find the JTAG port on an IC can be calculated. The first part of the algorithm has a total time complexity of $O((4! + n) \times k)$. Nevertheless, the factor n can be omitted since do not increase the number of tests and it derives only from the generation of the sets that require a negligible time with respect to the time required to perform a single test. Supposing a clock frequency of 50 KHz, the required number of clock cycles (K) is 80, the time T required to test a candidate solution is $T = 1.9$ ms. The total time T_{fst} required to run the first part is given by $T_{fst} = T * (4! * k) = 0.0019 \times 4800 = 9.12$ s. The time complexity of the second part of the algorithms is given by $O(4 \log_2 n)$, this also the number of tests required to find the four input pins among the four sets. As the first part, with clock frequency of 50 KHz and a *TargetIC* with 200 pins, the total time T_{snd} required to run the second part is given by $T_{snd} = T * (4 * log_2 n) \approx 0.0019 \times 30 \approx 0.057$ s. The total time required to find the JTAG port $T_{tot} = T_{fst} + T_{snd} = 9.12 + 0.057 = 9.177$ s.

Let $TRST, TCK, TMS, TDI$ the subsets of $PINS$ associated with the respective signals;
Let TDO the output pin of JTAG port;
Let $Sequence\,[0 \dots y-1]$ the sequence of bit to shift into DR;
Set all $pin \in \mathcal{PINS}$ as input with pull-down;
foreach $(set, signal) \in \mathcal{TRST}, \mathcal{TCK}, \mathcal{TMS}, \mathcal{TDI}$ **do**
  FindRecursive($set, signal, TDO$);
end

Function *FindRecursive (set, signal, TDO)*
  Split *set* in *setL* and *setH*;
  Set *signal* set as *setL*;
  return = PerformTest($TRST, TCK, TMS, TDI$);
  if $|setL| == 1$ *and return* **then**
    return *setL*;
  end
  if *return* **then**
    FindRecursive(*setL*, *signal*, *TDO*);
  end
  else
    FindRecursive(*setH*, *signal*, *TDO*);
  end

Function *PerformTest* ($TRST, TCK, TMS, TDI, TDO$)
  Set all $pin \in \mathcal{TRST}$ low or high ; `// low if TRST is active high, high if TRST is active low`
  Set all $pin \in \mathcal{TMS}$ high ; `// Start reset the JTAG state machine`
  for $i = 0; i < 5; i++$ **do**
    Set all $pin \in \mathcal{TCK}$ to pull-up;
    Set all $pin \in \mathcal{TCK}$ to pull-down;
  end
  Set all $pin \in \mathcal{TMS}$ to pull-up ; `// End reset the JTAG state machine`
  Move the JTAG state machine to Shift-IR;
  Set all $pin \in \mathcal{TDI}$ high ; `// Start shifting ones in IR`
  for $i = 0; i < x; i++$ **do**
    Set all $pin \in \mathcal{TCK}$ to pull-up;
    Set all $pin \in \mathcal{TCK}$ to pull-down;
  end
  Set all $pin \in \mathcal{TDI}$ to pull-down; `// End shifting ones in IR`
  Move the JTAG state machine to Shift-DR;
  `/* Start shifting DR */`
  foreach *bit* $\in \mathcal{S}equence$ **do**
    Set all $pin \in \mathcal{TDI}$ to the level of *bit* ;
    Set all $pin \in \mathcal{TCK}$ to pull-up;
    Set all $pin \in \mathcal{TCK}$ to pull-down;
    Read *TDO*, compare the value with *bit*, if it differs mark solution as working;
  end
  `/* End shifting DR */`
  if *solution marked as working* **then**
    return true;
  end
  else
    return false;
  end

Algorithm 5: Pseudo-code used to identify the specific pins in the subsets

Chapter 5

Comparison

In chapters Chapter 3 and Chapter 4 are described a series of algorithms that solve the problem of finding a JTAG port. Every algorithm has his pros and cons, in this chapter, we will compare the reliability and the performance in terms of time needed to find the JTAG port. In Table 5.1, are shown the execution times of the various algorithms related to the number of pins of the *TargetIC*. The clock frequencies used for calculating the times are: 100 KHz for the naive versions and 50 KHz for all other versions. The values of clock frequencies are carefully chosen to obtain the maximum speed, simultaneously maintain the maximum compatibility with all JTAG ports and to reduce the clock deformation. Naive version with time complexity $O(n^5)$ must be considered only as a reference to estimate the performance improvements of the various algorithms. The algorithm that guarantees the better compatibility for searching the JTAG port on both PCBs and desoldered ICs is the naive $O(n^4)$. This version is the only usable technique when the search process must be performed directly on PCBs; in order to reduce the time required to find the JTAG port, in Chapter 6 will be described some techniques to reduce the number of pins on which is performed the brute force. The major improvements in terms of execution time are better appreciable with the algorithms that exploit the 4-states GPIO. These algorithms in spite of the reduction of the clock frequency run much faster than the naive versions. The Figure 5.1 shows the execution times of the algorithms. As can be noticed the randomized solution implemented using the Monte Carlo algorithm described in Section 4.6, is not always the best solution. For *TargetIC* with low pin count is preferable to use the algorithms described in Section 4.3 or Section 4.4. The trade-off between the brute force algorithms that exploit the 4-state GPIO and the randomized algorithm can be calculated using the intersection between the time complexity functions. To obtain the maximum efficiency with $O(n^2)$ and randomized algorithm, for *TargetIC* that ave less than 70 pins is convenient to use the $O(n^2)$ algorithm, with a higher number of pins is recommended to use the randomized algorithm. Moreover, the threshold between $O(n^3)$ and the randomized algorithm is lowered to 17 pins. From the threshold just calculated, we can conclude by giving a guideline to obtain the maximum performance and precision. Since the TRST signal is optional and not all the manufacturers implements it, our suggestion is to try with the pull-up/down algorithm that is the fastest one and only in the case that this method fails to proceed with the normal brute force. The random algorithm has a semi-constant execution time, this can be used as the first trial but the correctness of the result is related to k. If is needed 100% accuracy firstly

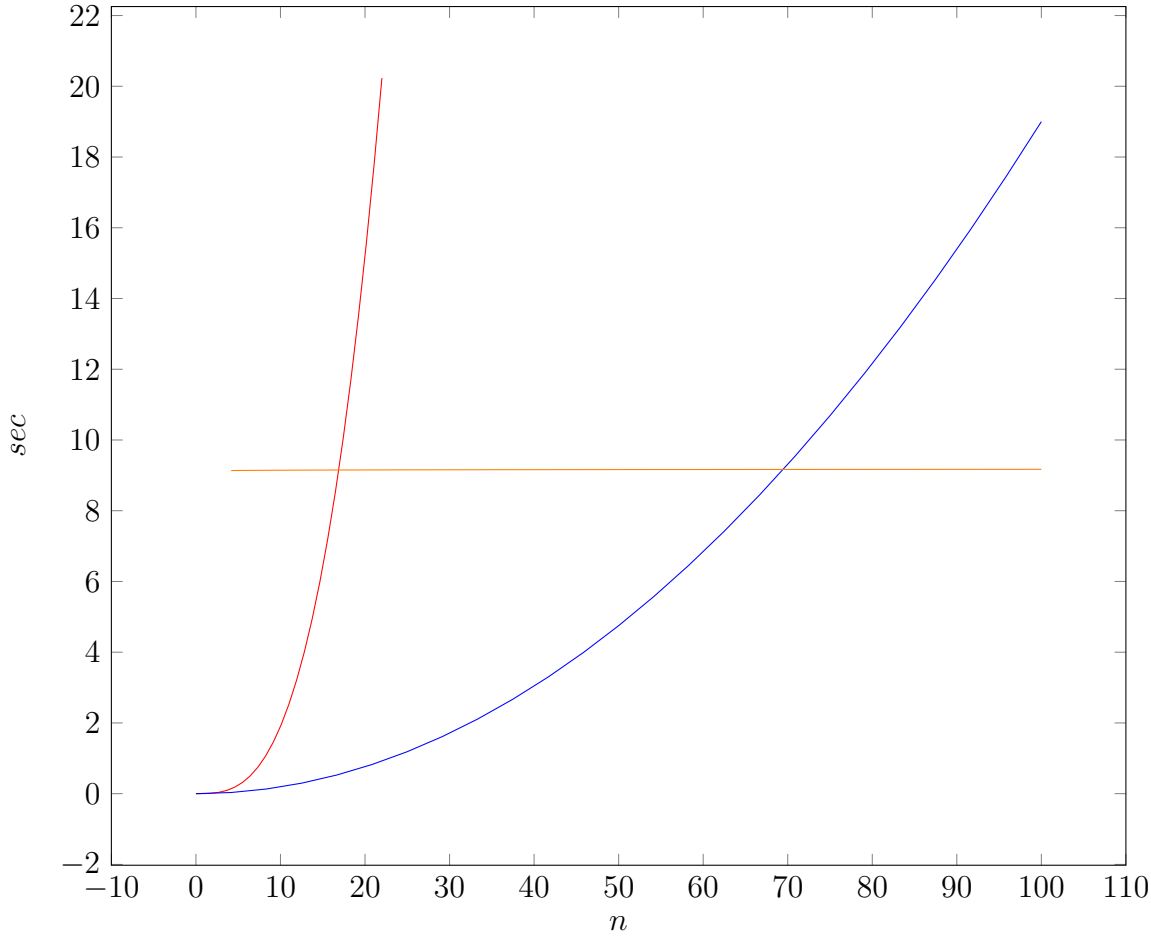


Figure 5.1: Comparison of execution times between the improved algorithms and the random solution

can be executed the $O(n^2)$ algorithm, that does not use include the TRST signal, and if it does not succeed the tester can use the more complete test $O(n^3)$, that include the TRST signal, to obtain the maximum accuracy and performance. This allows us to save time in the case where TRST is not implemented. Otherwise, if TRST is implemented in the design, the time spent for running the $O(n^2)$ algorithm is negligible in comparison to the time that we will spend on the $O(n^3)$ algorithm or on the basic brute force solution. From the discussion above can be wrongly deducted that the $O(n^4)$ algorithm describe in Chapter 3 is not usable in practice. Although the base brute force solution is useful when the tests must be performed directly on a PCB. In this case, the 4-state GPIO technique is not guaranteed to work, since the PCB includes other components along with the *TargetIC* that can modify the electrical properties of the pins.

Table 5.1: Comparison of the algorithms execution times; 100 MHz TCK frequency for Naive, 50 MHz TCK frequency for Improved and Random

# of pins	Naive $O(n^5)$					Naive $O(n^4)$				
	Y	D	H	M	S	Y	D	H	M	S
8	0	0	0	0	5	0	0	0	0	1
16	0	0	0	6	59	0	0	0	0	34
44	0	1	4	57	36	0	0	0	43	26
64	0	8	11	19	13	0	0	3	23	19
80	0	26	17	4	1	0	0	8	26	6
100	0	83	15	40	1	0	0	20	54	47
144	1	169	10	53	38	0	3	19	37	14
200	7	262	9	20	3	0	14	8	59	11
300	59	223	17	0	5	0	73	12	13	10
500	777	0	5	0	9	1	206	18	49	57
700	4202	353	3	40	13	6	14	3	25	8
900	14813	240	21	0	17	16	194	13	58	43
1000	25115	15	6	40	19	25	78	19	19	55
2000	807718	335	13	20	38	404	244	2	39	50

# of pins	Improved $O(n^3)$					Improved $O(n^2)$			Random	
	D	H	M	S	MS	M	S	MS	S	MS
8	0	0	0	0	638	0	0	106	9	143
16	0	0	0	6	384	0	0	456	9	150
44	0	0	2	30	982	0	3	595	9	160
64	0	0	7	54	970	0	7	661	9	166
80	0	0	15	36	624	0	12	8	9	168
100	0	0	30	43	380	0	18	810	9	169
144	0	1	32	35	722	0	39	125	9	173
200	0	4	9	32	760	1	15	620	9	177
300	0	14	6	28	140	2	50	430	9	181
500	2	17	34	36	900	7	54	50	9	187
700	7	12	15	9	660	15	29	670	9	190
900	15	23	28	6	420	25	37	290	9	194
1000	21	22	11	43	800	31	38	100	9	194
2000	175	15	53	27	600	6	36	200	9	202

Chapter 6

Performing attacks on PCBs

In this chapter, we will present the techniques and problems of finding the JTAG port on PCBs. In Section 6.1 are presented the main problems of searching JTAG ports on PCBs. Following in Section 6.2 are described the techniques used to find the possible JTAG pins on a PCB and the problems that an attacker can encounter during this process.

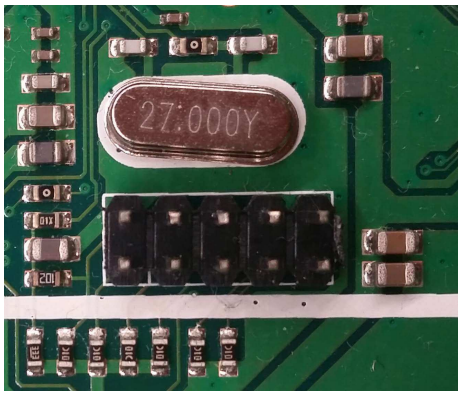
6.1 Introduction

In the real world, there are some cases where the *TargetIC* cannot be desoldered from the PCB for various reasons such as: chained JTAG components, other devices connected to the *TargetIC* or the necessity to not power off the target to preserve volatile memory. PCBs usually have more than one IC and commonly there are interconnected (*e.g.,* a micro-controller connected to an external flash memory or an FPGA connected to an audio processor); in this case, once the attacker gained the JTAG port control, can access the flash or the other components by using the boundary scan functions. Performing the attack directly on the PCB is useful also in the case in which the objective is the reverse engineer the entire board exploiting the chains of JTAG components. After this premise, we can now focus on how to find the JTAG port on PCBs. In order to achieve this goal, we must have a fully working PCB and an appropriate power source. PCBs offers many possibilities for connecting the *AttackerIC* to the target board, the most common are:

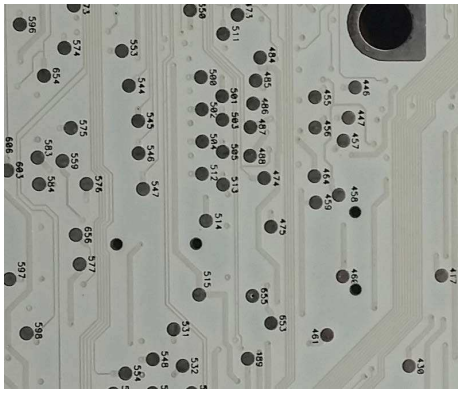
Populated connectors this is the easiest case if the PCB is equipped with populated connectors or ports as in Figure 6.1a the *AttackerIC* can be easily connected to the port using jumper wires.

Unpopulated connectors in this case, the manufacturer has removed the connector as in Figure 6.1d; usually is sufficient to solder pins or wires to the port in order to connect the *AttackerIC*. Sometimes manufacturer, to prevent unauthorized uses of the port, disconnects it from the rest of the PCB by removing resistors as can be seen in Figure 6.1c.

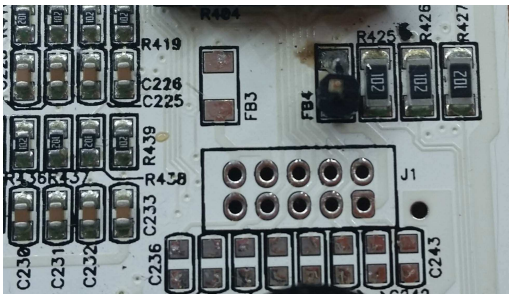
Test pads in this case test pads can be exploited by directly soldering on each one a wire that successively is connected to the *AttackerIC*, this method requires more time than the others, especially in the case in which the PCB has a high number of test pads.



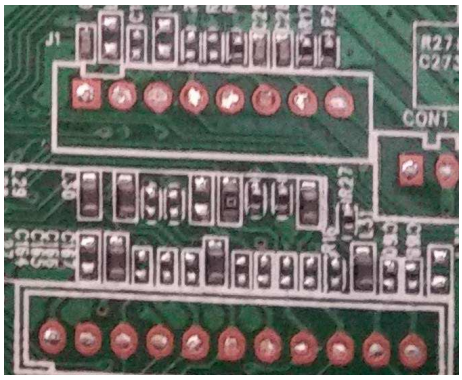
(a) Typical case of populated port



(b) Typical case of test pads (silver circles)



(c) Typical case of unpopulated and disconnected connector, manufacturer has intentionally removed resistors from the PCB



(d) Typical case of unpopulated port, pins has been removed by manufacturer after production

Figure 6.1: Photos of populated and unpopulated ports and test pads on PCBs

Direct soldering on pins when there are no connectors or test pad the only method to connect the *AttackerIC* to the PCB consists of directly soldering wires to the ICs. This is also the most accurate method since all the pins of the *TargetIC* can be connected to the *AttackerIC*. Soldering wires directly on the IC can be difficult when pins are small or impossible when the IC has BGA (see Section 6.1.1) pins.

6.1.1 IC package types

In some cases PCBs have no connectors or test pads. In this particular case the easiest solution for connecting *AttackerIC* to the target PCB, consists of soldering wires to the pins of the *TargetIC*. On the market there are several types of IC packages, each one with different characteristics and pin size. ICs packages can be subdivided in tree main categories:

- Through-hole packages that use holes drilled through the PCB for mounting the components;
- Surface mount that uses soldering pads on the surface of the PCB;
- Sockets that uses plastic or ceramic frames with electric connections to hold the IC (e.g. chip carrier or sockets for PGA/LGA chips).

After this brief introduction on packages, we can discuss the problems of each package type. The through-hole package is the most simple to treat, the pins are big and offer a good soldering surface. *AttackerIC* can be connected directly to the *TargetIC* by soldering wires on the pins and proceed like an unsoldered IC with some additional adroitness that will be described in the next sections. The socket for ICs usually makes use of through-hole technology. Differently, from through-hole ICs, the socket plastic frame exposes the pins from the bottom of the PCB. Wires that connect the *TargetIC* to the *AttackerIC* can be soldered on the socket pins that are exposed under the PCB. The main difficulties of this kind of ICs are the high number of pins and soldering required to fully connect the *TargetIC* to the *AttackerIC*. Nowadays the most common packages are the surface mount ones, there are different kinds of packages for SMT ICs.

Quad Flat Package (QFP) this package has pins around the four edges of the IC, pins are small leads and gaps between pins (pitch) usually vary from 1 mm to 0.4 mm. In this case, soldering wires directly on pins is more difficult but feasible also for small pitch. In this case, if soldering is not possible or it will require to much time, special clamps can be used in order to connect the *TargetIC* to the *AttackerIC*

Quad Flat No-leads (QFN) this package is similar to QFP, the QFM has no leads and soldering points are all around the corners. Wires can be soldered directly on the pads along the corners.

Small Outline Integrated Circuit (SOIC) this package is similar to the QFP but usually pins are only on two sides of the IC, has longer leads and the space between pins is bigger. In this case, we can use the same methods described for the QFP with the advantage of having longer and bigger leads.

Ball grid array (BGA) this package do not allow attacker to directly solder wires on the pins since it has contact points on the bottom of the IC, solder ball is applied on the pads of the IC than melted onto the pads of the PCB. In this case is impossible to directly solder wires on the IC pins and manufacturers usually provide test pads or connectors that exposes test features.

6.2 Finding the JTAG pins

After the brief introduction, in this section will be described the methods and the algorithms used to discover the JTAG port directly on PCBs. PCBs have other components along with the ICs that can alter the electrical properties of the pins since the 4-state GPIO technique is based on strict electrical properties that can not be guaranteed on PCBs, the most suitable algorithm for this application is the naive $O(n^4)$ algorithm described in Section 3.3. Having an algorithm with a high time complexity, there is the necessity of reducing the number of pins in order to keep the time, required to search the JTAG port, as low as possible. In order to reduce the number of possible pin combinations, the attacker can take advantage of some heuristics. Usually, on PCB pins or test pads that are part of the same debug port are grouped in a single connector/port or group of test pads. Taking advantage

of this property the number of combinations can be reduced by testing each PCB port or group of test pads separately.

In some cases, ports and test pads are disabled by the manufacturer. Usually, to prevent unauthorized use. During the production process JTAG IC pins are disconnected from test pads or ports (*e.g.,* example some solder bridges are left open or resistors are missing or removed). Another method that engineers use to lock JTAG ports, consists of strongly setting the voltage levels on the JTAG pins by adding some resistors in a way that if the attacker tries to connect a JTAG interface it is unable to communicate with the target. Moreover, in some cases manufacturers set the logic value of TRST pins, to set the JTAG state machine in the reset state. To spot these problems the attacker has to prior analyze the board and try to reverse engineer it in order to identify possible problems before executing the algorithm. The process of removing manufacturers locks varies case by case and there is no unique method to re-enable the ports. It may require some electronic background and soldering practices in the case that some connections are interrupted or some resistors need to be unsoldered. Before proceeding with the tests the attacker must take care of compatibility between *AttackerIC* and *TargetIC*. The most significant electrical characteristics are the voltage level, *AttackerIC* and *TargetIC* have to have the same voltage level (e.g. both use $3.3V$ or $5V$). If the voltage level of the target board differs from the level of the *TargetIC* a level shifter can be used in order to make them compatible without the risk of damaging the equipment and target.

6.2.1 More than one JTAG enabled chip

As stated in the introduction, JTAG offers a good entry point for PCBs reverse engineering. The JTAG standard is built to allow a complete analysis of all compatible IC soldered on a board. On one PCB can be more than one JTAG enabled chip and usually, they are chained together to reduce the number of pins required for connecting the board to the test equipment and the test time. By standard definition there are three ways to chain JTAG enabled ICs we can see it in Figure 2.6a, Figure 2.6b and Figure 6.2. All the algorithms described in the previous chapters are compatible with the three configurations, nevertheless, some modifications must be taken into consideration.

Serial connection using one TMS signal

The most common configuration is composed by: common TCK and TMS; the TDO pin from the first TAP is connected directly to the TDI of the subsequent TAP as can be seen in Figure 2.6a. This configuration uses the smallest number of pins on a port or test pads and the algorithm has to find only the for or five standard signals. The algorithms described in Chapter 3 are able to detect this configuration but requires some small modifications. The algorithm that will be used to find this particular configuration must take into consideration the length of the TAP chain. Usually, when trying to discover the JTAG port on desoldered ICs using the BYPASS method, the JTAG chain has length one and each bit shifted into TDI is shifted out in TDO with a delay of one clock cycle. The algorithm must be adapted to detect delay greater than one clock cycle and must use a longer test sequence in such a way that it is long enough to guarantee a correct identification and stability. The such modified algorithm can also figure out the total length of the chain by

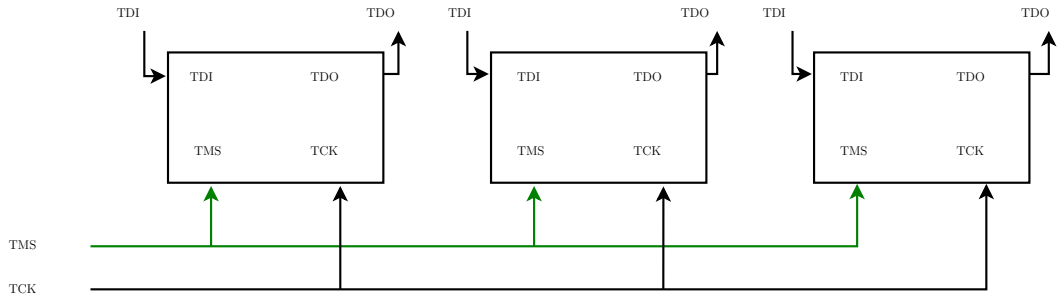


Figure 6.2: TAP interconnection with common TMS and TCK signals

counting the number of clock cycles of delay; this information is vital for reverse engineering the PCB.

Connection in two paralleled serial chains

This configuration is composed by: common TCK and dedicated TMS for each chain; the TAPs of each chain are arranged as in the pure serial connection described above. The algorithms described in Chapter 3 are able to detect this configuration but requires some small modifications. The algorithm that will be used to find this particular configuration must take into consideration the length of the TAP chain and the presence of more than one chain (also of different length). Chain length problem can be solved as shown in the paragraph above. To detect more than one chain, the algorithm must take into consideration that there can be more than one combination of pins that correspond to a different JTAG chain with the same TCK pin.

Multiple independent paths with common TMS and TCK signals

This configuration (see Figure 6.2) is composed by: common TCK and TMS; the TDI and TDO pin of each TAP are independent and separate. The algorithms described in Chapter 3 are able to detect this configuration but requires some small modifications. The algorithm that will be used to find this particular configuration must take into consideration the presence of more than one TDO and TDI. To detect all the TDI and TDO, the algorithm must take into consideration that there can be more than one combination of pins that correspond to a different JTAG port with the same TCK and TMS pins.

Chapter 7

Conclusions

The JTAG interface can expose valuable information about the attached IC, ranging from manufacturer's name, IC model and revision, to the flash memory content and the internal processor registers. During the reverse engineering process of an unknown, unlabelled or undocumented IC it is often required to correctly identify the pins associated to the JTAG interface. In this thesis, we have analyzed the problem of discovering of a mapping between JTAG signals and the corresponding pins using an automated procedure. We have described a naive solution based on a brute-force attack of the full set of pins combination of an IC package, and we have demonstrated the poor performance of this solution: $O(n^4)$ with n the number of tested pins. In order to reduce the time complexity of this task, we have presented a new and efficient algorithm that exploits the physical nature of the problem. In particular, our technique uses a combination of input/output, high/low impedance GPIO modes along with the integrated pull-up/down resistors to achieve a four-states GPIO configuration. Under reasonable assumptions, the time complexity of our algorithm is one to two orders of magnitude lower than the trivial solution. In this work, we have given an overview of the methods that allow us to find the JTAG port on an unknown/undocumented IC or PCB. We provided a set of algorithms and electronic techniques to speed up the process.

7.1 Identifying the IC model (if possible)

The JTAG standard offers to manufacturers the possibility of implementing an identification register. Once the attacker has found the JTAG port on the IC or on the PCB, he can exploit the standard to retrieve the identification code of an unknown chip. The standard prescribes that if the IDCODE instruction and consequently the identification register are implemented, the TAP controller must load by default in the identification register as data register when the state machine is in *Test-Logic-Reset*. This fact allows a blind identification of the model and manufacturer of the unknown IC. Having the identification code (32 bits), the attacker can use web services that convert the code into manufacturers and model information and subsequently retrieve valuable information such as the datasheet and if is available the BSDL document with all the specification of the JTAG port.

7.2 Security drawbacks

The techniques presented in this work has shown that debug port such as JTAG can be found also in case of unlabeled and undocumented ICs. Security through obscurity is not the correct method to secure debug ports on embedded devices. Once the attacker gained access to the JTAG port, the content of the IC's memory can be dumped and reverse engineering techniques can be applied in order to extract data and decompile the firmware. The attacker can also use software such as JREV [27] that allows he to reverse engineer the connections between the various ICs on the board.

Find hidden features The JTAG standard contemplates also the presence of hidden features, called private instructions. These instructions and their codes are not documented on datasheet or BSDL since they are intended only for manufacturer service uses. Once detected the pins combination of the JTAG port, an attacker can perform a blind search of all instructions implemented by the particular TAP and can discover private instructions. To perform this kind of attack described by Domke [11], can be used a software called UrJTAG that implements the algorithm that allows attackers to discover hidden or undocumented instructions. As stated by Domke [11], once the attacker has discovered the instruction codes of the hidden features, the process to figure out the functions of each instruction cannot be automated and is different for each IC.

7.3 Some attempts of securing JTAG

IC Manufacturers, in order to supply a security layer for ICs that implements JTAG, offers some functions that are capable to temporary or permanently disable JTAG ports. The most common techniques are e-fuses and disable bits; e-fuses allows programmers to permanently disable JTAG and are used only where the manufacturers do not have the necessity to service the devices through JTAG. Disable bits are a temporary method to disable JTAG ports, these bits can be reset only by fully erase the IC memory, this guarantee that the content of the IC internal memory could not be read via JTAG. The methods already described can be circumvented, e-fuses can be erased by trying to reprogram them with higher voltages or other techniques that are specific for each case; enable bits can be circumvented using glitching techniques that allows the attacker to skip the execution of the instructions that checks the state of the bits. In recent years researchers and manufacturers have developed and implemented some new technique to secure the JTAG port [7], [22], [23]. These new techniques, differently from fuses and enable bits, allow engineers to keep the JTAG port enabled and exposed, allowing manufacturers to service ICs and PCB more easily. In the light of recent studies, our techniques may seem subtle but, in spite of that, a lot of manufacturers continue to produce electronic device that does not implement these new defensive technologies, and for reducing production and support costs, they tend to continue to use old and untrusted techniques (*e.g.,* hiding the port on the PCB, or putting components that at first sight lock the JTAG access)to prevent unauthorized use of JTAG ports.

Bibliography

[1] Ieee standard test access port and boundary scan architecture. *IEEE Std 1149.1-2001*, 2001.

[2] Ronnie Brash. Bypassing security through side-channel attacks.

[3] Ing. M.F. Breeuwsma. Forensic imaging of embedded systems using jtag (boundary-scan). *Digital Investigation*, 2006.

[4] Marcel Breeuwsma, Martien De Jongh, Coert Klaver, Ronald Van Der Knijff, and Mark Roeloffs. Forensic data recovery from flash memory. *Small Scale Digital Device Forensics Journal*, 1(1):1–17, 2007.

[5] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *ACM Sigplan Notices*, 2014.

[6] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 2017.

[7] Christopher J Clark. Anti-tamper jtag tap design enables drm to jtag registers and p1687 on-chip instruments. In *Hardware-Oriented Security and Trust (HOST), 2010 IEEE International Symposium on*, 2010.

[8] Wikimedia Commons. Package on package asic plus memory pop schematic, 08 2008.

[9] Amitabh Das, Jean Da Rolt, Santosh Ghosh, Stefaan Seys, Sophie Dupuis, Giorgio Di Natale, Marie-Lise Flottes, Bruno Rouzeyre, and Ingrid Verbauwhede. Secure jtag implementation using schnorr protocol. *Journal of Electronic Testing*, 29(2):193–209, 2013.

[10] Eric DeBusschere and Mike McCambridge. Modern game console exploitation. Technical report, tech. rep., Department of Computer Science, University of Arizona, 2012.

[11] Felix Domke. Blackbox jtag reverse engineering. *Update*, 2009.

[12] Dhananjay V. Gadre and Sarthak Gupta. *Digital Input/Output*. 2018.

[13] S. Ghosh, A. Basak, and S. Bhunia. How secure are printed circuit boards against trojan attacks? *IEEE Design Test*, 2015.

[14] Swaroop Ghosh, Abhishek Basak, and Swarup Bhunia. How secure are printed circuit boards against trojan attacks? *IEEE Design & Test*, 32(2):7–16, 2015.

[15] F. Majeric, B. Gonzalvo, and L. Bossuet. Jtag combined attack - another approach for fault injection. In *2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, 2016.

[16] Microchip. *16-bit Digital Signal Controllers (up to 128 KB Flash and 16K SRAM) with Motor Control PWM and Advanced Analog*, 08 2011.

[17] S. Moein, F. Gebali, T. A. Gulliver, and M. W. El-Kharashi. Hardware attack risk assessment. In *2015 Tenth International Conference on Computer Engineering Systems (ICCES)*, 2015.

[18] Samer Moein, F Gebali, and Issa Traore. Analysis of covert hardware attacks. 2014.

[19] Johannes Obermaier and Stefan Tatschner. Shedding too much light on a microcontroller's firmware protection. In *WOOT*, 2017.

[20] Keunyoung Park, Sang Guun Yoo, Taejun Kim, and Juho Kim. Jtag security system based on credentials. *Journal of Electronic Testing*, 26(5):549–557, 2010.

[21] Luke Pierce and Spyros Tragoudas. Multi-level secure jtag architecture. In *On-Line Testing Symposium (IOLTS), 2011 IEEE 17th International*, pages 208–209. IEEE, 2011.

[22] Xuanle Ren, R. D. Blanton, and Vitor Grade Tavares. A learning-based approach to secure jtag against unseen scan-based attacks. *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2016.

[23] Xuanle Ren, Vitor Grade Tavares, and R. D. Blanton. Detection of illegitimate access to jtag via statistical learning in chip. *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2015.

[24] Renesas Electronics Corporation. *Hardware Manual Renesas 16-Bit Single-Chip Microcomputer H8S Family/H8S/2200 Series H8S/2218 Group, H8S/2212 Group*, 12 2008. Rev.7.00.

[25] G. D. Robinson. Why 1149.1 (jtag) really works. In *Electro/94 International. Conference Proceedings. Combined Volumes.*, pages 749–754, May 1994.

[26] K. Rosenfeld and R. Karri. Attacks and defenses for jtag. *IEEE Design Test of Computers*, 2010.

[27] Stanislaw Skowronek. Nsa@home jrev, 2007.

[28] STMicroelectronics NV. *Medium-density performance line ARM -based 32-bit MCU, STM32F103x8 STM32F103xB*, 08 2015. Rev.17.

[29] STMicroelectronics NV. *ARM® Cortex®-M4 32b MCU+FPU, 225DMIPS, STM32F446xC/E*, 09 2016. Rev.6.

[30] STMicroelectronics NV. *Low & medium-density value line, advanced ARM-based 32-bit MCU, STM32F100x8*, 11 2016. Rev.9.

[31] Texas Instruments. *MSP430G2x33, MSP430G2x03 Mixed-Signal Microcontrollers*, 04 2011.

[32] Daichuan Wang and Pizhuang Zhang. The technology research of remote automatic detection and fault diagnosis based on jtag boundary scan. *Procedia Engineering*, 7:270–274, 2010.