UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Posgrados

SoC Design For Test Automation

Tesis en torno a una hipótesis o problema de investigación y su contrastación

Raúl André Borja Cajiao

Luis Miguel Prócel, Ph.D.

Director de Trabajo de Titulación

Trabajo de titulación de posgrado presentado como requisito
para la obtención del título de Magíster en Nanoelectrónica
Mención en Sistema Embebido e Integración

Quito, 7 de diciembre de 2022

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

COLEGIO DE POSGRADOS

HOJA DE APROBACIÓN DE TRABAJO DE TITULACIÓN

SoC Design For Test Automation

Raúl André Borja Cajiao

Nombre del Director del Programa:	Luis Miguel Prócel
Título académico:	Doctor of Philosophy
Director del programa de:	Maestría en Nanoelectrónica
Nombre del Decano del colegio Académico:	Eduardo Alba
Título académico:	Doctor of Philosophy
Decano del Colegio:	Colegio de Ciencias e Ingenierías
Nombre del Decano del Colegio de Posgrados:	Hugo Burgos
Título académico:	Doctor of Philosophy

Quito, diciembre 2022

© DERECHOS DE AUTOR

Nombre del estudiante:	Raúl André Borja Cajiao
Código de estudiante:	00323173
C.I.:	1722643275
Lugar y fecha:	Quito, 7 de diciembre de 2022.

ACLARACIÓN PARA PUBLICACIÓN

Nota: El presente trabajo, en su totalidad o cualquiera de sus partes, no debe ser considerado como una publicación, incluso a pesar de estar disponible sin restricciones a través de un repositorio institucional. Esta declaración se alinea con las prácticas y recomendaciones presentadas por el Committee on Publication Ethics COPE descritas por Barbour et al. (2017) Discussion document on best practice for issues around theses publishing, disponible en http://bit.ly/COPETheses.

UNPUBLISHED DOCUMENT

Note: The following graduation project is available through Universidad San Francisco de Quito USFQ institutional repository. Nonetheless, this project – in whole or in part – should not be considered a publication. This statement follows the recommendations presented by the Committee on Publication Ethics COPE described by Barbour et al. (2017) Discussion document on best practice for issues around theses publishing available on http://bit.ly/COPETheses.

AGRADECIMIENTOS

El tiempo que pasé en NXP Semiconductors como pasante de marzo a agosto de 2022 ha sido realmente valioso para mí, ya que enriqueció mi experiencia tanto a nivel personal como profesional. Me permitió explotar mi potencial en un equipo internacional altamente competitivo y comprometido, ayudándome a desarrollar y descubrir nuevas habilidades invaluables que moldearán para siempre mi vida profesional y mi crecimiento personal.

En primer lugar, me gustaría expresar mi más sincero agradecimiento a mi tutor Vincent CHALENDARD por ofrecerme esta oportunidad de pasantía y guiarme a trabajar en un proyecto emocionante y desafiante. Su guía caracterizada por su paciencia, entusiasmo e inmenso conocimiento me motivó durante el desarrollo de este trabajo. Finalmente, agradezco sinceramente la confianza que depositó en mí al permitirme participar en reuniones de gran relevancia con proveedores externos y ser parte de decisiones importantes para proyectos futuros.

Extiendo mi gratitud y aprecio al equipo de NXP DFT, especialmente a Frederic HIEBEL y Herve VINCENT por su ayuda, apoyo y consejos durante mi pasantía. Este estudio no hubiera sido posible sin su valioso y continuo trabajo.

Un gran agradecimiento a Pascal CUSSONEAU y Paul BRADY por su confianza y convicción en mi trabajo, y especialmente por su cálida bienvenida y motivación durante mi tiempo en NXP que me permitió sentirme como un miembro valioso del equipo desde el primer día de mi internado. Finalmente, me gustaría agradecer a todos mis colegas de NXP que hicieron posible este trabajo.

Un profundo agradecimiento y aprecio a mis profesores que fueron parte fundamental de mi desarrollo académico, especialmente a mis directores de maestría, Luis Miguel Prócel en Ecuador (Universidad San Francisco de Quito) y Adam QUOTB en Francia (Institut National Polytechnique de Toulouse).

ACKNOWLEDGEMENTS

The time I spent at NXP Semiconductors as an intern from March to August 2022 has been truly valuable for me since it enriched my experience both personally and professionally. It allowed me to explode my potential in a highly competitive and engaged international team helping me to develop and discover new invaluable skills that will forever shape my professional life and personal growth.

Foremost, I would like to express my sincere gratitude to my tutor Vincent CHALENDARD for offering me this internship opportunity and leading me working on an exciting and challenging project. His guidance characterized by his patience, enthusiasm and immense knowledge motivated me during the development of this work. Finally, I sincerely appreciate the trust he placed in me by allowing me to participate in highly relevant meetings with external providers and to be part of important decisions for future projects.

I extend my gratitude and appreciation to the NXP DFT team, specially to Frederic HIEBEL and Herve VINCENT for their help, support and advice during my internship. This study would not have been possible without their valuable and continuous work.

A very great thankful to Pascal CUSSONEAU and Paul BRADY for their confidence and conviction in my work, and especially for their warm welcome and motivation during my time at NXP that allowed me to feel like a valued member of the team from the first day of my internship. Finally, I would like to thank all my colleagues from NXP that made this work possible.

A deep gratitude and appreciation to my professors who were a fundamental part of my academic development, especially my master's directors, Luis Miguel Prócel in Ecuador (Universidad San Francisco de Quito) and Adam QUOTB in France (Institut National Polytechnique de Toulouse).

RESUMEN

En el presente trabajo se estudia la importancia de garantizar la capacidad de prueba de un producto y como implementarla durante el proceso de diseño de un circuito integrado. Durante este estudio se analizaron y aplicaron distintos estándares utilizados a nivel industrial en el diseño de chips como el IEEE 1149.1 para garantizar la capacidad de prueba en los dispositivos. Se desarrolló una comparación del uso de dos herramientas “TASS” y “IJTAG” para el diseño de pruebas utilizadas por la compañía NXP Semiconductors. Se estudió, modificó y automatizó el flujo completo de diseño de pruebas de la compañía utilizando un producto ya desarrollado y probado como vehículo de prueba. Finalmente se analizaron los resultados de la comparativa demostrando las ventajas y desventajas del uso de las dos herramientas en función de los requerimientos de la compañía y el equipo de trabajo para futuros proyectos.

Palabras clave: System on Chip, Design for Test, Circuito Integrado, Automatización, TASS, IJTAG.

ABSTRACT

This work studies the importance of guaranteeing the testability of a product and how to implement it during the design process of an integrated circuit. During this study, different standards used at an industrial level in chip design such as IEEE 1149.1 were analyzed and applied to guarantee the testability of the devices. A comparison of the use of two tools "TASS" and "IJTAG" was developed for the design of tests used by the company NXP Semiconductors. The company's entire test design flow was studied, modified, and automated using an already developed and tested product as a test vehicle. Finally, the results of the comparison were analyzed, demonstrating the advantages and disadvantages of the use of the two tools based on the requirements of the company and the work team for future projects.

Key words: System on Chip, Design for Test, Integrated Circuit, Automation, TASS, IJTAG.

NXP SEMICONDUCTORS FRANCE

Final Internship Report

SoC Design For Test Automation

Student:
Andre BORJA
USFQ - INP ENSEEIHT

Internship Mentor:
Vincent CHALENDARD
NXP Semiconductors

France

September 2022

CONTENT TABLE

TABLE INDEX

FIGURE INDEX

SOC DESIGN FOR TEST AUTOMATION

1 Introduction

1.1 NXP Semiconductors France

NXP Semiconductors is a worldwide known semiconductor design and manufacture company with headquarters in Eindhoven, Netherlands. Stablished in 2006, it is currently present in 33 countries around the word. NXP is internationally recognized and considered as one of the most important providers for the automotive, communications and mobile industries. It has been the first supplier of microcontrollers with 19% of market share for very prestigious consumers [1].

NXP in France includes 4 sites located in Toulouse, Caen, Paris and Mougins. The R&D (Research and Development) site on Cote d'Azur in Mougins is well known thanks to the development of secure connectivity solutions [2]. Thus, the NXP business line inspired in these solutions is C&S (Connectivity and Security), responsible to design mixed signal devices for secured transactions (NFC and/or Secure Element Controllers) specially for the mobile devices market.

1.2 Context

During the process of Digital SoC Design it is fundamental to guarantee the testability of the product. Ensuring the quality and correct operation of electronics is fundamental to keep and improve the market position of a brand or company, it reduces the design and production testing times, resulting in economic benefits. Detecting failures in the earliest stages of a product development is fundamental to save costs and time. Due to this reason, it is necessary to add certain features to the design to make easier the testability of the product and to ensure that all possible failures generated during the manufacture process are covered and able to be detected before reaching the costumer.

DFT (Design for Test) is a design process that describes how testing must be done. This process includes several stages such as the development of the DFT design specifications, the translation of this specifications into DFT structures and the corresponding RTL description. The flow continues with the Verification of this DFT structures and their interconnections and ends with the Software (test patterns) creation with its corresponding testbench.

This internship study aims to analyze and compare two DFT tools used in the Software (test patterns) creation and verification. This comparison will be useful in the migration process from the current NXP DFT tool "TASS" to the desired third part tool "IJTAG" (Siemens former Mentor Graphics proprietary). The result of this work is the automation of the DFT design process through scripting (using Shell, TCL, Perl and/or Python).

The motivation for this work is to determine if the new tool accomplishes with the requirements of the NXP DFT team for pattern generation. Finally, it is necessary to provide enough proofs through a deep analysis and simulation results in order to demonstrate that the desired tool for the migration is or is not adequate and provides significative benefits during the NXP DFT flow in terms of design time, complexity, costs, among others.

1.2 Objectives

Identify the differences between TASS and IJTAG, their benefits and drawbacks for pattern description and generation.

Adapt and/or create a new NXP DFT flow for the new tool and report the main changes for the new flow.

Handle the corresponding test pattern creation.

Simulate the test patterns and debug them on RTL and Gate-Level Netlist using Cadence Xcelium tool.

Develop the DFT design process automation through scripting methods in order to generate automatically the files/directories for the new flow.

Make a proof of concept through on-silicon trials using an ATE with test patterns generated by the new IJTAG tool.

Demonstrate if the migration is feasible and represents a real benefit for the new DFT flow and if not, give enough proofs and arguments to justify this decision.

2 State of the Art

In order to guarantee the testability of an IC (Integrated Circuit) it is necessary to have physical access to its internal components, be able to configure desired features and have the capability to measure the response of the circuit based on the applied stimulus. Because of those requirements, a four-port serial interface called JTAG (Joint Test Action Group) was developed in the late 1980s [3]. This is the basis of the IEEE Standard 1149.1, the IEEE Standard TAP (Test Access Port) and the Boundary-Scan Architecture which together support the mentioned testability requirements [4].

2.1 IEEE Standard 1149.1

The IEEE Standard 1149.1 also known as JTAG, or boundary-scan stablishes a four-port serial JTAG interface to access the embedded logic of an IC. This debug port interface is connected to a TAP embedded on-chip which acts as a JTAG main controller during the test. The TAP interface includes four main signals: TCK (Test clock), TMS (Test Mode Select),

TDI (Test Data Input) and TDO (Test Data Output); and an optional reset signal TRST (Test logic reset).

The JTAG standard for the architecture stablishes one Instruction Register (IR) with several Data Registers (DR). Through these structures it is possible to load or configure specific features for the IC as well as read or measure its response. The IR is responsible for providing the necessary address and control signals to access a specific DR. This control is done by the TAP by loading a specific OPCODE (Operation Code or Instruction) in the IR. The DR is responsible of controlling specific test configurations for the IC when it is written. Besides, the DR is in charge of capturing some internal signals data to be read or shifted out for test purposes.

The size of the IR and the number of supported instructions depends on the design and requirements (e.g., an 8-bit IR supports up to 256 instructions). Each DR is linked to an Instruction but not all instructions require a DR. The JTAG standard stablishes three mandatory instructions (BYPASS, EXTEXT, SAMPLE/PRELOAD). The other DR and instructions depend on the IC design.

The NXP DFT Reference Architecture propose two main types of Data Registers which are TCB (Test Control Block) and TPR (Test Point Register). TCBs can control some IC features depending on the "mode" that is written on them, but they are not physically designed to capture internal data to be read or shifted it out. Moreover, TPRs can control some IC features when writing on them but also, they can capture internal signals data to be shifted out. Through Figure 1 it is observed a basic JTAG TAP control structure example with TCBs and TPRs chains.

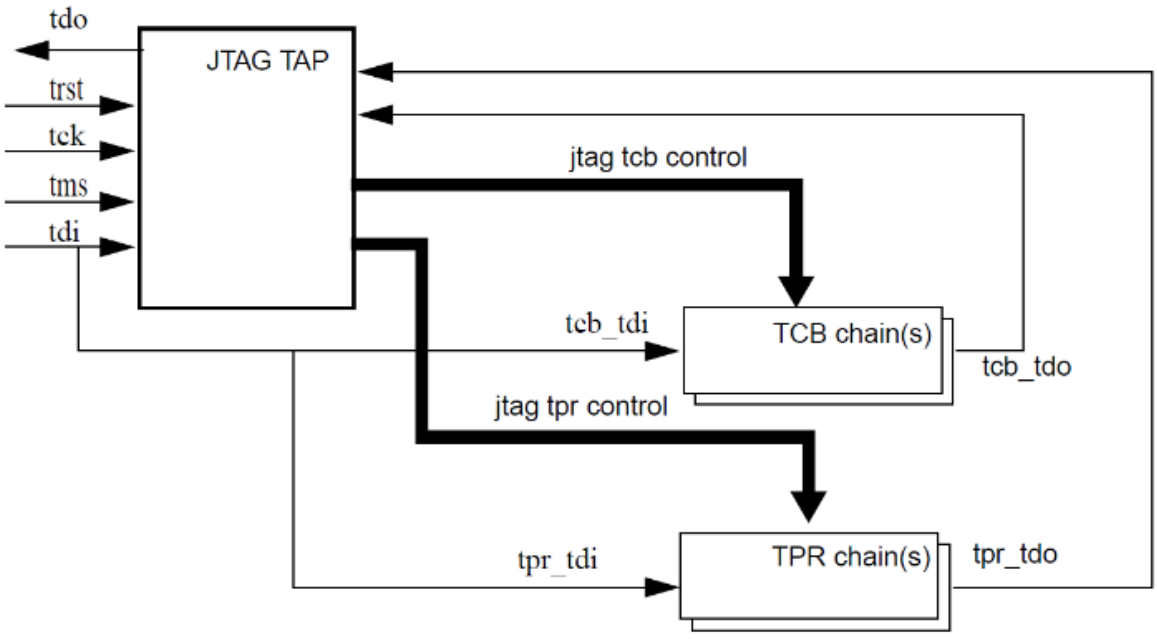
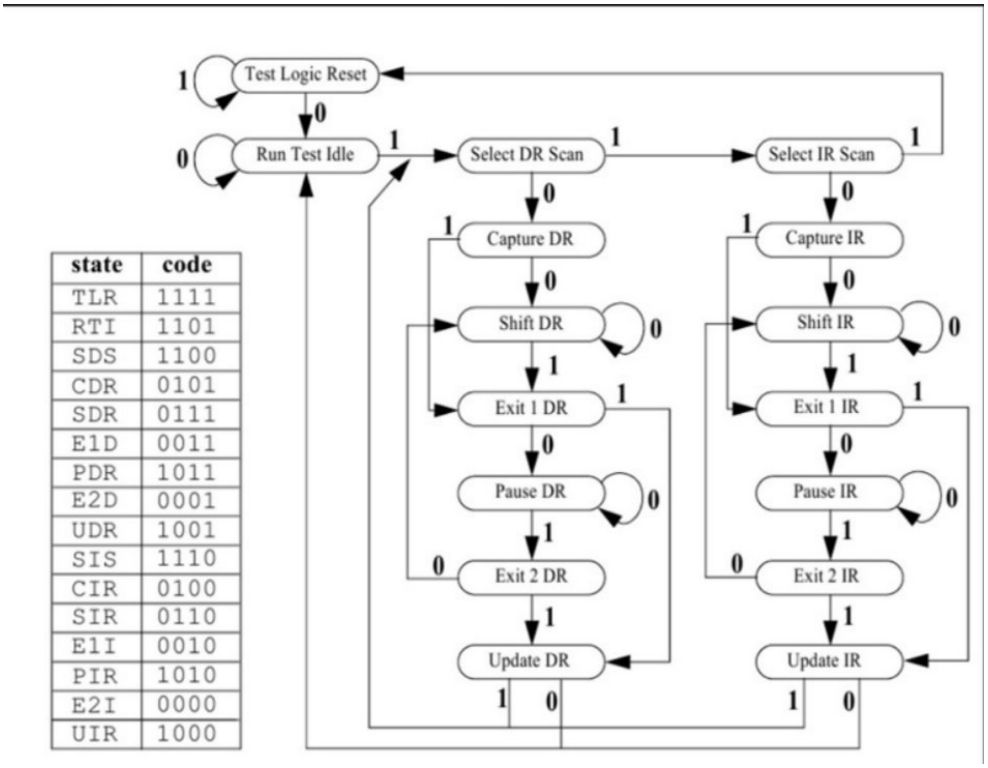


Figure 1: JTAG TAP TCB-TPR Control Structure [5].

It is seen that the JTAG TAP interface corresponds to the JTAG standard port interface including the four main signals and the reset optional signal. It is interesting to mention that in the design TDI is broadcast directly to the TCBs and TPRs while the clock signal TCK is gated through the JTAG TAP structure to the DR blocks through the JTAG TCB and TPR control buses. The data out signals from each DR is gated through the JTAG TAP to the TOP level TDO. Furthermore, through Appendix Figure 1 it can be observed each of the control signals delivered both to the DRs and the IR.

2.2 Test Access Port Architecture

The IEEE Standard 1149.1 stablishes a default structure for the TAP controller FSM (Finite State Machine) which can be found through Figure 2. The TAP is controlled thanks to the test clock TCK, and the test mode select TMS inputs. Consequently, these inputs are in charge of handling the access to the DRs and IRs in order to write and/or read them. According to the JTAG standard the test bus uses both clock edges of TCK. TMS and TDI signals are sampled on the rising edge of TCK while TDO output changes after the falling edge of TCK.



state	code
TLR	1111
RTI	1101
SDS	1100
CDR	0101
SDR	0111
E1D	0011
PDR	1011
E2D	0001
UDR	1001
SIS	1110
CIR	0100
SIR	0110
E1I	0010
PIR	1010
E2I	0000
UIR	1000

Figure 2: TAP Finite State Machine.

The adjacent value shown close to each state transition indicates the required TMS signal on the rising edge of TCK to generate the state change. The JTAG Standard stablishes 5 steady states when TMS is 0: Run Test/Idle, Shift-DR, Pause-DR, Shift-IR, Pause-IR; but only 1 steady state when TMS value is 1: Test Logic Reset. This particular feature of the TAP FSM allows to reset the test logic (i.e., soft reset) by setting TMS input high during at least 5 TCK cycles independently of the current state.

2.3 JTAG Data formats & Pattern Description

As mentioned above, the main goal of DFT is to ensure the testability of an IC. To achieve this, it is necessary to develop software that allows signals to be input to the IC and executed to test the device. This software is known as test patterns which are nothing more than a group of stimulus or signals applied to the chip in order to measure or test specific features and ensure the device is working properly. Those patterns generate a signal response that are analyzed by comparing them with a testbench simulation. In order to describe this software, it is necessary to have a link with the hardware defined previously. This link is generated through a hardware representation using a data format that supports IEEE Std 1149.1. Currently there are several data formats capable to understand the DFT hardware structures such the ones mentioned in the JTAG standard. One of this data formats is the industry standard language originated from VHDL: Boundary-Scan Description Language (BSDL) (along this document it will be studied other data formats for similar capabilities and purposes). The main objective of BSDL and other languages is to describe how the IEEE Std 11.49.1 is implemented in the IC and specially how to access and control it.

Once there is a representation of the DFT hardware, it is possible to write or describe the test patterns representing the stimulus and expected response. There are many ways to describe test patterns, one of them is by using a pattern generation tool (such as TASS or IJTAG). These tools receive as an input the pattern description in a human-readable format. Then, they are going to verify that the DFT hardware description is compliant with the standard and automatically create a pattern description in a supported language for a test equipment or ATE (Automatic Test Equipment) which validates physically the IC with the patterns. Additionally, these tools are used to generate simulation testbenches to verify the expected behavior and debug the pattern description if necessary.

3 My Project: Design For Test Automation

3.1 Context

3.1.1 TASS

One of the pattern generation tools widely used on NXP during DFT flow is TASS or Test Pattern Assembler [6]. TASS is an internal tool part of the NXP DFT package designed to assemble test patterns in order to generate test vectors for ATE test systems and simulation testbenches. The used version on this study is TASS 8.0.6. Through Figure 3 it is observed the basic flow to use TASS tool.

TASS inputs consist of one or more of the following files:

1. Test Data Files (.td)
2. Application Test Files (.atf)
3. Waveform Generation Language Files (.wgl)
4. Test Protocol Files (.tpf)
5. STIL Files (.stil)
6. Waveset Definition Files (.wdf)
7. Command files (.cmd)

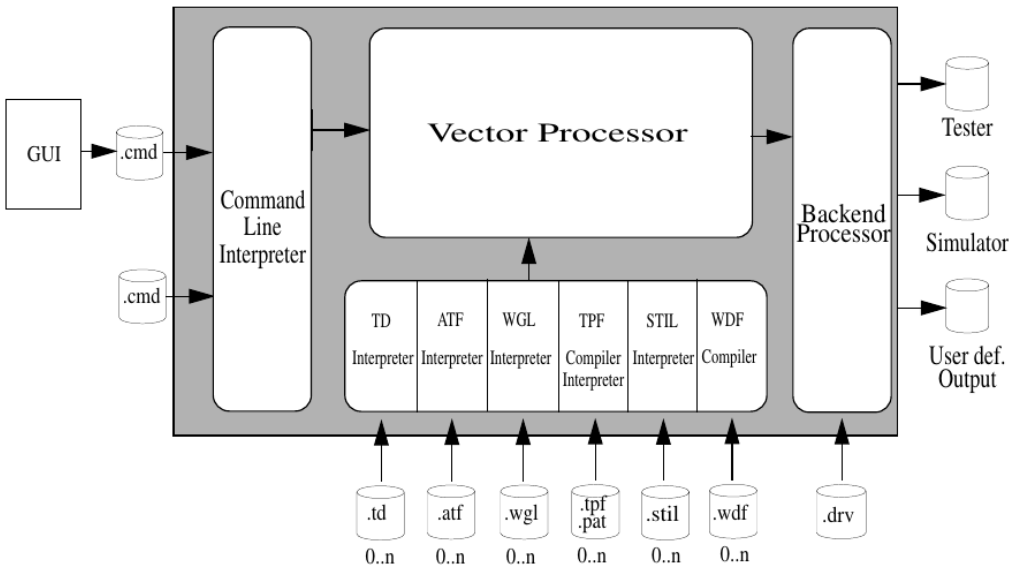


Figure 3: TASS Flow Diagram [6].

For the study purposes the most important inputs are TD (Test Data files), TDL (Test Description Language, i.e. .tpf files), ATF (Application Test Files) and STIL (Standard Test Interface Language).

The first step of the flow is to read the TD files. These are one type of data format files which contain all the necessary information regarding the JTAG network such as the chip pin definition, DFT structures, etc. Next step is to start the vector processor by calling the corresponding interpreter depending on the input pattern format. The pattern description could be done through one or several TDL, ATF or STIL, or a combination of them. TDLs are usually manually described patterns created by DFT engineers, so it is based on a human-readable format, TDL language is NXP property. On the other hand, ATF and STIL are formats typically created by design tools (e.g., NXP or Siemens/Mentor-Graphics design tools).

Each interpreter will deliver its results which are the test vectors and waveforms. With this information the vector processor develops the timing calculations to generate the wavesets (timing related definitions specifying the events on each tester cycle). Finally, TASS postprocessor is going to transform the generated data by the interpreters into an ATE pattern sequence format (such as STIL) and a corresponding testbench output file (such as a Verilog testbench). Other possible outputs such as debug purpose files (e.g., ATF) could also be generated by TASS.

3.1.2 IJTAG

Tessent IJTAG or simply IJTAG is a third-party pattern generation tool property of Siemens (former Mentor Graphics). IJTAG is one sub-product or component of the main Tessent Siemens product. Tessent is a tool and IP package offering many solutions for several DFT phases such as pattern generation, debug, design as well as manufacturing test solutions [7]. For this study it was used the version Tessent 2022.1. Through Figure 4 it is shown the IJTAG flow.

The main IJTAG inputs are:

1. Instrument Connectivity Language files (.icl)
2. Procedural Description Language pattern files (.pdl)
3. Optional user defined TCL files (.tcl)

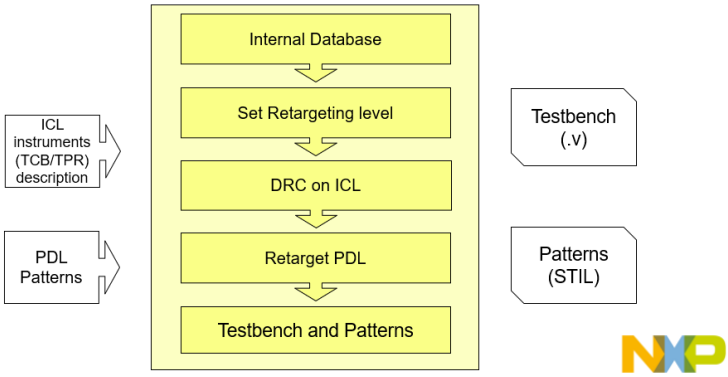


Figure 4: Tessent IJTAG Flow.

As well as TASS, IJTAG needs a file which defines the JTAG network and their components. The ICL network description is formed by one or more .icl files which describes all the DFT "instruments" (i.e., all the DFT blocks/components such as DRs, IRs, TAPs, etc.) and their interconnections [8]. The main objective of ICL for IJTAG as TD for TASS is to describe the test access view of the instruments so that the tool can understand the DFT architecture and target the high-level pattern commands to the desired instruments in order to transform them into a test sequence.

The second main input for IJTAG are the PDL patterns. The PDL files describe the instrument usage on a targeted level [8]. In other words, PDL can describe patterns through a high-level description language if there is an ICL description of the targeted instrument, its interconnections and a clear description of the hierarchy that guarantees the access to it. PDL takes the advantage of TCL language as .pdl files are considered as tcl DOfiles [8]. Because of the high level and human-readable format of PDL language, it is as easy to read/write as TDL language for TASS.

As observed in Figure 4, IJTAG flow starts with an Internal Database containing all the required ICL and PDL files. Then, the tool builds the ICL hierarchy by reading the ICL instruments definition and instantiations. During this step IJTAG is understanding the way to move or access different hierarchy levels and consequently each of the DFT components. Thanks to that, IJTAG is capable to execute and understand what to do with specific PDL commands. Third step is to check the design rules (DRC) on the ICL hierarchy and instrument levels. The fourth step (Retarget PDL) objective is to read the PDL files, create the corresponding pattern sets and finally write the PDL patterns in STIL format and with a testbench (Verilog format). IJTAG supports other industry standard output files such as SVF (Serial Vector Format) which is a standard ASCII format representing test patterns [9]. PDL Retargeting step is the main step in Tessent IJTAG flow, and it has its own flow which is going to be analyzed in section 4.3 (Non-ATPG IJTAG Pattern Generation Flow).

3.2 EOS B2 (Handled with TASS)

For this migration study it was necessary to use an already developed project as a test vehicle. The selected project for this purpose was EOS B2 which is already on silicon, so it passed successfully all the DFT flow with the internal tool TASS. EOS is a TSMC-28nm Secured NFC Controller (NFC + Secure Element). A top level SoC diagram of EOS is observed through Figure 5 (Secure Element Domain is represented as a black box because it is NXP confidential). Regarding DFT for EOS, it is important to consider some constraints, requirements and design specifications.

EOS DFT JTAG architecture is based on a star hierarchical approach. Another consideration is the control and data interfaces handling, EOS DFT was designed with a TAM (Test Access Mechanism) and a DAM (Data Access Mechanism) for this purpose. Final consideration is the pattern approach, EOS uses a Modular Pattern Approach in which each modular is part of the complete pattern sequence which represents a specific feature or configuration. All these design characteristics provide benefits but also contain some constraints that must be handled during the DFT Flow.

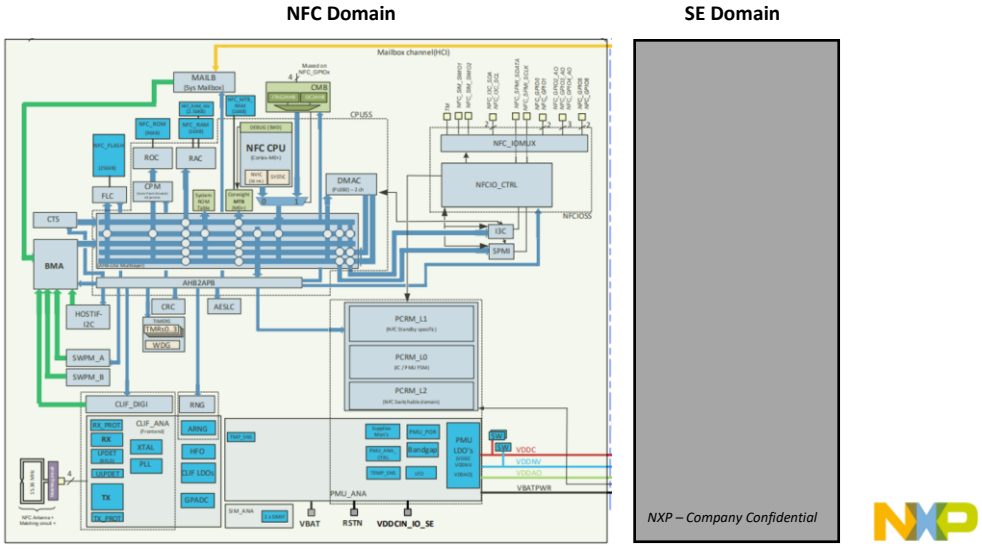


Figure 5: EOS Top SoC Block Diagram [10].

3.3 TAM/DAM Mechanisms

In order to handle the data between the IC and the tester machine ATE during the execution of DFT pattern tests, it is needed a DFT interface. This interface is formed by a mandatory control interface and an optional, but widely used in NXP, data interface. The main objective of these access components is to handle the JTAG port signals and all other port signals used for control and data during pattern tests. In case of EOS, it has a Test Access Mechanism (TAM) in charge of the control interface and a Data Access Mechanism (DAM) handling the data interface. As observed through Figure 6 in both cases for control and data, the interfaces are shared thanks to TAM and DAM. This means that the top pads are always the same and TAM/DAM are in charge of handling the signals between the pads and the different Sub-IPs.

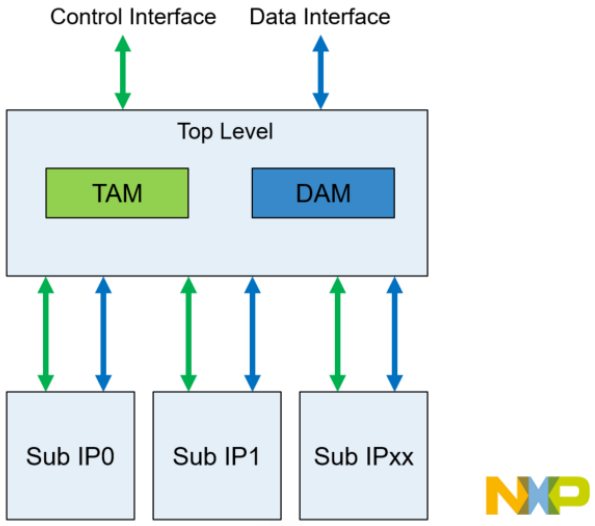


Figure 6: EOS DFT Interface Specifications.

The TAM block is in charge of handling the signals TDO and TCK between the sub-IPs and the top level as shown in Figure 7. In case of TDI, TMS and TMER (Test Mode Entry Request, DFT test reset signal), they are broadcast directly to the sub-IPs. TAM structure is formed by 3 main blocks which are TAM Gating, the TAP controller and the Top TAM TCB. TAM Gating is the block which handles the signals between the sub-IPs and the TOP, it is controlled through some selector signals coming from the TCB. The TAP contains the IR. This last one could be configured with instructions defined by the IEEE Std 1149.1, some NXP common instructions and an instruction to access to the top TAM TCB. Finally, the top TAM TCB or UP TCB is a data register that can be configured with different modes in order to access the sub-level TAPs and therefore, it controls the signals between the sub-IPs and the top level through the TAM and DAM.

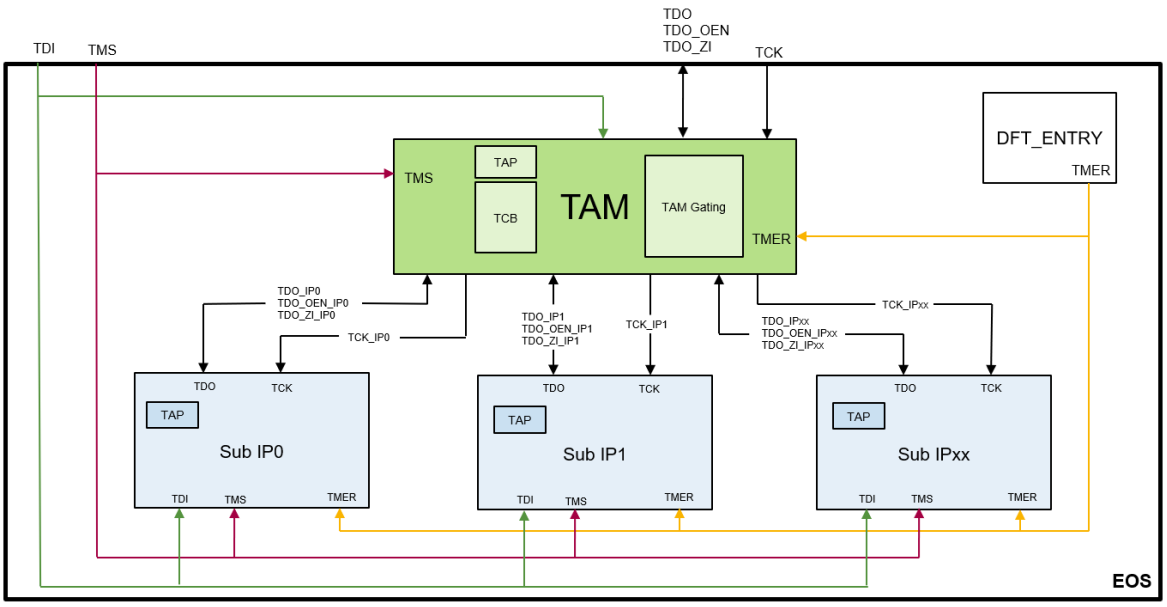


Figure 7: Shared Control Interface.

Figure 8 shows the Data Access Mechanism structure. This block receives selectors signals from the TAM TCB which are used to control the data signals between the sub-IPs and the TOP level IOs. As the IOs are bidirectional, the top cell bidirectional buffers receive 3 signals A, ZI and OEN. From the cell perspective, A is an input from the core logic (from the IC to the exterior), ZI is an output to the core logic (from the exterior to the IC) and EN is the active low enable output driver (when 0 the pad works as output, when 1 as input). A functional diagram of the MFIO1V8SF bidirectional cell is found in the Appendix Figure 2 [11].

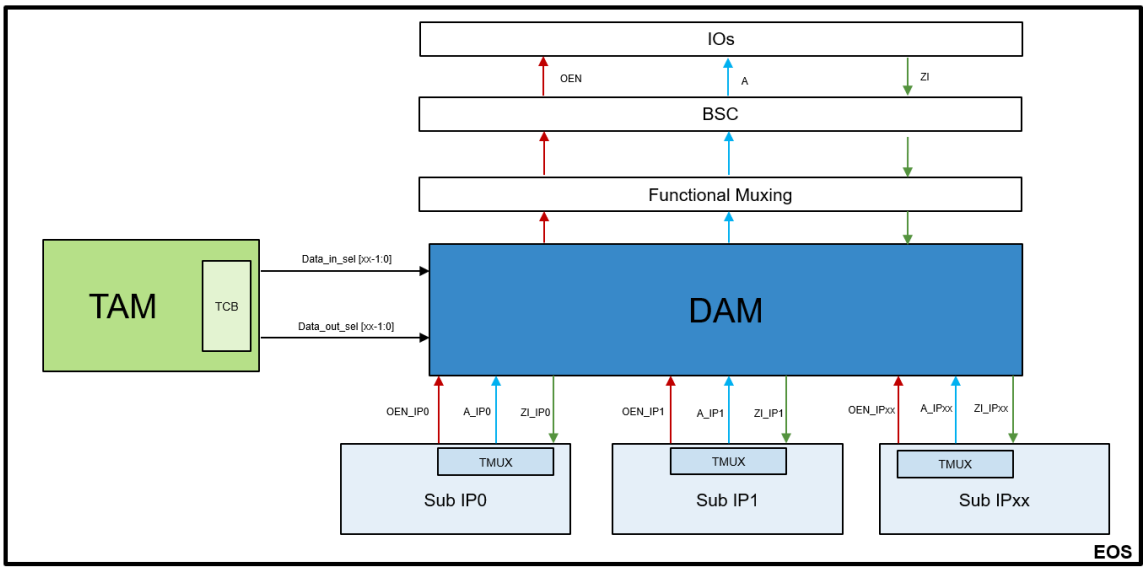


Figure 8: Shared data interface

Between the TOP IOs and the DAM there are other structures such as functional muxes which are necessary to use the IOs in different functional modes. Boundary Scan Chains (BSC) are also inserted at top level to test and control the IOs. Finally for each sub-IP level, there are TMUX blocks or test mux to control the IOs direction during the tests.

3.4 JTAG Star Architecture

The EOS DFT JTAG Star Architecture is a hierarchical design in which there is a clearly defined structure based on a multi-level approach. Different from Daisy Chain Architecture in which there is a single level with multiple TAP serially connected, in Star Architecture, there is a TOP TAM Mechanism and multiple sub-level either TAP or TAM structures. If a N-level structure has sub-levels, this sub-hierarchy must be handled by a TAM block like the TOP TAM, as observed in level 2d in Figure 9 . Thanks to TAM architecture previously analyzed, it is possible to gate TCK and TDO, so that there is the possibility to talk to a single sub-level TAP without affecting the others. On EOS design, a TAP reset spy logic was implemented to avoid resetting the TOP TAM and the TCB UP, therefore, the TAP controller of the TOP level is always “listening” what happens with the sub-level TAPs [12].

The basic flow to access a specific hierarchy level is the following. First, start the chip in test mode (Execute a Test Mode Entry Request, TMER) by a bootstrap initialization (force the JTAG ports to specific constant values to initialize the test mode). Then, it is necessary to program the TOP level TAM in order to root the desired sub-level. To do this, the TOP TAP controller IR must be set with the corresponding instruction to select the UP TCB data register. Once the UP TCB is selected, this one can be configured to access the desired sub-level. Finally, after having reached to the sub-level TAP, it is possible to access the corresponding Data Registers and DFT structures of this level.

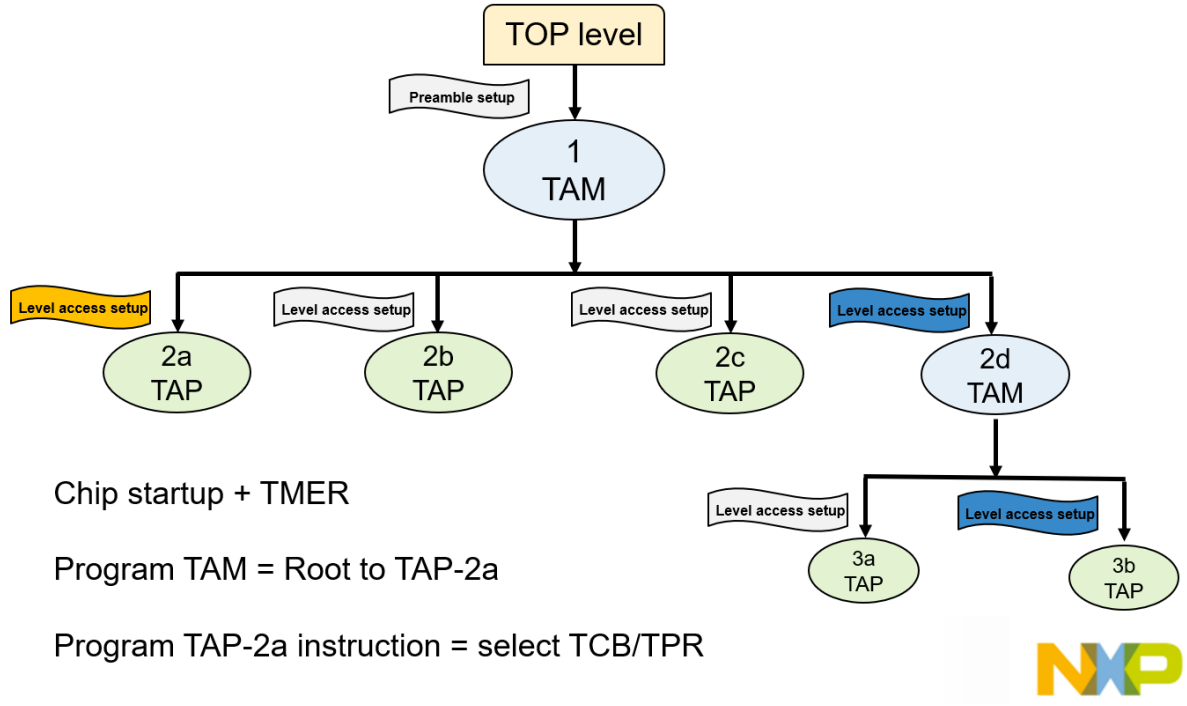


Figure 9: EOS DFT Star Hierarchy design.

3.5 Modular Pattern Approach

EOS Modular Pattern Approach is based on partial pattern descriptions representing a specific feature or configuration in which each modular is part of the complete test sequence. These features could be power, clock, hierarchical access, core patterns (e.g., BIST for SRAM tests) and the chip startup.

Thanks to this modularity there are several benefits:

- Multiple input pattern formats: Modular patterns could be TDL, STIL or ATF. Useful when the patterns are delivered by IPs from other NXP teams or external providers (TSMC).
- Regularity and file optimization: As many test sequences could be considerably similar, through modular approach it is not necessary to rewrite the complete sequence but only to modify the desired modular.
- Tester and simulation time reduction: Thanks to SDM modular (System DFT Mode) it is possible to change from one DFT test mode to another concatenating the right modulars without rebooting the device (TMER modular).

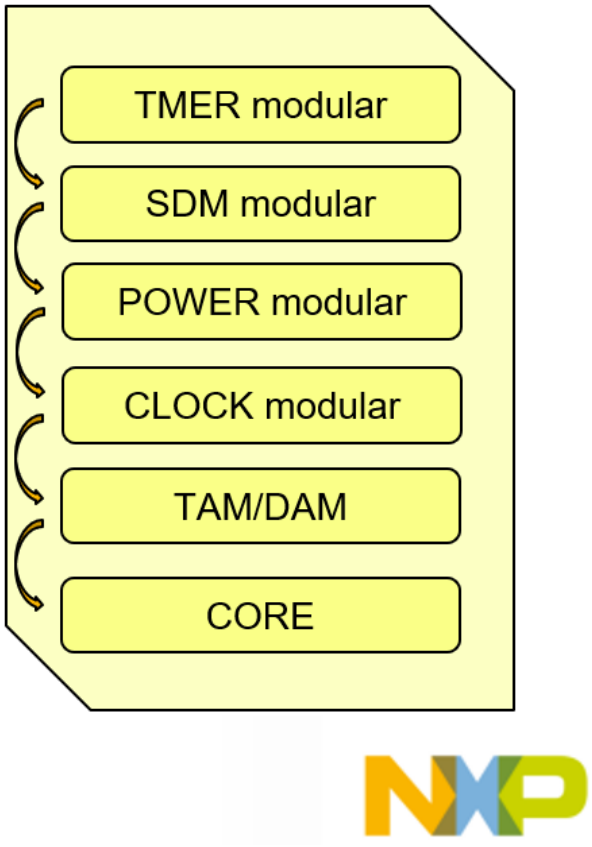


Figure 10: Modular Pattern Approach.

3.6 EOS NXP DFT Flow (TASS-based)

Through Figure 11 it is shown the complete EOS NXP DFT Flow with the current pattern generation tool TASS. The main 3 stages are DFT Generation, Verification and Pattern Generation. This flow describes each of the stages starting from the DFT blocks specifications until the generation of pattern sequences and testbenches. The main objective of stage 1, DFT Generation, is the RTL generation of the different DFT blocks such as TCBs, TPRs, TAPs and TMUXs; to then integrate them to the design and develop the synthesis of the Gate Level Netlist.

During DFT Verification, the objective is to validate the DFT blocks generated on the previous flow, verify their interconnections with the JTAG network, that means ensure that there is a hierarchical definition to access and configure them. Finally, during Verification it is created the DFT Database with the required files for the pattern generation tool (Test data files for TASS). Last step on the flow is the Pattern and Testbench generation in which starting from the DFT database and a high-level pattern description (e.g., TDL) it is generated the pattern sequences and the corresponding testbenches.

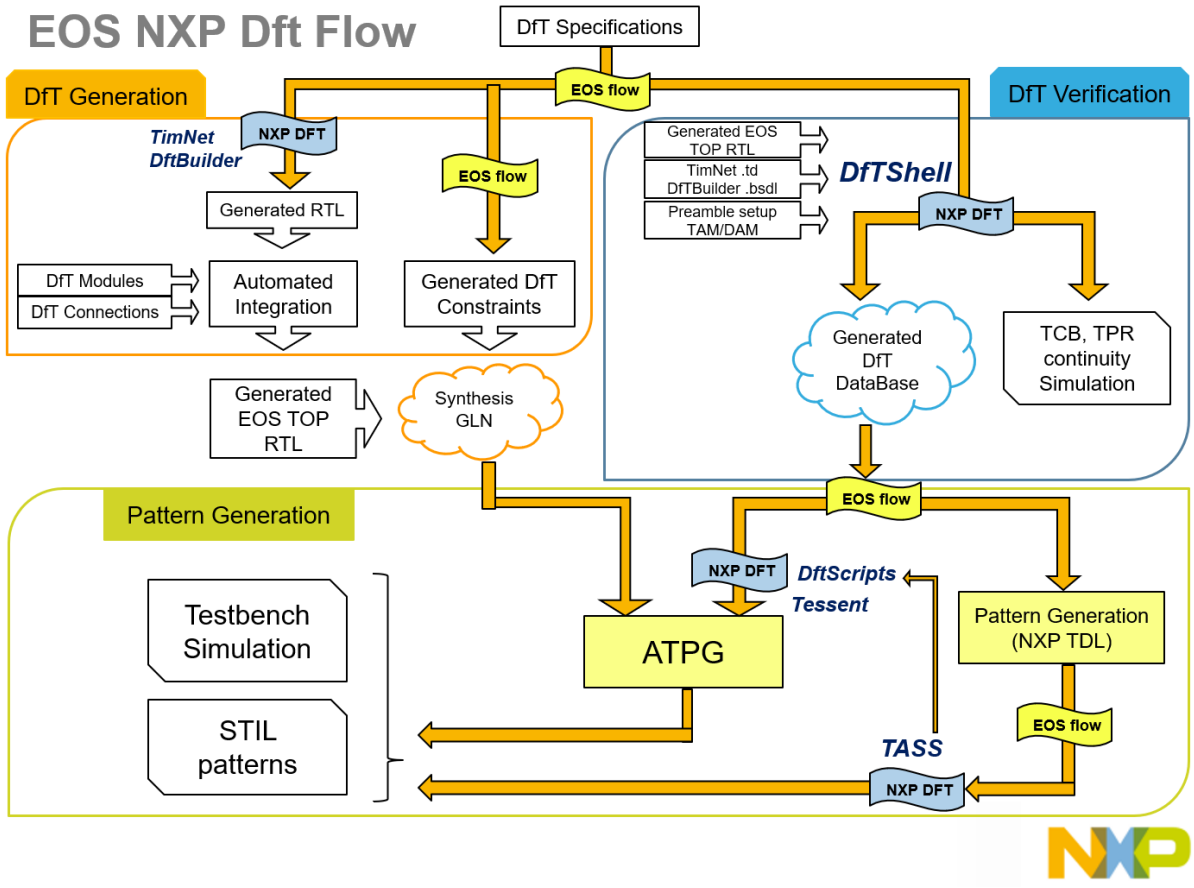


Figure 11: EOS NXP DFT Flow with TASS tool.

3.6.1 DFT RTL Generation

DFT RTL Generation Flow starts from the DFT specifications. This corresponds to the description of each DFT block, their signals and functionality. This information is specified in multiple excel spreadsheets. Through some excel macros it is possible to translate these specifications into specific file formats (.nif and .csv) understandable by the DFT tools TimNet and DFTBuilder. These tools are NXP DFT internal tools and each of them is in charge of generating specific DFT structures. In case of TimNet, it generates the RTL corresponding to the Data Registers: TCB and TPR. On the other hand, DFTBuilder generates the RTL for the TAP controllers and the TMUXs.

Other outputs of this flow are the DFT views through the TD files. They contain the information of each TCB and TPR, the register signals and the configurable modes. There is also generated a BSDL representation containing the JTAG port definition and the information regarding the interconnection between the DFT blocks.

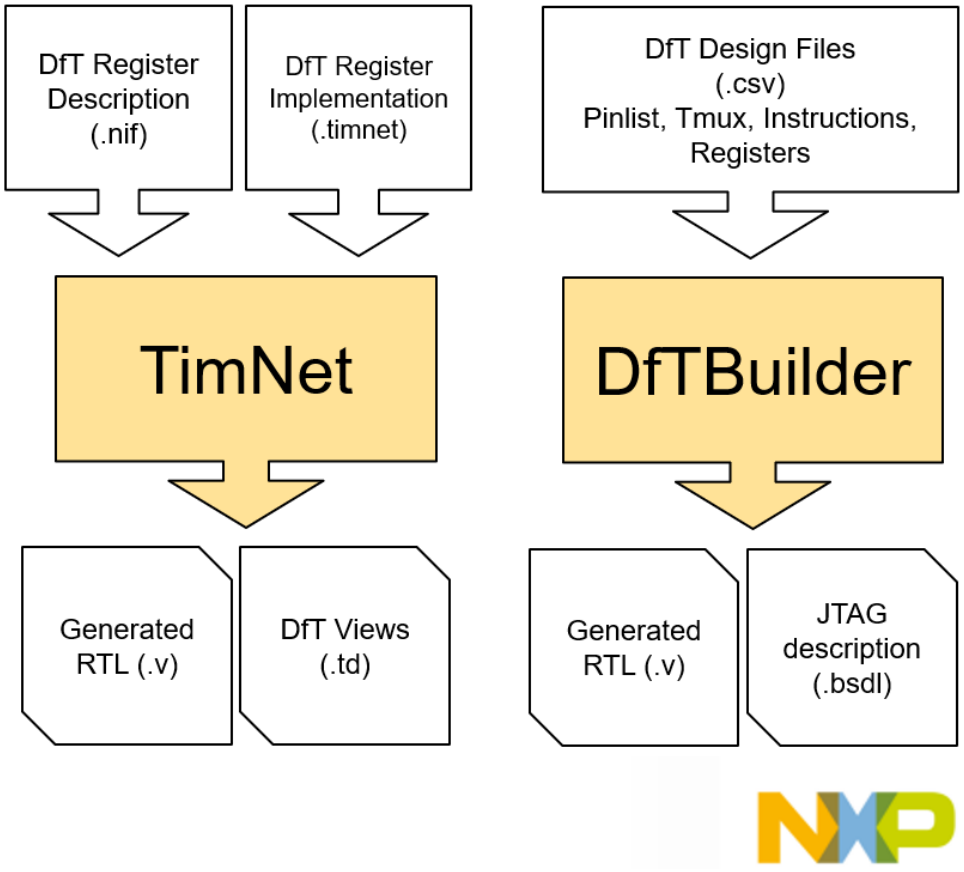


Figure 12: DFT RTL Generation with TimNet & DFTBuilder.

3.6.2 DFT Verification

DFT Verification Flow inputs are the generated outputs from DFTBuilder and TimNet. The internal NXP DFT tool used during this flow is DFTShell. This tool starts with the DFT views of the TD files and executes a verification process of the TCB and TPR chains. During this process it checks that the design information regarding instances and ports are defined correctly, as well as the test protocols of the Data Registers [13]. Then it develops a JTAG network validation in which it is check that there is the possibility to access to each of the hierarchy levels through a preamble setup (i.e., through the TAM/DAM Mechanisms).

After verification is done, DFTShell generates a database with the deliverable files for Pattern Generation. These files are the TD chains which are a simplified and concatenated version of the TD views of the previews flow. It is also generated some configuration TCL files.

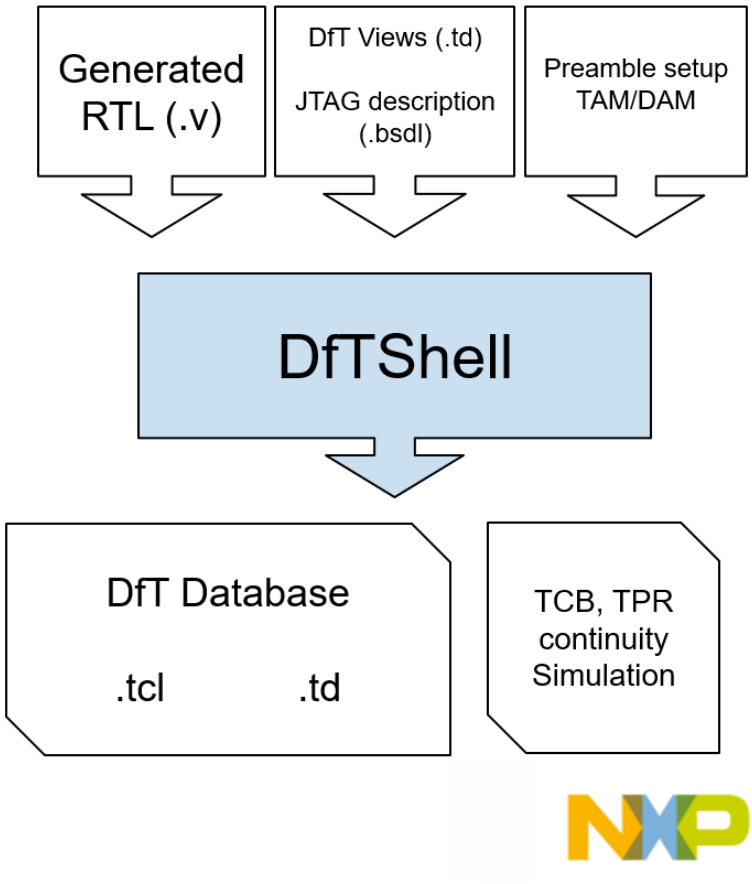


Figure 13: DFT Verification with DFT Shell.

3.6.3 Non-ATPG DFT Pattern and Testbench Generation Flow

The main objective of Non-ATPG patterns is to generate test sequences using the pattern generation tool TASS, starting from a high-level description of the patterns. As explained in Section 3.1.1 TASS, TASS targets the input pattern description to the corresponding DFT structure in the JTAG network. Patterns are mainly described in TDL NXP DFT language. But there are also some particular cases for third party IPs or blocks designed by other teams. In these situations, it could be a specific DFT network design for those structures and therefore, unique patterns to test them which could be represented as a modular according to EOS modular approach. For this reason, TASS is capable to support other input patterns formats such as STIL or ATF corresponding to external source patterns. This could be the case of Secure Element team patterns (ATF) or third party TSMC IPs patterns (STIL).

The final objective is to obtain a test equipment pattern description (STIL) of each modular pattern (part of the whole test sequence) and an equivalent testbench for the complete sequence.

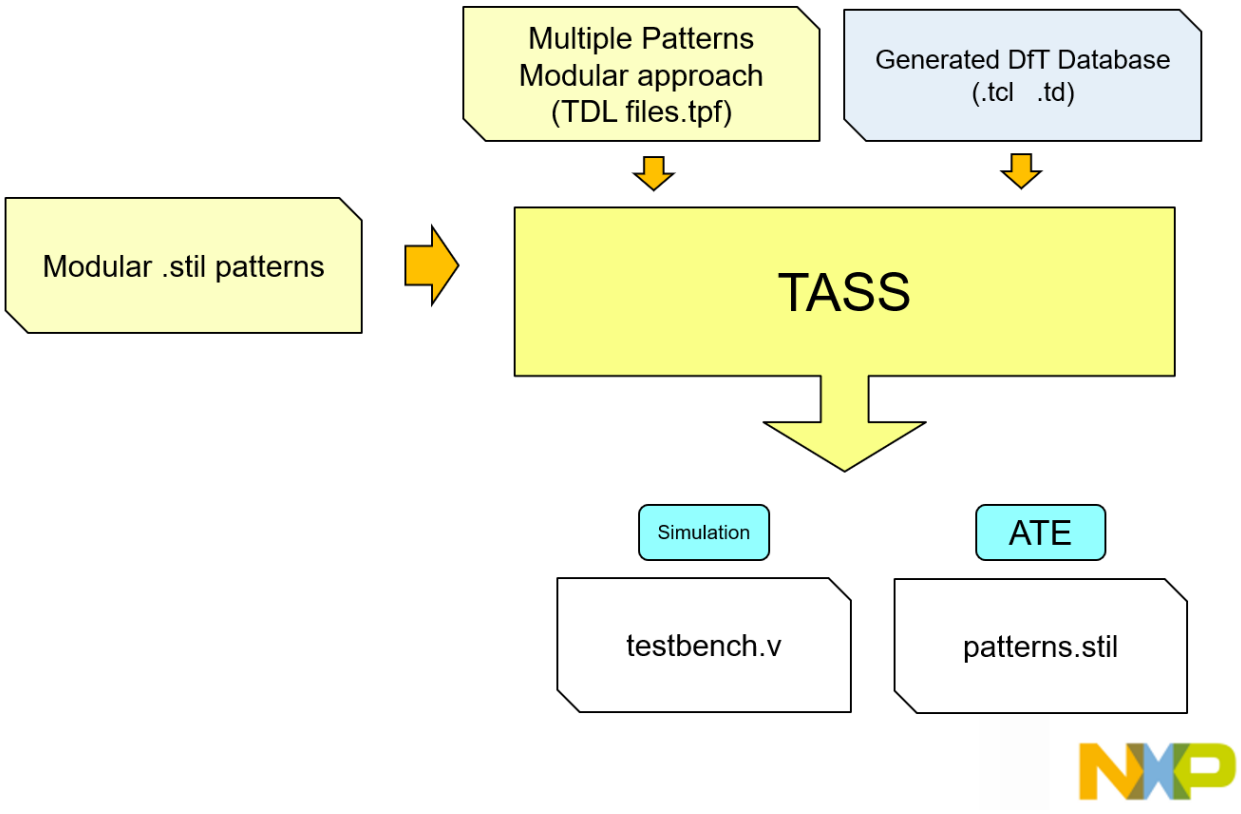


Figure 14: DFT non-ATPG Pattern & Testbench generation.

3.6.4 ATPG DFT Pattern and Testbench Generation Flow

Automatic Test Pattern Generation (ATPG) is a DFT automation methodology widely use on SoC design. The main advantage of ATPG is that it generates a plenty amount of pattern sequences to detect or cover possible manufacturing defects of the IC. Due to the complexity of ICs, it is a difficult task to have a complete fault coverage with manual written patterns, then ATPG becomes fundamental.

NXP ATPG DFT Flow uses two tools: TASS and the third-party tool Tessent Shell. TASS objective is to provide Tessent a Test Setup. A Test Setup is a test procedure that contains force, expect and pulse event statements in order to set ports to specific values, read and compare values from them and initialize clocks [14]. Generally, this procedure is used at the beginning of the test sequence. In this flow, the Test Setup is in charge of all the startup sequence, hierarchy access and necessary configurations for ATPG tests. In EOS case with TASS, it generates an ATF sequence corresponding to the Test Setup patterns. Using some internal scripts, this file is transformed into Tessent Shell test procedure format (.testproc).

Thanks to a DOfile (TCL based file) it is possible to evoke Tessent Shell and sequentially run all the required commands to call the Test Procedure and launch the ATPG test generation. As expected, the result is a STIL pattern description with a testbench.

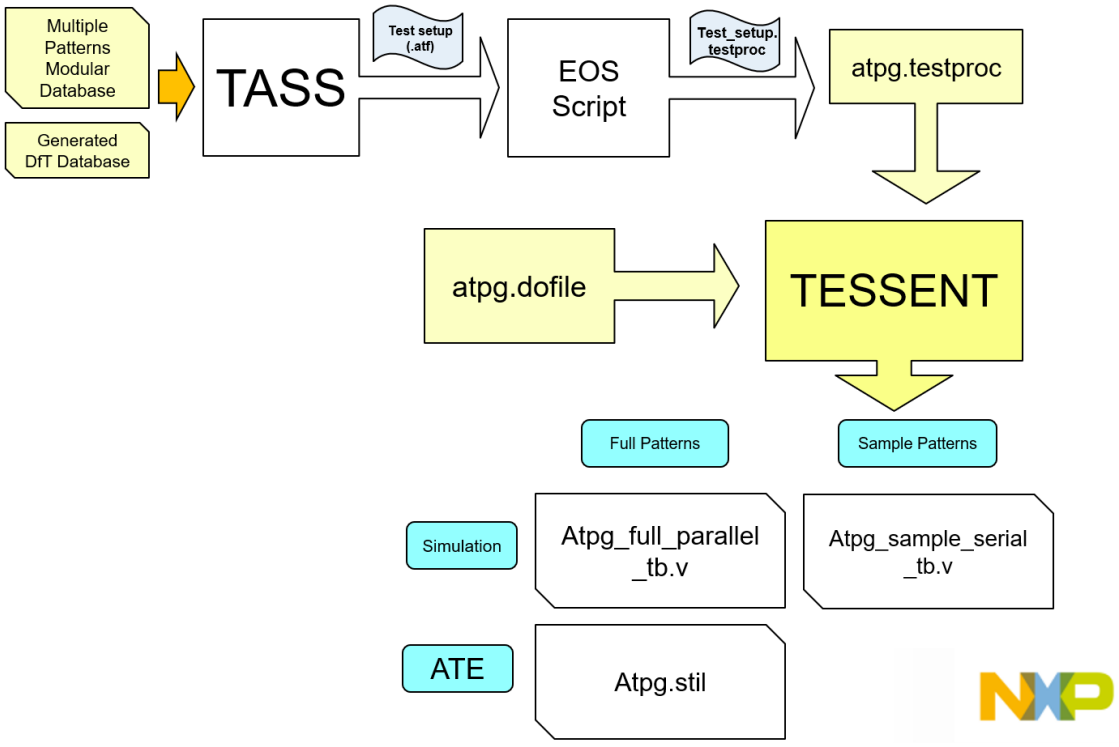


Figure 15: DFT ATPG Pattern & Testbench generation

3.7 General EOS DFT Design Requirements Summary

Through Table 1 it is shown a summarized version of the features that are expected with the new IJTAG flow considering the current TASS situation.

Table 1: EOS DFT Features.

Topic	Feature	TASS
JTAG Network Description	The pattern generation tool must be able to understand the multiple tap network and generate the pattern sequence to handle the hierarchy access before accessing the DFT target element (e.g., TCB or TPR).	Limited hierarchical understanding but possibility to manually write a TDL cycle-based function to handle the hierarchy access.
Modular approach	Test sequence generation based on a supported modular approach.	Modular approach completely supported through independent modular TDL pattern files.
Multiple input pattern sources	Modular patterns could be TDL (manually written) or delivered by external teams (ATF or STIL).	TASS support multiple sources: TDL, ATF and STIL on the same test sequence generation.
Non-ATPG patterns	Complete control to force JTAG ports and multiple timing definition.	TASS has absolute control over JTAG ports, being able to force, expect and pulse signals manually as well as generating multiple timing definition in the same TDL file.
ATPG patterns	Accurate simulation, multiple timing definition.	TASS Test Setup does not include neither the bootstrap initialization nor multiple timing definition (Feasible but not implemented).
Annotation on pattern files	Detailed annotations on STIL pattern output files	TASS generates per cycle and highly detailed annotations on STIL outputs.

4 IJTAG Analysis Results

4.1 New EOS NXP DFT Flow (IJTAG-based)

One difference between TASS and IJTAG tools is the JTAG network description. As explained previously, TASS uses TPR and TCB TD chain files representation while IJTAG needs an instrument based ICL description. Due to this new input, the NXP DFT Verification flow must be modified, specifically the DFT database generation. However, the most important modifications are developed in the Pattern and Testbench Generation Flow. The TDL-based modular patterns database was changed by PDL. It was necessary to develop several TCL and Shell scripts in order to automate non-ATPG pattern generation flow. On ATPG case, the NXP base scripts were adapted for the new flow.

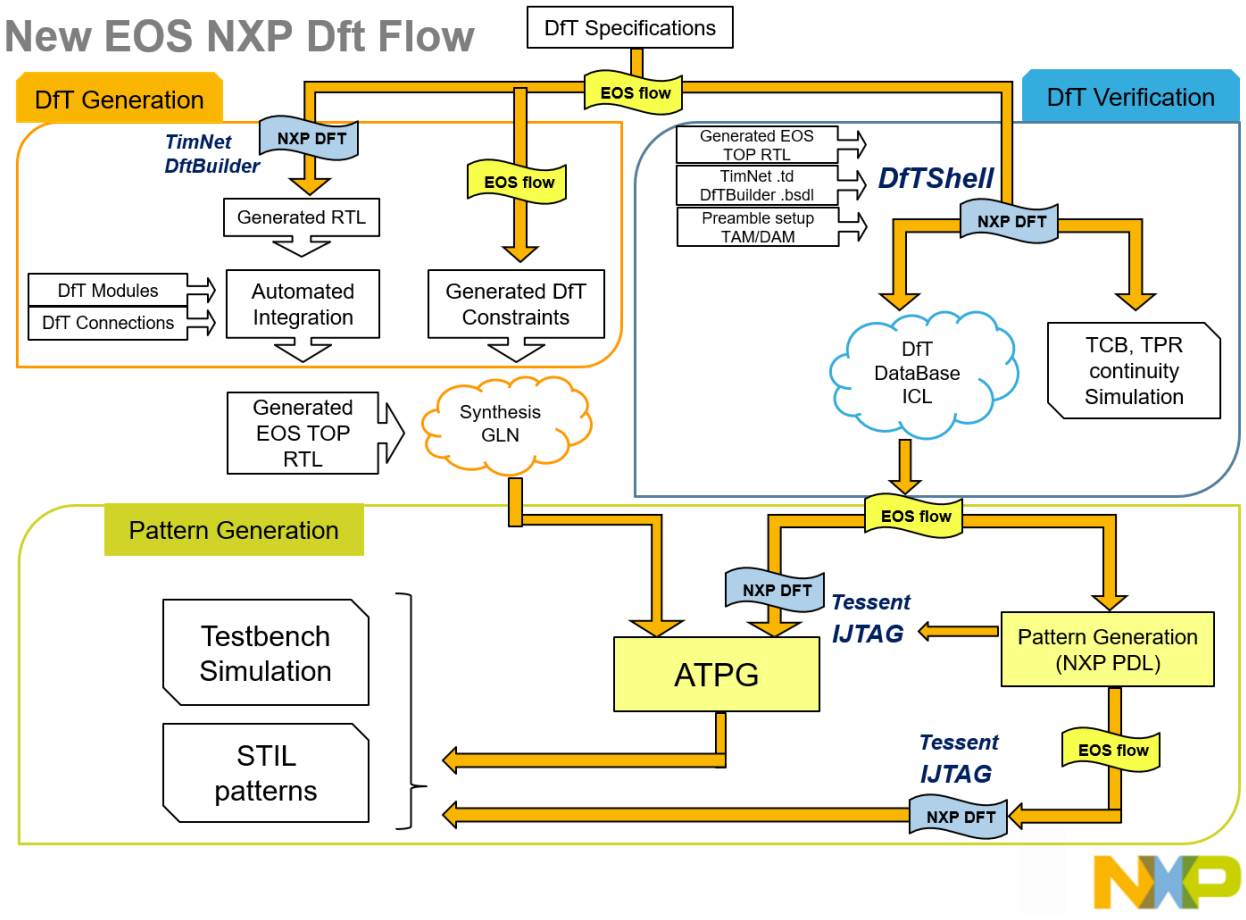


Figure 16: EOS NXP DFT Flow with new IJTAG tool.

4.2 ICL Generation

The ICL generation was an additional step implemented in the EOS DFT Verification Flow. The original TASS flow finishes with the TCB and TPR chains generation. On IJTAG case, it is necessary to create the TOP ICL Network representation through the chains using DFTShell NXP internal tool. The chains will provide the complete information regarding the register signals and the modes. Additionally, it is necessary to provide DFTShell other TD files such as the pin and jtag description, they have information regarding the device input/output pins and the IR OPCODES (Instructions). With all this information, DFTShell writes the JTAG network representation in ICL format to an output file. All these processes were developed through Shell/TCL scripts to evoke DFTShell tool and generate the files automatically.

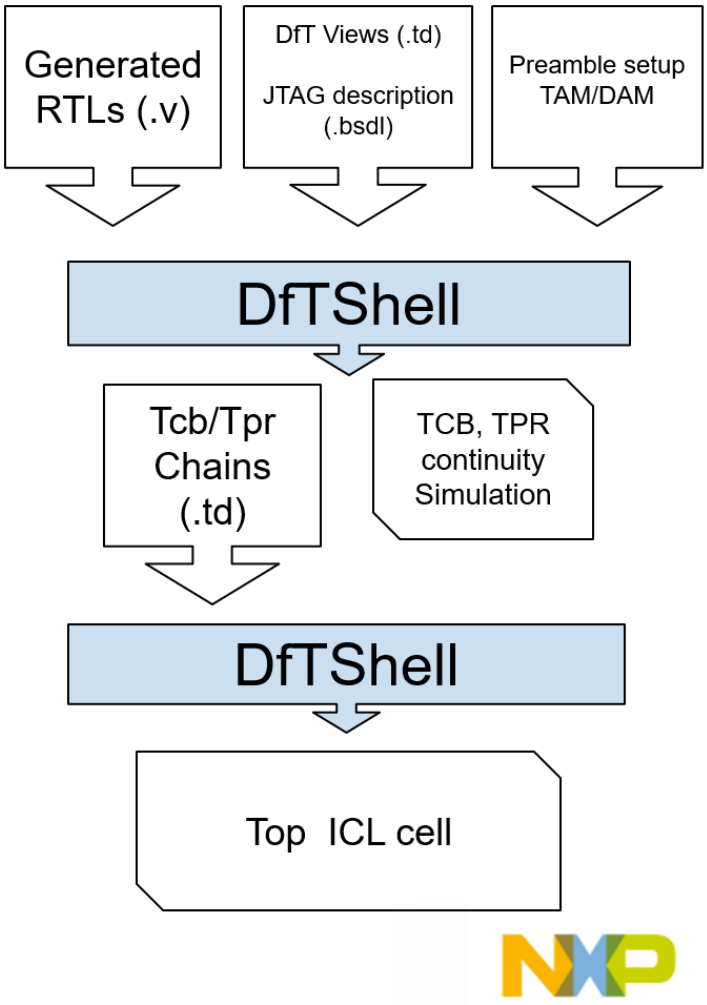


Figure 17: DFT New Validation Flow for ICL generation.

It is possible to analyze the ICL network through Tessent Visualizer which is the Tessent visualization and debug graphic environment. Through Figure 18, it is shown the ICL schematic from the network generated by DFTShell. As observed, the result differs from the original DFT design for EOS since the multiple-tap architecture is not represented. DFTShell

only generates a single tap ICL which in this case corresponds to the NFC sub-IP tap. Since there is not the TOP TAM tap, TCK (GPIO1) is not gated before NFC tap so it is broadcast directly from the TOP to the NFC sub-IP same as the other JTAG signals (TDO or GPIO0, TDI or GPIO2 and TMS or GPIO5).

Since there is not a hierarchical representation, when generating patterns with this ICL, IJTAG will identify NFC sub-IP as the unique level. Therefore, generated patterns targeting one TCB/TPR will contain the sequence only for NFC tap. For this reason, if it is used the ICL from DFTShell without any post-process, it is necessary to describe manually the patterns for the hierarchy access, in this case from TOP TAM tap to NFC tap. In IJTAG case this is done through cycle based sub-procedures. These are one kind of Tessent procedures which used on basic patterns allow three things: force primary inputs, measure primary outputs and pulse the capture clock [14]. Which means it is possible to manipulate the JTAG ports to write a cycle based (per cycle defined) sequence to describe the access from one level to another. Sub-procedures can only be called within another procedure, therefore, cannot be used in PDL. It is necessary to use Test Setup procedure to call them.

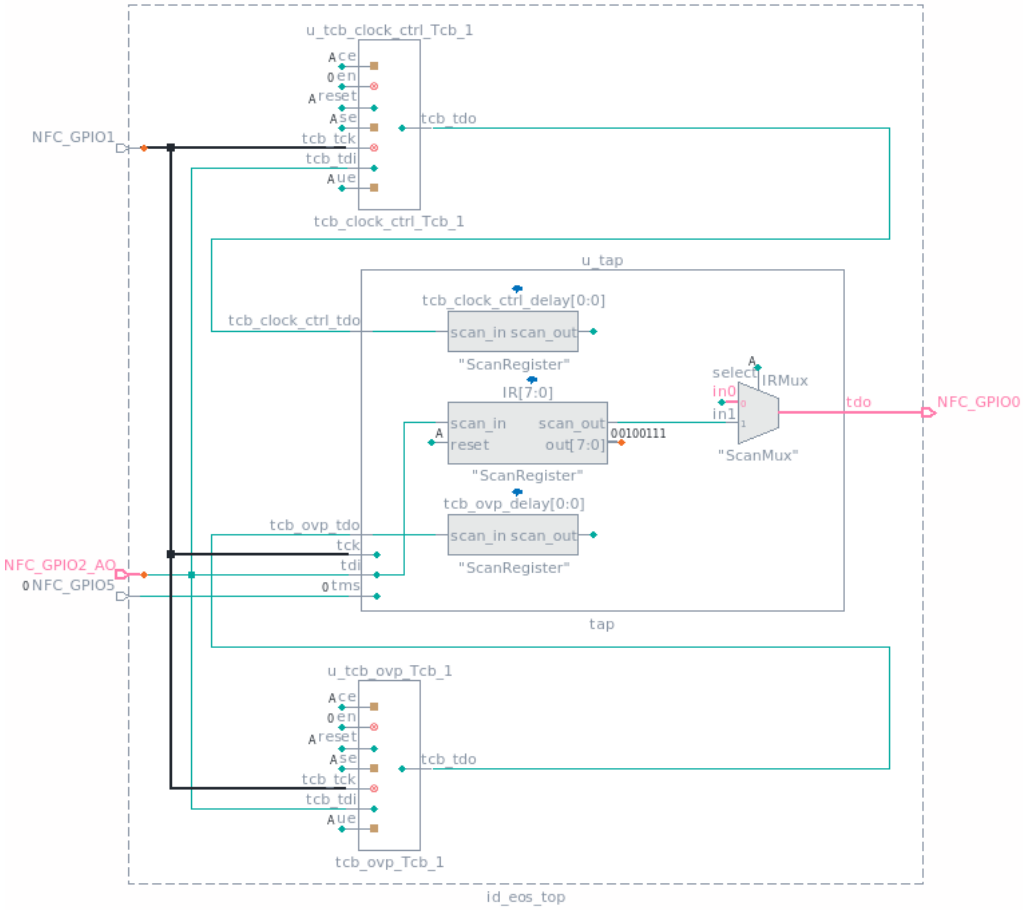


Figure 18: ICL Schematic from DFTShell on Tessent Visualizer.

4.3 Non-ATPG IJTAG Pattern Generation Flow

Through Figure 19 it is observed the new Pattern Generation Flow using IJTAG for non-ATPG patterns. The main IJTAG input is the ICL description, with which IJTAG will know the instrument (DFT blocks) description, how to access and target the PDL commands in order to create patterns. The ICL description has information only about DFT structures and JTAG ports, if the patterns include information regarding on-JTAG ports it is necessary to read the TOP Verilog interface which contains information about the whole device ports. Analog ports must be removed from the Verilog interface as are unsupported by the tool.

The most common pattern description input format for IJTAG is PDL, but it is also possible to use SVF and test procedure patterns (e.g., Test Setup). The Siemens recommended flow suggests using a Test Setup procedure at the beginning of the test sequence before PDL patterns (left yellow flow from Figure 19) [8]. The objective of this Test Setup is to initialize the test and describe the bootstrap start. The main benefit of this procedure is that it could force and pulse all the ports (included JTAG ports) which is not possible to be done with IJTAG PDL patterns (JTAG port manipulation is not allowed). Nevertheless, Test Setup can only be used at the beginning of the sequence, therefore, with this flow it is not possible to manipulate the JTAG ports during the test sequence. This flow is developed completely in patterns -ijtag context. The context specifies certain features and the current usage of Tessent Shell. Patterns -ijtag context provide functionality related to IJTAG Pattern generation, ICL extraction and -ijtag switch enables PDL commands on pattern sets. Once Test Setup is read, it is possible to open multiple pattern sets, within patterns sets it is possible to read and populate the PDL pattern files and define timing features through timeplates (analog to wavesets for TASS TDL files).

Another possible flow is through only Test Setup based patterns (right blue flow from Figure 19). As mentioned, through Test Setup it is possible to force JTAG ports which is the main advantage of this flow. However, in order to be able to use PDL commands inside the Test Setup Procedure, it is necessary to change the context to patterns -scan. The scan switch on pattern context is usually used on ATPG patters as it enables Tessent TestKompress (Siemens DFT product to implement compression or EDT logic) [15]. On non-ATPG flow it is useful to enable PDL usage and JTAG ports manipulation both inside Test Setup. As seen in Figure 19, before reading the test setup, it is necessary to source the procedural files. It contains all the sub-procedures which manipulates directly the JTAG ports with force or pulse commands, these sub-procedures are called inside the Test Setup.

Both flows finish with the Write Patterns step, which is generating a STIL equivalent pattern file either for each pattern set or for each Test Setup. It is also generated a complete sequence Testbench in Verilog format. During both flows it is usually necessary to change the system mode. This feature enables the usage of different commands by the tool during the flow. Setup mode is specially used for initial settings such as read the input files (ICL, Verilog, PDL,

Test Setup) while Analysis mode is used for PDL retargeting during pattern set creation, Procedural Files analysis and Pattern/Testbench generation.

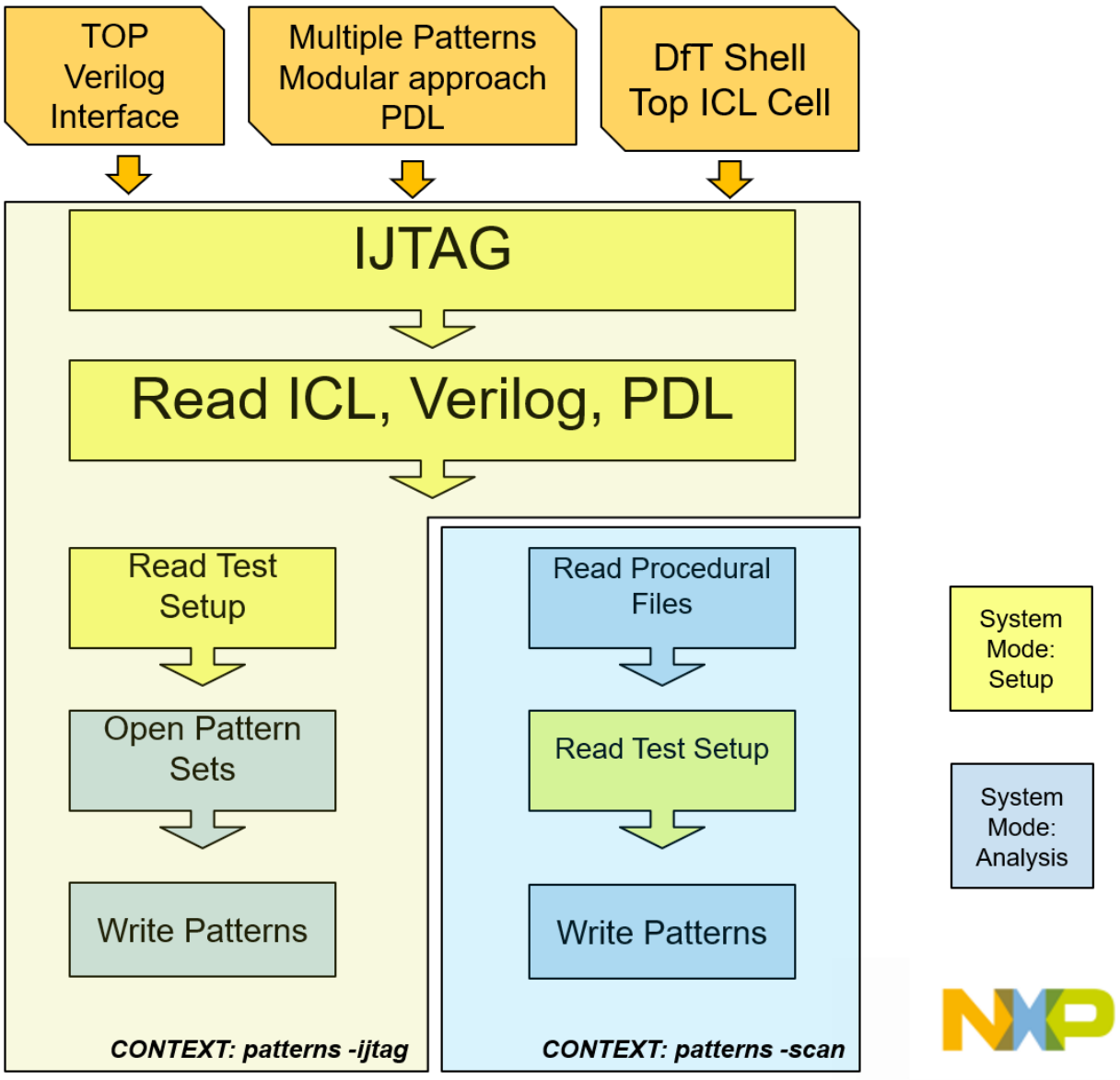


Figure 19: PDL Retargeting Flow for Pattern Generation.

4.3.1 Non-ATPG IJTAG Pattern Simulation Results

Hierarchy Access, IR and DRs management.

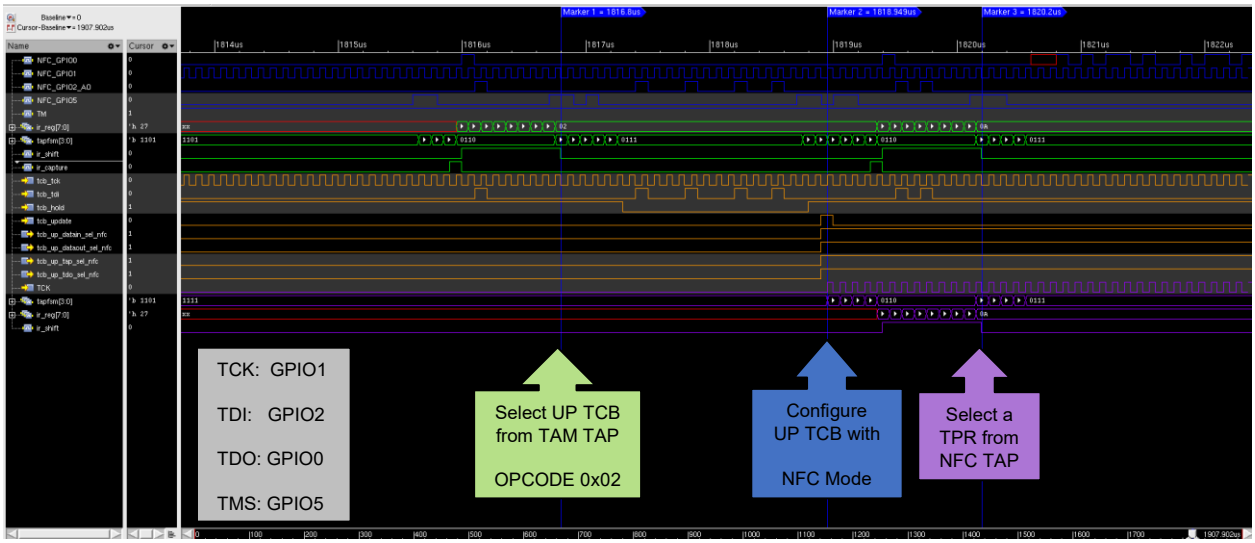


Figure 20: Non-ATPG IJTAG Testbench with Manual Hierarchy Access.

Due to multiple-tap hierarchical DFT design of EOS, in order to access and write/read a data register (TCB/TPR) it is required a particular pattern sequence which is described in Figure 20. The mentioned example shows the sequence to access the NFC level, control the NFC tap and access a TPR to read it. It is shown in blue the JTAG signals TDO, TCK, TDI TMS (GPIO0, GPIO1, GPIO2_AO, GPIO5 respectively) and the TM (Test Mode) signal. In green it is shown the TOP DAM tap signals from the IR and the FSM state. In orange it is represented the UP TCB signals and in purple the NFC tap signals corresponding to the IR and the FSM state.

The first part of the sequence is to access the IR of the TOP DAM tap and write the OPCODE 0x02 which corresponds to the address of the UP TCB. Since the ICL generated by DFTShell only represents the NFC tap structure, the access must be done manually through cycle based sub-procedures as IJTAG has not enough information to do it automatically. As observed, the sequence starts with the TOP TAM tap in state “1101” that according to the FSM diagram of Figure 2 corresponds to Run Test/Idle state to modify the IR it is necessary to access it and shift the data inside. It is done through TMS signal which when it is a logic 1 during two rising edges of TCK it is possible to reach Select IR Scan state. Once IR it selected next state is Capture IR, through the waveforms it is observed in green the ir_capture signal which indicates when the FSM is in this state. Finally, it is selected UP TCB by shifting inside the IR its OPCODE, during Shift IR state the signal ir_shift is set to 1. In order to check if the OPCODE is correct and the UP TCB is selected it could be analyzed the ir_reg value which as observed is 0x02.

Next part of the hierarchy access is to configure the UP TCB. This is a DR controlled by the TOP TAM tap which can be programed with 5 modes. Each of the modes allows to select a specific sub-tap such as NFC, SE, NV (non-volatile or flash), CMB (Control Master

Block) or to return the control to the TOP TAM tap. Through the right TMS sequence it is reached the Shift DR state to shift-in data to UP TCB, during the shift process the tcb_hold orange signal changes to 0. It is interesting to observe that TCK and TDI signals for the TCB are driven directly from the top as observed in the ICL schematic. Once shift is over, tcb_hold returns to 1 and tcb_update signal is asserted during one TCK cycle indicating that the TCB output is valid and visible. At this time, it is possible to see that TCB signals related to NFC are set to 1 (datain_sel_nfc, dataout_sel_nfc, tap_sel_nfc, tdo_sel_nfc).

After UP TCB is configured with NFC mode, TCK (purple TCK signal) is transmitted to NFC tap (as mentioned sub-IPs TCK is gated). Until this part of the pattern the sequence was defined manually, but once NFC tap is selected, it is possible to use PDL commands to write and read data from the DRs. Since this point the approach is very similar as with TOP TAM tap, it means, set the IR OPCODE to 0x0A which corresponds to PCRM_STATUS TPR and read expected values from it. The whole process is developed by IJTAG after using an “iRead” command inside PDL. To use this command, it is necessary to provide the information about the instrument path (hierarchy defined in the ICL network description) and the expected values for one or more bits of the DR. When it is needed to write, it is used “iWrite”. Same as previously, it is required to give the path and define the value to be written.

4.4 New Test Sequence Approaches

4.4.1 Approach 1: Instrument Level PDL pattern description

One of the main targets in pattern description is to maintain the Modular Approach used with TASS tool. IJTAG offers different ways to write test sequences. First of them and highly recommended by Siemens IJTAG manual [8] is through an Instrument Level description. As shown through Figure 21, this approach starts with a PDL description targeting each DFT Data Register (TCB or TPR). Inside this PDL files it is used procedures (iProc) with PDL commands to read, write and, in general, to manipulate the instrument. Second level is the pattern set level which is the equivalent to Modular Approach. Each pattern set represents a unique modular pattern and it is described in a different PDL file. Inside this PDL pattern set, the instrument level PDL commands are called in order to generate the complete pattern description for the modular feature. It could be called as many instrument PDL procedures as needed.

Finally, the Test Setup and the modular pattern sets are concatenated to generate the complete test sequence. The main drawback of this flow is the limitation to force JTAG port only on Test Setup. Despite Modular Approach is kept, it is necessary an extra level (Instrument Level) to describe patterns. Last point to consider is the limitation to only one timeplate per pattern set. Therefore, if a modular requires more than one timeplate, Modular Approach is affected since the modular must be separated into multiple pattern sets.

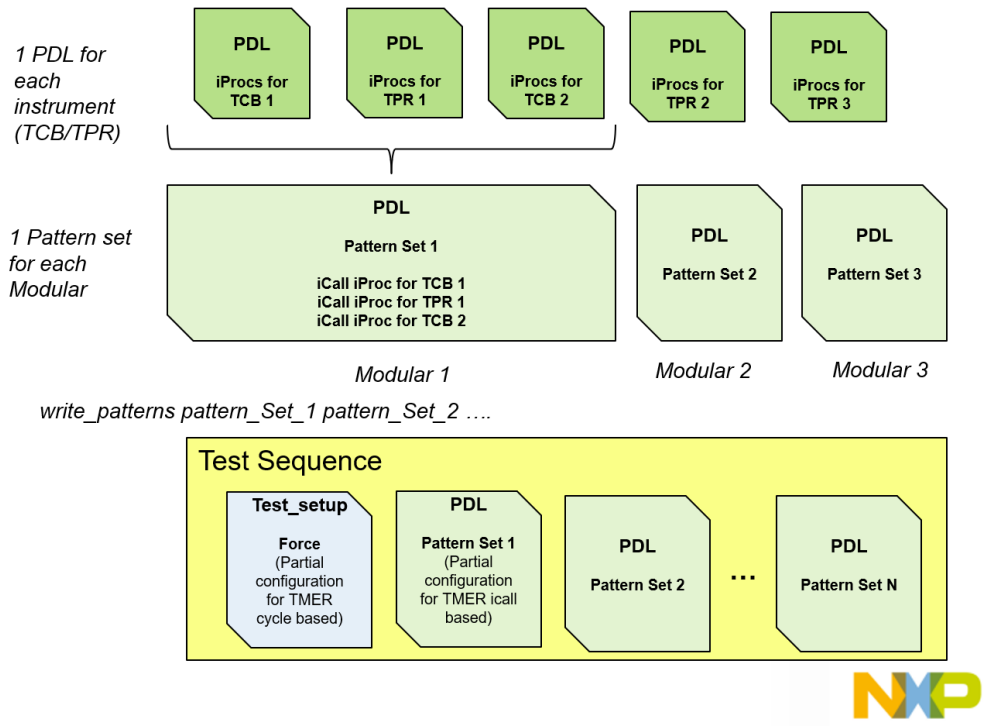


Figure 21: Instrument Level PDL pattern description approach.

4.4.2 Approach 2: TOP Level PDL pattern description

A second PDL-based pattern description approach could be developed through a TOP level targeting. In this approach instead of using procedures to target instruments (iProc), the TOP level of the design is targeted through a particular procedure (iTopProc). The main improvement of this approach is that it keeps modular approach as each iTopProc PDL corresponds to one modular, without needing an extra instrument level description. As observed through Figure 22, all the pattern description is done on the first TOP PDL level, therefore, pattern sets are only simple wrappers, each of them calls one TOP procedure. Consequently, it is not mandatory to use a PDL for pattern set representation, a pattern set could be opened by calling directly the TOP procedure (iCall command).

Test sequences generation is the same as previews approach. It is the possibility to concatenate a Test Setup at the beginning of the sequence. Nevertheless, there is still the limitation to force JTAG ports and have multiple timeplates in the same PDL. The main advantage of this new method compared with the last one is that this is closer to modular approach design as it targets the TOP level (same as TDL patterns).

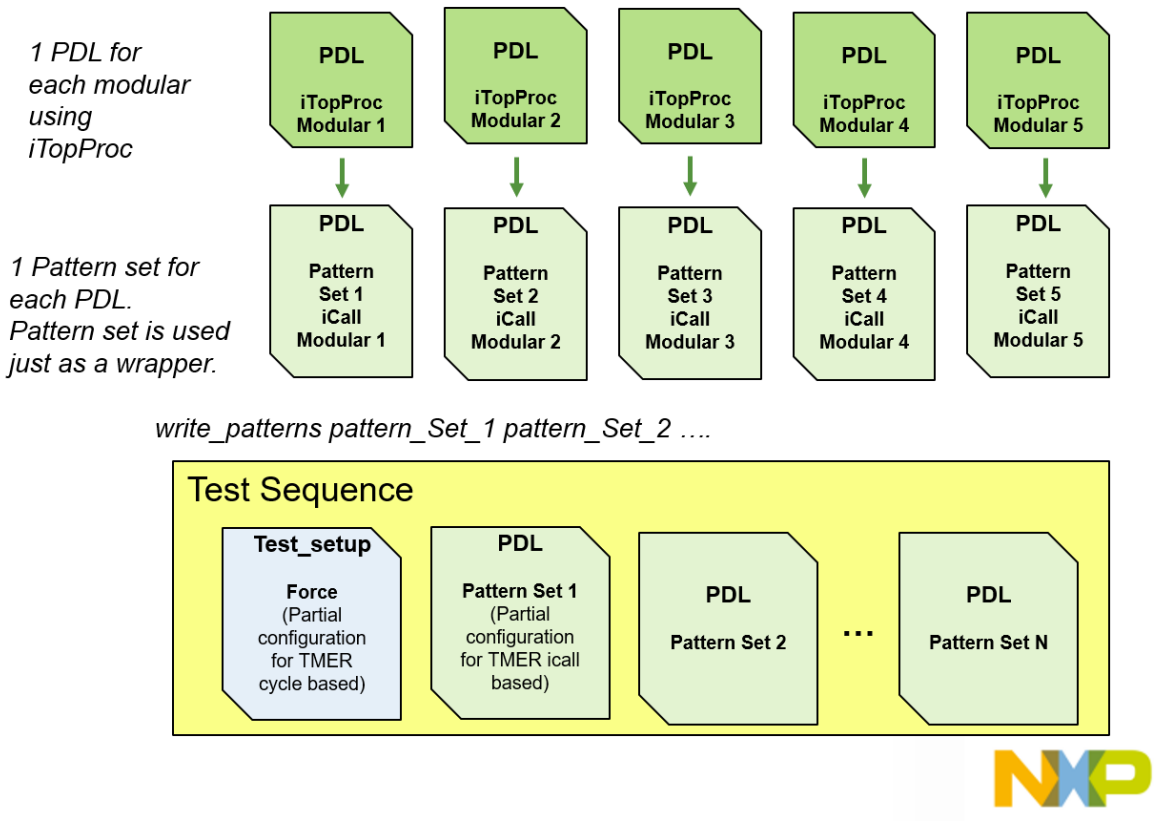


Figure 22: TOP Level PDL pattern description approach.

4.4.3 Approach 3: Test Setup pattern description

Last approach is Test Setup based as observed in Figure 23. Different from the two previous options in which Test Setup procedure was used at the beginning of the sequence only for the chip startup, some initial patterns and the hierarchy access, in this approach Test Setup contains the complete sequence. Mainly, only cycle based sub-procedures are used in Test Setup; in order to use PDL, it is necessary to follow a different IJTAG flow described through Figure 19 in section 4.3 Non-ATPG IJTAG Pattern Generation Flow. The main advantage of this new approach is the possibility to force JTAG pins at any part of the sequence through cycle based sub-procedures.

However, there are some limitations regarding Modular Approach and multiple timeplates patterns. First, it is necessary to consider that there are some patterns in which it is necessary to force JTAG ports for specific modulars. In that case, previous flow supported this feature through TDL (TASS flow), now it is necessary to use sub-procedures since PDL does not support this feature. That means, it is necessary to split those patterns into multiple PDL plus sub-procedures as observed in third modular of Figure 23. As a result, Modular Approach is affected.

Second issue is multiple timeplates usage inside the same PDL or sub-procedure. On PDL case, it was determined that for each timeplate it is necessary a different PDL, splitting a modular in as many PDLs as timeplates used. In case the same PDL is recalled within Test Setup but with a different timeplate definition it can be done thanks to -timeplate switch on iCall command available only on Test Setup environment and patterns -scan context. Nevertheless, with sub-procedures this option is not available, so even if it is needed to reuse the same sub-procedure with a different timeplate, a new sub-procedure is required to be written. Same as PDL, in sub-procedures it is not possible to define more than one timeplate.

Last consideration concerns to the ICL network state. Since from IJTAG solver perspective it is not possible to understand what happened to the network during a sub-procedure when required it is necessary to use iState command. iState is used to manipulate the internal IJTAG network state, so that it is possible to manually describe what was done during a sub-procedure only if the network was modified.

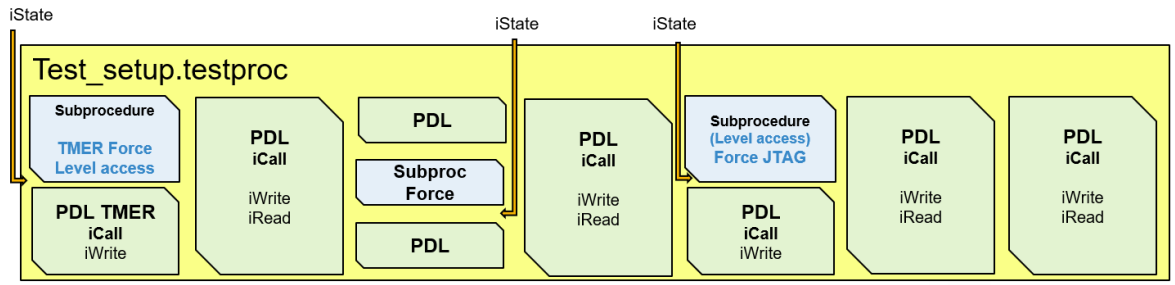


Figure 23: TOP Level PDL pattern description approach.

4.5 New ATPG Flow

The first main objective of the new ATPG flow using IJTAG PDLs patterns is to simplify the flow steps and therefore to reduce the bug's risk during pattern generation. The second main objective is to guarantee ATPG testbench simulation is accurate and matches real output patterns used for ATE.

Analyzing previous TASS flow from Figure 15 it was determined that in order to generate Test Setup procedure for Tessent Shell it is required an intermediate step to transform ATF output from TASS into a .testproc file understandable by Tessent Shell. To do so it was used some internal scripts. This extra step could be the source of some bugs, so it is always better to avoid it. In the new ATPG flow with IJTAG from Figure 24 it is used directly Test Setup in .testproc format as a Tessent input.

ATPG Test Setup is written following Approach 3 (Test Setup based sequence), so that it is possible to use both PDL and sub-procedures. In the new flow, Test Setup procedure is copied into the main atpg.testproc file which contains other necessary procedures for ATPG. In order to use PDLs in Test Setup, it is necessary to read the TOP level ICL network.

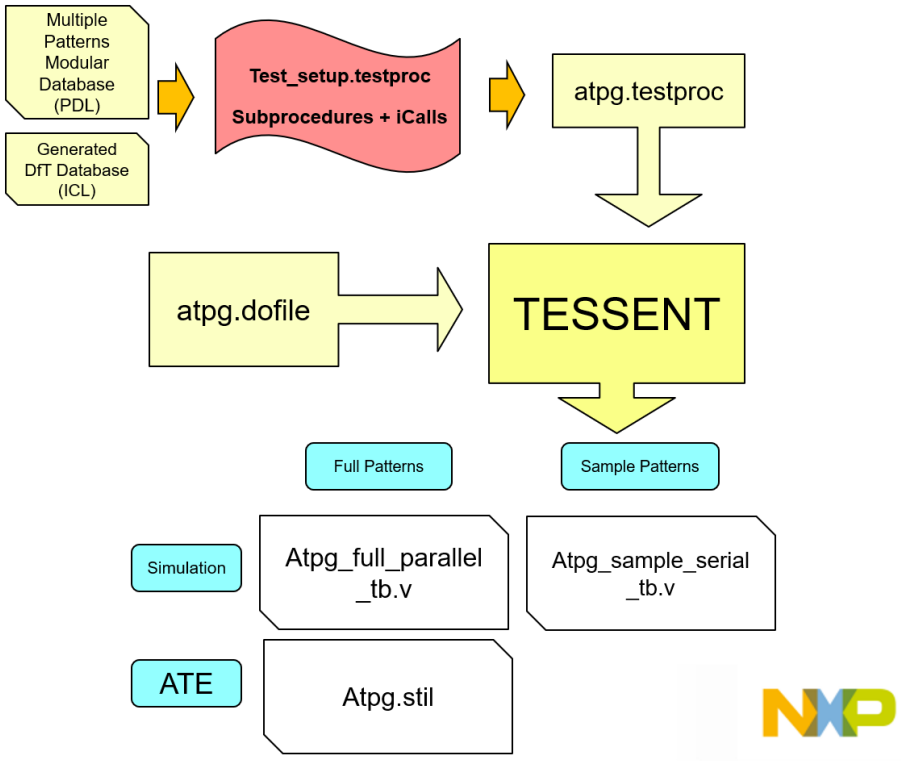


Figure 24: New ATPG Flow with IJTAG PDL Test Setup.

4.5.1 ATPG PDL-based Test Setup

Through Figure 25 it is observed the differences between Test Setup for ATPG patterns from TASS and IJTAG. First advantage from new IJTAG Test Setup is the possibility to include the bootstrap startup at the beginning of the sequence which corresponds to Modular 1 or TMER modular. This means that the startup will also be included in the testbench, generating more accurate results between simulation and ATE patterns. In TASS case it was not done (but it is possible), so it was only used a sub-procedure and some adaptations in the DOfile script in order to force the entry without the bootstrap sequence in the testbench.

Second IJTAG Test Setup advantage is the possibility to simulate multiple timeplates. Initial TASS Test Setup was limited to only one timeplate for the complete sequence (represented as Timeplate 0 in Figure 25) and TCK pulsed each two cycles to match the timing. However, this does not match the ATE patterns reality in which each modular could have one or more different timeplate. With IJTAG Test Setup it is possible to use as many timeplates as desired with the already discussed limitation of only one timeplate per sub-procedure or PDL. Since some modulars have different timeplates or they are forcing JTAG ports, they must be separated in several files (ATPG modular case).

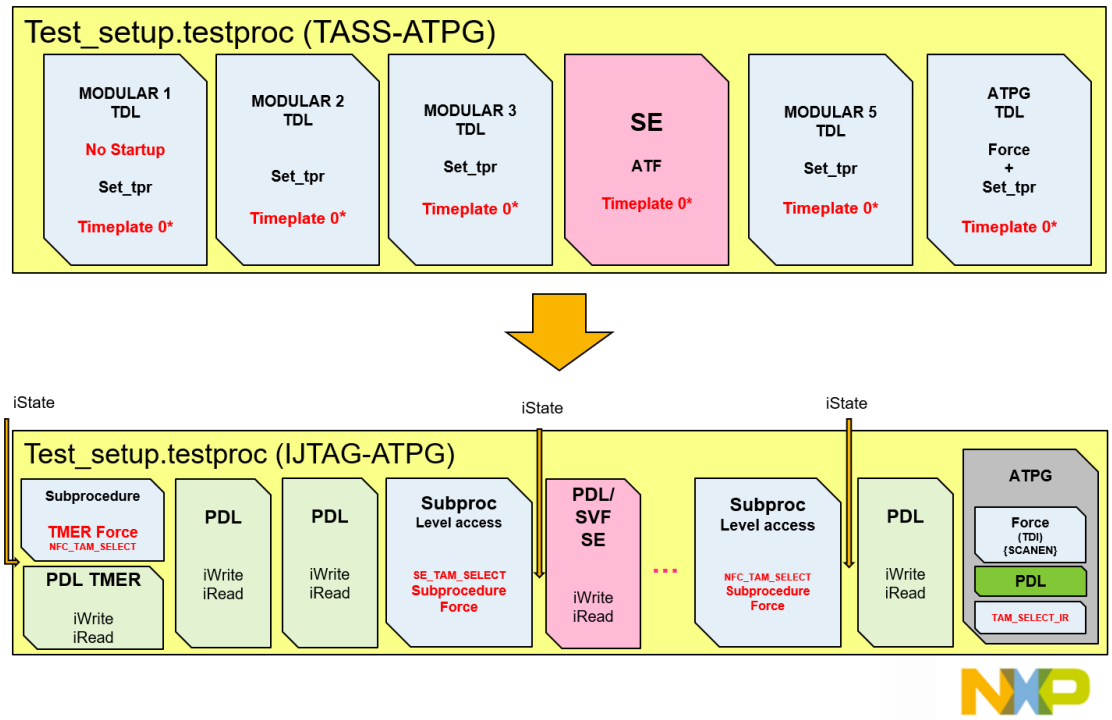


Figure 25: ATPG Test Setup modification using IJTAG PDL patterns.

Since some ATPG modulars are only in charge of the hierarchy access, they are pure sub-procedures on IJTAG Test Setup (represented in blue as Level Access, Figure 25). Due to the DFTShell generated ICL does not match the hierarchy design, IJTAG is not capable to understand the multi TAP architecture (same as TASS) and that is the reason why hierarchy access can only be developed through a manually described cycle-based sequence both on

TASS and IJTAG. In case multi Tap architecture is defined, hierarchy access could also be developed through an iTopProc using PDL-based commands inside a PDL file.

It has been mentioned that it is required to use external modular patterns from other NXP teams such as the case of Secure Element patterns (SE). In that case, patterns format is neither TDL nor PDL necessarily but could be ATF or STIL. As ATF is an internal NXP format, the only option for IJTAG is to receive STIL patterns as it is a standard format. Unfortunately, STIL is not supported in the current IJTAG version (2022.1) both on pattern -ijtag and pattern -scan contexts.

4.5.2 ATPG Tests and Simulations

EOS ATPG tests are designed based on three main islands which are: GP – ADC (General Purpose Analog-to-Digital Converter) represented as G, the boost island or B, and the Secure Element or S. Both the G and B islands are digital islands inside the analog module. Depending on if the scan chains are enabled or not, the islands are represented as 1 (enabled) or 0 (disabled). Therefore, as an example, the first test is G0B0S0 meaning that the scan chains from all the islands are disabled, and the test target is only the TOP level scan chains. The second test is G1B1S0, in which the scan chains from both digital islands inside the analog module are enabled. The last case is G1B1S1, in which all the islands scan chains are enabled.

ATPG pattern tests could be classified depending on the test model, the test type, the compression configuration, and the testbench type, this last one applies only for simulation. Table 2 shows the ATPG test classification.

Table 2: ATPG Test Classification

Fault Model	Test Type	Compression	Testbench
Transition	Chain test (shift only)	ON (EDT ON)	Serial (First N patterns)
Stuck-At	Full ATPG test (shift + capture)	OFF (EDT Bypass)	Parallel (Full patters, no shift)

A. Test Model

The fault model is the way to describe and simulate manufacturing defects within the IC. The most traditional test is call Stuck-At, which targets the gate level of the design (logic gates, basic logic structures, etc.) and test their interconnections. This test is efficient to test several production defects of the IC and quantify the fault coverage. The Stuck-At test models the IC logic stuck-at 1 and 0 based on the logic operation of the gates. As an example, from Figure 26, a stuck-at 0 fault could be analyzed through an OR gate easily, the target is to

demonstrate that the gate is not stuck-at 0 so controlling only one of it inputs with a logic 1 is enough to observe the fault. Summarizing, this fault model tests if a logic structure, due to manufacturing defects, is stuck-at 1 or 0 irrespective of any value changes on the inputs.

However, Stuck-At model does not include timing considerations such as a possible gate delay larger than the expected. Due to manufacturing defects, timing characteristics changes from one IC to another and in some cases, this could be severe enough to generate faults in the desired behavior. This gate delay issues causes a node value to change but not in the time it should, that is why this type of faults occurs during transition giving the name to the Transition Model.

A Transition Model could be understood as a Stuck-At test within a time window. This means that a Transition Model test can cover some Stuck-At faults. The process to obtain the equivalent Stuck-At faults covered by Transition is called Fault Grading. This process rates the testability with the percentage of possible Stuck-At faults to detect thanks to Transition Model compared with the total fabrication defects. At the end of all the ATPG tests, the objective is to have the maximum test coverage (100%), however this is only an ideal case. In real cases test coverage is around 98%.

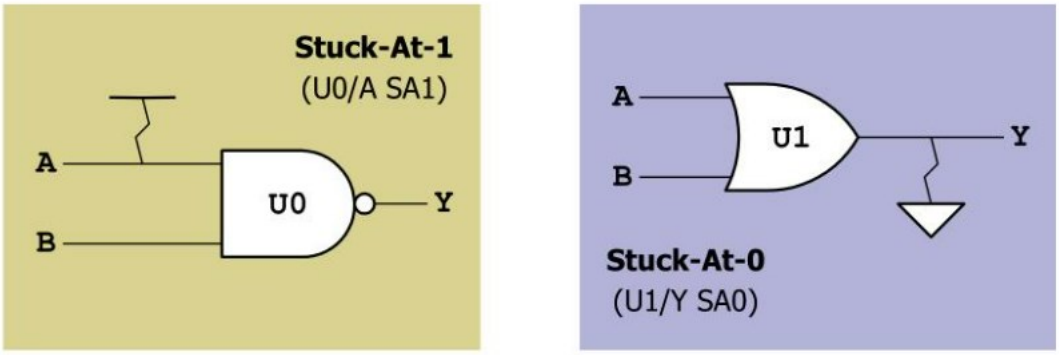


Figure 26: Single Stuck-At Fault example.

B. Test Type

For each of the two Fault Models used on EOS ATPG, it is possible to execute two types of tests: Chain test and the ATPG Full test. During the design flow (between Synthesis and Backend) it is developed the Scan Chain Insertion or DFT Insertion process (Appendix Figure 3). During this step the standard Flip-Flops of the design are converted into Scan Flip-Flops (SFF). The main feature of SFF is that they are capable to work on two operation modes: normal and test mode.

In order to test a node and detect a stuck-at fault, the node must be controllable and observable. In other words, it is necessary to have full access to the input and output of the node which is related to combinational logic. To solve this issue, SFFs are connected serially generating Scan Chains during the Synthesis Scan Chain Insertion process, a simple

implementation example is found in Figure 27. Using scan chains, it is possible to connect the Scan flip-flop input either to the combinational logic output (Normal Mode and Test Mode) or to the Scan-in pin (only Test Mode) [16]. During normal mode, the scan enable signal is 0 and the SFF can only perform a "Capture" operation which means take the input data and apply it on the next clock cycle. Normal mode is equivalent to the original behavior of the Flip-Flop before the Scan Chain Insertion. During test mode, the scan enable signal starts at 1, this allows the SFF to perform "Shift" operation, which means, the SFF shift-in the data (test pattern control stimuli). Then, SFF load the response (Capture) of the combinational logic by the SFF functional input during one cycle (for stuck-at faults). Finally, the SFF shift-out the observed response through scan-out.

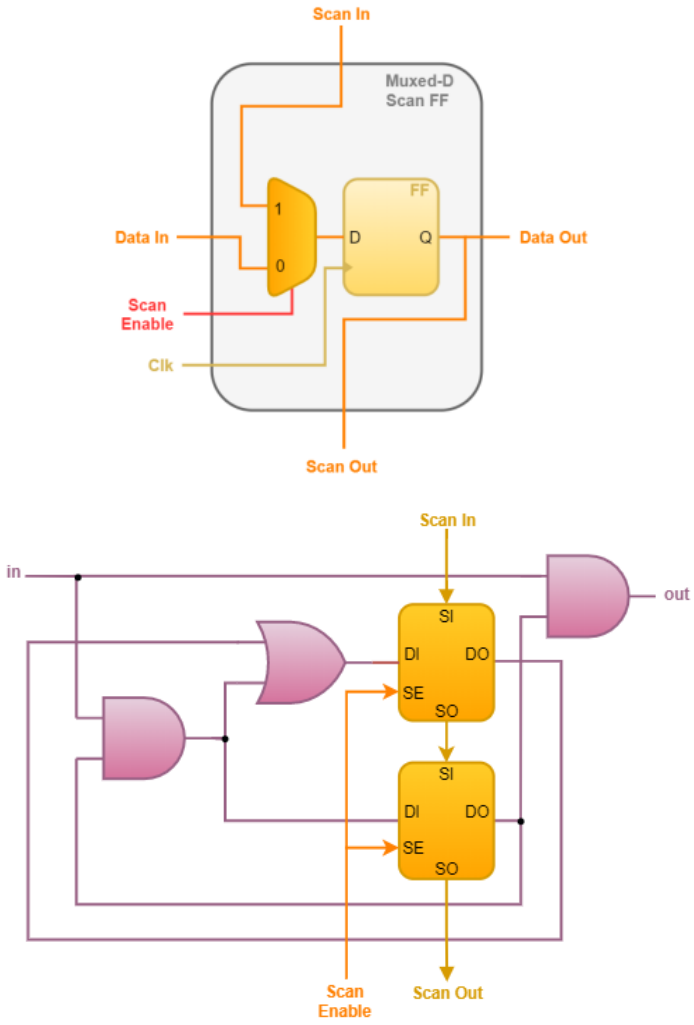


Figure 27: Scan Flip-Flop (left) and Scan Chain (right) [16].

The first test to verify is the Chain or Scan Chain test, its main objective is to evaluate that there are issues that prevent the use of scan chains, therefore, to control and observe the SFFs. During Scan Chain test, the Scan Chains work on scan mode but they do not execute the "Capture" step, only the "Shift" steps. Therefore, Scan Chain test is a shift-only test as mentioned in Table 2. The basis of Chain test is to introduce a pattern into the different chains and compare it with the chain output. As the SFFs are never capturing the combinational

response, the output must be the same as the input pattern since the chain is only working as a shift register. However, if it is needed to evaluate a stuck-at fault which is related to combinational logic, it is necessary to execute the full test mode flow, this is called Full ATPG test which is formed by the two “Shift” steps, plus “Capture” in between.

Finally, it is important to remark that during a shift-out process it could also be executed a shift-in as observed through the waveforms of Figure 28.

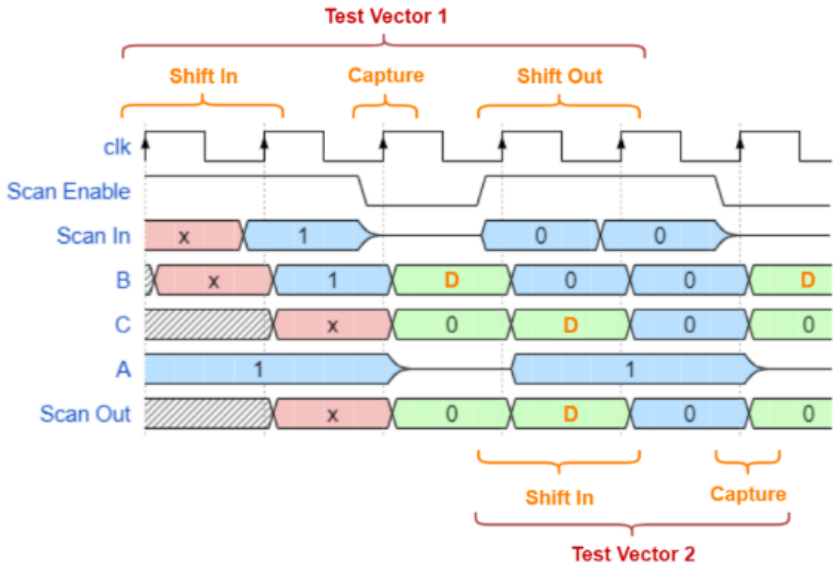


Figure 28: Full ATPG Stuck-At test: shift-in, capture, shift-out.

Through Figure 28 it is observed a complete ATPG sequence since it has the 3 stages (shift-in, capture, shift-out) and it corresponds to a Stuck-at fault test as there is only one Capture cycle. In case of Transition fault tests, it is required two cycles during Capture.

C. Compression

As mentioned, scan chains are formed by SFFs serially connected through their scan-in and scan-out signals. However, in very complex designs there could be a large amount of scan chains depending on the amount of SFFs. In EOS case, it has 416 scan chains. However, EOS has only 4 channel-in and 4 channel-out. The channels are the physical external ports that could be used as scan-in or scan-out ports. Therefore, in order to keep a good pattern generation throughput in terms of test time and quality, it is necessary to implement some techniques to compress and decompress the test patterns.

Siemens offers through Tessent a sub-tool called TestKompress which helps to implements compression techniques to create test patterns with considerably less test data volume and therefore reduced test times on ATE. The compression architecture is described through Figure 29, it is formed by two blocks the Decompressor and the Compactor. The Decompressor objective is to handle the compressed input patterns and through some techniques extract the patterns, decompress them and input the scan chains. At the scan chains inputs, it is necessary to recompress the patterns which is the task of the Compactor.

When the compression process is applied it is called EDT (Embedded Deterministic Test) ON. However, for debug purposes compression could be disabled. In that case it is said that the compression is bypassed (EDT Bypass). Evidently when bypassing the compression, the test time is severally affected since the 416 scan chains are connected in a way, they form only 4 chains (for the available channels). Since test time is very important, using compression is very necessary.

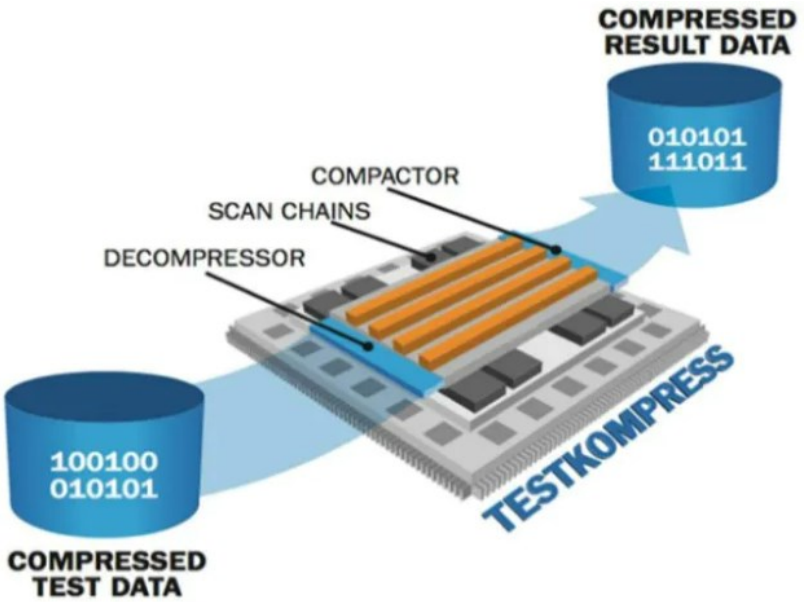


Figure 29: Tessent TestKompress Decompressor and Compactor Structures [15].

D. Testbench Type

Last ATPG tests feature is the testbench type. The standard testbench is known as Serial testbench which represents exactly the behavior of the patterns used for ATE. However, due to the large number of scan chains (416) and SFFs, this testbench takes a considerable amount of time to be simulated. The steps which affect the most the simulation time are the shift-in and shift-out processes. While as mentioned, Capture is only one or two cycles. In some cases, including EOS, Serial testbench generation is configured to include only the first N patterns (N complete shift-in, capture, shift-out processes) where N is between 10 and 100. This way simulation time is reduced.

For simulation purposes it could not be very necessary to simulate the shifts processes since their main objective is to load the SFFs with the desired input pattern sequence before capturing the combinational logic response. But the most important objective is to analyze that the response of the logic is the correct. To simulate this behavior directly without having to load and unload the chains each time, it is designed a Parallel testbench. It is called Parallel testbench because all the chains are forced to the desired final values (after shift-in). Same way, in order to avoid shift-out process, Parallel testbench reads directly the values from the registers saving all the simulation time required for shift-out. This technique is valid only on simulation since physically there is no way to read the SFFs or program them without the shift processes.

4.5.3 ATPG Testbenches

G0B0S0: Transition, Chain test, Compression ON, Serial Testbench.

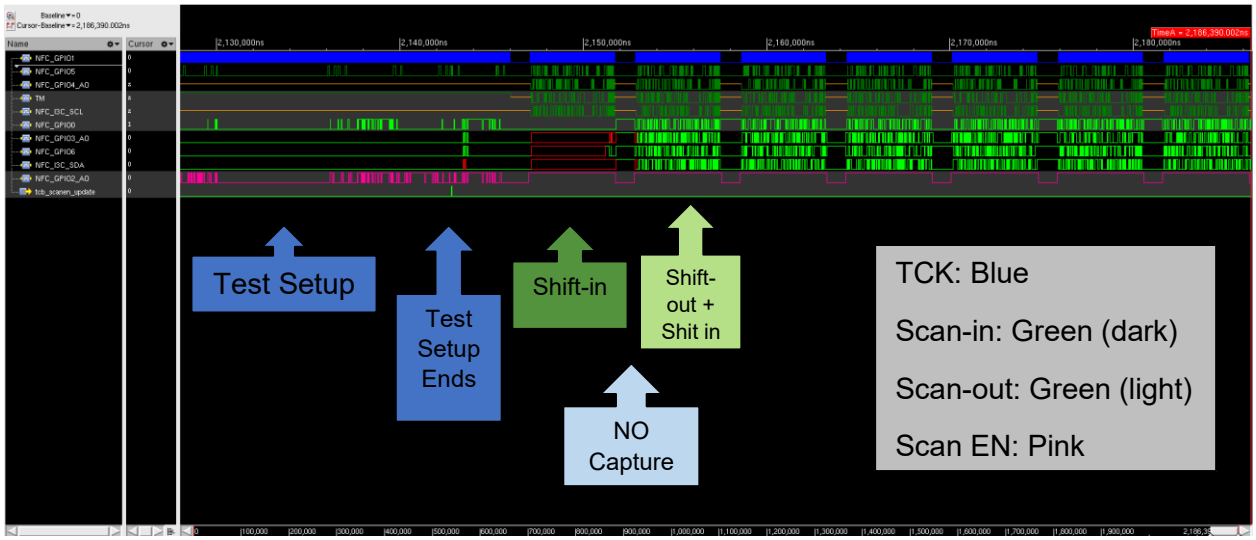


Figure 30: ATPG Test 1.

Figure 30 shows a G0B0S0 test which means that the scan chains from all the islands are disabled, and the test target is only the TOP level scan chains. Fault model is Transition however this is not relevant since it is a Chain test, therefore, the objective is to test the integrity of the SFFs chains and not to perform Capture. Compression is ON (EDT ON), that means that decompressor and compactor are enabled. Finally, it is a Serial testbench since the objective in this case is to simulate the complete shift-in and shift-out processes.

This simulation ensures that the device can be correctly configured into scan mode and that they are able to shift data in and out through the scan chains at a targeted frequency. A way to test this is to analyze that the scan-in patterns are equivalent to scan outputs response. Since it is G0B0S0 this affirmation is only valid for TOP level scan chains but not sure for the islands. Due to Compression is ON and it is a Serial testbench it is also possible to confirm that the EDT is working as expected. In case of an EDT configuration mismatch or timing violation, the chain test will fail and the mismatches on the scan outputs will be displayed in Cadence SimVision Console. A possible cause of a wrong behavior of the Decompressor or the Compactor could be an incorrect setting during Test Setup. It is important to mention that

during Test Setup some TCBs and TPRs are programed in order to set the required configurations for ATPG tests.

From Figure 30 it can be observed TCK, Scan INs, Scan OUTs and Scan Enable signals. TCK which is the clock signal is represented in blue color and corresponds to GPIO1. It is observed that TCK is not toggling when Scan Enable (GPIO2_AO) is 0. This is expected since it is a Chain test, and it is not desired to execute Capture operation. As described previously, EOS has 4 scan-in channels which correspond to (GPIO5, GPIO 4, TM and I3C_SCL) and 4 scan-out channels (GPIO0, GPIO3_AO, GPIO6, I3C_SDA). Due to some of these ports are GPIOs, they are not only used as scan ports but also as JTAG ports which is the case of GPIO0, GPIO2_AO and GPIO5 which correspond to TDO, TDI and TMS respectively. JTAG ports are used during Test Setup configuration which corresponds to all patterns before ”tcb_scanen_update” signal observed at Figure 30. This last signal shows the last TCB (SCAN_CONFIG_tcb) that is configured during Test Setup procedure before starting ATPG test to isolate the TAP from its control signal (TMS) to prevent a JTAG FSM state modification, therefore an undesired change on the DR setup.

The final step to analyze the waveforms is to understand the shift-in and shift-out processes. During shift-in, scan enable is 1 (GPIO2_AO pink signal), as it is a serial testbench, each chain must be loaded serially. Same process must be done when unloading the chains (shift-out). As mentioned, during a shift-in it is possible to execute a shift-out so data is read and write on the chains. However, as observed it is only possible after the second patterns because during first pattern there is no data to shift-out (red waves on scan-out ports).

G0B0S0: Stuck-At, Full ATPG test, Compression ON, Parallel Testbench.

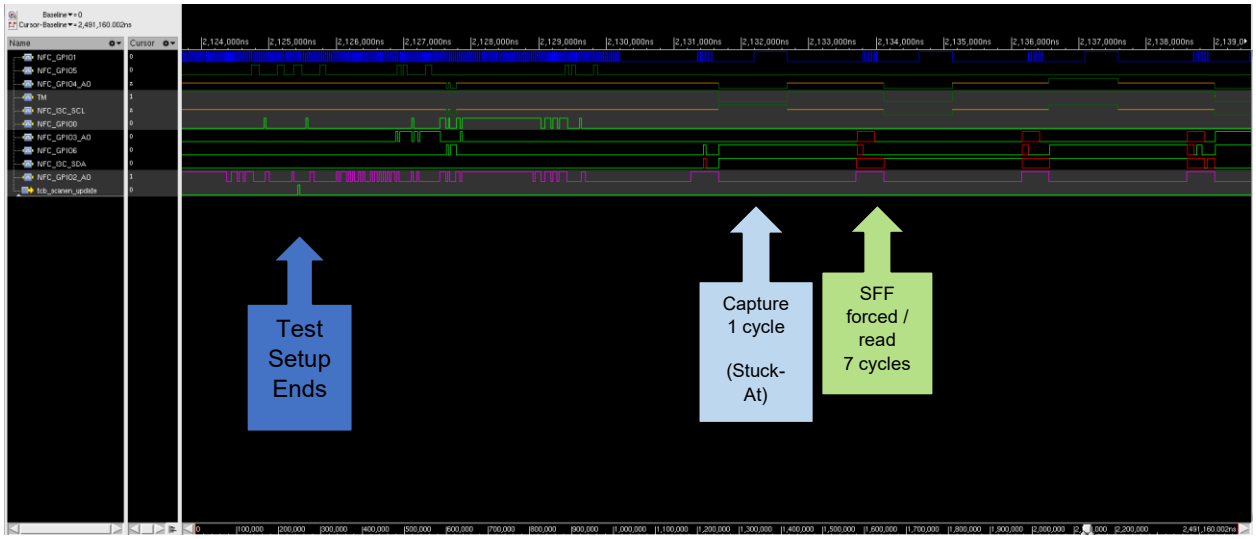


Figure 31: ATPG Test 2.

This second case corresponds to Stuck-At fault test. Due to it is a Full ATPG test, Capture is developed to obtain the response of the logic and capture it on the chains. However, since it is a parallel testbench, shift-in and shift-out are not simulated. Instead, the chains are preloaded directly with the right values (instead of shift-in) and directly read in parallel (instead

of shift-out). Since it is a parallel testbench, it is not relevant if Compression is enabled or disabled because on simulation the stimuli are forced at the SFFs level.

As observed through Figure 31 during scan enable signal equal to 1 (which means shift-in, shift-out), there are only 7 pulses of TCK, during these pulses the chains are parallelly loaded and read, this is equivalent to the shift processes. Those 7 pulses are also necessary to load some parallel flip-flops which are not part of the chains, so cannot be preloaded as SFFs. Then, it is observed a long TCK pulse which is the required clock pulse to test Stuck-At faults, during this pulse the response of the logic is captured by the chains. Since for this simulation the data is not loaded to the chains through the scan channels ports, they are not giving relevant information. To analyze if the captured data is correct, it is necessary to check directly at the SFF level after the Capture.

For EOS parallel testbenches, TCK period during the 7 shift cycles is adjusted. Originally on Serial testbenches the shift period is 30ns, however on Parallel it must be changed and the time between the falling edge and releasing the chain forces is needed to be 11ns. The objective is to have a longer time to avoid timing issues due to clock propagation. It also needed to have a delay time between the forces are released and the Primary Inputs (PI) are forced for the two events not to happen at the same time to prevent simulation issues, that is why the forces occurs 1ns after the beginning of the new cycle. Finally, as observed through Appendix Figure 4, the period is increased to 41ns with the same ON time (15ns) but adding 11ns OFF after. These modifications to avoid timing issues are developed by post processing the testbench using scripts to automate the process.

G0B0S0: Transition, Full ATPG test, Compression OFF, Serial Testbench.

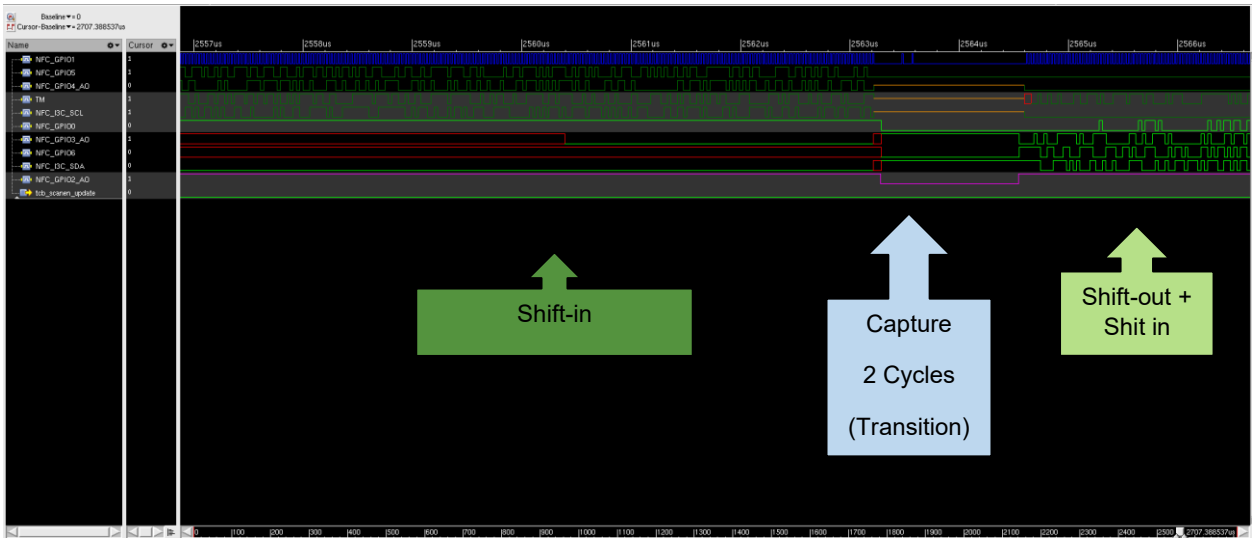


Figure 32: ATPG Test 3.

The third example from Figure 32 corresponds to a Full ATPG Transition test in which the compression is bypassed, it was simulated a Serial testbench. Different from previous simulations in this case the shift-in and shift-out are not developed using TestKompressor algorithms since the decompressor and compactor are disabled. Since this feature is not used, the simulation time is severally affected because the shift-in and shift-out processes requires

more cycles to be completed. Due to this reason, through the DOfile it is configured to generate the testbench for a limited number of patterns (10 first patterns).

Since it is a Transition fault test, it is required two cycles during Capture to perform it. As observed through Figure 32 during Capture, the scan enable signal (GPIO2_AO, pink) is set to 0 meaning normal mode, during this time, TCK (GPIO1, blue) pulses twice (thanks to an internal clock generator). After Capture is done, the chains return to test mode in order to shift-out the data and shift-in the next patterns. Due to it is a Serial testbench it is possible to demonstrate through it if the chains are working properly but since compression is OFF, it is not feasible to confirm if the decompressor and compactor operates in the right way.

5 Improvements for Future Projects

5.1 Multiple TAP ICL representation

As described through Figure 7 EOS DFT architecture matches a multi-TAP Star design in which there is a TOP tap controller and multiple sub-level Taps, therefore there is a multi-level structure with a clearly defined hierarchy. Unfortunately, TASS is not capable to represent this architecture since it can only support a single tap design. In IJTAG case the ICL network generated by DFTShell contains a single tap architecture. Consequently, both TASS and IJTAG work based on a single tap DFT design corresponding to NFC sub-IP tap.

For EOS project purposes both representations are enough since the hierarchical access is handled manually (cycle-based patterns) which is a desired feature, and because external sub-IPs delivers their own patterns (SE and TSMC-flash). However, for future projects NFC tap will be separated into two independent taps in such case it is mandatory to handle a hierarchical design. In that case, the most critical issue is that multiple data registers from different sub-IP taps (TCB/TPR) could have the same address (IR OPCODE). In that case representing the design with a single tap is not feasible with the current approach.

5.1.1 Merged Tap Controller IJTAG

First approach using IJTAG is based on representing a multiple-tap architecture through a single merged main tap. During IJTAG flow there is no verification between the real RTL design and the ICL representation since the RTL is only used to read the interface (TOP level input/output ports) and not the instances (sub-IP RTLs). This means that it is not necessary the ICL to match the real RTL design and this methodology takes advantage of this. However, it is necessary to consider that through this approach since the design is not matching the real design, there is a higher risk to have undesirable effects on the patterns.

The objective is to merge all the data registers into a single tap instance. The main issue is that IJTAG is not able to work in case of two or more data registers with the same address within the same tap sub-IP. Therefore, it is needed to modify the ICL description manually in order to make IJTAG understand that even if two data registers have the same address, they are different. To do so, the proposal is to add an extra bit (LSB) in the instruction register in case of two DR with the same address. This extra bit is used during the OPCODEs definition to make a difference between two identical addresses. Through Figure 33 a hierarchical design with two sub-taps is transformed into a single-tap representation. As observed, IR is 8-bit length and there are two TCBs with the same address but in different taps. To merge them a LSB is added to the IR, and it is assigned a unique value for this new bit for each TCB.

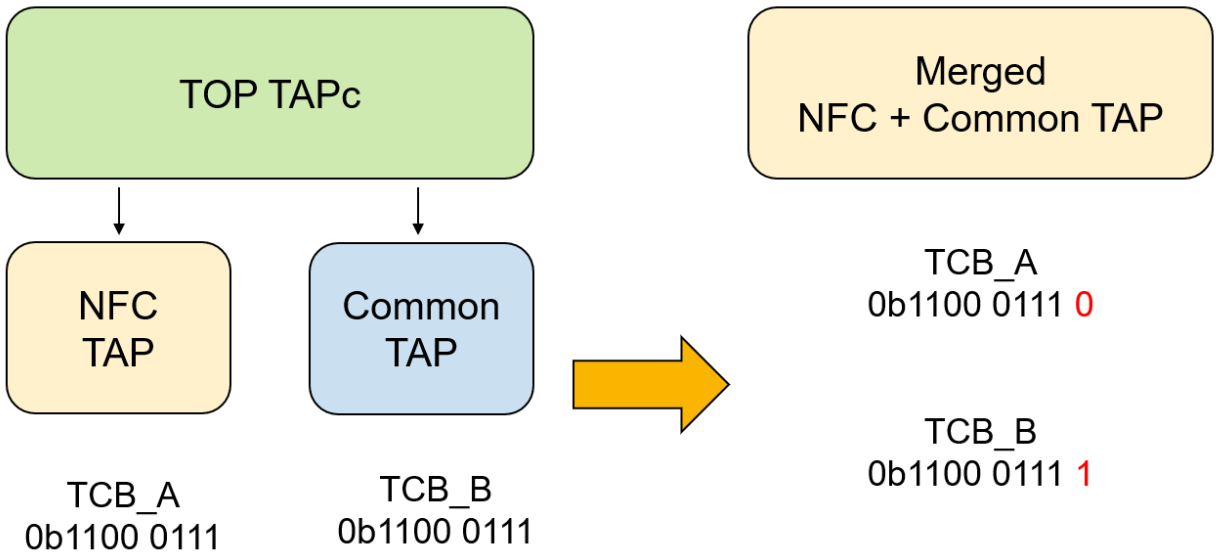


Figure 33: Multi-tap into merged single-tap architecture.

The flow using this approach could be:

- Generate the ICL description of each sub-IP tap through DFTShell in independent sessions.
- Concatenate all the data registers (TCB/TPR) instruments from all the taps into a single ICL file.
- Modify the single-tap IR by adding the extra LSB, if the address is repeated more than once, it could be added extra bits at the end of the IR.
- Define the OPCODEs by assigning a unique value of the added bits to the DRs with same addresses.

From IJTAG perspective the IR has 9 bits instead of 8 (from Figure 33 and Figure 34 example), therefore it is going to generate the sequences based on this. Figure 34 shows a test case in which it was defined the same address intentionally for a TCB and a TPR and used an extra bit to indicate IJTAG they are different instruments. The effect in patters is that there will be an extra cycle due to the new bit, however this extra bit will be over shifted so the real behavior is not going to be affected. The extra cycle is not significant compared with the complete test sequence so it will not affect the simulation or tester time. Despite through this approach it is possible to handle a hierarchical design it is still not a representation of it. This means that the hierarchy access must be done manually as it has been done previously.

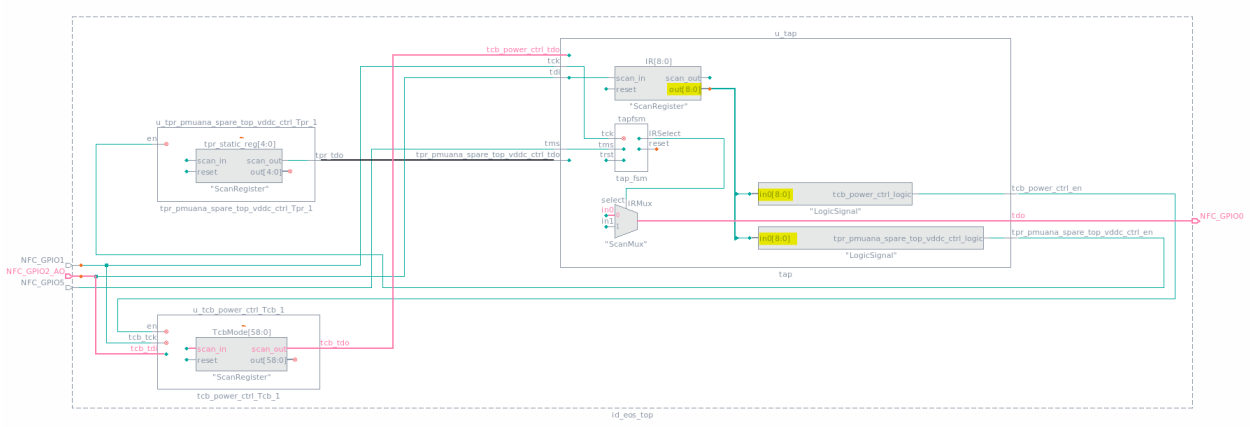


Figure 34: DFT understood architecture by IJTAG (Tessent Visualizer).

5.1.2 Merged Tap Controller TASS

There is the possibility to develop a similar solution for TASS in order to handle the mentioned scenario of a multiple tap design with same DR addresses. In IJTAG case it was necessary to add an extra IR bit since the tool executes an ICL network validation and declare as an error if two DR have the same address. With TASS it is possible to do implement the same solution. However, TASS does validate the TD input files (DR and tap interconnections) so, it can use multiple data registers even if they have the same address. The only requirement for them is to have different paths and names.

Therefore, for a multi-tap architecture it is possible to merge it into a single-tap architecture by concatenating the TD chains and describing all the OPCODEs in the same tap TD file. The drawback is that it is not matching the real design, so the hierarchical access must be handled manually.

5.1.3 TDO-Gated Multi-Tap Representation IJTAG

The third solution is based on modifying the ICL network in order to generate a hierarchical representation of the multi-tap Star architecture. First trials were developed to gate TCK in order to match the original design as observed in Figure 35. The target is to implement a ClockMux controlled by the UP TCB (TCB in charge of selecting the sub-IP taps) before the TCK input each TAP level. Despite the clock gating is done within the TAM block level using this ClockMux is the closer ICL description compared with the original design.

However, trials demonstrated that IJTAG is not able to work based on an ICL network in which TCK is gated. During the ICL elaboration and check, IJTAG reports an ICL semantic error. IJTAG does not allow to use a ClockMux output as an input of a sub-IP TCK port.

Unfortunately, this error cannot be downgraded into a warning, so it is necessary to change the approach to handle a multi-tap architecture.

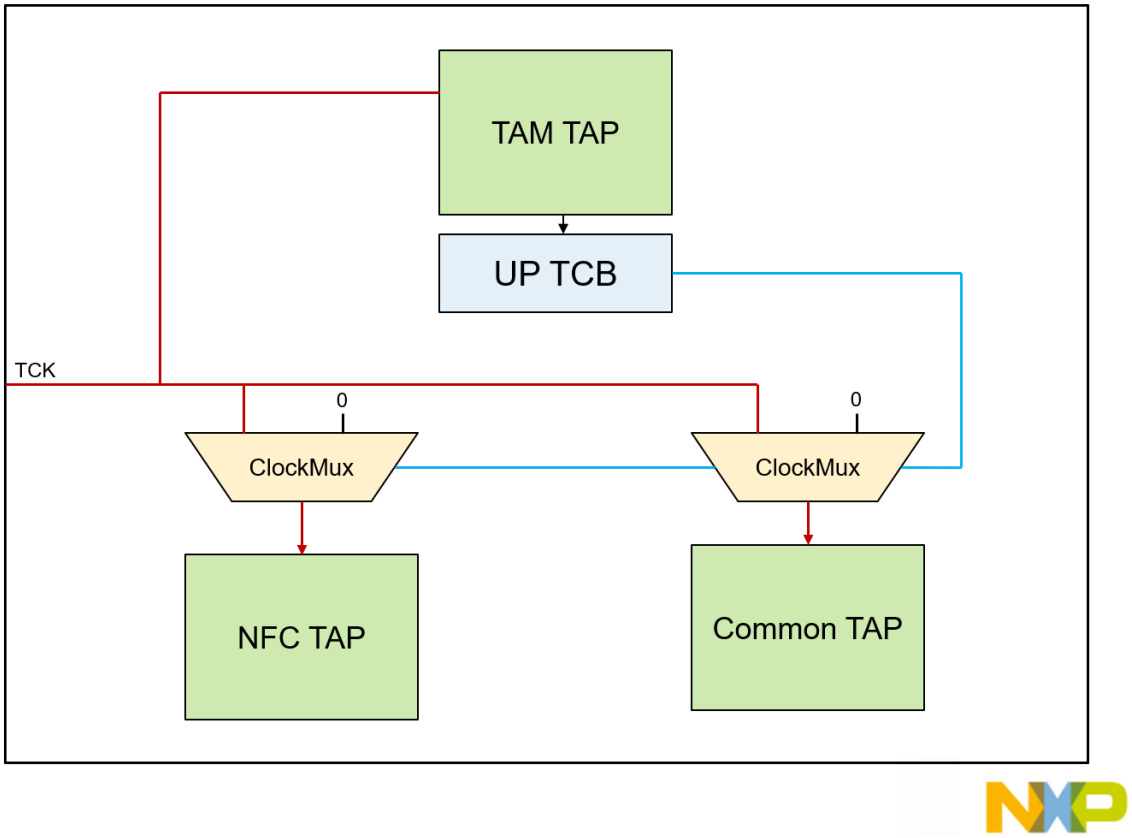


Figure 35: TCK gated ICL design.

Since there is no possibility to gate TCK it was decided to gate TDO. The main objective is to describe a hierarchical architecture that IJTAG understands and supports. This way, during pattern generation it is expected the tool to generate the sequences for the different tap accesses. It is proposed to gate TDO through a ScanMux controlled by UP TCB as observed in Figure 36. During IJTAG flow it was observed an ICL error because the ScanMux is deselecting the TAM TAP controller. It means that the main tap loses control and will never be retargetable.

In practice, this is not the case because the TAM TAP was designed in order to spy the signals, meaning that the TAM TAP never lose control and it is retargetable. Also, it was implemented some extra logic to prevent the top tap to reset accidentally when resetting the sub-IP taps [12]. However even if these features work as expected at hardware level, it is not possible to represent them into an ICL description. Therefore, for IJTAG since the UP TCB is configured by the TAM TAP and this TCB is in charge of selecting the TDO, the tool must generate the patterns to select the right TAP related with a TDO and the desired DR to be

read/written. Once this new TAP is selected there is no way for IJTAG to retake TAM TAP control and rewrite UP TCB to select another TAP.

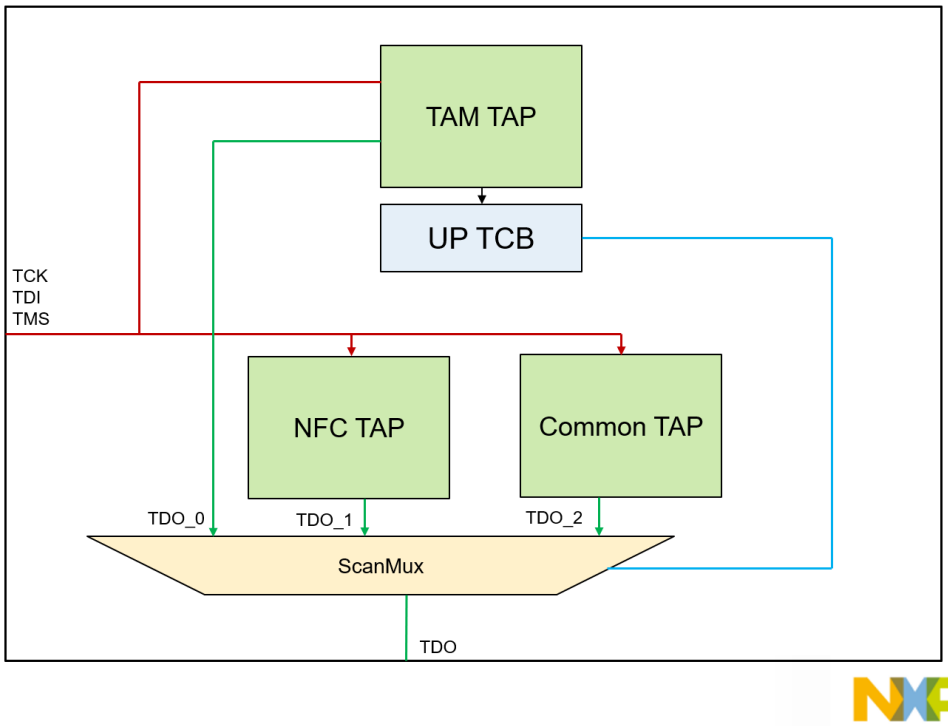


Figure 36: TDO gated ICL design.

Nevertheless, the ScanMux error can be downgraded into a warning. Consequently, IJTAG understands the multiple tap architecture and the TDO gating mechanism. It is shown through Figure 37 the IJTAG ICL schematic on Tessent Visualizer tool for EOS project TOP TAM tap and FSM tap.

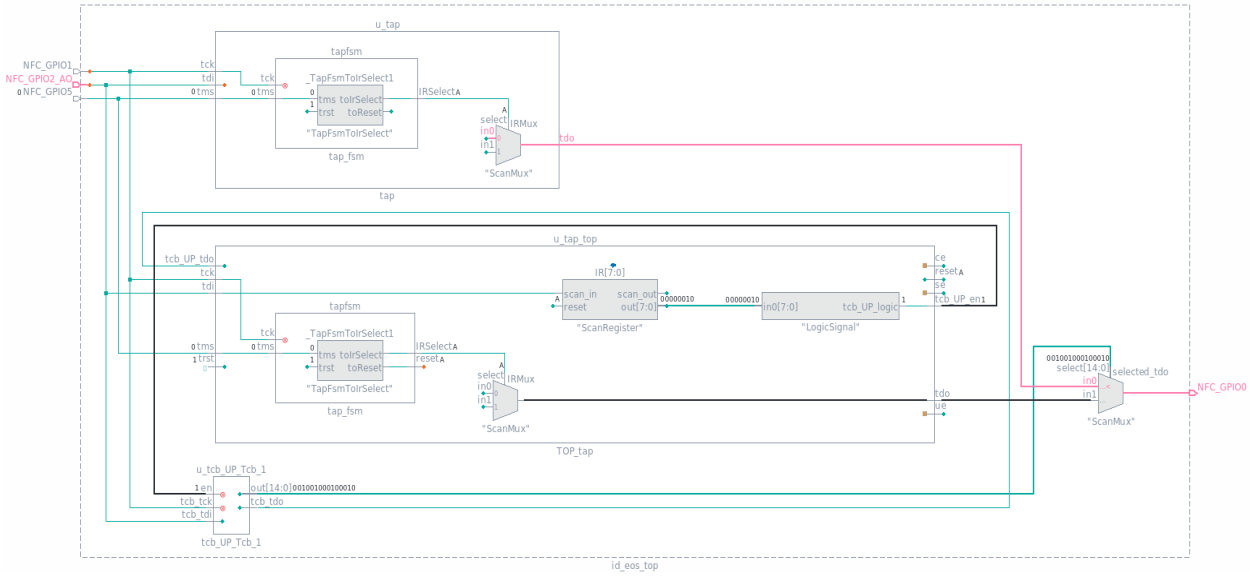


Figure 37: Hierarchical ICL network description understood by IJTAG.

As mentioned, for IJTAG it is not possible to retake TAM TAP control. Because of that, it is necessary to use iState command in the PDL pattern description when it is required to access a different sub-IP or to retake TAM TAP control. With iState it is possible to manually define the state of the ICL network, but it does not generate any patterns. The flow is as follows:

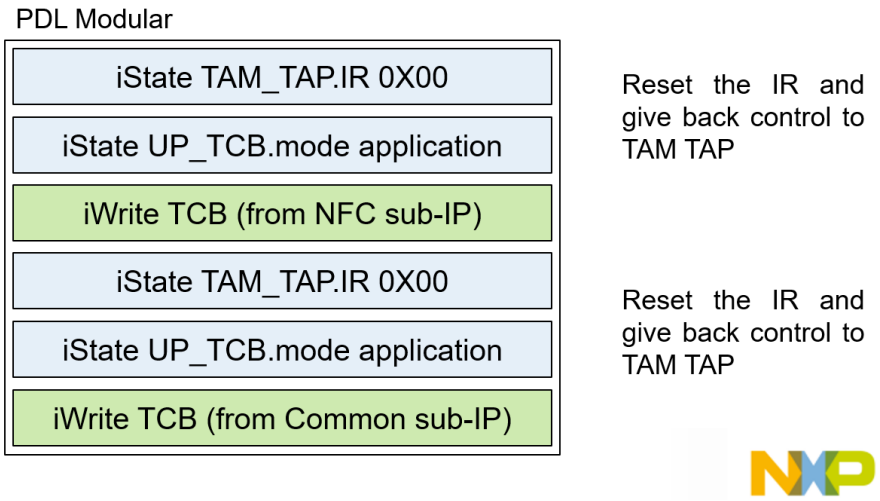


Figure 38: Hierarchical PDL example using iState.

5.2 Force JTAG Ports on PDL patterns

As mentioned on previous chapters (4.4 New Test Sequence Approaches) the only way to force JTAG ports with IJTAG is through sub-procedures used on Test Setup. However, complex functions which require loops, arguments or different timeplates are hard to implement through sub-procedures since their language is very limited as it is designed to describe simple cycle-based functions. The second drawback is that this is available only with Approach 3 in patterns -scan context losing all the advantages of patterns -ijtag context and pattern sets.

Therefore, the target is to use PDL to force JTAG ports since PDL is a language with more flexibility and as mentioned, it is compatible with TCL commands. Finally, PDL can be used on patterns -ijtag context taking advantage of pattern sets. The way to force and compare the IC ports is through iForcePort and iComparePort PDL commands however this is limited for non-JTAG ports. Siemens propose a Tessent IJTAG command called iSuspendPdlRetargeting in order to force JTAG ports. This command is going to stop the IJTAG retargeting tool which is in charge of analyzing the state of the ICL network. In other words, through it, there is the possibility to force JTAG ports within PDL patterns and IJTAG will not complain about it or any modification of the ICL network state. However, after this command is used it is mandatory to use an iReset command which purpose is to give control again to IJTAG retargeting tool. Since the ICL network state could be modified during this process it is necessary to use iState to specify the modifications (network end state, e.g., TCB or TPR changes). A flow diagram for iSuspendPdlRetargeting is shown through Figure 39.

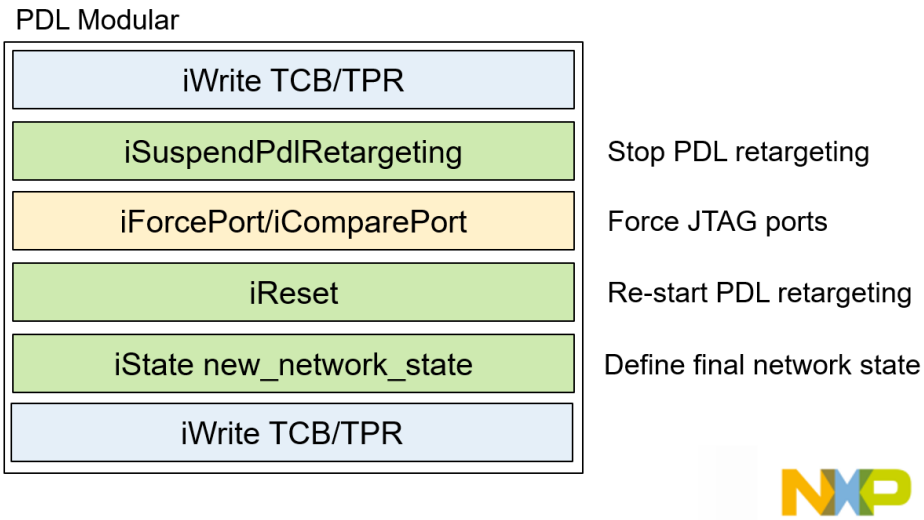


Figure 39: iSuspendPdlRetargeting flow to force JTAG ports.

However, on current releases (up to Tessent 2022.2) the only way to give control again to the retargeting tool is through iReset command. Unfortunately, this command also generates a soft reset (TMS set to 1 during 5 TCK cycles). This is an undesired behavior that could affect the TAP FSM state. The second issue is the limitation to force all JTAG ports. It is not possible to use any command to force TCK port, so it is always pulsed as a clock during PDL patterns and does not react to iForcePort. The way to solve these issues is through a workaround to modify the iReset effect and force TCK port.

In order to eliminate iReset effect it is proposed to develop a post-processing step of the output files (STIL and Testbench) to remove the soft reset. The first step is to create a new PDL procedure called “iResume” which will contain the iReset command but also some comments (through iNote PDL command) before and after it. These commands are useful to automate the post-processing. In case of STIL patterns, the file is post-processed using a script that searches the comments and forces TMS signal to 0 during iReset, this way it prevents the soft reset. For the Verilog Testbench, there is the possibility to add special Tessent comments (tessent_pragma) before and after iReset that which will force the TMS GPIO to 0 during the iReset and releases it at the end before exiting iResume. This way it is not necessary to post-process through scripts the Testbench.

For TCK port (GPIO1) issue the proposal is to create a pseudo-port to replace TCK. The original TCK from the ICL definition will be pulsed during all the PDL pattern sequence, but the pseudo-port since it is not TCK, it can be manipulated freely. The first step is to declare it as a clock in the DOfile. Then it is needed to pulse inside all the timeplates definition in which it will be used as TCK (e.g., for a part of TMER modular since TCK is forced to a fixed value during bootstrap, it must not be pulsed in the corresponding timeplate). Since this pseudo-port, different from TCK, is not pulsed by default, it is necessary to pulse it at the beginning of each PDL pattern when required. Finally, when writing the patterns (write_patterns command) the original TCK is overwritten or replaced by the pseudo-TCK which matches the real requirements (using the command set_write_pattern_options).

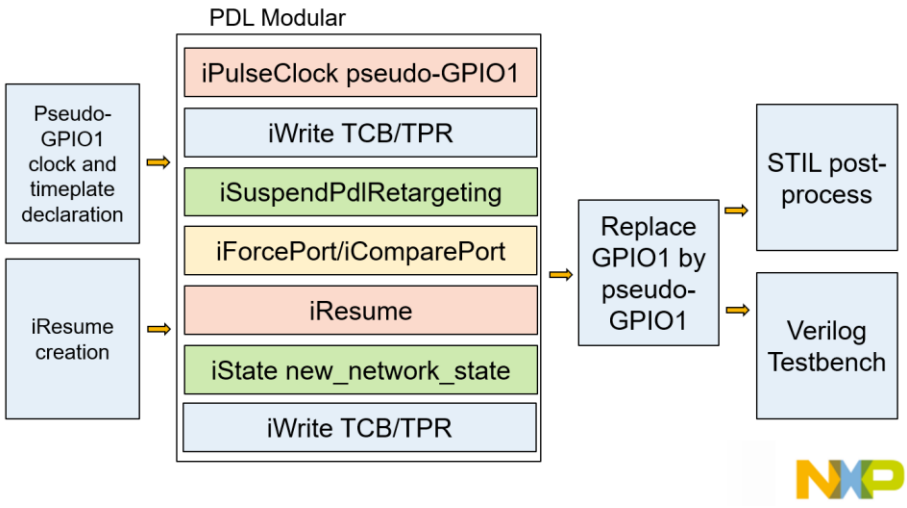


Figure 40: Force JTAG ports flow through pseudo-TCK and iResume.

5.3 Use of external IP Patterns

Due to some IPs deliver their own patterns it is necessary to integrate them into the test sequence for simulation. This is the case for Secure Element patterns that must be integrated in test setup for ATPG patterns, or the Flash patterns coming from TSMC. Since IJTAG supports only PDL or SVF input pattern formats, it is not possible to directly use STIL input patterns. First approach was to use a Tessent utility called STIL2MGC (STIL2TESSENT on last Tessent version 2022.2) This is a tool that converts STIL files into a DOfile and a test procedure file. The objective is to call this sub-procedure into test setup as other cycle-based procedure. The main limitation with this solution is that it works only with test setup approach, since sub-procedures cannot be called within PDL files. Therefore, for ATPG patterns, the solution fits adequately, but for non-ATPG patterns it can only be used with pattern Approach 3 (Test Setup Pattern Description).

It was developed some trials with several STIL pattern files, but the tool presented some limitations. It is only possible to use STIL2MGC for STIL patterns with chain definition (scan structures and scan chains). Due to this constraint, it is not possible to generate a procedure file for all STIL files.

Second solution is the development of internal NXP scripts to transform STIL into PDL. This is the best option and the selected one for future projects because PDL format is allowed in all pattern generation approaches. Additionally, the external STIL patterns (both from SE and Flash) are cycle based described, meaning that there will be a one-by-one transformation into cycle based (iForcePort and iComparePort) PDL commands. An example for SE patterns used on Test Setup for ATPG is shown through Figure 41.

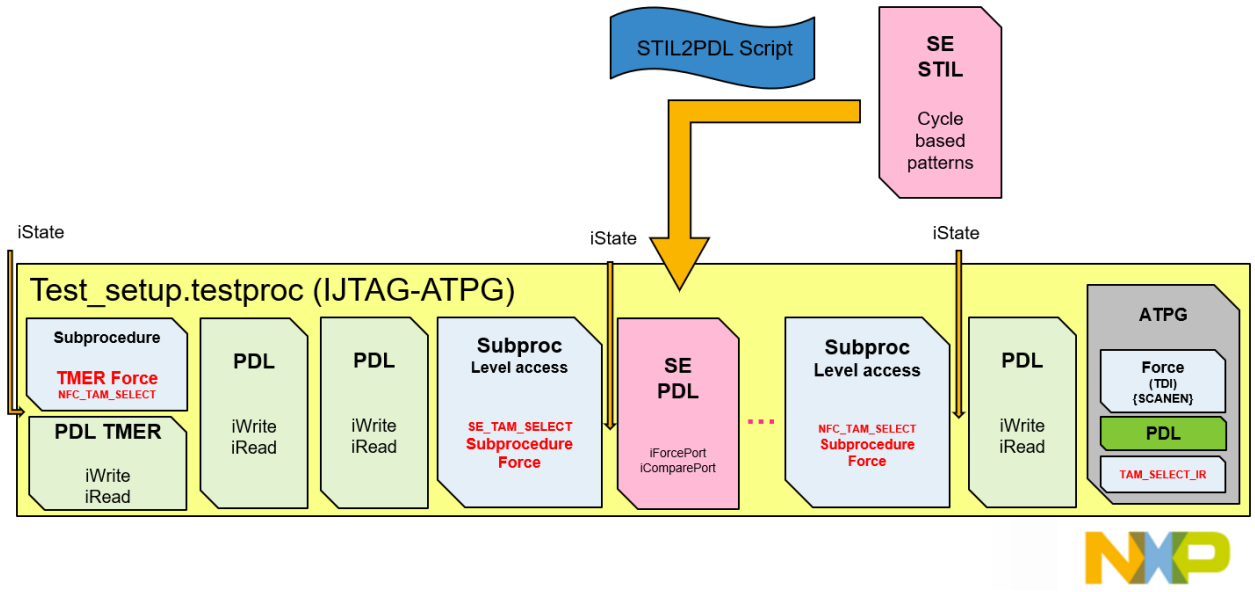


Figure 41: Test Setup for ATPG using STIL to PDL transformation.

5.4 Detailed Annotations using iReadVar/iWriteVar

One important requirement is based on the annotation level of the STIL output patterns. Due to the generated patterns will be used by test engineers on ATE, the delivered STIL files must be well commented for debugging purposes. The minimum requirement is to have per-cycle annotations with the information of the JTAG bits. Detailed annotations regarding the read and write processes are also required. However, IJTAG does not give detailed comments on STIL such as the full DR information when using iWrite/iWrite commands for specific DR bits. On the other hand, TASS supports all these features, that is the reason way annotation is considered an IJTAG weakness.

IJTAG annotations are created through iNote PDL command which prints the comments in the output pattern files. iNote is a command that can be used before or after and iRead/iWrite, so it is not possible to add a comment during the iRead/iWrite process. Nevertheless, IJTAG supports the association of an iRead or iWrite command with a symbolic variable such as iReadVar and iWriteVar [8]. There is the possibility to use a "TESSENT_PRAGMA annotation" within an iNote. This particular annotation command allows the tool to track the symbolic variables during the PDL retargeting process and replace them with an associated ICL instance (e.g., TCB/TPR). Which means that it is possible to associate the symbolic variables to each of the DRs of the ICL description and then, generate the pattern output file such as STIL with a high annotation level.

The process starts with the extraction of the ICL DRs and their "aliases". An alias is a name for a virtual vector signal that contains one or several signals from an ICL structure such as a TCB or TPR [8]. Aliases help to understand better the function/origin of the TCB/TPR register signals. Then, it is necessary to create one iNote using the new symbolic variable approach for each alias of the TCB/TPR. This process must be repeated several times for each TCB and TPR. Due to and iRead can only be associated with an iReadVar and an iWrite with

an iWriteVar respectively, it is required two iNote commands for each DR alias (except for TCB which does not support iRead). Finally, since iNote is a PDL command it is possible to develop an iTopProc for each TCB/TPR which can be saved in a PDL file to be added to the PDL include database. This iTopProc can be used according to user needs before an iRead/iWrite. A general overview of the flow is shown through Figure 42.

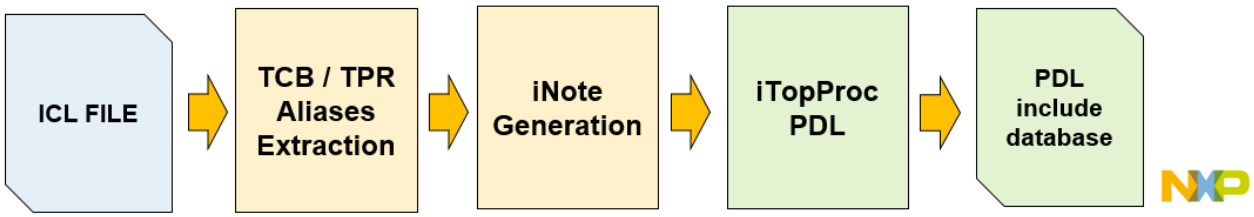


Figure 42: iNote iTopProc PDL procedures generation using symbolic variables.

6 TASS & IJTAG Comparison Summary

Table 3: TASS vs. IJTAG comparison.

Topic	Feature	IJTAG	TASS
Basic Patterns		√	√
	Timing	√	≈
	Modular Approach	√	√
Force JTAG Ports		≈	√
	iSuspendPdlRetargeting	**X**	
	Test Setup Approach 3	≈	
	Test procedures from TCL	≈	
	iResume	**?**	
External STIL Patterns Reuse		**?**	√
	SVF/PDL + ICL	≈	
	STIL2MGC	**X**	
	STIL input support	**X**	√
	STIL to cycle based PDL	**?**	
ATPG		√	↓
	Bootstrap on simulation	√	≈
	Accurate Timing on TS	√	≈
Annotation Level		**X (≈)**	√
	Current: Per Cycle Annotations	√	√
	Expected: Detailed Annotations	**X**	√
	iWriteVar / iReadVar	√	
User Friendly		√	↓
Hierarchy Understanding		√	**X (≈)**

This comparison study was based on 7 main topics as observed in Table 3. The features for each topic were evaluated for both tools based on the risk and difficulty to implement. The symbols definition is the following:

√ : No risk, implemented and tested, workarounds do not represent a risk.

↓ : Slight disadvantage compared with the other tool, but feasible without risks.

≈ : Moderate risk, feasible but required workaround or post-process.

X : Not feasible.

? : Moderate risk, workaround is not completely tested.

X(≈) : Limited workaround which does not accomplish all the requirements.

First one is Basic Patterns which corresponds to only PDL or TDL based patterns for IJTAG and TASS respectively. It has been determined that IJTAG supports multiple timing definitions through timeplates while TASS does the same through wavesets. In IJTAG case, when multiple timeplates must be used within a modular it is required to open a new pattern set. However, for TASS to use multiple wavesets in the same modular it is necessary a script-based workaround, that is why TASS timing feature is marked as "≈**".** Nevertheless, both tools can support and represent successfully the Modular approach and all its requirements, reason why Basic Pattern is "√" for both.

Second topic is how to force JTAG ports. This requirement is fundamental since it is used for the bootstrap initialization, cycle-based functions such as the manual multi-tap access and in some specific modular patterns. In case of TASS, TDL language supports this characteristic without any risk or limitation. However, in case of IJTAG as observed in previous chapters, it is not possible to force JTAG ports with PDL directly, therefore some workarounds are required. Since there is the soft reset issue when using iSuspendPdlRetargeting command with iReset, this solution is discarded. Second possibility is to use Approach 3 which is based on sub-procedures, but as mentioned these are based on a very limited language. A third possibility which is an extension of Approach 3 is to automate the sub-procedures creation by using TCL scripts. However, both last two solutions are not worth due to time and complexity.

Therefore, the only feasible solution to force JTAG ports with IJTAG is to use the iResume approach (5.2 Force JTAG Ports on PDL patterns) which requires a workaround, but it fulfill successfully the required feature.

The third requirement is related with external IP patterns in STIL format. Since IJTAG only supports either PDL or SVF the initial solution is to ask one of this pattern formats. However, this is not possible for third party patterns (TSMC case). As it was mentioned, STIL2MGC tool is not suitable on this case. Finally, IJTAG does not support STIL format in all its pattern contexts. Consequently, the only feasible solution is the STIL to PDL transformation through scripts (5.3 Use of external IP Patterns). Since this solution has not been completely tested it is marked as "?".

Fourth topic is ATPG patterns in this case, IJTAG has no limitations compared with TASS. Additionally, as shown previously (4.5.1 ATPG PDL-based Test Setup) IJTAG simulates the bootstrap and is more accurate in timing since it supports multiple timeplates in Test Setup procedure. Currently, these two features are not supported with TASS, but they are feasible with additional workarounds. In conclusion ATPG patterns are both well supported by IJTAG and TASS with some benefits from IJTAG.

The next requirement is based on the annotation level of the STIL output patterns. It has been mentioned that this characteristic is fundamental because it is a requirement used not only by DFT engineers but also test engineers. Since IJTAG does not accomplish the detailed annotation requirement, it was developed a workaround to solve it. Different from other proposed solutions, this one requires only PDL manipulation. However, it requires certain automation through scripting in order to generates the PDL procedures. Which is a disadvantage compared with TASS that fulfills this requirement without additional work.

Next topic aims to compare TDL and PDL languages from the user perspective. After the different test developed on this study it was determined that PDL has certain advantages. Since PDL is based on TCL, it is user friendly and easy to learn while TDL has its own structure for variable creation, loops, others. PDL files also have a clear modularity and hierarchy. It is possible to target the instrument level and then create pattern sets to reach the Modular approach or it is possible to target the top level avoiding independent PDL for each instrument, therefore PDL is flexible. But, in terms of functionality all what is possible to do with PDL is also possible with TDL.

Final requirement is based on how accurate it is possible to represent the EOS DFT design (hierarchical multi-tap architecture). Initially, the ICL or TD delivered by DFTShell does not match the real design since the tool generates a single tap representation, therefore, in both cases it is necessary a post-processing stage. In case of TASS, the tool does not understand a hierarchical architecture. TASS workaround is useful to avoid a blocking point and handle a multi-tap scenario, but the real design is not represented by the TD, so the hierarchical access must be described manually. In case of IJTAG is has been observed that it is possible to post-process the ICL in order to represent the hierarchical architecture, but it gates TDO and not TCK so even if the tool understand there is a multi-tap structure, the representation does not match the real design. In conclusion, IJTAG is better than TASS in terms of hierarchy management since there is a possibility to represent a multi-tap design but with some limitations.

7 Proof of Concept using MUTEST

The final objective of this study is to develop a proof of concept of IJTAG pattern sequences through on-silicon trials using an ATE. MUTEST is a FPGA-based ATE system used by NXP DFT engineers to debug patterns before their final deliver to test engineers. This step permits to reduce drastically the pattern debug time, so the DFT engineer is able to make his own on-silicon trials.

In order to test pattern sequences both for ATPG and non-ATPG it is necessary to post-process the STIL files. During this step it is used a script which is going to analyze the STIL and develop some modifications required by MUTEST. For example, this ATE does not allow STIL patterns to start or end with loops, so the post-process step prevent this. Additionally, MUTEST does not support STIL timing variables, so it was developed a solution through an IJTAG flow modification plus scripting to remove the variables and equations from STIL and replace them with numerical values.

MUTEST works with MuTool IDE which is capable to import and generate tests for each STIL file and allows to assign different power and timing features to each of them [17]. This ATE can also concatenate the tests into complete flows, therefore MUTEST and MuTool are compatible with Modular Approach. Multiple ATPG and non-ATPG pattern sequences were tested giving positive results. It is observed through Figure 43. A passing trial in MuTool environment.

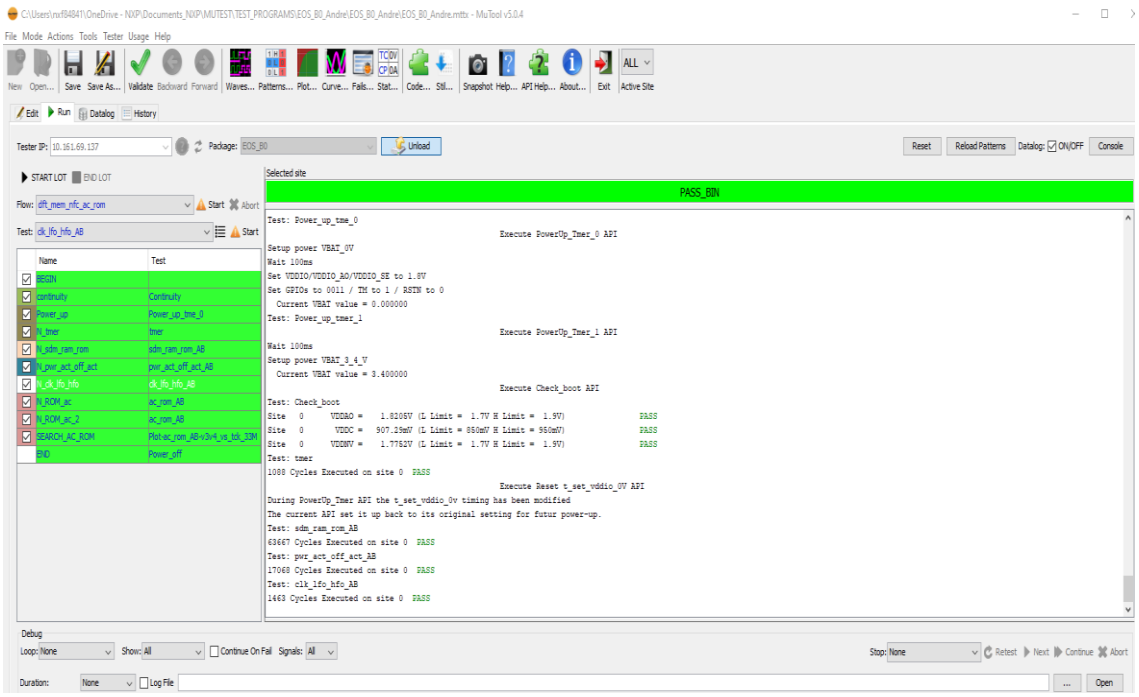


Figure 43: Pattern sequence test example using MuTool IDE.

Through MUTEST, it is possible to perform SHMOO plots to test the device under different conditions of voltage and frequency. SHMOO plots have a great importance in the debugging process. Through them, it is possible to detect pattern failures related with pattern stability issues [18]. In order to develop exhaustive tests, test engineers have to develop several trials of the patterns provided by DFT engineers under different process conditions and with several combinations of voltages and frequencies. To represent all these tests, this information is print into a SHMOO plot. Basic SHMOO plots are formed by two axis X-Y, where X-axis usually represents frequency (or period) while voltage is represented through Y-axis. But there is also the possibility to vary a third variable through a third-dimensional SHMOO plot [18].

Through Figure 44 it is observed a SHMOO plot generated with MuTool IDE. It corresponds to a MBIST (Memory Build-In Self-Test) ROM test in which the two parameters are VDDC which is the logic/RAM/ROM VDD voltage and TCK period. The default TCK period is 30ns which corresponds to 33 MHz while VDDC default value is 0.9V. These conditions guarantee the pattern stability. On this test, TCK was swept from 50% to 120% its default value, similarly, it was done with VDDC but between the 60% and 100% of its default value. It is shown in green all the tests in which the patterns succeeded and in red the failing cases. It is observed that when increasing VDDC voltage, patterns succeed at higher frequencies until reaching a threshold close to TCK ≈ 18.5ns after this point, patterns fail for faster frequencies despite VDDC increment. On the other hand, for VDDC voltages lower than ~575mV, patterns always fail regardless of VDDC.

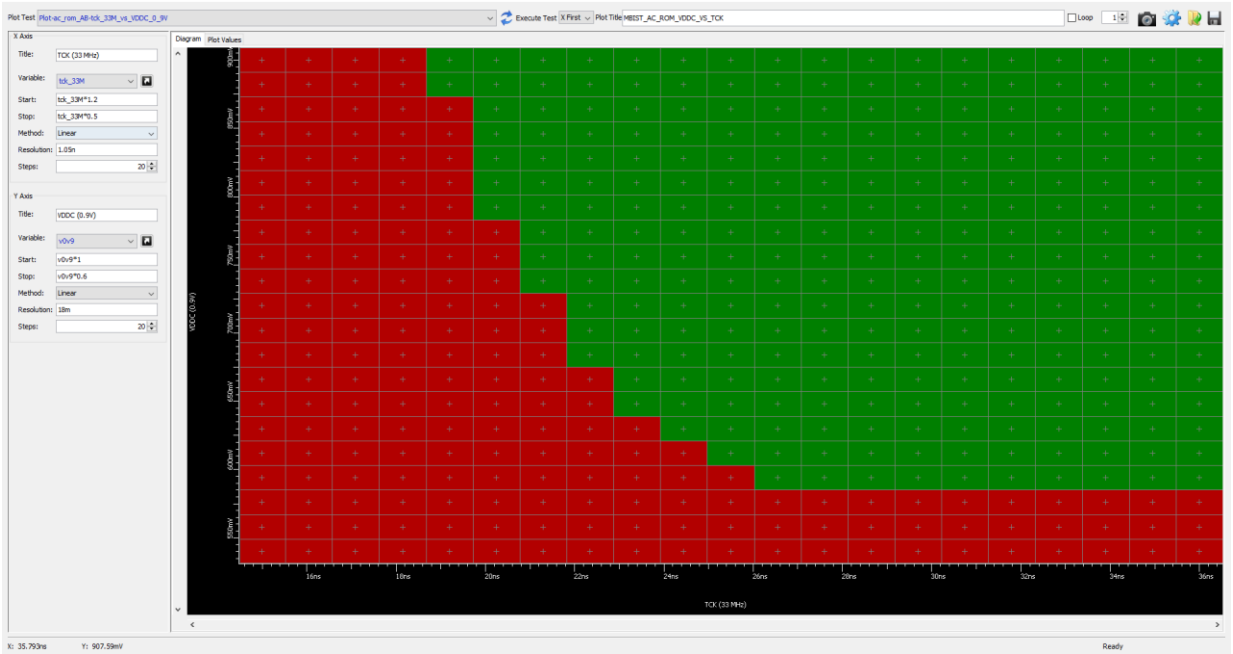


Figure 44: SHMOO Plot generated with MuTool IDE.

8 Conclusions

This study shows the importance to guaranty product testability and how to implement it during the design process of an IC. This encourages the development and application of different standards such as the IEEE 1149.1 which ensures device testability. The usage of standards is fundamental in the industry because guarantees that the design and its architecture is compatible with market tools used during the DFT flow such as the ones used for pattern generation. The main objective of this work was to compare and analyze two of these DFT tools (TASS and IJTAG) for a possible migration for future projects. The key point to analyze was if the new tool accomplishes with the requirements of NXP for DFT pattern generation by giving enough proofs and arguments to take a decision.

Thanks to this work it was possible to study each of the steps of the DFT flow such as the DFT RTL Generation, DFT Verification and DFT Pattern and Testbench Generation. Due to the importance of DFT, it is involved during many steps of the IC design development. Therefore, this study demonstrates that choosing a tool instead of another is a complex task in which many parameters must be considered. To analyze them and describe the different test cases and possible challenging scenarios during the DFT flow, it was used on an already tested test vehicle (EOS).

First constraint is related with feasibility which means how easy or convenient is to implement certain feature. As a general conclusion it was demonstrated that there are no blocking points for none of the two tools, but some characteristics does not accomplish all the requirements. This is the case of the expected annotation level with the new IJTAG tool that even if does not block the pattern generation flow, it represents a considerable risk for debugging purposes during testing.

Closely related with feasibility it is complexity. Some blocking points were solved through different workaround processes. Nevertheless, some of them are hard to develop and increases the difficulty of the flow which means risk to commit errors. As an example, it is found the iResume workaround to force JTAG ports with IJTAG, which is feasible and fulfill the requirements of the project but requires additional steps that even if they are tested, they always represent a disadvantage compared with a tool such TASS that can cover this feature without any additional step.

Furthermore, complexity means more time for development and test. In case of external patterns, there is no possibility to use STIL with IJTAG tool, therefore it is necessary to develop a workaround to solve it. Which means additional time to test the workaround and close the new flow. Consequently, all these parameters indicate how feasible is the migration.

However, all these challenges allowed to develop a deep analysis of the current DFT flow which means understand the purpose of each step, their inputs, outputs and how to improve the automation process. Nowadays, automation is fundamental since it reduces cost and time by taking advantage of design regularity. That is why there are tools such as Timnet and DFTBuilder to automate the RTL generation of the DFT structures. Besides, it is always the task of the engineer to design and set the specifications according to the requirements of the project. However, automation is conditioned by the flexibility and characteristics of the tools.

Through this study, it is explained that even if TASS and IJTAG tools could do the same tasks, automation is restricted by some tool's limitations which in the worst case the task cannot be fully automated and require manual manipulation. This is the case of multiple-tap access. In which to access a sub-level tap such as NFC from the TOP TAM tap, the patterns must be described manually cycle by cycle (TASS and initial IJTAG case). This requires a high understanding of the tap controller state as well as the IR and DRs configuration.

During the last stage of this work, it was simulated and debugged several ATPG patterns. It was determined the importance of ATPG and the different available tests in order to detect faults related with manufacture processes. The main target of these patterns is to have the highest possible coverage in order to eliminate the risk to deliver a faulty chip to a costumer. Those simulations were compared with on-silicon trials on ATE which is the final step to ensure that patterns developed with the new IJTAG tool are working as expected.

Finally, through this internship it was possible to have a real experience on the semiconductor design industry with a leading company such as NXP. From a technical perspective, this work allowed to work with unique design and simulation tools, often available only for the industry, which allowed the development of strong skills and the application of concepts learned during academic life to the design of real products that will reach a costumer. Additionally, and importantly, being able to work in an international environment with passionate engineers committed to their work was essential for the development of fundamental interpersonal skills for communication both with the internal team and with external technical support providers.

References

[1] NXP Semiconductors, "NXP History," July 2022. [Online]. Available: https://www.nxp.com/company/about-nxp/history:NXP-HISTORY. [Accessed 2022].

[2] NXP Semiconductors, "NXP in France," July 2022. [Online]. Available: https://www.nxp.com/company/about-nxp/worldwide-locations/france:FRANCE. [Accessed 2022].

[3] JTAG Technologies, "IEEE 1149.1," 2022a. [Online]. Available: https://www.jtag.com/ieee-1149-1/. [Accessed 2022].

[4] JTAG Technologies, "What is Boundary-Scan?," 2022b. [Online]. Available: https://www.jtag.com/boundary-scan/. [Accessed 2022].

[5] NXP Semiconductors, "DfT Reference Architecture for DfX," 2008. [Online]. Available: https://confluence.sw.nxp.com/spaces/viewspace.action?key=DfTfXRA. [Accessed 2022].

[6] NXP Semiconductors, Tass Reference Manual 8.0, 2009.

[7] SIEMENS, "Tessent Solutions," 2022. [Online]. Available: https://blogs.sw.siemens.com/tessent/. [Accessed 2022].

[8] SIEMENS, Tessent IJTAG User's Manual v2022.1, 2022.

[9] Texas Instruments, IEEE Std 1149.1 (JTAG) Testability, TI Semiconductor Group, 1997.

[10] N. Lainé, EOS - TSMC First C028 Secured NFC Controller, NXP Semiconductors, 2021.

[11] NXP Semiconductors, TS_CLN28ESF3_5X1Y2ZALRDL_NXP_IO Family — MFIOSF. Preliminary Datasheet, 2020.

[12] V. Chalendard, DFT EOS TAM Specification, NXP Semiconductors, 2019.

[13] NXP Semiconductors, DFT Reference Flow 9.13 User Manual: DFT Validation Steps, 2016.

[14] SIEMENS, Tessent Shell User's Manual v2022.1: The procedures, 2022.

[15] SIEMENS, Tessent TestKompress User's Manual v2022.1, 2022.

[16] A. Ghosh, "Internal Scan Chain-Structured Techniques in DFT," 2022. [Online]. Available: https://technobyte.org/internal-scan-chain-structured-dft-techniques/. [Accessed 2022].

[17] Mu-TEST, MuTool Training: The ATE Game Changer, 2019.

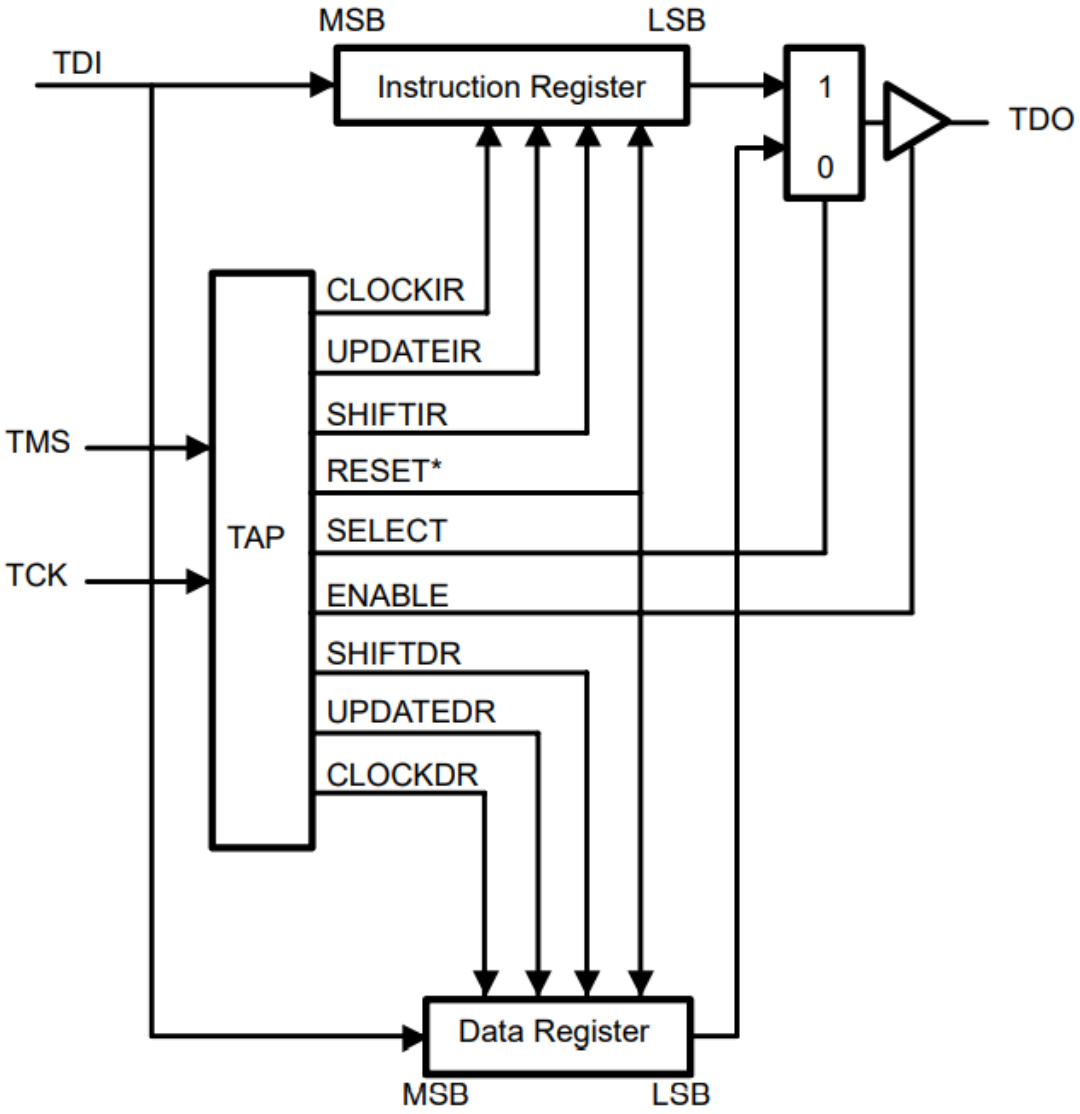
[18] E. Pal, "Understanding Shmoo Plots and Various Terminology of Testers," Design & Reuse, 2022. [Online]. Available: https://www.design-reuse.com/articles/47330/understanding-shmoo-plots-and-various-terminology-of-testers.html. [Accessed 2022].

[19] NXP Semiconductors, DFT Reference Flow 9.13 User Manual: General Flow, 2016.

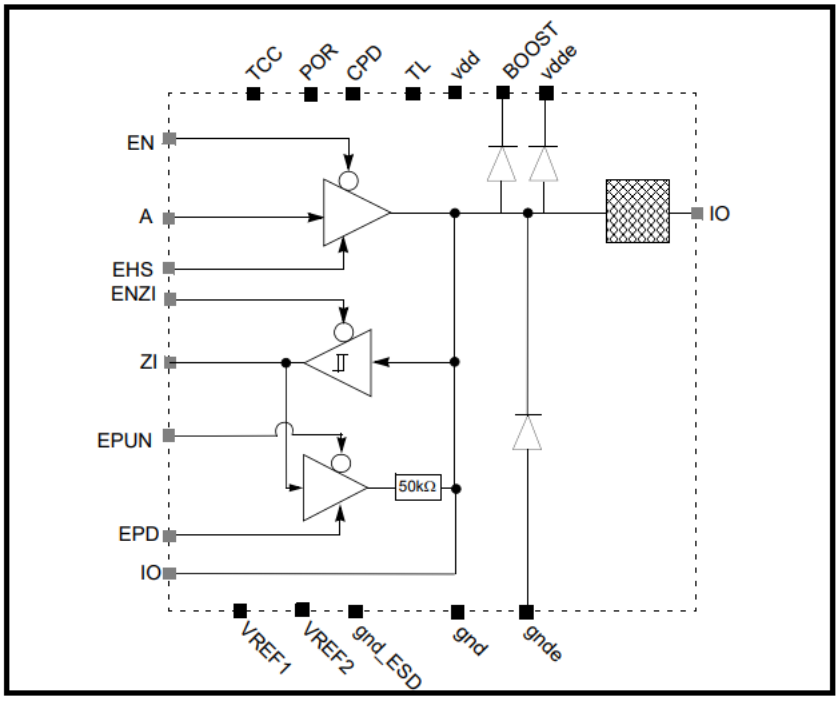
[20] SIEMENS, "Tessent TestKompress," 2022. [Online]. Available: https://eda.sw.siemens.com/en-US/ic/tessent/test/testkompress/. [Accessed 2022].

Appendix

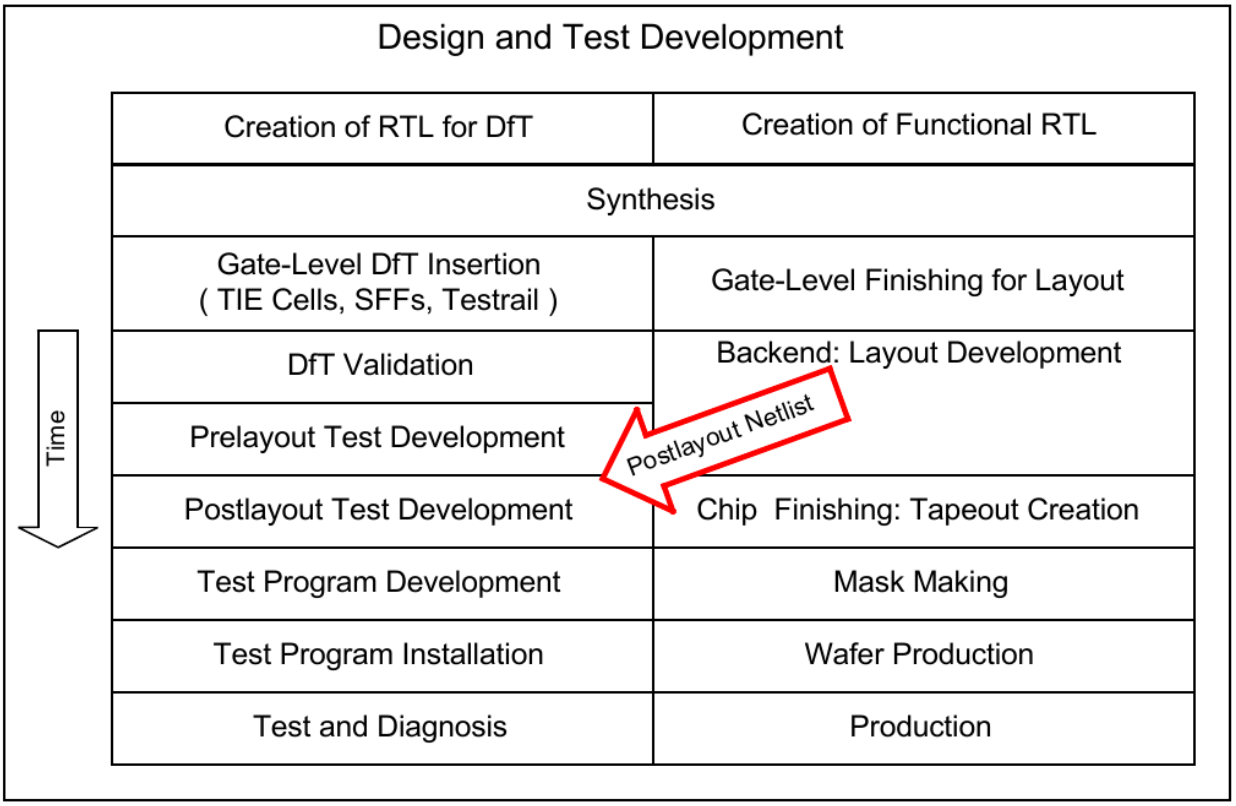
Additional Figures & Tables



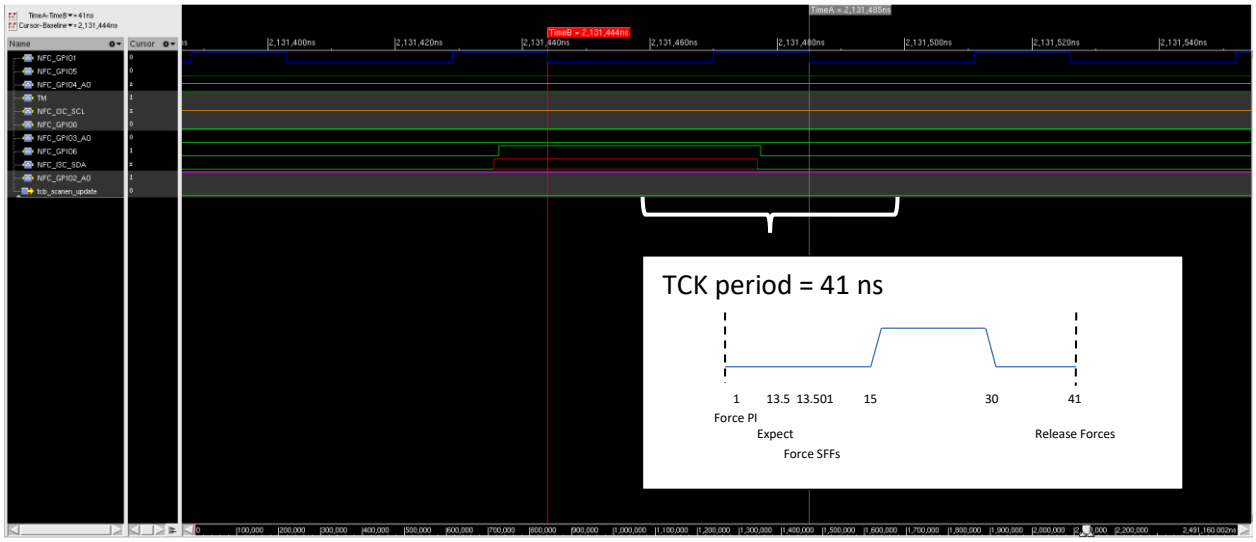
Appendix Figure 1: TAP Control Output Interconnect Diagram [9].



Appendix Figure 2: Functional Diagram of MFIO1V8SF bidirectional cell [11].



Appendix Figure 3: Chip Design Flow vs. DFT Flow Tasks [19].



Appendix Figure 4: Timing adjustment on shift period for parallel patterns.

Acronyms

Table 4: Acronyms Table.

Acronym	Description
ATE	Automatic Test Equipment
ATF	Application Test File
ATPG	Automatic Test Pattern Generation
BIST	Built In Self-Test
BSC	Boundary Scan Chain
CCB	Clock Control Block
DAM	Data Access Mechanism
DFT	Design for Test
DR	Data Register
DR	Data Register
DRC	Design Rule Check
EDA	Electronic Design Automation
EDT	Embedded Deterministic Test (Mentor Graphics test compression logic)
FSM	Finite State Machine
GPIO	General-Purpose Input/Output
HFO	High Frequency Oscillator
IC	Integrated Circuit
ICL	Instrument Connectivity Language
IP	Intellectual Property
IR	Instruction Register
LDO	Low-Dropout Regulator (DC-DC Regulator)
LFO	Low Frequency Oscillator
NFC	Near Field Connection
OVP	Overvoltage Protection Circuit
PCRM	Power Control and Reset Module
PDL	Procedural Description Language
PI	Primary Input

PMUANA	Power Management Unit (Analog)
PO	Primary Output
RTL	Register Transfer Level
SDM	System DFT Mode
SFF	Scan Flip-Flop
SI	Secondary Input
SO	Secondary Output
SPMI	System Power Management Interface
STIL	Standard Test Interface Language
TAM	Test Access Mechanism
TAP	Test Access Port
TASS	Test Pattern Assembler (NXP pattern generation tool)
TCB	Test Control Block
TCK	Test Clock
TD	Test Data File
TDI	Test Data Input
TDL	Test Description Language
TDO	Test Data Output
TDR	Test Data Register
TMER	Test mode Entry Request
TMS	Test Mode Select
TMUX	Test Multiplexer
TPF	Test Protocol File
TPR	Test Point Register
TRST	Test Reset
WGL	Waveform Generation Language