# Improving Apple Test Chips Efficacy through the Integration of SSN and ICL Automation Flows

Master's Degree Thesis

**Candidate: Fulvio Castello**

Supervisor: Dirk Claussen

Advisors: Prof. Stefano Quer
Prof. Paolo Bernardi

Department of Computer, Cinema and Mechatronic Engineering
Polytechnic University of Turin

This report was created in collaboration with *Apple Inc.*

October 2023

*To my beloved grandfather Lucio Borriello, Electrotechnical Engineer and transistorization pioneer at Olivetti S.p.A., who never got to see me finally become a bit like him...*

# Disclaimer

I hereby declare that, except where specific reference is made to the labor of third parties, the contents of this Master's Thesis are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This work is the product of my own endeavors, and contains nothing which is the outcome of work done in collaboration with others, except as specified inside the text and the below acknowledgments.

Due to the signing of a non-disclosure agreement as part of the employment contract, two separate versions of this document had to be written up in order to include a different level of technical detail and sensitive data. As such, all project code names, confidential labels and any other related piece of information have been replaced with generic descriptors in this redacted revision.

# Acknowledgments

Firstly, I would like to recognize my advisors, Professors Stefano Quer and Paolo Bernardi, for introducing me to this incredible opportunity: I could have never dreamed of pursuing such a prestigious career start, and the link they created between the Polytechnic University of Turin and *Apple Inc.* came to be of inestimable value.

I am also extremely thankful to the whole company and, more specifically, to the Munich-based team for helping me grow both professionally and as an individual. Alessandro Ciraci acted as the daily reference point for all my technical and subjective doubts, showing an unprecedented level of patience and enthusiasm, and always motivating me to perform at my best. Dirk Claussen fittingly covered the role of both manager and fatherly figure during our collaboration, and proved to be a competent as well as a perceptive supervisor. Zhiguo Chen and Stephane Lecomte were the ones who originally envisioned my project, and pulled the strings for it to be completed in time. Pietro Macori was my friendly companion since the beginning, providing much appreciated advice and making me feel at home just a bit more.

Of course, I am deeply indebted to my family, and particularly to my parents, who continuously supported me in all ways imaginable and made me the person I am today. Nothing I have achieved so far would have been possible without them, and their countless encouragements really pushed me to keep bettering myself, and always will. A special mention goes to my dear grandmother Livia, who is not longer with us, and who sadly missed this chapter of my life by just a few days. I am sure she would have been so eager to hear all the stories I would have wanted to share with her.

Finally, I could not have undertaken this journey without my partner Sara. She consistently stayed by my side, even when we were not physically together, and cheered for me during both the highs and the lows of this experience. She always paid attention to everything I wanted to get off my chest, lovingly providing helpful suggestions and keeping me company at all times. I will forever be grateful to her for the loyalty she demonstrated during this dreamlike adventure.

# Abstract

The increasing importance of the Design For Testing engineering approach within the silicon industry is exemplified by the need of validating the manufacturing process of electronic devices that have become more complex than ever before. The optimization of test time and efficiency for an ever-growing amount of dies has become the major focus of all tech corporations that strive to deliver higher quality products more frequently at a lower cost.

The *Apple Inc.* team based in Munich is developing pre-production prototypes called "test chips" in order to organize all the output of their manufacturing processes into a flexible hardware arrangement meant for testability purposes. Their building blocks are grouped into clusters, which are the basic functional units for all trials performed on silicon. The essential capability in this context is represented by the capability of dynamically managing different cluster combinations, activating only the selected ones while bypassing the others.

The former state of the art of the company consisted of a rigid selection mechanism that did not provide this commodity in a serviceable way. The conception of a modern, automated flow able to render all upcoming architectures compatible with the Streaming Scan Network technology represents an answer to these challenges. In fact, it can be used to streamline the delivery of test data to each node connected inside a common network and seamlessly works in conjunction with the latest market standards.

Its implementation requires the development of an auxiliary characterization incorporating a condensed description of the hardware, so that the automated tool of choice offered by *Siemens* can use it to communicate with the device under test in a more optimized way. After applying several case-specific configurations, the final goal is to generate pattern sets, which are the primary input stimuli ultimately needed to perform the ensuing test runs.

The solution presented so far was previously non-existent at the company, and as such both its implementation and its validation constituted the main task of the presented

project. Firstly, a central script is responsible for automatically deriving a simplified model of the design required by the utilized software tool. The interface with the user is provided by a set of command line arguments, whereas the principal input resides in a unique data structure that contains all the information used to also define the reference architecture through the most common Hardware Description Languages.

The two parallel archetypes represented by the original, complete depiction and the newly obtained schematized arrangement must then be matched against one another as to check for their 1:1 correspondence. Consequently, a whole new automation flow has been developed around trying to prove this identity and writing out the requested testbench files that can be run in order to get a definitive yes/no answer in terms of quality acceptance.

The positive impact of the fabricated deliverables can be demonstrated in terms of timing performance, flawless execution and adaptability and opportunity for reuse. In fact, this automated solution is orders of magnitude faster and less error-prone than a handcrafted approach, while its key advantage resides in its adaptability to all future test chip architectures with little to no need for manual intervention. Finally, its outcomes have already been certified within two of the company's current projects and are being extensively used in production.

# Table of contents


**1 Introduction** 1

**2 Background concepts** 4

2.1 Key DFT techniques . . . 4

2.1.1 IEEE 1149.1 (JTAG) . . . 4

2.1.2 IEEE 1500 (ECT) . . . 6

2.1.3 IEEE 1687 (IJTAG) . . . 8

2.1.4 Embedded Deterministic Test (EDT) . . . 13

2.1.5 Streaming Scan Network (SSN) . . . 14

2.2 Test chip architecture . . . 17

**3 State of the art** 21

3.1 Previous bypass solutions . . . 21

3.2 Manual ICL insertion . . . 22

**4 Technical documentation** 25

4.1 Main objective . . . 25

4.2 Conversion script . . . 26

4.3 Model verification . . . 33

**5 Results** 37

5.1 Timeliness . . . 38

5.2 Trustworthiness . . . 38

5.3 Scalability . . . 39

**6 Conclusion** 41

**Nomenclature** i

References v

# List of figures


2.1 JTAG architecture overview . . . . . 5
2.2 The TAP controller state machine . . . . . 7
2.3 ECT architecture overview . . . . . 9
2.4 EDT architecture overview . . . . . 14
2.5 SSN architecture overview . . . . . 15
2.6 SSN packetized transactions for two unequal blocks . . . . . 16
2.7 SSN continuity pattern validating the primary data bus . . . . . 17
2.8 SSN loopback pattern verifying an active SSH node . . . . . 18
2.9 Serial chain pattern diagnosing external scan chains . . . . . 18
2.10 High-level view of a test chip . . . . . 19

3.1 Simplified view of two former bypass architectures . . . . . 22

4.1 Present state of the SSN flows for *Apple* test chips . . . . . 26
4.2 Technical blueprint of the internal architecture . . . . . 31
4.3 Partition-level view of two successive bypass architectures . . . . . 32
4.4 State diagram summarizing the verification flow . . . . . 35

# Chapter 1

# Introduction

One of the main challenges that all companies involved in the silicon industry have to face is the increase of transistor count inside dies more intricate than ever before, an unavoidable side effect of the continued strive for better hardware performance and efficiency. More specifically, the modern methodologies used to mass-produce these chips require a lot of effort to validate an immense quantity of circuits, effectively reinforcing the significance of the Design For Testing approach. The ensuing processes nowadays play a major role in dilating the NRE[1] cost to an unprecedented magnitude, so that optimization in this field has evidently acquired a paramount importance.

The DFT[2] team at *Apple* Munich responsible for shaping all their upcoming iterations of test chips (i.e. surrogate structures exploited to verify different combinations of IPs[3]) has decided to deal with the above issues by investigating an innovative method aimed at optimizing their internal flows by reducing total test time and complexity. Their solution revolves around integrating the Streaming Scan Network mechanism into their devices, crucially boosting the efficacy of the utilized scan architectures and trials by a considerable margin. What's more, said arrangement is also capable of dealing with a practical restraint boasted by the target platform itself. In fact, since each scan partition must have a dedicated number of pads inside the target platform, the amount of concurrently active units would be narrowed down to the limited quantity of available I/O[4] if it wasn't for the aforementioned technology.

[1]Non-Recurring Engineering
[2]Design For Testing
[3]Intellectual Properties
[4]Input/Output

The introduction of such a revolutionary feature required the development of a brand-new model (consisting of an `ICL`[5] characterization) equivalent to the RTL[6] description of the IJTAG[7] functionality included within the DUT[8] itself, plus an entire sequence of scripts and commands aimed at supporting its configuration handled by a newly adopted EDA[9] tool powered by *Siemens*. These practices undoubtedly prefaced the development of a correspondent set of automation elements, completely revamping the previous methodologies and eliminating many inconsistencies within the former state of the art. In fact, the latter essentially relied upon a handcrafted baseline of error-prone code revisions and rigid logic layouts that exhibited limited flexibility.

This Master's Thesis is devoted to detailing the implementation of the aforementioned improvements, giving an insight into the motives that reside behind their conception, the electronic standards from which they originated, the technical aspects that characterize their functionality and their effect on the correlated projects.

## Document organization

This report consists of six main chapters, the first one being the current summary of both the central topics covered by the Master's Thesis and its primary goals. Subsequently, chapter 2 will include an extensive overview of the subjects that constituted the foundations of the presented project, whereas a brief depiction of the state of the art of DFT engineering at *Apple* Munich will follow in chapter 3. The detailed description of the output of the entire internship period is contained in the fourth chapter, while its benchmarks and effects on the team's performance will be listed in chapter 5. Finally, a closing section is devoted to providing a high-level recap of the previous concepts, before giving a few personal remarks about the experience as a whole and mentioning some possible starting points for future work.

[5] Instrument Connectivity Language
[6] Register-Transfer Level
[7] Internal JTAG
[8] Device Under Test
[9] Electronic Design Automation

# Chapter 2

# Background concepts

In the following, an overview of several notions behind DFT engineering is presented, focusing on the most used and innovative standards within the industry at the time of writing. These elements serve as a crucial starting point for the project that is going to be detailed in later sections, and as such should be grasped in full before inspecting the underlying exposition.

## 2.1 Key DFT techniques

Design For Testing is a rampant VLSI[1] approach which entails many IC[2] design methodologies aimed at adding testability features to a digital circuit. This process facilitates the conception and exploitation of manufacturing trials used to check for hardware defects during silicon bring-up. As a direct consequence, debugging complexity and overall test time and cost are greatly reduced, resulting in a higher quality product and a more competitive time-to-market.

### 2.1.1 IEEE 1149.1 (JTAG)

Boundary-scan is a methodology that applies to the interface of an IC package in order to transform extremely challenging printed circuit testing problems into well-structured topics that can be easily dealt with. As shown in Figure 2.1, a set of boundary-scan cells (displayed in *green*) are added between the I/O pads of the circuit and the primary ports of the target logic. These are connected in such a way that the resulting shift

[1]Very Large Scale Integration
[2]Integrated Circuit

register allows for controlling input pins during a Capture operation and then observing the corresponding outputs during an Update event [2].

Fig. 2.1 JTAG architecture overview

The standard also defines four mandatory test pins, namely Test Data In (TDI) and Test Data Out (TDO) to provide a path for the payload, and Test Clock (TCK) plus Test

Mode Select (TMS) for control capabilities, while an optional Test Reset (TRST) signal can also be specified. These all make up the so-called Test Access Port (TAP), which is directly fed into a controller to manage the rest of the system by means of the 16-state FSM[3] illustrated in Figure 2.2. Here, synchronization is provided by TCK pulses, while state transitions are regulated by each sampled TMS value. According to a crucial safety mechanism, if the latter signal is held high for five clock edges then the controller will be immediately sent back to the Test-Logic-Reset state, discarding any current operation that might still be in progress. The two ensuing paths for data and instructions are used during load/unload procedures of the related storage elements.

In fact, each compliant device usually includes many scannable registers connected in parallel which are multiplexed (as shown in *yellow* in Figure 2.1) in such a way that several instructions can effortlessly yield different results. One significant example is represented by the Bypass register, which allows any IC that is not taking part in a specific use case to be seen as one single bit from the outside, effectively introducing a minimum amount of delay inside the overall daisy chain. On the other hand, the optional Device Identification register holds a unique 32-bit identifier which can be compared with an expected value in order to check for possible assembly mismatches [1].

### 2.1.2 IEEE 1500 (ECT)

This standard wrapper architecture represents a scalable solution to the increased need for design reuse within the silicon industry, especially when it comes to complex multi-core SOC[4] integration. Its mandatory hardware elements include (as shown in Figure 2.3):

- Wrapper Serial Input/Output (WSI/WSO), a 1-bit port pair necessary for shifting both data and control sequences into/from the wrapper (a parallel counterpart represented by WPI[5] and WPO[6] can optionally be included for high bandwidth testing as well)
- Wrapper Interface Port (WIP), a collection of seven signals regulating the operation of the whole structure by means of, among others, a selector which chooses whether WSI and WSO are used for instructions or test data, plus a few flags triggering a series of homonymic events

[3] Finite State Machine
[4] System On a Chip
[5] Wrapper Parallel Input
[6] Wrapper Parallel Output

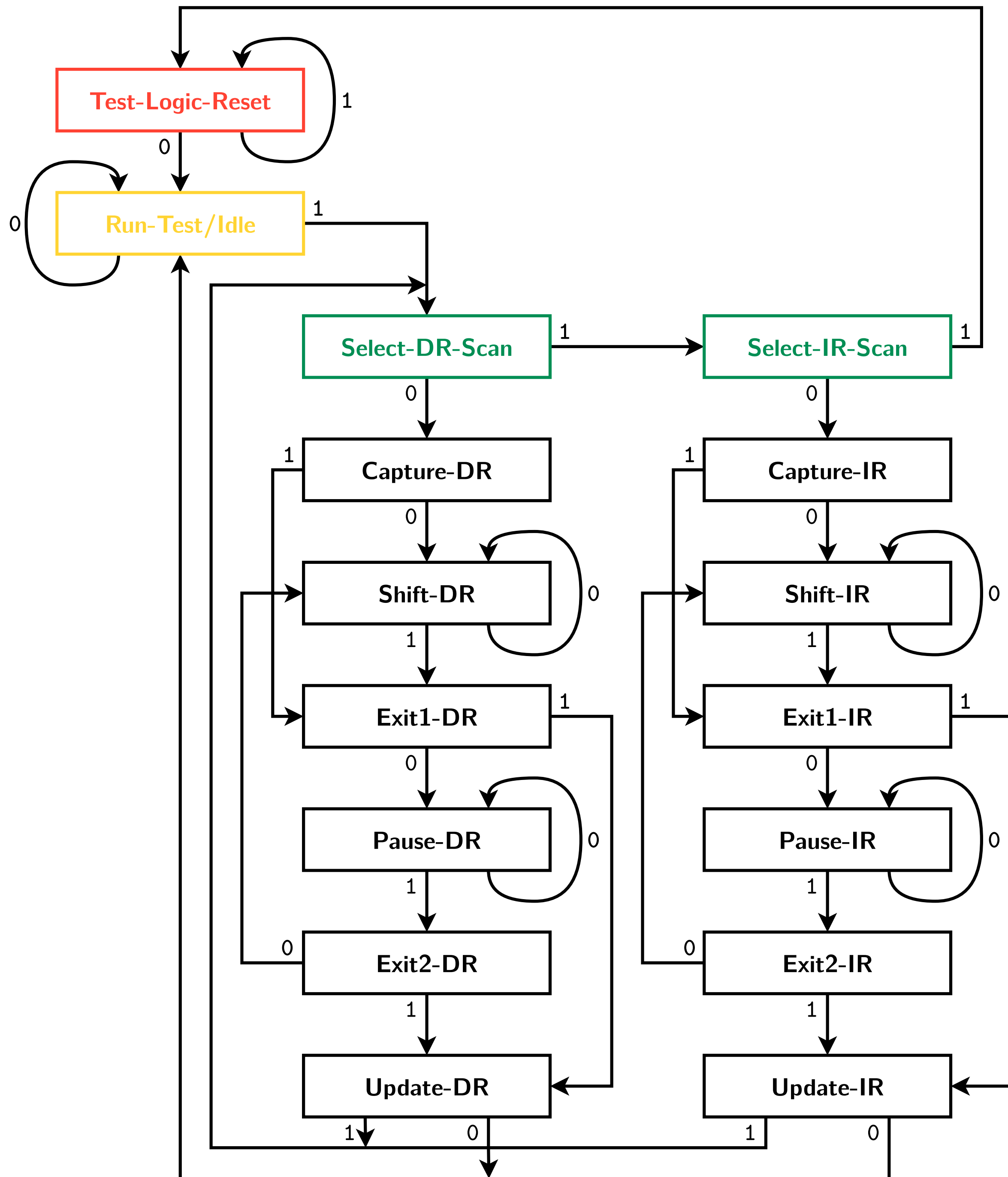


Fig. 2.2 The TAP controller state machine

- Wrapper Bypass Register (WBY), the default data register that establishes a direct path between WSI and WSO when no other selection is made
- Wrapper Instruction Register (WIR), the main configuration register used to force the wrapper into the required mode of operation for any given instruction
- Wrapper Boundary Registers (WBR), two data registers constituted by input/output wrapper cells which provide test access to the core terminals.

The activity of the WBRs is then defined by the following set of events, which are all controlled by the WIP:

1. Shift — the data stored inside the shift path of the first WBR is advanced by one position toward its output, while the data present at the input of the second WBR is firstly loaded into its input shift path
2. Capture — the data present at the input/output of any wrapper cell is stored inside a sequential element of the WBR
3. Update (optional) — the data stored inside a shift path is loaded into a corresponding non-shift-path element, if present
4. Transfer (optional) — data is moved to the input shift path or toward the output of a wrapper cell by one position, depending on where it was previously stored during Capture and/or on the length of the shift path itself.

All the aforementioned serial terminals are active high (except for the reset signal), whereas new WSI data will be captured on a rising edge of WRCK[7] and WSO data will change on a falling edge instead. Additionally, a design can still partially respect the standard by means of the so-called "unwrapped" compliance, which dictates that a wrapped core can be automatically generated out of it without any additional modification to the original hierarchy [5, 13].

### 2.1.3 IEEE 1687 (IJTAG)

The main focus of this standard is to provide a test access methodology within an IEEE[8] 1149.1-compliant device in response to the increased variety of ways that exist to debug different embedded IPs. The main architectural components described by IJTAG are [3]:

[7] Wrapper Clock
[8] Institute of Electrical and Electronics Engineers

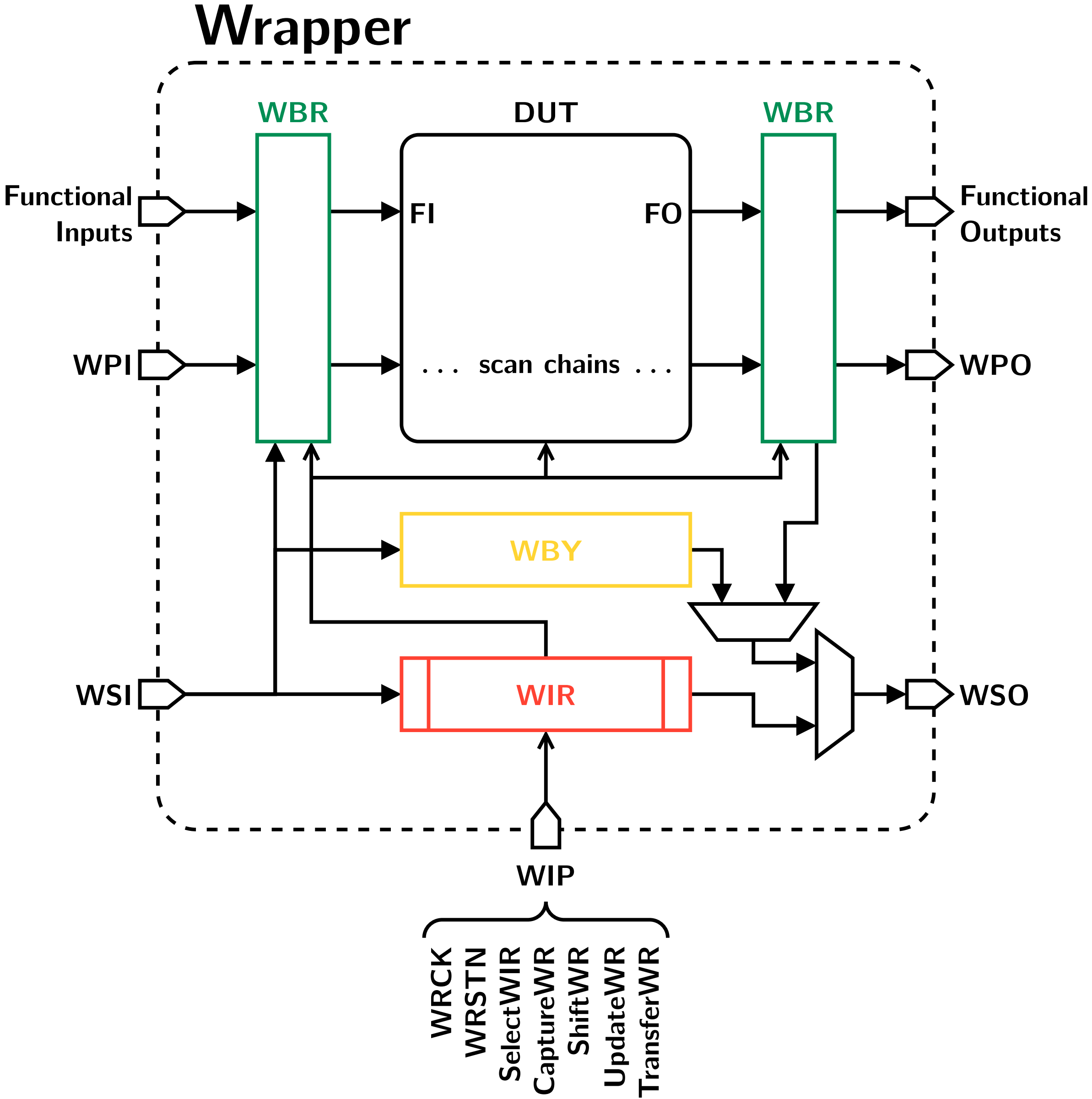


Fig. 2.3 ECT architecture overview

- Eight interface signals (namely TCKPort, ShiftEnPort, CaptureEnPort, UpdateEnPort, ResetPort, SelectPort, ScanInPort and ScanOutPort), which are functionally similar to those included in IEEE 1500, but with no distinction between instruction and data registers
- Scan bits, a shift chain able to capture relevant data in a transparent fashion
- Update bits, supplying stable information to the core while the chain is shifting
- Segment Insertion Bit (SIB), a toggle for opening or closing access to the newly introduced re-configurable scan chains at any level
- Scanned IP Macro, a control chain used to define a memory wrapper for added flexibility during testing
- One or more TAP controllers that can handle the daisy chain of active boundary scan devices
- A ring of BIST[9] receivers connected to a central server through a common bus.

Consequently, it becomes possible to seamlessly access any Test Data Register (TDR) via the usual TAP by means of a simplified model of the design. The latter is constituted by a procedural software protocol as well as a network interface description, which complement each other in order to make packaging for reuse possible without the need for further modifications [9, 14].

### Procedural Description Language (PDL)

Presented as a buffered extension of `TCL`[10], it is used as an amenity to provide communication and functionality to embedded instruments. All the operations needed to run the preconceived tests are described by a set of commands to read/write from ports, scan chains and registers within the network. These transactions are described with respect to the boundaries of the design, and get inserted into a non-deterministic queue of events which has to be flushed in order for them to be applied concurrently. In the following, a snippet of some code derived from one of the involved projects is depicted:

```
# Generating custom iProcs for "main_part"
if {$main_part eq "soc_enc"} {
```

[9] Built-In Self-Test
[10] Tool Command Language

```
  set scan_en   "GPIO0_4"
  set scan_clk  "GPIO0_2"
} elseif {$main_part eq "cpu_rlm"} {
  set scan_en   "cpu_rlm_scan_enable"
  set scan_clk  "cpu_rlm_scan_shift_clock"
}

iProcsForModule [get_single_name [get_current_design]]

iProc check_ssn_access_neg {} {
  # Generated DRC for checking accessibility of all active SSN Hosts
  foreach_in_collection sib_config_bit \
  [get_icl_objects -object_types scan_reg_icl \
  -filter name=~*shift_register_sib*] {
    iWrite [get_single_name $sib_config_bit] 0b1
    iRead  [get_single_name $sib_config_bit] 0b0
    iApply

    iWrite [get_single_name $sib_config_bit] 0b0
    iApply
  }
}

# Fix for SSN issue on stable channel
iProc close_pipes_cpu {} {
  global      scan_en
  global      scan_clk

  iForcePort  ${scan_en}  0b0
  iRunLoop    5
  iPulseClock ${scan_clk}
  iApply

  iForcePort  ${scan_clk} 0b1
  iApply
}
```

### Instrument Connectivity Language (ICL)

This hardware-related facility describes the abstract, hierarchical definition of I/O ports, registers, and scan chains that are necessary to control the instrument at the interface

level. Every module is instantiated as a black box, since the internal implementation is not included in favor of the essential segments needed to apply the associated PDL procedures. Interconnections between different components are constructed by means of primitive building blocks such as multiplexers or storage elements, while some rudimentary logic between coupling signals can also be characterized within the network. The example below illustrates part of a cluster definition from one of the known architectures:

```
Module cpu_cluster1_wbr {
  ResetPort cpu_res_str {
    ActivePolarity 1;
  }
  TCKPort cpu_wrap_clock;
  CaptureEnPort cluster1_wrap_capt_enable;
  ShiftEnPort   cluster1_wrap_shift_enable;
  UpdateEnPort  cluster1_wrap_update_enable;
  ScanInPort    cluster1_wrap_scan_in;
  ScanOutPort   cluster1_wrap_scan_out {
    Source CPU_CLU1_COMMON_REG_WBR.ijtag_so;
    Attribute LaunchEdge = "Rising";
  }
  ScanInterface cpu_cluster1_res {
    Port cluster1_wrap_scan_in;
    Port cluster1_wrap_scan_out;
  }
  ToResetPort ssh_wir_res_str {
    ActivePolarity  1;
    Source          cpu_res_str;
  }

  Instance CPU_CLU1_COMMON_REG_WBR Of CPU_CLU1_COMMON_REG_WBR {
    InputPort ijtag_reset              = cpu_res_str;
    InputPort ijtag_clock              = cpu_wrap_clock;
    InputPort ijtag_sel                = 1'b0;
    InputPort ijtag_ce                 = cluster1_wrap_capt_enable;
    InputPort ijtag_se                 = cluster1_wrap_shift_enable;
    InputPort ijtag_ue                 = cluster1_wrap_update_enable;
    InputPort ijtag_si                 = cluster1_wrap_scan_in;
    Attribute tessent_design_instance  = "common_reg";
  }
}
```

### 2.1.4 Embedded Deterministic Test (EDT)

Introducing test compression into a DFT flow has become more essential than ever before, since larger and larger circuits require increasing test data volume, time and memory requirements, which ultimately stack up to an ever-growing cost:

$$\text{Test Cost} \propto \begin{cases} \text{Test Data Volume} \approx \text{Scan Cells} \times \text{Scan Patterns} \\ \text{Test Time} \approx \dfrac{\text{Test Data Volume}}{\text{Scan Chains} \times \text{Tester Frequency}} \end{cases}$$

The main idea to fix the above issues resides in reducing the number of effective scan chains that the external tester has to interact with, in such a way that a smaller amount of total comparisons has to be made in the end, reducing the budget required to run these processes. Such simplification can be achieved by means of the composition shown in Figure 2.4: the design core is left untouched, but two additional blocks now manage the actual interface with the ATE[11].

Firstly, the decompressor is constituted by a ring generator and a phase shifter. The former is implemented as a LFSR[12] with external inputs called “injectors”, or more commonly referred to as “EDT channels”. These additional pins are XORed with the ordinary taps in order to provide the necessary compressed stimuli, while cutting down on the propagation delay and internal fanout introduced by the feedback network. On the other hand, the second component minimizes the probability of a diagonal relationship existing between adjacent LFSR bit streams. These are shifted by a set number of cycles thanks to an XOR tree, obtaining a much broader level of randomness and allowing the ring generator to drive a considerably larger amount of scan chains [11].

The spatial compactor is again implemented as an XOR tree in order to reduce the number of data outputs. Some additional masking logic is embedded into the structure, so that a specific code gets shifted into the matching register thanks to the dedicated edt_clock, edt_update and edt_mask signals. An ensuing decoder is then capable of making only one scan chain observable at a time while leaving all the others in a shadowed state: this way, identifying the defective one and modeling its type of fault become very straightforward. This mechanism is intrinsically responsible for tackling two critical issues [7, 10]:

[11] Automated Test Equipment
[12] Linear Feedback Shift Register

- an unknown value propagation event that would surely end up completely disrupting the resulting coverage and which may originate from black boxes, non-scan cells and false paths
- fault aliasing due to an even number of flawed scan cells happening to line up at the same location in different scan chains that are compacted to the same output channel, effectively concealing each other's effect.

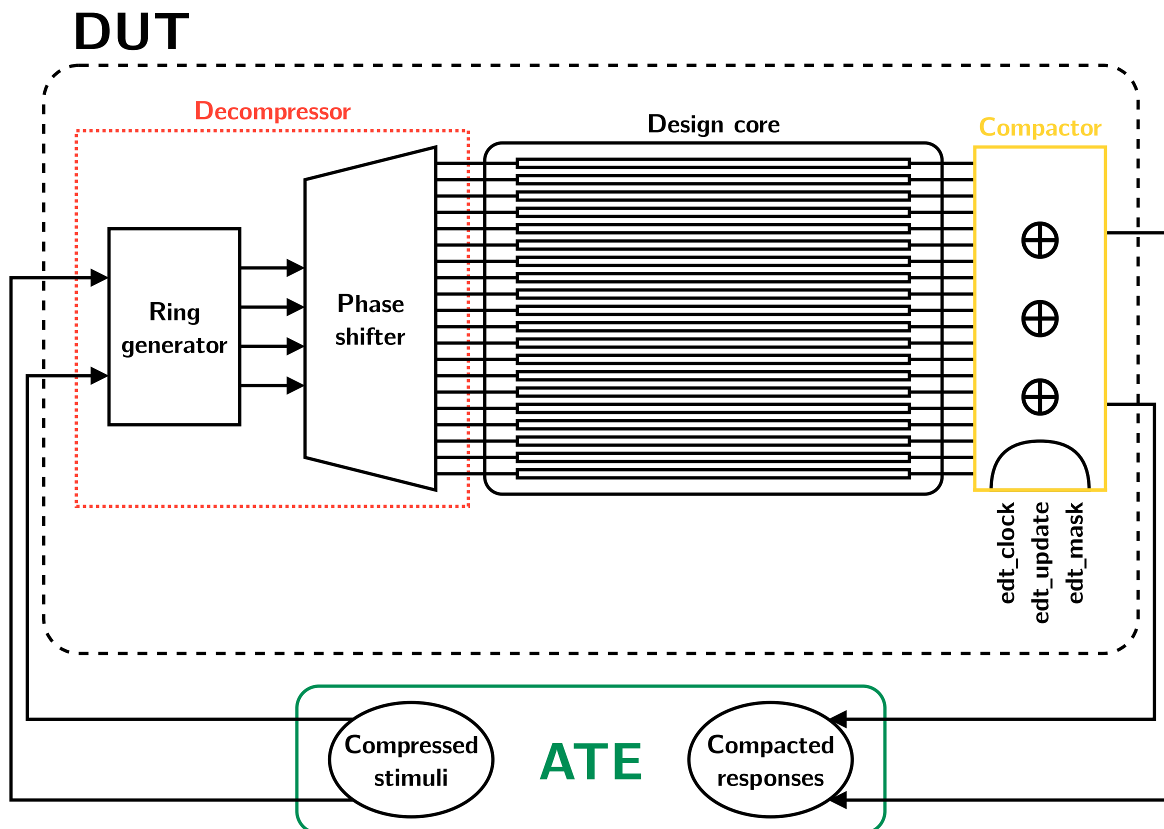


Fig. 2.4 EDT architecture overview

### 2.1.5 Streaming Scan Network (SSN)

This cutting-edge technology was firstly introduced in 2020 in an effort to further increase the number of internal scan channels that can be driven with respect to a reduced number of chip-level scan pins, effectively complementing and improving the advantages already offered by EDT compression. As shown in Figure 2.5, its hardware implementation consists in building a continuous datapath made of a main $N$-bit bus (displayed in *red*)

which connects all active core instances, plus a parallel 1-bit IJTAG interface (drawn in *yellow*) mainly used for configuration. Inside each node, the Streaming Scan Host (SSH) is responsible for communicating with the corresponding EDT controller (by means of the *green* wires) to deliver test data, although uncompressed scan chains and hybrid architectures are supported as well.

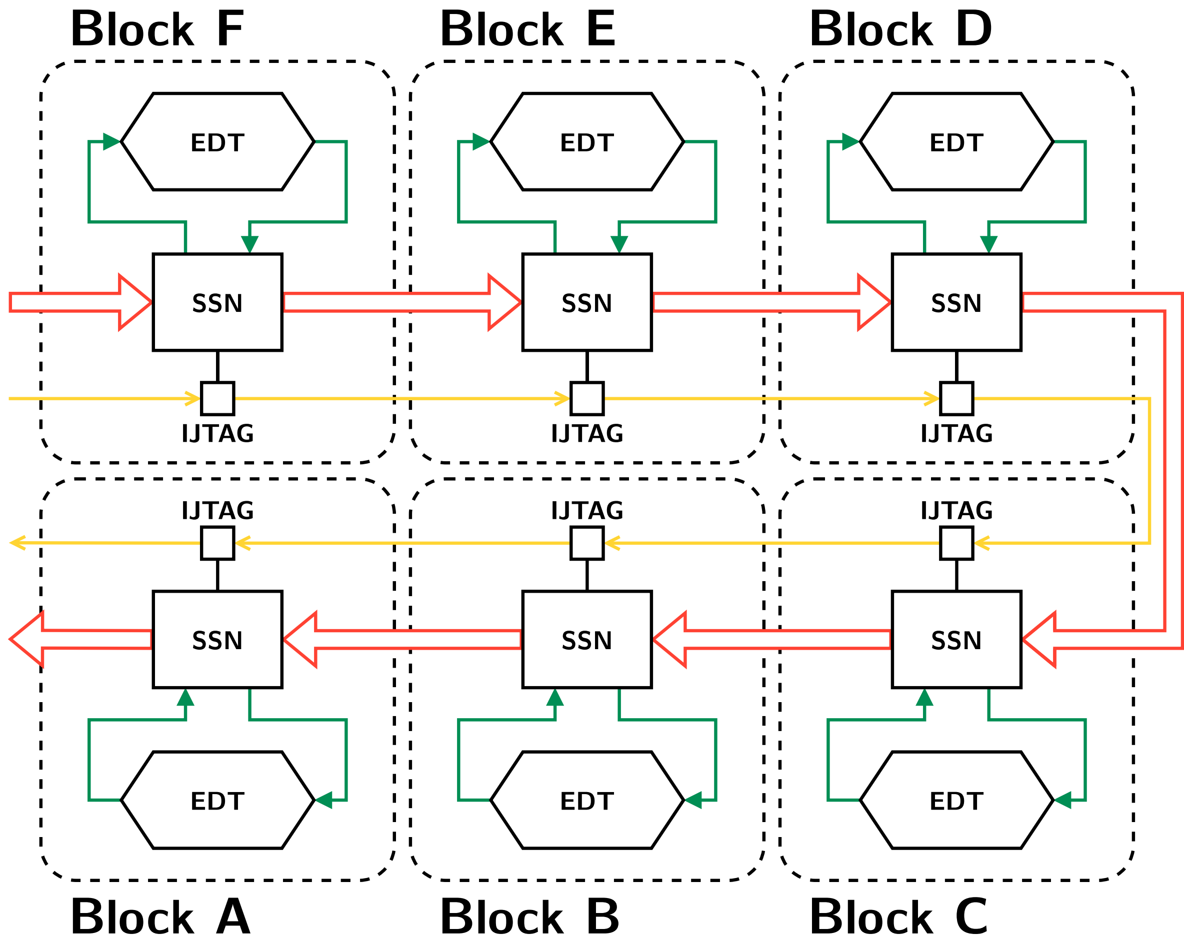


Fig. 2.5 SSN architecture overview

The essential advantage offered by said arrangement resides in its packetized nature: a “packet” is defined as the scan data needed for all the active SSH nodes to perform a single scan shift operation. Figure 2.6 illustrates an example in which the chip includes an 8-bit SSN bus, while two subsequent blocks display a different number of EDT pins (4 and 5, respectively). Since 9 total channels have to be driven concurrently, the stream of data gets folded across multiple cycles so that only 8 bits at a time will be transmitted. The initial setup provided by the IJTAG interface will be in charge of pre-assigning

each of them to the correct location at the right time, so that no opcode or address information has to ever be sent during testing.

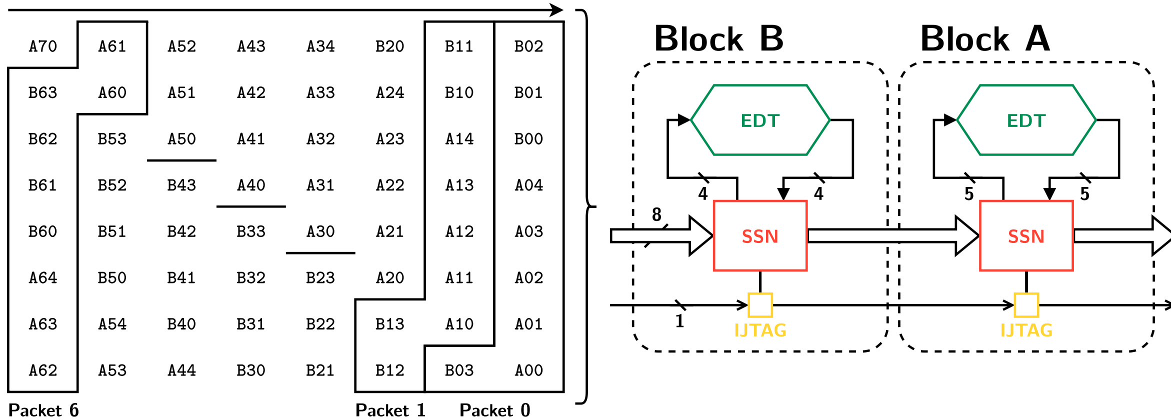


Fig. 2.6 SSN packetized transactions for two unequal blocks

The separation between distinct packets will just shift and follow the available bandwidth, obtaining an easily scalable, plug-and-play way of communicating with the different cores without the need of statically characterizing the chip at design time, while also fully optimizing the overall test duration. The 1-bit IJTAG network can even be used for this purpose in case the main one presents hardware defects during silicon bring-up or is non-existent altogether in very streamlined platforms. As a side note, it also becomes possible to use the above features to increase the external tester frequency without exceeding the internal constraints, since all active nodes will only access a fraction of the payload during each period [4, 6].

There exist several pattern sets to verify the integrity of the SSN datapath inside a compliant design, each one of them respectively increasing the scope (and thus the strictness) of the testing mechanism they provide, as detailed below [12].

### Continuity

The simplest form of sanity check consists in validating the main SSN bus dedicated to the scan payload, as displayed in Figure 2.7. The internal logic of every SSH gets completely bypassed, effectively treating each node as if it was disabled, with the aim of confirming whether the whole network can be reached and is unbroken.

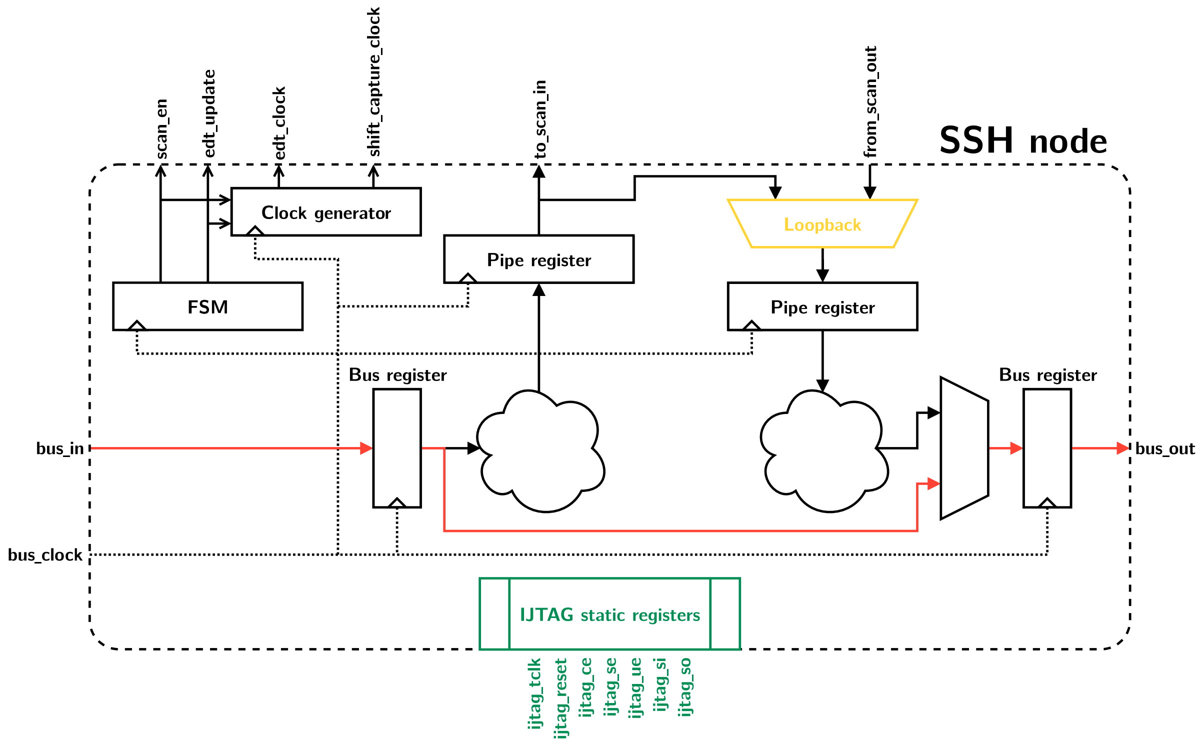


Fig. 2.7 SSN continuity pattern validating the primary data bus

### Loopback

A special purpose multiplexer is included inside the structure of each SSH, so that it becomes possible to check whether it is programmable when activated without having to communicate with the EDT controller attached to it. The internal path used for this kind of evaluation is shown in Figure 2.8.

### Chain

The classic binary transition test (originating from sequences of "0011..." pulses) used to check for flawed scan chains again crosses the entirety of the SSH hierarchy, but this time the payload also has to go through some external pipelining in order to reach the EDT controller of each SSN node, as depicted in Figure 2.9.

## 2.2 Test chip architecture

A test chip is a proof-of-concept design that is manufactured for purposes other than reaching the mass market. Its main application is to provide a platform to validate the

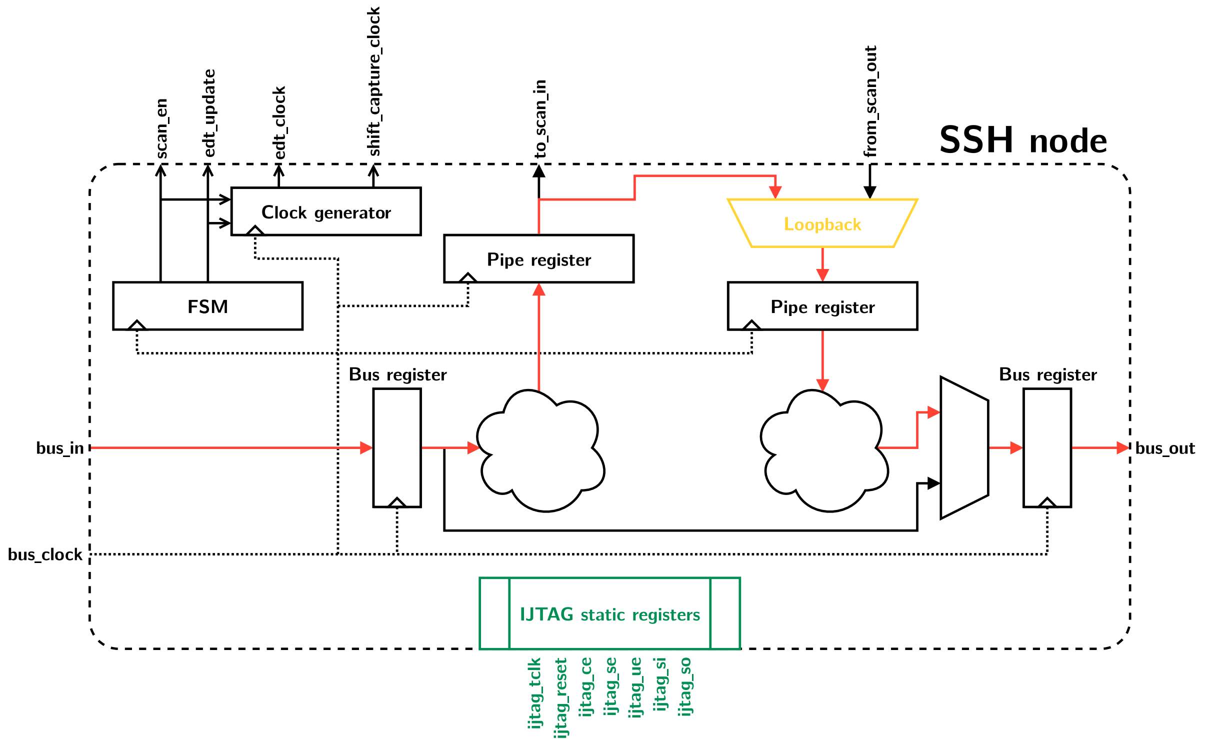


Fig. 2.8 SSN loopback pattern verifying an active SSH node

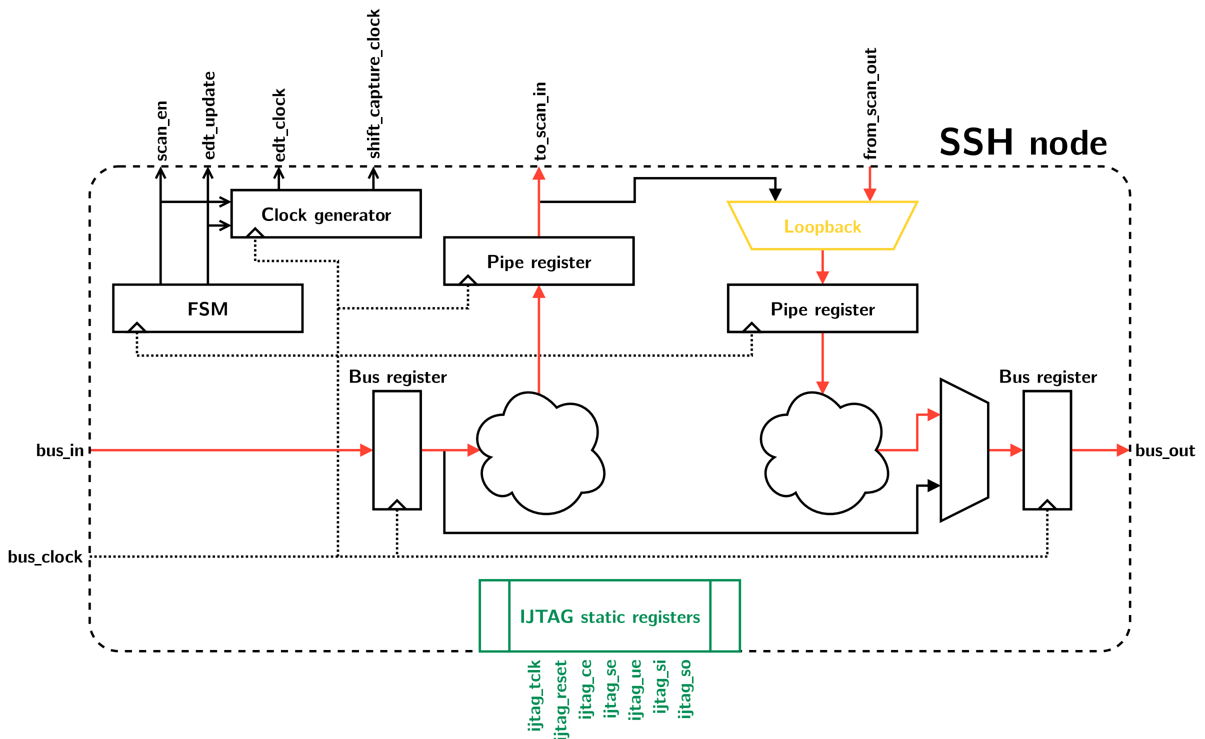


Fig. 2.9 Serial chain pattern diagnosing external scan chains

IPs that will be included in its volume production counterpart. This way, it becomes possible to characterize the essential working parameters of each component, including voltages, frequencies and power delivery metrics.

The aforementioned IPs constitute the building blocks of a hierarchical structure, since several instances of them are gathered into a so-called "cluster". A set of many different clusters can be of varying dimensions and will define one "partition" of a "chiplet", which in turn will contain numerous equivalent occurrences of said organization. A partition usually entails diversified clock and voltage domains in order to provide seamless access to all the individual features provided by the conceived test platform, and to better isolate the location, magnitude and cause of a particular type of defect [8].

The top-level design will finally be constituted by a multitude of chiplets, being them destined to the main SOC (usually taking up the majority of the total die area) or to AMS[13] interconnects (incorporating PLLs[14], metrology sensors, and current/voltage, period jitter monitors). Lastly, Figure 2.10 shows the hierarchy that constitutes a test chip, with many sample IPs included in its internal organization.

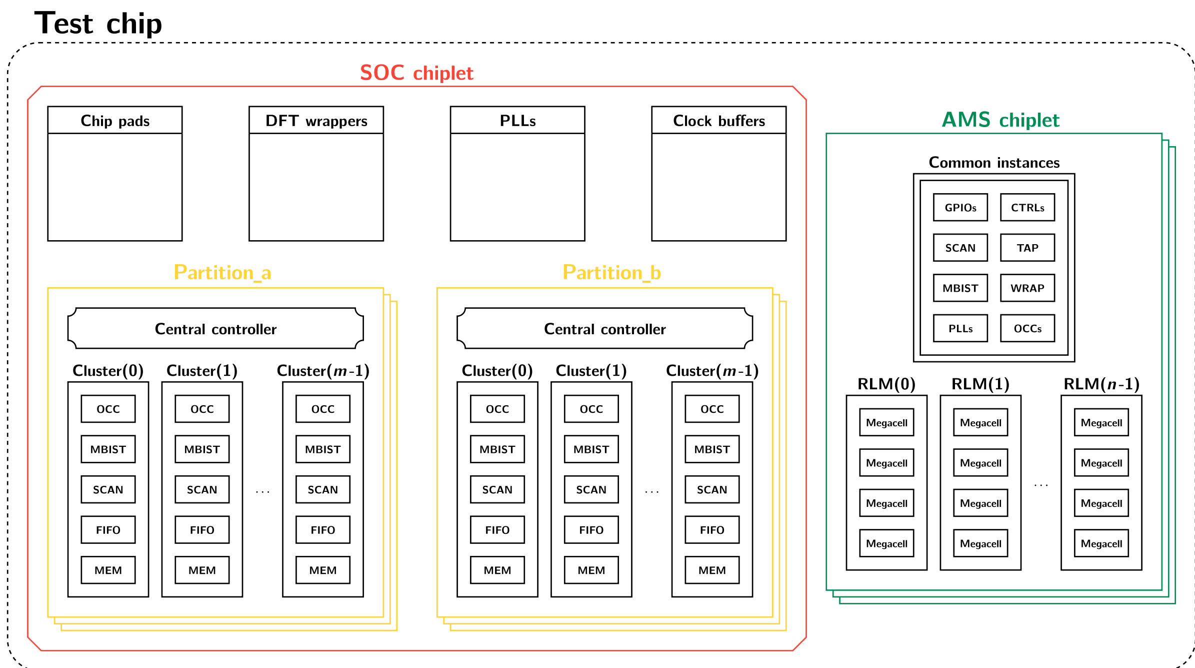


Fig. 2.10 High-level view of a test chip

[13]Analog and Mixed Signal
[14]Phase Lock Loops

# Chapter 3

# State of the art

This chapter encompasses a quick overview of the past approach to test chip verification exploited by the *Apple* DFT team in Munich, before the introduction of the SSN technology together with the necessary `ICL` automation that comes along with it. The main limitations that will figure shortly are what originally drove the interest of the company toward investing into such an innovative solution aimed at increasing overall productivity, thus mandating the creation of ad hoc flows to support it.

## 3.1 Previous bypass solutions

The principal functionality required by DFT engineers out of a test chip is the capability to assess a wide variety of clusters combinations, being able to stimulate either one at a time, several at once or even the whole enclosing partition. This kind of facility was already offered by the top schematic visible in Figure 3.1, with the selection mechanism being essentially represented by the *yellow* multiplexers. Given that such picture only represents a streamlined version of the actual design, its straightforwardness is made evident by the presence of a clear-cut daisy chain of *red* blocks of logic (which would be intertwined by a more complex pipelining network) between SI[1] and SO[2].

The huge concern that arises in this case is the linear accumulation of scan elements. In fact, the lengths of the internal scan chains intrinsic to each operating cluster (exemplified in *green*) end up accumulating as in $1000 + 1000 + 1000 + 400 = 3400$, if all of them are active. Being much smaller than its predecessors, the last element would also waste a considerable amount of bandwidth. What's more, every time the configuration needs

[1]Scan In
[2]Scan Out

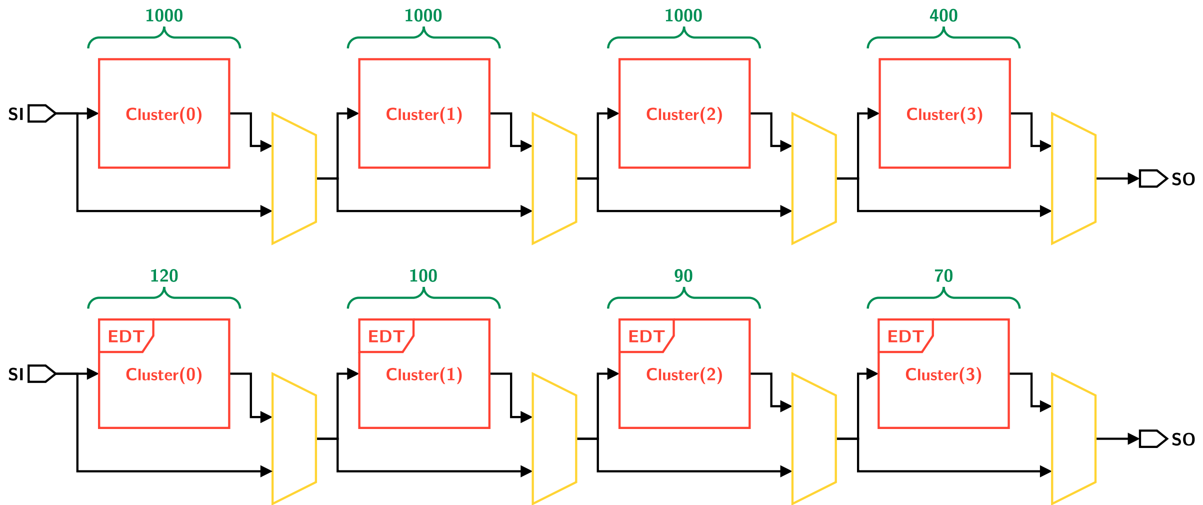


Fig. 3.1 Simplified view of two former bypass architectures

to be changed, the whole initialization sequence of the platform in question has to be re-run from the beginning before injecting the actual payload. It thus becomes evident how total test time is negatively impacted by this kind of non-optimized solution.

A later improvement that would reduce the huge amount of scan elements is pictured in the bottom half of the above schematic, mainly consisting in adding scan compression features to every unit, easily obtaining a speed-up of more than one magnitude. However, since it is never possible to concatenate multiple EDT instances, this kind of arrangement could only be used for One Hot testing (i.e. strictly one cluster selected at all times). Any other desired combination would have to fall back to the initial architecture, again revealing the same limitations.

## 3.2 Manual ICL insertion

Once the decision was made to transition to the Tessent™ Shell software (provided by *Siemens*) as to make future designs SSN-ready, it immediately became clear that said automated tool would require a simplified model of the IJTAG network in order to be able to access its components and provide optimized testing mechanisms. Since `ICL` was the language of choice to integrate this kind of technology, a partition-level description had to be manually created and checked for compatibility. Within the resulting set of files, many blemishes would derive from a heavy use of the "copy and paste" mechanism, and the abundance of hard-coded values and properties would make the code totally platform-dependent and hardly scalable.

Additionally, a totally new replica of the same structure would have to be created for each different combination of clusters selection, such that only those which are active in any given scenario would be included and correctly wired to the network, leaving the others in a bypassed state. The case-specific tweaking of the test instance would also result in a tedious and error-prone course of action that clearly needed to be summarized into a standardized, universal flow that could run autonomously starting from a few basic user inputs.

A couple of templates for all of the above were initially fabricated by hand, before quickly realizing the urgent necessity of an automated process that would produce the same outcome and even complement it with a few additional features that were later discovered to be necessary all along. These samples would later serve as the basis for the work that is hereby being presented.

# Chapter 4

# Technical documentation

The main task assigned during the internship period is hereby described, focusing on the technical aspects that characterized both the actual conception of a brand-new piece of code and its consequent integration inside the methodologies utilized by the *Apple* DFT team in Munich.

## 4.1 Main objective

The idea behind this project stemmed from the need to streamline both the design phase and the verification process of two generations of test chips by developing a new framework for integrating and testing the Streaming Scan Network technology into future architectures, thus giving birth to a whole new set of methodologies aimed at optimizing payload delivery and managing test time in a more efficient manner.

Firstly, a format conversion script had to be developed following many well-defined internal guidelines, in order to obtain an `ICL` description of the essential structure of the IJTAG network included in every test chip. After that, an extensive effort was put into proving the correspondence between the newly generated `ICL` model and both the original RTL and the netlist that would later be used for pattern generation. The double-edged advantage of this second step resides in the fact that a test "PASS" would not only intrinsically prove the sought-after identity, but it would also mean that the SSN datapath had been successfully implemented inside the DUT. In order to obtain these results, several runs and simulations had to be attentively diagnosed, eventually leading to the establishment of an automated verification flow that will continue to be exploited and improved for future projects as well.

A high-level overview of what has been described so far is shown in Figure 4.1, where the top path represents the reference point for the automation/validation sequence that complements the previously utilized state of the art, while the bottom part depicts the set of new additions developed during the internship period.

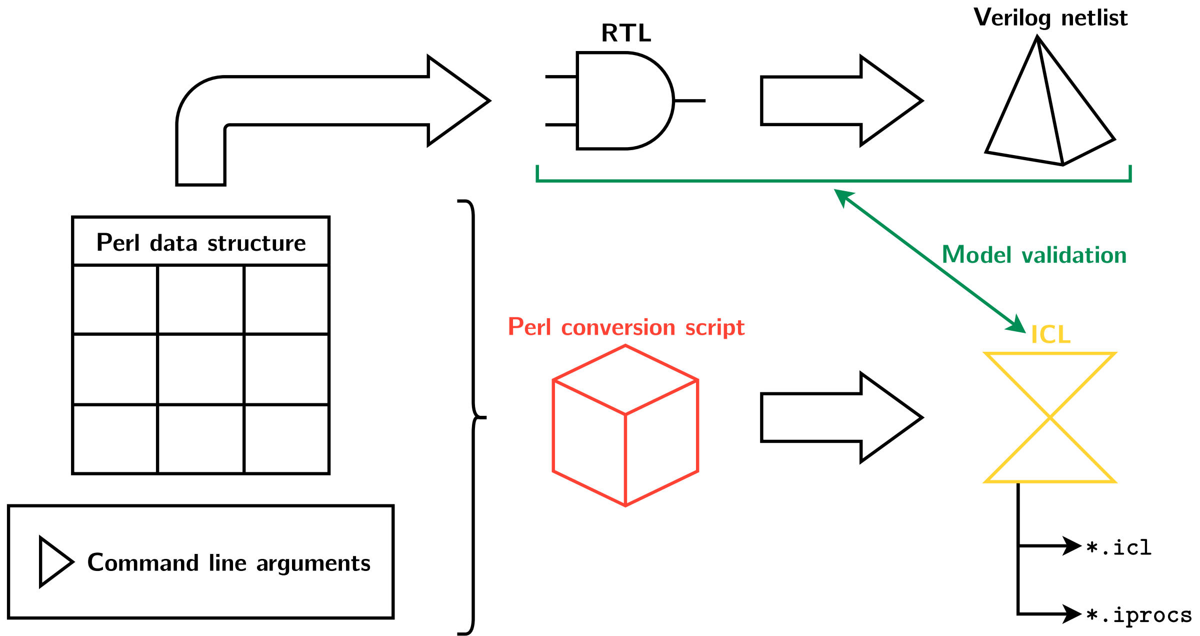


Fig. 4.1 Present state of the SSN flows for *Apple* test chips

## 4.2 Conversion script

The main deliverable of this working experience was represented by a `Perl` script responsible for automating the generation of a multitude of **.icl** and **.iprocs** files as per the IEEE 1687 standard. Its essential inputs are listed below.

**Perl data structure** A properly formatted set of design information stored in several **.pm** headers that can work as a unique "read-only" data source for both the RTL and the `ICL` descriptions, effectively eliminating any possible misalignment between the two. The intent is to ultimately render the code extremely flexible, in such a way that the same group of tables can be reused for different iterations of test chip implementations, regardless of the content (but given the same exact variable names, scopes, formats and locations). An extract of one of the preeminent hashes it entails is depicted below:

```
our %clusters = (
    %clusters,
    [...]
    cluster1 => {
        rlm       => "cpu",
        wrap      => 1,
        id        => 1,
        mem       => 1,
        run       => 1,
        auto      => 1,
        health    => 1,
        ref       => "CLU1_STR",
    },
    [...]
);
```

Here, the main attributes needed for the instantiation of each test unit are provided, e.g. information about the identifiers linked to a few of the aforementioned standards, several internal properties of the included logic structure, as well as references to some external records reserved to low-level component definitions.

**Command line arguments** The interface with the user is provided by means of an unordered and case-insensitive list of parameters that can be passed to the program in order to handle both mandatory configurations and optional selections, as in:

```
$ script.pl <rlm> <clusters> [options]
```

The two essential elements that must always be present are:

- `<rlm>` — a string used to choose one between many modular platforms that are hooked up to the corresponding voltage domain inside the final SOC, for instance `cpu`, `gpu` or `misc`
- `<clusters>` — a list of numbers of the clusters that need to be activated (and thus modeled) in the current configuration, it can be written as a sequence of decimal numbers, ranges or even by means of a binary One Hot-encoded number (e.g. `0b00001110` to select clusters 1 to 3).

The following flags are instead used to either turn on a particular feature or to specify a particular setting during run-time:

- `-a | --all` — a switch that automatically activates all available clusters inside the selected `<rlm>` (overriding `<clusters>`)
- `-h | --help` — a handy way to get a brief description of the script usage, printing a multi-line text similar to the current section onto the console output
- `-k | --keep` — a safety mechanism used to prevent the application from deleting (by default) all **.icl**/**.iprocs** files found inside the target directory
- `-p | --pipes` — the latest enhancement provided to the program, it allows choosing whether to manually model all the pipelines encountered inside the base architecture, or to let the Tessent™ tool trace them back automatically
- `-t | --target` — a preamble of the output folder for the `ICL` generation to take place into (defaulting to `./icl/`).

If not enough arguments are found, the script will simply print out the usage text (mimicking the `-h | --help` option), and will always try to skip redundant and/or invalid inputs while informing the user of whichever precautions are being taken during the initial parsing phase.

What's more, since the implementation of each cluster is unique inside any given RLM[1], the script has to be able to pinpoint all the related references and descriptors in order to fill in the gaps found inside a static template with many dynamic properties. As a result, the corresponding set of **.icl** and **.iprocs** files will be generated inside the target directory for every single one of them. For instance, the following call will activate clusters number 1, 2 and 3 for the `cpu` platform:

```
$ script.pl cpu 0b000001110 -t ../gen -p
```

Any existing trace of older runs found inside the same location will be deleted, right before creating the following files:

[1] Route-Level Module

```
../gen/icl/cpu_main.icl
../gen/icl/cpu_main.iprocs
../gen/icl/cpu1500Ctl.icl
../gen/icl/cpu1500Ctl.iprocs
../gen/icl/ssn_pipeline.icl
../gen/icl/cluster1_wir2.icl
../gen/icl/cluster1_wir2.iprocs
../gen/icl/cluster1_wdr.icl
../gen/icl/cluster1_wdr.iprocs
../gen/icl/cluster2_wir2.icl
../gen/icl/cluster2_wir2.iprocs
../gen/icl/cluster2_wdr.icl
../gen/icl/cluster2_wdr.iprocs
../gen/icl/cluster3_wir2.icl
../gen/icl/cluster3_wir2.iprocs
../gen/icl/cluster3_wdr.icl
../gen/icl/cluster3_wdr.iprocs
```

Note that the above result could also be obtained by using any other combination and format of program arguments, including (but not limited to):

```
$ script.pl 0b000001110 -p cpu --target ../gen
$ script.pl cpu -t ../gen --pipes 1 2 3
$ script.pl 1..3 cpu -p -t ../gen
$ script.pl --pipes --target ../gen cpu 1-3
```

Each created file will contain a standard header with the same format as the one automatically generated by Tessent™ Shell, keeping the whole ICL aligned to a unique style. These lines of text will simply contain the username of the local account that launched the program, plus the current date of the file system which is running the process, as in:

```
//----------------------------------------------------------------------
//
//----------------------------------------------------------------------
//  File created by: Fulvio Castello
//       Created on: 27.10.2023
//----------------------------------------------------------------------
```

Additionally, a few subroutines are used to both avoid duplication and reduce the amount of mnemonic steps that need to be taken by the programmer in order to access the input data structure, reducing the possibility of introducing flaws during development

while also beautifying the script itself. All the unavoidable hard-coded segments have been purposefully made as modular and as easily extensible as possible, while initial/constant values are clearly highlighted at the beginning of the code to be edited as needed for scalability purposes.

As a final note, regular expressions and interpolation of variables represent the main tools which sit at the core of this implementation: because of this (and for historical reasons), `Perl` proved to be the correct language of choice for the assigned task.

## IJTAG network

The implementation for the functionalities that are made accessible through the generated `ICL` model is depicted in Figure 4.2: here, a main W1500 controller is responsible for managing the currently selected test units as per the ECT[2] standard architecture. Its contents include the main WIR and WDR[3] (an alias for the more common WBR), which are used to (respectively) enable/disable and reset all available instances by means of a few dedicated signals, plus a "TOP" cluster that introduces the custom WIR2 layer. Said arrangement is to be intended as the driving source of configuration for the whole scan network, and all that comes later depends on it in a crucial manner. Inside each connected cluster, the WIR2 acts a selector in order to provide granular access to each WDR/SSN/EDT element, effectively opening up a path to the lower-level units inside the complex structure of a test chip.

The developed application was always intended to work seamlessly with two present architectures and possibly many future hardware implementations as well, so both scalability and versatility were of paramount importance. The two variants that were tested during the internship period are pictured in Figure 4.3, where the right side simply represents the successor to the left one. Here, all clusters are connected in a daisy-chain loop that connects a single SI port to a unique SO pin, but only those which were activated by the `<clusters>` program argument are modeled in `ICL`. The actual bypass functionality is achieved by means of the *yellow* multiplexers, together with the pipelining mechanism represented by the *green* storage elements. Aside from a different list of available clusters inside each structure, the main difference between the left and the right side of the above schematic resides in the distribution of these sequential units (and thus the number of cycles introduced by each different selection between the SI

[2] Embedded Core Test
[3] Wrapper Data Register

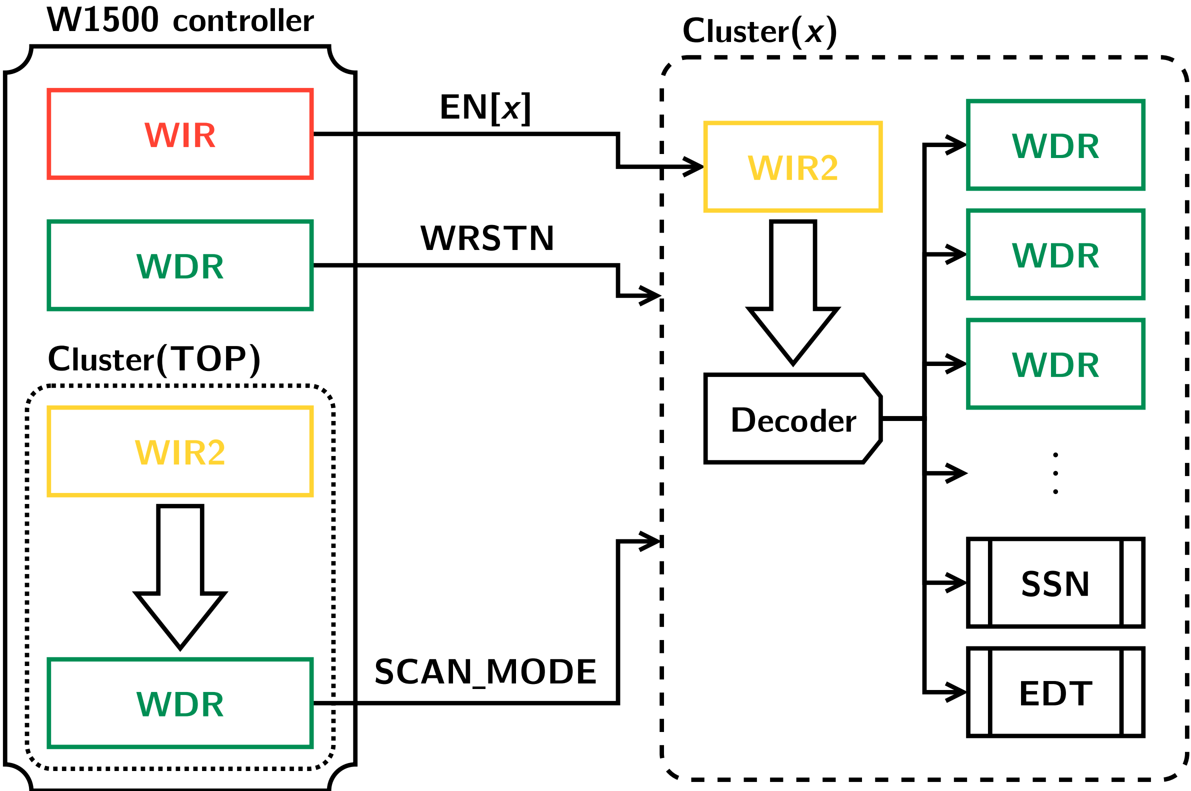


Fig. 4.2 Technical blueprint of the internal architecture

and the SO). In fact, each cluster in the older implementation needs one input and two output registers with the exception of “Cluster($n$-1)”, whereas said irregularity is no longer present in the newer one at the cost of adding a fixed pipeline right at the beginning of the chain.

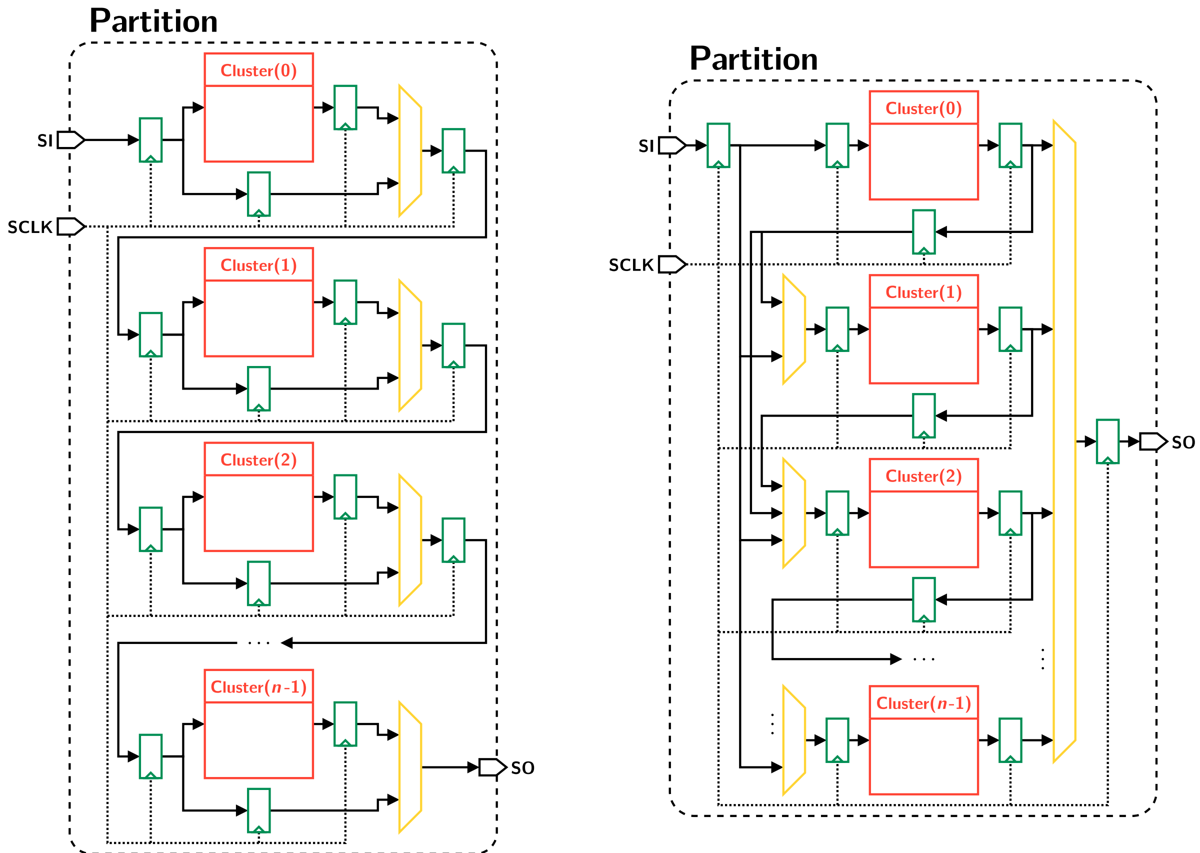


Fig. 4.3 Partition-level view of two successive bypass architectures

The same issue was later eliminated altogether thanks to a newly introduced feature of Tessent™ Shell called “pipe tracing”. The latter consists in letting the automated tool proceed backwards within the hierarchy, autonomously inferring all the encountered storage elements as to complement the `ICL` model on its own. The `-p | --pipes` switch is responsible for triggering the creation of the pipeline network, which is left out by default when the same program argument is not specified. At the time of writing, said experimental feature is still part of a snapshot version of the aforementioned *Siemens* software suite and, as such, requires a dedicated access code to be made available during the configuration of the ensuing trials.

## 4.3 Model verification

After having successfully coded a functioning application, it was necessary to check the validity of its outputs. In this case, the principal task was to demonstrate a 1:1 identity between the RTL "golden" reference and the newly generated `ICL` characterization, to then proceed to its validation against the ensuing netlist that would be used to generate test patterns. Such a sought-after property was eventually proven by means of several `Verilog` simulations with increasing abstraction levels. In fact, starting with one cluster only, their scope was gradually expanded to include more and more units, eventually entailing whole partitions to finally reach entire chip-level runs.

Even if the actions performed on both the known test chip variants were the same, the line of thought differed slightly while working on them. On one hand, the first design had already been 100% manually verified previously, and just served as a sort of "playground" on which to test the `Perl` deliverable in order to try and plug it into the internal processes, especially aiming at chip-level pattern generation. On the other hand, the second one provided a more compact hardware architecture with less clusters and RLMs, but the correctness of its SSN implementation was yet to be checked altogether. This meant that a simulation "FAIL" could either indicate that the `ICL` model was not compliant with the underlying RTL description or even that the latter was plainly wrong in the first place.

The sequence of steps used to achieve the aforementioned identity was ordered as depicted in Figure 4.4:

**TCL configuration** Starting from a template generated by an internal ATPG[4] tool, the corresponding **.tcl** file would need several specific tweaks in order to correctly set up the environment for loading both the RTL and the `ICL` models before going through the ensuing DRC[5] phase for scan chain tracing. This is the step in which the `Perl` script was seamlessly inserted, so that the required `ICL` for any given test case could be automatically written out just by turning on a switch and letting the application pick up a few internal variables in the form of program arguments

**Patterns generation** Once the rule verification step completes successfully inside Tessent™ Shell, it becomes possible to generate the chosen type(s) of pattern set and write out the related testbench files. Here, a multitude of internal parame-

[4]Automatic Test Pattern Generation
[5]Design Rule Checking

ters have to be carefully specified in order to both define the correct attributes of the DUT and define a proper location for the run to take place into

**Build phase** This optional event is in charge of compiling and elaborating the `Verilog` design before launching the ensuing simulation. Since all generated testbenches are generic and already suitable for every supported pattern set, this step can be skipped during successive runs as long as no modifications are done to the code

**Run phase** Once all the previous settings have been consolidated, the actual trial is finally started by calling the requested testbench configuration (e.g. continuity/loop-back/chain). After an initial test setup sequence is executed in order to prepare the DUT for the selected test case, an additional setup step is managed by the *Siemens* tool as to correctly initialize the SSN datapath, pre-assign the payload to each SSH and assert the correct SIB combination. After a variable period of time, the final result will be expressed by means of a clearly visible “PASS”/“FAIL” message, explicitly stating all existing errors (if any) and dumping a waveform database for debugging purposes

**Debug sessions** Obviously, each of the above bullet points can take place if and only if its predecessor goes through successfully. At any given moment, every error had to be inspected and fixed before trying to proceed within the verification flow, so that this entry would be repeated multiple times and ended up taking the vast majority of both human hours and effort during the whole internship period.

Fig. 4.4 State diagram summarizing the verification flow

# Chapter 5

# Results

The positive impact of what was developed during the internship period left nothing to be desired in terms of overall productivity for the DFT team at *Apple* Munich. In fact, since SSN was not available altogether before the conception of this project, the effort put into creating an automated flow to support it in both current and future test chip architectures represented more of a necessity rather than a simple improvement. The subsequent sections have been dedicated to diving into each of the key advantages introduced by this solution, providing meaningful data to support and highlight their significance regarding three major metrics:

1. Human hours/CPU[1] time
2. Intrinsic code correctness
3. Innate reuse capabilities.

## At a glance

Overall, the related benchmarks can be outlined by drawing a direct comparison between the same methodologies before and after the product of this experience was put into use:

| *Metric* | **Before** | **After** |
|---|---|---|
| Delay | weeks | seconds |
| Flaws | likely | absent |
| Reuse | taxing | seamless |

[1]Central Processing Unit

It is of vital importance to underline the fact that this chapter is in no way meant to discredit the state of the art that existed before the integration of the work in question. All the ensuing comparisons are just aimed at showing the effectiveness of an idea that the members of the team themselves had in the first place, given that an implementation for the functionality described in this report was, in principle, non-existent anyway.

## 5.1 Timeliness

The initial `ICL` template took around one week in total to be firstly drafted manually, and its initial verification was carried out during the consecutive months in order to prove both its correspondence with the already existing RTL model and its compatibility with Tessent™ Shell. After this, the obtained file hierarchy was hardly usable, and laboriously reached a very sought-after simulation "PASS" for a very limited number of clusters and after many case-specific `TCL` settings had been applied. Of course, extending such arrangement for any other different combination would again require almost the same amount of effort, effectively meaning that a fresh start would be needed each time.

On the other hand, the `Perl` script described in the previous chapter only requires one initial trigger in order to autonomously generate the set of **.icl/.iprocs** files that make up the alternative model essential to the *Siemens* tool. What's more, its total run-time exclusively depends on the server load in the instant in which it gets called, ranging from a few seconds to a maximum of less than a minute in total (independently of the number of active clusters inside the selected partition). The verification flow built around it has been seamlessly integrated into the internal procedures in such a way that all test cases should be covered automatically, or require very little human intervention in order to reach a positive simulation outcome.

## 5.2 Trustworthiness

Fortunately, the initial phase of manually verifying the handcrafted `ICL` template was far from useless and served as a crucial starting point upon which it became possible to integrate a fully-fledged SSN automation flow. Once the correctness of said baseline was proven by a manual course of action, the main endeavor of the task assigned during the internship period was to generalize that property, in such a way that all the outputs could then be intrinsically trusted upon. Thanks to this, the probability of error inside a piece of code that has already been extensively validated (and should thus give birth to

impeccable outputs) realistically plummets to zero. Of course, the same could not be stated referring to the original methodology, which entailed a series of error-prone "copy and paste" operations.

An additional benefit to the solution hereby presented is that the requested file hierarchy gets instantiated locally to the final user starting from a centralized data structure. Because of this, any external modification that the latter might have to undergo (e.g. if the ensuing RTL description has to be altered) will automatically appear inside the consequent `ICL` description as well, leaving the two parallel models aligned by definition. Unlike before, a unique "golden" reference for a set of up-to-date **.icl/.iprocs** files will no longer have to reside in the work area of whoever its maintainer happens to be, since all the members of the DFT team can just get a functional copy effortlessly created for them at any given moment.

## 5.3 Scalability

Before the establishment of an automated set of processes inside the verification flow of *Apple* test chips, almost all the enclosed steps correlated to a unique scenario and featured an overwhelming abundance of hard-coded values and commands. All this resulted in the need to conceive a custom array of constraints and hierarchies for every single test case, requiring a lot of time and manual effort to restart from the ground up each time. Aside from the annoyance brought about by this method, the most crucial factor was represented by the unknown validity of the obtained output, meaning that everything would again need to be checked for its correctness and purposefully tweaked in order to reach a satisfactory level of certainty.

The presented solution not only nullifies the above issue by implementing a generic sequence of actions that automatically applies to all possible cluster configurations, but goes way beyond just that scope. In fact, the `Perl` script in particular was conceived as to be easily extensible not only to all possible changes/additions within its project of reference, but also to entire future test chip architectures with little to no need of code manipulation. This property is clearly demonstrated by the fact that the solution developed during the internship period contributed (and were later deemed essential) to the silicon verification of the first SSN-capable device produced by the company, the delivery of an SSN-ready RTL description of its direct successor, and is already being plugged into even farther iterations down the pike.

# Chapter 6

# Conclusion

This Master's Thesis presents an automation flow able to support the integration of the Streaming Scan Network technology into both present and future test chip iterations at *Apple Inc.*, detailing the output of six months of full-time employment at the DFT team in Munich. The need for such a cutting-edge answer to ever-increasing test complexity, time and cost is exemplified by the continued search for a flexible mechanism aimed at verifying different components inside the platform.

The *Siemens* Tessent™ software suite requires a simplified characterization of the design in question in order to provide IJTAG functionality to its scan elements. Consequently, an `ICL` model has to be compared against its RTL counterpart in order to ensure their 1:1 correspondence, so that the ensuing netlist can be used to generate patterns for simulation purposes. This non-trivial task is the pivot around which the assigned project revolved, so that finding an automated way of making all upcoming architectures SSN-ready immediately proved to be an absolute necessity.

The first phase of the internship period was devoted to studying the underlying standards that the aforementioned technology yields, therefore developing an executable script that could generate a correct `ICL` description of the chosen hardware arrangement starting from a common `Perl` data structure. This step required the mastery of the same programming language (i.e. `Perl`) as well as the analysis of many novel digital design concepts and methods pertaining both to the intellectual property of the company and to public literature. The code was laid out starting from a handcrafted template, and was clearly commented and structured in order to be as easily extensible as possible.

Once the correctness of the ensuing set of generated files was proven, a brand-new, adaptable flow was put into place as to optimize the whole testing mechanism of different cluster combinations inside the selected DUT. The generalization of all the settings and

rules it contains required a remarkable effort, and ended up resulting in a fast, scalable solution that will be integrated into all successive architectures without having to be altered in any significant way. The necessity for human intervention has thus been confined to a few manual triggers aimed at just starting the verification sequence in order to ultimately receive a “PASS”/“FAIL” outcome to be debugged at the user’s discretion.

Aside from the countless technical aspects that were analyzed during this experience, the personal and professional growth that came along with working in such a stimulating, respectful but also demanding environment truly represents the core essence of this whole opportunity. More in particular, the mid-year deadline represented by a major internal milestone for one of the targeted projects really required an indisputable level of passion and dedication in order to manage a somewhat serious degree of pressure and to meet the highest standards required by the company. This would soon after create an immense sense of gratification and accomplishment, since the snippets created until then proved to be a key element in the final delivery of the requested items, effectively impacting the performance of many other coworkers in a positive way.

Finally, the output of the development described so far is already a crucial part of the production methodologies used internally, and is able to automatically pick up any architectural changes in order to seamlessly adapt to all imminent project revisions. Possible future work ideas could include some improvements to the `ICL` composition itself, expanding it as to support additional features provided by Tessent™ Shell, or even rearranging the base design description with the aim of making it more modular and thus increasing its overall scope.

# Nomenclature

**Acronyms / Abbreviations**

| | |
|---|---|
| AMS | Analog and Mixed Signal |
| ATE | Automated Test Equipment |
| ATPG | Automatic Test Pattern Generation |
| BIST | Built-In Self-Test |
| CPU | Central Processing Unit |
| DFT | Design For Testing |
| DRC | Design Rule Checking |
| DUT | Device Under Test |
| ECT | Embedded Core Test |
| EDA | Electronic Design Automation |
| EDT | Embedded Deterministic Test |
| FSM | Finite State Machine |
| I/O | Input/Output |
| IC | Integrated Circuit |
| ICL | Instrument Connectivity Language |
| IEEE | Institute of Electrical and Electronics Engineers |
| IJTAG | Internal JTAG |

| | |
|---|---|
| IP | Intellectual Property |
| JTAG | Joint Test Action Group |
| LFSR | Linear Feedback Shift Register |
| MBIST | Memory BIST |
| NRE | Non-Recurring Engineering |
| PDL | Procedural Description Language |
| PLL | Phase Lock Loop |
| RLM | Route-Level Module |
| RTL | Register-Transfer Level |
| SIB | Segment Insertion Bit |
| SI | Scan In |
| SOC | System On a Chip |
| SO | Scan Out |
| SSH | Streaming Scan Host |
| SSN | Streaming Scan Network |
| TAP | Test Access Port |
| TCK | Test Clock |
| TCL | Tool Command Language |
| TMS | Test Mode Select |
| UPF | Unified Power Format |
| VLSI | Very Large Scale Integration |
| WBR | Wrapper Boundary Register |
| WDR | Wrapper Data Register |

| | |
|---|---|
| WIP | Wrapper Interface Port |
| WPI | Wrapper Parallel Input |
| WPO | Wrapper Parallel Output |
| WRCK | Wrapper Clock |
| WSI | Wrapper Serial Input |
| WSO | Wrapper Serial Output |

# References

[1] Andrews, J. (1991). IEEE Standard Boundary Scan 1149.1: an Introduction. In *Electro International, 1991*, pages 522–527. https://doi.org/10.1109/ELECTR.1991.718268.

[2] Arya, N. and Singh, A. P. (2016). IEEE 1149.1 Test Access Port (JTAG) Verification Using Verilog Simulation. In *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, pages 106–111. https://doi.org/10.1109/ICEEOT.2016.7754838.

[3] Brown, J. (2012). Example of how IEEE P1687 (IJTAG) Creates IP Test Standardization and Portability. https://grouper.ieee.org/groups/1687/IJTAG_ITC_Poster_2012_rev3.pdf.

[4] Côté, J.-F., Kassab, M., Janiszewski, W., Rodrigues, R., Meier, R., Kaczmarek, B., Orlando, P., Eide, G., Rajski, J., Colon-Bonet, G., Mysore, N., Yin, Y., and Pant, P. (2020). Streaming Scan Network (SSN): an Efficient Packetized Data Network for Testing of Complex SoCs. In *2020 IEEE International Test Conference (ITC)*, pages 1–10. https://doi.org/10.1109/ITC44778.2020.9325233.

[5] DaSilva, F., Zorian, Y., Whetsel, L., Arabi, K., and Kapur, R. (2003). Overview of the IEEE P1500 Standard. In *International Test Conference, 2003. Proceedings. ITC 2003.*, volume 1, pages 988–997. https://doi.org/10.1109/TEST.2003.1271086.

[6] Eide, G. (2021). Streaming Scan Network: No-Compromise Packetized Test. Technical report, Siemens Digital Industries Software. https://resources.sw.siemens.com/en-US/white-paper-streaming-scan-network-a-no-compromise-approach-to-dft.

[7] Huang, Y., Cheng, W.-T., and Rajski, J. (2005). Compressed Pattern Diagnosis for Scan Chain Failures. In *IEEE International Conference on Test, 2005.*, pages 1–8. https://doi.org/10.1109/TEST.2005.1584037.

[8] Liu, Z. (2020). *A Test Chip Design for Automatic Insertion of Logic Circuit Demographics*. PhD thesis, Carnegie Mellon University. https://doi.org/10.1184/R1/12167925.v1.

[9] Payakapan, T., Kan, S., Pham, K., Yang, K., Côté, J.-F., Keim, M., and Dworak, J. (2015). A Case Study: Leverage IEEE 1687 Based Methods to Automate Modeling, Verification, and Test Access for Embedded Instruments in a Server Processor. In *2015 IEEE International Test Conference (ITC)*, pages 1–10. https://doi.org/10.1109/TEST.2015.7342407.

[10] Poehl, F., Beck, M., Arnold, R., Muhmenthaler, P., Tamarapalli, N., Kassab, M., Mukherjee, N., and Rajski, J. (2003). Industrial Experience with Adoption of EDT for Low-Cost Test without Concessions. In *International Test Conference, 2003. Proceedings. ITC 2003.*, volume 1, pages 1211–1220. https://doi.org/10.1109/TEST.2003.1271110.

[11] Rajski, J., Tyszer, J., Kassab, M., and Mukherjee, N. (2004). Embedded Deterministic Test. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(5):776–792. https://doi.org/10.1109/TCAD.2004.826558.

[12] Reynick, J. (2021). SSN and Deterministic On-Chip Compare. https://resources.sw.siemens.com/en-US/video-ssn-and-deterministic-on-chip-compare.

[13] Vermaak, H. (2005). *Design-for-Delay Testability Techniques for High-Speed Digital Circuits*. PhD thesis, University of Twente. http://doc.utwente.nl/57440/.

[14] Zakiy, M. F. (2016). HW-SW Co-Design of an On-Chip IJTAG Dependability Processor. Master's thesis, University of Twente. https://purl.utwente.nl/essays/70623.