UNDERGRADUATE FINAL YEAR PROJECT REPORT

Department of Computer & Information Systems Engineering

NED University of Engineering and Technology

Hardware Acceleration of CRYTALS-DILITHIUM for Post-Quantum Cryptography

Group Number: 29 **Batch: 2022 - 2026**

Group Member Names:

Aqiba Abdul Qadir	CS-22003
Mahwish Hussain	CS-22016
Tooba Aftab	CS-22020
Neha Nauman Khan	CS-22024

Approved by

……………………………………………………………………………………

Ms Fauzia Yasir

Lecturer at CIS Department, NEDUET

Project Advisor

[THIS PAGE IS INTENTIONALLY LEFT BLANK]

Author's Declaration

We declare that we are the sole authors of this project. It is the actual copy of the project that was accepted by our advisor(s) including any necessary revisions. We also grant NED University of Engineering and Technology permission to reproduce and distribute electronic or paper copies of this project.

Signature and Date	Signature and Date	Signature and Date	Signature and Date
…………………….	………………………	…………………….	……………………..
Aqiba Abdul Qadir	Mahwish Hussain	Tooba Aftab	Neha Nauman Khan
CS-22003	CS-22016	CS-22020	CS-22024
aqiba4530087@cloud.neduet,edu.pk	*hussain4503445@cloud.neduet,edu.pk*	*aftab4502649@cloud.neduet,edu.pk*	*khan4531086@cloud.neduet,edu.pk*

Statement of Contributions

Mahwish Hussain

- ✓ Build Python based reference model of Schnorr’s Algorithm
- ✓ Performed Testing and Simulations of Modules

Tooba Aftab

- ✓ Implemented Pseudo Random Number Generator Module in Verilog
- ✓ Implemented Key Generation Module in Verilog

Neha Nauman Khan

- ✓ Integrated SHA256 Open Source Hash IP
- ✓ Implemented Signature Generation Module in Verilog

Aqiba Abdul Qadir

- ✓ Created Signature Verification Module in Verilog
- ✓ Top File integration and Synthesis.

Executive Summary

With rapid advancement in the field of quantum-computing, threat to classical cryptography such as AES, RSA, ECC etc. is increasing. The hardness of these algorithms depend upon **integer factorization** and **discrete logarithm** which can be solved by quantum computers within no time. This threat provoked **NIST (National Institute of Standards & Technology)** to design and standardize quantum-safe algorithms for key encapsulation and authentication. NIST declared **CRYSTALS-DILITHIUM** (ML-DSA: Module Lattice Based Digital Signature Algorithm) as a standard signature scheme for post-quantum cryptography.

This project focuses on the design and implementation of a **hardware accelerator for CRYSTAL-DILITHIUM digital signature scheme** on FPGA/SoCs. Since Lattice based mathematics is complex and require massive parallelism, it cannot be implemented on CPUs. Therefore, we proposed a solution to have a **co-processor** for the acceleration of PQC algorithms interfaced via AXI to a processing system. This is a hardware-software co-design approach where PS(Processing System) part is responsible for sending and receiving messages, keys, siglets, etc from the external world and control the PL(Programmable Logic) part. The PL part performs **signature generation** and verification using CRYSTAL-DILITHIUM algorithm.

For the sake of simplicity, the mid year phase of this project implements **Schnorr's Signature Scheme**, which shares the same **Fiat-Shamir Transform** based procedure as **Dilithium**. This helps us understand the overall architecture and data flow of the project without being concerned of the complex Lattice based computations. Having the complete data path of Schnorr's algorithm implemented on FPGA will enable us to easily make transition to Lattice based cryptography of CRYTALS-DILITHIUM in the later stages.

In conclusion, this project helps in accelerating lattice-based cryptography algorithms. Due to the complexity of algorithm, hardware accelerator-based approach is used to enhance performance and reduce latency in overall design. Because high latency can proved to be very costly in security and authentication systems.

Acknowledgments

In the name of ALLAH, the Most Gracious, the Most Merciful. All praises belongs to Almighty ALLAH. Indeed, we can't do anything without His guidance and blessings. He gave us ability and strength to complete this project. After ALLAH (swt), we would like to express our sincere gratitude to our project supervisors Dr Majida Kazmi (Professor at CIS Department, NEDUET), Ms Fauzia Yasir(Lecturer at CIS Department, NEDUET), and Dr Fahim ul Haq (Assistant Professor at Telecommunications Engineering Department, NEDUET) for their continuous guidance and valuable feedback through out the course of this project. We owe our achievements to our truly sincere and loving parents, guardians, friends and family members. Finally, we would like to thank our Department's staff and available resources to make conductive environment for the successful completion of this project.

Dedication

This work is dedicated to our parents and teachers for their constant prayers and mentorship throughout our journey. It is also dedicated to our beloved country and to the entire Muslim Ummah. We hope that it adds value to technological advancement and contributes to meaningful progress.

Table of Contents

Catalog

United Nations Sustainable Development Goals

The Sustainable Development Goals (SDGs) are the blueprint to achieve a better and more sustainable future for all. They address the global challenges we face, including poverty, inequality, climate change, environmental degradation, peace and justice. There is a total of 17 SDGs as mentioned below. Check the appropriate SDGs related to the project.

- ☐ No Poverty
- ☐ Zero Hunger
- ☐ Good Health and Well being
- ☐ Quality Education
- ☐ Gender Equality
- ☐ Clean Water and Sanitation
- ☐ Affordable and Clean Energy
- ☑ Decent Work and Economic Growth
- ☑ Industry, Innovation and Infrastructure
- ☐ Reduced Inequalities
- ☑ Sustainable Cities and Communities
- ☐ Responsible Consumption and Production
- ☐ Climate Action
- ☐ Life Below Water
- ☐ Life on Land
- ☑ Peace and Justice and Strong Institutions
- ☐ Partnerships to Achieve the Goals

Chapter 1 Introduction

1.1 Background Information

The exponential growth in computational power over the last several decades has fundamentally transformed modern digital life, yet it poses an existential threat to the very systems designed to secure it. The current global digital infrastructure relies heavily on **Public-Key Cryptography (PKC)** schemes like RSA and Elliptic Curve Cryptography (ECC) for tasks such as digital signatures, secure communication (e.g.HTTPS), and key exchange. The security of these schemes is predicated on the mathematical difficulty of solving problems like the integer factorization problem or the discrete logarithm problem on classical computers.[9]

However, the theoretical advent of a large-scale **quantum computer**, capable of executing Shor's algorithm, is poised to shatter the foundations of classical PKC. Shor's algorithm can solve the underlying mathematical problems of RSA and ECC in polynomial time, rendering them completely insecure. This impending reality often referred to as the **"Quantum Threat"**necessitates an urgent and global migration to new cryptographic primitives that are inherently resistant to quantum attacks.[1] This new area of research is known as **Post-Quantum Cryptography (PQC)**.

In response to this critical challenge, the National Institute of Standards and Technology (NIST) initiated a multi-year standardization process to identify, evaluate, and select a suite of quantum-resistant algorithms. Among the algorithms selected for standardization, **CRYSTALS-Dilithium** has emerged as a key candidate for the digital signature standard. Dilithium is a **lattice-based cryptographic scheme** whose security is rooted in the presumed hardness of the Module Learning with Errors (Module-LWE) problem, a mathematical challenge that is believed to remain intractable even for quantum computers.[1] This project focuses on addressing the computational demands of this vital, quantum-resistant digital signature algorithm.

1.2 Significance and Motivation

The transition to quantum-safe algorithms like CRYSTALS-Dilithium is not merely an upgrade; it is a fundamental shift required to secure future digital communications and data integrity. This project's significance is two fold: addressing the **global security imperative** and tackling the **implementation challenges** of PQC.

1.2.1 Global Security and Timeliness

The most direct motivation is the urgent need for robust, quantum-resistant security in critical applications. All systems, from national security infrastructure and financial transactions to protected health information and device firmware updates, require digital signatures to verify authenticity and integrity. By focusing on Dilithium, this project contributes directly to the deployment of a **NIST-selected, quantum-secure signature standard**. [9] Delaying the preparation for the quantum era is not an option, as data encrypted today, known as "harvest now, decrypt later," could be stored and compromised by a future quantum computer.

1.2.2 The Implementation and Performance Challenge

While quantum-safe, lattice-based cryptography presents a new set of computational demands that differ significantly from its classical predecessors. The mathematical operations in Dilithium are based on **polynomial arithmetic in lattice structures**, requiring extensive, highly parallel computation. Key generation, signature generation, and verification involve complex operations like Number Theoretic Transform (NTT), polynomial multiplications, and modular arithmetic.[3]

- ✓ **Lattice-based computations demand massive parallelism and high data throughput.**
- ✓ **Central Processing Units (CPUs)**, while versatile, are often unable to manage these heavy workloads efficiently due to their sequential architecture.
- ✓ This performance bottleneck is a critical hurdle for the widespread deployment of Dilithium in environments requiring high-speed processing, such as servers handling millions of transactions per second or resource-constrained embedded systems.

The motivation for this project is therefore to overcome this limitation through **hardware acceleration**. By designing and implementing a co-processor architecture, we aim to enhance the operational speed and energy efficiency of the Dilithium algorithm, making its adoption practical and feasible for real-world application.

1.2.3 Utilizing Foundational Architecture (Schnorr's Algorithm)

To establish a solid foundation and validate the core architectural principles, this project begins by understanding and implementing the datapath of **Schnorr's Algorithm**. Although Schnorr's algorithm is *not* quantum-safe (it relies on the Discrete Logarithm Problem), it shares structural similarities with the fundamental components of CRYSTALS-Dilithium. The initial focus on a recognized architecture allows for the development and optimization of a high-performance scalar arithmetic unit, which is a necessary building block before scaling up to the polynomial arithmetic required by Dilithium. This staged approach ensures a robust, end-to-end understanding of the signature architecture before transitioning to the complexities of the lattice-based scheme.[8]

1.3 Aims and Objective

The primary aim of this project is to design, implement, and evaluate a high-performance **hardware accelerator** for the CRYSTALS-Dilithium digital signature algorithm to mitigate the computational overhead associated with post-quantum cryptography.[2]

The specific objectives that will be accomplished to achieve this aim are:

1.3.1 Architectural Design and Implementation

- ✓ To thoroughly study the complete end-to-end signature architecture of CRYSTALS-Dilithium, identifying the most computationally intensive kernels (e.g., polynomial multiplication via NTT).[3]

- ✓ To design a parallel, efficient hardware architecture (co-processor/datapath) optimized for the polynomial and modular arithmetic operations inherent in Dilithium.
- ✓ To implement the designed architecture using a Hardware Description Language (HDL), targeting a Field-Programmable Gate Array (FPGA) platform for proof-of-concept and performance validation.

1.3.2 Functional Verification and Validation

- ✓ To first implement and verify the core datapath of Schnorr's Algorithm to establish and validate the foundational signature architecture and arithmetic units.
- ✓ To functionally verify the final Dilithium hardware accelerator against known test vectors to ensure compliance with the NIST standard specification.[9]

1.3.3 Performance Evaluation

- ✓ To evaluate the performance of the implemented accelerator in terms of execution time (latency) for Key Generation, Signature Generation, and Signature Verification.
- ✓ To compare the achieved performance metrics (e.g., throughput and latency) against comparable software and other hardware implementations, demonstrating the efficiency and operational gains provided by the hardware acceleration.

1.4 Methodology

The project will follow a structured design flow, moving from theoretical understanding to practical implementation and performance evaluation.

1.4.1 Phase 1: Literature Review and Architectural Study

A detailed analysis of the CRYSTALS-Dilithium specification, including its mathematical foundations (Module-LWE) and the underlying components such as NTT and polynomial arithmetic. A comparative study will also be conducted on the structural similarities and differences between Dilithium and Schnorr's algorithm (Discrete Logarithm Problem vs. Lattice-based).

1.4.2 Phase 2: Schnorr's Hardware Implementation

A detailed analysis of the CRYSTALS-Dilithium specification, including its mathematical foundations (Module-LWE) and the underlying components such as NTT and polynomial arithmetic. A comparative study will also be conducted on the structural similarities and differences between Dilithium and Schnorr's algorithm (Discrete Logarithm Problem vs. Lattice-based).

We implement the three core pillars of the Schnorr scheme using Verilog HDL

Key Generation (*key_gen*): Utilizing a *prng* module to generate a private key and a *mod_exp* module to derive the public key
$P = g^x (\text{mod p})$..(1.1)

Signature Generation (*sign_gen*): A 6-stage FSM (*START, NONCE, PUBNONCE, CHALL, MUL, ADD*) that coordinates the interaction between the *PRNG*, the *SHA-256 hash core*, and modular multipliers.

Signature Verification (*sign_ver*): A verification engine that re-computes the challenge and verifies the mathematical identity.

$g^s{=}R\,(\text{mod}\,p)$..(1.2)

1.4.3 Functional Verification and Comparison

The design is simulated using a timescale 1ns/1ps to ensure timing accuracy. We verify that the valid_sign and valid_ver signals assert correctly for valid messages. This phase also includes a comparative analysis of the **Schnorr datapath** versus the requirements for **CRYSTALS-Dilithium**, specifically comparing scalar multiplication to polynomial Number Theoretic Transforms (NTT).

Chapter 2 Literature Review

2.1 Background Information

Post-quantum cryptography has become a critical area of research as the expected capabilities of quantum computers threaten the security of widely used public-key cryptosystems. Algorithms based on integer factorization and discrete logarithms are particularly vulnerable, prompting the need for quantum-resistant alternatives. In response, the National Institute of Standards and Technology (NIST) standardized several post-quantum schemes, among which CRYSTALS-Dilithium has emerged as a leading digital signature algorithm due to its strong security foundation, moderate key and signature sizes, and implementation efficiency. Built on lattice-based cryptography, Dilithium primarily uses structured polynomial arithmetic and modular computations, making it well suited for hardware acceleration while still imposing strict constraints on area, latency, and correctness.

This literature review briefly surveys prior work on post-quantum cryptographic algorithms, with a specific focus on hardware implementations of CRYSTALS-Dilithium. Existing studies show a clear progression from basic, proof-of-concept designs to more optimized architectures aimed at reducing hardware cost and execution time without compromising correctness. Most of the reported work explores trade-offs between area utilization and latency, along with efficient handling of polynomial operations. These contributions form the technical basis for current Dilithium accelerators and highlight remaining gaps that motivate the design choices adopted in this project.

2.2 Foundational Algorithm Specification: Ducas et al. (2018)

Ducas et al. (2018) [1] introduced Dilithium, a lattice-based signature scheme from the CRYSTALS suite, designed for strong security and high efficiency in the post-quantum era,This foundational work establishes CRYSTALS-Dilithium as a lattice-based signature scheme built upon the Module Learning With Errors (MLWE) problem. The paper provides complete mathematical specifications, security proofs, and detailed software performance benchmarks across three security levels. However, it contains no discussion of hardware implementation considerations, creating the fundamental research gap that all subsequent papers address.

The algorithm's computational structure relies on three primary operations that present distinct hardware challenges: Number Theoretic Transform (NTT) for polynomial multiplication requiring high-precision modular arithmetic, Keccak (SHA-3) for hashing and random sampling demanding substantial state memory, and rejection sampling with data-dependent execution paths that complicate timing analysis. The software performance data presented establishes crucial baselines that hardware implementations would reference, but the paper's omission of hardware-specific guidance regarding memory hierarchy design, arithmetic precision management, and area-latency trade-offs defines the essential research questions that motivate all subsequent hardware investigations.

Table 2.1 Parmeters and Software Cycles

Parameter Set	**Security Level**	**Public Key Size**	**Signature Size**	**Approximate Software Cycles (Sign)**
Dilithium2	128-bit	1,312 bytes	2,420 bytes	110,000 cycles
Dilithium3	192-bit	1,952 bytes	3,293 bytes	170,000 cycles
Dilithium5	256-bit	2,592 bytes	4,595 bytes	220,000 cycles

2.3 First Hardware Implementation: Area-Constrained FPGA Design by Ricci et al. (2021)

Ricci et al. (2021) presented the first VHDL implementation of the CRYSTALS-Dilithium signature scheme for Field-Programmable Gate Arrays (FPGAs) [2]. Their work addresses the fundamental question of the minimum hardware resources required to implement Dilithium. The architectural strategy employs extreme resource sharing through a single time-multiplexed NTT core and single Keccak engine that sequentially processes all algorithm operations.

The implementation achieves remarkable area efficiency, requiring only 3,950 Look-Up Tables (LUTs) on an Artix-7 FPGA, but reveals severe performance penalties with signing latency exceeding 400,000 cycles. This establishes the area-constrained extreme in the design space and demonstrates the fundamental trade-off between area minimization and computational speed. While demonstrating functional feasibility on constrained platforms, this implementation proves impractical for applications requiring reasonable throughput or latency.

2.3 First Hardware Implementation: Area-Constrained FPGA Design by Ricci et al. (2021)

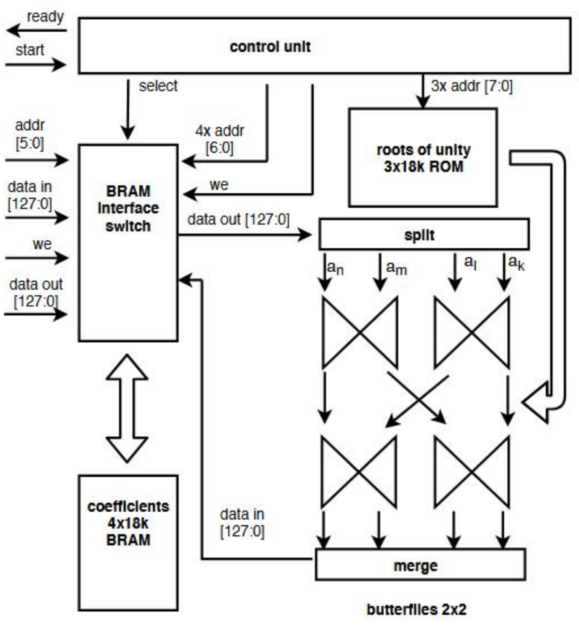


Figure 2.1 NTT implementation for FPGA

Table 2.2 Implementation Results and Analysis

Resource Type	Utilization	Percentage of Artix-7 FPGA	Performance Impact

Look-Up Tables (LUTs)	3,950	2.2%	Enables low-cost deployment
Flip-Flops (FFs)	1,370	0.8%	Limits pipeline depth
DSP Blocks	8	0.9%	Constrains arithmetic throughput
Block RAM (BRAM)	3.5 blocks	1.3%	Restricts memory bandwidth
Maximum Frequency	100 MHz	-	Limited by control complexity
Signing Latency	>400,000 cycles	-	Prohibitive for real-time applications

2.4 Performance-Optimized Implementation: Parallel FPGA Architecture by Land et al. (2021)

Land et al. (2021), published concurrently, present a direct counterpoint to Ricci et al.'s approach by pursuing maximum performance through aggressive parallelization [3]. While Ricci et al. prioritize minimal resource usage, Land et al. focus on the highest achievable performance using modern FPGA technology. Their design incorporates four dedicated NTT cores operating concurrently, dual Keccak processors, and deeply pipelined datapaths, achieving an unprecedented signing latency of only 6,468 cycles at 250 MHz.

However, this performance requires extraordinary resource utilization: 148,237 LUTs (85% of FPGA capacity), 824 DSP blocks (92% utilization), and 128 Block RAMs. This establishes the performance-optimized extreme opposite the area-constrained extreme, demonstrating that while near-software performance is achievable in hardware, it comes at prohibitive area cost for many practical applications. The design directly addresses the latency problem of Ricci et al. but creates a new challenge of excessive resource consumption.

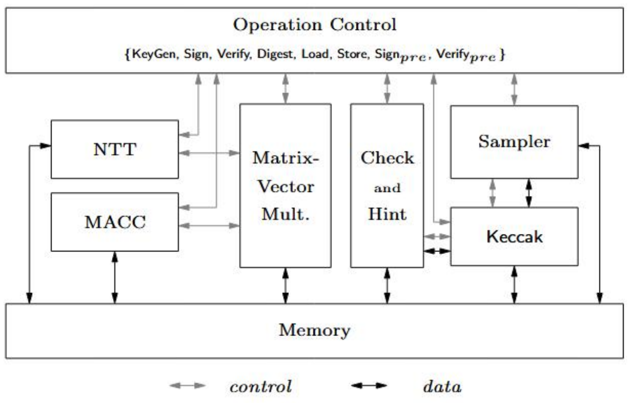


Figure 2.2 Parallel Architecture diagram

Table 2.3 Performance evaluation

Performance Metric	**Value**	**Comparison to Ricci et al. [2]**	**Implementation Cost**
Signing Latency	6,468 cycles	98.4% reduction	148,237 LUTs (37.5× more)
Operating Frequency	250 MHz	2.5× higher	Complex clock distribution
Throughput	38,650 signatures/sec	96.8× higher	824 DSP blocks (103× more)
Initialization Latency	2,560 cycles	Comparable	Additional control logic
Power Consumption	18W @ 250 MHz	22.5× higher	Parallel unit activation

2.5 Balanced ASIC Implementation: Unified Architecture by Zhao et al. (2022)

Addressing the limitations of both previous extremes, Zhao et al. in 2022 introduced an ASIC implementation optimizing for area-time efficiency through architectural innovation [4]. Their key contribution is a unified NTT/INTT butterfly unit that performs both forward and inverse transforms with shared arithmetic resources, reducing control complexity and silicon area while maintaining computational accuracy.

Implemented in 40nm CMOS technology, the design achieves 75,000 cycles latency at 400 MHz within 0.20 mm^2 silicon area, demonstrating a balanced middle ground between previous extremes. The unified butterfly architecture represents significant advancement in arithmetic unit design specifically for cryptographic applications. The work also highlights the energy efficiency advantages of custom silicon, with 0.05W power consumption representing orders-of-magnitude improvement over FPGA approaches.

2.6 Systematic Design-Space Exploration: Beckwith et al. (2021, 2022)

Beckwith et al. in 2021 and 2022 conducted comprehensive design-space exploration across all Dilithium parameter sets, moving beyond individual implementations to systematic analysis of fundamental bottlenecks [5,6]. Their work reveals through detailed cycle analysis that Keccak hashing operations account for 60-70% of total execution cycles, challenging the NTT-focused optimization assumptions of previous implementations.

Their throughput-optimized design achieves 10,000 signatures/second with 30,240 LUTs on Virtex-7 FPGAs, positioning between previous extremes while providing comprehensive parameter scalability. Most significantly, the 2022 extended version explicitly acknowledges the side-channel security gap that all previous implementations ignored, marking a pivotal expansion of design considerations beyond pure performance metrics.

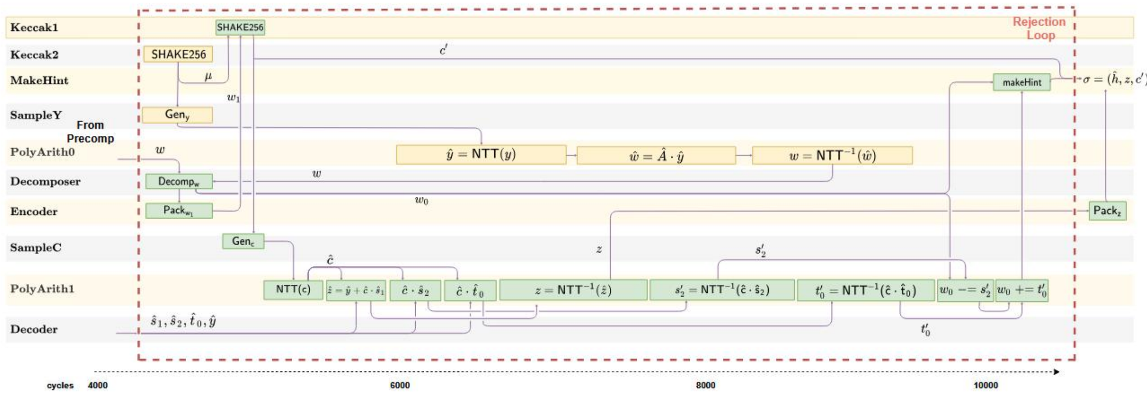

Keccak1
Keccak2
MakeHint
SampleY
PolyArith0
Decomposer
Encoder
SampleC
PolyArith1
Decoder
From Precomp
SHAKE256
SHAKE256
Rejection Loop
c'
μ
w_1
Gen_y
w
$\hat{y} = \mathrm{NTT}(y)$
$\hat{w} = \hat{A} \cdot \hat{y}$
$w = \mathrm{NTT}^{-1}(\hat{w})$
Decomp_w
w
Pack_{w_1}
w_0
makeHint
$\sigma = (\hat{h}, z, c')$
Pack_z
Gen_c
$\hat{c}$
z
s_2'
$\mathrm{NTT}(c)$
$\hat{z} = \hat{y} + \hat{c} \cdot \hat{s}_1$
$\hat{c} \cdot \hat{s}_2$
$\hat{c} \cdot \hat{t}_0$
$z = \mathrm{NTT}^{-1}(\hat{z})$
$s_2' = \mathrm{NTT}^{-1}(\hat{c} \cdot \hat{s}_2)$
$t_0' = \mathrm{NTT}^{-1}(\hat{c} \cdot \hat{t}_0)$
$w_0 -= s_2'$
$w_0 += t_0'$
$\hat{s}_1, \hat{s}_2, \hat{t}_0, \hat{y}$
t_0'
cycles
4000
6000
8000
10000

Figure 2.3 cycle breakdown analysis

2.7: Research Evolution and Methodological Advancement in Dilithium Hardware

This table represents a chronological comparison of major CRYSTALS-Dilithium implementations, highlighting how each work addresses limitations in prior designs. It clearly illustrates the progression from algorithm specification to area-optimized, performance-driven, and finally balanced hardware architectures, providing context for positioning the proposed FPGA-based accelerator.

Table 2.4 Research Evolution & Methody

Reference	Primary Contribution	Limitation Addressed	Design Focus	Key Technique
Ducas *et al.* [1]	Formal specification of Dilithium	N/A	Algorithmic baseline	Mathematical construction of lattice-based signature scheme
Ricci *et al.* [2]	Area-efficient hardware design	Lack of hardware focus in [1]	Area optimization	Time-multiplexed computation

				units
Land *et al.* [3]	High-performance implementation	Increased latency in [2]	Throughput optimization	Parallel NTT and Keccak cores
Zhao *et al.* [4]	Balanced ASIC architecture	Area overhead in [3] and latency in [2]	Area–performance trade-off	Unified NTT/INTT butterfly structure
Beckwith *et al.* [5], [6]	Comprehensive design analysis	Limitations across prior implementations	Methodological optimization	Bottleneck-aware architectural refinement

2.7 Synthesis of Research Progress and Correlations

The chronological analysis reveals important correlations between the six papers. Each successive work directly addresses limitations revealed by its predecessors: Ricci et al. [2] implements the algorithm but reveals latency problems; Land et al. [3] addresses latency but demonstrates area costs; Zhao et al. [4] balances both but loses reconfigurability; Beckwith et al. [5,6] provide systematic methodology but highlight security gaps. This progression demonstrates increasing sophistication in addressing complex implementation trade-offs.

The research also shows evolving understanding of performance bottlenecks. Early implementations [2,3] focused primarily on NTT optimization, while systematic analysis [5,6] revealed that the Keccak function (the core of the SHA-3 hash standard) is the dominant bottleneck ,suggesting different architectural prioritization. The papers collectively establish the boundaries of the design space, providing valuable reference points for evaluating new implementations.

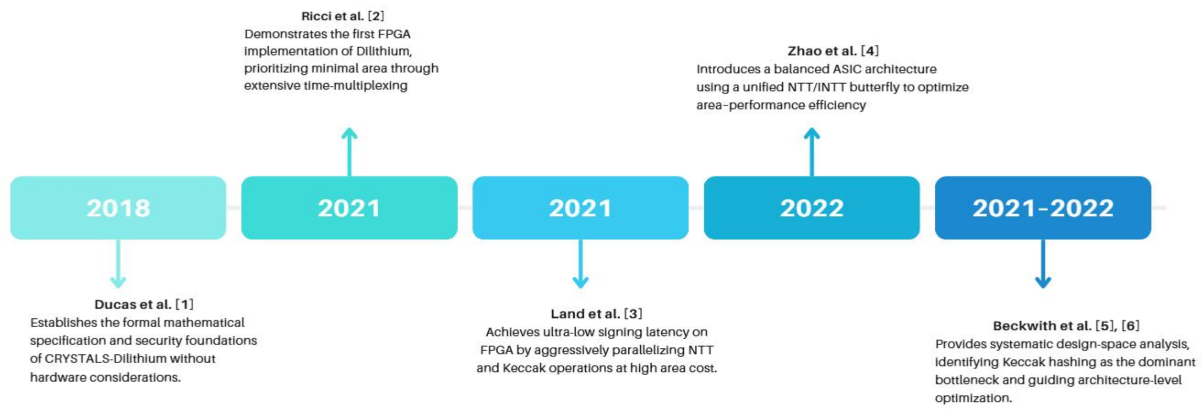

Ricci et al. [2]
Demonstrates the first FPGA implementation of Dilithium, prioritizing minimal area through extensive time-multiplexing
Zhao et al. [4]
Introduces a balanced ASIC architecture using a unified NTT/INTT butterfly to optimize area-performance efficiency
2018
2021
2021
2022
2021-2022
Ducas et al. [1]
Establishes the formal mathematical specification and security foundations of CRYSTALS-Dilithium without hardware considerations.
Land et al. [3]
Achieves ultra-low signing latency on FPGA by aggressively parallelizing NTT and Keccak operations at high area cost.
Beckwith et al. [5], [6]
Provides systematic design-space analysis, identifying Keccak hashing as the dominant bottleneck and guiding architecture-level optimization.

Figure 2.1 Cycle breakdown for Keccak

2.8 Identified Research Gaps and Future Directions

Despite significant progress, several critical research gaps remain unaddressed. First, precision-accuracy trade-offs are inadequately explored, with all implementations assuming bit-accurate reproduction without examining potential benefits of reduced precision in non-critical operations. Second, dynamic adaptation to different security levels is limited, with most designs optimized for specific parameter sets rather than providing efficient reconfiguration. Third, standardized evaluation metrics are absent, preventing fair comparison across implementation platforms. Fourth, the universal neglect of side-channel protections creates fundamental security vulnerabilities for real-world deployment.

2.9 Our Preliminary FPGA Implementation

As a preliminary step, we implemented the Schnorr algorithm on FPGA because it shares the core modules of key generation, signature generation, and verification used in CRYSTALS-Dilithium. While Schnorr relies on discrete logarithm hardness, Dilithium is based on lattice mathematics. This approach allows us to validate measurement methods and architectural decisions early, before tackling the higher computational complexity of lattice-based schemes. It provides a staged, practical pathway without deviating from the main research objective.

2.10 Conclusion

Research on hardware acceleration of CRYSTALS-Dilithium has progressed from algorithm specification to a range of FPGA and ASIC implementations exploring different points in the area–latency design space. Early FPGA implementations demonstrated functional feasibility under strict resource constraints, while later designs emphasized high throughput through extensive parallelization. More recent work introduced balanced architectural techniques and systematic design-space analysis, improving understanding of dominant computational bottlenecks.

Despite these advances, existing FPGA-based implementations do not simultaneously optimize arithmetic accuracy, space utilization, and signing latency within a single, scalable architecture. Most prior designs prioritize one metric at the expense of others, resulting either in impractically high latency or excessive resource consumption. Furthermore, several implementations are tailored to fixed parameter sets, limiting flexibility and fair comparison across platforms.

The analysis presented in this chapter establishes clear performance and resource baselines for FPGA implementations of CRYSTALS-Dilithium and highlights unresolved trade-offs relevant to reconfigurable hardware. These observations motivate the FPGA accelerator design proposed in subsequent chapters, which aims to improve accuracy preservation while achieving reduced resource usage and competitive latency, and to evaluate its effectiveness through direct benchmarking against established implementations.

Chapter 3 Methodology

This section follows a progressive, bottom-up methodology to design and implement post-quantum cryptographic primitives on FPGA.The methodology begins with a thorough theoretical foundation, followed by a classical cryptographic hardware implementation. This staged approach enables risk reduction, tool familiarity, and architectural validation as we move forward towards building a fully verifiable and optimised Dilithium implementation.

3.1 Literature Review and Theoretical Foundation

3.1.1 Study of Classical Cryptography

The first phase involved studying classical public-key cryptographic algorithms, particularly those based on the Discrete Logarithm Problem (DLP) and Elliptic Curve Cryptography (ECC). This included:

- ✓ Digital signature schemes (Schnorr, ECDSA)
- ✓ Hardness assumptions of DLP and ECC
- ✓ Comparison between discrete logarithm–based and elliptic curve–based constructions

The objective of this phase was to understand how mathematical hardness assumptions translate into secure cryptographic protocols and how these protocols can be mapped to hardware architectures.

3.1.2 Introduction to Post-Quantum Cryptography

Following classical cryptography, an extensive study of post-quantum cryptography was conducted, focusing on algorithms resistant to quantum attacks. Special focus was given to lattice-based schemes, particularly CRYSTALS-Dilithium and CRYSTALS-KYBER, due to their standardization status and suitability for hardware implementation. Literature review was conducted to understand recent progress in the algorithms and articles were written by members, two of which are published on Medium.[14][15]

3.1.3 Mathematical Foundations

To have an intuitive understanding of post-quantum schemes, the following mathematical concepts were studied:

- ✓ Modular arithmetic and finite fields
- ✓ Discrete logarithms and exponentiation
- ✓ Linear algebra and polynomial arithmetic
- ✓ Lattice theory fundamentals

This phase ensured a strong mathematical understanding necessary for correct and efficient hardware design.

3.2 Design Strategy and Rationale

3.2.1 Motivation for Incremental Implementation

Direct implementation of Dilithium on FPGA involves complex polynomial arithmetic, large parameter sizes, and memory-intensive operations. To mitigate these challenges, an incremental design strategy was adopted.

As an initial step, a classical Schnorr digital signature scheme based on the discrete logarithmic exponentiation problem was selected and implemented. This approach provided practical exposure to cryptographic hardware design along with familiarity with FPGA tools and workflows. The validation of control logic, datapath design, and memory usage led to a smaller yet meaningful cryptographic system.

3.2.2 Choice of Schnorr's Algorithm for the MVP

A 16-bit Schnorr's signature algorithm was chosen because:

- ✓ It is mathematically simple yet cryptographically significant.
- ✓ It forms the conceptual foundation for many modern signature schemes.
- ✓ It involves key cryptographic operations such as modular exponentiation, hashing, and verification.

Its structure aligns with the basis of Dilithium's signing and verification flow.

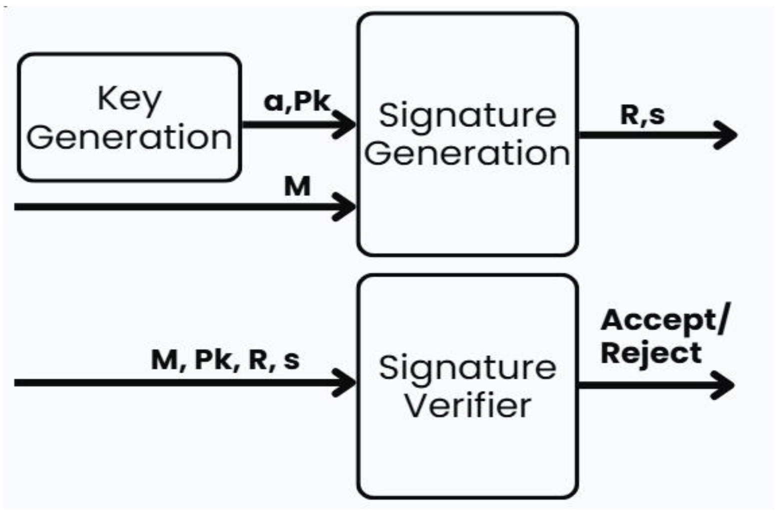


Figure 3.1 Digital Signature Scheme

3.3 Building the MVP

The FPGA implementation of a 16-bit Schnorr signature scheme utilising discrete logarithmic hardness is essentially for learning purposes. Its deployment and real-world usage is impractical as better variants such as ECDSA exist. Therefore, no official work in support of such an implementation existed before our design.

Hence, the team worked on two separate designs to understand the trade-offs; the serial implementation and the parallel implementation. For this phase, Artix 7 100t csg324 FPGA was used and the EDA tool selected was Vivado Design Suite 2018. But before moving towards the FPGA implementation, a Python Reference Model[9] was created.

3.3.1 The Reference Model

The model included a clean separation of Key Generation, Signing, and Verification modules ensuring cryptographically secure randomness, and SHA-256–based challenge computation.The minimal, auditable codebase also included unit tests to verify results against known test vectors.[16]

3.3.2 Algorithm Parameters

The Schnorr Algorithm requires the definition of some parameters to facilitate computation given below:

- ✓ **p:** Large prime modulus
- ✓ **q:** Prime order of the subgroup p
- ✓ **g:** Generator of the subgroup p

They satisfy the Schnorr requirement:

$g^q \bmod p = 1$ .... (3.1)

3.3.3 Key Generation

1. Choose private key $x \in [1, q-1]$
2. Compute public key:

$P = G^x \bmod p$ .... (3.2)

3.3.4 Signature Generation

Given message *m* and private key *x*:

1. Generate random nonce *r*
2. Compute commitment:

$R = g^r \bmod p$ .... (3.3)

3. Compute challenge:

$c = H(m \| R)$ .... (3.4)

4. Compute response:

$s = (r + c \cdot x) \bmod q$ .... (3.5)

Signature: (R, s)

3.3.5 Signature Verification

Given (R, s) and public key Y:

1. Recompute:

$c = H(m \| R)$ .... (3.6)

2. Verify:

$g^s \bmod p == R.P^c \bmod p$ .... (3.7)

If the equation holds, the signature is valid.

3.3.6 The Primitive Functions

- ✓ ***SHA256 Hash Function*:** The SHA256 hash function code was acquired from a GitHub repository[15] as the official Xilinx IP and other custom IPs were paid. Although archived, the code from the repository after being rigorously tested against known test vectors was fully verified.
- ✓ ***The G Function*:** The G function in Discrete Logarithmic hardness is equivalent to modular exponentiation. Due to high computational complexity of multiplication operation, the code was optimised to support binary modular exponentiation significantly reducing n operations to log2(n) operations where n is the power of to some base.[15]

3.3.7 The Serial Implementation

This implementation was more space optimised but lacked throughput and modularity. [10].The selection of one of the three modes, namely, Key Generation, Signature Generation, and Signature Verification, was determined by an input mode which in turn directed the FSM to the relevant flow.

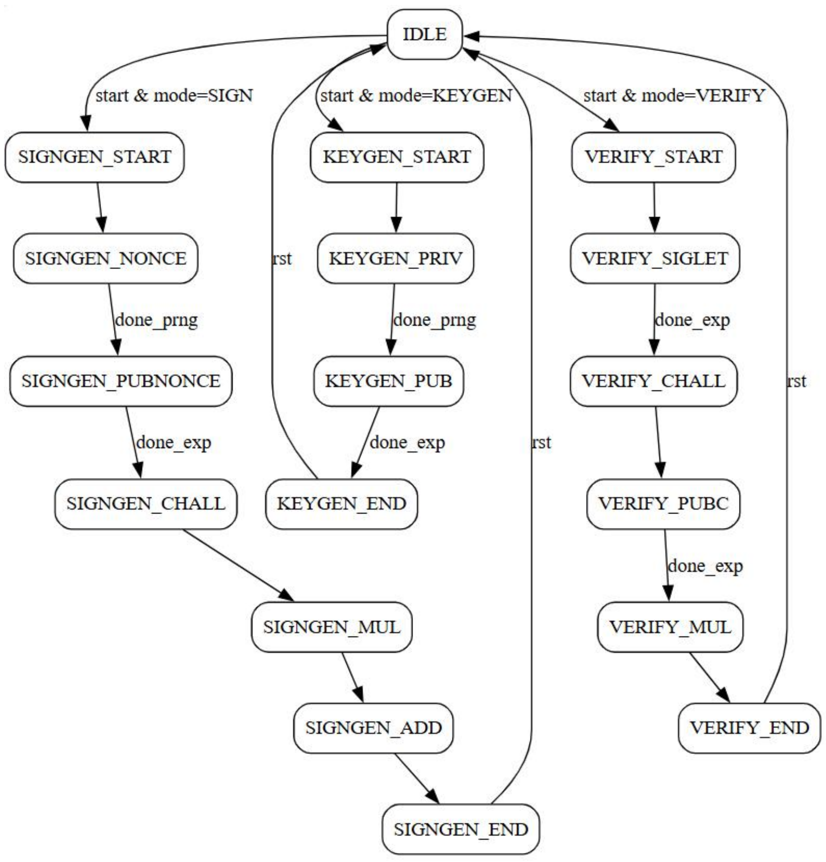


Figure 3.2 Schnorr's Algorithm FSM

Single instances of hash function and G function were initialized leading to lower area and power utilization. The schematic of the design is given below.

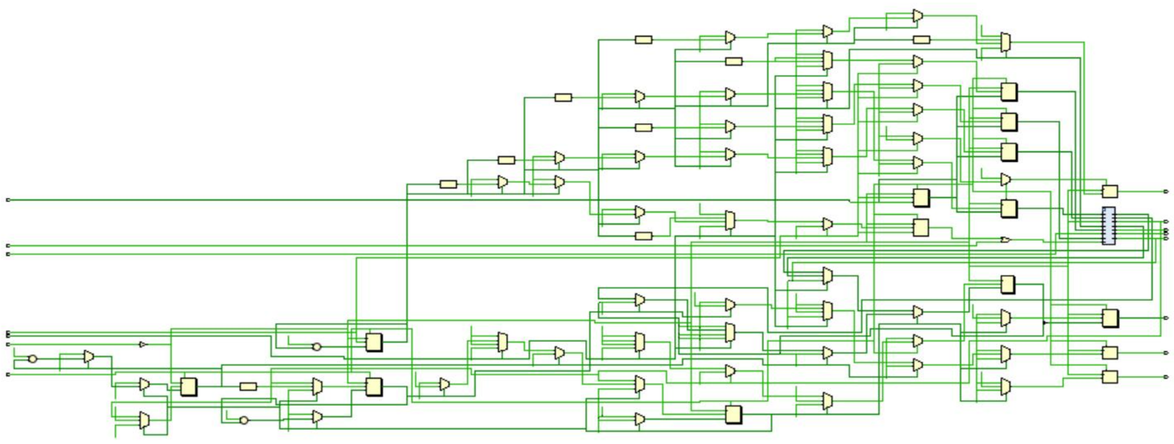


Figure 3.3 Schnorr's Serial Architecture Schematic

3.3.8 The Parallel Implementation

Building upon the lessons learned from the serial implementation [10], a parallel architecture was developed to maximize throughput and enable simultaneous cryptographic operations. This design represents a complete, modular hardware realization of the Schnorr digital signature scheme using 16-bit parameters for demonstration.

3.3.8 (a) System Architecture Overview

The system employs a hierarchical, modular architecture comprising nine interconnected Verilog modules that collectively implement the complete Schnorr signature workflow. The design separates cryptographic operations into distinct functional units while promoting code reuse through shared arithmetic modules. At the core of the system lies a top-level integration module that controls three primary operations: key pair generation, signature creation, and signature verification.

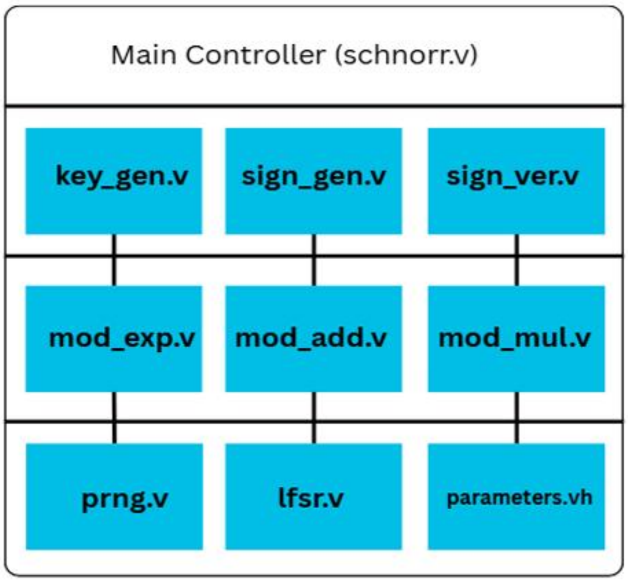


Figure 3.4 Block Based Architecture

The architecture follows a bottom-up approach where fundamental arithmetic modules form the computational foundation for higher-level cryptographic functions. All modules operate within a synchronous design paradigm, ensuring predictable timing behavior and reliable state machine operation.

3.3.8 (a) Parameterized System Configuration

Central to the system's flexibility is a global parameter file that defines all cryptographic constants. This includes setting the bit-width to 16 bits for development purposes [11], selecting a prime modulus of 65521, defining a generator value of 373, and calculating the subgroup order as p-1. This centralized parameterization ensures mathematical consistency across all operations while simplifying future scaling to cryptographically secure bit-widths.

3.3.8 (b) Core Computational Modules

✓ ***Pseudorandom Number Generation Subsystem**:*

The system incorporates a 32-bit Linear Feedback Shift Register (LFSR) employing a maximal-length feedback polynomial ($x^{32} + x^{21} + x + 1$) to generate entropy [12]. This LFSR core is wrapped within a controller module that collects multiple outputs to produce wider random values. The PRNG subsystem provides random numbers for private key generation and nonce creation, utilizing deterministic seeding during testing for reproducible results.

✓ ***Modular Arithmetic Units**:*

Three specialized arithmetic modules form the computational backbone:

- **Modular Addition**: A combinational circuit implementing (a + b) mod r with careful bit-width management to prevent overflow.
- **Modular Multiplication**: Performs (a × b) mod r using native multiplication operators followed by modular reduction. This module has been identified for optimization to reduce area utilization.
- **Modular Exponentiation**: Implements the square-and-multiply algorithm through an 8-state finite state machine that reuses the modular multiplier [13]. This sequential module requires multiple clock cycles and is employed across all cryptographic operations.

3.3.8 (c) Cryptographic Operation Modules

Key Generation Module

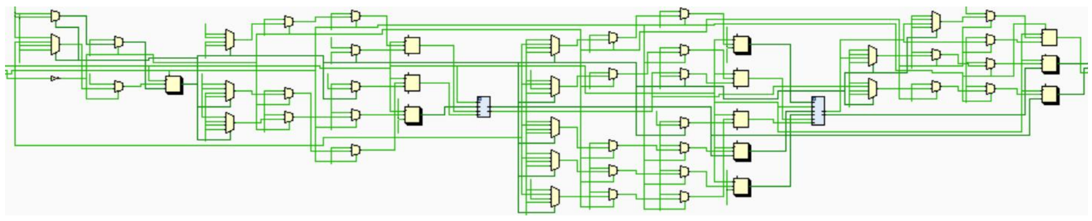


Figure 3.5 Key Generation Module Schematic

This component generates Schnorr key pairs through a 3-state finite state machine. It produces a private key as a random 16-bit integer and computes the corresponding public key using modular exponentiation (y = g^x mod p). The module includes proper handshaking signals to indicate completion.

Signature Generation Module

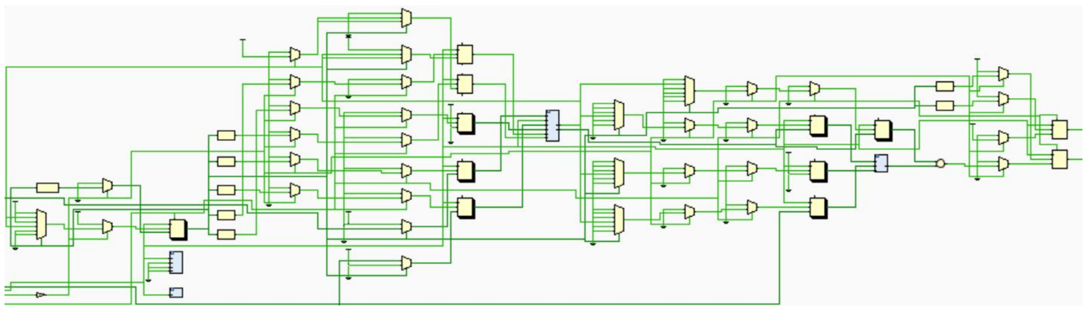


Figure 3.6 Signature Generation Module Schematic

Implementing the signing equation s = (k + c·x) mod q, this module follows a 6-state control flow. It generates a random nonce k, computes the public commitment R = g^k mod p, and produces the final signature. Currently, the cryptographic hash function for challenge computation is simulated with a hardcoded value, representing a known limitation for future enhancement.

Signature Verification Module

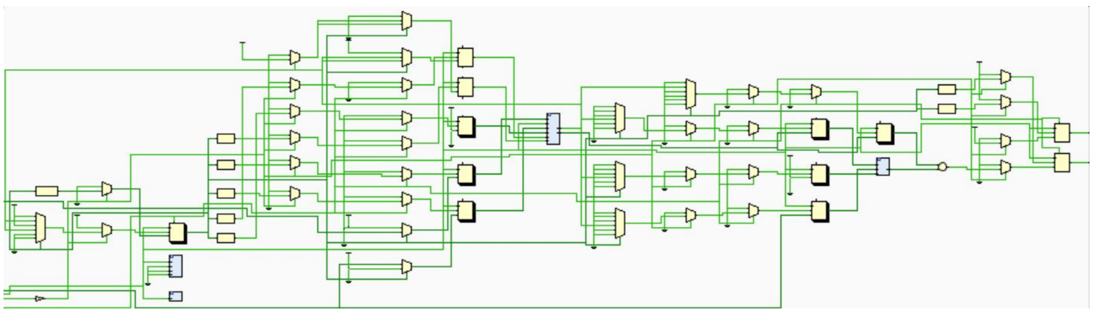


Figure 3.7 Signature Verification Module Schematic

This component validates signatures by checking the equivalence g^s mod p ≡ R × P^c mod p [8]. A 5-state finite state machine computes both sides of the verification equation using modular exponentiation and multiplication, then compares the results to determine validity.

3.3.8 (d) Top-Level Integration and Data Flow

The system integrates all modules through a top-level controller that enables independent operation of key generation, signing, and verification. This parallel architecture allows multiple operations to proceed concurrently when resources permit, significantly improving overall throughput compared to the serial implementation [10].

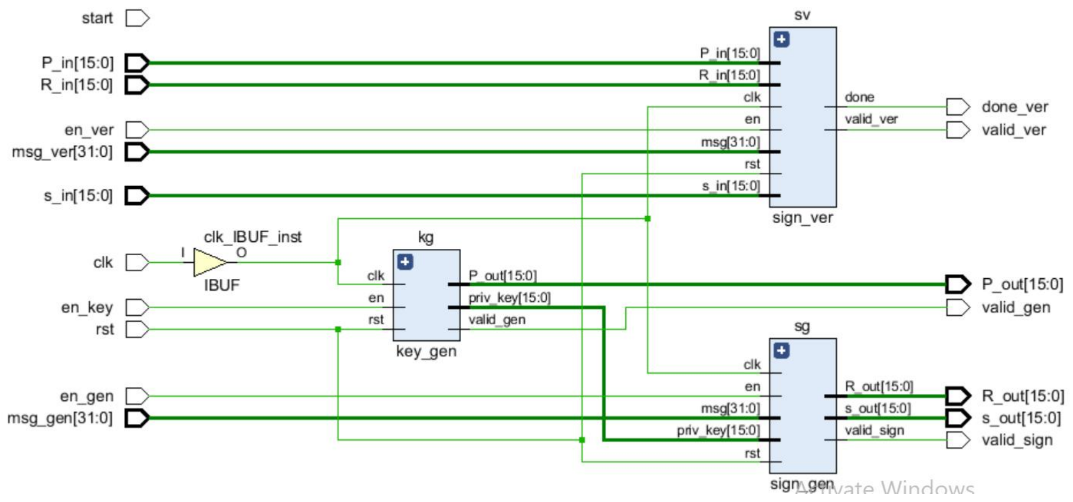


Figure 3.7 Integrated Data Path of Schnorr (Parallel) Schematic

Chapter 4 Results & Analysis

This chapter presents the experimental and implementation results of the proposed Schnorr digital signature system. The results are divided into two major parts. First, the verification of the reference (software) model is discussed, which serves as the golden model for validating the hardware implementation. Second, the results of the Schnorr datapath hardware architectures are presented, including functional correctness, simulation outcomes, latency analysis, and resource utilization.

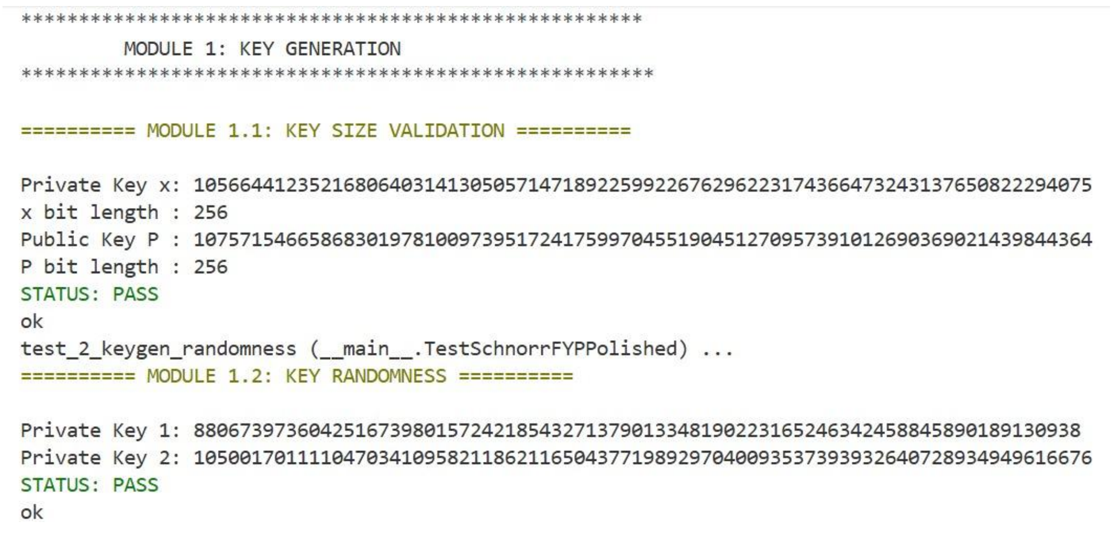
4.1 Results of Reference Model

The reference model was rigorously tested using multiple test cases to ensure functional correctness and reliability. This validated reference model is later used as the golden reference for RTL verification of the hardware datapath.[16]
The reference implementation covers all major functional blocks of the Schnorr digital signature scheme, namely key generation, signature generation, and signature verification.

The successful execution of all test cases confirms that the reference model produces correct and consistent outputs, making it suitable for comparison with the RTL implementation.

4.1.1 Key Generation

The key generation module of the reference model was tested with various random seeds and input parameters. The generated public and private keys were verified for correctness according to the Schnorr algorithm specifications.
All test cases produced valid and reproducible key pairs.[16]

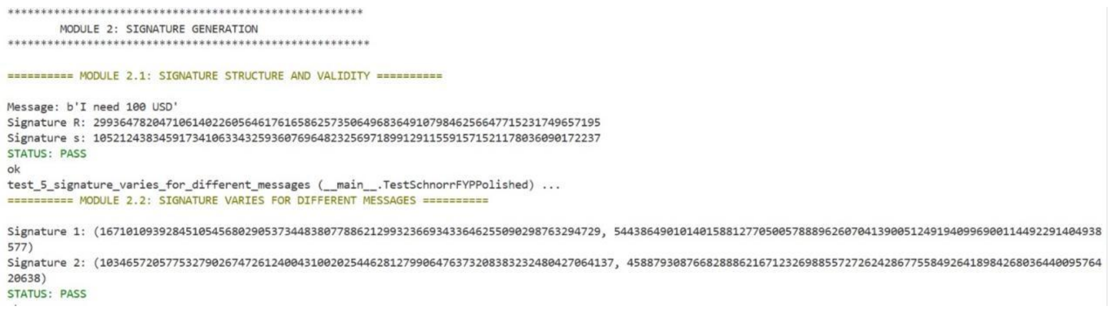


```
*******************************************************
         MODULE 1: KEY GENERATION
********************************************************

========== MODULE 1.1: KEY SIZE VALIDATION ==========

Private Key x: 105664412352168064031413050571471892259922676296223174366473243137650822294075
x bit length : 256
Public Key P : 107571546658683019781009739517241759970455190451270957391012690369021439844364
P bit length : 256
STATUS: PASS
ok
test_2_keygen_randomness (__main__.TestSchnorrFYPPolished) ...
========== MODULE 1.2: KEY RANDOMNESS ==========

Private Key 1: 88067397360425167398015724218543271379013348190223165246342458845890189130938
Private Key 2: 105001701111047034109582118621165043771989297040093537393932640728934949616676
STATUS: PASS
ok
```

Figure 4.1 Key Generation Output

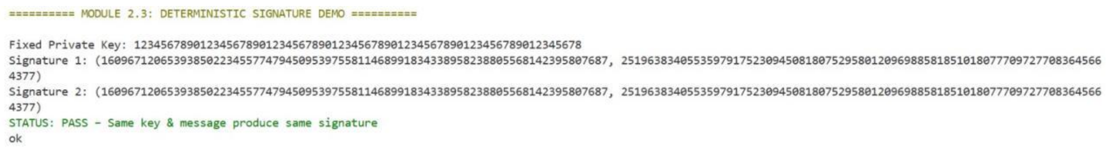
4.1.2 Signature Generation

The signature generation module was evaluated using different messages and corresponding private keys. For each test case, the generated signature components matched the expected values produced by the reference implementation.



```
*******************************************************
        MODULE 2: SIGNATURE GENERATION
********************************************************

========== MODULE 2.1: SIGNATURE STRUCTURE AND VALIDITY ==========

Message: b'I need 100 USD'
Signature R: 29936478204710614022605646176165862573506496836491079846256647715231749657195
Signature s: 105212438345917341063343259360769648232569718991291155915715211780360901722237
STATUS: PASS
ok
test_5_signature_varies_for_different_messages (__main__.TestSchnorrFYPPolished) ...
========== MODULE 2.2: SIGNATURE VARIES FOR DIFFERENT MESSAGES ==========

Signature 1: (1671010939284510545680290537344838077886212993236693433646255090298763294729, 54438649010140158812770500578889626070413900512491940996900114492291404938
577)
Signature 2: (103465720577532790267472612400431002025446281279906476373208383232480427064137, 458879308766828886216712326988557272624286775584926418984268036440095764
20638)
STATUS: PASS
```

Figure 4.2 (a) Signature Generation Output

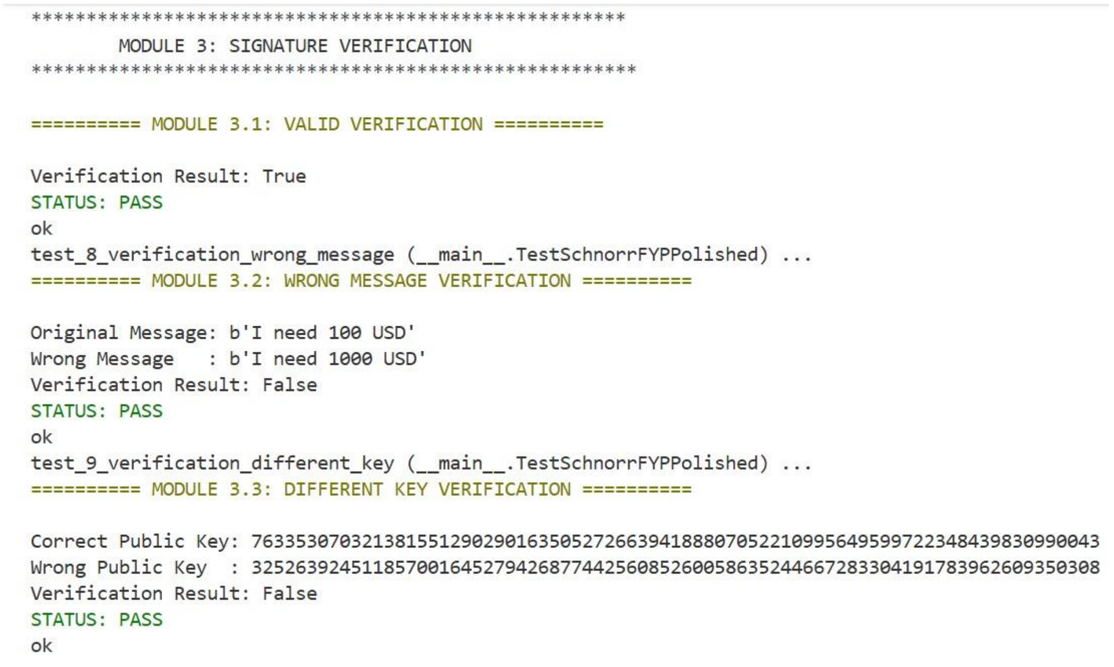


```
========== MODULE 2.3: DETERMINISTIC SIGNATURE DEMO ==========

Fixed Private Key: 12345678901234567890123456789012345678901234567890123456789012345678
Signature 1: (16096712065393850223455774794509539755811468991834338958238805568142395807687, 2519638340553597917523094508180752958012096988581851018077709727708364566
4377)
Signature 2: (16096712065393850223455774794509539755811468991834338958238805568142395807687, 2519638340553597917523094508180752958012096988581851018077709727708364566
4377)
STATUS: PASS - Same key & message produce same signature
ok
```

Figure 4.2 (b) Signature Generation Output

4.1.3 Signature Verification

The signature verification module was tested using valid and invalid signatures. Valid signatures were successfully verified, while altered or incorrect signatures were correctly rejected.
This confirms the correctness and robustness of the reference model verification process.



```
*****************************************************
        MODULE 3: SIGNATURE VERIFICATION
******************************************************

========== MODULE 3.1: VALID VERIFICATION ==========

Verification Result: True
STATUS: PASS
ok
test_8_verification_wrong_message (__main__.TestSchnorrFYPPolished) ...
========== MODULE 3.2: WRONG MESSAGE VERIFICATION ==========

Original Message: b'I need 100 USD'
Wrong Message   : b'I need 1000 USD'
Verification Result: False
STATUS: PASS
ok
test_9_verification_different_key (__main__.TestSchnorrFYPPolished) ...
========== MODULE 3.3: DIFFERENT KEY VERIFICATION ==========

Correct Public Key: 76335307032138155129029016350527266394188807052210995649599722348439830990043
Wrong Public Key  : 32526392451185700164527942687744256085260058635244667283304191783962609350308
Verification Result: False
STATUS: PASS
ok
```

Figure 4.3 Signature Verification Output

4.2 Results of Schnorr Datapath

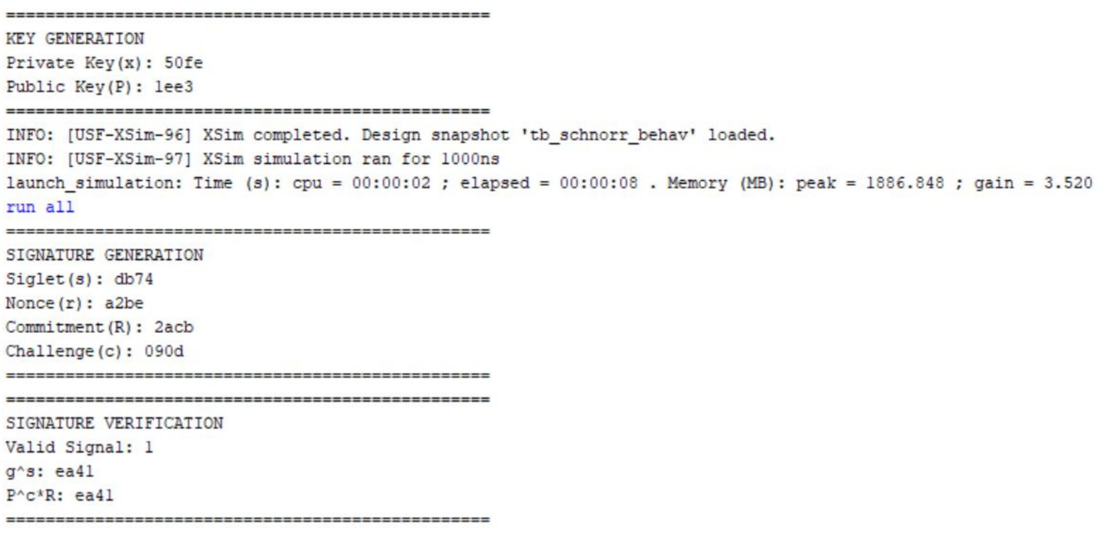
The Schnorr datapath was implemented using two different hardware architectures:

- ✓ Serial Architecture
- ✓ Parallel (Block-Based) Architecture

Both architectures were designed to perform Schnorr signature operations and were verified against the reference model. Although the architectural approaches differ, both implementations produce identical functional outputs.[16]

4.2.1 Testbench Output (Accuracy)

A comprehensive RTL testbench was developed to validate the functionality of both datapath architectures. The testbench compares the RTL outputs directly with the reference model outputs. The outputs of both the serial and parallel datapath architectures match 100% with the reference model, confirming functional correctness and accuracy.

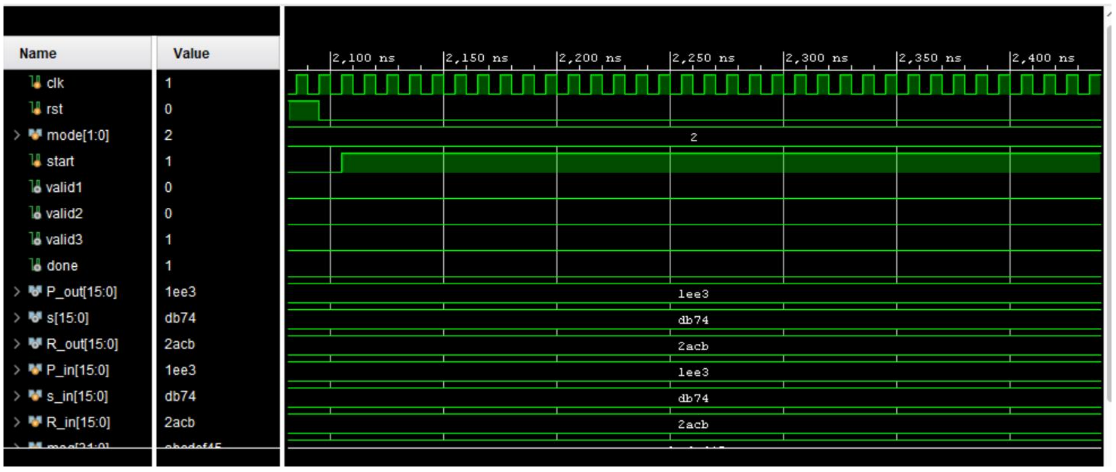


```
==================================================
KEY GENERATION
Private Key(x): 50fe
Public Key(P): 1ee3
==================================================
INFO: [USF-XSim-96] XSim completed. Design snapshot 'tb_schnorr_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:02 ; elapsed = 00:00:08 . Memory (MB): peak = 1886.848 ; gain = 3.520
run all
==================================================
SIGNATURE GENERATION
Siglet(s): db74
Nonce(r): a2be
Commitment(R): 2acb
Challenge(c): 090d
==================================================
==================================================
SIGNATURE VERIFICATION
Valid Signal: 1
g^s: ea41
P^c*R: ea41
==================================================
```

Figure 4.4 Testbench Output

4.2.2 Simulation Waveform

Simulation waveforms were analyzed to verify correct signal transitions, control sequencing, and data flow across all modules. The waveforms confirm correct timing behavior and functional operation of both datapath designs.

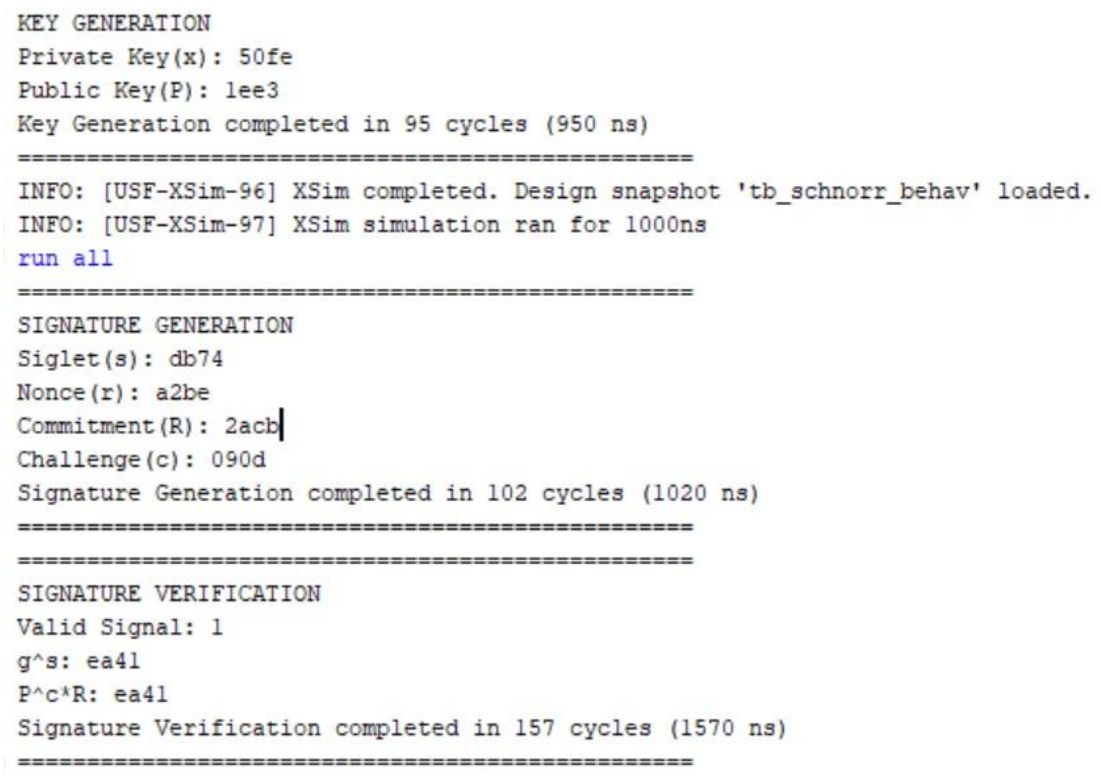


Name	Value
clk	1
rst	0
mode[1:0]	2
start	1
valid1	0
valid2	0
valid3	1
done	1
P_out[15:0]	1ee3
s[15:0]	db74
R_out[15:0]	2acb
P_in[15:0]	1ee3
s_in[15:0]	db74
R_in[15:0]	2acb

Figure 4.5 Simulation Waveforms

4.2.3 Latency Analysis

Latency measurements were performed for both the serial and parallel architectures. Despite differences in internal processing styles, both designs exhibit identical latency, as they follow the same algorithmic sequence of operations.



```
KEY GENERATION
Private Key(x): 50fe
Public Key(P): 1ee3
Key Generation completed in 95 cycles (950 ns)
==================================================
INFO: [USF-XSim-96] XSim completed. Design snapshot 'tb_schnorr_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
run all
==================================================
SIGNATURE GENERATION
Siglet(s): db74
Nonce(r): a2be
Commitment(R): 2acb
Challenge(c): 090d
Signature Generation completed in 102 cycles (1020 ns)
==================================================
==================================================
SIGNATURE VERIFICATION
Valid Signal: 1
g^s: ea41
P^c*R: ea41
Signature Verification completed in 157 cycles (1570 ns)
==================================================
```

Figure 4.6 Latencies Of All Modules

4.2.4 Implemented Design – Serial Architecture

The elaborated RTL design of the serial architecture shows a compact hardware structure. Due to resource sharing, this architecture consumes less area, making it suitable for area-constrained environments.

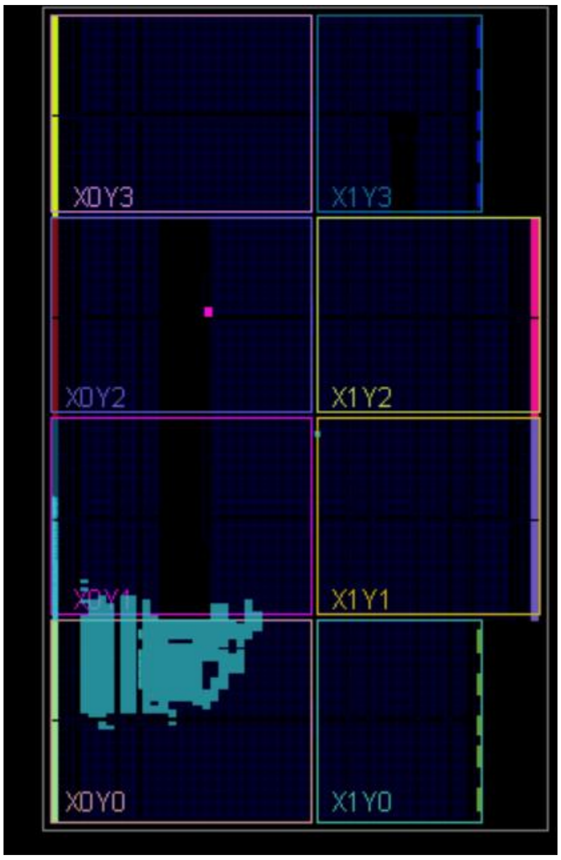


Figure 4.7 Implemented Design (Serial)

4.2.5 Implemented Design – Parallel Architecture

The parallel (block-based) architecture contains multiple functional blocks operating concurrently. This increases hardware utilization but improves throughput by enabling faster execution of operations.

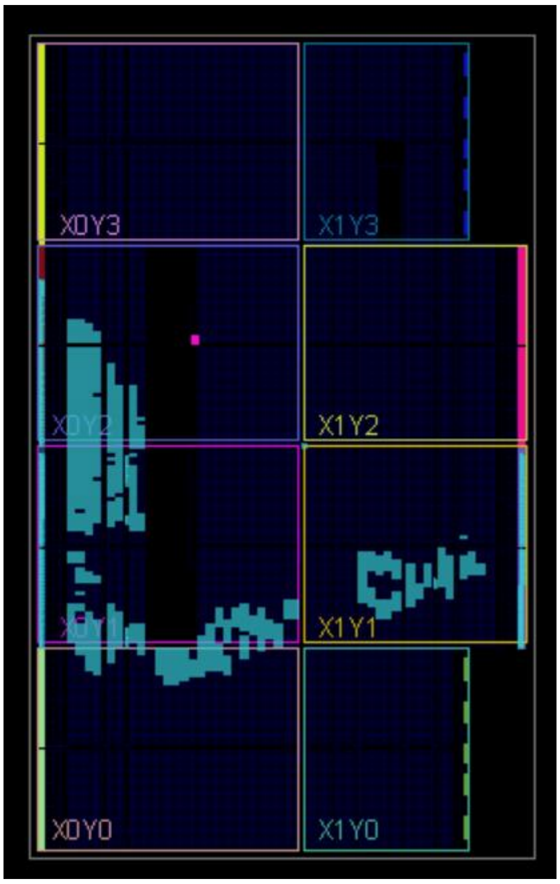
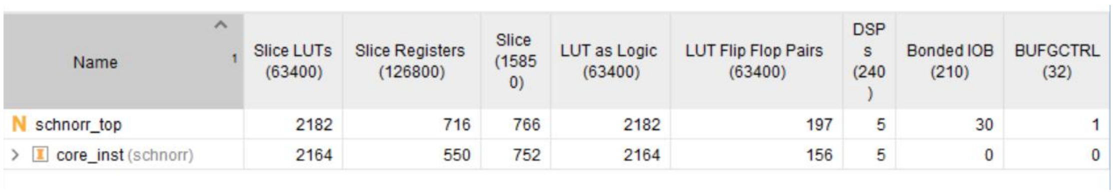


Figure 4.8 Implemented Design (Parallel)

4.2.6 Resource Utilization Summary – Serial Architecture

The device utilization report indicates that the serial architecture uses fewer logic resources, making it more area-efficient.

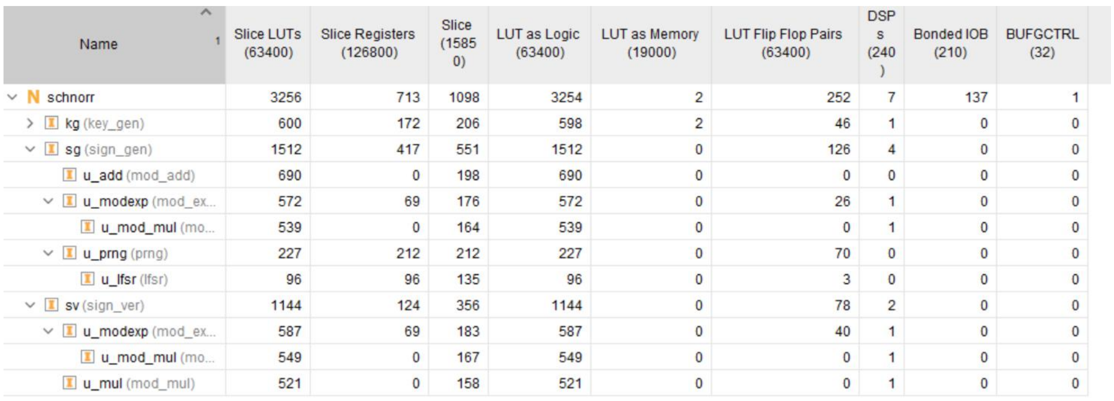


Name	Slice LUTs (63400)	Slice Registers (126800)	Slice (15850)	LUT as Logic (63400)	LUT Flip Flop Pairs (63400)	DSPs (240)	Bonded IOB (210)	BUFGCTRL (32)
schnorr_top	2182	716	766	2182	197	5	30	1
core_inst (schnorr)	2164	550	752	2164	156	5	0	0

Figure 4.9 Resource Utilization Summary (Serial)

4.2.7 Device Utilization Summary – Parallel Architecture

The parallel architecture shows higher utilization of LUTs, registers, and other hardware resources. However, this increased resource usage directly contributes to higher throughput.



Name	Slice LUTs (63400)	Slice Registers (126800)	Slice (15850)	LUT as Logic (63400)	LUT as Memory (19000)	LUT Flip Flop Pairs (63400)	DSPs (240)	Bonded IOB (210)	BUFGCTRL (32)
schnorr	3256	713	1098	3254	2	252	7	137	1
kg (key_gen)	600	172	206	598	2	46	1	0	0
sg (sign_gen)	1512	417	551	1512	0	126	4	0	0
u_add (mod_add)	690	0	198	690	0	0	0	0	0
u_modexp (mod_ex...	572	69	176	572	0	26	1	0	0
u_mod_mul (mo...	539	0	164	539	0	0	1	0	0
u_prng (prng)	227	212	212	227	0	70	0	0	0
u_lfsr (lfsr)	96	96	135	96	0	3	0	0	0
sv (sign_ver)	1144	124	356	1144	0	78	2	0	0
u_modexp (mod_ex...	587	69	183	587	0	40	1	0	0
u_mod_mul (mo...	549	0	167	549	0	0	1	0	0
u_mul (mod_mul)	521	0	158	521	0	0	1	0	0

Figure 4.10 Resource Utilization Summary (Parallel)

4.3 Comparative Analysis of Serial and Parallel Architectures

Table 4.1 presents a comparative summary of both data path architectures in terms of accuracy, latency, throughput, and area utilization.

Table 4.1 Comparitive Analysis

Parameter	Serial Architecture	Parallel Architecture
Accuracy	100% (Matches reference model)	100% (Matches reference model)
Latency	Same	Same
Throughput	Lower	Higher
Area Utilization	Low(Area-Optimized)	High
Resource Sharing	High	Low
Performance Trade-off	Area efficient, slower	Faster, Area intensive

4.4 Summary

The results demonstrate that both serial and parallel Schnorr datapath implementations are functionally correct and fully validated against the reference model. The serial architecture offers better area efficiency, while the parallel architecture achieves higher throughput at the cost of increased resource usage.
This trade-off allows system designers to select an appropriate architecture based on application requirements and hardware constraints.

References

[1] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2018, no. 1, pp. 238–268, 2018. Available:https://eprint.iacr.org/2017/633

[2] S. Ricci, L. Malina, P. Jedlička, D. Smékal, J. Hajny, and P. Cíbik, "Implementing CRYSTALS-Dilithium Signature Scheme on FPGAs," *Cryptology ePrint Archive*, Report 2021/108, 2021. Available:https://eprint.iacr.org/2021/108

[3] G. Land, P. Sasdrich, and T. Güneysu, "A Hard Crystal – Implementing Dilithium on Reconfigurable Hardware," *Cryptology ePrint Archive*, Report 2021/355, 2021. Available:https://eprint.iacr.org/2021/355

[4] C. Zhao, N. Zhang, H. Wang, B. Yang, W. Zhu, Z. Li, M. Zhu, S. Yin, S. Wei, and L. Liu, "A Compact and High-Performance Hardware Architecture for CRYSTALS-Dilithium," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2022, no. 1, pp. 1–22, 2022.Available: https://eprint.iacr.org/2021/1451

[5] L. Beckwith, D. T. Nguyen, and K. Gaj, "*High-Performance Hardware Implementation of CRYSTALS-Dilithium,*" *Cryptology ePrint Archive*, Report 2021/1451, 2021.Available:https://eprint.iacr.org/2021/1451

[6] L. Beckwith, D. T. Nguyen, and K. Gaj, "*High-Performance Hardware Implementation of Lattice-Based Digital Signatures (Extended Version),*" *Cryptology ePrint Archive*, Report 2021/1451, 2022. Available: https://eprint.iacr.org/2021/1451

[7] A. Cano Aguilera, C. Rubio García, D. Lawo, J. L. Imaná, I. Tafur Monroy, and J. J. Vegas Olmos, "*In-line rate encrypted links using pre-shared post-quantum keys and DPUs,*" Scientific Reports, vol. 14, no. 21227, 2024. Available:https://research.tue.nl/en/publications/in-line-rate-encrypted-links-using-pre-shared-post-quantum-keys-a/

[8] C. Schnorr, *"Efficient identification and signatures for smart cards,"* in Advances in Cryptology CRYPTO' 89 Proceedings, 1990, pp. 239-252. [Online]. Available: https://link.springer.com/chapter/10.1007/0-387-34805-0_22

[9] National Institute of Standards and Technology, *"Post-Quantum Cryptography* Standardization," NIST, 2023.Available:https://csrc.nist.gov/Projects/postquantum-cryptography

[10] M. Khan et al., "*Efficient Hardware Implementation of Cryptographic Primitives for Embedded Systems,*" IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 67, no. 5, pp. 1565- 1578, 2020.Available: https://ieeexplore.ieee.org/document/8966147

[11] J. M. Rabaey, A. P. Chandrakasan, and B. Nikolić**,** "*Digital Integrated Circuits: A Design Perspective*", 2nd ed. Pearson Education, 2003. Available:https://web.stanford.edu/class/ee371/handouts/books/Rabaey03_book.pdf

[12] P. L'Ecuyer, "*Tables of Linear Congruential Generators of Different Sizes and Good Lattice Structure,*" *Mathematics of Computation*, vol. 68, no. 225, pp. 249-260, 1999. Available: https://www.ams.org/journals/mcom/1999-68-225/S0025-5718-99-00996-5/

[13] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996. Available: https://cacr.uwaterloo.ca/hac/

[14] T. Aftab, *"How our FYDP project 'Hardware Acceleration Of Crystals, Dilithium for Post Quantum Cryptography' benefit society, industry, economy, or the environment?"*, Medium, Nov. 30, 2025. [Online]. Available: https://medium.com/@aftabtooba72/how-our-fydp-project-hardware-acceleration-of-crystals-dilithium-for-post-quantum-crptography-55cdd1acb191

[15] A. Abdul Qadir, *"An Overview of the CRYSTALS-Dilithium Algorithm"*, Medium, Nov. 1, 2025.Available: https://medium.com/@aqibaabdulqadir/an-overview-of-the-crystals-dilithium-algorithm-655c339b54f7

[16] Mahwish H *" Schnorr Algorithm Python Implementaion Refrence Model"Dec*. 11, 2025. Available: https://github.com/Mehwishh?tab=repositories