POLITECNICO DI TORINO

MASTER OF SCIENCE THESIS

Testing and Validation of a Hardware Accelerator for Tensor Core Units in AI Applications

Supervisor:

Prof. SONZA REORDA MATTEO

Author:

FAIYAZI FARKHAD MILAD

Co-supervisors:

Dr. GUERRERO BALAGUERA JUAN DAVID

Dr. RODRIGUEZ CONDIA JOSIE ESTEBAN

Master of Science degree in

Electronics Engineering

March 2026

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Professor Matteo Sonza Reorda, for his invaluable guidance, unwavering support, and insightful feedback throughout this thesis. His profound expertise and continuous encouragement have been instrumental in shaping both the direction and depth of this research.

My sincere thanks also go to Dr. Juan David Guerrero Balaguera and Dr. Josie Esteban Rodriguez Condia for their exceptional technical advice, constructive discussions, and generous assistance during the design and experimentation phases. Their contributions significantly enriched the quality and rigor of this work.

Together with Dr. Josie Esteban Rodriguez Condia and Dr. Juan David Guerrero Balaguera, I extend our sincere appreciation to the entire faculty and staff of the Department of Control and Computer Engineering at Politecnico di Torino for fostering a collaborative and stimulating academic environment that has greatly supported my research journey and personal growth

Finally, I am forever grateful to my family, whose unconditional love, patience, and constant belief in my abilities have been the bedrock of my academic journey. Their support has been both my anchor and my inspiration throughout this path.

Abstract

This thesis presents a structured methodology for testing and validating a hardware accelerator for TCUs, with a focus on fault coverage analysis using ATPG. The architecture under study consists of six DPUs designed for matrix multiplication in AI applications.

The verification flow begins with RTL simulation in QuestaSim, followed by synthesis using Synopsys DC and gate-level validation to ensure functional equivalence. ATPG is conducted using Synopsys TetraMAX, applying SA fault models to assess test coverage. Specific configuration settings are used to generate deterministic patterns and export external test vectors for further analysis.

A key contribution of this work is the classification and analysis of undetected faults, categorized as NC and NO. These represent functionally untestable faults due to limited controllability or observability. Each DPU is individually analyzed to identify common causes of test limitations, such as hardwired logic paths, reconvergent fan-outs, and logic masking. Additionally, structurally untestable faults—namely UB, UT, and DS—are documented to complete the testability assessment.

The results highlight design areas with low fault detectability and suggest improvements through enhanced scan strategies and test-point insertion. The proposed methodology offers a validated approach for testing AI-focused hardware accelerators and supports future advances in DFT.

Table of Contents

List of Figures

List of Tables

Acronyms

AI	Artificial Intelligence
ALU	Arithmetic Logic Unit
APA	Automatic Pattern Application
ASIC	Application-Specific Integrated Circuit
ATPG	Automatic Test Pattern Generation
BIST	Built-In Self-Test
CAD	Computer-Aided Design
CMOS	Complementary Metal-Oxide Semiconductor
CPU	Central Processing Unit
CUT	Circuit Under Test
CUDA	Compute Unified Device Architecture
DFT	Design for Testability
DL	Deep Learning
DPU	Dot Product Unit
DUT	Device Under Test
EDA	Electronic Design Automation
FF	Flip-Flop
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GDSII	Graphic Data System II
GPU	Graphics Processing Unit
HDL	Hardware Description Language
HLS	High-Level Synthesis
IP	Intellectual Property (core)
ISA	Instruction Set Architecture
LFSR	Linear Feedback Shift Register
LLM	Large Language Model
MUX	Multiplexer
NC	Not Covered (fault)
ND	Not Detected (fault)
Netlist	A list describing the connectivity of an electronic circuit
NN	Neural Network
NO	Not Observed (fault)

Acronyms

PPA	Power, Performance, Area
RTL	Register Transfer Level
SA	Stuck-At (fault model)
SCAN	Scan Chain-based Test Architecture
SDC	Static Design Constraint
SoC	System on Chip
STA	Static Timing Analysis
TC	Tensor Core
TCL	Tool Command Language
TCU	Tensor Core Unit
TDP	Thermal Design Power
TetraMAX	Synopsys ATPG Tool
TLB	Translation Lookaside Buffer
TMR	Triple Modular Redundancy
TP	Test Point
UUT	Unit Under Test
VCD	Value Change Dump
VHDL	VHSIC Hardware Description Language
VLSI	Very Large Scale Integration

Chapter 1 – Introduction

1.1 Research Context and Motivation

In the era of artificial intelligence, the demand for high-performance computing has surged, particularly for tasks such as image recognition, natural language processing, and autonomous decision-making. These computational workloads are characterized by extensive matrix operations, which are both data- and compute-intensive. To meet these demands, hardware accelerators such as TCUs have been developed. TCUs are specialized architectures designed to perform matrix multiplications and dot-product operations efficiently, often embedded within larger AI SoC designs.

Among the building blocks of TCUs are DPUs, which execute multiple multiplications and additions in parallel. While these units enable significant speedups in AI inference and training tasks, their complexity poses a new challenge: ensuring reliability and fault tolerance at the hardware level. With growing reliance on such accelerators in critical sectors—ranging from healthcare to autonomous vehicles—even a minor undetected fault can propagate and corrupt model outputs, potentially leading to severe consequences.

As technology nodes shrink and integration density increases, circuits become more susceptible to faults due to process variations, environmental noise, aging, or transient radiation effects. The traditional focus on performance must now be matched with robust validation strategies to ensure fault-free operation. In particular, testability evaluation and ATPG-based fault detection play a vital role in identifying weaknesses in design that could hinder fault activation or observation.

However, commercial ATPG tools face challenges in achieving full fault coverage for deep combinational datapaths typical of DPUs. Despite the application of established techniques such as stuck-at fault modeling and full-scan synthesis, many faults remain undetected. These undetected faults may fall into two main functional classes:

- Not Controlled (NC): Faults that cannot be activated due to unreachable logic paths or constant signal masking.
- Not Observed (NO): Faults that are activated but whose effects fail to propagate to observable outputs.

The identification, classification, and reasoning of these undetected faults are crucial for improving test quality and guiding future hardware design enhancements. Moreover, understanding the structural limitations, such as tied inputs, blocked nets, or redundancy, further aids in differentiating between tool-limited and functionally untestable regions of the design.

This thesis is motivated by the need to bridge the gap between hardware performance and hardware reliability in AI accelerators. It aims to develop a methodical framework for analyzing fault behavior in a real TCU design, leveraging industrial-grade tools such as

Synopsys Design Compiler, QuestaSim, and TetraMAX. Through this work, we seek not only to measure ATPG efficiency but also to gain functional insights into undetected fault mechanisms, ultimately proposing informed recommendations for improved design-for-test practices in future AI hardware.

1.2 Relevance in Scientific and Social Context

The rapid advancement of AI technologies has led to their integration into numerous applications that directly impact modern society, such as autonomous driving, intelligent medical diagnostics, natural language processing, and smart manufacturing systems. At the core of many of these applications are deep learning models that rely on vast computational resources for real-time data processing and decision-making. To support such requirements, specialized hardware accelerators—particularly (TCUs)—have emerged as essential components of modern computing architectures.

The relevance of this work lies in its focus on ensuring the functional correctness and fault resilience of TCUs, which are increasingly deployed in critical systems where safety, accuracy, and reliability are non-negotiable. A fault in the arithmetic path of a TCU, such as in one of its DPUs, could lead to silent data corruption, incorrect AI outputs, or even system failures. As a result, validating and testing these architectures is no longer a secondary consideration but a core requirement in hardware development cycles.

From a scientific perspective, this research contributes to the fields of reliable computing and EDA by proposing a structured methodology to identify and classify undetected faults in complex datapaths. It goes beyond traditional coverage metrics and delves into the functional reasoning of ATPG-resistant faults, offering insights into how design choices affect testability. This level of granularity is essential to bridge the gap between theoretical fault models and practical test coverage limitations observed in state-of-the-art tools.

Socially, this work aligns with the increasing demand for trustworthy AI systems. In domains like healthcare, finance, and transportation, the dependability of the underlying hardware directly influences public safety and confidence. By focusing on structural validation of TCUs at the gate level and analyzing the origins of undetected faults, this thesis supports the broader goal of developing transparent, safe, and accountable AI infrastructures.

In summary, the relevance of this research spans both technical advancement and societal impact. It addresses a critical need in current and future computing systems by ensuring that the acceleration of AI computation does not come at the cost of hardware reliability and safety. This dual significance reinforces the importance of the work within the academic, industrial, and social domains.

1.3 State-of-the-Art and Research Gaps

Modern AI workloads, particularly those based on deep learning, demand high-throughput computation that traditional CPUs and GPUs struggle to efficiently support. This challenge has led to the development of specialized hardware accelerators, such as TPUs and TCUs, which execute operations like matrix multiplications and dot products with optimized speed and parallelism. Among these, TCUs are widely integrated into commercial and research-grade systems due to their ability to deliver both performance and energy efficiency.

From a testing and verification perspective, existing state-of-the-art methods primarily focus on functional simulation, formal verification, and deterministic ATPG techniques. Tools such as Synopsys TetraMAX are widely used in industry to identify testable faults at the gate level using structural ATPG. However, despite their sophistication, these tools face significant limitations when applied to complex datapath architectures such as DPUs, particularly due to logic redundancy, unobservable paths, and low-controllability internal nodes.

Recent academic research has made notable progress in fault modeling, DFT, and scan-based methodologies. Nonetheless, there is a lack of comprehensive studies specifically addressing the functional analysis of undetected faults in modern AI accelerators. Most evaluations stop at reporting fault coverage percentages, without investigating why specific faults remain undetected, whether due to structural limitations or functional design constraints. Moreover, existing literature does not extensively cover the classification and root-cause reasoning of faults that are not detected by ATPG, particularly in the context of arithmetic-intensive hardware modules like those found in TCUs.

The gap thus lies in the absence of a systematic methodology for:

- Identifying and reasoning about functionally untestable faults (NC and NO) that escape structural ATPG.
- Understanding the architectural elements—such as full adders, multipliers, and rounding logic—that contribute to fault masking or unobservability.
- Providing fault classification insights that can inform future DFT enhancements or redesign decisions.

This thesis aims to address these gaps by not only quantifying undetected faults across multiple DPU instances within a TCU but also providing functional reasoning for each undetected fault class. The analysis contributes both to improving hardware test strategies and guiding designers toward more testable and fault-resilient accelerator architectures, which remain a pressing need in the current landscape of AI hardware development.

1.4 Aims and Scopes

The primary aim of this thesis is to evaluate and enhance the testability of a hardware accelerator composed of multiple DPUs, which form the core computational blocks of a TCU designed for AI applications. By employing advanced ATPG techniques using Synopsys TetraMAX, the study seeks to classify and analyze undetected faults, identify their root functional causes, and assess structural limitations in the design.

The specific objectives of the thesis are as follows:

- To synthesize and validate six combinational DPU designs using industry-standard tools such as Synopsys Design Compiler and QuestaSim.
- To perform structural ATPG analysis using TetraMAX, measuring fault coverage metrics and identifying patterns that lead to fault detection and undetection.
- To classify undetected faults into functional categories—NC and NO—and investigate their occurrence across different DPU architectures.
- To provide functional reasoning for NC and NO faults, linking them to structural features such as unobservable outputs, logic masking, and unreachable nodes.
- To analyze structural undetectable fault types, including UT, UB, and DS to contextualize the ATPG coverage results.
- To identify design-level insights that can inform improved test point insertion, fault isolation, or architectural modifications aimed at increasing overall testability.

The scope of the thesis is deliberately focused on combinational DPU logic within the TCU architecture, avoiding sequential elements or memory-based testing. It considers both the design-level flow—from RTL to synthesized netlist—and the testing environment setup, with coverage and classification analysis limited to gate-level faults in synthesized netlists.

By bridging design methodologies and fault analysis, this work offers a complete flow from synthesis to testability insight, with practical implications for designing future AI hardware accelerators that are both high-performing and robust against undetected functional faults.

1.5 Thesis Structure Overview

This thesis is structured into seven chapters, each addressing a specific aspect of the research on test generation and fault analysis for a TCU hardware accelerator.

- **Chapter 1 – Introduction** presents the motivation, context, research relevance, existing gaps, and the specific aims and structure of the thesis.
- **Chapter 2 – Background and Related Work** offers an overview of AI accelerators, reliability in VLSI, fault models, testability principles, and the tools used throughout this work.
- **Chapter 3 – Architecture of the Tensor Core Accelerator** describes the structure, operation, and implementation details of the six DPUs that form the core of the accelerator.
- **Chapter 4 – Materials and Methods** explains the synthesis flow, validation strategy, and ATPG setup using commercial EDA tools including Synopsys Design Compiler, QuestaSim, and TetraMAX.
- **Chapter 5 – Results and Discussion** details the ATPG results for each DPU, discusses generated patterns and coverage, visualizes fault types, and correlates findings to the research objectives.
- **Chapter 6 – Functional Reasoning of Undetected Faults** focuses on the classification and analysis of NC and NO faults, interprets their causes, and reviews structural undetectable fault classes.
- **Chapter 7 – Conclusion and Future Work** summarizes the key findings, highlights the research contributions and limitations, and proposes directions for further development and improvement.

Each chapter builds upon the previous one, leading to a comprehensive analysis of the testability challenges in TCU designs and the effectiveness of structured ATPG methodologies.

Chapter 2 – Background and Related Work

2.1 AI Accelerators and TCUs

The exponential growth of AI and ML workloads has driven a fundamental shift in processor design, demanding specialized hardware known as AI accelerators. These accelerators are tailored to perform highly parallelizable tasks such as matrix multiplications, convolutions, and tensor operations with improved speed and energy efficiency compared to general-purpose processors. Among the most impactful AI workloads are those involving deep learning inference and training, which rely on multiply-accumulate operations executed over large datasets.

In response to these needs, hardware designers have developed architectures that exploit fine-grained parallelism through specialized units. One of the most significant architectural innovations in this domain is the TCU, first introduced by NVIDIA and later adopted in various custom ASICs and FPGA implementations. TCUs are dedicated hardware blocks optimized for tensor-based operations, commonly involving matrix multiplications and dot products**,** which are foundational in neural networks and other AI models.

At the core of TCUs are DPUs—combinational arithmetic blocks that compute the scalar result of two input vectors. Each DPU typically integrates multipliers, adders, and logic for rounding and overflow handling, contributing to high-throughput and low-latency execution of matrix-vector computations. In modern AI accelerators, these DPUs are instantiated in parallel, forming a hierarchical compute structure that supports massive data parallelism and accelerates tensor-based workloads efficiently.

Given their complexity and critical role in AI computation, ensuring the functional correctness and testability of TCUs and their constituent DPUs is essential. Faults in these units can lead to silent data corruption or degraded inference accuracy in deployed models. Thus, structured test strategies—such as (ATPG)—are imperative for validating these designs before fabrication.

This section contextualizes the relevance of TCUs in the broader field of AI hardware and sets the stage for the detailed investigation and testing of DPU designs carried out in this thesis.

2.2 Reliability in VLSI Hardware

As VLSI technology advances into deep submicron and nanoscale regimes, ensuring the reliability of digital hardware has become increasingly challenging and critical. In AI accelerators and other high-performance systems, reliability directly impacts system correctness, safety, and long-term operational consistency, especially in mission-critical and data-intensive applications.

VLSI circuits are prone to a variety of fault mechanisms, including manufacturing defects, process variations, aging-related degradation (such as Bias Temperature Instability and Hot Carrier Injection), and environmental effects like cosmic radiation or voltage fluctuations. These issues may result in permanent faults—such as stuck-at conditions caused by defects in interconnects or transistors—or transient faults like soft errors, which occur momentarily but may still corrupt data and computations.

To address these vulnerabilities, hardware designers employ DFR and DFT methodologies. These include techniques such as redundancy, error correction codes (ECC), built-in self-test (BIST), and fault-tolerant design patterns. At the verification and validation stage, fault modeling and test generation tools play a central role in assessing a circuit's resilience against potential failures.

In the context of this thesis, reliability is assessed through structural fault analysis and ATPG-based validation of the arithmetic blocks inside DPUs. The goal is to identify faults that are functionally untestable or weakly testable using conventional patterns, and to develop insights that may inform future improvements in design architecture or test strategies.

The increasing complexity and parallelism of AI hardware, such as TCUs, make fault isolation and coverage enhancement vital not only for yield improvement but also for ensuring dependable AI computations in real-world applications.

2.3 Fault Models: Stuck-at, Transient, and Functional Faults

Fault models provide a framework to represent and simulate possible hardware failures during testing. The most widely used and industry-accepted model is the stuck-at fault model, which assumes that a signal line in the circuit is permanently stuck at a logic '0' (sa0) or logic '1' (sa1). This model is particularly effective for identifying permanent logic faults in combinational circuits.

In this thesis, all ATPG analysis was performed using the stuck-at fault model, as selected in Synopsys TetraMAX. The other available models, such as transition faults, path delay faults**,** bridging faults, and IDDQ faults, were not used, as the focus was on static combinational behavior.

The choice of the stuck-at model aligns with the objective of achieving broad structural coverage using deterministic and random pattern generation without scan-chain-based design. This ensured a practical and focused analysis of the design's static fault detectability.

2.4 Design for Testability and ATPG Techniques

DFT plays a critical role in enhancing the test efficiency of digital systems by incorporating structures or strategies that simplify the generation, application, and evaluation of test patterns. In traditional approaches, this is often achieved by inserting scan chains or boundary scan registers that provide full controllability and observability across sequential elements. However, this thesis adopts a non-scan-based approach, focusing on combinational ATPG to assess fault coverage directly at the gate level without modifying the original datapath structure.

The primary test model employed is the stuck-at fault model, which assumes that a signal line can be permanently fixed at logic '0' or '1'. ATPG aims to generate a minimal set of patterns that can detect all modeled faults, ensuring that each fault is both activated and propagated to an observable output.

For this work, Synopsys TetraMAX was configured to operate in combinational ATPG mode, which means no scan elements were inserted into the design. Instead, testability was evaluated by generating patterns that activate faults through primary inputs and propagate their effects through purely combinational logic paths. This mode is particularly relevant for data-intensive AI hardware components, such as TCUs, where the introduction of scan logic may disrupt performance or alter architectural behavior.

Additionally, internal pattern generation and random pattern injection techniques were employed to enhance fault activation in deep arithmetic structures. The ATPG configuration also enabled the classification of undetected faults into categories such as NC and NO, facilitating further analysis of functional limitations in the design.

This combinational-focused ATPG methodology provides valuable insights into the inherent testability of AI accelerators and supports the evaluation of structural robustness without relying on DFT hardware overhead.

2.5 Tools and Environments (Design Compiler, QuestaSim, TetraMAX)

This thesis employed three core EDA tools to support the full digital design and testing workflow: Synopsys Design Compiler, Siemens QuestaSim, and Synopsys TetraMAX. Each tool was selected based on its compatibility with the objectives of non-scan, combinational ATPG and its ability to handle high-complexity arithmetic logic blocks found in AI accelerators.

Design Compiler was used to perform synthesis from RTL to a gate-level netlist, ensuring logic optimization while preserving functional intent. The tool supported cell-based technology mapping, and the generated netlist formed the foundation for fault modeling and test generation in later stages.

QuestaSim was utilized for both RTL and gate-level simulation, verifying behavioral correctness and enabling waveform-based comparison through VCD exports. Its role was essential in confirming signal propagation, especially across arithmetic structures where timing alignment and gate-level mapping affect observability.

TetraMAX, configured in a combinational ATPG mode, generated deterministic patterns targeting stuck-at faults without scan chain insertion. This mode allowed internal pattern activation and propagation using controllable primary inputs and accessible internal signals. The tool also classified faults by detection status and provided detailed insights into testability gaps, particularly through NC and NO fault reporting.

The integration of these tools allowed for consistent progression across synthesis, simulation, and test generation phases, tailored specifically for combinational logic paths in TCU datapaths. This setup was crucial for exploring functional testability in realistic design environments where scan insertion is not feasible.

Chapter 3 – Architecture of the Tensor Core Accelerator

3.1 Overview of TCU Structure and Operation

TCUs are highly specialized hardware modules designed to accelerate matrix multiplication, a fundamental operation in most deep learning and AI workloads. These units achieve high throughput by leveraging data-level parallelism, where multiple operations are performed simultaneously using fixed datapath structures. TCUs are often integrated into modern AI SoCs and GPU accelerators, offering optimized pipelines for small matrix sizes (e.g., 4×4 or 8×8 GEMM operations), particularly in low-precision formats such as FP16 or INT8.

At a high level, a TCU is composed of an array of DPUs. Each DPU performs multiply-accumulate operations on input vectors and feeds its partial result to either an accumulator register or a downstream summation network. The modularity of this design allows scaling and reconfiguration based on performance and area requirements.

Internally, each DPU is made up of combinations of floating-point multipliers, floating-point adders, and, in some cases, FMA blocks. The architectural decisions — such as chaining operations or organizing logic in tree structures — significantly affect not only performance but also testability and fault coverage.

This chapter presents the six DPU architectures provided at the beginning of this thesis work. Each variant reflects a different internal datapath structure aimed at implementing the same dot product functionality with different fault and performance characteristics.

3.2 Functional Description of DPU1 to DPU6

The six DPUs analyzed in this thesis — DPU1 through DPU6 — all implement dot product computation using various combinations of arithmetic units. Although they share a common top module name (`dot_unit_core`), their internal configurations differ substantially. These architectural variations directly influence their fault activation paths, fan-in/fan-out behavior, and ultimately, their test coverage levels under ATPG analysis.

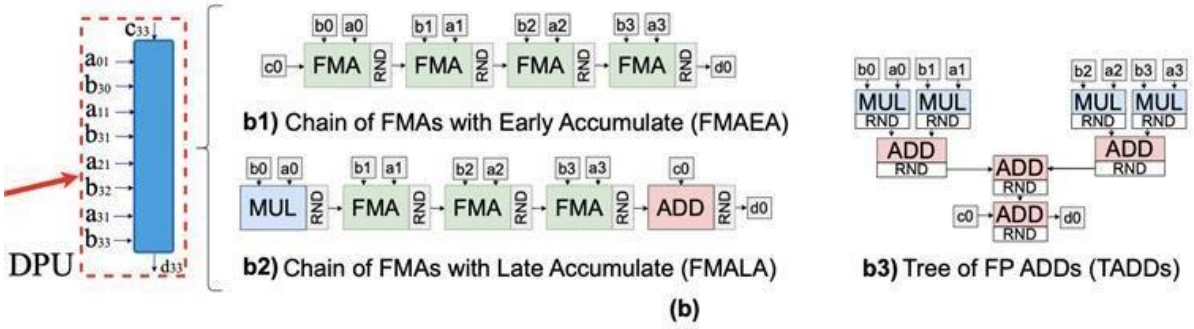

Figure 3.2 - Architectural Variations of the six Dot Product Units (DPUs) used in the Tensor Core Unit accelerator

This figure serves as a reference for the structural differences between the six designs and helps contextualize why certain units result in higher or lower test coverage during ATPG.

Below is a brief overview of each DPU's internal characteristics:

DPU1 – Baseline Multiplier-Tree Design

DPU1 consists of multiple floating-point multipliers, with their outputs directly feeding into a shared floating-point adder. This structure represents a basic chained datapath where multipliers work in parallel and results are summed sequentially. Its simplicity makes it a reference point for coverage comparisons.

DPU2 – Multiplier-Adder with Pre-Combining

This variant combines two multipliers into one adder early, before feeding results downstream. This may reduce logic depth but introduces structural overlap, affecting fault activation paths. It demonstrates slight improvements in timing but more complex testability due to merged signal cones.

DPU3 – Deep Adder Tree (TADD Structure)

DPU3 replaces the chaining approach with a balanced adder tree, where multiplier outputs are combined in multiple levels of addition. This reduces fan-out and increases parallelism. However, deeper adder networks increase the number of internal paths, introducing more fault detection complexity.

DPU4 – Pure FMA Unit

In this configuration, the entire computation is handled by a single combinational FMA (Fused Multiply-Add) unit, effectively reducing gate-level complexity but

centralizing the logic cone. The FMA may result in higher gate-level efficiency but lower fault observability due to dense internal logic.

DPU5 – Chained FMAEA Structure

DPU5 adopts a chained Fused Multiply-Add Extended Architecture (FMAEA) approach, where each FMA block accumulates results from previous stages. This configuration increases fault propagation depth but enhances throughput. It demonstrates high pattern compression in ATPG due to repeated logic structures.

DPU6 – Hybrid Multiplier-FMA-Adder Combination

The final variant, DPU6, combines standalone multipliers, adders, and an FMA block. It aims to balance propagation paths and maintain both modularity and fault coverage. Among all DPUs, DPU6 shows the best testability trade-off, achieving the highest fault coverage during ATPG.

Each of these architectures was synthesized, validated, and tested independently using a standardized tool flow. The differences observed during ATPG analysis, including undetected fault categories, are directly linked to these architectural design choices.

3.3 Combinational Design Assumptions and Implementation Details

All six DPUs (DPU1–DPU6) analyzed in this thesis were designed and synthesized as purely combinational logic blocks. This design choice was intentional to allow for simplified validation and straightforward application of combinational ATPG techniques without the complexities introduced by sequential behavior, such as clock domains or state retention.

The RTL implementations did not contain flip-flops or memory elements, and were structured using combinational operators such as multipliers, adders, inverters, and logic gates. Each DPU design was isolated and synthesized independently using Synopsys Design Compiler with a 15nm technology library, under uniform constraints to ensure fair comparative evaluation.

This design approach enabled efficient fault modeling under the stuck-at fault model, where all faults were treated as static logic conditions. The absence of latches or registers simplified controllability and observability analysis and allowed precise correlation between structural elements and ATPG results.

This section sets the foundation for the synthesis, simulation, and testing procedures described in Chapter 4.

Chapter 4 – Materials and Methods

4.1 Research Design and Rationale

The methodology adopted in this thesis is rooted in a structural testing approach applied to combinational hardware blocks. The goal is to analyze the testability and fault behavior of six DPUs variants that serve as core arithmetic elements in TCUs. Rather than relying on software-level validation or redundancy-based resilience, this research focuses on gate-level fault analysis using ATPG techniques.

The decision to isolate combinational logic stems from two primary motivations:

1. It eliminates the confounding effects of sequential dependencies and timing constraints.
2. It enables a clearer evaluation of datapath-driven testability, independent of control logic or registers.

This black-box methodology treats each DPU as a self-contained datapath accelerator. The analysis is then conducted based on its synthesis footprint and fault observability under stuck-at fault modeling.

4.2 Input Data, RTL Files, and Synthesis Library Details

Each DPU was implemented in VHDL and wrapped in a top module named `dot_unit_core`, designed to perform dot product operations using different arithmetic structures (e.g., parallel multipliers, cascaded adders, fused multiply-add). Each unit consisted of:

- A primary `.vhd` file for RTL logic,
- Optional VHDL testbenches or wrappers,
- Internal instantiations of adders, multipliers, and custom arithmetic units.

All DPUs were synthesized using the NanGate 15nm Open Cell Library. No clock elements or registers were included; therefore, synthesis did not require timing constraints.

4.3 RTL to Gate-Level Flow using Design Compiler

The synthesis phase was performed using Synopsys Design Compiler (DC Shell) to convert each DPU's VHDL RTL design into a gate-level Verilog netlist. This process ensures compatibility with downstream testing tools like TetraMAX.

Each DPU variant (DPU1–DPU6) was synthesized individually using a structured TCL script. The process involved setting up library paths, parsing VHDL files, elaborating the top-level module (`dot_unit_core`), compiling the design, and exporting reports and netlists.

The synthesis script included directives for:

- Library linking with NanGate 15nm OCL cells,
- Parsing RTL VHDL sources,
- Enabling Verilog-compliant naming,
- Exporting both `.v` and `.db` netlists,
- Reporting area, timing, and power metrics,
- Saving the full script for reproducibility.

To visually represent the RTL-to-netlist flow, the following figure summarizes the synthesis process:

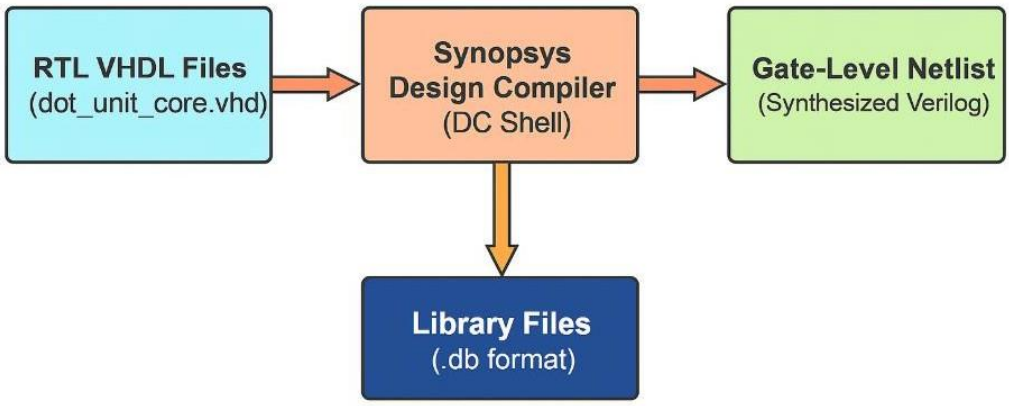

Figure 4.3 - RTL to Gate-Level Synthesis Flow in Design Compiler

4.4 Validation with QuestaSim (RTL vs Netlist Signal Matching)

Following synthesis, it was essential to verify that each DPU's gate-level netlist functionally matched its original RTL behavior. This validation step ensured that the test vectors generated during ATPG would interact correctly with the logic structure and not be blocked due to signal mismatches or synthesis-related modifications.

The Mentor QuestaSim simulation tool was used to perform this RTL-to-netlist correlation. The validation methodology included the following:

Step-by-Step Process:

1. Stimulus Preparation:
    - A common VHDL testbench or stimuli waveform was developed and applied to both RTL and synthesized gate-level versions of each DPU.
2. Simulation of RTL Design:
    - The RTL VHDL files for each DPU were simulated to generate a reference waveform for signal behavior, inputs, and outputs.
3. Simulation of Synthesized Netlist:
    - The gate-level Verilog netlist (output from Synopsys Design Compiler) was instantiated into a testbench with identical stimuli and simulated under the same conditions.
    - The corresponding standard cell `.db` library was included for gate-level simulation accuracy.
4. Signal Comparison and Debugging:
    - Waveforms were analyzed in the QuestaSim GUI to visually compare output signals from both RTL and synthesized versions.
    - Particular attention was given to port renaming, signal propagation delays, and any nets missing due to optimization.
    - When mismatches were found (e.g., due to signal renaming during synthesis), manual corrections were applied to the testbench or netlist structure to ensure semantic alignment.
5. Final Check for ATPG Compatibility:
    - The validated netlist, now verified to match the RTL behavior, was confirmed as suitable for TetraMAX ATPG.
    - This check ensured full controllability and observability of input-output signals within the ATPG environment.

This validation phase was critical to prevent undetected faults arising from incorrect signal mapping or synthesis errors. Each DPU was individually verified to maintain a fault-free simulation environment.

4.5 Test Strategy and ATPG Setup in TetraMAX

To validate the testability and fault detection capability of each DPU architecture, the synthesized gate-level netlists were imported into Synopsys TetraMAX for ATPG. The ATPG process was conducted consistently across all six DPUs following a systematic GUI-based configuration approach.

Setup Steps:

1. Read Netlist and Library Files:
    - Loaded the synthesized Verilog netlist file (`synthesized_netlist.v`) for each DPU.
    - Imported the standard cell library file (`.db` format) using the GUI.
2. Design Build and Configuration:
    - Set fault model to Stuck-at (SA0, SA1).
    - Enabled options for modeling faults at cell outputs, suppressing untestable faults, and defining dominances.
3. DRC (Design Rule Check):
    - Verified the design topology and ensured that test protocol parameters were correctly set (e.g., oscillation passes, skew, clock constraints).
    - Enabled multi-capture/load settings and transparent latches for accurate modeling.
4. ATPG Execution:
    - Ran the Basic Scan ATPG with high abort limits (up to 5000) to maximize the number of generated patterns.
    - Selected internal combinational patterns in the initial phase.
    - Enabled pattern merging and random pattern injection to further increase fault coverage, especially for hard-to-detect faults.
5. Pattern Export:
    - External test patterns were saved in `.stil` format for future use and analysis.
    - Coverage and fault summary reports were generated per DPU.

Each DPU was tested individually using the above process. This structured and GUI-driven approach enabled effective visualization and refinement of ATPG configurations.

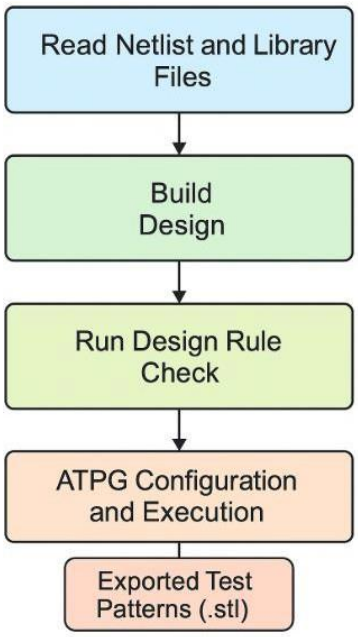

Figure 4.5 - ATPG GUI Setup Flowchart

A visual overview of the TetraMAX ATPG process used for all DPUs. It highlights the sequential steps of loading netlists and libraries, validating design rules, generating combinational scan patterns, and exporting. still files.

4.6 ATPG Configuration and Applied Techniques

The ATPG configuration for all DPUs was unified and fine-tuned for maximum stuck-at fault coverage. Key settings included:

Setting	Value/Option
Fault Model	Stuck-at (SA0, SA1)
Capture Mode	Zero-cycle
Abort Limit	High (≥ 3000)
Pattern Compression	Enabled
Random Pattern Injection	Enabled (combinational)
Test Point Insertion	Disabled
Pattern Output Format	External (.still)
Fault Classification	DT, ND, NC, NO, UD (CSV exported)

Table 4.6 - TetraMAX ATPG Configuration Parameters and Output Settings

Each DPU was processed as follows:

- Initially, internal combinational ATPG patterns were generated with a high abort limit of 3000**.**
- To increase coverage**,** internal random combinational patterns were injected, again with a high abort threshold.
- Patterns were stored and simulated back with QuestaSim to ensure response integrity.

This multi-step ATPG methodology enabled the detection of deep combinational faults while preserving structural integrity. It also facilitated a detailed breakdown of functionally undetected faults, later analyzed in Chapter 6.

Chapter 5 – Results and Discussion

This chapter presents a detailed analysis of the results obtained from the ATPG process applied to the six DPUs that constitute the core of the Tensor Core accelerator. The evaluation covers key metrics such as total fault count, detected versus undetected faults, ATPG-generated test patterns, and categorized fault types including NC, NO, and structurally untestable faults. In addition to reporting numerical coverage statistics, the chapter offers functional reasoning for undetected faults and highlights their correlation with specific structural and logical elements within each DPU. This analysis not only validates the applied ATPG methodology but also identifies opportunities for enhancing design testability in future TCU architectures.

5.1 Summary of ATPG Results

The table below summarizes the total number of faults per DPU, the number of faults detected, those classified as undetectable, and the remaining not detected. Moreover, the undetected faults have been further divided into two functionally significant categories: NC and NO. These functional classifications help identify testability challenges due to architectural or signal control limitations.

DPU	Total Faults	Detected Faults (DT)	Undetectable (UD)	Not Detected (ND)	NC	NO	Coverage (%)
DPU1	88,880	78,974	144	9,762	7,611	1,779	89.00%
DPU2	89,664	79,721	144	9,799	5,506	1,496	89.05%
DPU3	89,664	79,877	144	9,643	4,473	1,307	89.23%
DPU4	84,856	81,517	368	2,971	4,376	1,317	96.48%
DPU5	107,632	100,738	252	6,634	5,281	1,353	93.81%
DPU6	84,856	82,343	368	2,145	1,776	369	97.46%

Table 5.1 - Summary of ATPG Results for DPU1–DPU6

5.2 Analysis of Generated Patterns and Coverage

The number of test patterns generated by the ATPG engine (both internal and external) reflects the complexity and testability of each DPU. A higher pattern count generally indicates increased logic depth or structural irregularities affecting controllability and observability.

DPU	Internal Patterns	External Patterns	Final Fault Coverage (%)
DPU1	1,915	1,942	89.00%
DPU2	1,917	1,960	89.05%
DPU3	1,982	562	89.23%
DPU4	2,274	2,296	96.48%
DPU5	2,201	2,251	93.81%
DPU6	2,512	2,570	97.46%

Table 5.2 - Number of Generated Patterns per DPU and Final Coverage

These results indicate that the use of both internal combinational patterns and externally applied scan patterns was essential to achieving high fault coverage, particularly for structurally complex or deeply pipelined DPUs like DPU5 and DPU6.

In DPU3, the significantly lower number of external patterns compared to other DPUs highlights its relatively higher inherent testability. Conversely, the higher external pattern count for DPU6 demonstrates the additional effort required by the ATPG tool to ensure full controllability and observability of internal signals.

5.3 Visualization of Faults

To better understand the distribution and nature of faults across the six Dot Processing Units (DPUs), this section visualizes the fault classification results. These visual aids highlight the effectiveness of ATPG techniques and help identify fault-prone areas in each architecture.

5.3.1 Fault Distribution per DPU

A bar chart (Figure 5.1) can be introduced to illustrate the number of total faults, detected faults, UD, and ND per DPU. This provides an immediate visual comparison of test quality across all six DPUs.

This figure visualizes the breakdown of total, detected, undetectable, and not detected faults for each DPU, allowing for a quick comparative analysis of detection efficiency across designs.

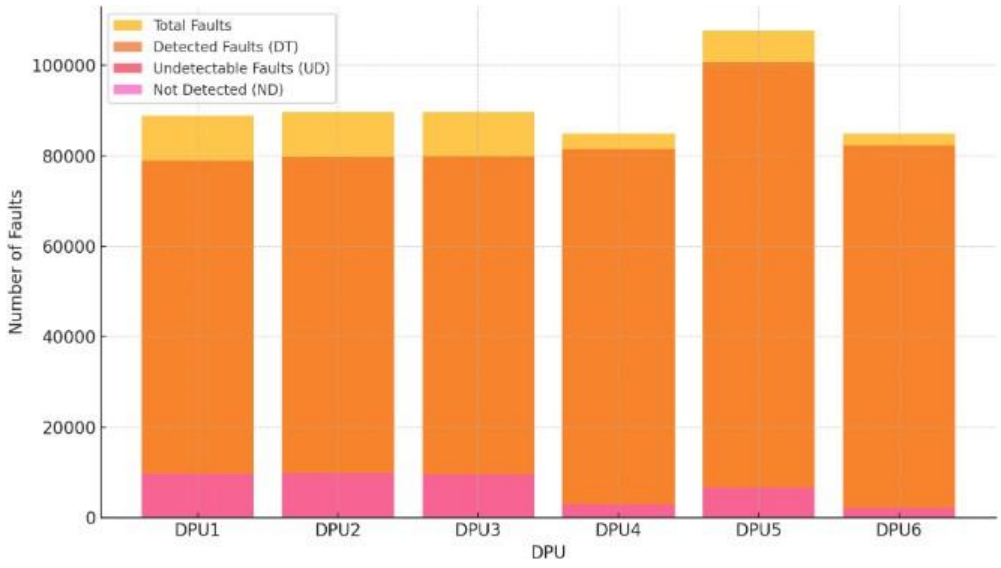

Figure 5.3.1 - Fault Classification Summary per DPU

5.3.2 Functional Classification of Not Detected Faults

Further insight can be gained by breaking down the ND faults into the NC and NO categories. Figure 5.2 will show a stacked bar chart to visually represent the share of NC and NO in each DPU's ND group.

This stacked bar chart separates Not Detected faults into functional categories: NC and NO, helping to highlight the root causes behind undetected faults.

This stacked bar chart separates Not Detected faults into functional categories: NC and NO, helping to highlight the root causes behind undetected faults.

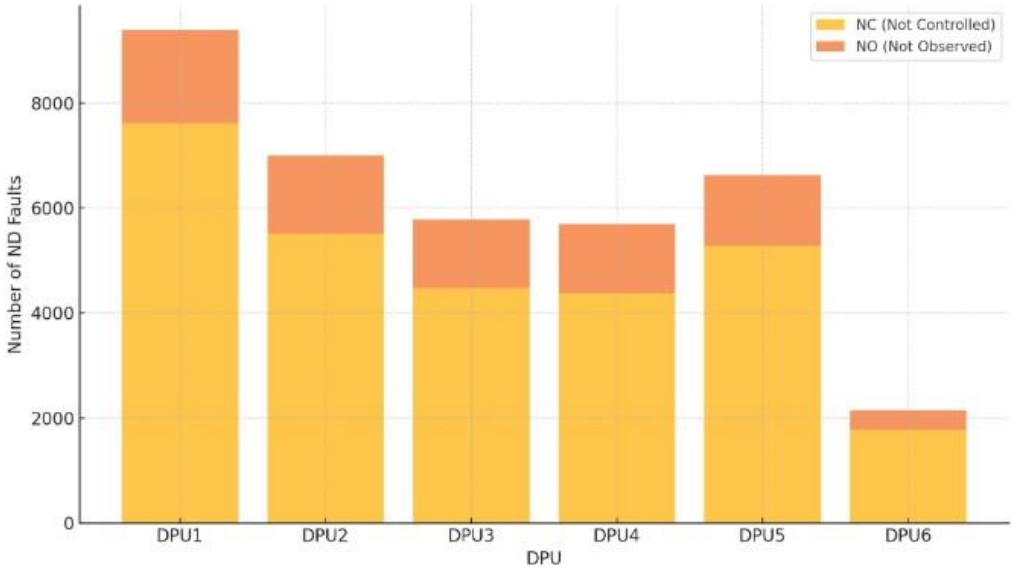

Figure 1.3.2 - Functional Breakdown of ND Faults

5.3.3 ATPG Pattern Count Analysis

Figure 5.3 presents the number of internal and external scan patterns generated for each DPU. This chart complements the numerical data in Table 5.2 and reveals how pattern count correlates with architecture complexity and coverage effectiveness.

This chart compares the number of internally and externally generated scan patterns used to achieve maximum fault coverage for each DPU.

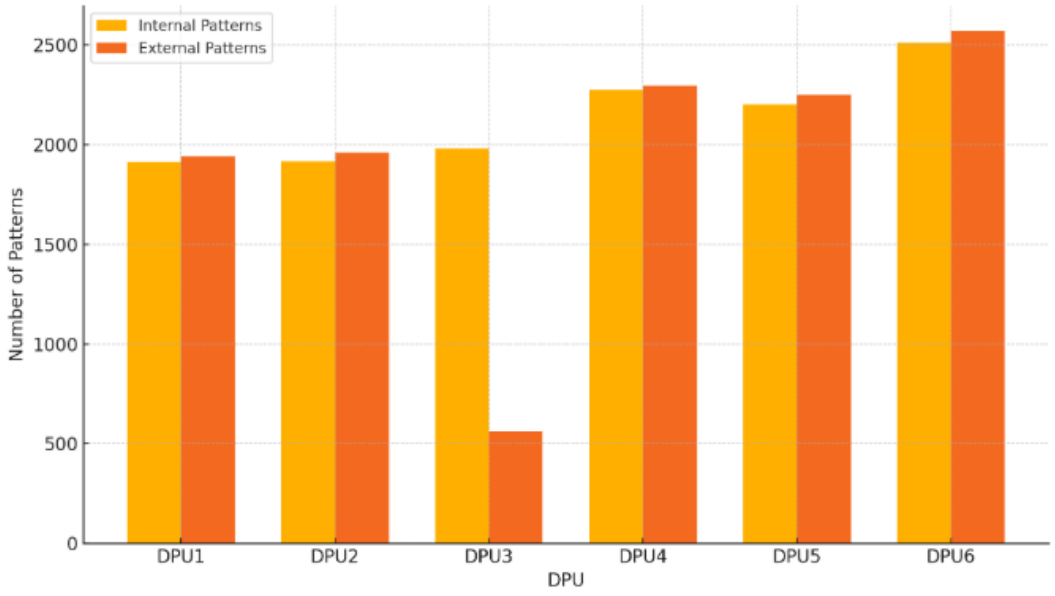

Figure 5.3.3 - Internal vs External Pattern Counts per DPU

5.4 Interpretation and Relevance to Research Objectives

The presented ATPG results across all six DPUs have revealed valuable insights into both the testability and inherent limitations of the Tensor Core accelerator design. The significant variation in coverage statistics among the DPUs highlights how architectural and structural factors affect fault activation and propagation.

In particular, DPU, DPU2, and DPU3 exhibited a high volume of NO faults, typically exceeding 4,000 entries per unit. These faults, although activated internally, failed to propagate to an observable output. A detailed analysis shows that these issues are predominantly linked to complex arithmetic logic within full adders and multiplier submodules, where reconvergent paths, output masking, and limited scan visibility prevent full observability. These patterns underscore the importance of enhancing observation logic—potentially through scan chain expansion or improved logic partitioning—to improve fault detection coverage.

Conversely, the NC faults, while fewer in number, often reveal hard-to-reach areas in the design, particularly around isolated logic elements, inverters, and rounding modules. Units like DPU4 and DPU6, which displayed relatively low NC fault counts, suggest structurally more accessible regions, albeit still challenged by propagation issues (reflected in the NO faults). On the other hand, DPU5, while showing modest NC numbers, retained one of the highest counts of NO faults, indicating that fault observability is a greater challenge than fault activation in most DPUs.

These findings are further substantiated by the analysis of structural undetectable faults, which include UB, UT, DS, and AN. The high number of DS faults, particularly in DPU5 and DPU6, indicates the ATPG tool's capacity to handle a large portion of fault modeling through simulation, but also emphasizes the gap left by functional issues, not attributable to tool limitations.

Ultimately, these results align with the research goal of identifying functionally untestable segments in AI accelerator architectures. The mixed presence of NC and NO faults across all DPUs demonstrates that while some limitations are structural, many are inherent to the functional design decisions, such as input/output multiplexing, arithmetic structure depth, and internal routing.

By focusing the fault analysis on NC and NO categories and excluding structurally untreatable faults (UB, UT, AN), this study isolates areas where design-for-test improvements could be most effective. These may include reconfigurable logic to bypass unreachable gates, improved scan path integration, or architectural redesigns to enhance internal observability. The consistent methodology applied across all six DPUs ensures that the conclusions are systematic, reproducible, and scalable to other AI hardware modules with similar complexity.

This combined result-and-discussion approach thus directly supports the thesis aim of enabling a functionally guided enhancement strategy for AI hardware testability.

Chapter 6 – Functional Analysis about Undetected Faults

6.1 Classification of NO and NC Faults

During the ATPG process using Synopsys TetraMAX, various fault outcomes were recorded for each DPU. These outcomes were categorized based on their detectability and underlying cause. Among them, only a subset of faults remained not detected and were selected for functional analysis. Faults classified as structurally undetectable or limited by the tool's capability were excluded from this phase.

This section focuses exclusively on the two types of undetected faults attributed to functional causes:

- Not Controlled (NC): These faults could not be activated during the ATPG process due to the inability to apply the necessary logic value to the fault site. This is often the result of unreachable logic paths, constant signal masking, or isolated portions of the design that are not functionally sensitizable.
- Not Observed (NO): These faults were successfully activated at their origin, but the resulting erroneous logic value failed to propagate to any primary output or observation point. Causes include reconvergent fanouts, logic masking, or the absence of an observable path to external ports.

The following structural classes are not analyzed in this chapter:

- Detected by Simulation (DS): Faults that are already detected and are therefore not included in the set of not detected faults.
- Undetectable-Tied, Blocked, Redundant (UT, UB, UR): These are structurally untestable faults caused by constant signal values, redundant logic, or physical design limitations.
- ATPG Untestable (AN): Faults that are inherently untestable due to tool limitations, unmodeled behavior, or synthesis constraints.

For the purpose of this analysis, only NC and NO faults are included, as they represent realistic opportunities for testability improvement through functional design enhancements. Each DPU's exported `.csv` report includes the complete list of NC and NO faults, detailing the following information:

- Fault Type: The specific fault condition (stuck-at-0 or stuck-at-1)
- Class: The classification type (NO or NC)
- Location: The logic gate or signal instance where the fault occurs
- Cell Type: The type of standard cell affected
- Details: Additional fault context, such as SCOAP controllability/observability or fault instance metadata

6.1.1 DPU1 – Functional Analysis of Undetected Faults

In the case of DPU1, ATPG using Synopsys TetraMAX resulted in a total of 1,566 NC faults and 4,075 NO faults. These faults represent the subset of undetected issues that arise from functional design limitations rather than structural or tool-based constraints.

The NC faults indicate points in the design where the ATPG engine was unable to sensitize the fault due to inaccessibility or masking at the logic input level. These often occur in logic cones driven by constant signals, isolated sub-modules, or functionally unreachable nodes. On the other hand, NO faults were successfully sensitized but failed to propagate to any observable output, suggesting issues such as reconvergent fan-out, output masking, or lack of scan visibility.

A representative sample of 10 faults from each category (NC and NO) is shown in the following table. These entries are selected from the top of the fault list to illustrate typical scenarios encountered in DPU1. The full fault lists are included in the appendix.

Fault Type	Class	Location	Cell Type	Details
sa0	NC	FMUL0/U3/ZN	INV_X1	(1: 7010/0/0)
sa0	NC	FMUL0/U4/ZN	INV_X1	(2: 6604/0/0)
sa0	NC	FMUL0/U7/ZN	INV_X1	(3: 6630/0/0)
sa0	NC	FMUL0/U8/ZN	INV_X1	(4: 6631/0/0)
sa0	NC	FMUL0/U9/ZN	INV_X1	(5: 6601/0/0)
sa0	NC	FMUL0/U10/ZN	INV_X1	(6: 6628/0/0)
sa1	NC	FMUL0/U13/ZN	INV_X1	(7: 6629/1/0)
sa1	NC	FMUL0/U15/ZN	INV_X1	(8: 6614/1/0)
sa0	NC	FMUL0/U16/ZN	INV_X1	(9: 6615/0/0)
sa0	NC	FMUL0/U253/B1	OAI22_X1	(10: 7162/0/1)
sa0	NO	FMUL2/U432/A2	NOR2_X1	(1: 20576/0/2)
sa0	NO	FMUL0/mult_52/S1_22_0/S	FA_X1	(2: 6372/0/0)
sa1	NO	FMUL0/mult_52/S1_22_0/S	FA_X1	(3: 6372/1/0)
sa0	NO	FMUL0/mult_52/S1_21_0/S	FA_X1	(4: 6363/0/0)
sa1	NO	FMUL0/mult_52/S1_21_0/S	FA_X1	(5: 6363/1/0)
sa0	NO	FMUL0/mult_52/S1_20_0/S	FA_X1	(6: 6355/0/0)
sa1	NO	FMUL0/mult_52/S1_20_0/S	FA_X1	(7: 6355/1/0)
sa0	NO	FMUL0/mult_52/S1_19_0/S	FA_X1	(8: 6344/0/0)
sa1	NO	FMUL0/mult_52/S1_19_0/S	FA_X1	(9: 6344/1/0)
sa0	NO	FMUL0/mult_52/S1_18_0/S	FA_X1	(10: 6335/0/0)

Table 6.1.1 - Sample of NC and NO Faults in DPU1

6.1.2 DPU2 – Functional Analysis of Undetected Faults

For DPU2, ATPG with Synopsys TetraMAX resulted in 1,582 NC faults and 4,124 NO faults. These faults reflect functional limitations in testability, where NC faults could not be sensitized due to unreachable or masked logic paths, while NO faults failed to propagate to any observable output.

Most NC faults occurred in inverter structures within the FMUL0 module, suggesting low controllability. NO faults were concentrated in arithmetic logic such as full adders and NOR gates, indicating challenges in fault observability.

A representative set of 10 NC and 10 NO faults is listed below. The complete fault lists are available in the appendix.

Fault Type	Class	Location	Cell Type	Details
sa0	NC	FMAC0/FMUL0/U3/ZN	INV_X1	(1: 7042/0/0)
sa0	NC	FMAC0/FMUL0/U4/ZN	INV_X1	(2: 6618/0/0)
sa0	NC	FMAC0/FMUL0/U7/ZN	INV_X1	(3: 6648/0/0)
sa0	NC	FMAC0/FMUL0/U8/ZN	INV_X1	(4: 6649/0/0)
sa0	NC	FMAC0/FMUL0/U9/ZN	INV_X1	(5: 6615/0/0)
sa0	NC	FMAC0/FMUL0/U10/ZN	INV_X1	(6: 6646/0/0)
sa0	NC	FMAC0/FMUL0/U11/ZN	INV_X1	(7: 6647/1/0)
sa1	NC	FMAC0/FMUL0/U15/ZN	INV_X1	(8: 6630/1/0)
sa0	NC	FMAC0/FMUL0/U16/ZN	INV_X1	(9: 6631/0/0)
sa0	NC	FMAC0/FMUL0/U253/B1	OAI22_X1	(10: 7213/0/1)
sa0	NO	FMAC0/FMUL0/U431/A2	NOR2_X1	(1: 6202/0/2)
sa0	NO	FMAC1/FMUL0/U431/A2	NOR2_X1	(2: 13190/0/2)
sa0	NO	FMAC2/FMUL0/U431/A2	NOR2_X1	(3: 20178/0/2)
sa0	NO	FMAC3/FMUL0/U431/A2	NOR2_X1	(4: 27166/0/2)
sa0	NO	FMAC0/FMUL0/mult_52/S1_22_0/S	FA_X1	(5: 6389/0/0)
sa1	NO	FMAC0/FMUL0/mult_52/S1_22_0/S	FA_X1	(6: 6389/1/0)
sa0	NO	FMAC0/FMUL0/mult_52/S1_21_0/S	FA_X1	(7: 6380/0/0)
sa1	NO	FMAC0/FMUL0/mult_52/S1_21_0/S	FA_X1	(8: 6380/1/0)
sa0	NO	FMAC0/FMUL0/mult_52/S1_20_0/S	FA_X1	(9: 6372/0/0)
sa1	NO	FMAC0/FMUL0/mult_52/S1_20_0/S	FA_X1	(10: 6372/1/0)

Table 6.1.2 - Sample of NC and NO Faults in DPU2

6.1.3 DPU3 – Functional Analysis of Undetected Faults

For DPU3, ATPG with Synopsys TetraMAX identified 1,559 NC faults and 4,073 NO faults. These functionally undetected faults stem from controllability and observability limitations within the design.

The NC faults mainly occurred in inverter gates within FMUL0, caused by unreachable or masked logic paths that prevented fault activation. Meanwhile, NO faults were found in NOR and full adder cells across FMAC and FMUL modules—faults that were sensitized internally but failed to propagate to observable outputs, likely due to reconvergent logic or insufficient scan paths.

The following table presents a representative sample of 10 faults from each category. Complete lists are included in the appendix.

Fault Type	Class	Location	Cell Type	Details
sa0	NC	FMUL0/U3/ZN	INV_X1	(1: 7040/0/0)
sa0	NC	FMUL0/U4/ZN	INV_X1	(2: 6616/0/0)
sa0	NC	FMUL0/U7/ZN	INV_X1	(3: 6646/0/0)
sa0	NC	FMUL0/U8/ZN	INV_X1	(4: 6647/0/0)
sa0	NC	FMUL0/U9/ZN	INV_X1	(5: 6613/0/0)
sa0	NC	FMUL0/U10/ZN	INV_X1	(6: 6644/0/0)
sa1	NC	FMUL0/U11/ZN	INV_X1	(7: 6645/1/0)
sa1	NC	FMUL0/U15/ZN	INV_X1	(8: 6628/1/0)
sa0	NC	FMUL0/U16/ZN	INV_X1	(9: 6629/0/0)
sa0	NC	FMUL0/U253/B1	OAI22_X1	(10: 7210/0/1)
sa0	NO	FMUL0/U431/A2	NOR2_X1	(1: 29542/0/2)
sa0	NO	FMAC0/FMUL0/U431/A2	NOR2_X1	(2: 13164/0/2)
sa0	NO	FMAC1/FMUL0/U431/A2	NOR2_X1	(3: 20782/0/2)
sa0	NO	FMAC2/FMUL0/U431/A2	NOR2_X1	(4: 27770/0/2)
sa0	NO	FMUL0/mult_52/S1_22_0/S	FA_X1	(5: 6387/0/0)
sa1	NO	FMUL0/mult_52/S1_22_0/S	FA_X1	(6: 6387/1/0)
sa0	NO	FMUL0/mult_52/S1_21_0/S	FA_X1	(7: 6378/0/0)
sa1	NO	FMUL0/mult_52/S1_21_0/S	FA_X1	(8: 6378/1/0)
sa0	NO	FMUL0/mult_52/S1_20_0/S	FA_X1	(9: 6370/0/0)
sa1	NO	FMUL0/mult_52/S1_20_0/S	FA_X1	(10: 6370/1/0)

Table 1.1.3 - Sample of NC and NO Faults in DPU3

6.1.4 DPU4 – Functional Analysis of Undetected Faults

In DPU4, ATPG using Synopsys TetraMAX identified 74 NC faults and 1,576 NO faults. While the NC fault count is relatively low, it still highlights functional gaps in controllability and observability within the design.

NC faults are mostly located at the CI of full adder cells (FA_X1) in the RoundingAdder module, where limited controllability may result from logic isolation or hardwired paths. NO faults are found across OAI21 and OR gate configurations in input and output routing structures, likely impacted by logic masking or inadequate scan visibility.

A representative sample of 10 faults from each class is shown below. The complete lists are included in the appendix.

Fault Type	Class	Location	Cell Type	Details
sa0	NC	FMUL0/FlopocoFPMult/RoundingAdder/add_1_root_add_385_2/U1_32/CI	FA_X1	(1: 25400/0/1)
sa0	NC	FMUL0/FlopocoFPMult/RoundingAdder/add_1_root_add_385_2/U1_31/CI	FA_X1	(2: 25398/0/1)
sa0	NC	FMUL0/FlopocoFPMult/RoundingAdder/add_1_root_add_385_2/U1_30/CI	FA_X1	(3: 25396/0/1)
sa0	NC	FMUL0/FlopocoFPMult/RoundingAdder/add_1_root_add_385_2/U1_29/CI	FA_X1	(4: 25394/0/1)
sa0	NC	FMUL0/FlopocoFPMult/RoundingAdder/add_1_root_add_385_2/U1_28/CI	FA_X1	(5: 25392/0/1)
sa0	NC	FMUL0/FlopocoFPMult/RoundingAdder/add_1_root_add_385_2/U1_27/CI	FA_X1	(6: 25390/0/1)
sa0	NC	FMUL0/FlopocoFPMult/RoundingAdder/add_1_root_add_385_2/U1_26/CI	FA_X1	(7: 25388/0/1)
sa0	NC	FMUL0/FlopocoFPMult/RoundingAdder/add_1_root_add_385_2/U1_25/CI	FA_X1	(8: 25386/0/1)
sa0	NC	FMUL0/FlopocoFPMult/RoundingAdder/add_1_root_add_385_2/U1_24/CI	FA_X1	(9: 25384/0/1)
sa0	NC	FMUL1/FlopocoFPMult/RoundingAdder/add_1_root_add_385_2/U1_32/CI	FA_X1	(10: 26043/0/1)
sa1	NO	FMUL0/OutputR/U34/A2	OAI21_X1	(1: 25421/1/1)
sa1	NO	FMUL1/OutputR/U34/A2	OAI21_X1	(2: 26064/1/1)
sa1	NO	FMAC2/OutputR/U34/A2	OAI21_X1	(3: 27466/1/1)
sa1	NO	FMAC3/FMUL0/OutputR/U34/A2	OAI21_X1	(4: 28109/1/1)
sa0	NO	FMUL0/InputX/U23/A3	OR3_X1	(5: 398/0/3)
sa0	NO	FMUL0/InputX/U23/A2	OR3_X1	(6: 398/0/2)
sa0	NO	FMUL0/InputX/U23/A1	OR3_X1	(7: 398/0/1)
sa1	NO	FMUL0/InputX/U23/A1	OR3_X1	(8: 398/1/0)
sa0	NO	FMUL0/InputX/U22/A3	OR4_X1	(9: 399/0/3)
sa0	NO	FMUL0/InputX/U22/A2	OR4_X1	(10: 399/0/2)

Table 6.1.4 - Sample of NC and NO Faults in DPU4

6.1.5 DPU5 – Functional Analysis of Undetected Faults

In DPU5, the ATPG process identified 110 NC and 3,931 NO faults. Despite the low number of NC faults, the high count of NO faults indicates significant observability issues in the design.

NC faults mainly appear in the normalization and control logic, often in NOR and NAND gates, pointing to poor controllability or inactive paths. NO faults, on the other hand, dominate in the RightShifterComponent and NOR/AOI logic, where fault effects are generated but fail to reach observable outputs—suggesting masking or reconvergent logic.

Below is a sample of 10 faults from each class. Full listings are available in the appendix.

Fault Type	Class	Location	Cell Type	Details
sa0	NC	FMAC0/NormalizationShifter/U1007/ZN	NOR2_X1	(1: 29637/0/0)
sa0	NC	FMAC0/NormalizationShifter/U243/ZN	NOR2_X1	(2: 29638/0/0)
sa1	NC	FMAC0/IEEEFPFMA_8_23_comb_uid2LeadingZeroCounter/U49/ZN	NAND2_X1	(3: 27752/1/0)
sa1	NC	FMAC0/IEEEFPFMA_8_23_comb_uid2LeadingZeroCounter/U57/ZN	NAND2_X1	(4: 27758/1/0)
sa1	NC	FMAC0/IEEEFPFMA_8_23_comb_uid2LeadingZeroCounter/U67/ZN	NAND2_X1	(5: 27764/1/0)
sa1	NC	FMAC0/IEEEFPFMA_8_23_comb_uid2LeadingZeroCounter/U71/ZN	NAND2_X1	(6: 27770/1/0)
sa1	NC	FMAC0/NormalizationShifter/U912/ZN	AOI22_X1	(7: 29219/1/0)
sa0	NC	FMAC0/NormalizationShifter/U563/ZN	AOI22_X1	(8: 29225/1/0)
sa0	NC	FMAC0/NormalizationShifter/U133/A1	AOI22_X1	(9: 29043/0/1)
sa0	NC	FMAC0/NormalizationShifter/U137/A1	AOI22_X1	(10: 29096/0/1)
sa0	NO	FMAC0/RightShifterComponent/U5/ZN	OAI22_X1	(1: 29587/0/0)
sa1	NO	FMAC0/U62/ZN	NOR4_X1	(2: 29820/1/0)
sa0	NO	FMAC0/U57/ZN	NOR3_X1	(3: 29791/1/0)
sa1	NO	FMAC0/U58/ZN	NOR2_X1	(4: 29772/1/0)
sa0	NO	FMAC0/U59/ZN	NOR3_X1	(5: 29793/1/0)
sa1	NO	FMAC0/U60/ZN	NOR2_X1	(6: 29770/1/0)
sa1	NO	FMAC0/U55/B2	AOI22_X1	(7: 29831/1/2)
sa1	NO	FMAC0/U52/ZN	NOR3_X1	(8: 29782/1/0)
sa1	NO	FMAC0/U53/ZN	NOR2_X1	(9: 29806/1/0)
sa0	NO	FMAC0/U56/ZN	NAND4_X1	(10: 29812/0/0)

Table 6.1.5 - Sample of NC and NO Faults in DPU5

6.1.6 DPU6 – Functional Analysis of Undetected Faults

For DPU6, Synopsys TetraMAX ATPG detected 42 NC faults and 1,179 NO faults. NC faults primarily affected CI pins of full adder cells in the RoundingAdder module, indicating limited controllability due to internal logic isolation. NO faults were mostly located in output routing and input multiplexer paths, including OAI21, OR3, and OR4 gates, suggesting blocked fault propagation and poor observability.

A representative sample of 10 NC and 10 NO faults is presented below. Complete data is available in the appendix.

Fault Type	Class	Location	Cell Type	Details
sa0	NC	FMUL0/FlopocoFPMult/RoundingAdder/add_1_root_add_385_2/U1_32/CI	FA_X1	(1: 25400/0/1)
sa0	NC	FMUL0/FlopocoFPMult/RoundingAdder/add_1_root_add_385_2/U1_31/CI	FA_X1	(2: 25398/0/1)
sa0	NC	FMUL0/FlopocoFPMult/RoundingAdder/add_1_root_add_385_2/U1_30/CI	FA_X1	(3: 25396/0/1)
sa0	NC	FMUL0/FlopocoFPMult/RoundingAdder/add_1_root_add_385_2/U1_29/CI	FA_X1	(4: 25394/0/1)
sa0	NC	FMUL0/FlopocoFPMult/RoundingAdder/add_1_root_add_385_2/U1_28/CI	FA_X1	(5: 25392/0/1)
sa0	NC	FMUL0/FlopocoFPMult/RoundingAdder/add_1_root_add_385_2/U1_27/CI	FA_X1	(6: 25390/0/1)
sa0	NC	FMUL1/FlopocoFPMult/RoundingAdder/add_1_root_add_385_2/U1_32/CI	FA_X1	(7: 26043/0/1)
sa0	NC	FMUL1/FlopocoFPMult/RoundingAdder/add_1_root_add_385_2/U1_31/CI	FA_X1	(8: 26041/0/1)
sa0	NC	FMUL1/FlopocoFPMult/RoundingAdder/add_1_root_add_385_2/U1_30/CI	FA_X1	(9: 26039/0/1)
sa0	NC	FMUL1/FlopocoFPMult/RoundingAdder/add_1_root_add_385_2/U1_29/CI	FA_X1	(10: 26037/0/1)
sa1	NO	FMUL0/OutputR/U34/A2	OAI21_X1	(1: 25421/1)
sa1	NO	FMUL1/OutputR/U34/A2	OAI21_X1	(2: 26064/1/1)
sa1	NO	FMUL2/OutputR/U34/A2	OAI21_X1	(3: 27466/1/1)
sa1	NO	FMUL3/OutputR/U34/A2	OAI21_X1	(4: 28109/1/1)
sa0	NO	FMUL0/InputX/U23/A3	OR3_X1	(5: 398/0/3)
sa0	NO	FMUL0/InputX/U23/A2	OR3_X1	(6: 398/0/2)
sa0	NO	FMUL0/InputX/U23/A1	OR3_X1	(7: 398/0/1)
sa0	NO	FMUL0/InputX/U23/Z	OR3_X1	(8: 398/0/0)
sa0	NO	FMUL0/InputX/U22/A3	OR4_X1	(9: 399/0/3)
sa0	NO	FMUL0/InputX/U21/A3	OR3_X1	(10: 400/0/3)

Table 6.1.6 - Sample of NC and NO Faults in DPU6

6.2 Functional Root Cause Summary for the Six DPUs

Across all six DPUs, the undetected faults classified as NC and NO stem from recurring functional limitations related to internal design structure.

NC faults were primarily observed in isolated inverter chains, unconnected arithmetic modules, or in regions where signal propagation was blocked due to hardwired logic values. These conditions reflect low controllability, often due to unreachable logic cones or lack of input propagation paths.

NO faults, in contrast, originated in deeper combinational blocks such as full adders, NOR/OR gates, and control logic where error signals, though activated, failed to propagate to observable outputs. Common contributing factors include reconvergent fanouts, insufficient scan access, or logic masking from surrounding gates.

General trends observed across the DPUs include:

- DPU1–DPU3: High incidence of NC and NO faults, particularly in multiplier and full-adder modules, highlighting test challenges in dense arithmetic blocks.
- DPU4–DPU6: Fewer NC faults but still a significant number of NO faults, often concentrated in rounding and normalization paths, or output multiplexers.

These findings emphasize the importance of integrating testability features—such as scan insertion and observe points—during the design phase, especially in datapath-intensive modules.

To better capture and compare the distribution of functionally undetected faults across DPUs, the table below provides a consolidated overview of both NC and NO faults, along with their relative percentages over total faults. This unified view allows clearer insight into which DPUs are more susceptible to controllability or observability limitations, highlighting potential optimization points for future hardware testing strategies.

DPU	NC Faults	% of Total Faults	NO Faults	% of Total Faults
DPU1	1,566	3.41%	4,075	8.88%
DPU2	1,582	3.44%	4,124	8.96%
DPU3	1,559	3.38%	4,073	8.84%
DPU4	74	0.16%	1,576	3.42%
DPU5	110	0.19%	3,931	6.86%
DPU6	42	0.09%	1,179	2.59%

Table 6.2 - Summary of NC and NO Faults per DPU

6.3 Overview of Structurally Undetectable Faults

In addition to the functionally caused undetected faults, the ATPG reports also classified a separate set of structurally undetectable faults. These include:

- UB: Faults residing in logic that is never sensitized due to structural disconnection or masking.
- UT: Faults on signals tied to constant values that cannot be toggled during simulation.
- DS: Faults detected outside of ATPG by simulation mechanisms, excluded from ND counts.
- AN: A small class of faults considered untestable due to tool limitations or unsupported logic modeling.

The following table summarizes the structural undetectable fault distribution for each DPU:

DPU	UB	UT	DS	AN
DPU1	40	48	44,079	–
DPU2	36	52	44,470	–
DPU3	36	52	44,544	–
DPU4	8	336	45,826	–
DPU5	60	96	57,311	8
DPU6	8	336	46,255	–

Table 6.3 - Structural Fault Classes per DPU

The relatively low number of **UB** and **UT** faults across most DPUs indicates that most logic is structurally accessible, though DPU4 and DPU6 show elevated counts of tied logic. The **AN** category appears only in DPU5, likely due to complex or unsupported gate structures. The consistently high **DS** counts suggest that simulation-based detection covered a substantial portion of fault space, thereby complementing the ATPG effort.

6.4 Summary and Implications

This chapter presented a detailed functional and structural analysis of undetected faults in six DPUs following ATPG using Synopsys TetraMAX. The undetected faults were divided into two main categories: functionally induced faults, including NC and NO, and structurally untestable faults, including UB, UT, DS, and AN.

From a functional standpoint, NC and NO faults accounted for the most critical cases of undetected behavior. NC faults reflect nodes where fault activation was impossible due to the lack of controllable stimulus, often caused by constant signal propagation, logic isolation, or design topology. These faults were frequently located in inverter chains or

carry input pins of arithmetic elements, where test pattern application was functionally blocked.

NO faults, conversely, highlight areas of poor observability. Here, fault activation occurred, but the erroneous value failed to propagate to a primary output or scan observable point. This condition is commonly associated with reconvergent logic, output masking, or inadequate output tracing within full adders, multipliers, and control structures. The NO category dominated the undetected fault population in all six DPUs, indicating a broader visibility limitation across the datapath designs.

On the structural side, UB and UT faults were generally fewer but still relevant. They represent logic segments that are inherently disconnected or tied to static values, making fault detection structurally impossible. DS faults formed a large portion of the total fault space, indicating that standard simulation techniques successfully covered many fault scenarios outside deterministic ATPG scope. The AN class appeared only minimally and exclusively in DPU5, likely due to synthesis irregularities or tool-specific modeling limitations.

The following key conclusions can be drawn:

- Functional undetected faults (NC and NO) reveal the most significant testability bottlenecks and are directly actionable through DFT strategies.
- DPU4 and DPU6 displayed notable UT counts, suggesting hardwired or redundant logic that could benefit from early DFT review.
- Simulation-based coverage was substantial, but deterministic ATPG remains essential for high-confidence validation and formal coverage reporting.

In light of these results, several recommendations can be made for future hardware designs:

1. Enhance observability by integrating scan chains, test points, or additional probe logic, particularly in datapath-heavy units.
2. Improve controllability by avoiding isolated logic segments and minimizing the use of constant drivers where test access is required.
3. Adopt proactive DFT planning early in RTL development to reduce structurally untestable regions and tool-limit violations.

Ultimately, this fault classification and root cause analysis highlights the need for test-aware design in AI accelerators and arithmetic-intensive blocks. Addressing these undetected fault classes will contribute to more robust silicon implementations and higher fault coverage in production test environments.

Chapter 7 – Conclusion and Future Work

7.1 Summary of Main Contributions

This thesis presented a structured methodology for evaluating the testability of a custom TCU architecture, focusing specifically on the six constituents DPU1 to DPU6 of its architecture. A complete RTL-to-gate-level workflow was implemented using industry-standard tools, including Synopsys Design Compiler, QuestaSim, and TetraMAX.

The contributions of this work include:

- The synthesis and validation of a combinational TCU design supporting parallel dot product computation.
- The configuration and execution of ATPG processes to extract fault coverage metrics and analyze undetected faults.
- A classification of faults into NC and NO categories, followed by detailed functional reasoning for each DPU.
- A summary of structurally untestable faults, such as UB, UT, DS, and AN, to provide a complete picture of testability limitations.
- Integration of test results into a functional analysis framework to guide future architectural enhancements and DFT strategies.

This study bridges the gap between fault modeling and practical fault diagnosis for AI-specific accelerators and contributes to the development of more reliable neural hardware.

7.2 Theoretical and Practical Implications

From a theoretical standpoint, this work demonstrates how ATPG-based test synthesis and fault classification can expose critical weaknesses in AI hardware components. It supports the idea that a deeper understanding of functionally untestable faults is essential in next-generation accelerators.

On a practical level, this methodology offers a reference model for engineers aiming to implement efficient test strategies in VLSI-based AI cores. The use of representative fault tables and structural statistics enhances visibility into the root causes of untestability, which is crucial for future test-point insertion and logic redesign.

7.3 Identified Limitations

While the results provide strong insights, the study is limited by the following factors:

- The analysis focused exclusively on stuck-at fault models, while other realistic models such as delay faults or Cell-Aware faults were not considered.
- Functional reasoning was constrained to available simulation data and did not include exhaustive formal verification of fault propagation.
- The implemented design is based on a simplified TCU core for testing purposes; results may differ in a full-scale production-level architecture.

These limitations suggest potential areas for further exploration and improvement.

7.4 Future Work Recommendations

Several directions are proposed for extending this research:

- Incorporate advanced fault models (e.g., transition and path delay faults) to capture timing-related issues in the TCU datapath.
- Apply formal methods or symbolic simulation to validate functional unobservability beyond deterministic ATPG tools.
- Implement and assess DFT techniques such as test-point insertion or scan-based enhancement in the RTL phase.
- Extend the current fault reasoning approach to more complex or sequential AI accelerator architectures, including pipelined or memory-intensive modules.
- Explore fault injection and resilience techniques to evaluate runtime behavior under faulty conditions, particularly in safety-critical applications.

Overall, this work lays a solid foundation for ongoing innovation in testability analysis and fault resilience in AI-oriented hardware systems.

Bibliography

[1] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a Tensor Processing Unit," *Proc. 44th Annual Int. Symp. on Computer Architecture (ISCA)*, pp. 1–12, 2017.

[2] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," *ACM SIGARCH Computer Architecture News*, vol. 44, pp. 367–379, 2016.

[3] T. Chen *et al.*, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *ACM SIGPLAN Notices*, vol. 49, no. 4, pp. 269–284, 2014.

[4] C. Zhang *et al.*, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," *Proc. 2015 ACM/SIGDA Int. Symp. on FPGAs*, pp. 161–170, 2015.

[5] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," *OSDI*, vol. 16, pp. 265–283, 2016.

[6] A. Paszke *et al.*, "Automatic differentiation in PyTorch," *NIPS-W*, 2017.

[7] NVIDIA Corporation, "NVIDIA Volta GPU Architecture Whitepaper," [Online]. Available: https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf

[8] NVIDIA Corporation, "NVIDIA Turing GPU Architecture Whitepaper," [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf

[9] NVIDIA Corporation, "CUDA C Programming Guide," [Online]. Available: https://docs.nvidia.com/cuda/

[10] S. Markidis *et al.*, "NVIDIA Tensor Core Programmability, Performance & Precision," *IEEE Int. Parallel and Distributed Processing Symp. Workshops (IPDPSW)*, pp. 522–531, 2018.

[11] Z. Jia *et al.*, "Dissecting the NVIDIA Volta GPU Architecture via Microbenchmarking," *arXiv preprint arXiv:1804.06826*, 2018.

[12] Z. Jia *et al.*, "Dissecting the NVIDIA Turing T4 GPU via Microbenchmarking," *arXiv preprint arXiv:1903.07486*, 2019.

[13] J. Devlin *et al.*, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[14] A. Vaswani *et al.*, "Attention is All You Need," *arXiv preprint arXiv:1706.03762*, 2017.

[15] S. Chetlur *et al.*, "cuDNN: Efficient Primitives for Deep Learning," *arXiv preprint arXiv:1410.0759*, 2014.

[16] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[17] S. Ren *et al.*, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *NeurIPS*, pp. 91–99, 2015.

[18] NVIDIA Corporation, "CUTLASS: CUDA Templates for Linear Algebra Subroutines," [Online]. Available: https://github.com/NVIDIA/cutlass

[19] M. Wolfe, "More Iteration Space Tiling," *Proc. ACM/IEEE Supercomputing Conf.*, pp. 655–664, 1989.

[20] NVIDIA Corporation, "cuBLAS Developer Guide," [Online]. Available: https://docs.nvidia.com/cuda/cublas/

[21] NVIDIA Corporation, "cuDNN Developer Guide," [Online]. Available: https://docs.nvidia.com/deeplearning/sdk/cudnn-developer-guide/

[22] J. Appleyard and S. Yokim, "Programming Tensor Cores in CUDA 9," [Online]. Available: https://devblogs.nvidia.com/programming-tensor-cores-cuda-9/

[23] X. Mei and X. Chu, "Dissecting GPU Memory Hierarchy Through Microbenchmarking," *IEEE Trans. Parallel Distrib. Syst.*, 2017.

[24] H. Wong *et al.*, "Demystifying GPU Microarchitecture Through Microbenchmarking," *IEEE Int. Symp. on Performance Analysis of Systems & Software (ISPASS)*, pp. 235–246, 2010.

[25] J. Choquette, O. Giroux, and D. Foley, "Volta: Performance and Programmability," *IEEE Micro*, vol. 38, no. 2, pp. 42–52, 2018.

[26] M. Gebhart, S. Keckler, and W. Dally, "A Compile-Time Managed Multi-Level Register File Hierarchy," *Proc. IEEE/ACM MICRO*, pp. 465–476, 2011.

[27] D. Moloney, "Embedded Deep Neural Networks: The Cost of Everything and the Value of Nothing," *Hot Chips 28*, 2016.

[28] A. Bakhoda *et al.*, "Analyzing CUDA Workloads Using a Detailed GPU Simulator," *Proc. ISPASS*, pp. 163–174, 2009.

[29] J. Lew *et al.*, "Analyzing Machine Learning Workloads Using a Detailed GPU Simulator," *arXiv preprint arXiv:1811.08933*, 2018.

[30] NVIDIA Corporation, "Parallel Thread Execution ISA v6.0," [Online]. Available: https://docs.nvidia.com/cuda/