Functional Verification of CMS PIXEL 28nm Design for the Large Hadron Collider Using Unit-Level Testing and Assertions

By

LILIAN MOLINA
THESIS

Submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

in

Electrical and Computer Engineering

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Chair Rajeevan Amirtharajah

Hussain Al-Asaad

Giuseppe Di Guglielmo

Committee in Charge

2024

Table of Contents

Abstract

The evolution of data processing at the Large Hadron Collider (LHC) requires advancements in the design and implementation of specialized hardware to manage large quantities of data generated by high-energy collisions of subatomic particles in experiments. The Application-Specific Integrated Circuit (ASIC) Design Group at Fermi Lab has developed an integrated chip that merges analog data acquisition systems with digital logic. This chip is designed to perform on-site classification of collision events, effectively distinguishing between low-energy, irrelevant particles and those important to high-energy physics investigations. This design integrates two versions of analog chips with a superpixel architecture, facilitating the direct application of neural network algorithms for data classification, thus significantly reducing the data bandwidth requirements by preprocessing data at the source. Using functional testing, design errors pertaining to undefined logic signals were addressed and corrected in the ASIC. To correct undefined logic signals in the ASIC chip, extensive simulations revealed issues with the `testselect` and `encoderOut` signals in the `superPixel` submodule, which initially included only row values for certain pixel coordinates. By converting these into a 2D signal array to encompass both rows and columns, the design was modified. Additional design updates included removing the `scanCLK` signal to simplify the clocking architecture and introducing a `Resetnot` signal across several modules to ensure the chip starts in a known, stable state. These corrections and enhancements have significantly improved the functionality and reliability of the ASIC.

Acknowledgments

I would like to extend my deepest gratitude to my advisor, *Dr. Rajeevan Amirtharajah*, for his invaluable guidance and unwavering support throughout my studies. His kindness and dedication have been pivotal in shaping my academic journey. I am also profoundly thankful to my undergraduate advisor, *Dr. Shiva Abbaszadeh,* whose mentorship and insights have played a crucial role in advancing my educational and professional training.

I extend my gratitude to the ECE department at UC Davis. In particular, I am thankful to ECE graduate advisors, *Ms. Michelle Theresa Walker* and *Dr. Juan Sebastian Gomez-Diaz*, for arranging logistics as and when required and guiding me through the university formalities.

Finally, I am immensely thankful to my parents for their extraordinary support and motivation throughout my career.

Contributors and Funding

Contributors

The RTL file for this research project was provided by Farah Fahim (farah@fnal.gov) and Manu Blanco Valentin (manuelbv@fnal.gov). The Deep Neural Network and Cocotb Design Environment used for this graduate research thesis were provided by Giuseppe Di Guglielmo Ph.D. (gdg@fnal.gov). Furthermore, this project includes collaborations with Corrinne Mills, Gauri Pradhan, Jennet Dickinson, Jieun Yoo, Nhan Tran, Suresh Kumar, and Suresh Senthilkumar.

The rest of the work was carried out independently by the student under the guidance of Giuseppe Di Guglielmo, Ph.D. and Benjamin Parpillon, Ph.D.

Funding Sources

This thesis project was fully supported by the Department of Energy IC Design Traineeship Program With Applications In High-Energy Physics (HEPIC) Fellowship. The research topic is part of an ongoing Compact Muon Solenoid (CMS) project in collaboration between Fermilab and CERN.

CHAPTER 1

Introduction

The Large Hadron Collider (LHC) generates an extensive array of particle collision data, a substantial portion of which includes low-energy particles that provide little value to high-energy physics research. Traditional data processing methods involve transferring all collision data off-chip for classification, which consumes considerable bandwidth and processing power [1]. To address this inefficiency, the ASIC Design Group at Fermilab has developed a chip that incorporates both analog and digital components to preprocess and classify data directly within the readout hardware. This approach aims to handle data processing of very large data sets by filtering out irrelevant low-energy particle data at the source.

The chip features two distinct versions of analog circuits, each optimized for specific performance metrics such as power efficiency, noise levels, and area utilization. These circuits are part of a larger superpixel configuration—comprising a 16x16 pixel array—connected to an embedded neural network classifier [2]. This setup allows for immediate data classification at the pixel level, enhancing the selection process for data transfer and analysis.

To ensure the reliability of the ASIC design, functional verification strategies are employed throughout the ASIC tapeout process. Additionally, functional verification techniques are used to detect, correct, and validate the chip's functional design. The verification process is supported by testing protocols that involve both random and deterministic simulations to cover a broad spectrum of behaviors or scenarios. We also explore formal verification methods like theorem proving and model checking, which provide mathematical assurances of design correctness. These formal methods are not implemented for this ASIC design but they are presented in order to understand other methods available apart from functional simulations.

This document elaborates on the design approaches taken by the ASIC Design Group, detailing the architecture and functionality of the integrated chip. It also discusses various verification methodologies implemented to validate the design, ensuring that the chip operates reliably under all expected conditions. Through this integration of analog and digital technologies, the project not only enhances the efficiency of data processing at the LHC but also sets a precedent for future developments in particle physics instrumentation. The following sections will delve into the ASIC design flow, formal verification techniques, functional verification processes, and the foundational pillars of verification used to achieve a robust design. Then we look at our design approach implemented through the Cocotb simulator and review our verification results for different modules within the design.

1.1. Background

The Large Hadron Collider (LHC) is a 27-kilometer ring-shaped tunnel located on the border between Switzerland and France. It operates by accelerating subatomic particles to near-light speeds in opposite directions, causing two proton beams to precisely collide within the LHC. Following these collisions, the Compact Muon Solenoid (CMS) detector captures data from the events [3]. Positioned in Cessy, France, the CMS's strategic location enables a probing of these particle collision events, advancing the study of particle physics.

The CMS detector acts as an imaging system, rendering 3D images of the particle collisions at a rate of about 40 million times per second, by analyzing the energy and momentum data from these events [3]. This data is crucial for reconstructing images of the collisions. The primary components of the CMS detector include a solenoid magnet, silicon trackers, the Electromagnetic Calorimeter (ECAL), the Hadron Calorimeter (HCAL), and sub-detectors that capture the final particles. The solenoid magnet, a cylindrical coil made from special fibers that can conduct a current of up to 18,500 amps without electrical resistance, generates a magnetic field of about 4 Tesla—100,000 times stronger than Earth's magnetic field [3]. Its main function is to bend the trajectories of the collided particles, helping to determine their charge and momentum. High momentum particles experience a greater degree of bending compared to those with low momentum. Precise image

reconstruction is achieved by tracing the paths of these particles with millions of electronic sensors that serve as trackers. Additionally, the CMS includes calorimeters in inner and outer layers to measure photon energy. The ECAL measures the energy of particles by completely stopping photons that scatter outwardly at the time of collision, while the Hadron Calorimeter in the outer layers stops other subatomic particles. Furthermore, sub-detectors identify the final particles that the calorimeters do not stop.

ASICs in the CMS pixel tracker, which include both analog and digital functionalities, are designed to manage the substantial data rates and harsh radiation conditions typical at the LHC. Analog ASICs, used mainly in the readout chips of the pixel detectors, amplify and buffer the electrical signals produced by particle interactions with the silicon sensors. Conversely, digital ASICs handle more complex data processing. Specifically, the Token Bit Manager (TBM), a digital ASIC, synchronizes data collection with the LHC's collision frequency of 40 MHz [3].

The design and implementation of ASICs in such an extreme environment faces significant challenges. These circuits must be highly radiation-tolerant to resist the intense particle flux and radiation without degradation over time. Additionally, the high data throughput demands that ASICs process data at extremely high speeds, calling for innovative chip design to ensure reliable and precise data handling [1]. Lastly, considering the compact nature of the CMS tracker and the constraints on power, ASICs must operate with high power efficiency. Excessive power consumption can lead to increased heat, which can diminish the performance and lifespan of the electronics. To combat this, ASICs are designed to be power-efficient.

Overall, the integration of ASICs into the CMS pixel tracker represents a significant advancement in particle physics instrumentation. These chips play a significant role in managing the complex demands of particle detection and data processing, allowing physicists to explore the mysteries of the universe with precision.

1.2. ASIC Design Flow

Application-Specific Integrated Circuits (ASICs) play a crucial role in high-energy physics, particularly in experiments and facilities where precision and efficiency are crucial. These specialized circuits are integral to the operation of detectors and instrumentation used in large-scale physics experiments, such as those conducted at the Large Hadron Collider (LHC) and other similar facilities around the world.

In high-energy physics, ASICs are designed to handle the massive volumes of data generated by particle collisions. They are used in particle detectors to process signals directly from detection elements, such as silicon trackers or calorimeters converting these signals into digital data that can be analyzed by physicists. The speed and accuracy with which ASICs operate allow for real-time data processing, which is essential for filtering and selecting relevant particle collision events among the millions of interactions that can occur every second.

The ASIC design process begins at a conceptual level and extends into physical implementation. For large complex designs, designers must first draft a Register-Transfer Level (RTL) description using hardware description languages such as Verilog or SystemVerilog. The RTL provides a detailed map of the ASIC's digital logic, outlining a behavioral model of how the chip functions. Advances in VLSI design have tremendously scaled ASIC complexity, making IC designs consisting of billions of transistors compacted onto a small silicon die. Due to this large complexity, the likelihood of errors and bugs has also increased. Therefore, it is crucial for bugs to be caught during the design process before chips are fabricated and distributed. The task of verifying the RTL design falls on verification engineers who use comprehensive strategies that include functional or formal verification methods to find errors within the RTL design. Usually, verification engineers use logic simulators to simulate the RTL code and analyze whether the logic outputs match the expected outputs and help debug any discrepancies. Tools such as Cadence Xcelium or ModelSim are employed for simulations to verify behavioral functionality.

Once the RTL is successfully verified, the next step is logic synthesis. A logic synthesis tools works similarly to a compiler that translates source code into assembly language. In this case, the RTL

code is mapped into a library of logic gates called standard cells. The mapping is done with the objective to minimize the area while ensuring timing constraints are being met [4]. Tools like Synopsys Design Compiler translate the RTL code into a gate-level netlist. This netlist includes the individual gates and their connections, which can often comprise thousands of transistors. Designers then revisit this stage to optimize area, speed, and power efficiency according to the initial specifications.

Physical design begins with floorplanning, where the ASIC's major functional blocks are positioned to optimize efficiency and connections. In floorplanning, design engineers estimate the area and other critical components to make sure the proposed architecture fits into the area budgeted. As soon as the design specification is given, a general floorplan is proposed and the design goes through various feedback and modification cycles in order to come up with the best design.

Following floorplanning is placement and routing, where tools such as Cadence Innovus or Synopsys IC Compiler are used to lay out countless standard cells and their connections. Engineers then perform static timing analysis to ensure the design meets all the timing constraints to prevent future failures or hazardous behaviors. Then, the design goes through post-layout verification, which includes Design Rule Checks (DRC) and Layout versus Schematic (LVS) checking to verify that everything is functioning according to the original specifications.

In the final phase of the design process, the design files are sent to a semiconductor foundry where the files are used to create photolithographic masks used to create the patterns of transistors and wires that are then etched into silicon wafers and made into individual chips. Verification also follows the design at this stage, ensuring that the physical chips meet all functional and performance criteria.

Despite multiple iterations of verification at each design stage, the complexity and scale of modern ASICs mean that not every potential fault can be tested or detected. Furthermore, the rapid pace of technological advancements places immense pressure on verification to catch and fix bugs before they are distributed.

1.3. Overview of Thesis

The Large Hadron Collider (LHC) at CERN generates large amounts of data for each particle collision. The ASIC design group at Fermilab is addressing the challenge of handling large amounts of data through the development of an Application-Specific Integrated Circuit (ASIC). The ASIC is designed to filter out low-energy particle data. By integrating both analog and digital components, the chip preprocesses and classifies the data. This thesis covers the verification processes involved in correcting specific faults of this ASIC.

Chapter 2 covers the essential techniques and principles of formal verification in hardware design. We begin by introducing formal verification and highlighting its significance in ensuring that the design meets the specifications across all scenarios. Then the chapter examines the methodologies of theorem proving and model checking. We also discuss functional verification, which includes both random and deterministic strategies to assess system behavior. Throughout this chapter, we emphasize the aspects of verification such as stimulus, coverage, and checking.

Chapter 3 delves deeper into designing a robust verification approach, building on functional verification concepts. The verification test environment is crucial for evaluating the design under test (DUT), which includes a test bench designed under Cocotb, a tool that facilitates verification using Python. This chapter then covers the setup of test benches and test suites that incorporate stimulus, coverage, and checking in order to validate the DUT's correctness.

Chapter 4 examines the ASIC verification process across various RTL modules, adopting both random-based and deterministic testing. This chapter describes the hierarchical verification process of the ASIC, across different components, including a deep neural network (DNN), shift register, scan-chain, ripple-carry adder (RCA), and the top module. The top module integrates all lower level modules and ensures a complete verification process.

Finally, Chapter 5 reviews the modifications made to the ASIC to correct the undefined logic values shown in Chapter 4. Challenges within the `superPixel` submodule were addressed and design updates were reviewed that included correctly instantiating signals `testselect` and `encoderOut`,

removing `scanCLK`, and introducing a `Resetnot` signal. This work lays the foundation for further research and development in ASIC design for applications in high-energy physics.

CHAPTER 2

Logic Verification Literature Review

Formal verification uses mathematical methods to prove the correctness of circuits and systems. We will explore two main techniques which are theorem proving and model checking. Theorem proving involves using logic formulas to verify that a system's behavior aligns with its specifications, while model checking examines the behavioral models of a system against specified properties using temporal logic. Additionally, we discuss functional verification, focusing on random and deterministic methodologies to validate system behavior under various conditions. Finally, we highlight the pillars of verification, such as stimulus, coverage, and checking, which provide a comprehensive framework for evaluating and enhancing the reliability of electronic circuits and systems. This structured approach ensures that every aspect of a design meets the specified criteria.

2.1. Formal Verification

Formal verification is a critical methodology in the field of hardware design, utilizing a mathematical approach to prove the correctness of electronic circuits and systems. Unlike functional simulations which only demonstrate the presence of defects but cannot conclusively prove their absence, this methodology uses mathematical techniques to ensure that a design behaves exactly as intended across all possible scenarios [5]. The primary goal of formal verification is to use a logical framework to prove that a given design adheres strictly to all the specifications, ensuring that the system remains fault-free and behaves as expected under all specified conditions.

2.1.1. Theorem Proving. Theorem Proving is a method that is intended to provide a formal way of describing the specification and the implementation of a digital system. For example, to prove the correctness of a circuit, this method involves using logic formulas to represent the specifications, intended behavior of a system, and then finding out how they are related [5]. A formal logic language can be used in describing the circuit inputs, outputs, and internal behaviors as well

as the intended behavior. The purpose of theorem proving is to show that the circuit model always satisfies the intended behavior, in relation to the specification.

We will explore the formal verification of hardware components, specifically the verification of an AND gate constructed from NAND and NOT gates [5]. This approach is grounded in the principles of higher-order logic to ensure that the gate's implementation adheres to its intended specification.

The process begins with a clear problem statement: proving that an AND gate, built by integrating NAND and NOT gates, functions as a standard AND gate should. The verification is conducted under the assumption that the individual NAND and NOT gates operate as specified [5]. The formal method employed here is theorem proving in higher-order logic, which offers a structured framework to demonstrate that the combined behavior of these gates meets the AND gate specification.

The verification begins by first defining the behavior of the NAND and NOT gates in higher-order logic. Then, the specification for the AND gate is set, assuming that the output should be true if both inputs are true, and false otherwise. The implementation using NAND and NOT gates involves feeding inputs through a NAND gate followed by a NOT gate, which inverts the output of the NAND gate.

To formally prove the correctness of this implementation, we begin with the assumption of the correct behavior of the NAND and NOT gates. Through a series of logical transformations and deductions, it's demonstrated that the output from the NAND followed by the NOT gate setup indeed matches the behavior defined in the AND specification [5]. The logical steps involve applying the definitions of the gates, reducing expressions, and finally proving that the output condition holds true for all input scenarios.

The AND gate implementation, ANDgate.IMP, initially assumes that both the input conditions i_1 and i_2 and the output o conform to the NAND gate operations negated twice [5]. This form uses De Morgan's laws. First, we assume a generic form of the AND gate implementation. This involves

both inputs i_1 and i_2 being processed through a NAND gate to produce an intermediary output z.

The direct application of NAND gates leads to the intermediary state z, which is essentially the negation of the conjunction of i_1 and i_2. The NAND operation that together with a NOT gate translates the output from the negation of z to a more direct expression. Substituting z with its expression in terms of i_1 and i_2, and simplifying using logical identities, the output o is refined to reflect the conventional AND operation between i_1 and i_2.

Finally, the specification of the AND gate, ANDgate.SPEC, is confirmed to be met by the implementation, linking the input conditions directly to the expected output, thus verifying that the implemented logic for the AND gate using NAND and NOT gates behaves as specified under all possible input conditions.

This process shows the inclusion of formal verification in hardware design, ensuring that implementations are error-free and function as intended under their specified operations.

Theorem proving offers a significant advantage in formal hardware verification due to its ability to remove any unclear details in the specification through temporal logic, making it less reliant on human error. This method is effective as it allows for the verification of complex digital circuits across multiple levels of abstraction. For example, it can demonstrate that circuit behaviors at various levels from RTL to physical design match their specifications, linking inputs and outputs in a manner that aligns with predefined logical relationships. This structured approach helps in refining and validating a design against its higher-level specifications and the incomplete understanding held by the circuit's designer.

However, theorem proving is not without its challenges. It requires considerable effort from the user in defining detailed specifications for each component and guiding the theorem prover throughout the stages of the verification process. The process also demands a substantial investment in time and computational resources to navigate through complex algebraic levels and relationships within

the hardware being verified.

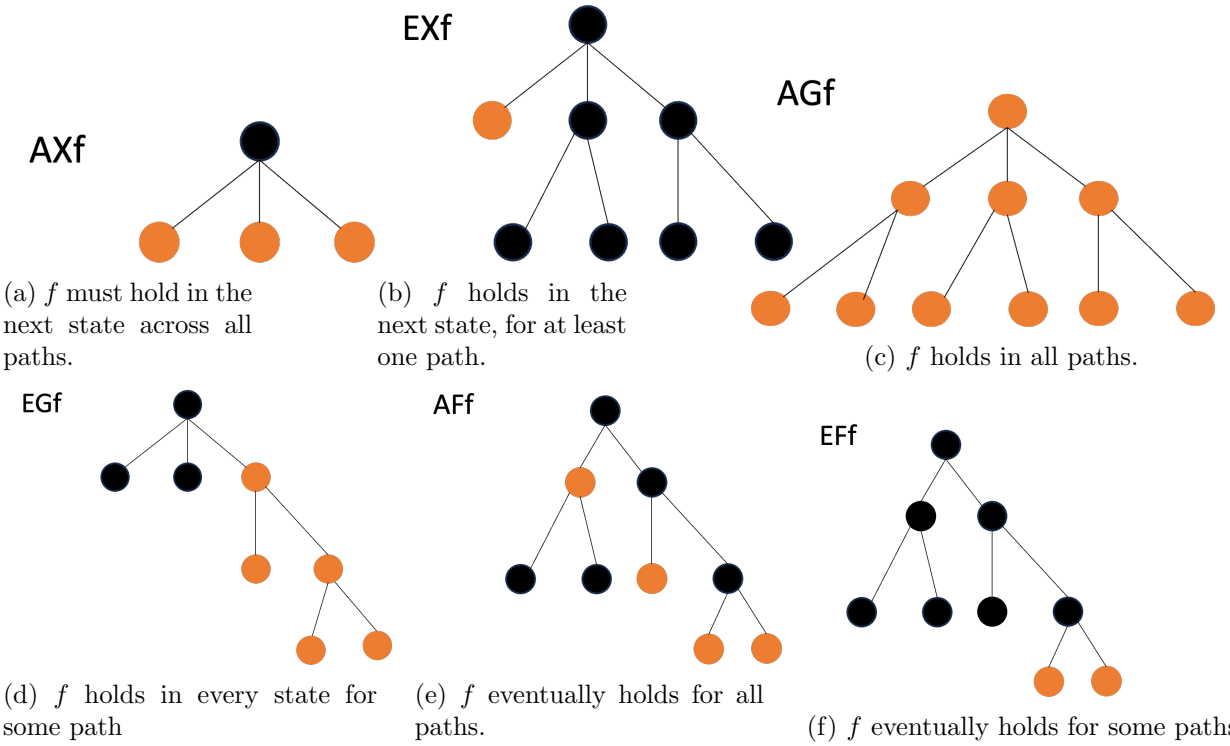
2.1.2. Model Checking. Model checking is a way of verifying the behavioral models of a system with respect to temporal logic. Essentially, model checking provides a way of specifying all the possible states that a system can be in and then checking if the system violates any specifications at a given state. A Computational Logic Tree (CTL) can then be employed to find the sequence of events that lead to a fault within the system.

A CTL formula is defined with respect to some set of atomic formulas. These atomic formulas should be thought of as basic properties of the individual states. An atomic formula, which is the most basic element, is considered a CTL formula and represents simple, state-dependent expressions [5]. CTL extends this foundational concept by enabling the construction of more complex formulas. If f is a CTL formula, various operations can be applied to expand its scope: the negation $\neg f$ inverts its truth value, while path quantifiers paired with temporal operators further refine its application. Specifically, $\mathbf{AX}f$ stipulates that f must hold in the next state across all paths, and $\mathbf{EX}f$ requires f to hold in the next state on at least one path. The operators $\mathbf{AG}f$ and $\mathbf{EG}f$ ensure that f holds across all states for all paths and at least one path. Meanwhile, $\mathbf{AF}f$ and $\mathbf{EF}f$ assert that f will eventually hold for all paths and at least one path [5]. This comprehensive framework facilitates the analysis and verification of complex system behaviors in various contexts.

The figures shown in Figure 2.1 below offer a visual representation of various CTL operators and how they are applied across different paths in a model. Each tree in the figure illustrates the specific application of one CTL operator, with black nodes representing the current state and orange nodes representing future states [5]. The labels such as AXf, EXf, AGf, EGf, AFf, and EFf correlate to different CTL formulas, each demonstrating how these operators dictate state relationships within computation trees.

The figure is a clear tool that helps explain the temporal and path-dependent nature of CTL formulas, showing how different conditions can be evaluated with state transitions within computational

models.



(a) f must hold in the next state across all paths.

(b) f holds in the next state, for at least one path.

(c) f holds in all paths.

(d) f holds in every state for some path

(e) f eventually holds for all paths.

(f) f eventually holds for some paths.

FIGURE 2.1. Computational Tree Logic paths formulas

CTL is used extensively in system verification for its precise and rigorous ability to define and check properties of system behaviors over time. One of its significant strengths is that the decision procedure is fully automated, allowing verification engineers to verify complex temporal properties without manually tracking the CTL structures [5]. This automation facilitates the detection of errors in both hardware systems, enhancing reliability and correctness.

However, CTL also has weaknesses. Specifically, it enumerates desired properties, which can make it difficult to conclude if the verification process has thoroughly covered all relevant system behaviors or just a subset. This can lead to potential oversights where some critical behaviors may not be checked [5]. Moreover, CTL faces the "state explosion problem" where the state space for systems with numerous components becomes extremely large, making verification computationally intensive. This complexity can decrease the effectiveness of CTL in environments with immense

state spaces.

2.1.3. Symbolic Trajectory. Symbolic trajectory uses temporal logic to enumerate a set of properties of a given circuit. The properties are then expressed as assertions, which take the form of "Antecedent" and "Consequent" [6]. This relationship is used to specify a set of properties of a system, where the antecedent describes a state or premise of a system, and the consequent describes the outcome that is produced if the antecedent is true. In symbolic trajectory simulation, the antecedents are simulated, and then consequences are asserted.

2.2. Functional Verification

Functional verification aims to check if a system satisfies its intended behavior through simulation. Functional verification can then be subdivided into random or deterministic methodologies. Random-based verification is usually used in black box scenarios where the designer does not have much prior knowledge of the functionality of the system. In deterministic verification, the designer has some prior knowledge as to the property set of the system.

2.2.1. Random. Typically, random-based simulations are implemented when the space of possible states of a system is substantially large. Random simulations can be further categorized as static and dynamic.

2.2.2. Deterministic. In deterministic testing, the test is generated based on knowledge of the chip. Deterministic techniques include ad hoc and general testing. In ad hoc testing, modules are subdivided and each block within the system is targeted to verify . General testing techniques include simulation of the possible transitions in a finite state machine.

2.3. Pillars of Verification

2.3.1. Stimulus. In the verification cycle, stimulus inputs are used to invoke specific states or behavior of the system. Stimulus vectors are intended to simulate various edge cases and to provide

a thorough verification of the behavior and functionality of the system. These stimulus vectors are used to exercise and detect potential faults and ensure that the system is meeting design constraints.

2.3.2. Coverage. Coverage is one of the most important metrics in the design and verification cycle. Coverage metrics can be user-defined and are based on human judgment if the design is free of all potential faults or bugs. Furthermore, the coverage metric helps assess the completeness of the verification cycle and leads the designer to areas that might have missed verification.

In the domain of electronic circuit verification, fault coverage (FC) serves as a crucial metric to gauge the efficacy of a test suite, T, in detecting potential faults within the circuit. The formula for calculating fault coverage is given by:

$$\text{FC}(T) = \frac{\text{Number of faults detected by } T}{\text{Total number of detectable faults}}$$

This ratio quantifies the proportion of the total detectable faults that the test suite T successfully identifies. The test suite comprises a collection of tests designed specifically to exercise various aspects of the circuit, aiming to trigger and thereby reveal any existing faults through the circuit's responses.

Fault coverage is fundamentally important for several reasons. Firstly, it is a direct measure of the test suite's effectiveness. A higher fault coverage indicates a more effective test suite that can detect a larger number of potential faults, enhancing the verified circuit's reliability . Secondly, in critical applications, such as aerospace or medical devices, achieving high fault coverage is essential to ensure safety and operational reliability. It confirms that the circuit can be trusted to perform under varied conditions without failure.

Moreover, fault coverage is an invaluable tool in the design verification process. It helps in identifying areas where the circuit design might be prone to failures, thus guiding engineers in refining the circuit architecture . This metric also assists in determining the robustness of a circuit against

a wide range of potential faults, informing further enhancements in the design or testing strategies.

In summary, fault coverage not only underscores the quality and thoroughness of the testing processes but also contributes significantly to the overall reliability and safety of electronic circuit designs. Its role is pivotal in certifying that a circuit meets its defined operational specifications and can withstand real-world operational stresses.

2.3.3. Checking. In verification, checking is used in monitoring a simulation output and asserting a pass or fail flag. The designer specifies failing conditions to find bugs, which is when the designer simulates scenario against a predefined criterion or reference model. At the beginning of the verification cycle, this reference model can consist of predefined conditions in the test bench.

At the beginning of the verification cycle, it is essential to have a reference model or a set of predefined conditions incorporated within the test bench. This reference model serves as a benchmark in which the circuit's behavior is compared during simulation tests. The use of a reference model helps in structuring the verification process, providing a clear and objective standard for measuring the circuit's compliance with the expected outcomes. By comparing the simulation results to this model, discrepancies are easily highlighted, enabling designers to make precise adjustments and improvements. This methodical approach ensures thoroughness in the testing process, thereby increasing the likelihood of achieving a robust and error-free circuit design.

CHAPTER 3

Verification Framework

3.1. Verification Approach

Designing a robust verification approach is integral to ensuring the reliability and correctness of electronic designs. We will build on the principles of formal verification discussed in the previous chapter. We start by establishing a robust verification environment that utilizes stimulus, coverage, and checking within a testbench framework to assess and validate the design under test (DUT). This environment is crucial for simulating real-world conditions and verifying that the DUT meets all specified operational requirements. We then delve into the specifics of the testbench setup, detailing its components and functionalities, which are essential for a comprehensive evaluation of the DUT. Following this, we introduce the Cocotb simulator, a powerful tool that utilizes Python for writing testbenches, providing a flexible and intuitive approach to verification. Lastly, we discuss the configuration of Python testbenches and test suites, which play a pivotal role in identifying and addressing potential faults in the design phase, ensuring that the hardware system functions correctly across a range of scenarios and meets all design specifications.

3.1.1. Verification Environment. The verification environment developed for this project uses stimulus, coverage, and checking to create a framework to validate and verify the correctness of the design under test (DUT). This verification environment is referred to as the testbench.

3.1.2. Testbench. The verification environment, crucial for assessing and validating the design under test (DUT), is often implemented through a testbench. A testbench is a specialized environment set up to simulate the electrical and logical conditions under which a piece of electronic hardware is expected to operate. It serves as a framework to provide inputs, monitor outputs, and inject various operational scenarios into the DUT to ensure that it behaves as expected across all specified conditions.

The testbench includes three fundamental components: stimulus, coverage, and checking.
Stimulus involves generating and applying a set of test vectors as inputs to the DUT to emulate both typical and extreme operational conditions. The stimulus is designed to challenge the DUT, exposing potential weaknesses and verifying robustness. It can simulate a range of conditions from standard operating inputs to edge cases and error states that are critical for testing the DUT's response capabilities.

Coverage metrics are used to quantify how thoroughly the tests explore the functional and operational aspects of the DUT. This component ensures that every part of the design is exercised by the tests, highlighting any areas that have not been tested. Effective coverage analysis is vital for guiding further test development.

Checking is a component that automatically evaluates the DUT's outputs in response to the test inputs. It checks the outputs against expected outcomes to assert whether the conditions for passing have been met. This process is crucial for identifying discrepancies or failures in the DUT's performance, allowing for rapid diagnosis and correction of any issues.

Including all of these components within the testbench not only tests and verifies the DUT against its expected specifications but also ensures it can reliably perform in its intended operational environment. By simulating real-world conditions and systematically verifying every aspect of the DUT, the testbench plays a pivotal role in the product development lifecycle, significantly enhancing the reliability and quality of the final hardware.

3.1.3. Test-Suite. Test-suites refers to an ensemble of test vectors that are chosen to validate or drive the system to a desired state or scenario. In other words, a test suite is a collection of test cases that are designed to validate the functionality and performance of a hardware system against its specifications . Each test case within the suite is a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements or works correctly.

The test vectors are chosen such that the designer can determine if the system satisfies a set of constraints or design specifications. This will help the designer determine if the system works correctly. Test-suites are designed to cover possible edge cases and potential fault conditions.

3.2. Cocotb Simulator

Verification processes in circuit design critically depend on simulators to observe the functional behavior of the inputs and outputs across various modules. One of the prominent simulators employed for this purpose is Cocotb. Standing for "Coroutine-based Co-simulation Test Bench," Cocotb is a specialized simulator framework that allows for writing testbenches in Python. This feature significantly simplifies the testing process, as Python is a high-level, versatile programming language familiar to many developers, in contrast to hardware description languages (HDLs) like VHDL, Verilog, and SystemVerilog that are typically used for writing testbenches.

Cocotb is known for its ability to operate as a coroutine simulation library, integrating with existing simulation frameworks without the need for additional adapters or special testbench languages. This approach allows developers to write sequential code to interact with the simulation, making the testbenches easier to implement and maintain.

Furthermore, Cocotb is open-source and supports cross-platform functionality, being compatible with Linux, Windows, and macOS. Its integration with different simulation tools and its support for multiple HDLs make it a versatile choice for testing RTL (Register Transfer Level) files. By using Cocotb, verification engineers can efficiently create and run complex test scenarios that simulate various operational conditions. Engineers can develop Python testbenches and run the simulation simultaneously with the RTL and ensure that all potential issues are identified and corrected in the design phase.

3.3. Python Testbench

In the field of hardware verification, the use of Python through Cocotb has improved the process of creating testbenches. Cocotb utilizes Python coroutines, which are a form of asynchronous

programming that allows the testbench to pause and wait for hardware events or specific conditions.

A typical testbench in Python testbenches uses Cocotb coroutines to interact with the design under test (DUT). The coroutines are defined using Python's `async def` syntax, and they manage the sequence of test actions through non-blocking waits. These waits are facilitated by triggers such as `RisingEdge`, which waits for a change in a signal's state, or `Timer`, which waits for a specific amount of simulation time to pass. This allows the testbench to synchronize with the DUT's operation, such as waiting for a clock edge before changing input signals or checking outputs. The Cocotb framework manages the scheduling and execution of these coroutines, integrating it with the simulation environment.

Testing the DUT involves sending a series of test vectors defined as sets of inputs intended to exercise various aspects of the circuit's functionality. These vectors are applied to the DUT's inputs, and the resulting outputs are captured and compared against expected results predefined by the test designer. This comparison is critical, as it verifies whether the DUT behaves as expected under different conditions. To facilitate this, Cocotb testbenches typically include functions to load test vectors from external files or to generate them during simulation. The test vectors are then driven into the DUT through assignments within the coroutine, often synchronized with the DUT's clock. For instance, a test vector might be a binary string representing a sequence of logic levels to be applied to a digital input. After applying each vector, the testbench uses assertions to check the DUT's outputs against expected values, effectively verifying the correctness of the DUT's design in an automated and repeatable manner.

Overall, Cocotb and its coroutine-based approach provide a powerful and flexible framework for developing testbenches. This methodology not only improves the reliability of the testing process but also enhances the productivity of verification engineers, as they can leverage Python as the main verification language and bypass the obstacles associated with learning more challenging languages such as C/C++.

CHAPTER 4

Verification Results

To test the correctness of the RTL modules we used functional verification which included both random-based testing and deterministic testing. We analyzed and tested each individual unit module before conducting comprehensive tests on the entire circuit via the top module. Figure 4.1 illustrates the layout of these modules.

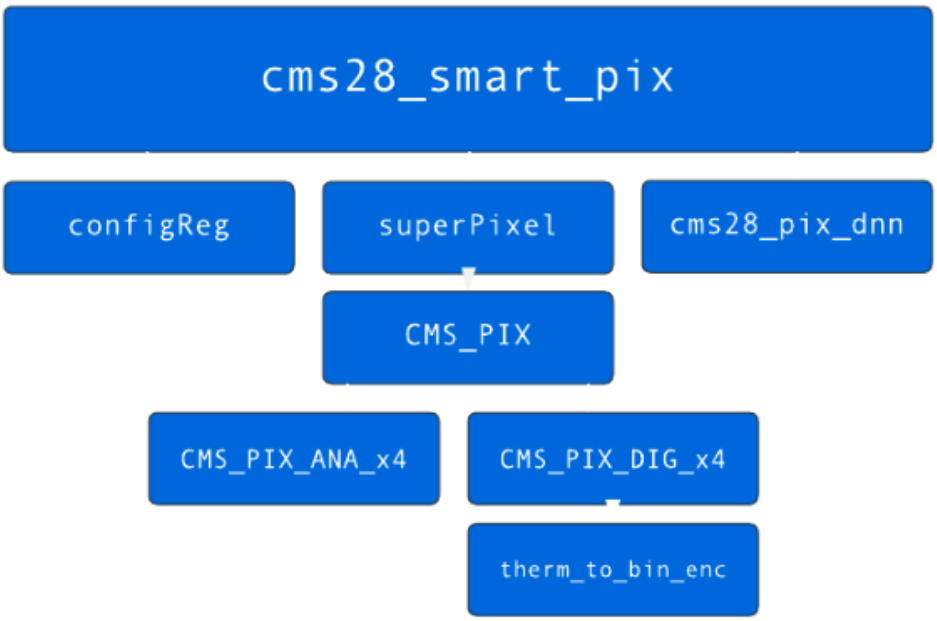


FIGURE 4.1. ASIC Module Hierachy

4.1. Deep Neural Network

Most of the particles observed during photon collision pertain to low-energy event activity. In order to improve bandwidth from the readout chip, low-energy particle clusters must be filtered out at the data source. A neural network was designed by the ASIC design group at Fermilab to identify

high momentum particle clusters.

A deep neural network (DNN) typically consists of three layers: the input layer, a hidden (dense) layer, and the output layer, as shown in Figure 4.2 This layout is typical in a neural network architecture. The first layer is the input layer. The input layer consists of three neurons (X1, X2, X3), representing the initial data points fed into the network. These could represent various features of a dataset.

The second layer is the hidden layer. The hidden layer includes three neurons (h1, h2, h3) connected to each input neuron. Each connection has a weight (A11, A12, A13, etc.), which adjusts as the network learns. The neurons in this layer transform the input values by weighted sums and typically apply a nonlinear activation function to pass an output to the next neural network layer:

$$
\begin{aligned}
h_1 &= A_{11}x_1 + A_{12}x_2 + A_{13}x_3 \\
h_2 &= A_{21}x_1 + A_{22}x_2 + A_{23}x_3 \\
h_3 &= A_{31}x_1 + A_{32}x_2 + A_{33}x_3
\end{aligned}
$$

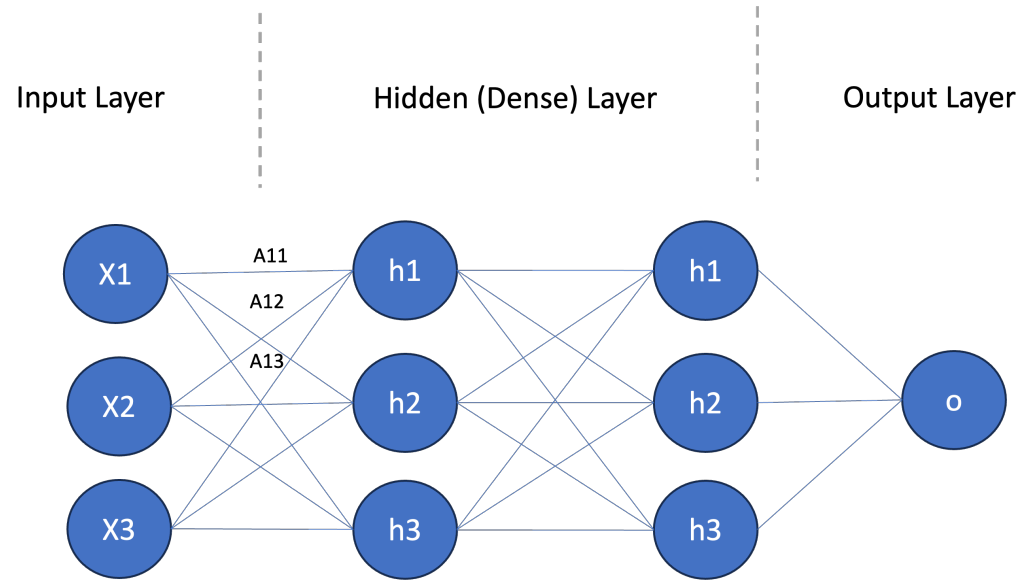


FIGURE 4.2. DNN Layers

The network's output is derived from processing the activation's of the hidden layer neurons through an activation function and then combining them to produce the final output. This structure allows the network to learn complex patterns and relationships within the data. In a neural network architecture, the equations

$$h = Ax \tag{4.1}$$

$$y = F(Ax) \tag{4.2}$$

explain the process of data transformation within the network. Here, A represents the matrix of weights and x denotes the input vector. The product Ax calculates the weighted sums of inputs for each neuron in the following layer.

The function F in the equation is known as the activation function. This function is crucial as it introduces nonlinearity into the model, enabling the network to capture complex patterns and

relationships within the data. Activation functions such as the sigmoid, ReLU, or tanh are common in DNNs and serve to transform the linear combination of inputs into an output signal. This output is either passed to subsequent layers or serves as the final network output.

The model uses two activation functions: ReLU and Argmax. An activation function is a mathematical operation applied to a layer's output to introduce nonlinearity [2]. This is crucial because it enables the network to learn and represent complex patterns and relationships in the data. Without nonlinearity, the network would essentially be a series of linear transformations, which cannot capture intricate relationships. ReLU (Rectified Linear Unit) is a popular activation function that sets negative values to zero while keeping positive values the same. This allows the network to efficiently learn complex patterns in data. Argmax is another activation function that selects the class with the highest score.

Overfitting occurs when a model learns the training data too well but struggles to generalize to new data. To combat this, a batch normalization (BN) layer was included [2]. Batch normalization standardizes the inputs to a layer by normalizing the data using its mean and variance. This reduces the sensitivity of the network to initialization and helps stabilize training.

The DNN model implemented in the CMS Pixel ASIC comprises 1,163 trainable parameters, with 986 located in the first dense layer and 177 in the second. These parameters include weights, which determine the influence of each input on the output, and biases, which independently adjust the output . Table 4.1 gives a description of the DNN input signals and the bit width for the corresponding weights and biases.

Weights and biases are stored as fixed-point numbers, which contain one sign bit to distinguish between positive and negative values and three bits for the fractional part.

The ReLU activation function is represented using 8-bit data, meaning it outputs values ranging from 0 to 255. This allows the ReLU to accurately capture variations in the data without requiring excessive memory [8]. The final model output is represented by a 2-bit unsigned integer, which can

hold up to four values. In this application, the output distinguishes among three classes: negative high (0), low (1), and positive high (2) [9]. These classifications pertain to oppositely charged particle trajectories that curve in opposite directions in a magnetic field. This limited representation minimizes the memory requirements and speeds up the computation.

To implement this deep neural network (DNN) efficiently, the ASIC design group at Fermilab used the hls4ml workflow [8]. This tool translates machine learning models into hardware using high-level synthesis, which produces a register-transfer level (RTL) design ready for ASIC development. This RTL code was then integrated into the system with the necessary registers. The hls4ml workflow allowed the group to develop a hardware solution that is both fast and resource-efficient.

The verification process for the digital hardware implementation employs fixed-point arithmetic to ensure the simulation closely mirrors real-world operations. Fixed-point arithmetic is advantageous for embedded systems due to its simplicity and speed compared to floating-point arithmetic, making it ideal for real-time applications.

Port Name	Bit-width	Description
input 1	16 * 6 = 96	DNN input features. 16 inputs, each input is unsigned integer on 6 bit
w2	928 * 4 = 3712	DNN first dense layer weights. 928 weights, each weight is a fixed-point value (4,1)
b2	58 * 4 = 232	DNN first dense layer biases 58 biases, each bias is a fixed-point value (4,1)
w5	174 * 4 = 696	DNN second dense layer weights. 174 weights, each weight is a fixed-point value (4,1)
b5	3 * 4 = 12	DNN second dense layer biases. 3 biases, each bias is a fixed-point value (4,1)
layer7 out	1 * 2 = 2	DNN output predictions. 1 input, each input is unsigned integer on 2 bits

TABLE 4.1. DNN module specifications.

The testbench utilizes the 'fxpmath' Python library to simulate fixed-point operations. This library allows for the customization of bit lengths for both the integer and fractional parts, enabling precision control over numerical accuracy and range. For our tests, values are converted to a fixed-point format with a specific configuration suitable for our hardware model—using four bits in total and rounding values to the nearest representable number.

During testing, critical parameters such as weights and biases of the neural network are first converted into this fixed-point format before they are fed into the Device Under Test (DUT). This step

is crucial as it ensures the test conditions accurately reflect the hardware's data handling capabilities. By comparing the output from the DUT against expected outcomes under these controlled conditions, we can effectively verify the hardware's ability to perform neural network computations correctly. Figure 4.3 shows the scoreboard assertion test for the DNN. This approach is vital in validating the design's accuracy before it goes into production, minimizing the risk of errors and ensuring the system's functionality and reliability.

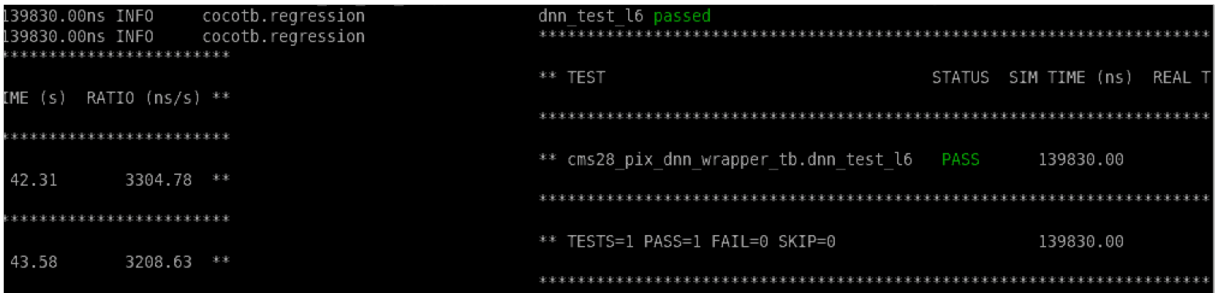


FIGURE 4.3. DNN scoreboard

4.2. Shift Register

All weights are stored in registers, and not static random access memories (SRAMs) [9] . In the shift register module, when a new bit is passed to the input (on a new clock cycle), all of the existing bits are shifted down to make room for it. At the other end of the register, the last bit is passed as an output before being discarded. This shift-register works as SIPO, that is **serial-in, parallel-out** [10]. The data is loaded into the shift register one bit at a time but the entire vector of bits can be loaded out completely. This is useful when the data is received sequentially, but all bits need to be retrieved together, such as the weights and input features of the neural network. Figure 4.4 shows a diagram of a serial-in, parallel-out shift register with its respective inputs and output ports. Table 4.2 provides a detailed explanation on how shift-register size was chosen. The full size of the shift register is composed of the sum of the weight bits from the DNN and the pixel bits. Therefore, the full size shift register as seen from the top module includes 5164 bits for the registers. Table 4.3 describes the bit width of each input signal for the shift register.

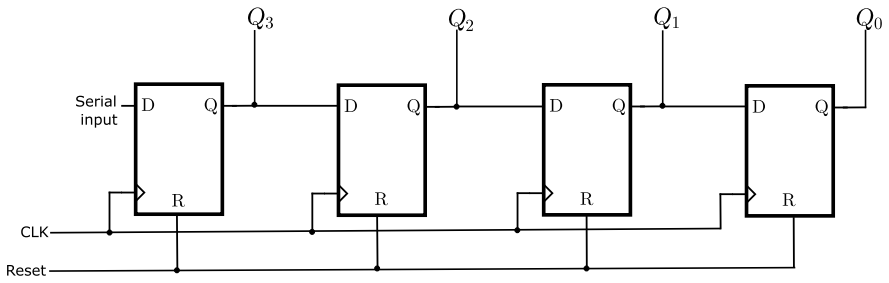


FIGURE 4.4. SIPO Shift Register

Port Name	Bit-width	Description
NWEIGHTS	4652	The total number of weight bits in the neural network. w2+b2+w5+b5 = 928*4+58*4+174*4+3*4 = 4652
PIXEL CONFIG	512	The total number of pixel bits. 16*16*2 = 512

TABLE 4.2. DNN Bit Specification.

Port Name	Bit-width	Description
configClk	1	Clock Signal
configRst	1	Reset Signal
configIn	1	Loads data serially into the shift register
configLoad	1	Enables data to be sent outside the shift register
configOut	1	Loads data serially out of the shift register
parallelOut	4652 + 512 = 5164	Loads parallel data outside the shift register. parallelOut = NWEIGHTS + PIXEL CONFIG **

TABLE 4.3. Shift Register Module Port Specification.

Functional testing of the shift register began with unit-level testing, involving a module with 12 bits. Three primary types of tests were performed on the shift register. The first test involved sending a single bit through the module to verify if it was transmitted through all 12 cells. This crucial test was conducted across all ASIC modules to ensure there were no disconnections between the circuit nodes and to verify the output bit passed along each cell, as depicted in Figure 4.5. `ConfigIn`, a one-bit input signal, carries the bit through the 12 cells of the shift register. After 12 rising edges, the output at `ConfigOut` is expected. The second test observed multiple vectors passing sequentially through the Device Under Test (DUT); these vectors were stored in a predefined file and loaded into the testbench, shown in Figure 4.6. The final test evaluated the shift register's parallel output, as shown in Figure 4.7. The purpose of this test was to verify if the test vectors could be observed through its parallel output functionality.

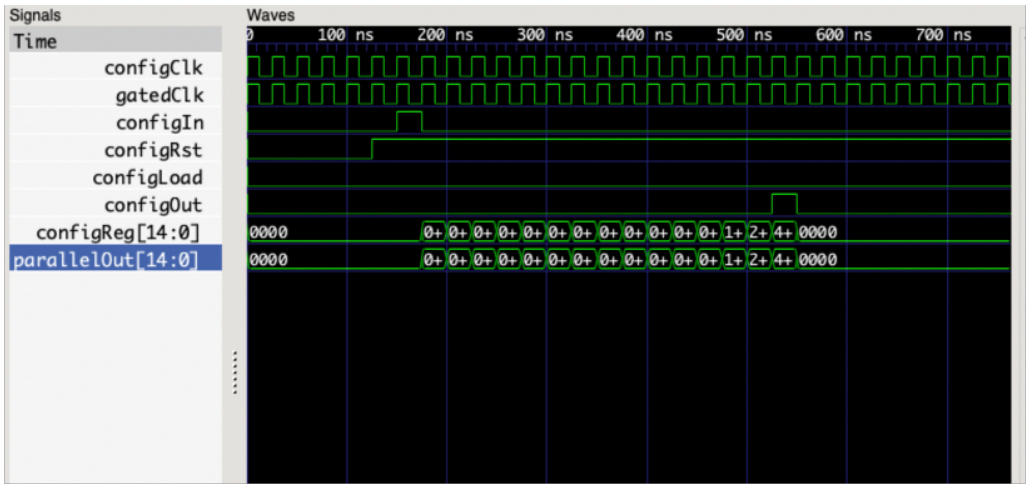


FIGURE 4.5. Test 1: Shift register test passing a single bit through the shift-register.

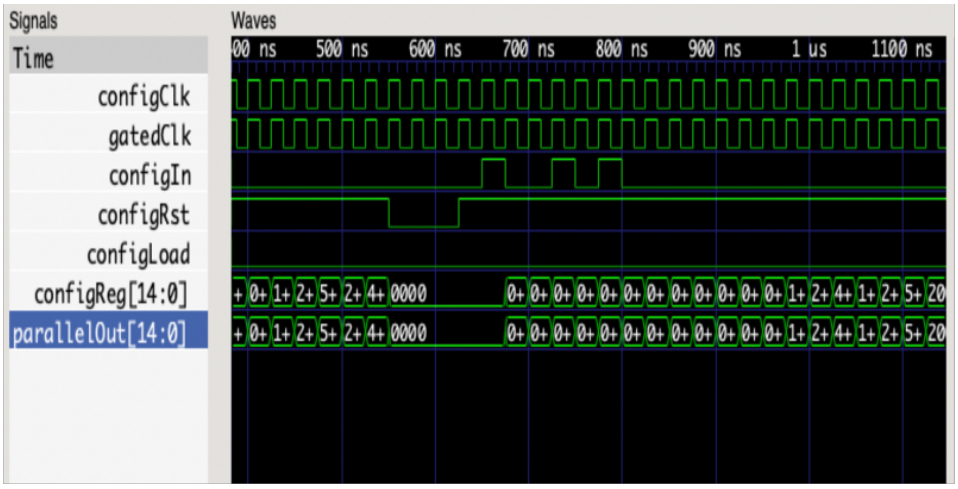


FIGURE 4.6. Test 2: Shift register test passing multiple test vectors.

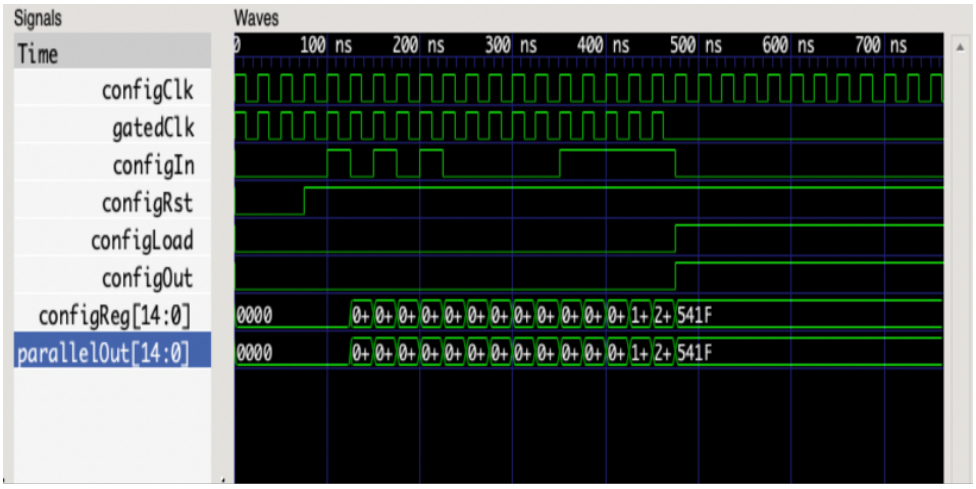


FIGURE 4.7. Test 3: Shift register test for parallel out of the shift register.

These three tests of the shift register reveal the challenges of detecting a disconnection or fault within the register. A disconnection may not disrupt the overall behavior in a way that is significantly detectable. This condition is due to the fact that the output of a shift register is driven by its input, making the intermediate stages less observable. Therefore, in functional verification, faults in shift registers can often go undetected if these components are tested in isolation, such as in these tess. This presents a significant challenge in ensuring the system proceeds to the next stage in the tapeout process fault-free.

4.3. Scan-Chain

Contrast to shift register testing, scan chain testing is highly detailed and aimed explicitly at discovering faults within the ASIC, including signal disconnections [10]. We performed similar tests using the scan chain module to achieve better visibility within the state of the shift register and ensure any disconnection is found.

In the scan-chain module, when the `scanload` signal is low, the scan chain is disabled. Figures 4.8 provides a detailed diagram of how the scan-chain works. When `scanload` is high, the scan chain is enabled and data is loaded using a 12 bit SIPO register. The `compOut` data (coming from the analog-digital converters or ADCs) from a 2x2 pixel is mapped from the Most Significant Bit (MSB) to the Least Significant Bit (LSB), being the Left Top (LT), Right Bottom (RB), Right

Top (RT), and Left Bottom (LB) pixel. When `scanCLK` is at rising edge, the register shifts one position to the right and the rightmost bit is dropped. This is when the `scanIn` input loads serial data into the leftmost bit. `ScanOut` drives data outside the chip for debugging and testing. Table 4.4 provides the bit-width description of the input and output signals for the scan-chain module.

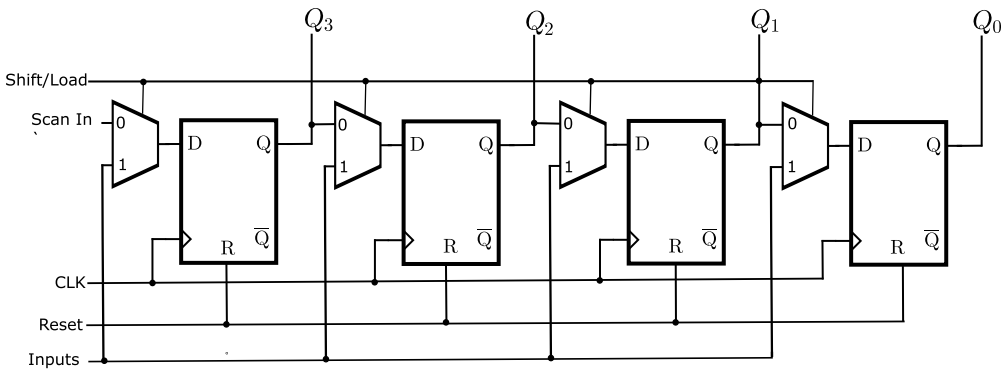


FIGURE 4.8. SIPO Scan Chain

Port Name	Bit-width	Description
BXCLK	1	Clock Signal
scanClk	1	Clock Signal for Scan Chain Module
scanRst	1	Reset Signal
scanIn	1	Loads data in serially to test functionality
scanLoad	1	Load data from the pixels to the scan-chain registers.
scanOut	1	loads data out of scan-chain registers
compoutLB	3	3-bit input signal from left bottom of pixel
compoutLT	3	3-bit input signal from left-top of pixel
compoutRB	3	3-bit input signal from right-bottom of pixel
compoutRT	3	3-bit input signal from right-top of pixel
encoderOutLT	3	3-bit output signal from the thermal to binary encoder for the left-top pixel
encoderOutRB	3	3-bit output signal from the thermal to binary encoder for the right-bottom pixel
encoderOutRT	3	3-bit output signal from the thermal to binary encoder for the right-top pixel

TABLE 4.4. Scan Chain Module Port Specifications.

In the scan chain module, the same tests conducted for the shift register were repeated. Initially, a single bit was passed through the 12-bit scan chain module, and the output for Scan-Out was observed. Subsequently, a second test involved passing pre-defined deterministic test vectors through the DUT to determine if multiple test vectors could be observed at the output. These pre-defined test vectors were stored in separate files and loaded into the python testbench. Figures 4.9 and 4.10 illustrate the results of these simulations. It is evident from the undefined logic states shown in

Scan-Out that there is a disconnection in the RTL signals.

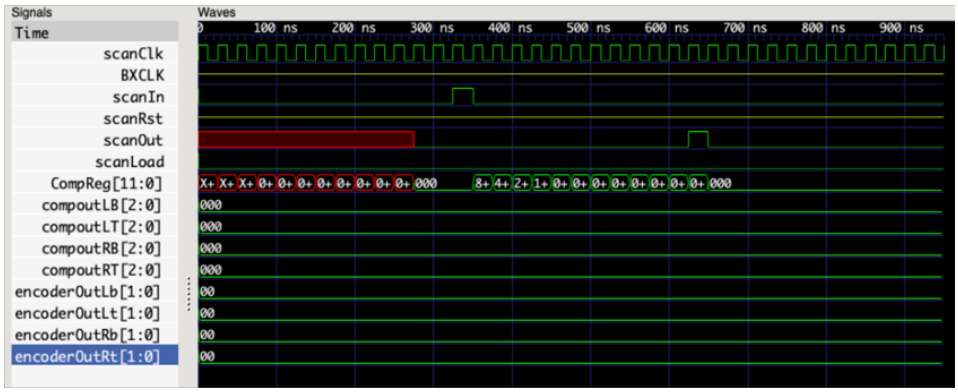


FIGURE 4.9. Test 1: Scan Chain test passing a single bit

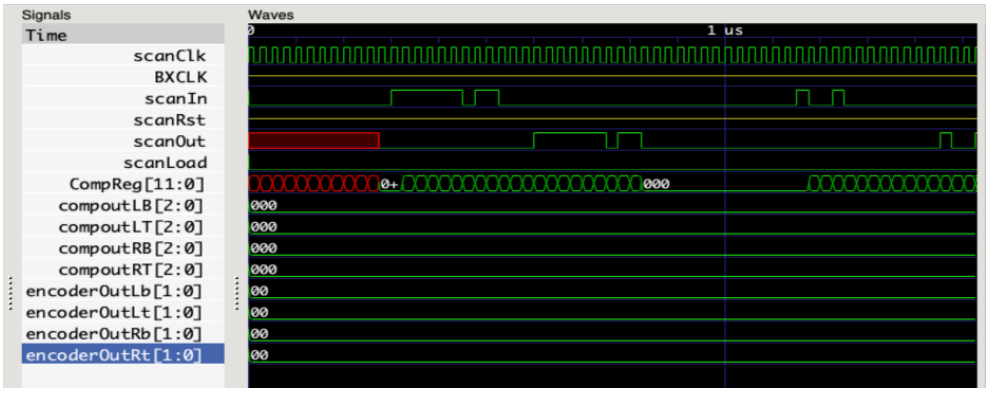


FIGURE 4.10. Test 2: Scan chain test passing multiple test vectors from vector file

These simulation diagrams indicate there are undefined states but do not pinpoint the exact location of the disconnections within the RTL design. To fully understand the where fault was located within the RTL, simulations from all the modules within the ASIC were required. Once all modules within the design were simulated, backtracing and following signals through the entire ASIC design was performed to find the faults within the design.

4.4. Updates for Scan-Chain and Shift Register

Extensive simulations were conducted to locate the disconnection within the ASIC module and ensure that the RTL produces clean and error-free synthesis reports, as well as passing all test

cases in the cocotb simulator. After multiple meetings and discussions with the ASIC design group, it was discovered that the RTL module `superPixel`, which contains an entire matrix of pixels and the digital signals associated with the thermo-to-binary encoder and scan chains, incorrectly instantiated signals for `testselect` and `encoderOut` for the left top, right bottom, right top, and left bottom pixels. Although the disconnection of these signals was not in the scan-chain module, scan-chain testing was sufficient to detect the undefined logic state values and perform corrective measures.

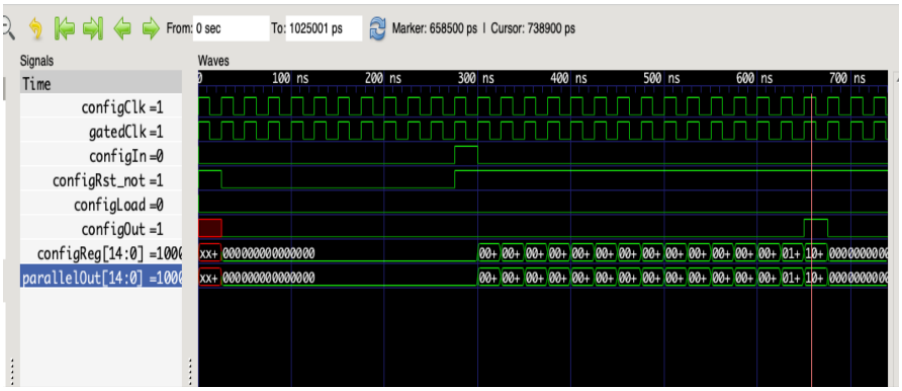


FIGURE 4.11. Update Test 1: Shift Register

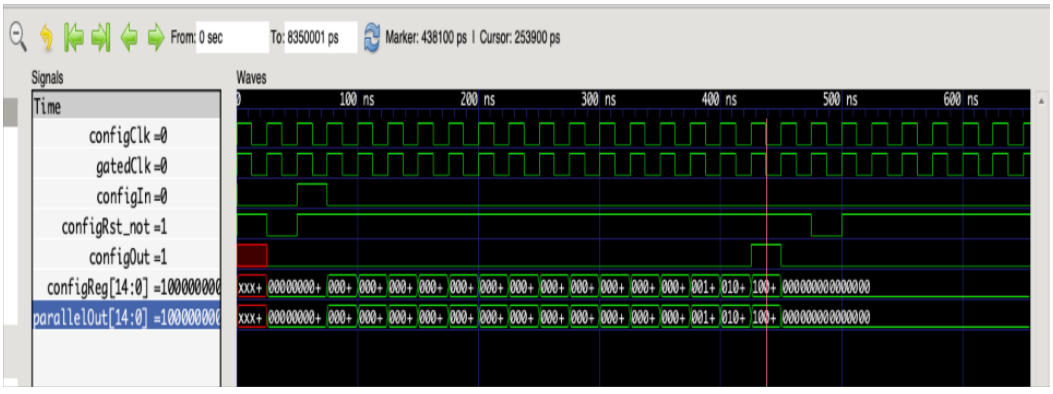


FIGURE 4.12. Update Test 2: Shift Register

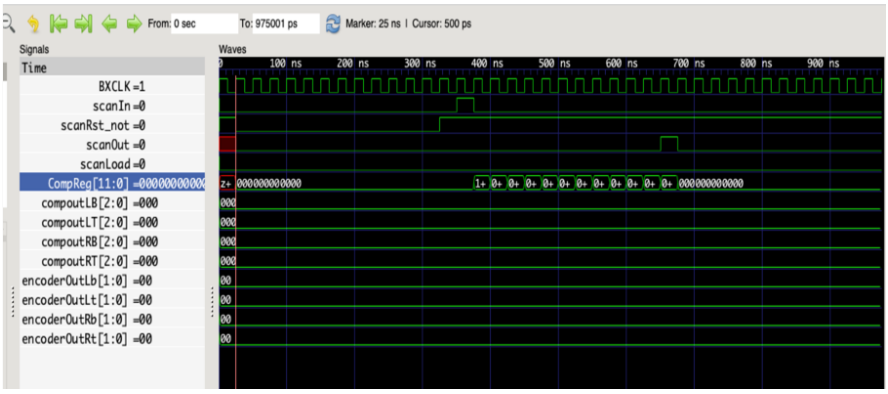


FIGURE 4.13. Update Test 1: Scan Chain

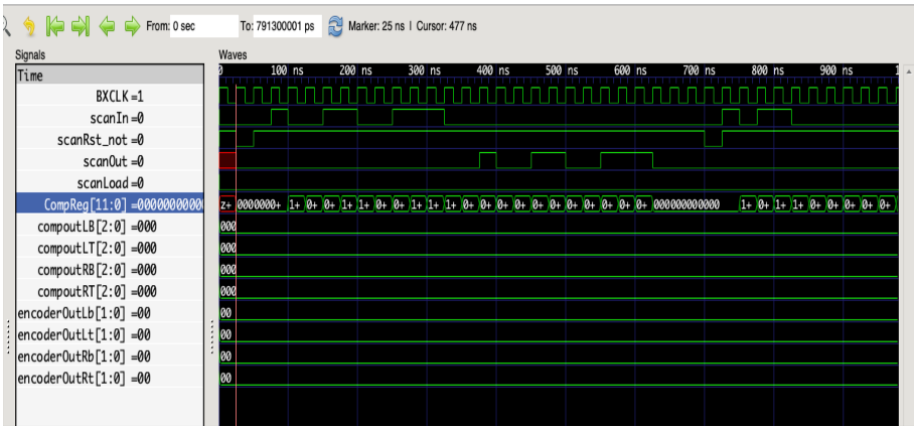


FIGURE 4.14. Update Test 2: Scan Chain

To correct faults associated with `testselect` and `encoderOut` for the left top, right bottom, right top, and left bottom pixels, `testselect` and `encoderOut` were converted from 1D to 2D arrays, with specific rows and columns designated for these signals. Figures 4.11 and 4.12 display the simulations for the shift register, while Figures 4.13 and 4.14 depict the simulations for the scan chain. These waveform diagrams demonstrate that the undefined logic values present in the previous version of the RTL chip are no longer apparent. Additionally, the updated version of the RTL code also generated clean synthesis reports. A clean synthesis report as well as cleared undefined values, was sufficient to mark the fault as corrected.

Further simulations and testbenches were created to examine the entire ASIC. Additional testbenches included ones for the adder and the top module. The adder incorporated signals propagated

through the binary-thermometric encoder. These additional testbenches provided more coverage and a broader view of the signals for `testselect` and `encoderOut` propagating through the ASIC and the rest of the modules.

4.5. Ripple Carry Adder

Adders within the chip architecture process the signals generated by charged particles interacting with the sensor. When a charged particle traverses the sensor array, it deposits charge in multiple pixels, forming a spatial pattern known as a charge cluster [9]. These adders are crucial for summing the charges collected across these pixels to determine the overall charge of the cluster.

The function of the adders is to sum the charges captured by individual pixels within a predefined cluster region. This integrated approach enables the ASIC to perform preliminary filtering of the data at the sensor level, using deep learning algorithms to distinguish between low and high-momentum particles [9]. By reducing the data volume at the source, these adders contribute significantly to bandwidth management.

The adders were ripple carry adders (RCAs), composed of several full adders. Each full adder combines two binary digits and a carry-in bit to produce a sum and a carry-out bit. Figure 4.15 represents the gate-level description of a full adder. Within an RCA, each full adder is constructed from basic logic gates: two XOR gates for sum calculation, two AND gates, and one OR gate for carry-out determination. The first XOR gate computes the preliminary sum of the input bits, while the second XOR gate combines this result with the carry-in to produce the final sum output. The AND gates generate signals indicating conditions where a carry should be generated, and the OR gate combines these signals to produce the final carry-out [10]. This configuration ensures that each bit's addition depends on the carry generated from the preceding lower bit, creating a sequential ripple of carry from the least to the most significant bit. Figure 4.16 shows a ripple carry adder composed of multiple full adders.

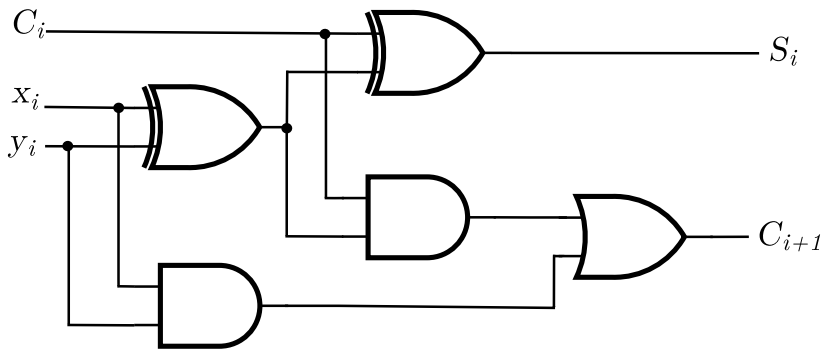


FIGURE 4.15. Full Adder Gate Level

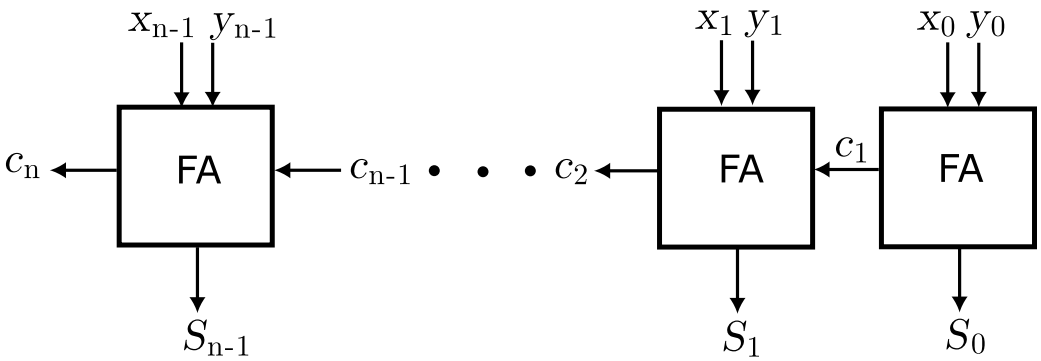


FIGURE 4.16. Ripple Carry Adder

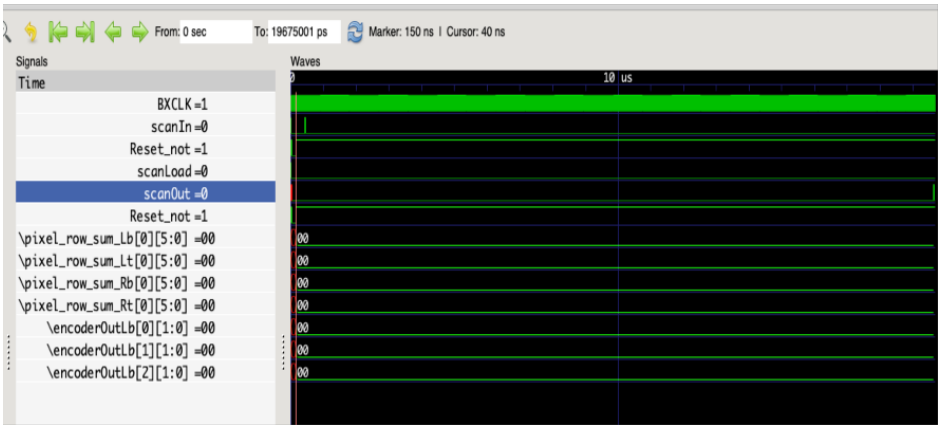


FIGURE 4.17. Test 1. Sending a single bit

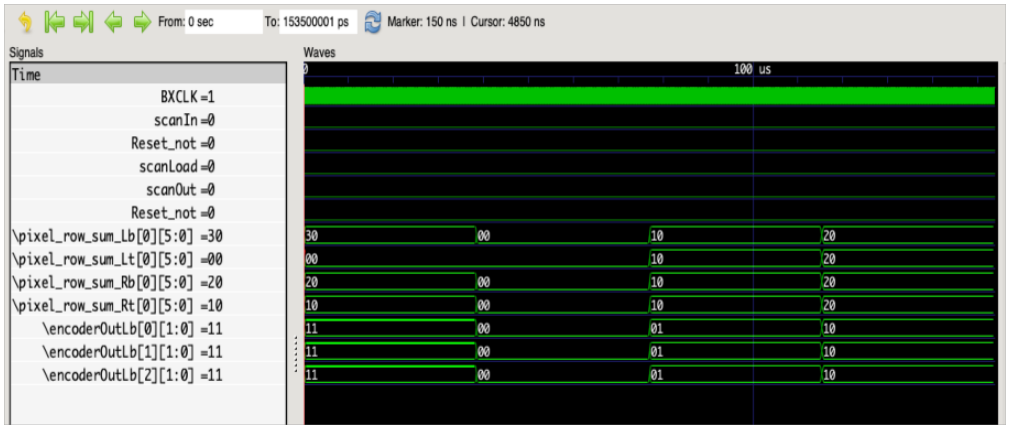


FIGURE 4.18. Test 2. Sending test vectors from pre-defined file

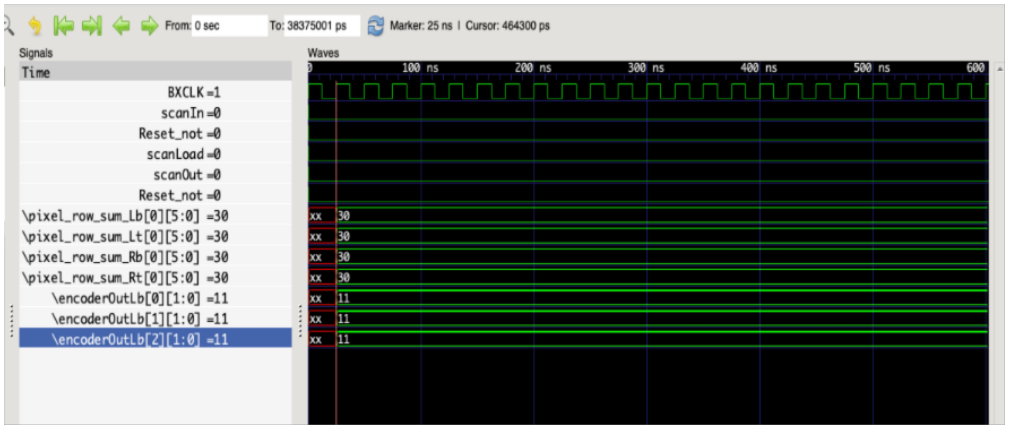


FIGURE 4.19. Test 3. Sending all 1's test vector

After faults were corrected for `testselect` and `encoderOut`, additional simulations were conducted to observe the `encoderOut` signals and their propagation to the output signals `pixelrowsum`. Figure 4.17 shows the initial test where a single bit was sent from `scanIn` and the output observed through `scanOut`. Unlike scan-chain testing, this single bit was sent from a different module within the design to ensure there is no disconnection and the bit is passed through the design. Figure 4.18 then tests the functionality of the RCA by sending test vectors through the input signal `encoderOut` and observing their sum through the output signal `pixelrowsum`. The waveform diagrams confirm that there is no disconnection and the input test vectors can be propagated throughout the module. Figure 4.19 shows the test where 12-bit all 1's vector was sent in through the `encoderOut` and seen at the output of `pixelrowsum`.

4.6. Top Module

For these verification tasks, we refer to the top module as the highest level of hierarchy within this chip's design. This module includes all lower-level modules, including the ripple carry adder, shift register, scan chain, DNN, and encoder. The primary goal of conducting verification through the top module is to ensure that all intermediate nodes and signals are correctly connected for the entire design, as opposed to verifying unit level signal connections.

For this verification task, we simulated the entire chip, which includes all 5,164 shift registers to verify their functionality before and after synthesis. The ASIC design team assigned specific tasks for this module. The first task was to verify that the output ConfigOut behaves correctly, toggling after 5,164 clock cycles following a pulse on the input ConfigIn. Next, we confirmed that we could reset all the registers using the top reset input.

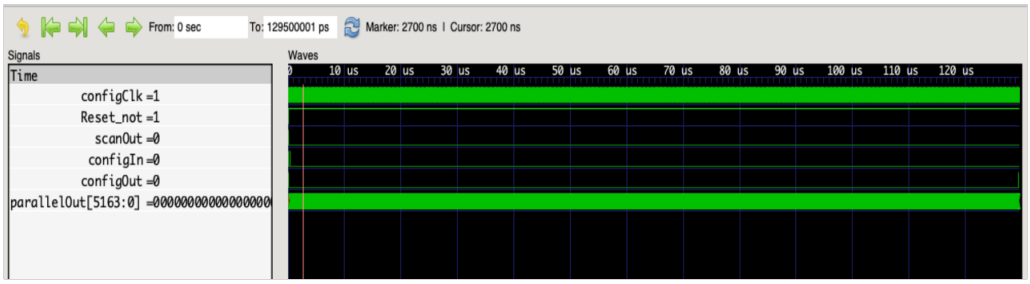


FIGURE 4.20. Test 1. Top module single bit test

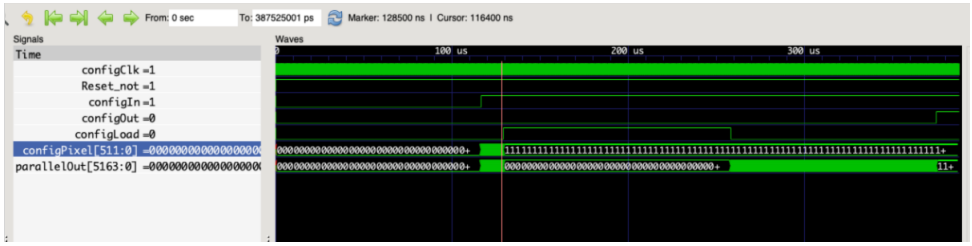


FIGURE 4.21. Test 2. Testing Parallel Out

Figure 4.20 shows a single bit being propagated through the entire length of the ASIC. Although simple, this test provided significant assurance that all intermediate nodes were connected and the bit could propagate throughout lower-level modules. Figure 4.21 shows the `Reset_not` signal

being toggled through the top module, enabling the reset of all lower-level modules. Resetting from the top-level modules ensures that all sub-modules start from a known, predictable initial state. Effective reset mechanisms are also crucial for debugging, as they allow verification engineers to bring the system back to a defined state for testing after modifications or when errors occur. This capability is invaluable in isolating and diagnosing specific issues within the system, significantly streamlining the troubleshooting process.

4.7. Updates

The updates to the ASIC chip, specifically the version 1 as of November 2023, reveal significant changes. In the initial version of the ASIC RTL, logic waveform simulations identified undefined signal values for `scanOut` and `CompReg`. Extensive simulations showed that the issues originated from how `testselect` and `encoderOut` for pixels coordinates top left, top right, right bottom, and left bottom were instantiated. Both signals, `testselect` and `encoderOut`, initially included only row values and were modified to a 2D signal array to incorporate both rows and columns for the pixel chip. This mismatch indicated potential issues in data coordination at these specific array positions

To address these initialization challenges, a `Resetnot` signal has been initiated across several modules including the top module `cms28smartpix`, `superPixelv1`, and the scan chain module `CMSPIXDIGx4v1`. Resetting these registers is a crucial step towards ensuring that the chip starts in a known state, which does not include high impedance or undefined logic values in the waveform diagrams, thereby enhancing the reliability and predictability of its operations.

Additionally, significant modifications were made in the clocking and scanning mechanisms within the chip. The `scanCLK` signal has been completely removed from the scan chain, and the system now solely relies on `BXCLK`. This change likely aims to simplify the clocking architecture or resolve previous synchronization issues but requires careful integration to ensure that the remaining clock signal `BXCLK` can adequately support the scan operations without introducing latency or timing errors.

The removal of `scanCLK` and the emphasis on using `BXCLK`, along with the systematic reset implemented across the modules, help in addressing and fixing the previously observed undefined logic values in the scan chain. These updates aim to enhance system functionality and simplify the clocking mechanism and also ensure that the chip can function without the intermittent undefined states that appeared previously in the scan chain. These updates reflect ongoing efforts to optimize ASIC chip designs for enhanced performance and dependability.

4.8. Future Works

Enhancing the verification process for each module within the ASIC design can significantly improve the overall robustness and reliability of the chip. Furthermore, developing more comprehensive testbenches that include a wider range of edge cases, such as signal overflow, will ensure that modules are resilient under all operational circumstances. Specifically, each module could be tested for its response to maximum and minimum input values to simulate overflow scenarios, which are critical for ensuring that the system maintains stability and accurate output under extreme conditions.

CHAPTER 5

Conclusion

This thesis covered the design and verification of Application-Specific Integrated Circuits (ASICs) for the Large Hadron Collider (LHC) at Fermilab. The development of a specialized chip that integrated both analog and digital components to preprocess and filter out low-energy particle data at the source addresses a critical need. This innovative approach not only optimizes data handling and processing bandwidth but also sets a new standard in the realm of particle physics research instrumentation.

By implementing a chip with dual analog circuits within a superpixel configuration, the ASIC Design Group at Fermilab has successfully met the needs of high-energy physics with the advanced capabilities of modern microelectronics. The 16x16 pixel array, combined with an embedded neural network, facilitates rapid on-chip data classification, significantly reducing the volume of data that requires off-chip processing. This not only speeds up data analysis but also enhances the accuracy and efficiency of capturing relevant high-energy events.

Throughout the design process, functional verification methodologies were essential to ensure the chip's reliability and functionality. The application of both random and deterministic verification simulations served to test the chip under a wide range of scenarios, ensuring robustness and operational integrity. The introduction of formal verification methods like theorem proving and model checking provided a theoretical foundation for asserting the correctness of the design, despite not being implemented in this particular ASIC.

The integration of design and verification techniques highlights the role of ASICs in advancing scientific research tools. The capability to process and analyze data efficiently at the source without the need for extensive off-chip resources is paramount. This approach not only improves the data acquisition process but also reduces the latency and energy consumption associated with data

processing in large-scale physics experiments.

Throughout the verification process, several critical ASIC design errors were identified, particularly in the scan-chain and shift register modules. One of the primary issues was the presence of undefined logic signals within the scan-chain module. These undefined values, appearing during the initialization and operation phases, were causing unpredictable behaviors and potentially compromising the stability and performance of the entire system.

To address these issues, functional testing was used which included the use of predefined test vectors and random testing scenarios, which facilitated the detection of faults at various operational stages. This testing ensured that all potential faults were identified and corrected before final implementation.

Significant updates o the chip design were made in response to the issues identified. One major update was the removal of the `scanCLK` signal within the scan-chain module, which simplified the clocking architecture and aimed to eliminate synchronization issues. This change required careful adjustment to ensure the remaining clock signal, `BXCLK`, adequately supported the scan operations without introducing latency or timing errors.

Furthermore, a `Resetnot` signal was introduced across several modules, including the cms28smartpixv1, superPixelv1, and CMSPIXDIGx4v1 modules. This reset mechanism was crucial for clearing any irregular states or data that might have persisted in the configuration registers, thereby ensuring the chip started in a known good state and enhancing the overall reliability of operations.

APPENDIX A

Verification Outputs

The verification results for the RTL modules within the CMS Pixel ASIC was performed through functional verification. Each unit module was individually analyzed and tested before these modules were integrated and tested collectively. Upon successful verification, the RTL was then synthesized, and the resulting images included in this appendix highlight the synthesized outcomes.

A.1. Shift Register

The shift register serves primarily to sequence digital data streams by shifting bits through a series of flip-flops, each representing a register element. Testing these registers involved sending a single bit through the system to ensure proper data transmission from input to output. The results from the synthesized RTL are included in Figures A.1 and Figures A.2.

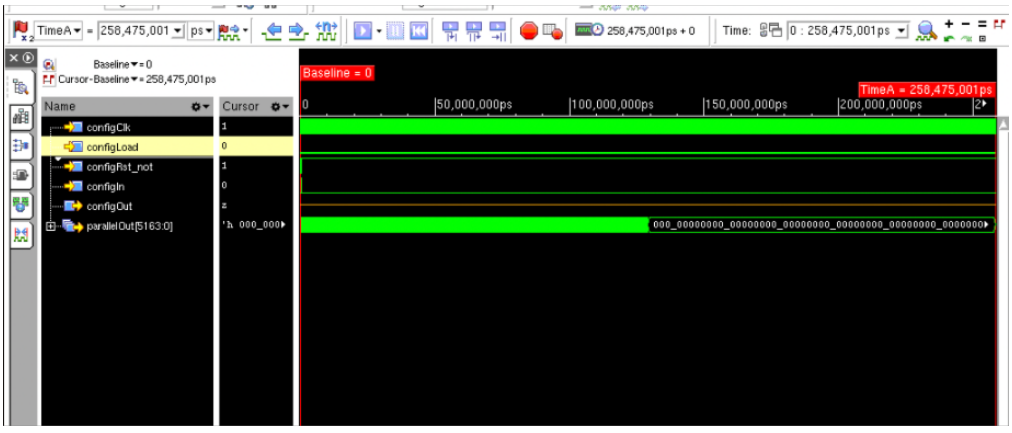


FIGURE A.1. Test 1: synthesized shift register

FIGURE A.2. Test 2: synthesized shift register

A.2. Scan-Chain

The scan chain primarily facilitates the testing and debugging of digital circuits by providing a mechanism to bypass normal signal paths and directly observe register values. The results for the synthesized results are included Figures A.3, Figures A.4, and Figures A.5.

FIGURE A.3. Test 1: synthesized scan-chain

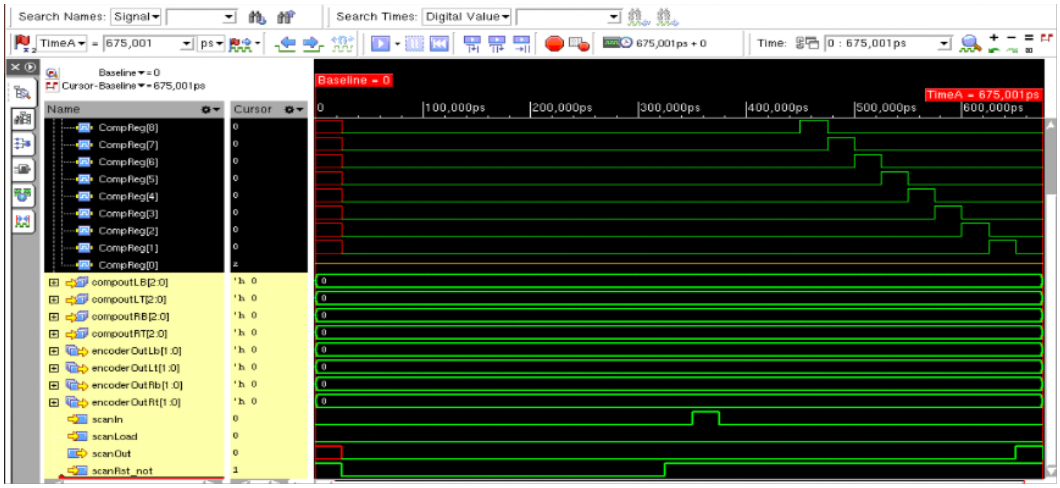


FIGURE A.4. Test 2: synthesized scan-chain

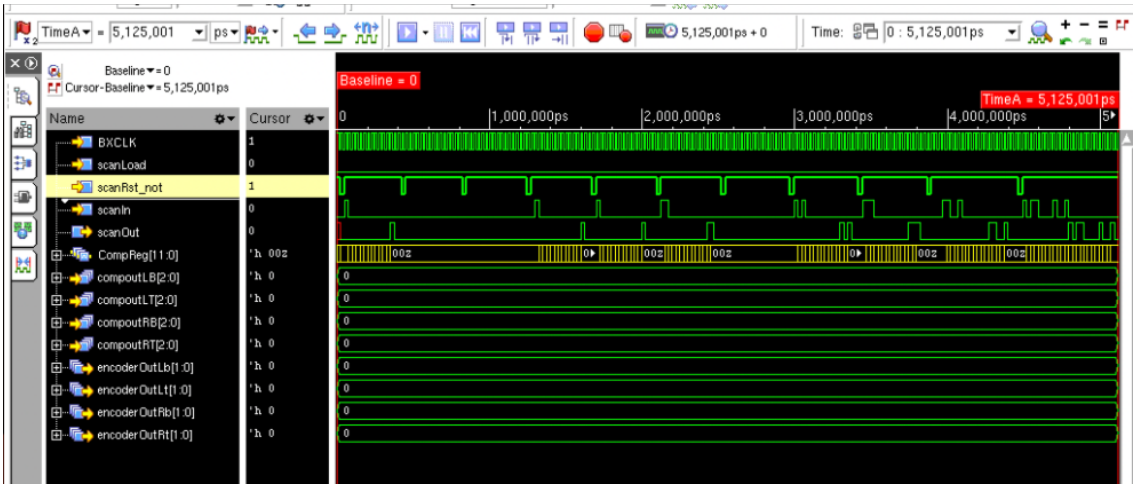


FIGURE A.5. Test 3: synthesized scan-chain

A.3. Ripple Carry Adder

To validate the Ripple Carry Adder (RCA), a series of tests were performed. Initially, single-bit inputs were passed through to monitor the carry propagation across all stages, confirming the functionality of each full adder. Subsequently, a range of predetermined and random test vectors were used to simulate normal and extreme conditions, ensuring accurate processing of inputs and robustness under stress. The synthesized results are included in Figures A.6, Figures A.7, and Figures A.8.

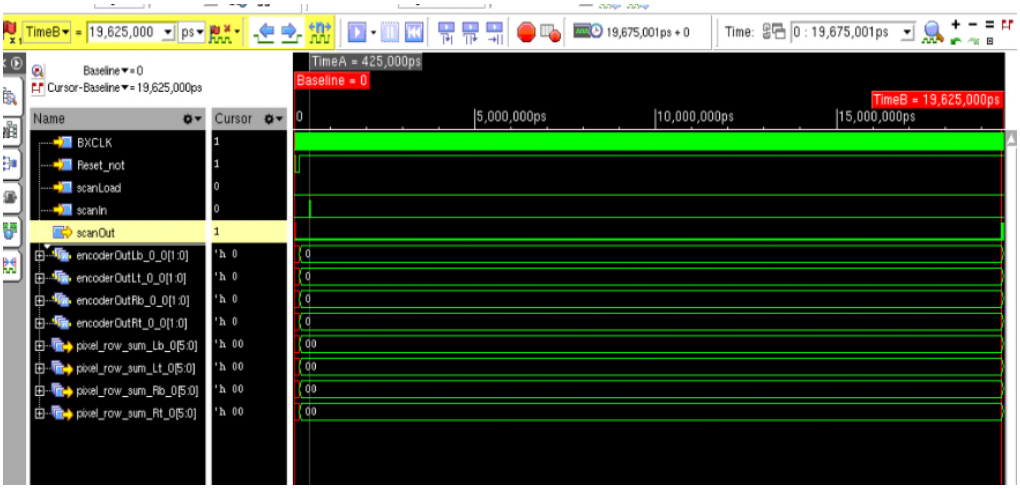


FIGURE A.6. Test 1. synthesized RCA

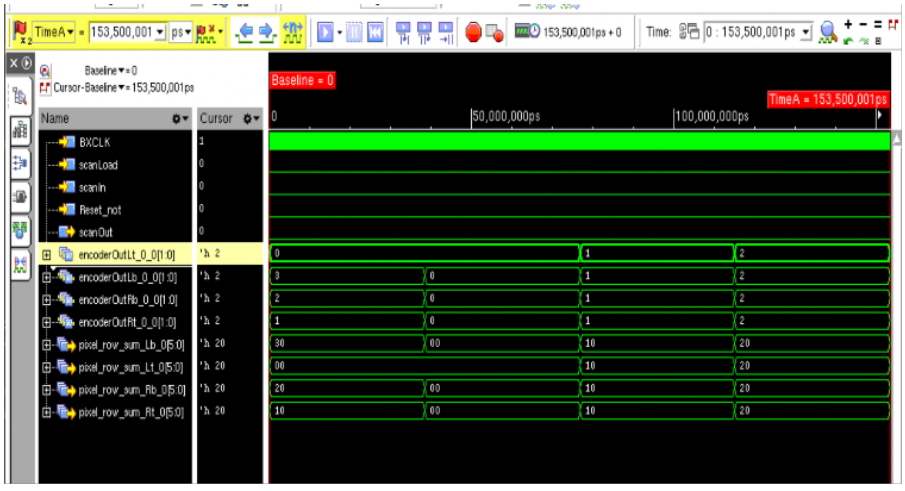


FIGURE A.7. Test 2. synthesized RCA

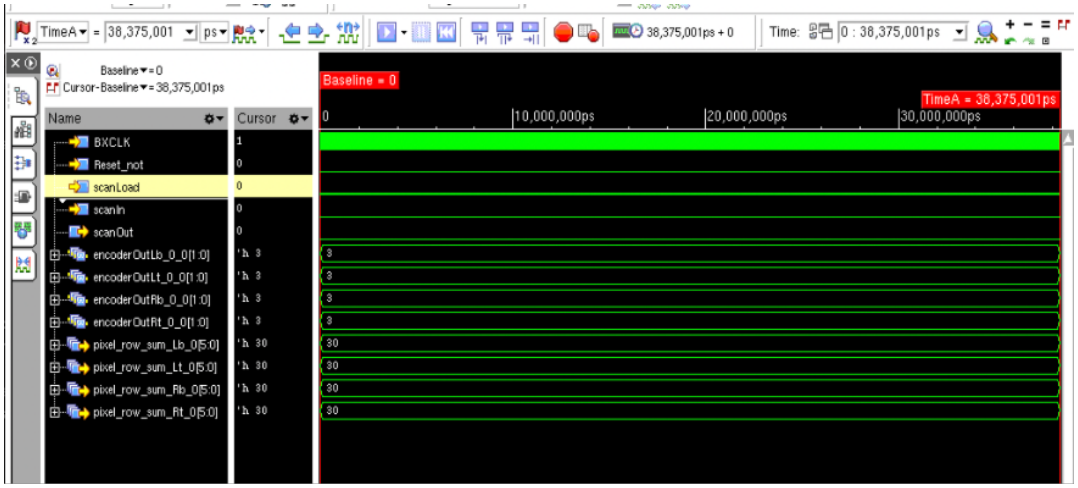


FIGURE A.8. Test 3. synthesized RCA

A.4. Top Module

For the top module in the ASIC design, testing focused on verifying the integration and interaction of all sub-modules, ensuring that they functioned cohesively as a single unit. The initial test involved sending a single bit through the entire system from the input to the output, tracking its correct propagation across all integrated modules. This test was crucial for verifying signal integrity and connectivity throughout the chip. The results for the synthesized RTL are included in the following images, Figures A.9 and Figures A.10.

FIGURE A.9. Test 1: synthesized top module

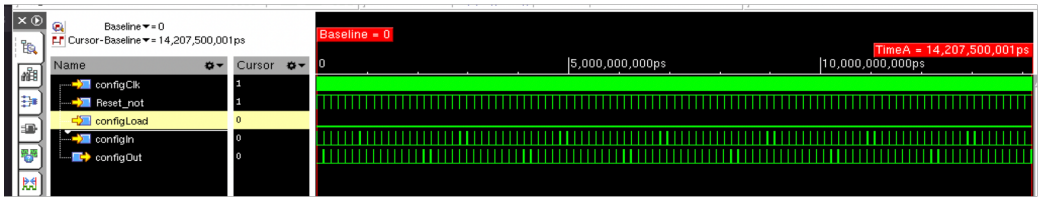


FIGURE A.10. Test 2: synthesized top module

Bibliography

[1] G. D. Guglielmo, F. Fahim, C. Herwig, M. B. Valentin, J. Duarte, C. Gingu, P. Harris, J. Hirschauer, M. Kwok, V. Loncar, Y. Luo, L. Miranda, J. Ngadiuba, D. Noonan, S. Ogrenci-Memik, M. Pierini, S. Summers, and N. Tran, "A reconfigurable neural network ASIC for detector front-end data compression at the HL-LHC," *IEEE Transactions on Nuclear Science*, vol. 68, no. 8, pp. 2179–2186, 2021.

[2] G. Carini, G. Deptuch, J. Dickinson, D. Doering, A. Dragone, F. Fahim, P. Harris, R. Herbst, C. Herwig, J. Huang *et al.*, "Smart sensors using artificial intelligence for on-detector electronics and ASICs," *arXiv preprint arXiv:2204.13223*, 2022.

[3] S. Chatrchyan, E. de Wolf, P. Van Mechelen *et al.*, "The CMS experiment at the CERN LHC," *Journal of Instrumentation.-Bristol, 2006, currens*, vol. 3, p. S08004, 2008.

[4] N. H. Weste and D. Harris, *CMOS VLSI Design: a Circuits and Systems Perspective*. Pearson Education India, 2015.

[5] C.-J. Seger, *An Introduction to Formal Hardware Verification*. University of British Columbia, Department of Computer Science, 1992.

[6] L.-C. Wang, M. Abadir, and N. Krishnamurthy, "Automatic generation of assertions for formal verification of powerpc/sup tm /microprocessor arrays using symbolic trajectory evaluation," in *Proceedings 1998 Design and Automation Conference. 35th DAC. (Cat. No.98CH36175)*, 1998, pp. 534–537.

[7] C. Pixley, N. R. Strader, W. Bruce, J. Park, M. Kaufmann, K. Shultz, M. Burns, J. Kumar, J. Yuan, and J. Nguyen, "Commercial design verification: Methodology and tools," in *Proceedings International Test Conference 1996. Test and Design Validity*. IEEE, 1996, pp. 839–848.

[8] T. Aarrestad, V. Loncar, N. Ghielmetti, M. Pierini, S. Summers, J. Ngadiuba, C. Petersson, H. Linander, Y. Iiyama, G. Di Guglielmo *et al.*, "Fast convolutional neural networks on fpgas with hls4ml," *Machine Learning: Science and Technology*, vol. 2, no. 4, p. 045015, 2021.

[9] J. Yoo, J. Dickinson, M. Swartz, G. Di Guglielmo, A. Bean, D. Berry, M. B. Valentin, K. DiPetrillo, F. Fahim, L. Gray *et al.*, "Smart pixel sensors: Towards on-sensor filtering of pixel clusters with deep learning," *arXiv preprint arXiv:2310.02474*, 2023.

[10] S. D. Brown, Z. G. Vranesic *et al.*, *Fundamentals of Digital Logic with Verilog Design*. McGraw-Hill New York, 2003, vol. 1.

[11] S. Taylor, M. Quinn, D. Brown, N. Dohm, S. Hildebrandt, J. Huggins, and C. Ramey, "Functional verification of a multiple-issue, out-of-order, superscalar alpha processor—the DEC Alpha 21264 microprocessor," in *Proceedings of the 35th Annual Design Automation Conference*, 1998, pp. 638–643.

[12] *IEEE Standard VHDL Language Reference Manual*, Institute of Electrical and Electronics Engineers (IEEE) Std., 2002.

[13] A. M. Deiana, N. Tran, J. Agar, M. Blott, G. Di Guglielmo, J. Duarte, P. Harris, S. Hauck, M. Liu, M. S. Neubauer *et al.*, "Applications and techniques for fast machine learning in science," *Frontiers in big Data*, vol. 5, p. 787421, 2022.