

Nuutti Mikkonen

# DFT IMPLEMENTATION FOR A LARGE SOC

Master's Thesis  
Faculty of Information Technology and Communication Sciences  
Examiners: Prof. Timo Hämäläinen  
Arto Oinonen, M. Sc.  
October 2023

# ABSTRACT

Nuutti Mikkonen: DFT Implementation for a Large SoC  
Master of Science Thesis  
Tampere University  
Master's Degree Programme in Electrical Engineering  
October 2023

---

Integrated circuit production is a complicated process with numerous possible points of failure. Designing the tests and test structures that verify that the chip is produced as planned is an important step in the physical design flow.

Among the commonly tested faults are static and dynamic faults in the standard cell logic and various fault models in memory core area. Besides these DFT is also used to detect faults in the PCBs connecting the chips.

This work plans and implements DFT structures for a large SoC as well as generates the tests required. The initial targets set for test coverage are partly met.

Keywords: DFT, Scan, Scan Compression, OPCG, MBIST, JTAG

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

# TIIVISTELMÄ

Nuutti Mikkonen: DFT toteutus suurelle moniprosessorijärjestelmäpiirille  
Diplomityö  
Tampereen yliopisto  
Sähkötekniikan Diplomi-insinöörin tutkinto-ohjelma  
Lokakuu 2023

---

Integroitujen piirien valmistaminen on monimutkainen prosessi useilla mahdollisilla epäonnistumisen kohdilla. Testien ja testirakenteiden suunnittelu piirin valmistusprosessin varmistamiseksi on tärkeä osa fyysistä suunnittelua.

Yleisesti testattujen vikojen joukossa on staattiset ja dynaamiset viat standardisoluissa sekä eri tyypiset vikamallit muistien ytimen alueella. Näiden lisäksi DFT:tä käytetään myös tunnistamaan vikoja piirejä yhdistävällä piirilevyllä.

Tässä työssä suunnitellaan ja toteutetaan DFT rakenteet suurelle moniprosessorijärjestelmäpiirille sekä generoidaan tarvittavat testit. Alkuperäisiin tavoitteisiin päästään osittain.

Avainsanat: DFT, skannaus, skannaus kompressio, OPGC, MBIST, JTAG

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck –ohjelmalla.

# **PREFACE**

This thesis was written for Nokia as part of the SoC-Hub project. I would like to thank everyone involved in the project, particularly those working in physical design, for making this thesis possible.

I would also like to express my gratitude to the examiners of this work, Timo Hämäläinen and Arto Oinonen, as well as my line manager at Nokia, Antti Hirvonen, for enabling this opportunity.

Tampere, October 13, 2023

Nuutti Mikkonen

# SISÄLLYSLUETTELO

1.INTRODUCTION .....	1
2.DESIGN FOR TEST.....	2
2.1    Fault types .....	3
2.1.1 Static.....	3
2.1.2 Bridge .....	4
2.1.3 Dynamic.....	4
2.1.4 Memory.....	5
2.1.5 PCB/Boundary .....	5
2.2    JTAG .....	6
2.3    Scan .....	12
2.3.1 Basic scan .....	12
2.3.2 Compression.....	16
2.3.3 Transition fault testing .....	18
2.3.4 Hierarchical test .....	20
2.3.5 IDDQ.....	23
2.4    MBIST .....	23
3.HEADSAIL .....	26
3.1    SysCtrlCPU.....	26
3.2    Subsystems .....	26
3.3    Physical design .....	27
3.4    DFT specification and requirements.....	29
4.HEADSAIL DFT IMPLEMENTATION.....	31
4.1    Boundary scan .....	31
4.2    Scan .....	32
4.3    MBIST.....	37
5.RESULTS .....	40
6.CONCLUSIONS.....	42
REFERENCES.....	43

# LIST OF SYMBOLS AND ABBREVIATIONS

AF	Address Decoder Fault
AI	Artificial Intelligence
ATE	Automatic Tester Equipment
ATPG	Automated Test Generation
AUXCKn	Auxiliary Clock Number n
AXI	Advanced eXtensible Interface
BIST	Built-In Self-Test
BSDL	Boundary Scan Description Language
C2C	Chip-to-Chip
CF	Coupling Fault
CFI	Cell Functional Input
CFid	Idempotent Coupling Fault
CFO	Cell Functional Output
CFst	State Coupling Fault
CODEC	Compressor-Decompressor
CPU	Central Processing Unit
CTI	Cell Test Input
CTO	Cell Test Output
DFT	Design For Test
DRF	Data Retention Fault
DUT	Device Under Test
FF	Flip-Flop
FV	Formal Verification
HDL	Hardware Description Language
HPC	High-Performance Computing
ICL	Instrument Connectivity Language
IDDQ	Quiescent Current
IEEE	Institute of Electrical and Electronics Engineers
IO	Input-Output
JTAG	An industry standard test interface
LBIST	Logic Built-In Self-Test
LEC	Logic Equivalence Checking
LOC	Launch Off Capture
LOS	Launch Off Shift
LSB	Least Significant Bit
LSSD	Level-Sensitive Scan Design
MBIST	Memory Built-In Self-Test
NVM	Non-Volatile Memory
OPCG	On Product Clock Generation
PCB	Printed Circuit Board
PDL	Procedure Description Language
PLL	Phase Locked Loop
PnR	Place and Route
RTL	Register Transfer Level
SAF	Stuck-At Fault
SC	Scan Clock
ScanDEF	Scan Design Exchange Format
SD	Scan Data
SEN	Shift Enable
SI	Scan In
SO	Scan Out
SoC	System-on-Chip

SOF	Stuck-Open Fault
SysCtrlCPU	System Control CPU
TAP	Test Access Port
TCK	Test Clock
TDI	Test Data in
TDO	Test Data out
TDR	Test Data Register
TI	Test Input
TMP	Test Mode Persistence
TMS	Test Mode Select
TO	Test Output
TRST	Test Reset
VHDL	Very High-Speed Integrated Circuit HDL
WBY	Wrapper Bypass
WIR	Wrapper Instruction Register
WPP	Wrapper Parallel Port
WRCK	Wrapper Serial Clock
WSI	Wrapper Serial Input
WSO	Wrapper Serial Output
WSP	Wrapper Serial Port

# 1. INTRODUCTION

As semiconductor node size decreases the manufacturing process gets harder to accomplish without defects. The number of possible faults is directly proportional to the number of standard cells in a design, and with designs ranging from millions to hundreds of millions of cells with a single fault potentially causing the chip to be inoperable, the risk of not testing chips before using them in products is not acceptable. The issue is that many faults are very laborious to detect if the design is synthesized from hardware description languages (HDL) that generally does not implement test structures in it.

Design for test (DFT) is the process of creating test structures and tests for a semiconductor circuit in order to be able to identify faulty dies after manufacturing. Test generation in DFT is automated, allowing designers to create tests for any typical digital design without needing to consider the testability when building the design.

The purpose of this thesis is to study how DFT can be utilized to improve testability of a chip and to implement said strategies to a design. The design used for this work is Headsail, a heterogeneous System-on-Chip with modern features designed as a collaboration of multiple entities. The most important results to analyze after implementation are test coverage of possible faults and the time it takes to apply these tests. Due to long lead-time for the physical chips, applying the tests is not a part of this work.

The thesis is structured as such: Chapter 2 describes the types of faults to be tested for and the structures needed to test for them. Headsail design and the flows used for it are presented in chapter 3, along with the targets and requirements of its DFT. The specific structures and strategies used for the implementation are described in chapter 4, with the results of it being analyzed in chapter 5. Chapter 6 concludes the thesis with a summary of the work.

## 2. DESIGN FOR TEST

As chip design processes have gotten smaller, smaller defects have started to cause observable faults in chips. On the other hand, chip area has been growing despite smaller process nodes as computing requirements are increasing. Due to these factors manufacturing a functioning circuit is ever harder and the manufacturer cannot guarantee that the chips are as they were designed. The responsibility of designing the test structures and tests falls on the chip designer.

Design For Test (DFT), sometimes Design For Testability or Design For Testing, is a methodology for designing a chip in such a way that it is possible to test its correctness after manufacturing to ensure that it is suitable for operation. On very simple chips such as the 7408 quad-AND-gate no special structures are needed as the chip is easily fully testable by probing only its IOs. In this case each AND gate can be fully tested with 3 patterns on its inputs (11, 01 and 10) and they can all be tested in parallel meaning only 3 patterns are required.

Any chips more complicated than this get increasingly hard to tests only with functional tests (without DFT insertion). For example, even on a simple processor testing would require software that applies practically all possible values on all wires and uses them in some way so that it would be possible to signal if something is not as expected. This would require not only an absurd amount of time to create and verify that everything is tested, but also the testing operation would likely be extremely long. [1]

Only standard cell based digital designs are considered here, but it is important to note that analog parts should have a test plan as well. Due to their analog nature however a pass/fail grade has to be carefully determined by the designer, whereas on digital side determining whether a part is functional is simple and the tools can easily create the test vectors.

The test vectors are created by software with a process called Automatic Test Pattern Generation (ATPG). As the structures are industry standard or IEEE standard with a long history the tools are quite capable of generating exhaustive yet minimal patterns to gain the maximum coverage. These patterns are then applied to the physical chip on an Automated Tester Equipment (ATE).

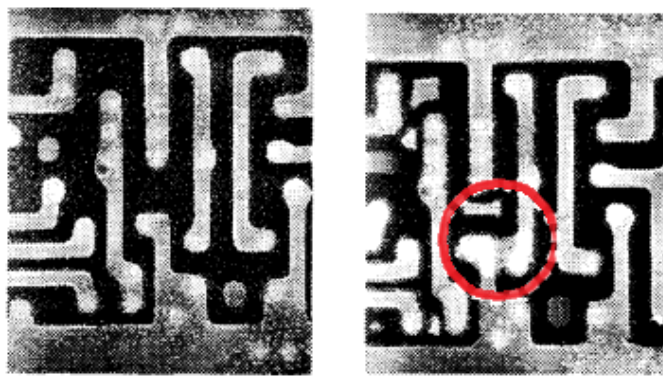
This chapter describes the various aspects needed to create a robust testability plan.

## 2.1 Fault types

The main purpose of DFT is to be able to detect possible faults created in device manufacturing. This chapter describes the faults that can be tested with DFT on a standard cell-based design as well as the ways that ATPG targets these faults.

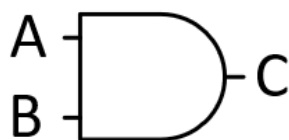
### 2.1.1 Static

Faults are modeled on the input and output pins of standard cells. A static fault is one where one of the pins of the cells is stuck at either 1 or 0. Physically this means the pin is shorted to either ground or power. [2] Figure 3 shows a stuck-at-fault detected with IDDQ testing [3].



**Figure 1.** Non-defective (left) and defective circuit as seen on a microscope [3]

The ATPG tool creates a fault model of the standard cells which has all the possible faults of it. These faults are then targeted so that if and only if the fault exists the observe point (output) of the cell will have a different value than what is expected. Figure 4 shows how an AND gate can be tested for all its faults. If the output C is not at the expected value some pin is faulty. In general, it is not useful to know which of the faults is active as the part will likely be rejected despite the type but by running all patterns the exact fault can often be pinpointed.



A	B	C	Fault detectable
1	1	1	A or B or C stuck at 0
1	0	0	B or C stuck at 1
0	1	0	A or C stuck at 1

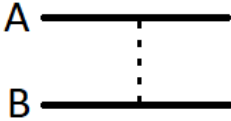
**Figure 2.** AND gate testable static faults

For more complicated gates (more than 2 inputs) the required input patterns might be more complicated than would be logically required, for example non-controlling inputs might still need to be driven to a specific value as the physical structure of the cell might cause specific faults to only be active with a seemingly redundant pattern. This is highly

specific to the used standard cell library and has to be taken into account when generating patterns. [2, 4]

### 2.1.2 Bridge

Bridge faults are a type of static fault where two adjacent nets are shorted together. The testing process is similar to other static faults except the control targets the chosen nets instead of cell pins. Testing for bridge faults requires a lot of effort from the designer on choosing which net pairs should be tested for faults. Figure 5 shows the required patterns to test a bridge fault between 2 nets. [2]



A	B	Without fault	With fault
1	0	A=1	A=0
0	1	A=0	A=1
1	0	B=0	B=1
0	1	B=1	B=0

**Figure 3.** Testable faults between 2 wires

Note that the inputs of some of the cases are redundant. The outputs however have to be measured on both nets to prove that no fault exists. Any single measured fault means a bridge fault exists between the nets. [2]

### 2.1.3 Dynamic

Dynamic faults are faults that cause issues in dynamic operation but are not visible as static faults. That means dynamic faults cause a slow transition on a net, but it will eventually reach the desired state. [5] These faults can be caused by a variety of reasons such as impurities in materials or over- or under-etching of metals. Inductance to nearby wires and high resistance shorts to power rails could also cause such faults. [4]

Delay defects cause a given transition to occur slower than expected. On an already slow path this could lead to a functional failure on the device, but on a path with excess slack the fault could initially be undetected. In some cases, they could lead to an early life failure of the chip, for example an increased resistance wire could degrade into an open through electromigration given enough time. [4, 5]

As there are a multitude of reasons leading to delay defects, so are there ways of modeling and characterizing them. Choosing the way that faults are modeled and tested for is a balance of computing resources, tester time and real-world accuracy. [6]

### 2.1.4 Memory

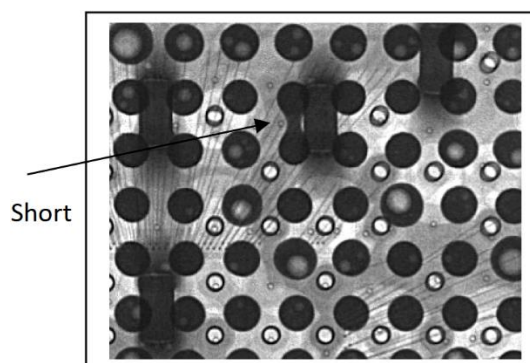
Miniaturization affects not only standard cells, but memories as well. On-chip memories are often specifically created from 6 transistor (6T) SRAM cells, the density of which is generally much higher than standard logic, meaning that faults are more likely to affect the functionality. In addition, faults may affect neighboring cells. This is why there are more fault models created for memories than standard logic.

Address decoder faults (AF) are faults affecting the address decoding logic of a memory, that is they are not faults in the core array. Faults affecting the core can most often be modeled as stuck-at faults (SAF), stuck-open faults (SOF), transition faults (TF), coupling faults (CF) or data retention faults (DRF). Coupling faults are further separated to state coupling faults (CFst), where the victim cell is forced to a value when the aggressor cell contains a specific value, and idempotent coupling fault (CFid), where the victim is forced to a value when the aggressor undergoes a transition. [7, 8]

As memories consume a large portion of space on modern chips and their fault density is higher, to mitigate yield issues caused by faulty memories, some memories provide additional mechanisms to allow for repairing them. This means extra rows, columns or banks are added to the memory which can be used instead of the faulty parts of the memory.

### 2.1.5 PCB/Boundary

Faults on a PCB may be open or shorted wires, caused by the PCB manufacturing. In most cases however, a fault is produced as a product of the soldering process. Figure 6 illustrates an x-ray filmed short of two pins. [9, 10]



**Figure 4.** X-ray image of shorted pins on an BGA package [10]

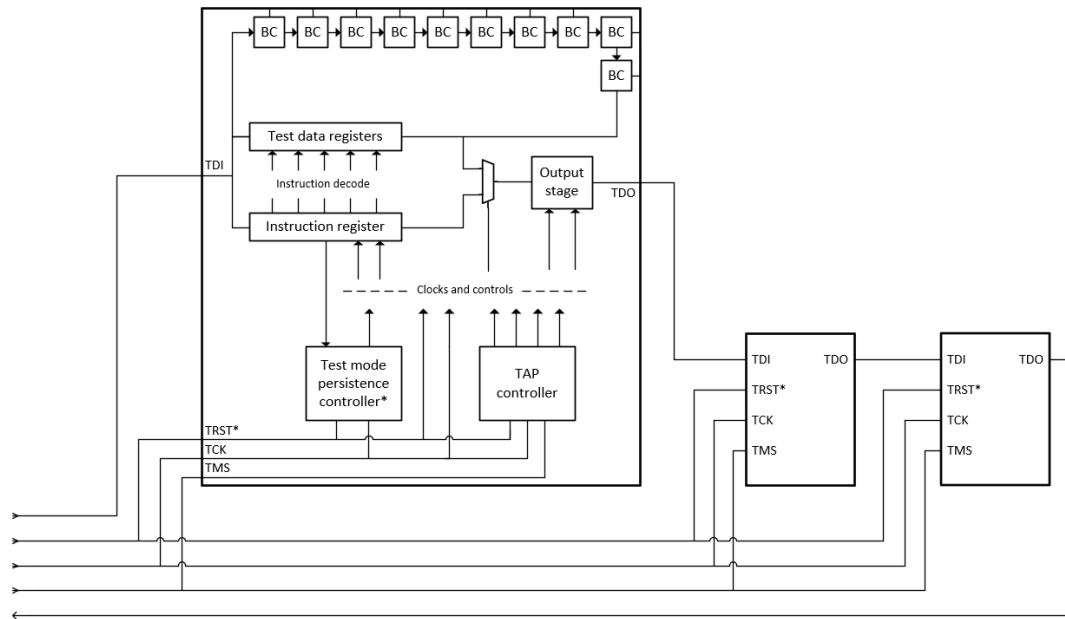
## 2.2 JTAG

PCBs can be verified for faults with mechanical In-Circuit-Testing (ICT) by probing the board. Large number of layers and Ball Grid Array (BGA) packages make this testing harder to apply. Instead, optical or x-ray inspection may be utilized, however either option requires time on specific testers which increases manufacturing costs. This chapter describes ways of testing the PCB without requiring expensive testers. [9, 11]

IEEE1149.1 is an IEEE (Institute of Electrical and Electronics Engineers) standard that defines architectures and languages used to allow testing of PCBs, system interconnects and the functionality of the chip itself. The standard is also known as the JTAG standard, named after the initial group designing the standard, the Joint Test Action Group. [12]

The access to boundary scan is done through a Test Access Port (TAP). It is defined as four mandatory ports and one optional one. The ports are test clock (TCK), test mode select (TMS), test data in (TDI) and test data out (TDO), and the optional test reset (TRST). All operations through the TAP are synchronous to the TCK except for the TRST, which is an asynchronous active-low reset to set the control logic to a known state. The input ports TDI and TMS are sampled on the rising edge of TCK and the output port TDO is set on the falling edge of it. [12]

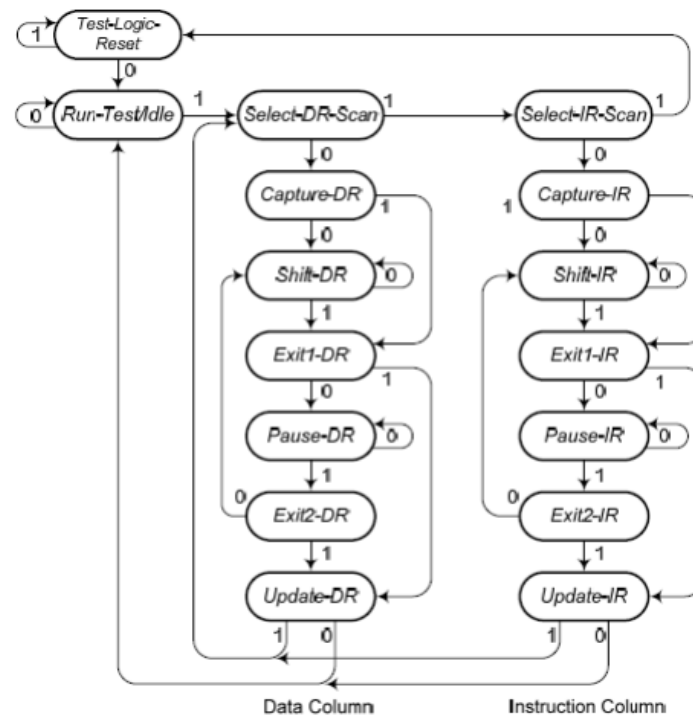
Multiple devices with JTAG can be connected in series to form a longer JTAG chain. This is done by connecting the TDI and TDO pins to create a long register between them and sharing the other signals between the devices. This way only one set of pins is needed to control all the devices and ideally test the entire PCB. [12] Figure 7 illustrates a chain of devices with internal structure of one. The left side is connected to the host driving the interface.



**Figure 5.** JTAG chain of three devices. Parts with \* optional

The logic connected to the TAP is called TAP controller. It implements a state machine as shown in figure 8 to control the test circuitry within the chip. The state of the machine is determined by the state of the TMS when sampled. The machine is reset either by holding the TRST low or by holding the TMS high for five cycles of TCK. [12]

The TAP controller is used to load data to the instruction register (IR) and test data registers (TDR). The instruction register contains the current active instruction operation code and the test data registers contain any data used for testing purposes. There are 2 mandatory data registers in bypass and boundary and there may be any number of optional standard conforming or user specified registers. [12]



**Figure 6.** JTAG state machine. Numbers represent value of TMS required for transition [12]

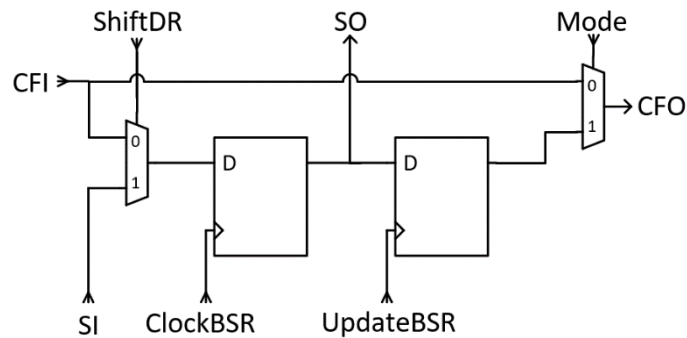
When operating on any registers, be it instruction or test data, the register is placed serially between the TDI and TDO pins serially, so that the least significant bit (LSB) is closest to the output. The data is shifted in on the rising edge of TCK and out on the falling edge. This way data can be moved both in and out of the chip in the same operation. [12]

The instruction register is loaded and updated in the Shift-IR and Update-IR instruction respectively. The capture-IR will additionally load at least a 1 to the LSB and a 0 to LSB+1. The purpose of this is fault isolation, so that it is always possible to shift out from the instruction register with the data containing both logic values. Other bits in this state can be set to user defined values. [12]

Test data registers are controlled similarly to the instruction register, shifting the value in Shift-DR and updating in Update-DR. The register to operate on is selected with the instruction. Capture-DR may capture a value internally in parallel to the register so that data can be shifted out of the chip with this state. [12]

Bypass is one of the mandatory data registers. It is a single register with the purpose of providing the shortest length through the data path. It is activated with an opcode of all ones as well as with any opcode that is not used for other instructions. The other required register is the boundary-scan register. It is constructed of boundary cells which are used to test connectivity of the chip on the PCB. [12]

The purpose of a boundary cell is to be able to control an output and to read an input pin of a chip without needing the functional logic to do this operation. The boundary cell is inserted between the core side input or output pin of a pad cell and the functional logic that it in normal operation drives. This way control to the pad can be bypassed and driven directly from the JTAG TAP. Figure 9 illustrates a possible implementation of a boundary cell. [12]



**Figure 7.** Example boundary-cell register design

If the cell in figure 9 is used as an output boundary cell, the cell functional input (CFI) would be connected to what would normally drive the output pin of the pad and the cell functional output (CFO) to the pad pin. The ShiftDR, Mode, ClockBSR and UpdateBSR are controlled from the TAP controller so that in normal operation the boundary cell is transparent and in test modes the value from CFI can be captured and a value can be driven directly to the CFO. Scan in (SI) and scan out (SO) connect to other boundary scan cells to form a shift register which is loaded in the Shift-DR state and captured in Capture-DR. Note that the standard defines various types of boundary cells used for different types of IOs. They may also for example drive the output enable pin of a pad to allow setting the IO in tristate. [12]

There are 3 mandatory instructions for boundary scan. Their opcodes can be chosen freely, except that an opcode of all ones is reserved for bypass. With the SAMPLE instruction values from input IOs and from chip logic outputs are loaded into the boundary register in the Capture-DR state through the CFI input in figure 9. This way the pins of the chip can be observed without affecting operation or needing physical probes. The PRELOAD instruction loads values to the boundary cells to be used for other instructions. The CFO is still fed directly from CFI and so normal operation is not affected. SAMPLE and PRELOAD can be combined to a single opcode, so that with a single instruction the inputs can be read, and values can be loaded in the registers. The last mandatory instruction EXTEST allows testing of chip connectivity and PCB. It sets the output of boundary cells to the values in the boundary registers, typically set with PRELOAD. [12]

Besides the mandatory instructions, the standard defines various others which will be briefly covered here. `INTEST`, `IC_RESET` and `RUNBIST` allow for testing of internal logic on the chip. `HIGHZ` and `CLAMP` extend the capabilities of testing external interconnects. `IDCODE`, `ECIDCODE` and `USERCODE` provide device identification means in case multiple instruments are included in a chain. `INIT_SETUP`, `INIT_SETUP_CLAMP` and `INIT_RUN` are newer additions that enable initialization of programmable components, for example pad cells, before application of test instructions. The user may define any additional instructions required for use-case based testing as long as they don't interfere with the standard instructions. They may be defined as private, which tells the user that the instructions are used for manufacturing tests and may be undocumented and potentially harmful to the chip. [12]

To describe the interface and structure of JTAG in a chip, the standard defines a language called Boundary Scan Description Language (BSDL). Based on VHDL, the language describes the IO structure of the chip and how it is connected to the boundary register. It also lists the opcodes of all instructions and all other test data registers and how they can be accessed. [12]

The 2013 revision of IEEE1149.1 adds an optional component besides the TAP controller in the form of a test mode persistence (TMP) controller. It allows keeping the chip in test mode state, that is the state in which outputs of the boundary scan cells are driven from the boundary register, even when instructions that normally would not cause this behavior are active. This ensures safe use of on-chip test capabilities when the chip is located on a board by preventing the IOs from being driven from possible uninitialized states, for example when loading new values with `PRELOAD`. Three new instructions are required if the TMP controller implemented; `CLAMP_HOLD` and `CLAMP_RELEASE` to set the TMP controller in and out of the persistence state, and `TMP_STATUS` to read the status of the controller. [12]

With faster communications becoming more common, the static testing provided by standard boundary scan has become inadequate. IEEE1149.1 is complemented with IEEE1149.6, which standardizes features for testing ac-coupled and differential digital IOs with the help of new instructions and boundary cells. It provides simple and minimally intrusive structures compliant with the original standard to allow testing of dynamic circuits. [13]

As chips have become more integrated with multiple chips in a single package and multiple debug interfaces in a chip, more non-standard IEEE1149.1-like interfaces have become common. To combat the issues caused by this, such as difficulty in intellectual

property (IP) integration and interoperability problems with tools, IEEE1149.7 was created to provide and standardize more advanced debugging structures. It enhances the architecture of JTAG by providing a way to use a hierarchy of on-chip TAP controllers, a method of power control and selection protocols so that other technologies may use 1149.7 TAPs. Pin efficiency is improved by supporting four and two pin TAPs with both a star and series topology. Lastly it increases capability by defining test and functional reset generation on chip as well as optimized scan transactions for test and debug [14]

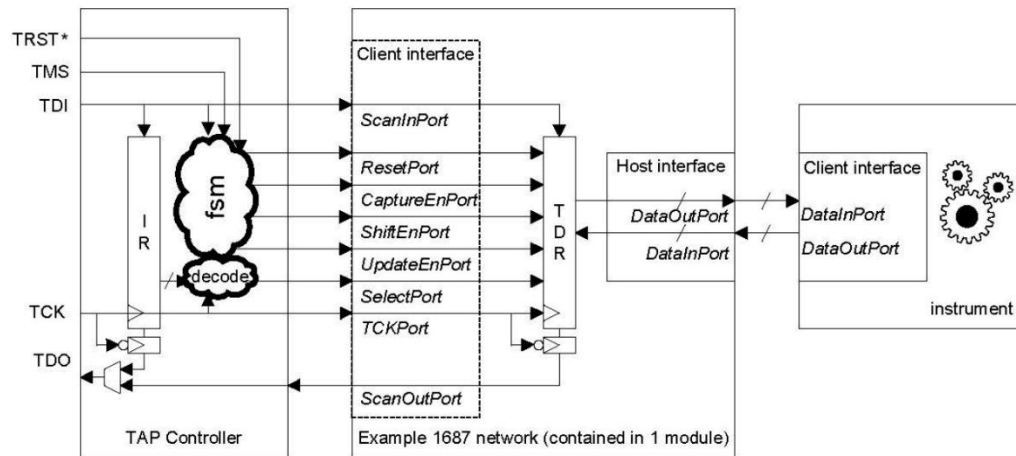
A widespread use of embedded instrumentation, such as built-in self-test (BIST) engines, device characterization and calibration controllers have emerged in chips. These instruments use a variety of mechanisms and protocols for access. IEEE1687 aims in standardizing the access between the instruments and chip-level access mechanisms. This is done by defining three things in the standard:

1. The hardware architecture for connecting a IEEE1149.1 TAP to embedded instruments
2. A hardware description language (HDL) used to document the components
3. A procedure description language to describe the operation of the instruments

The architecture described in the standard is quite unrestrictive to allow for various types of interfaces to be connected within the network. It particularly mentions examples of using a JTAG TAP and interfaces compliant with IEEE1500 as these are common interfaces used in industry, but other types may be used in the network as well when suitable adapters are used. [15]

The instrument access network is described with Instrument Connectivity Language (ICL) as defined in the standard. It documents the elements comprising the network and the logical connectivity between them. The hierarchical structure is contained in the language, but the operation of instruments and actual physical connections may be omitted, as the purpose is to merely describe the network. [15]

The Standard also defines the Procedure Description Language (PDL). It is used to define a way of operating the instrument defined in an ICL file through its interface ports. PDL consists of sequential statements that read or write to ports or registers within the instrument. The purpose of ICL and PDL are that when a hierarchical block is included in a larger design, the test patterns of the block can be retargeted to the top level. That is, even if the interface used is proprietary, with the concepts of IEEE1687 the instruments can be controlled with any interface from the top level, commonly with JTAG TAP. Figure 10 illustrates a possible configuration where an instrument is connected to a TAP controller through a 1687 network. [15]



**Figure 8.** Operation of an embedded instrument with IEEE1687 [15]

The instrument may be an IP that the designer does not have full visibility of. Despite this, if it is provided with adequate PDL and ICL files the testing may be done without knowing the exact functionality of the block.

Boundary scan can be used for testing multi-die package interconnects as well. IEEE1838 defines ways to test 2.5D and 3D chips with the help of JTAG. [16]

## 2.3 Scan

The main reason for requiring specific structures for testing purposes is that by default, a design will have very low controllability (1) and observability (2). This means that it is difficult to set the state of the circuit such that fault can be excited (1) and that if it is possible to activate the possible fault, it is hard to monitor whether the fault exists (2). Without scan the only observe and control points are generally the primary inputs and outputs (physical pins) of the chip. The purpose of scan is to extend these points to internal nodes of the chip. [17]

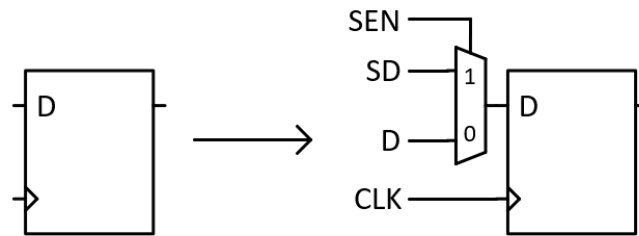
### 2.3.1 Basic scan

Scan is likely the most common structure used for testing on-chip logic and has been in use since at least 1978 [18]. The purpose of scan is to be able to control internal nodes such that specific patterns can be applied to the device under test (DUT) to find possible static, dynamic, bridge and possible other types of faults on the chip. [17]

The requirements for a chip to be compatible with scan is that it consists of standard cells and is of a traditional structure composed of combinational logic and flip-flops. The following types of circuits generally cannot be tested with scan.

- Asynchronous designs
- Latch-based designs
- Memories
- Analog blocks

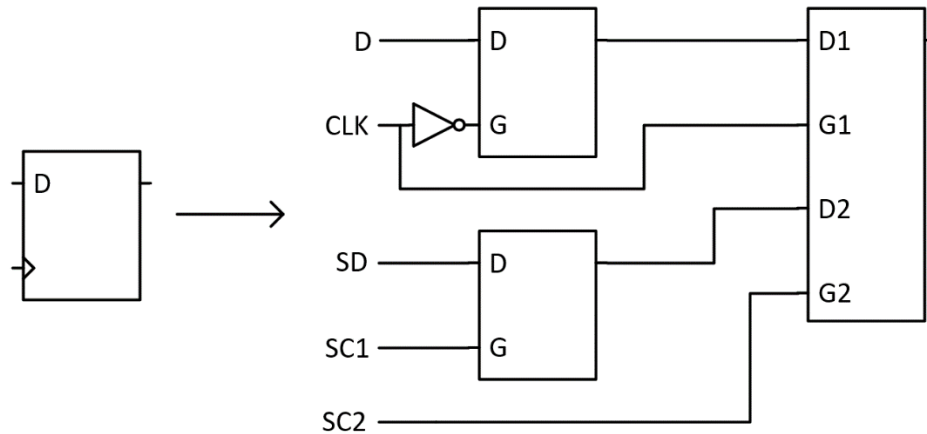
The goal of scan is to connect all flops of the design to scan chains that can easily be controlled from the primary inputs of the chip. To do this all the flops are replaced with scan equivalent cells. There are 2 different types of scan flops, the muxed scan and level sensitive-scan design (LSSD) flops. Figure 11 shows the conversion to muxed scan flop and figure 12 the conversion to LSSD scan flop. Scan flop mapping may be done at any point in synthesis, either separately or as part of general mapping. [4]



**Figure 9.** Muxed scan flop replacement

In muxed style scan a flop is replaced with one that has a mux added on the data input. The select of this mux is driven by a signal called shift enable (SEN). While the shift enable signal is at its inactive value, typically at 0, the flop acts as it did originally. When the shift enable is active, however, the flop gets its data from an extra input called scan data (SD) or scan in (SI). [4, 17]

The mux can be added as a separate component, but typically a standard cell library has scan equivalent cells for all or most of its flops, so that the number of instances does not change in this scan mapping process. A scan cell may as well have different clock polarities and set/reset inputs.



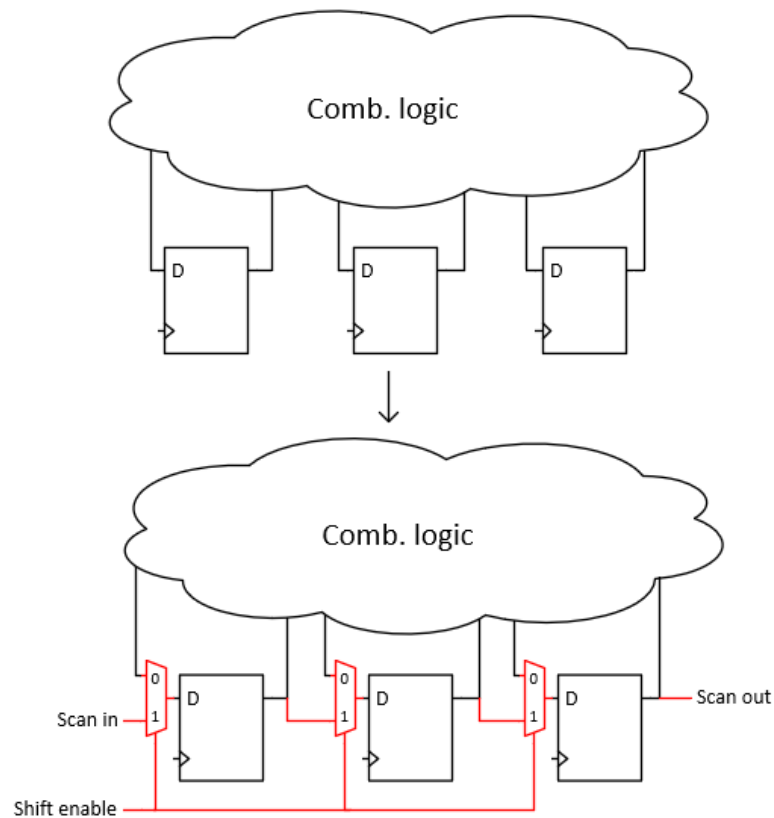
**Figure 10.** LSSD scan flop replacement

In LSSD style scan a flop is replaced with 2 single port latches and one dual port latch as shown in figure 12. One scan data input and 2 scan clock inputs (SCx) are added to create the same functionality as in muxed style. In normal operation both scan clocks are in their inactive states in which case the flop acts as before scan replacement. In scan operation the functional clock is held in its inactive state, which is the state that does not let the dual port latch propagate its value from the functional data port, in the case of figure 12 a logic 0. To move the data from the scan input to output the scan clocks are pulsed one at a time, SC1 first. [4]

Like with muxed style, LSSD flops are generally included in the standard cell library as replacements for non-scan flops and may include support for inverted clocks and set/reset functions. It is important to note that depending on the library either of the ports of the dual port latch could have priority and one should be careful not to use both the functional clock and the scan clocks at the same time.

LSSD style scan will not be elaborated further in this work, and all mentions of scan are assumed to be of muxed style unless otherwise noted.

After flops have been mapped to scan equivalent ones, they can be connected to scan chains. This is done by connecting the output of a flop to a scan in of another flop, and so creating one or multiple long chains where the signal proceeds through the entire chip. On the ends a physical scan in and out ports are connected to create one or multiple shift registers containing ideally all of the flops in the design. The output of flops will still feed into the functional logic along the newly created scan in connection, and functional data inputs of any flops are not touched at all, so the original functionality does not change while all relevant test signals are held at their inactive values. Figure 13 shows the scan chain stitching process. [17]



**Figure 11.** Scan mapping and stitching

The clocks of all flops should be controllable during scan. They may have separate clocks but working with a single clock is generally easier and possible due to low frequencies compared to functional modes. Either way designers need to make sure that all clocks can propagate to flops, possibly requiring some extra control over clock gates and other logic on the clock path. Similarly resets should be controllable at least to their inactive states during scan, but usually to both states in order to test the reset logic.

Once the scan synthesis is done the design is ready for ATPG for test generation. The scan process is as follows: At the start operation shift enable is set to its active value. After that data is shifted in from the chip's scan in pin or pins while pulsing the scan clock or clocks, shifting the data further through the scan chains. Once all flops have received their respective values, that is the clocks have been pulsed once for each flop in the chain the actual testing of logic can be done. [17]

Shift enable is set to its inactive value and as it is typically not a timed signal, some extra time is given for it to propagate. Then a single pulse is generated on the clocks. This way all of the flops will latch to data which goes through the functional logic instead of the scan chain. As it is possible to set any values on any flops with the scan shift operation, it is easy to control which parts of the logic the data will activate and so possible to target specific faults. [17]

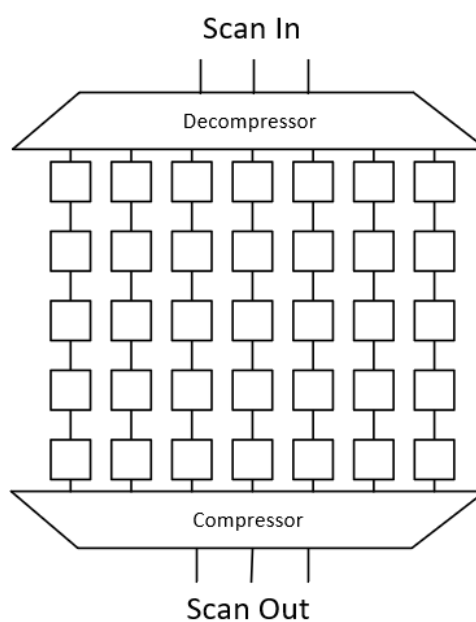
After the scan capture the shift enable signal is set to its active value again and the data that was captured can be shifted through the scan chain without any logic in between, and as such unaltered. During the shift out the ATE will read the received data and compare it to what is expected, that is to what a correctly functioning chip would produce. [17]

### 2.3.2 Compression

As the size of chips increases, so does the number of flops in a design. This leads to longer scan chains, which can to some extent be mitigated by using more chains in parallel. Eventually, however the required number of physical pins grows too large, and either the package does not have what is needed, or the tester cannot contact every pin at once.

To overcome this issue scan compression has been displacing standard compression techniques. It addresses the problem of increasing test data volume and tester time while keeping the test coverage at similar levels. [19, 20]

The purpose of compression is to bring the scan in and scan out points inside the chip, so that the data can be generated internally from fewer inputs than there are scan chains. The way the data is generated for the scan chains is called compression-decompression (CODEC). It can be done either combinational or sequentially where industry adoption has shown that sequential CODECs are slightly better. [21, 22] Figure 14 shows a scan network connected to a CODEC which performs a 7:3 compression, commonly referred as 2.3X.



**Figure 12.** Compressor-decompressor scan network

The actual CODECs are usually created of XOR gates [21, 23]. The purpose of the decompressor is to spread the inputs so that each scan chain can have either 0 or 1 input with as little dependency as possible with respect to the other scan chains, that is so that there are as many as possible different combinations on the inputs of the other scan chains. The compressor works in the opposite way and a good compressor generates the greatest number of distinct outputs based on the inputs.

Naturally with combinational logic either part of the CODEC can only create as many patterns as are possible given the number of scan IOs. For example, a 3:7 decompressor can only create  $2^3$  combinations on its outputs, while to create any arbitrary bit-pattern  $2^7$  combinations are needed. This means that using compression scan will always yield lower coverage since not all patterns can be applied to the DUT.

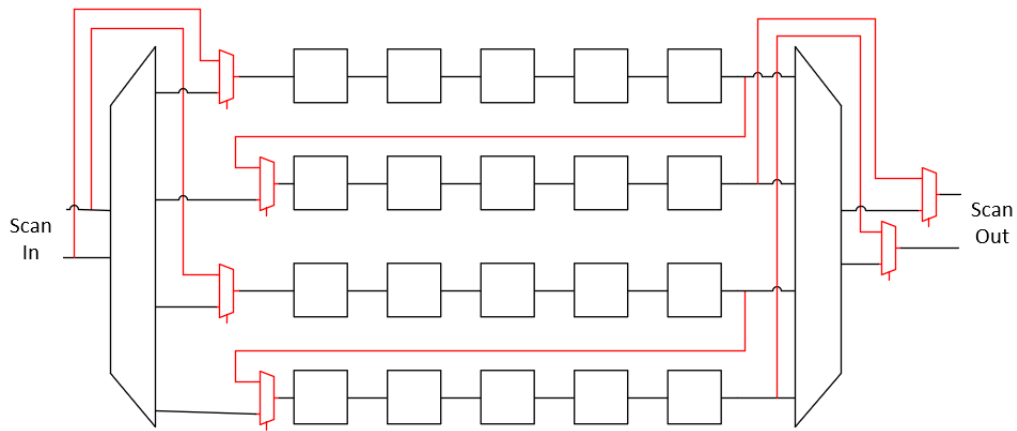
This problem can be mitigated by adding configurability to the CODEC, either so that the combinational logic can be rearranged to form different states or by adding registers so that the decompressor and compressor can use these registers as virtual Scan IOs. The data to these registers is loaded over multiple clock cycles, so using this type of scheme will increase tester time. [22]

The ratio of compression has to be carefully chosen so that test coverage does not get affected too much. [22] achieves a static fault coverage of over 99% with compression ratios of 50X and 230X. Generally, the larger the number of IOs the higher can the coverage ratio be pushed. The downside of higher compression ratio might be increased amount of test patterns as fewer faults can be tested in a single pattern. [22]

Another reason for lowered coverage besides the difficulty of generating required bit-patterns is X-sources in the logic. X-source in this context means logic whose output value cannot be determined. This could be memories or analog macros. If a flop in a scan chain captures an X-value the output of the compression macro for that clock cycle will be partly or entirely corrupted, thus losing actual measure data from the chains that did not capture an X.

The way to combat X propagation is by adding mask logic on the ends of the scan chains, before the compression macro. This mask logic can then set the X value to a known value so that the data on other scan chains does not get affected by the X. The mask can have either a fixed value (0 or 1) or it can be programmed. The mask will get only get activated for a single clock cycle at a time so typically a separate physical pin needed for its enable. [24]

A way of acknowledging the issue of lowered test coverage is by including a non-compression scan along the compressed design. Figure 15 shows how a 2X compressor can be bypassed in order to control the chains directly from IOs.



**Figure 13.** A way to bypass compression logic

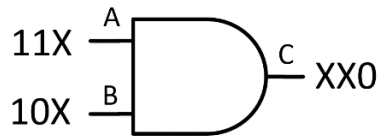
The muxes and paths shown in red can be added to allow the design to be tested without compression logic. The select of the muxes has to be controlled with a separate test signal, either from a physical pin or generated internally.

The advantage of including the bypass logic is that if there are faults that cannot be tested with compression the main compression patterns can be topped off with standard scan vectors. In case the CODECs are incorrectly designed or faulty they can also be bypassed to test the chip fully. The downside is that some extra logic and routing is needed, however this is very minimal.

### 2.3.3 Transition fault testing

The method for testing dynamic faults is the same as that for static faults, that is scan with the possibility of compression. Instead of a single pulse for capture however, for dynamic tests multiple pulses have to be administered to receive a dynamic response from the logic. [6]

The first of these pulses is called the launch and the second is the capture pulse. These pulses are timed such that the difference between them is the period of the frequency that is used to test the chip, typically the functional operation frequency. The initial value of the circuit is set with scan so that the targeted fault can be activated with the launch pulse and the response of that is then captured on the second pulse, and if the logic was dynamically faulty the response will not correspond to what was expected. Figure 16 shows how an AND gate would be tested for slow-to-fall faults on its input. [6]

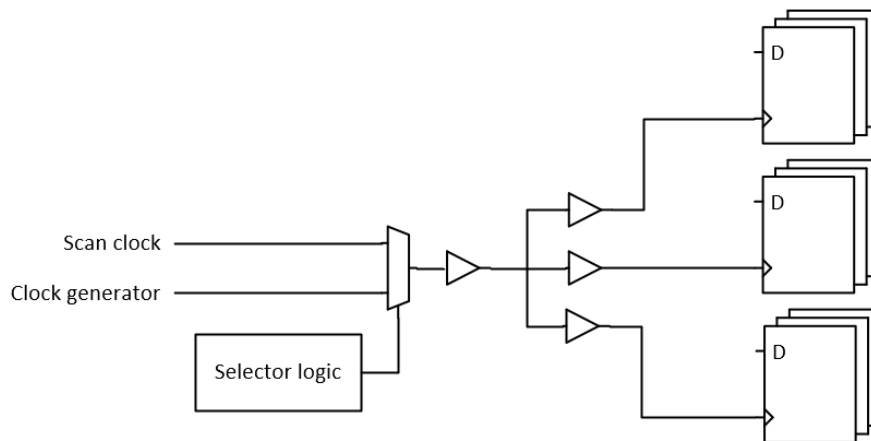


**Figure 14.** Testing the B-input of an AND gate for transition faults

The first value on the inputs shows the values loaded with scan which do not expose the fault. The second value is what is set on the launch pulse and the last value is what should be measured on the capture pulse. In this case, if the B-input is too slow to fall the value captured on the output C will be 1 and the fault will be found.

The way the pulses are generated can be from either a primary input or with On-Product Clock Generation (OPCG). It is hard to get a fast input clock from a pin so typically it is easiest to generate the clock for test internally with OPCG. With this method an internal already existing or new clock generator is used to generate the required fast clock pulses for the capture phase.

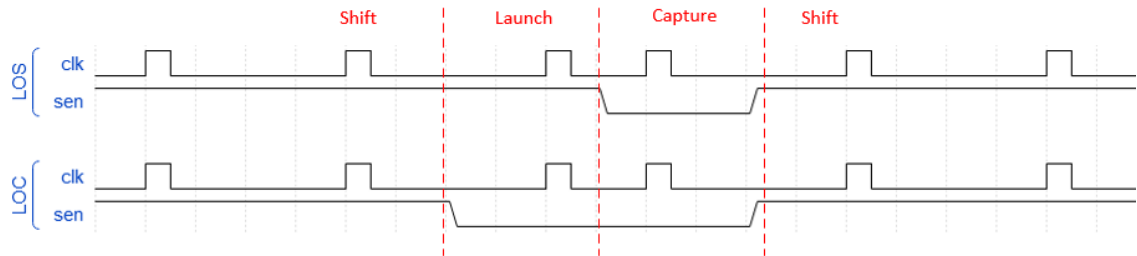
Scan shift is still done with a slower clock, so it does not need to be designed any differently from static scan. In fact, the only difference OPCG adds is the need for some clock selection logic at the beginning of a clock tree, which is common for the entire design and so should not affect timing in functional modes either. Figure 17 introduces an abstracted clock design for OPCG scan.



**Figure 15.** OPCG selection logic

If the same clock generator can be used for OPCG as is used for functional modes, only a single mux may be added on the clock path minimizing the impact on functional operation and timing closure. The selector logic will pass 2 clock cycles from the clock generator when so instructed by some test signal, typically controlled from a primary input, or when the chip is in a functional mode.

One important aspect of designing transition fault testing is choosing how the data for the launch cycle is generated. There are 2 main methods in Launch-off-shift (LOS) and Launch-off-capture (LOC). The difference in these is whether the shift enable signal is deasserted before or after the launch clock pulse. Figure 18 shows the timing of these methods. [6, 25, 26]



**Figure 16.** Timing of shift enable in LOS and LOC schemes

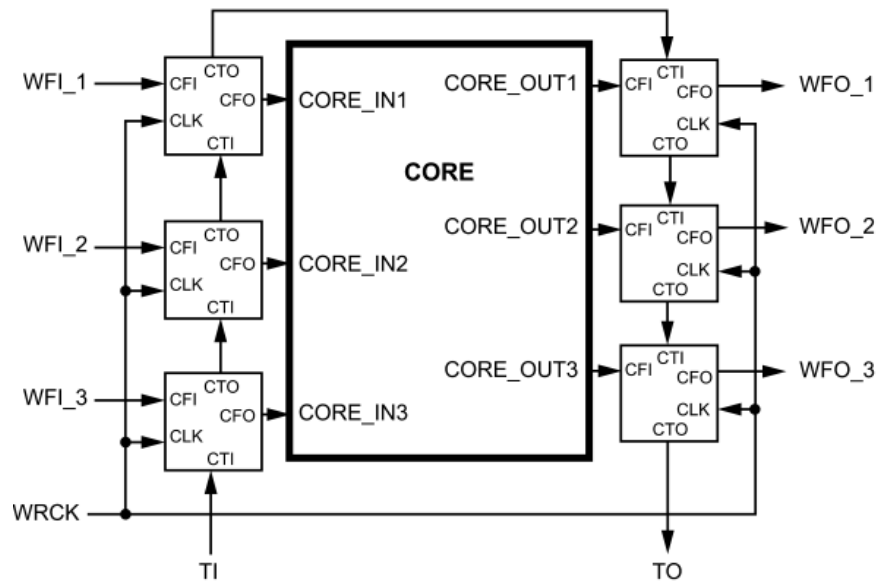
Either style has its advantages and disadvantages. In LOC style the values of flops for the launch cycle are captured through functional logic as the shift enable is not active and with this style it might not be possible to create all the possible needed patterns internally required to test all faults. In other words, LOC will have a lower coverage than LOS. The disadvantage of LOS is however that the shift enable signal has to be timed in such a way that its deassertion is propagated to the entire design between the fast clock cycles. In large designs with fast clocks this could prove challenging to implement. [3, 15, 16]

### 2.3.4 Hierarchical test

As chip size keeps increasing generating test vectors for it takes more time. Physical design for chips is often split into parts so that the entire chip does not have to be implemented at the same time. This helps parallelize the processing and reduces iteration time, allowing faster time to market. DFT structure insertion is typically done in synthesis so it would already gain benefits from hierarchical design of chips. ATPG without special structures however would have to be run on a flat chip top, increasing runtime. IEEE1500 standard addresses this issue by defining specific interfaces to be inserted on a separately synthesized block to allow ATPG to be run on a subblock and then combined on top level for the full patterns. [27, 28]

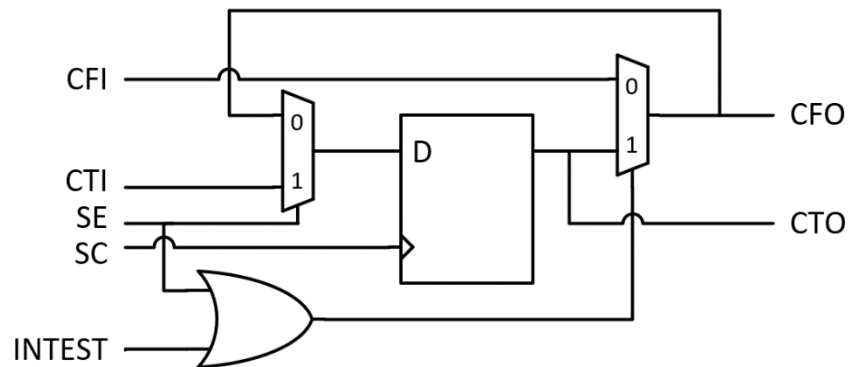
The way this is done is by isolating the blocks, or cores, so that all the interfaces between hierarchies can be tested separately from the internal logic, allowing the internal tests to be created targeting only the core and testing the interface along top level. Figure 19

illustrates how the interface is isolated inside the core by utilizing the Wrapper boundary register (WBR).



**Figure 17.** Core with WBR inserted [27]

The purpose of the WBR is to be able to set the inputs of the core and to observe the outputs of it independently of what they would be connected to without WBR. The wrapper cells connect serially to each other through the cell test input (CTI) and cell test output (CTO) pins and the internal values of the cells can be easily controlled and observed on the module level test input (TI) and test output (TO) pins. With the help of some internal test signals discussed further in this chapter, the internal values of the cells can be set through the CFI pin or the internal value can be set to drive the CFO output of the cells. In functional operation the wrapper cells will pass the value from the functional input CFI to output CFO. [27] Figure 20 illustrates a possible implementation of a wrapper cell to be used on the input side of the core.

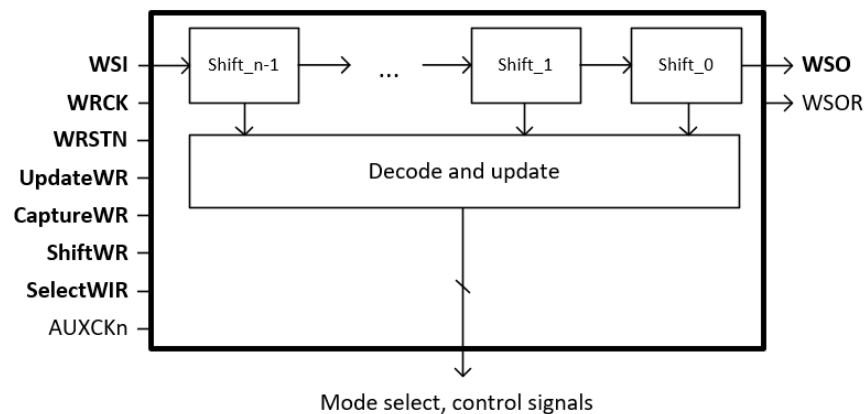


**Figure 18.** Input bounding wrapper cell implementation

When testing the core, the INTEST signal will be high, which ensures the output of the cell, connected to an input of the core, will receive its value from the internal register and not from the CFI input, that is driven with logic outside the core. This way a known value is always driven to the core, allowing testing of logic fed directly from the inputs. The internal register is loaded during scan operation, when SE is high, so the value driven can be changed for every pattern. An output bounding wrapper cell works similarly, with the purpose of capturing a value of an output port in the internal register and then shifting it out during scan. The boundary cell is clocked with the same clock that is used for scan. [27]

When the top level of the chip is tested, the cores will be set to EXTEST mode. In this mode the values to the input bounding cells will be captured from the functional inputs of the cells, that is across hierarchy boundaries. The output bounding cells will conversely drive their functional outputs from the internal registers. This way the hierarchical boundaries can be tested separately from the core. In both INTEST and EXTEST modes scan data of the boundary register is shifted serially through the TI and TO pins. [27]

The 1500 standard also defines a means to generate test signals at core level as Wrapper Instruction Register (WIR) and the method to control it in Wrapper Serial Port (WSP). This is useful in completely isolating the testing requirements inside the core so that top level does not have to generate the test signals for all possible instantiations of the core separately. Figure 21 shows the structure of WIR, with mandatory ports **bolded**. [27]



**Figure 19.** Structure of WIR

The WIR is controlled serially through the wrapper serial input (WSI) and wrapper serial output (WSO) ports. It is clocked through the wrapper clock (WRCK) port, which may be the same clock that is used for scan. WRSTN is an active low reset to reset the logic. The UpdateWR, ShiftWR and CaptureWR signals control the flow of the data and updating the values of the signals. SelectWIR is used to activate or disable a WIR. A WIR implementation may operate with multiple clocks, in which case the AUXCKn ports may

be used for this purpose. If the WSO is clocked on the falling edge of a clock, it is possible to include an WSOR terminal which clocks the output on rising edge. [27]

The signals that control whether the core is in INTEST or EXTEST must be generated in the WIR. The third mandatory instruction is BYPASS, which places a single register, Wrapper Bypass (WBY), in the path between WSI and WSO(R). This instruction is active when the wrapper is not used, for example during functional mode. Besides them, any number of other test signals, controlling for example compression or OPCG can be included in the WIR. [27]

The standard also defines that a Wrapper Parallel Port (WPP) may be included in a wrapped design. This port can be used for any purpose and may use the WRCK, AUXCKn and WRSTN ports. [27]

IEEE1500 only defines the structures previously described. It does not acknowledge how they should be accessed, and thus the way they are used does not have to conform to any standards according to IEEE1500. IEEE1687 addresses access to embedded instruments within a chip, which may be IEEE1500 compliant. This way the wrappers may be connected to a IEEE1149.1 TAP with the help of ICL and PDL languages. [15]

### **2.3.5 IDDQ**

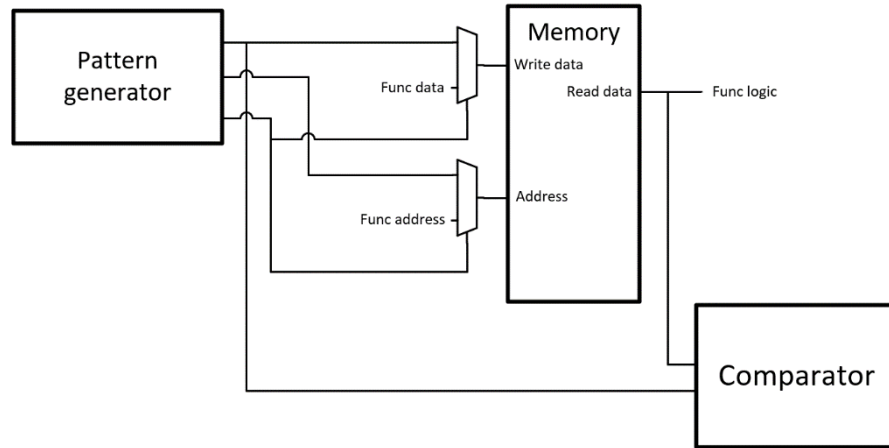
Another method for testing a circuit while using scan chains but without ATPG is quiescent current (IDDQ) testing. With this method no specific faults are targeted, rather the entire chip is evaluated as a whole. In IDDQ testing the chip is placed to a known state through scan loading. Its leakage current is then measured and compared to what is expected, which can be acquired from library characterization data, from simulations or from measured data from other chips. If the current is greater than expected, it can be assumed that a fault has taken place. [28, 29]

Even with IDDQ testing multiple patterns have to be applied to the chip in order to excite the faults so they can be seen on the measurements. The time that a single pattern requires is greater than with ATPG based tests, as the chip will require time for the current to stabilize. Another issue is that the cutoff point between a good and a bad chip must be chosen carefully and might not be clear in some cases. [4]

## **2.4 MBIST**

Memories are generally tested by adding extra logic called memory built-in self-test (MBIST). The logic is responsible for generating the test vector and reading the response of the memory. No standard exists for MBIST and as such its architecture varies between

implementations. The main operating principle is the same; executing the pattern, typically a march pattern, named for its function of marching through the address space of the memory, then comparing the data that was read back with what was written. Figure 22 illustrates a simplified view of a memory with MBIST added around it.



**Figure 20.** Example implementation of MBIST

March patterns are characterized by their signature. A signature consists of typically multiple passes through the address space, where a single pass may perform multiple operations on the memory. The order in which the addresses are traversed is denoted with either  $\uparrow$ ,  $\downarrow$  or  $\updownarrow$  representing ascending, descending or don't care order, respectively. The operations during the pass are either reads or writes, marked with rx or wx, where x can be either 0 or 1. The value 0 means that the value to read or write is the background value and value 1 is the inverse of it. Backgrounds are addressed later, for now let's use a solid background, which would make r0 operation to mean to read a 0 on all bits of the address. The signature

$$\uparrow (w0, r0), \uparrow (w1, r1)$$

would mean that the address space of the memory is cycled through twice in ascending order, so that each cycle will first write and then read, using the value 0 for the first pass and 1 for the second. [30]

March patterns are often referenced with a time complexity as well as their signature. The time signature denotes how long the operation will take as a function of the size of the address space of the memory. The previous example would be a march  $4N$ , as the time required for each address is 4 clock cycles (2 writes and 2 reads). Several different march algorithms have been proposed in literature, for detecting different types of faults. One of the more common ones is March C-, the signature of which is

$$\updownarrow (w0), \uparrow (r0, w1), \uparrow (r1, w0), \downarrow (r0, w1), \downarrow (r1, w0), \updownarrow (r0).$$

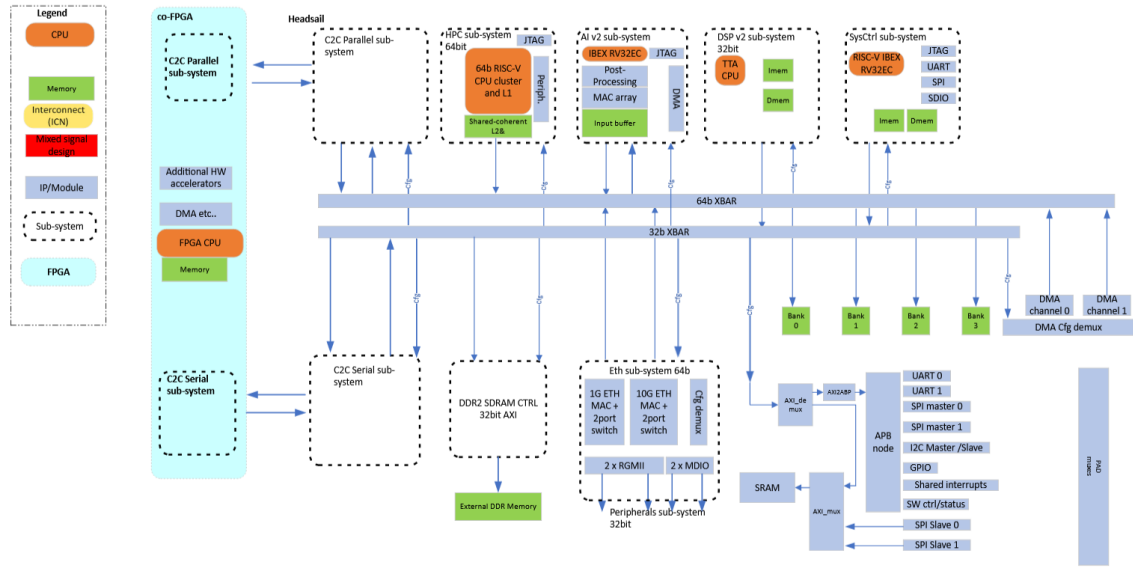
The complexity of March C- is  $10N$  and it can detect SAF, TF, ADF, CFin and CFid faults. Some more complex fault models, however, cannot be detected with this pattern. [31, 32]

Another thing to consider when choosing the MBIST algorithm is the background type. The background denotes what is written to different bits of the address. The simplest patterns are solid, where all bits of the word are the same value, and checkerboard, where bits are alternating 1s and 0s. The background selection is heavily influenced by the physical structure of the memory. It is important to note that some MBIST schemes allow for running the tests on a bitwise basis, targeting single bits instead of words. In this case a background is not used. [32, 33]

Besides testing the core array, MBIST logic should also test the peripherals of the memory. This means the read and write enables, address decoders, write masks and other possible logic outside the core. The address decoders and enables are tested along the core, but things such write masks, write-through ports, power down modes and retention modes are to be separately tested. March patterns don't target these specifically but may be used for their verification.

### 3. HEADSAIL

Headsail is the third System-on-Chip (SoC) done as a collaboration of various companies and institutions as part of SoC-Hub. It is comprised of 7 subsystems and a top level connecting them as illustrated in Figure 1. While the functionality of the chip is not relevant for DFT purposes, certain structures and physical partitioning which affect implementation are described in this chapter.



**Figure 21.** Headsail structure

Headsail is a good design to use for this work due to its heterogenous, hierarchical structure allowing various DFT structures, and as SoCs such as it are common in industry.

#### 3.1 SysCtrlCPU

Possibly the most crucial part of the chip is its system control central processing unit (CPU), as it controls resetting and clocking the various subsystems. In case it was not functioning the entire chip could be inoperable. For logic testing this is not any more important than any other part of the chip, but it being able to control clocks of other subsystems is important for memory testing. This will be further expanded in chapter 4.1.

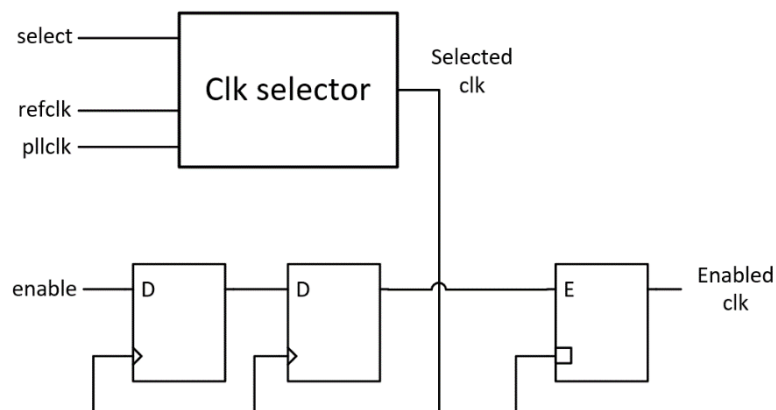
#### 3.2 Subsystems

As shown in figure 1, each subsystem is a separate hierarchy. This is not only logical, but the chip is also physically partitioned, and each subsystem is implemented as a hard macro. This helps bring down physical design effort, as changes in design often involve

only certain parts of the design and having to rerun full implementation for the full chip would take a considerable time.

All subsystems communicate through an Advanced eXtensible Interface (AXI) interconnect on the top level. This bus goes through hardened blocks called clock bridges so that timing between top level and subsystems must be done only for the clock bridges. The phase locked loops (PLL) which generate required clocks for each subsystem are controlled through a common interface from system control processor on the top level. Besides these common connections each subsystem might have some sidechannels such as interrupts or debug interfaces towards top level. Important part to note is that there are no signals which feed directly from subsystem to subsystem.

Figure 2 describes the architecture of the common subsystem clock control block. In functional operation this allows good control of the PLL of each subsystem as well as a backup of running the blocks with the slow reference clock. Controlling the clocks during DFT is elaborated on in chapter 4.1.



**Figure 22.** Simplified view of subsystem clock control

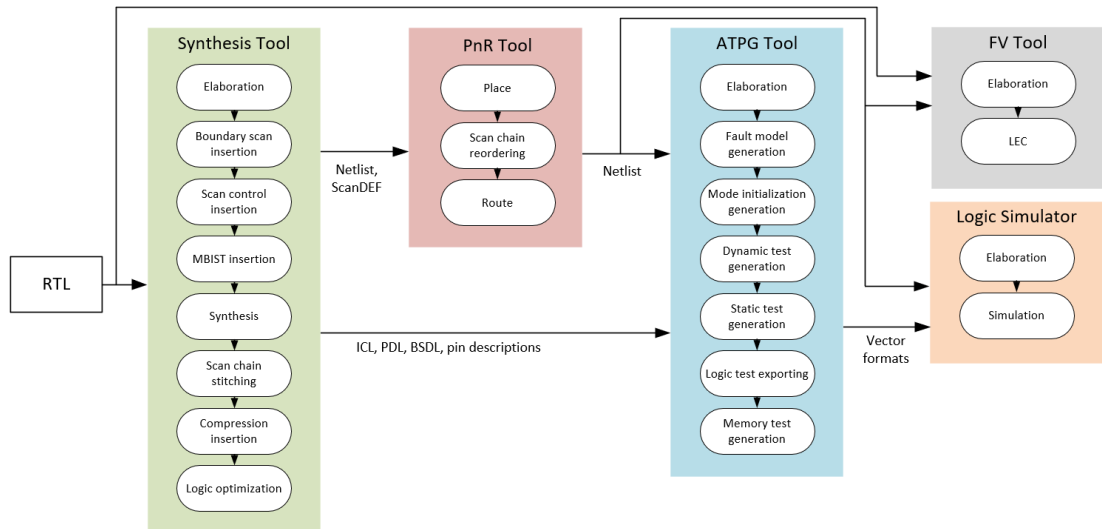
### 3.3 Physical design

Headsail is synthesized from VHDL and SystemVerilog descriptions to a 22 nm standard cell library. Total area for the whole chip is 25 mm<sup>2</sup>, with a flip-flop (FF) count of 662k.

Most of the chip runs with a shared 0.9 V power rail but parts of the high-performance computing subsystem (HPC) have their own supplies with level shifters inserted between voltage domain crossings. Some parts of the design also have power gates to enable low power operation. Both of these features are simple to control in DFT but have to be acknowledged. This is further discussed in chapter 4.1.

Figure 23 illustrates the relevant steps in each tool of the used flow. All of the insertion takes place in synthesis tool, but the place and route (PnR) tool may reorder scan chains

with the information in a Scan Design Exchange Format (ScanDEF) file to help minimize wire length within scan chains. The synthesis tool also writes the required files for the ATPG tool, namely the ICL, PDL and BSDL as well as descriptions for pin functionality. These are used to generate the start-up sequences of tests, which set, for example, the WIR bits to required values.



**Figure 23.** DFT flow in different tools

Most of the insertion is done before synthesis, with only scan chain creation and compression insertion being after synthesis. This is as some of the inserted structures are unmapped, thus requiring synthesis, but chain stitching not being possible before all flops are mapped to scan flops, which is done in synthesis. A formal verification (FV) tool is used to perform logic equivalence checking (LEC) to ensure the insertion does not affect the functionality of designs.

The ATPG tool writes out generated vectors in a format that allows them to be simulated with a logic simulator. There are certain things the ATPG tool assumes and does not verify, such as PLL operation, so running a full simulation after test generation is necessary.

To speed up test flows, the PnR tool can be ignored and the netlist from synthesis may be used. It is important to note that if scan chain reordering is done, the vectors generated for a synthesis netlist will not be compatible with the final netlist, and even if the chains retain their order, logic is optimized in PnR and as such all faults may not be covered.

### 3.4 DFT specification and requirements

Before design work had started for the chip, it was decided that as much effort as possible would be put into the DFT implementation, regardless of what the design was going to be. Limiting factors for the implementation are mostly in the size of the design and the number of available IOs but rough estimates for both were available before DFT design started. All DFT structures were implemented in synthesis tool except a small control block for MBIST was implemented in RTL.

The number of dedicated pins allocated to DFT purposes was 10. Table 1 documents how they were planned to be used. Note that no pins were allocated for scan IOs, as it was known that the number of these would be high, so it was decided that functional pins would be shared for scan purposes.

**Table 1.** Planned dedicated DFT pins

PIN	FUNCTION
dft_jtag_tck	JTAG TCK, scan clock
dft_jtag_tms	JTAG TMS
dft_jtag_tdi	JTAG TDI
dft_jtag_tdo	JTAG TDO
dft_jtag_trst	JTAG TRST
jtag_dft_disable	Global DFT mode disable
scan_enable	Global shift enable
mask_load	Global mask data load enable
mask_enable	Global mask enable
opcg_load	Global OPCG data load enable

As the tester to be used was unknown during implementation, some assumptions had to be made about it. The following things are required to test this chip:

1. Unlimited number of pins to connect to scan IOs
2. One clock synchronous to scan data with control over the timing between them
3. One asynchronous clock running at 30 MHz

The actual number of IOs used would be refined as test runs are done. No estimates for it were initially made.

Synthesis is constrained such that data inputs are expected to be driven on the falling edge of scan clock. Outputs are similarly clocked on the falling edge. Condition 2 should be trivial for any tester but is necessary to make sure timing is not violated on any chips IOs. The frequency was not defined for the scan clock but was expected to be between 10 and 50 MHz.

Number 3 is only required for delay testing. The asynchronous clock can possibly run at different frequencies as well, but the used PLL is designed for this frequency and could cause unexpected issues with other inputs. The frequency at which data is loaded

through scan inputs and shifted can be anything as long as it is synchronous to the JTAG clock.

Soft targets of 95% static faults and 75% dynamic faults detected were set initially, based on few test runs. Another requirement was minimal test time, but no actual value for this was set. All memories were planned to be tested for common fault model types.

## 4. HEADSAIL DFT IMPLEMENTATION

This chapter describes how DFT was implemented on Headsail considering the requirements and targets.

### 4.1 Boundary scan

An 1149.1 compliant boundary scan was added to the chip. As many as pins as possible were used for boundary testing. The pins that were not used include the required JTAG pins, compliance pins, reset pin and pins that used a different IO voltage or were differential.

For JTAG a five-pin interface including TRST was used. Compliance pins include the `jtag_dft_disable` pin used for disabling the DFT JTAG tap. This pin was not strictly necessary as TRST could have filled any functions that it was used for, however since the chip had room to add this signal it was done. In the end it only functioned only as a second reset pin for the JTAG macro and could easily have been dropped.

Other compliance pins were scan pins, namely `scan_enable`, `opcg_load`, `mask_load` and `mask_enable`. Technically only `scan_enable` had to be a compliance pin but as `opcg_load`, `mask_load` and `mask_enable` are tied to their compliance values on the PCB anyways and thus cannot be tested they were also not used for boundary scan. The last compliance pin was the global reset pin of the chip. This was not ideal as the reset pin traces absolutely should be tested but there were certain control pins on pads that needed to be driven to a specific value to have them function correctly. Ideally, they would have been driven from the JTAG macro but due to time constraints reset was added as a compliance value in its active state to make sure these configurations would stay in correct values during test.

The SDRAM subsystem uses 1.2 V IOs which are a different design from standard IOs used almost everywhere else. They had to specially designed and do not use the same interfaces as the standard pads so again due to time constraints they were not included in the boundary scan configuration. The same applies to pads used in C2C serial subsystem, which used a different voltage and were differential.

Only the mandatory instructions were used as it was not seen necessary to add any of the optional ones. `SAMPLE` and `PRELOAD` used different opcodes as extra safety in case either one of them did not function properly. The instruction register length was 6

to accommodate for all planned instructions, but only 17 were used. The non-boundary scan instructions were MBIST and IEEE1500 access instructions.

The boundary register was connected in physical order of pins in counterclockwise direction. On a small chip like this it is not absolutely necessary but on larger chips it is important in order to meet timing in the chain.

## 4.2 Scan

A single clock was used for shifting data. This was also the clock used for JTAG to minimize requirements for tester. Scan enable was similarly shared across the full chip. It was not timed in any way so when generating ATPG vectors a relatively long time was used for allowing it to propagate when toggling.

The design was hierarchically separated so utilizing hierarchical test in the form of IEEE1500 and IEEE1687 was a natural choice. Parallel interface was not used as there was no need for it. The serial interface was as defined in the standard, with the access mechanism for WIR from top level through JTAG. The core isolation signals WINT and WEXT were controlled from WIR.

To enable any scan modes in each of the blocks a scan mode test signal had to be asserted to control clock muxing as well as to control some other functionality such as pads and memories. This was initially considered to be shared across each subsystem and only be controllable from top level either from JTAG or from a port. In the end it was decided against this idea as requiring specific control for a port on a 1500 wrapped block, while standard compliant, was considered unnecessary and would require extra work at top level were the block reused or sold as an IP. As a result, the scan mode signal was added to each block's WIR.

For compression it was decided to use XOR for both compression and decompression as it was the simplest to implement and verify. Other methods could have increased test coverage, however since it was planned to use a non-compression test mode to top off any missed faults, implementing more complicated compression mechanisms was not seen necessary.

The length of compression channels chosen to be 300 flops based on early tests with a small design. Knowing this and the estimated numbers of total flops and available IOs, it was calculated that the compression ratio could be around 40X in total, with some smaller blocks as low as 15X and larger blocks with higher ratios. This estimate still had some margin, as the number of IOs used for these calculations was on the low side. Table 2 shows the final numbers. In the end the channels could have been shorter, but

this was an expected result as it was known that the early estimates would not have been completely accurate. Note that the total compression ratio expresses what could have been if compression was done on a flat design.

**Table 2. Compression ratios**

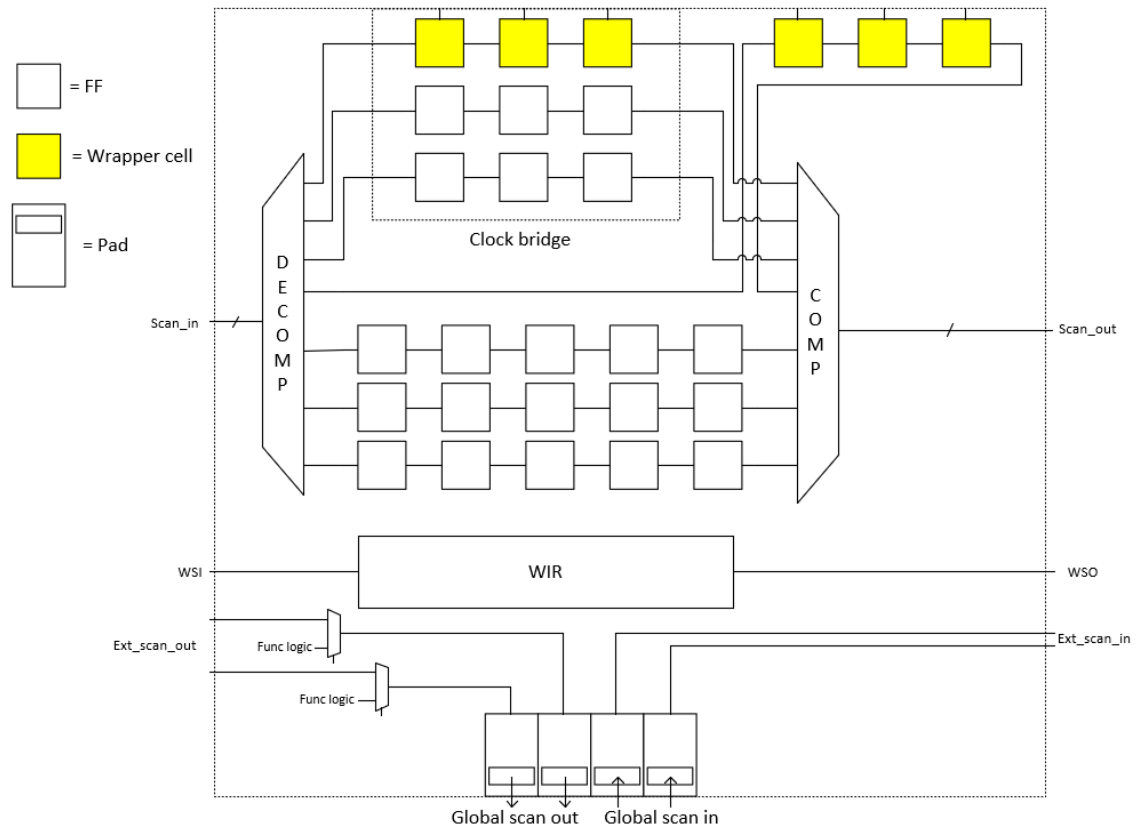
BLOCK	# FFS	SCAN PORT PAIRS	COMPRESSION RATIO
ETHERNET	85k	6	47
HPC	230k	14	55
AI	220k	12	61
C2C PARALLEL	7k	2	12
C2C SERIAL	-	-	-
DDR	9k	2	15
DSP	11k	2	18
TOP	100k	12	28
TOTAL	662k	50	44

As there were analog macros and pads which would feed unknown values to scan chains during scan capture, a compression mask was deemed necessary. A mask that sets channel outputs to 0 when active was chosen, however not much thought was used in choosing the appropriate mask type as it wasn't expected to make much of a difference in test coverage.

Mask load was an input to top level similarly to scan enable and was shared between every subsystem. Scan clock was used for mask loading by gating it from chains during mask load operation. Scan inputs doubled as mask chain inputs during this operation. As opposed to initial plans, a separate mask enable was used for each subsystem, as sharing a mask enable signal would increase tester time since an empty mask would have to be loaded to each subsystem except for ones that used their mask on a specific test cycle.

The IO subsystems were created such that they contained the IO pad cells that they needed for operation. This meant that there were not enough pads on top level to fully control the scan. This was easy to mitigate by using pads from the Ethernet and C2C parallel subsystems, as shown in figure 24. Controls of the pads were overridden with scan mode and outputs of any pads not used were turned off.

For clock bridges scan chains were built within, but compression was not added, as the chains would be connected to their parent instance's compression macros. Clock bridges were always at the boundary of a core, and they were physically placed such that WBR could not be added at their parent level. This meant that creating the wrapper boundary chain was necessary for clock bridges.

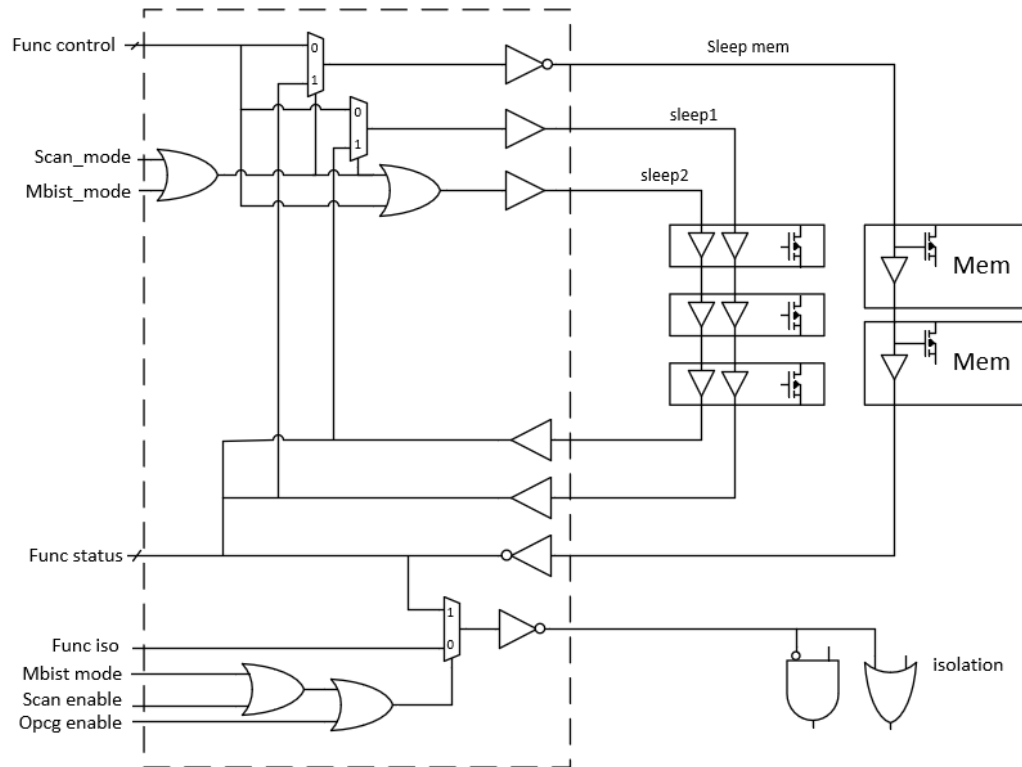


**Figure 24.** Scan paths through a subsystem

Clock bridges were hard macros, but from 1500 point of view they were not considered cores. As such, they did not have their dedicated WIRs, but would get their test signals from their parent instances. This meant scan mode, scan enable, and the signals needed for WBR control, namely WINT and WEXT.

The HPC subsystem was a bit different from any other subsystems as it had separately synthesized cores inside. They were easily handled the same way as clock bridges, by connecting the channels inside and bringing them to the subsystem's top level where they were connected to compression macros. None of the cores faced towards chip top level so adding a WBR was unneeded.

Parts of HPC as well as AI were in their own power domains with power gating. Naturally they were supposed to be tested, so making sure the gates were not blocking was necessary. This was easily accomplished by ORing the control signals of the first gate chain with the mode specific test signal and connecting all the chains in series. Figure 25 shows how this was done. The buffers and inverters were part of the RTL to provide fixed points for connections. Note inverted polarity of memory enable compared to switches. Isolation was tested in static scan modes but not with OPCG, as these paths were untimed.



**Figure 25.** Power control during test modes

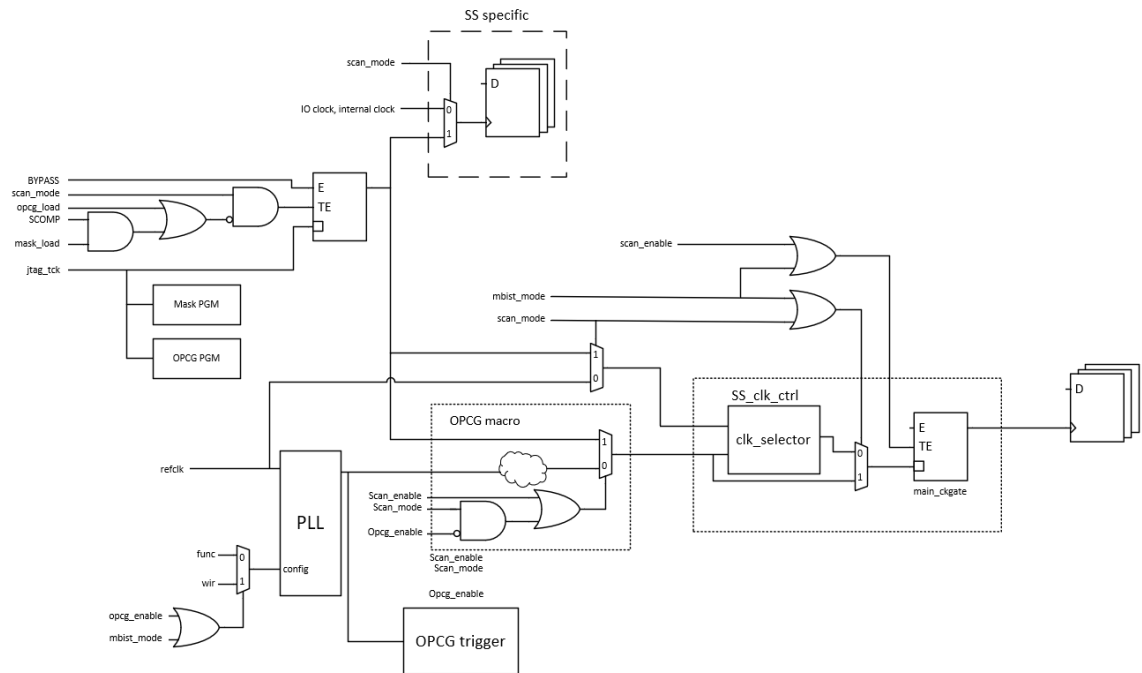
OPCG and delay tests were implemented as shown in figure 26. The OPCG macros were inserted in synthesis tool. They were programmed with scan data inputs and scan clock, such that the scan clock was gated from the rest of the design during programming to make sure any scan data values were preserved. Figure 26 shows how this was done in similar fashion to mask programming.

Since the PLL in each subsystem was identical, it was simple to control each of them in the same way. WIR was chosen for this purpose as it was already used and was easy to configure for different types of tests. The other option was to use a sideload chain for them in the same way as the OPCG macros are programmed, but as a tool proprietary method it is not as flexible and was quickly dismissed.

The required controls for the PLL were 3 different dividers and a loop control value to select the range of the internal oscillator. In addition to these there were several enables on the PLL, but since it was enough to tie all of them to 1, they were all controlled with a single OPCG enable bit from the WIR, which was also used in certain other places to control clocks and isolation.

To keep thing as simple as possible, the falling edge of scan enable was used as the trigger for OPCG. The actual delay from falling edge to generating the test pulses was chosen as 5 cycles of the fast clock as this was seen as a sufficiently long time to allow

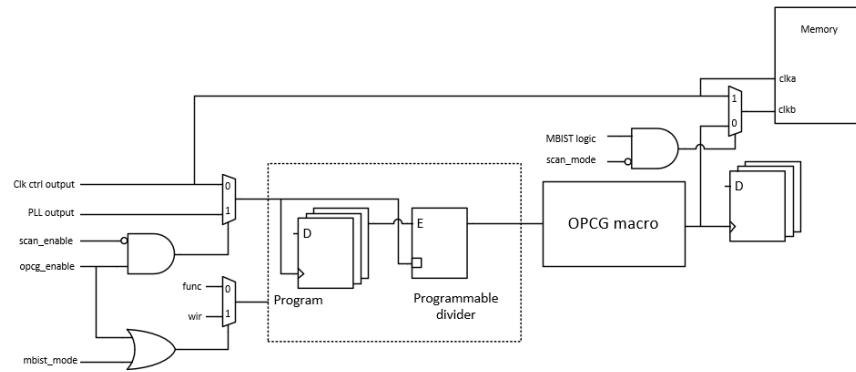
the scan enable to propagate to the whole design. This value was hard coded in the hardware and as such a relatively large value was used to make sure there would be no issues.



**Figure 26.** Control of clocks in OPGC-inserted subsystems

Due to time constraints, not every subsystem had OPGC implemented. Many of the ones that did implement it contained multiple clocks, either from IOs or divided from the PLL. These were subsystem specific, but generally logic clocked with an IO clock or logic clocked with a low frequency were not tested for transition faults.

This left the only things to be tested with OPGC the logic running with PLL and logic running with a clock divided from the PLL with a relatively low division value. For the divided clocks the options were either to use the already instantiated divider or to create a new divider as part of the OPGC macro. The former was chosen as this was simpler and would allow implicit transition testing of the divider, as it's possible faults would be detected as part of the tests. One important thing to notice is that the divider could not be fed as usual, since the OPGC macro used for the PLL domain would block the clock while testing the divided domain. For this reason, a mux would have to be inserted on the input clock of the divider to make sure it would always have a running clock. This is shown in figure 27.



**Figure 27.** Control of clock dividers during OPCG testing

Having the clock of the divider running meant that the counter value would be constantly changing. In addition to this, the counter registers were always a part of scan chains as they would be tested normally in static fault test modes. This meant that they would always be at completely unknown values. In general, this is not an issue, as the phase difference of the original and divided clocks is not relevant. The only issue that could rise from this is if the divider would be in an illegal state after scan shift. For example, if the divider were programmed to divide by 3, its counter would normally reset at 2. However, after scan, it could be at a value above 2. Synthesized counters work such that they always roll over to 0 from their largest value, so eventually even an illegal value will get to a legal state. The only issue arises when (1) the divider value is programmable (2), the maximum value is very large and (3) the value used during test is very small. In such case, the counter will take too long to reach a legal state, generating no clock pulses during the test period. Even in this worst case the test period can be increased to mitigate this problem, but importantly there were no such dividers in any design, so using the existing dividers was seen as the best solution.

In the designs that used programmable dividers the divisor value was controlled from WIR as shown in figure 27. In HPC some of the cores had their dedicated dividers from which they were fed. The OPCG macros for these cores were inserted on HPC top level, but they could as well have been inside the cores.

### 4.3 MBIST

The original target was to have all memories be testable. Due to time constraints MBIST was only implemented in Ethernet and DLA subsystems and top level.

As with other DFT structures, MBIST logic was inserted in synthesis tool. The chosen logic could be controlled either from the DFT JTAG macro or from system control CPU. When using the JTAG access method clocks and reset must be controllable from top

level ports or from JTAG. Many of the structures created for scan were used for MBIST. Resets only needed to be only controlled if there were memories which had their clocks fed from a clock divider, so that the divider could be lifted out of reset. This was the case in both of the subsystems. For PLL and divider control the existing test signals in WIR were reused. When testing from sysctrlcpu the chip is in its functional mode and all clocks and resets are controlled normally, MBIST does not change this. The MBIST logic requires all the memories to use a single clock if they have separate read and write clocks. For this reason, a mux was needed on some memories to ensure that both of the clocks were fed from the same source. During scan however the memories should run with the original clock and so control for the mux was also necessary in scan modes. This is shown in figure 27.

Testplans were chosen so that the most common peripheral and core faults could be tested, with the main core testing signature being

$$\uparrow (w0), \uparrow (r0, r0, w0, r0, w1), \uparrow (r1, r1, w1, r1, w0), \downarrow (r0, r0, w0, r0, w1), \\ \downarrow (r1, r1, w1, r1, w0)$$

with a solid background. The patterns were hard programmed into the logic and so they cannot be changed during runtime. This simplifies the software required for running the tests.

The memories used had redundant resources which can be utilized if there are faults within. For this purpose, they have a simple interface which can be controlled to repair a certain number of faults. Normally the repair interface would be tied to its inactive value, but when adding the MBIST logic it was possible to add control for this interface. There are two ways of repairing the memories: soft and hard. Soft repair means that the repair data is stored in flip-flops and thus erased every time the chip is reset. Hard repair requires either fuses or non-volatile memories (NVM) so that the repair data is retained whenever power is cycled off. There were no fuses or NVM available for this chip, so a soft repair scheme was chosen.

If the chip is turned on in DFT mode, that is if the `dft_jtag_disable` is 0, MBIST can be run through the DFT JTAG interface. To change the operation mode, `dft_jtag_disable` has to be set to 0, and the chip has to go through reset. This means that MBIST repair data cannot be set in JTAG mode and then carried over to functional operation. With fuse or NVM repair data this would be possible. This is how repair would be typically done in production chips, however as this was not an option the repair had to be able to be done in functional mode. This was the main reason for adding the control from `sysctrlcpu`. In a test chip this makes a lot of sense, especially when compared to fuses, as

they are one-time-programmable and if set incorrectly the chip or parts of it could become inoperable.

Typical operation from SysCtrlCPU to repair Ethernet memories would be as follows (in pseudocode):

```

clk_rst_setup(ETH);
start_fault_accumulation(ETH);
for (i=0; i<NUM_TESTPLANS; i++) {
    run_testplan(ETH, i);
}
repair_memories(ETH);
for (i=0; i<NUM_TESTPLANS; i++) {
    ret = run_testplan(ETH, i);
    if (ret) {
        puts("Fault after repair")
    }
}

```

The fault data accumulation is a specific operation after which any tests that encounter faults save them to an internal register which is then used with `repair_memories(ETH)` to attempt to repair faults that were encountered. All testplans are run again in case any memories had more faults than could be repaired.

It is important to note that repaired memories might run slower than unrepaired ones and this could limit performance of them. Even if tests fail after repair, it is possible they will pass with a lower frequency. Ethernet subsystem requires its memories to always run at a set frequency so it was designed such that it will work even if repair is needed but AI and top level memory performance might suffer from being repaired.

From the data that can be read back from MBIST logic it is possible to pinpoint which memory has failed but not at which location. This was not seen to be worth the extra logic that was required to implement the diagnostics.

## 5. RESULTS

No ATE was available for this project so the results are only hypothetical. However, if the chips need to be tested the vectors are available. All tests were simulated to be functional. In all other parts the structures allowed testing as planned, except for the AI block, where a late RTL change caused issues in testability and as such it cannot be considered in testing, and top level, where scan without compression could not be run.

The biggest challenge regarding DFT was the lack of time. Due to tight deadlines OPCG could not be added to many of the blocks and as such dynamic faults are not covered well. No optimization runs were done either, meaning the number of scan IOs was not optimal and the number of test vectors varies a lot between blocks.

ATPG was run so that OPCG mode was done first to cover dynamic faults and a majority of static faults. Tests were first created with compression and final vectors were added without compression to bring coverage up. Table 3 shows the number of vectors in each testmode for the different blocks.

**Table 3.** Number of tester cycles for each testmode

	DYNAMIC		STATIC	
	Compression	Non-compression	Compression	Non-compression
ETHERNET	7365	2623	598	977
HPC	-	-	19241	8706
AI	-	-	-	-
C2C PARALLEL	-	-	1600	2599
C2C SERIAL	-	-	-	-
DDR	-	-	1988	1558
DSP	-	-	1485	351
TOP	9368	-	4396	-

All subsystems would be run at the same time and top level would be run separately. This means that the total number of tester cycles required is  $7365 + 2623 + 19241 + 8706 + 9368 + 4396 = 51699$ . The length of chains is at most 300, meaning a single vector takes 300 clock cycles and time required for capture. It is sufficient to estimate the time of capture cycle as 3 shift clock cycles for both static and dynamic tests. At a shift frequency of 20 MHz this would mean testing a single chip takes 47 seconds. Table 4 documents the coverage across the chip. Subsystems reach reasonable rates, but due to non-compression scan not working on top level, especially dynamic fault testing does

not have great coverage. It is also important to note that these numbers include all possible faults, including those that are known to be untestable, such as unused pads or test structures such as the JTAG macro.

**Table 4.** Coverage of logic faults in hierarchies

	DYNAMIC	STATIC
ETHERNET	82.34	94.75
HPC	-	95.50
AI	-	-
C2C PARALLEL	-	91.65
C2C SERIAL	-	-
DDR	-	93.62
DSP	-	94.76
TOP	57.95	91.93

Initially, no limit on the number of IOs was set as no tester was designated for the purpose. The number of pins needed to achieve the 47 second test time is 90. If the tester cannot support this many pins, subsystems can be tested separately, bringing the minimum of pins needed down to (scan\_enable + JTAG + subsystem scan IOs), which would allow a tester to completely test the chip without compression and OPCG at 19 pins, with compression and OPCG (previous + mask\_enable + mask\_load + opcg\_load + async. clock) at 23 pins.

MBIST was inserted in the Ethernet and AI blocks as well as the top level. The coverage of it is hard to analyze, especially without measured data but it should be sufficient to test most of the faults, depending on which fault model is used. As with OPCG, tight deadlines kept MBIST from being added to some blocks. The most important of these is the HPC, as it contains a lot of memories but with AI and top covered most of the memory area on the chip can be tested and repaired if needed. Note that unlike scan, MBIST can be run as the interfaces connect to the SysCtrlCPU, but at the time of writing chips have not been manufactured and cannot be tested on.

Boundary scan is the only part that was completed as planned. IOs of C2C serial were not included in the boundary chain as this block was done completely without DFT and SDRAM IOs were excluded due to different structure, excluding that all IOs were part of the boundary chain. This is expected to help the PCB designers a lot if any issues arise.

## 6. CONCLUSIONS

The purpose of this thesis was to explore and implement a robust DFT strategy for a modern SoC. The faults to be tested for were first expanded on in chapter 1. Common strategies, conventions and standards were summarized in chapter 2. The used platform and targets and requirements for it were presented in chapter 3 and lastly the implementation was described in chapter 4 with results of it in chapter 5.

The initial target was to be able to test at least 95% of static faults and 75% of dynamic faults, as well as some types of faults present in memories. The targets were partly met, with short timeline being the major obstacle. Without DFT the chips would have been practically untestable, as the test would have to have been created using software, and as such the work was successful. No tests were yet performed on a physical chip as they were not available at the time of writing.

Only research left for this work is to apply the tests on the physical chips. Issues may still be found this way, and the interesting part would be diagnosing faulty chips and possibly even pinpointing faults that are present. Any of the strategies may be used in future chips as well, possibly complemented with more exhaustive testing strategies, such as IEEE1149.6 and Logic Built-In Self-Test (LBIST).

## REFERENCES

- [1] H. H. Butt, ASIC DFT techniques and benefits, *Sixth Annual IEEE International ASIC Conference and Exhibit*, Rochester, NY, USA, 1993, pp. 46-53
- [2] Modus guide 4: Faults, Cadence Design Systems
- [3] J. Rearick, B. Eklow, K. Posse, A. Crouch and B. Bennetts, IJTAG (internal JTAG): a step toward a DFT standard, *IEEE International Conference on Test, 2005.*, Austin, TX, USA, 2005, pp. 815-823
- [4] Modus guide 5: ATPG, Cadence Design Systems
- [5] F. Zokaee, H. Sabaghian-Bidgoli, V. Janfaza, P. Behnam and Z. Navabi, A novel SAT-based ATPG approach for transition delay faults, *2017 IEEE International High Level Design Validation and Test Workshop (HLDVT)*, Santa Cruz, CA, USA, 2017, pp. 17-22
- [6] J. Mahmood, S. Millican, U. Guin and V. Agrawal, Special Session: Delay Fault Testing - Present and Future, *2019 IEEE 37th VLSI Test Symposium (VTS)*, Monterey, CA, USA, 2019, pp. 1-10
- [7] Von-Kyoung Kim and T. Chen, On comparing functional fault coverage and defect coverage for memory testing, in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 11, pp. 1676-1683
- [8] Jin-Fu Li, Kuo-Liang Cheng, Chih-Tsun Huang and Cheng-Wen Wu, March-based RAM diagnosis algorithms for stuck-at and coupling faults, *Proceedings International Test Conference 2001 (Cat. No.01CH37260)*, Baltimore, MD, USA, 2001, pp. 758-767
- [9] H. B. Shashidhara, S. Yellampalli and V. Goudanavar, Board level JTAG/boundary scan test solution, *International Conference on Circuits, Communication, Control and Computing*, Bangalore, India, 2014, pp. 73-76
- [10] J. T. Sousa, T. Shen and P. Y. K. Cheung, Realistic fault extraction for boards, *Proceedings ED&TC European Design and Test Conference*, Paris, France, 1996, p. 612
- [11] G. Lakshmi Ch, V. U. Sankar and S. S. Yellampalli, A Reference Based Approach to Detect Short Faults in PCB, *2021 Fifth International Conference On Intelligent Computing in Data Sciences (ICDS)*, Fez, Morocco, 2021, pp. 1-7
- [12] IEEE Standard for Test Access Port and Boundary-Scan Architecture, *IEEE Std 1149.1-2013 (Revision of IEEE Std 1149.1-2001)*, 13 May 2013, pp. 1-444
- [13] IEEE Standard for Boundary-Scan Testing of Advanced Digital Networks, *IEEE Std 1149.6-2015 (Revision of IEEE Std 1149.6-2003)*, 18 March 2016, pp. 1-230
- [14] IEEE Standard for Reduced-Pin and Enhanced-Functionality Test Access Port and Boundary-Scan Architecture, *IEEE Std 1149.7-2022 (Revision of IEEE Std 1149.7-2009)*, 14 Oct. 2022, pp.1-1048

- [15] IEEE Standard for Access and Control of Instrumentation Embedded within a Semiconductor Device, *IEEE Std 1687-2014*, 5 Dec. 2014, pp. 1-283
- [16] IEEE Standard for Test Access Architecture for Three-Dimensional Stacked Integrated Circuits, in *IEEE Std 1838-2019*, 13 March 2020, pp. 1-73
- [17] A. H. Boyce, An introduction to digital testing, *1988 IEE Colloquium on Design for Testability (Digest No. 1988/32)*, London, UK, 1988, pp. 1-3
- [18] S. DasGupta, E. Eichelberger and T. Williams, "LSI chip design for testability," *1978 IEEE International Solid-State Circuits Conference. Digest of Technical Papers*, San Francisco, CA, USA, 1978, pp. 216-217
- [19] A. Chandra, S. Chebiyam and R. Kapur, A Case Study on Implementing Compressed DFT Architecture, *2014 IEEE 23rd Asian Test Symposium*, Hangzhou, China, 2014, pp. 336-341
- [20] K. J. Balakrishnan, Emerging Techniques for Test Data Compression, *14th Asian Test Symposium (ATS'05)*, Calcutta, India, 2005, pp. 462-462
- [21] S. Alampally, J. Abraham, R. A. Parekhji, R. Kapur and T. W. Williams, Evaluation of Entropy Driven Compression Bounds on Industrial Designs, *2008 17th Asian Test Symposium*, Hokkaido, Japan, 2008, pp. 13-18
- [22] S. Bahl *et al.*, Unifying scan compression, *2014 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, Amsterdam, Netherlands, 2014, pp. 191-196
- [23] J. Rajski, N. Tamarapalli and J. Tyszer, Automated synthesis of large phase shifters for built-in self-test, *Proceedings International Test Conference 1998 (IEEE Cat. No.98CH36270)*, Washington, DC, USA, 1998, pp. 1047-1056
- [24] A. A. Bawa, M. T. Rab and N. A. Touba, Using partial masking in X-chains to increase output compaction for an X-canceling MISR, *2012 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, Austin, TX, USA, 2012, pp. 19-24
- [25] J. Savir, Skewed-Load Transition Test: Part I, Calculus, *Proceedings International Test Conference 1992*, Baltimore, MD, USA, 1992, p. 705
- [26] J. Savir and S. Patil, Broad-side delay test, in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 8, pp. 1057-1064
- [27] IEEE Standard Testability Method for Embedded Core-based Integrated Circuits, *IEEE Std 1500-2022 (Revision of IEEE Std 1500-2005)*, 12 Oct. 2022, pp.1-168
- [28] J. Hirase, N. Shindou and K. Akahori, Scan chain diagnosis using IDDQ current measurement, *Proceedings Eighth Asian Test Symposium (ATS'99)*, Shanghai, China, 1999, pp. 153-157
- [29] R. Kawahara, O. Nakayama and T. Kurasawa, The effectiveness of IDDQ and high voltage stress for burn-in elimination [CMOS production], *Digest of Papers 1996 IEEE International Workshop on IDDQ Testing*, Washington, DC, USA, 1996, pp. 9-13

- [30] A. Sharma and V. Ravi, Built in self-test scheme for SRAM memories, *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Jaipur, India, 2016, pp. 1266-1270
- [31] L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel and S. Borri, March iC-: an improved version of March C- for ADOFs detection, *22nd IEEE VLSI Test Symposium, 2004. Proceedings.*, Napa Valley, CA, USA, 2004, pp. 129-134
- [32] A. Benso, S. Di Carlo, G. Di Natale and P. Prinetto, An optimal algorithm for the automatic generation of March tests, *Proceedings 2002 Design, Automation and Test in Europe Conference and Exhibition*, Paris, France, 2002, pp. 938-943
- [33] S. Hamdioui, R. Wadsworth, J. Delos Reyes and A. J. van de Goor, Importance of dynamic faults for new SRAM technologies, *The Eighth IEEE European Test Workshop, 2003. Proceedings.*, Maastricht, Netherlands, 2003, pp. 29-34