ENHANCING THE HARDWARE-BASED ADVANCED ENCRYPTION STANDARD IMPLEMENTATION FOR OPTIMIZED PERFORMANCE IN CYBER-PHYSICAL SYSTEMS

A Thesis

by

USAMA TARIQ

Submitted to the Graduate and Professional School of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2023

Major Subject: Engineering Technology

ABSTRACT

Cyber-Physical Systems play a pivotal role in critical infrastructures but face distinct security challenges. Traditional software-based security measures often disregard to account for the resource limitations that these systems encounter. Field Programmable Gate Arrays offer a solution, enabling hardware-based cryptographic implementations to enhance security. This study delves into the design and analysis of optimized variants of cryptographic encryption implemented on hardware, shedding light on design methodologies and optimization techniques. Our research includes physical verification, rendering the encryption core readily applicable to be deployed in existing systems. Notably, this study goes beyond the existing research landscape, where the focus on specific metrics often neglects the holistic perspective. Our performance analysis, with a particular emphasis on throughput, undergoes rigorous scrutiny and provides a comprehensive view previously unexplored. This work significantly contributes to enhancing security by introducing efficient hardware-based cryptographic solutions and advancing the field with a comprehensive performance analysis.

DEDICATION

To my family, who have stood by me through countless challenges and continue to inspire me with their unwavering support and enduring patience.

ACKNOWLEDGEMENTS

I extend my heartfelt gratitude to my committee chair, Dr. Crosby, for his constant encouragement and invaluable mentorship during the course of my research. I am also deeply appreciative of my committee members, Dr. Fink and Dr. Davis, for their invaluable guidance and insightful suggestions. Furthermore, I wish to acknowledge Dr. Hounsinou for her vital contribution as a mentor in this journey.

I would like to express my profound thanks to my parents and my colleagues, whose unwavering support and continuous encouragement have been a driving force throughout my educational journey. Their reminders of their pride in my accomplishments have been a constant source of motivation.

Above all, I humbly thank and acknowledge the Lord and Creator for bestowing upon me the perseverance, strength, and the overall ability to reach this significant milestone. Without His divine guidance and grace, I could not have accomplished this feat.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supervised by a thesis committee consisting of Professor Garth V. Crosby [advisor] and Professor Rainer J. Fink of the Department of Engineering Technology and Industrial Distribution and Professor Katherine Davis of the Department of Electrical and Computer Engineering.

All other work conducted for the thesis was completed by the student independently.

Funding Sources

Graduate study was not funded by outside sources.

NOMENCLATURE

CPS	Cyber-Physical Systems
AES	Advanced Encryption Standard
FPGA	Field Programmable Gate Array
RTL	Register-Transfer Level
CLB	Configurable Logic Block
LUT	Look-Up Table
FF	Flip Flop

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1. Background

In the evolving landscape of technological innovation, Cyber-Physical Systems (CPS) have emerged as a transformative paradigm that merges the realms of physical processes and computational power. This convergence holds the potential to revolutionize critical sectors like healthcare, avionics, and automotive industries by fostering efficiency, automation, and real-time decision-making. However, this amalgamation of physical and digital domains also ushers in distinct security challenges that demand careful consideration. Within the realm of CPS, the seamless interaction of physical elements with computational components introduces a series of unique challenges. While CPS are designed to enhance system performance and offer new possibilities, they also bring along vulnerabilities. The convergence of operational technology (OT) and information technology (IT) creates potential gateways for cyberattacks that could compromise critical functions and have far-reaching consequences.

CPS, acting as the cornerstone of automated critical applications, carry the responsibility of safeguarding themselves against malicious incursions. CPS security is characterized by the imperative to prevent unauthorized access to system data and network capabilities. This task is especially daunting due to the spatial-temporal dynamics inherent to the underlying physical environment [1]. Within CPS, processes operate in interconnected loops, orchestrated to function autonomously. However, managing the complexities, unpredictabilities, and constraints of these processes presents a formidable task. The inherent nature of CPS entails an intricate interplay between the cyber and physical realms. The physical domain, encompassing chemical, mechanical, or electrical systems, interfaces with the cyber world, which processes and stores gathered information [2]. The dynamic feedback loop between these realms culminates in the ability of CPS to modify physical processes in predefined ways, enhancing overall performance and usability.

A distinguishing aspect of CPS security is its departure from conventional IT security paradigms [3]. Unlike traditional IT systems, where virus detection updates can be delivered upon system log-in, CPS's heterogeneous and deeply embedded nature hinders such frequent updates and patches for malware detection. This intricacy underscores the urgent need for intensive research to formulate effective security methodologies tailored to CPS's unique challenges.

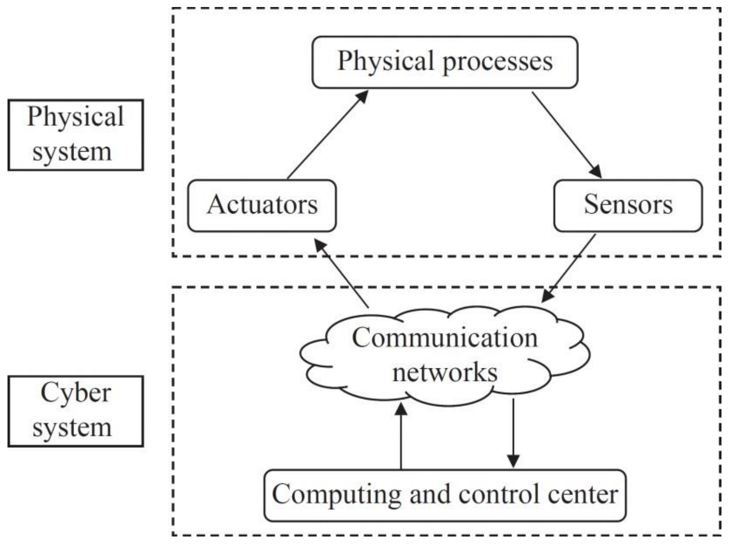


Figure 1 CPS Architecture. Reprinted From [4]

The scope of CPS applications spans from automated healthcare to energy-efficient smart cities, fault-free power generation and distribution, robotics, intelligent automobiles, and safe highways [5]. The integration of independent CPS units into cohesive systems gives rise to new challenges in the form of Cyber-Physical System of Systems (CPSoS) [6]. In the domain of automated healthcare, ensuring security for implanted medical devices within Medical CPS proves to be a fundamental challenge [5]. Instances abound where adversaries could exploit networked medical devices worn by individuals, accentuating the criticality of CPS security.

With over 90 million Americans living with chronic illnesses, accounting for 70 percent of all deaths in the US, the financial burden and demand for secure Medical CPS are compelling. Chronic illnesses contribute to approximately 90 percent of the total 3.8 trillion dollars of healthcare costs in the US, with a significant percentage of the aging population expected to face diagnoses. Amid this context, the safe development of Medical CPS has become a pressing imperative [7].

Medical CPS encompasses three fundamental functionalities: sensor integration on the human body to collect organ-specific data, expert medical decision-making based on accessible reports, and precise prescription recommendations to regulate organ behavior. Compromising a medical CPS through attacks exposes sensitive medical data to unknown adversaries, elevating the risk of data falsification and erroneous prescriptions. The implications are grave, as incorrect drug dosages can have severe health repercussions, including potentially fatal outcomes.

Beyond medical contexts, several instances underscore the vulnerability of critical applications, leading to unexpected and devastating consequences. A failure in an electric power transmission grid can trigger widespread blackouts, impacting multiple cities within a country. Reports reveal successful cyberattacks on power systems resulting in substantial disruptions [8]. Instances like the development of CarShark, capable of remotely disabling car brakes via CAN packet sniffing, demonstrate the dire ramifications of compromised systems [9]. Notably, the interconnectedness of CPS and physical systems can lead to cyberattacks directly influencing physical infrastructure, as seen in the German steel-mill incident [10].

The cornerstone of CPS security revolves around ensuring safety, security, and real-time functionality. Achieving this demands lightweight security solutions [2] that enhance CPS resilience against Cyber-Physical and Cyber-Attacks [11]. Amidst these challenges, the necessity to foster CPS security that aligns with the complexities of their hybrid nature remains paramount.

At the heart of these challenges lies the vital need for secure communication. Ensuring the confidentiality, integrity, and availability of data within CPS is a paramount concern. Data security is not just a matter of protecting information—it directly influences the operational integrity of safety-critical systems. A breach in security could have dire implications, such as compromised patient care in healthcare settings or accidents stemming from tampered sensor data in aviation. The conventional methods of software-based security that have served well in other domains exhibit limitations in the context of CPS. Resource constraints, real-time demands, and the necessity for deterministic behavior pose hurdles for the application of traditional security mechanisms. These methods, designed with enterprise IT systems in mind, struggle to adapt to the unique and demanding requirements that CPS environments entail. In response to these challenges, the realm of hardware-based security solutions comes to the forefront, promising to bolster the security of CPS. Among these solutions, Field-Programmable Gate Arrays (FPGAs) stand out as a beacon of potential. FPGAs introduce unprecedented flexibility in implementing cryptographic algorithms at the hardware level. Their reprogrammable nature facilitates the creation of security solutions tailored to the distinctive demands of CPS environments.

By weaving the intricate fabric of CPS, the security challenges it presents, and the need for robust encryption mechanisms made possible through hardware solutions like FPGAs, this section sets the stage for a comprehensive exploration of the intricate interplay between technology and security in the realm of CPS.

1.2. Evolution of Encryption Techniques and the Role of FPGAs

The evolution of encryption techniques has witnessed a dynamic shift from conventional software-based approaches towards hardware-based solutions. This transition has been particularly pronounced in the context of CPS, where the convergence of physical and digital domains necessitates robust security mechanisms. One pivotal player in this evolution is the deployment of FPGAs, which offer distinctive advantages for implementing cryptographic algorithms at the hardware level. FPGAs stand as adaptable systems enabling the realization of intricate application-specific logic designs through an array of programmable logic gates. These gates can be configured to fulfill specific functionalities, relying on pivotal FPGA building blocks that operate in synergy: the interconnect, the fabric logic, and the configurable logic blocks or CLBs [12].

The Interconnect and Logic Fabric: The interconnect constitutes a network of wires interlinking diverse FPGA components to facilitate signal routing. It establishes connections between the CLBs and other functional units, ensuring effective signal propagation while minimizing delay and power consumption. Complementing this, the fabric logic, also known as the programmable interconnect array (PIA), encompasses programmable switches governing signal routing between the interconnect and CLBs. This component offers configurable pathways for signals, enabling dynamic reconfiguration and optimal utilization of the interconnect.

Configurable Logic Blocks: At the core of an FPGA, the CLBs serve as foundational elements, housing Look-Up Tables (LUTs) as programmable memory units capable of data storage and processing. The incorporation of LUTs empowers intricate Boolean function implementation, rendering the FPGA adaptable across a range of applications. Notably, prominent FPGA manufacturer Xilinx adopts a distinctive CLB architecture encompassing a LUT, a flip-flop, and a carry chain, enhancing CLB functionality and versatility.

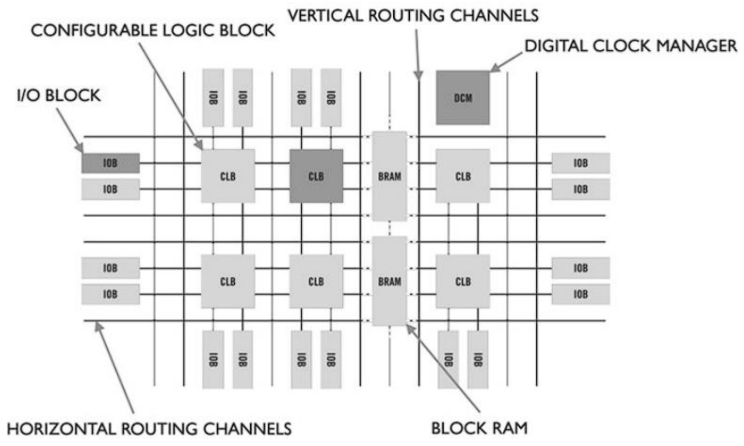


Figure 2 FPGA Architecture. Reprinted From [12]

Moreover, contemporary FPGAs integrate various specialized cores, often provided by third-party vendors, alongside additional memory components. Memory plays a pivotal role in FPGA architecture by enabling real-time data storage and retrieval, thereby facilitating rapid and efficient computation. FPGAs incorporate both on-chip and off-chip memory resources, serving various purposes such as data buffering, caching, and storage. Harnessing these memory resources significantly augments FPGA performance, especially in applications demanding substantial data processing and storage capabilities [12].

Historically, software-based encryption methods have been the cornerstone of data security strategies. These techniques operate in tandem with functional processes, using algorithms to scramble and protect data. However, as the complexity and sensitivity of CPS applications increased, the limitations of software-based security mechanisms became apparent. These methods, while effective in conventional settings, encountered challenges in the resource-constrained environment of CPS. The heavy computational demands of software-based encryption strained the already limited computing and memory resources, compromising real-time performance and responsiveness. To address these limitations, the security landscape witnessed a significant transformation with the emergence of hardware-based solutions. Hardware implementations of encryption algorithms offer a distinct advantage by capitalizing on the parallel processing capabilities of specialized hardware components. This hardware-centric approach optimizes performance by offloading encryption tasks from the central processing units, thereby enhancing the overall efficiency and real-time responsiveness of CPS.

FPGAs also present numerous advantages in Edge Computing for CPS and IoT applications, as well as in hardware-based security solutions. Their flexibility allows for dynamic adaptation to changing computing requirements, making them well-suited for the adaptability demanded in the forementioned scenarios. Moreover, FPGAs deliver high computing performance through parallel processing and hardware-based architecture. Their reconfigurability supports time-multiplexing of computing resources, enhancing suitability for CPS adaptability. Additionally, FPGAs have emerged as a pivotal technology in hardware-based security, enabling tailored hardware implementations of cryptographic algorithms. Their parallel processing capabilities enhance encryption efficiency, while runtime reconfigurability optimizes resource utilization to meet evolving security needs. In both contexts, FPGAs offer a compelling solution for high performance, energy efficiency, and adaptability demands [13].

By recognizing the evolution of encryption techniques from software-based approaches to hardware-centric solutions and acknowledging the significant influence of FPGAs in this paradigm shift, it becomes evident that harnessing hardware for bolstering data security within CPS yields substantial implications.

1.3. Enhancing Cybersecurity Through AES in CPS

The landscape of CPS demands resilient and effective security measures to safeguard data transmissions and critical operations within these infrastructures. The evolution of encryption techniques from conventional software-based approaches to hardware-based solutions has been pivotal in meeting these security requirements. This transition aligns seamlessly with the growing importance of the Advanced Encryption Standard (AES) in ensuring the integrity and confidentiality of data transmissions within CPS environments.

As CPS have advanced, the limitations of software-based security methods have become apparent, particularly in relation to the real-time demands and resource constraints these systems face. This transition has prompted the adoption of FPGAs as a powerful tool for enhancing data security within CPS. FPGAs, with their adaptability, efficiency, and reconfigurability, present a promising avenue for addressing the unique security challenges of CPS and bolstering the protection of sensitive data. Amid the array of cryptographic algorithms, the AES has emerged as a cornerstone for ensuring secure data communication. Its robust encryption mechanism, providing confidentiality, integrity, and authenticity, makes it essential for safeguarding data transmission, especially in critical

CPS scenarios. The wide adoption of AES across industries highlights its significance in secure communication.

Despite the growing importance of AES and hardware-based security approaches in CPS, there exists a gap in achieving optimal AES implementation on FPGAs. Existing research provides valuable foundations, but a need persists for enhancing AES's performance in CPS contexts. Consequently, the central objective of this thesis is to address this gap. It aims to meticulously design and evaluate FPGA-based implementations of the AES algorithm to enhance both its efficiency and efficacy within the resource-constrained CPS environments. The research seeks to elevate AES implementations, offering improved data security and optimized performance for critical CPS operations. In this context, this section lays the groundwork by emphasizing the shift towards hardware-based encryption, the pivotal role of FPGAs, and the significance of AES in the realm of CPS data security. The upcoming sections will delve into the intricacies of AES implementation on FPGAs, detailing the methodologies employed to achieve the overarching research objectives.

2. HARDWARE IMPLEMENTATION OF AES

2.1. Cryptographic Solutions From Software to Hardware - Overview of Encryption Methods and Their Limitations

In the realm of securing data transmissions within CPS, cryptographic algorithms play a fundamental role by concealing information through mathematical functions. These algorithms encompass two key operations: encryption and decryption. Encryption transforms plain text into cipher text, while decryption reverts the cipher text back to its original form [14]. These operations are executed using cryptographic keys, leading to the classification of cryptographic algorithms into two main categories: symmetric cryptography (SC) and asymmetric cryptography (AC).

Symmetric cryptography (also known as secret-key or shared-key cryptography) operates with a shared secret key between the sender and receiver for both encryption and decryption processes. This approach ensures that the intended recipient is the sole entity capable of recovering the original plain text, contingent upon keeping the shared key confidential. Consequently, symmetric cryptography offers a degree of assurance as data encrypted with a specific symmetric key cannot be decrypted using any other. Furthermore, the reliance on a single key eliminates the need for intricate mathematical calculations and resource-intensive processing for generating and handling additional keys. This efficiency makes symmetric encryption algorithms notably faster compared to their asymmetric counterparts [15], rendering them a favorable choice for resource-constrained applications.

Nevertheless, a notable limitation of secret-key cryptography arises in the secure sharing of the secret key. The necessity to maintain key privacy often involves encrypting the secret key with another key, creating a perpetual dependency on additional keys [16].

Numerous algorithms have emerged to define symmetric key cryptography, often operating in rounds. Each round applies mathematical operations, such as substitutions and permutations, to transform plaintext into ciphertext and vice versa. These operations typically act on fixed-size data blocks, rather than the entire message at once. The substitution box (S-box) is a common component of symmetric algorithms, functioning as a lookup table for substituting values during encryption or decryption. The S-box contributes to properties like confusion and diffusion, essential for robust cryptographic algorithms.

Moving on to asymmetric cryptography, also known as public-key cryptography, this class employs a pair of keys: a public key and a private key. The public key can encrypt a message, and only the intended recipient with the corresponding private key can decrypt it. Asymmetric cryptography offers unique advantages, including the elimination of key distribution issues since key exchange is unnecessary. This enhances security, as private keys are never transmitted or disclosed. Additionally, it enables the use of digital signatures to verify message origins. However, the computational complexity of asymmetric cryptographic algorithms often renders them slower compared to symmetric alternatives.

In summary, this section delves into the overarching landscape of cryptographic solutions. It encompasses symmetric cryptography, which leverages shared keys for encryption and decryption, and asymmetric cryptography, characterized by public-private key pairs. By comprehensively understanding these cryptographic categories, we establish a firm foundation for exploring the subsequent hardware implementations within the context of AES and CPS security.

2.2. Unraveling the AES Algorithm - Deep Dive Into the AES Algorithm and Its Significance

Be sure that there is at least one line of text below any subheading at the bottom of a page. Within the realm of securing data transmissions in Cyber-Physical Systems (CPS), the Advanced Encryption Standard (AES) emerges as a critical cornerstone, employing a substitution-permutation network algorithm to obfuscate data through multiple rounds of encryption [17]. AES encryption supports three key lengths: 128-bit, 192-bit, and 256-bit, while consistently maintaining a fixed block size of 128 bits or 16 bytes. As the key length increases, so does the security level, albeit at the cost of augmented power consumption and memory requirements within CPS [17]. This encryption scheme employs varying numbers of rounds based on the key length. AES-128 utilizes 10 rounds, AES-192 uses 12 rounds, and AES-256 employs 14 rounds. The AES algorithm stands as the cornerstone of symmetric encryption, operating on fixed-size data blocks and leveraging a sophisticated substitution-permutation network (SPN) structure that comprises multiple rounds of distinct operations, each contributing to the algorithm's formidable security [18].

Central to the AES algorithm's operations is the state matrix: a 4x4 matrix that serves as the intermediary repository for data during both encryption and decryption processes. The innate structure of the algorithm mandates adherence to a series of fundamental steps, each aimed at ensuring the confidentiality and integrity of the transmitted data.

1. **Substitute Bytes:** In this phase, every byte within the state matrix undergoes replacement with a corresponding byte sourced from a predefined S-box. This substitution is denoted by the transformation equation:

$$\mathrm{state}(i, j) = \mathrm{Sbox}(\mathrm{state}(i, j))$$

This intricate byte substitution step contributes significantly to the algorithm's ability to obfuscate data, enhancing its security.

2. **Shift Rows:** Shifting rows is a cyclic operation applied individually to each row within the state matrix. The first row remains unaltered, while the subsequent rows experience cyclic left shifts. The shift progression is as follows:

$$\mathrm{state}(i, j) = \mathrm{state}(i, (j+i) \bmod 4)$$

This operation plays a crucial role in disassociating the relationship between the original data and the encrypted result, further fortifying the encryption process.

3. **Mix Columns:** The mix columns operation targets columns within the state matrix. Each column is subjected to a complex manipulation involving polynomial multiplication within a finite field. The formula for this transformation is:

$$\mathrm{state}\ (i, j) = (2 * \mathrm{state}(i, j)) \oplus (\ 3 * \mathrm{state}((i+1) \bmod 4, j)) \oplus \mathrm{state}((i+2) \bmod 4, j) \oplus \mathrm{state}((i+3) \bmod 4, j)$$

This intricate mixing operation contributes significantly to the diffusion of the data, preventing the persistence of any recognizable patterns.

4. **Add Round Key:** The final step in each encryption round is the addition of a round key. The 128-bit input data matrix is XORed with a 128-bit round key, ensuring that each encryption round operates on a distinct key. This step adds a unique layer of complexity to the encryption process:

```
state(i, j) = state(i, j) ⊕ RoundKey(i, j)
```

The fusion of these operations within each round engenders an encryption mechanism that is both robust and intricate, rendering it highly resistant to cryptanalysis. As the AES algorithm seamlessly iterates through these rounds, the transformation of the state matrix evolves, culminating in an encrypted output that effectively safeguards data within the CPS environment.

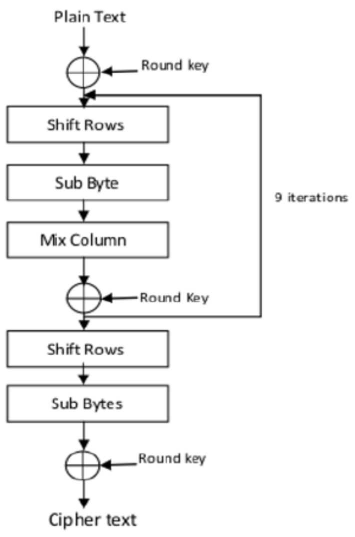


Figure 3 AES Architecture Block Diagram. Reprinted From [19]

In the context of CPS security, the AES algorithm's distinctive structure and encryption methodology make it an indispensable asset for safeguarding sensitive data during transmissions. The upcoming sections will venture into the world of FPGA-based AES implementation, leveraging the strengths of FPGAs to augment the efficiency and efficacy of this encryption process within the realm of CPS. Through a meticulous examination of the intricacies in AES, this research seeks to fortify data security within the dynamic and resource-constrained landscape of CPS.

2.3. Previous AES Implementations - Reviewing Existing Research on AES Implementations

Transitioning from the exploration of the role of cryptography in CPS, our attention now turns to the examination of existing AES implementations in this domain. This investigation serves as a vital bridge between AES theory and practical execution, enabling us to glean insights from past successes and limitations. By dissecting diverse strategies, we extract valuable lessons that inform our approach in designing CPS-focused AES architectures. This analysis equips us to transcend prior constraints and tailor AES implementations to CPS demands. As we navigate this landscape, we gather crucial knowledge that propels us towards innovative and adaptable solutions. This synthesis of past experiences empowers us to craft resilient FPGA-based AES implementations optimized for CPS, enhancing its cybersecurity landscape.

The research in [20] introduces an optimized AES architecture for CTR mode, achieving a very high throughput. By strategically inserting registers, the design minimizes byte transformation delay within one clock period. Simulated on Xilinx Foundation ISE 10.1, it achieves a clock frequency of 576.07MHz and a resource efficiency of 3.21Mbps/LUT. This design outperforms previous solutions in throughput and resource utilization, presenting a valuable advancement for high-speed encryption applications. However, the model name of the FPGA used in the implementation of the proposed design is not explicitly mentioned in the paper. The authors state that they used Xilinx Foundation ISE 10.1 FPGA design tool for the synthesis of the design, but the specific hardware is not mentioned.

Another paper [21] introduces an FPGA-based AES implementation for cryptographic applications. The authors propose a novel approach by incorporating a modified parallel-pipelined round module (MPPRM) to optimize hardware resource usage and minimize circuit delay. While the paper delves into the intricacies of the AES-MPPRM architecture and explores its modified key expansion module, it leans toward a mathematical and theoretical orientation. Notably, the study doesn't extensively address the practical applicability of the proposed architecture in the context of CPS. However, it's worth noting that the paper doesn't specify the FPGA model used for implementation and compares the simulations on different FPGA platforms.

Going over the previous AES implementations, a notable trend comes to light – a significant portion of the existing research lacks precise details regarding the hardware implementations of these architectures. This void highlights a critical gap in the field of CPS-focused AES design. While various papers offer innovative strategies and optimizations, the absence of explicit hardware specifications and platform information limits their practicality and comparability. This limitation underscores the pressing need for a comprehensive and rigorous approach, which this thesis endeavors to fulfill. By embarking on a journey to build foundational optimizations from the ground up, this thesis project aims to bridge this gap and provide robust, adaptable, and optimized FPGA-based AES architectures specifically tailored for the demands of CPS environments. Through this research project, we strive to empower CPS with enhanced resilience and efficiency, ushering in a new era of secure communication and operation.

3. FPGA DESIGN OF AES

3.1. Platform Selection: Leveraging Artix-7 and Vivado Design Suite for AES

3.1.1. Hardware Platform

To realize the objectives of this thesis, a meticulous selection of the appropriate hardware platform was imperative. The chosen platform, the Artix-7 FPGA, is a widely recognized and versatile device from Xilinx, uniquely positioned to cater to the intricate demands of our AES design project within the realm of CPS.

The Artix-7 FPGA, renowned for its balance of high-performance capabilities and low-power consumption, is an ideal candidate for a range of applications, including wireless communication, aerospace, and defense. Its diverse resource offerings and performance variations make it well-suited for intricate cryptographic tasks, such as AES, within the context of CPS security enhancement. Characterized by its extensive range of resources, the Artix-7 FPGA family boasts up to 215K logic cells and clock speeds reaching up to 600 MHz. These capabilities empower it to accommodate complex AES implementations while ensuring real-time and high-throughput operations demanded by CPS environments. With up to 500 user-defined I/O pins that can be flexibly configured as single-ended or differential I/O [22], the Artix-7 FPGA aligns perfectly with the requirements of secure data transmission in CPS.

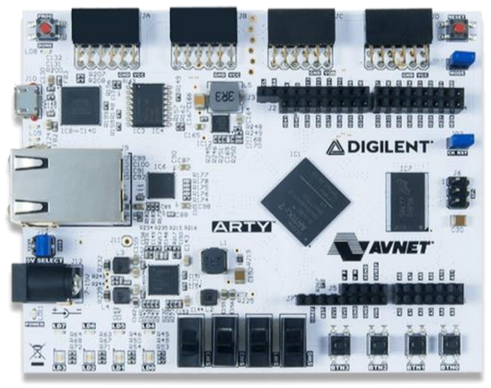


Figure 4 Artix 7 Development Board. Reprinted From [23]

Moreover, the Artix-7 FPGA family's inclusion of up to 100 differential pairs further fortifies its suitability for CPS scenarios by facilitating high-speed communication interfaces like DDR3/4 memory, PCIe, and Ethernet. This diverse set of connectivity options ensures that the designed AES architectures can seamlessly integrate with various CPS components while maintaining efficient and secure data exchange. The availability of on-chip memory resources, including block RAMs, distributed RAMs, and UltraRAMs, further amplifies the FPGA's capability to store and manipulate data efficiently, aligning with the memory-intensive requirements of cryptographic operations. Additionally, with up to eight GTX transceivers and up to four PCIe Gen2/3 hard IP blocks, the Artix-7 FPGA family ensures that CPS security enhancements are seamlessly integrated into the broader CPS ecosystem.

Overall, the Artix-7 FPGA family provides a good balance between performance, power consumption, and cost. With its wide range of resources and connectivity options, it is well-suited for a variety of applications, from high-performance computing to embedded systems.

3.1.2. Software Platform

The IDE used for development in this project is Xilinx Vivado v2021.2. Vivado is a comprehensive design suite used for the development of digital systems based on Xilinx FPGAs and SoCs. The suite offers a wide range of tools and features for designing, implementing, and verifying FPGA-based digital systems, including a graphical user interface for creating and editing HDL code, a plethora of tools for synthesizing, implementing, debugging FPGA designs, with the added support for simulating designs and generating test benches for verification [24].

One of the key features of Vivado is its ability to optimize and implement FPGA designs for the target hardware, taking into consideration the specific resources available on the device. This feature, along with Vivado's extensive library of IP cores, accelerates the design process and enables developers to create complex designs in an efficient manner. For this reason, Vivado is an excellent choice for this project.

Furthermore, Vivado includes a range of debugging and analysis tools such as a logic analyzer, a waveform viewer, and a software debugger, which aids in the development and debugging of FPGA designs. Vivado also provides comprehensive support for the entire RTL to GDS flow, including several stages such as synthesis, place and route, timing analysis, and bitstream generation. The suite includes a Synthesis tool that takes the RTL code as input and generates a gate-level netlist, which can be optimized for the target device's architecture and resources. The synthesized design can then be passed to the implementation stage, where it can be placed and routed onto the target

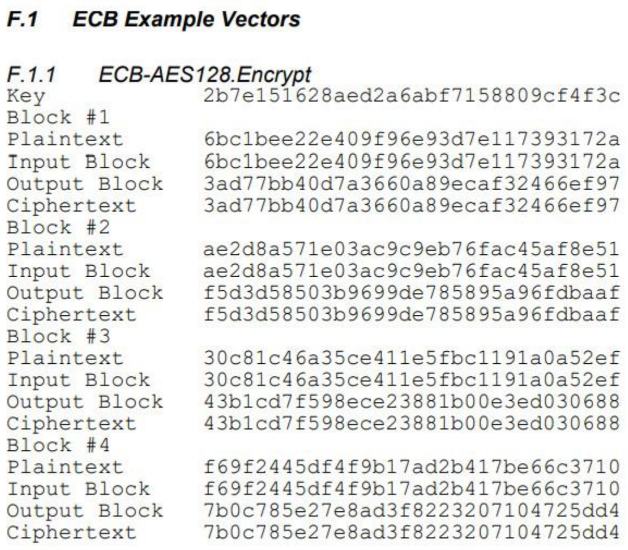
FPGA. The implementation stage also includes tools for performing timing analysis and optimization to ensure that the design meets the timing constraints.

Once the design is implemented, Vivado generates a bitstream file that can be loaded onto the target FPGA – the Artix-7 in this case. The bitstream file contains the configuration information for the FPGA, including the placement and routing information. Another additional useful feature that Vivado provides is the High-Level Synthesis (HLS), which allows the users to generate RTL from high-level languages such as C or C++. Overall, Xilinx Vivado is a powerful and comprehensive tool for the development of FPGA-based digital systems, providing extensive support for the RTL to GDS flow and advanced features for optimizing FPGA designs.

3.2. Test Vectors and Reference Code

In this section, we delve into the intricate details of the FPGA design and implementation process of the AES architecture. Utilizing VHDL (VHSIC Hardware Description Language), a powerful hardware description language, we meticulously outline the structure of our AES architecture. VHDL allows us to define the behavior and structure of digital systems, providing a high-level representation of our AES design. Our approach is substantiated using NIST test vectors, a set of standardized test cases for AES that enables thorough validation of our design's accuracy and compliance with established encryption standards. Additionally, for reference and cross-validation purposes, we present a C implementation of the AES algorithm. This comprehensive exploration underscores our commitment to building a robust and dependable AES architecture tailored for FPGA implementation within CPS environments.

Test vectors are taken from the National Institute of Standards and Technology (NIST). NIST is a U.S. government organization responsible for developing and promoting technology, standards, and measurements to enhance economic security and improve the quality of life. It has played a crucial role in the development of various technologies, including the internet, nanotechnology, and cryptography. NIST is known for its role in developing and maintaining standards such as the AES which is widely used to encrypt sensitive information. It also provides resources and guidelines for cybersecurity and risk management, including the Cybersecurity Framework, which is used by organizations to manage and reduce cybersecurity risk. NIST continues to be a key player in advancing technology and ensuring its security and reliability [25].

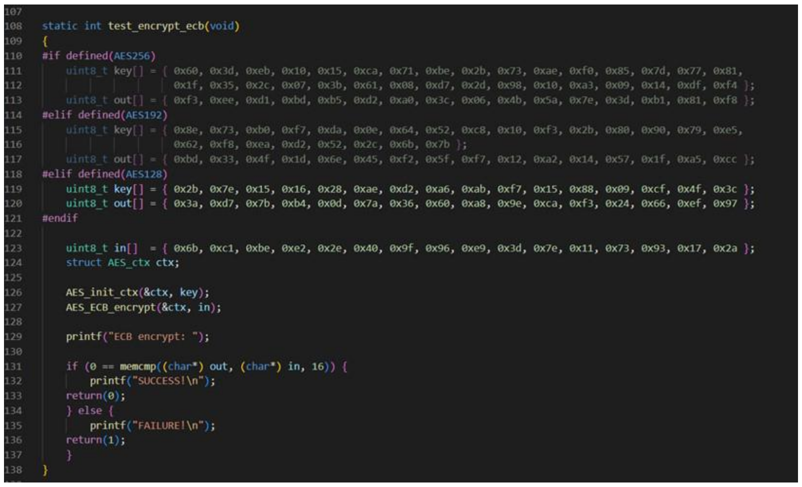


F.1 ECB Example Vectors

F.1.1 ECB-AES128.Encrypt

```
Key             2b7e151628aed2a6abf7158809cf4f3c
Block #1
Plaintext       6bc1bee22e409f96e93d7e117393172a
Input Block     6bc1bee22e409f96e93d7e117393172a
Output Block    3ad77bb40d7a3660a89ecaf32466ef97
Ciphertext      3ad77bb40d7a3660a89ecaf32466ef97
Block #2
Plaintext       ae2d8a571e03ac9c9eb76fac45af8e51
Input Block     ae2d8a571e03ac9c9eb76fac45af8e51
Output Block    f5d3d58503b9699de785895a96fdbaaf
Ciphertext      f5d3d58503b9699de785895a96fdbaaf
Block #3
Plaintext       30c81c46a35ce411e5fbc1191a0a52ef
Input Block     30c81c46a35ce411e5fbc1191a0a52ef
Output Block    43b1cd7f598ece23881b00e3ed030688
Ciphertext      43b1cd7f598ece23881b00e3ed030688
Block #4
Plaintext       f69f2445df4f9b17ad2b417be66c3710
Input Block     f69f2445df4f9b17ad2b417be66c3710
Output Block    7b0c785e27e8ad3f8223207104725dd4
Ciphertext      7b0c785e27e8ad3f8223207104725dd4
```

Figure 5 NIST Test Vectors. Reprinted From [25]



```c

static int test_encrypt_ecb(void)
{
#if defined(AES256)
    uint8_t key[] = { 0x60, 0x3d, 0xeb, 0x10, 0x15, 0xca, 0x71, 0xbe, 0x2b, 0x73, 0xae, 0xf0, 0x85, 0x7d, 0x77, 0x81,
                      0x1f, 0x35, 0x2c, 0x07, 0x3b, 0x61, 0x08, 0xd7, 0x2d, 0x98, 0x10, 0xa3, 0x09, 0x14, 0xdf, 0xf4 };
    uint8_t out[] = { 0xf3, 0xee, 0xd1, 0xbd, 0xb5, 0xd2, 0xa0, 0x3c, 0x06, 0x4b, 0x5a, 0x7e, 0x3d, 0xb1, 0x81, 0xf8 };
#elif defined(AES192)
    uint8_t key[] = { 0x8e, 0x73, 0xb0, 0xf7, 0xda, 0x0e, 0x64, 0x52, 0xc8, 0x10, 0xf3, 0x2b, 0x80, 0x90, 0x79, 0xe5,
                      0x62, 0xf8, 0xea, 0xd2, 0x52, 0x2c, 0x6b, 0x7b };
    uint8_t out[] = { 0xbd, 0x33, 0x4f, 0x1d, 0x6e, 0x45, 0xf2, 0x5f, 0xf7, 0x12, 0xa2, 0x14, 0x57, 0x1f, 0xa5, 0xcc };
#elif defined(AES128)
    uint8_t key[] = { 0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7, 0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c };
    uint8_t out[] = { 0x3a, 0xd7, 0x7b, 0xb4, 0x0d, 0x7a, 0x36, 0x60, 0xa8, 0x9e, 0xca, 0xf3, 0x24, 0x66, 0xef, 0x97 };
#endif

    uint8_t in[]  = { 0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96, 0xe9, 0x3d, 0x7e, 0x11, 0x73, 0x93, 0x17, 0x2a };
    struct AES_ctx ctx;

    AES_init_ctx(&ctx, key);
    AES_ECB_encrypt(&ctx, in);

    printf("ECB encrypt: ");

    if (0 == memcmp((char*) out, (char*) in, 16)) {
        printf("SUCCESS!\n");
    return(0);
    } else {
        printf("FAILURE!\n");
    return(1);
    }
}
```

Figure 6 Reference Implementation in C

3.3. FPGA AES Architecture Design in VHDL With Testbench

With a solid understanding of the hardware platform and the design environment in place, the focus now shifts to the core of this project - the FPGA design and implementation of the AES architecture. This section delves into the intricacies of implementing AES using VHDL, along with the comprehensive methodology of testing and verification through testbenches. The AES architecture is explored in three distinct variants:

1. standard ECB mode,
2. pipelined architecture and,
3. unfolded architecture.

Each variant is meticulously constructed and optimized for efficiency, aiming to address the specific requirements of different CPS scenarios. The journey of translating AES theory into functional VHDL code, conducting rigorous testing, and optimizing the design unfolds in the upcoming subsections, revealing the hands-on aspect of this research project.

3.3.1. Standard

In the realm of CPS, the need for reliable encryption techniques cannot be overstated. As we delve into the FPGA design and implementation of the AES architecture, the standard Electronic Codebook (ECB) mode takes center stage. This subsection delves into the intricacies of the standard implementation, shedding light on the crucial role it plays in securing data transmission within CPS environments. By adhering to the AES standard and leveraging the fundamental ECB mode, we aim to establish a strong foundation for further optimizations tailored to the unique demands of CPS scenarios. The subsequent sections will explore the development process, testing methodologies, and the outcomes of this standard AES implementation, contributing to the larger goal of enhancing security and efficiency in CPS through FPGA-based solutions. This implementation will be used as a control to compare and weigh the two optimization techniques that this research evaluates.

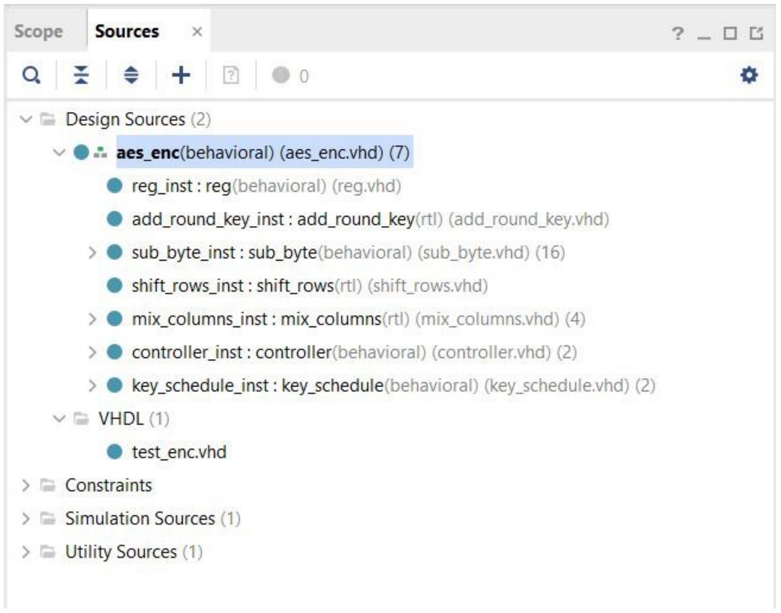
The standard AES implementation entails the realization of the AES block structure depicted in Figure 3. This structure encompasses four main transformation steps: Substitute Bytes, Shift Rows, Mix Columns, and Add Round Key, as outlined in Section 3.2. The top-level AES entity orchestrates these transformations through four sub-entities: sub_byte, shift_rows, mix_columns, and add_round_key. The VHDL description of these entities translates the theoretical AES steps into functional code. The top-level entity takes inputs:

- encryption key,
- plaintext data,
- reset signal, and
- clock signal,

and generates the outputs:

- ciphertext, along with
- done signal.

At the core of VHDL, an "entity" serves as a fundamental construct used to define the structure and behavior of a digital component. In the context of the AES implementation, an entity encapsulates the functionality of a specific module or sub-component. Each of these steps is realized as an "entity" in VHDL, encapsulating the necessary logic and operations that correspond to their respective AES steps.

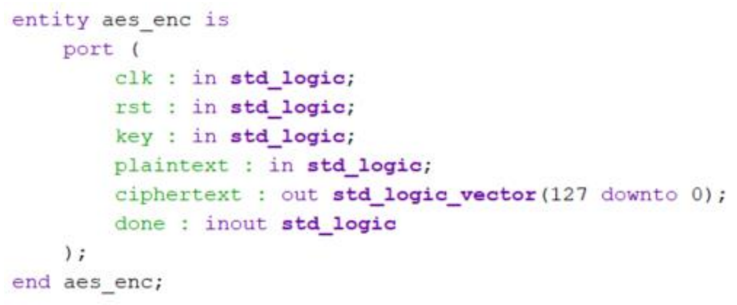

Scope Sources

Design Sources (2)
aes_enc(behavioral) (aes_enc.vhd) (7)
reg_inst : reg(behavioral) (reg.vhd)
add_round_key_inst : add_round_key(rtl) (add_round_key.vhd)
sub_byte_inst : sub_byte(behavioral) (sub_byte.vhd) (16)
shift_rows_inst : shift_rows(rtl) (shift_rows.vhd)
mix_columns_inst : mix_columns(rtl) (mix_columns.vhd) (4)
controller_inst : controller(behavioral) (controller.vhd) (2)
key_schedule_inst : key_schedule(behavioral) (key_schedule.vhd) (2)
VHDL (1)
test_enc.vhd
Constraints
Simulation Sources (1)
Utility Sources (1)

Figure 7 Entities That Make up AES HDL Model

In addition to the four main transformation sub-entities, the AES architecture includes two crucial components: the key_schedule and the controller. The key_schedule is responsible for generating round subkeys, which are used in each AES round for the add_round_key step. The Controller manages the AES rounds by keeping track of a counter signal, ensuring that each transformation occurs in the correct sequence. Notably, the last round of AES skips the Mix Columns step, and the controller plays its part to make sure the implementation respects the algorithm.

Equally pivotal to the actual implementation steps is a critical factor that demands utmost consideration – the management of data flow. Just as the encryption steps are integral to the AES process, so too is the intricately devised strategy for handling the flow of data. Instead of transmitting the entire key and plaintext on separate pins, a serial data transmission process is employed. This technique significantly reduces the number of pins required for data transmission, saving power in CPS scenarios. The serial data transmission process involves loading the data from the testbench into a 128-array, which is then transmitted serially over a single pin. Although this method introduces some overhead, it is a strategic trade-off to optimize power consumption in CPS.



```
entity aes_enc is
    port (
        clk : in std_logic;
        rst : in std_logic;
        key : in std_logic;
        plaintext : in std_logic;
        ciphertext : out std_logic_vector(127 downto 0);
        done : inout std_logic
    );
end aes_enc;
```

Figure 8 VHDL Code of Top-Level Entity

However, it's important to note that for metrics calculations, the overhead introduced by the "READ" process will be omitted, as its primary purpose is power optimization. This standard AES implementation lays the groundwork for further optimizations, catering to CPS-specific requirements. The subsequent sections will detail the intricacies of pipelined and unfolded architectures, adding depth to our exploration of FPGA-based AES implementations tailored for enhanced performance within CPS contexts.

3.3.2. Pipelined

In the realm of VLSI, optimization techniques play an indispensable role in achieving enhanced performance. One of the most impactful strategies in this domain is pipelining, a technique that efficiently divides a complex process into multiple stages, allowing for parallel execution of tasks. This division not only accelerates the overall process but also enables a balanced utilization of hardware resources. In the context of our AES architecture, the introduction of pipelining stands as a crucial enhancement.

Pipelining is implemented by inserting registers between the stages of a process (figure 3.6), enabling the output of one stage to be immediately used as the input for the next stage. This continuous handover of data ensures that various stages can work concurrently on different inputs. In our AES implementation, two potential spots for the addition of a pipeline were identified: between the individual rounds and between the processing stages within a single round. A pipeline is a sequence of stages in which a complex process is divided into smaller tasks or sub-processes. Each stage performs a specific operation on the data and passes it along to the next stage. Pipelining is used to improve the efficiency and speed of processing by allowing multiple stages to work concurrently on different parts of the data [19]. This parallelization reduces the overall time taken to complete a task.

Sub-pipelining refers to breaking down the stages of a pipeline even further. In a sub-pipelined architecture, each stage of the main pipeline is divided into smaller sub-stages. This allows for even finer-grained parallelism and can result in further performance improvements. Sub-pipelining is often used to optimize complex processes in hardware design, where the main pipeline stages might involve multiple sub-steps that can be parallelized. After rigorous evaluation, the final iteration that demonstrated superior results was the introduction of pipelining between the processing stages.

To unlock the true benefits of pipelining, further enhancements were implemented. In the initial design, the AES entity processed a single block of plaintext, limiting the apparent advantages of pipelining. However, research asserts that the substantial benefits of pipelining emerge when multiple blocks are processed. For a single block, the introduction of pipelining can appear counterintuitive due to the area overhead it entails. Yet, for multiple blocks, the efficiency gains overshadow the incremental increase in resource utilization. Therefore, the adoption of multiple-block processing not only maximizes throughput but also lends justification to the trade-off in terms of area utilization.

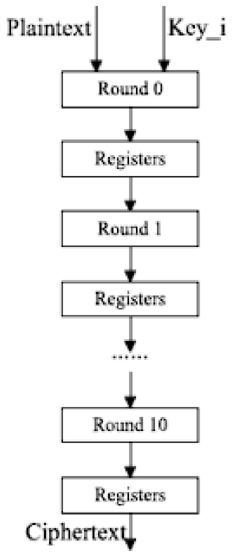


Figure 9 Pipelined Architecture. Reprinted From [20]

The pipelining strategy employed here exemplifies a significant aspect of our research project – the synergistic integration of optimization techniques with a comprehensive understanding of hardware architecture. This strategic fusion enables the realization of designs that transcend conventional limitations and excel in the demanding realm of CPS.

3.3.3. Unfolded

Unfolding, often referred to as loop unrolling, is another optimization technique that merits exploration in the context of AES implementation. While pipelining focuses on parallelism between different stages, unfolding takes advantage of inherent parallelism within a single iteration. This technique involves replicating and expanding the iterative stages of AES to allow for simultaneous processing of multiple data blocks, thereby increasing the overall throughput. In our AES implementation, we delve into the unfolding approach, aiming to capitalize on its potential benefits. Unfolding exploits the inherent parallelism within each AES round, enabling the simultaneous processing of multiple rounds for different data blocks. By unraveling the iterative stages of AES, we aim to enhance both efficiency and throughput.

The anticipated benefits of unfolding are twofold: First, it aligns with the underlying parallel structure of AES, allowing us to fully utilize the available resources. Secondly, similar to pipelining, unfolding also demonstrates its true prowess when applied to scenarios involving multiple blocks of data. Just as we witnessed with the pipelined approach, the trade-off between increased throughput and higher resource utilization becomes more favorable when dealing with multiple data blocks. The unfolding factor plays a pivotal role in this approach. It determines how many times the iterative stages of AES are expanded and replicated. This factor directly influences the amount of parallelism that can be exploited. In our implementation, we opt for an unfolding factor of 2, which means that the steps in each round are duplicated twice. This choice strikes a balance between increased parallelism and resource utilization.

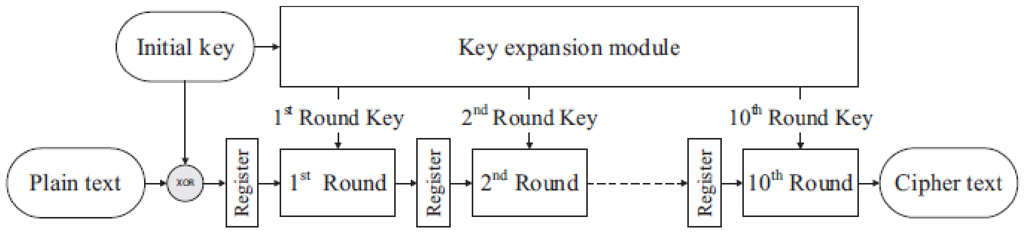


Figure 10 Unfolding in AES. Reprinted From [26]

Unfolding the AES architecture is an intricate process that involves careful design considerations. The iterative nature of AES stages provides fertile ground for exploiting this technique. By expanding and unrolling these stages, we aim to harness the inherent parallelism encoded within the algorithm itself. This approach represents yet another facet of our comprehensive optimization strategy, allowing us to adapt AES to the intricacies of CPS. As we progress through the exploration of unfolding, we aim to uncover the extent to which this technique can contribute to the efficiency and adaptability of our FPGA-based AES implementations.

3.4. Metrics for Evaluating Hardware Implementations

In evaluating the feasibility and effectiveness of hardware implementations, a comprehensive set of metrics is essential to provide a holistic assessment. The performance and resource utilization of the AES implementations in CPS will be critically analyzed across several dimensions. These metrics serve as quantifiable benchmarks to gauge the capabilities of the designed architectures, offering insights into their efficiency and suitability for real-world deployment.

1. **Power**: Power consumption stands as a critical consideration in any FPGA-based system, particularly those operating within CPS environments. This metric quantifies the amount of electrical power consumed during the device's operation, typically measured in watts. The significance of power efficiency lies in its direct impact on battery life and heat generation. The evaluation of power encompasses both dynamic and static power requirements, allowing a comprehensive understanding of the energy footprint of the implementation.

2. **Area**: Area utilization refers to the physical size of the FPGA and the allocation of its internal resources, including lookup tables (LUTs), flip-flops (FFs), and memory blocks. This metric is pivotal in assessing the hardware footprint of the AES implementations. Area utilization is typically presented as a percentage of the total available resources or quantified by the number of slices employed. Optimizing area usage is paramount, especially in applications where compactness or cost-

effectiveness is a priority.

3. **Throughput**: Throughput measures the rate at which data can be efficiently processed by the FPGA-based system. This metric quantifies the system's data handling capacity and is expressed in units like bits per second or transactions per second. For AES implementations tailored to CPS, optimizing throughput is essential for rapid data processing and communication tasks, aligning with the real-time demands of CPS environments.

4. **Timing and Latency**: Timing and latency metrics play a pivotal role in determining the responsiveness and effectiveness of an FPGA-based system. Timing represents the signal propagation time within the system, while latency signifies the duration between input and output generation. Both metrics are crucial in ensuring timely and accurate system responses. Properly defining timing constraints and ensuring their compliance are fundamental steps in achieving the desired system behavior.

5. **Frequency**: The frequency of an FPGA corresponds to its operating clock rate, often measured in units such as MHz or GHz. Enhancing frequency can lead to improved system performance; however, this increase may come at the cost of heightened power consumption and potential timing challenges. Frequency optimization is a careful balance that contributes to the overall efficiency and responsiveness of the system.

As this project aspires to furnish a robust and efficient hardware-based solution to bolster data security within Cyber-Physical Systems (CPS), the triumphant execution of the FPGA implementation of AES is intrinsically tied to its performance across these pivotal metrics. Through a meticulous evaluation of the AES architectures against these dimensions, we glean insights into their pragmatic viability and adaptability within the intricate landscape of CPS. With these foundational aspects in place, we now embark on a comparative analysis of the two distinct techniques—pipelining and unfolding. This analysis delves into their relative merits and comprehensively scrutinizes their performance vis-à-vis these metrics, unraveling their comparative strengths and weaknesses and ultimately charting a course for an optimized AES implementation within CPS scenarios.

4. IMPLEMENTATION ANALYSIS

In the preceding sections, a comprehensive exploration of the FPGA design of the AES algorithm was undertaken. This journey commenced with a foundational understanding of the standard AES variant, encompassing its design principles, architectural intricacies, and code representations. Subsequently, innovative adaptations were explored, including the pipelined and unfolded variants, each characterized by its unique attributes and advantages. The primary objective was to establish a profound comprehension of these AES variants, setting the stage for the ensuing practical implementation analysis.

After completing the RTL design, VLSI designers progress through the phases of synthesis, physical implementation, bitstream generation, and FPGA programming. These phases culminate in an exhaustive phase analysis. The primary focus is on evaluating three essential dimensions of FPGA-based AES encryption: resource utilization and area efficiency, power consumption, and timing performance. These metrics have been comprehensively covered in previous sections.

Vivado Design Suite emerges as a powerful tool for analyzing FPGA designs. Notably, it possesses the capability to generate detailed reports on various facets of the design. These reports can be generated using Tcl commands [27], providing a flexible and customizable means of design analysis. For instance, the "report_utilization" command [27] facilitates the generation of a report concerning FPGA resource utilization, encompassing elements like LUTs, flip-flops, and BRAMs. This report proves invaluable in identifying areas of the design that may be over-utilizing resources, thereby enabling optimization for enhanced performance.

Likewise, the "report_power" [27] command contributes to generating a report that delves into power consumption within the design. This report aids in pinpointing areas of the design consuming excessive power, facilitating optimization for reduced power consumption.

Furthermore, timing analysis, a critical aspect of FPGA design, is facilitated through various Tcl commands provided by Vivado. The "report_timing" command [27], for instance, generates a report on the timing performance of the design, encompassing parameters such as setup and hold times, clock skew, and others. This report assists in identifying areas of the design that may not meet timing constraints, thus guiding optimization efforts for improved timing performance.

Overall, Vivado Design Suite offers a potent toolkit for analyzing FPGA designs, with Tcl commands offering flexibility and customization in generating detailed reports on diverse design aspects. Through effective utilization of these tools, designers can optimize their designs to achieve superior performance, lower power consumption, and enhanced timing performance.

In Section 5.1, a deep dive into the standard AES implementation ensues, involving an intricate examination of resource allocation, area utilization, power characteristics, and timing performance. Transitioning to Section 5.2, the focus shifts to the pipelined AES implementation. The objective remains consistent: the evaluation of resource utilization, area efficiency, power consumption, and timing performance.

Insights garnered from the standard AES analysis serve as a foundation for exploring how pipelining augments data throughput. Finally, Section 5.3 culminates the implementation analysis with the unfolded AES implementation.

Through these meticulous analyses, the aim is to illuminate not only the strengths but also the trade-offs inherent in each AES variant. The overarching goal is to draw meaningful conclusions regarding their suitability for specific application scenarios.

4.1. Data Serialization and Deserialization

Before embarking on the core of this study, it is imperative to address a pivotal concern – the management of 128-bit plaintext and ciphertext signals. The practicality of utilizing 128 pins for both plaintext and ciphertext on an Arty 7 FPGA board presents a formidable challenge.

In response to this challenge, a pragmatic approach has been adopted, namely, data serialization and deserialization. This technique enables the sequential transmission of data over a singular plaintext pin and its subsequent reception over a solitary ciphertext pin.

The schematic representation of our project workflow, employing “serdes”, is depicted below:

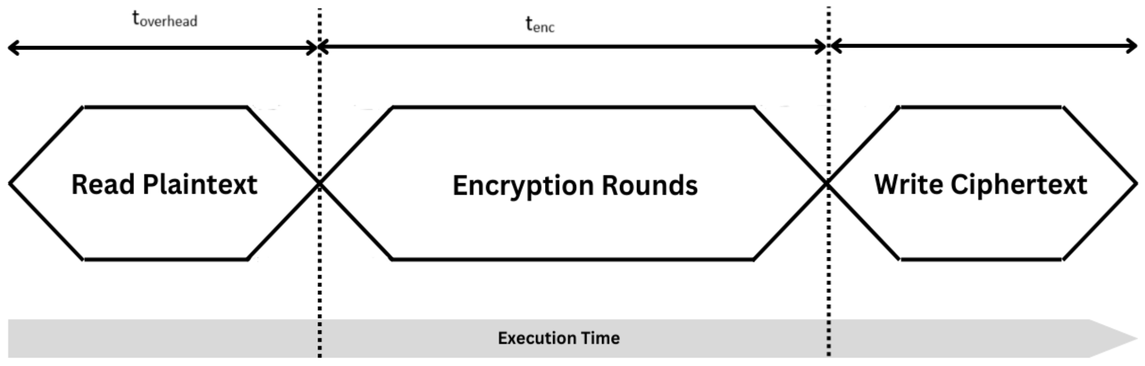


Figure 11 Encryption Flow

Read and write with respect to encryption module inside the FPGA. Through the judicious application of data serialization and deserialization, the resource utilization of the FPGA is optimized, concurrently ensuring the integrity of the 128-bit signals. This methodological refinement serves to streamline our data handling processes, enhancing both efficiency and effectiveness in our study.

4.2. Comprehensive Summary of Project Components

In this section, an overview focusing on the over-arching framework and meticulous planning that underpins the execution of the thesis project is presented. The success of any research endeavor heavily relies on a well-crafted roadmap and a clear delineation of tasks and objectives. The framework guiding the research is examined, emphasizing the strategic choices made to ensure the project's coherence and effectiveness. This comprehensive snapshot, presented in Figure 12, enables the readers to gain a holistic perspective of the aim and purpose of the research.

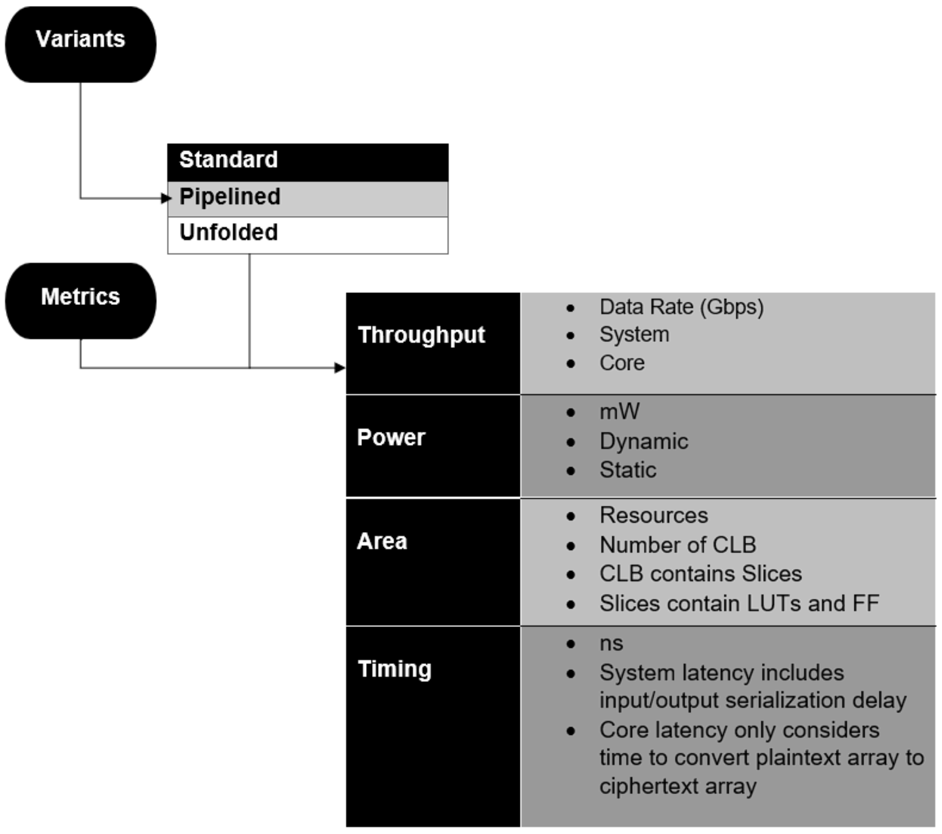

Variants
Standard
Pipelined
Unfolded
Metrics
Throughput
Data Rate (Gbps)
System
Core
Power
mW
Dynamic
Static
Area
Resources
Number of CLB
CLB contains Slices
Slices contain LUTs and FF
Timing
ns
System latency includes input/output serialization delay
Core latency only considers time to convert plaintext array to ciphertext array

Figure 12 Summary of Thesis Project

4.3. Standard Implementation

4.3.1. Area Utilization

The following data represents the outcomes obtained from the execution of the "report_utilization" command. To illustrate the influence of data serialization and deserialization, in this variant the output side is left unserialized. It is from this comparison that one can discern the substantial impact that this technique imparts. In contrast, it is important to note that in the remaining two variants, both input and output data are fully serialized, reflecting a distinct design choice.

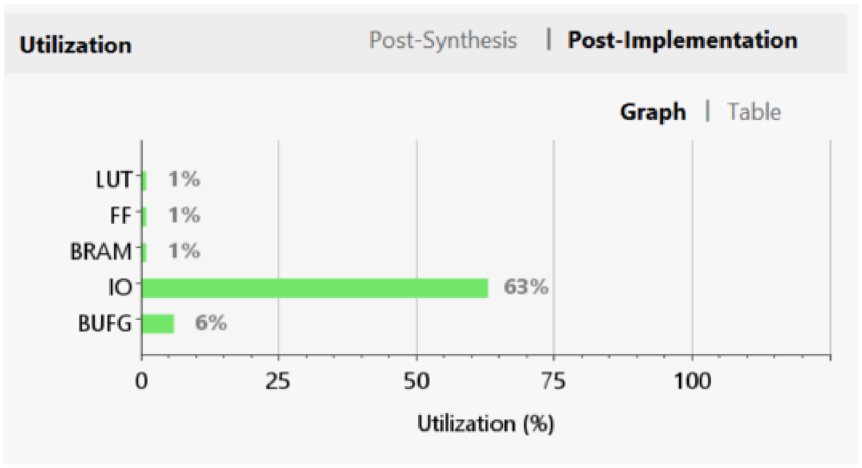
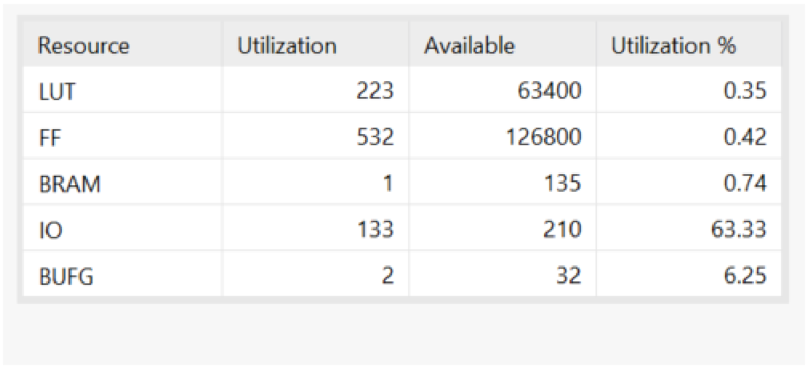


Figure 13 Percentage Utilization



Resource	Utilization	Available	Utilization %
LUT	223	63400	0.35
FF	532	126800	0.42
BRAM	1	135	0.74
IO	133	210	63.33
BUFG	2	32	6.25

Figure 14 Absolute Utilization

A relatively straightforward AES design that does not consume much of the FPGA resources. However, an anomaly in the count of input/output (IO) pins, is conspicuous. This section presents a detailed analysis of the area and resource utilization in the standard AES implementation on an FPGA. Understanding how FPGA resources are utilized is crucial for optimizing the design and ensuring efficient cryptographic processing.

Slice Logic Utilization

The heart of our analysis lies in Slice Logic Utilization, where we examine various critical components:

Table 1 Resource Utilization Snapshot.

Site Type	Used	Available	Utilization
Slice LUTs	223	63400	0.35%
Slice Registers	532	126800	0.42%
F7 Muxes	0	31700	0.00%
F8 Muxes	0	15850	0.00%

This table provides a detailed breakdown of the utilization of Slice Logic components, which are fundamental building blocks in an FPGA. The key components analyzed here include:

- Slice LUTs: These are Look-Up Tables within the slices, and 223 out of 63,400 available are used. LUTs are essential for implementing logic functions.
- Slice Registers: Registers are used for storing intermediate results and control signals. In this case, 532 out of 126,800 available registers are utilized.
- F7 Muxes and F8 Muxes: These are multiplexers within the slices, which are used for selecting different inputs. In this implementation, no F7 or F8 Muxes are used.

Summary of Registers by Type

An in-depth categorization of registers by their functionality:

Table 2 Register Types.

Type	Used
Clock Enable	0
Synchronous	531
Asynchronous	0

This table categorizes registers based on their functionality, providing insights into their utilization:

- Clock Enable: Registers that enable clock control, but in this implementation, none are used.
- Synchronous: 531 registers are employed for synchronous operations, typically associated with clocked data.
- Asynchronous: No asynchronous registers are used.

Slice Logic Distribution

Understanding the distribution of Slice Logic:

Table 3 Distribution of Slices in the Design.

Site Type	Used	Available	Utilization
Slice	142	15850	0.90%
SLICEL	96	-	-
SLICEM	46	-	-
LUT as Logic	223	63400	0.35%

This table focuses on the distribution of Slice Logic components across different sites:

- Slice: A total of 142 out of 15,850 slices are used, indicating that 0.90% of slices are utilized.
- SLICEL and SLICEM: These represent different types of slices with different capabilities. 96 SLICEL and 46 SLICEM slices are utilized.
- LUT as Logic: This category provides additional details about LUT utilization. Out of 63,400 available LUTs, 223 are used. The table also shows how some LUTs are used exclusively for specific outputs.

Memory Utilization

A look into memory utilization:

Table 4 Utilization of Memory in the Standard Implementation.

Site Type	Used	Available	Utilization
Block RAM Tile	1	135	0.74%
RAMB36/FIFO*	0	135	0.00%
RAMB18	2	270	0.74%

*Note: RAMB36/FIFO occupies a Block RAM Tile but is not used in this implementation. Memory utilization is crucial in FPGA designs. This table presents information on memory resources:

- Block RAM Tile: Only 1 out of 135 Block RAM Tiles is used for this implementation, which has implications for data storage.
- RAMB36/FIFO and RAMB18: These represent different types of memory resources, but neither is used extensively in this implementation.

IO Utilization

Table 5 Utilization of I/O in the Standard Implementation.

Site Type	**Used**	**Available**	**Utilization**
Bonded IOB	133	210	63.33%
IOB Master Pads	62	-	-
IOB Slave Pads	66	-	-

IO resources are vital for communication with external devices. This table provides insights into IO resource utilization:

- Bonded IOB: 133 out of 210 bonded Input/Output Blocks are used, indicating that 63.33% of available IOBs are utilized. This may suggest a significant level of external interfacing in the design.
- IOB Master Pads and IOB Slave Pads: These components provide further details about IOBs' master and slave pads usage.

Clocking Utilization

Table 6 Clocking Resource Utilization.

Site Type	**Used**	**Available**	**Utilization**
BUFGCTRL	2	32	6.25%

Efficient clocking is essential for synchronization in FPGA designs. This table outlines clocking resource utilization:

- **BUFGCTRL**: 2 out of 32 BUFGCTRLs are used. BUFGCTRLs play a crucial role in managing clock signals.

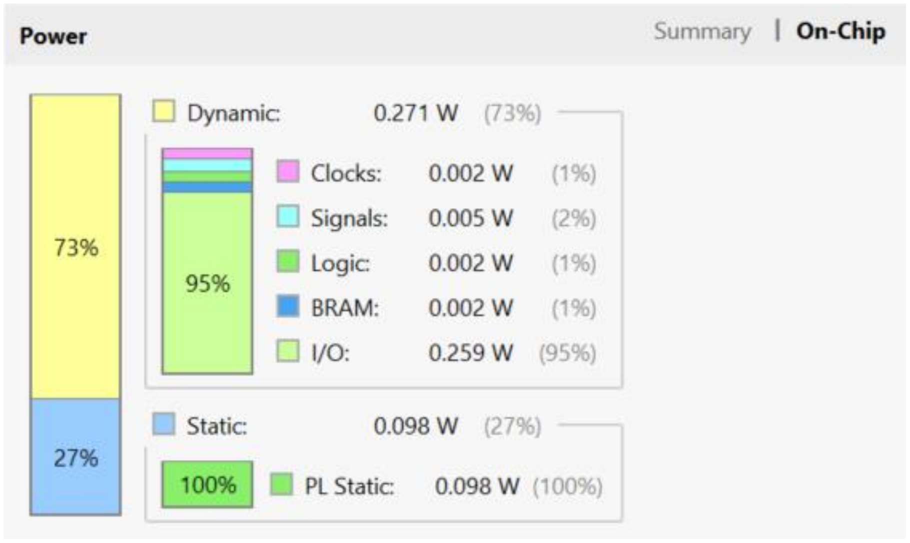
In conclusion, this comprehensive analysis of the standard AES implementation's area and resource utilization on an FPGA highlights key metrics related to Slice Logic, Memory, DSP, IO, and Clocking. These insights are invaluable for optimizing the design and ensuring the efficient deployment of cryptographic algorithms.

4.3.2. Power Efficiency

Overall, the power profile showcases characteristics aligning with standard expectations.

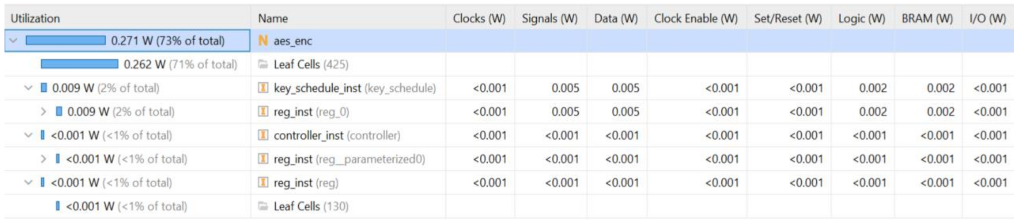
The reason why the I/O percentage stands out prominently is due to the fact that the ciphertext signal is not serialized. In resource-constrained environments, those often encountered by CPS, this significant I/O utilization raises major concerns. This is exactly why we resorted to serialize the input and output signals, despite the attendant overhead. A more detailed exploration of this overhead will ensue as we delve into a comparative analysis of throughput between the pipelined and unfolded implementations.

The standard variant consumed around 300mW of power, where the major chuck of about 70% is attributed to dynamic power. Dynamic power pertains to the power consumed during the switching of logic gates and interconnections, primarily influenced by activity within the circuit. In contrast, static power, often referred to as leakage power, represents the power consumed when transistors are in a non-switching state. It is largely dependent on the transistor characteristics and process technology.



Dynamic:	0.271 W	(73%)
Clocks:	0.002 W	(1%)
Signals:	0.005 W	(2%)
Logic:	0.002 W	(1%)
BRAM:	0.002 W	(1%)
I/O:	0.259 W	(95%)
Static:	0.098 W	(27%)
PL Static:	0.098 W	(100%)

Figure 15 Power Profile



Utilization	Name	Clocks (W)	Signals (W)	Data (W)	Clock Enable (W)	Set/Reset (W)	Logic (W)	BRAM (W)	I/O (W)
0.271 W (73% of total)	aes_enc								
0.262 W (71% of total)	Leaf Cells (425)								
0.009 W (2% of total)	key_schedule_inst (key_schedule)	<0.001	0.005	0.005	<0.001	<0.001	0.002	0.002	<0.001
0.009 W (2% of total)	reg_inst (reg_0)	<0.001	0.005	0.005	<0.001	<0.001	0.002	0.002	<0.001
<0.001 W (<1% of total)	controller_inst (controller)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	reg_inst (reg__parameterized0)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	reg_inst (reg)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	Leaf Cells (130)								

Figure 16 Hierarchical Power Usage

4.3.3. Optimizing Timing and Latency

In this section, we provide a comprehensive summary of the timing analysis for your AES implementation on an FPGA using the unfolding technique. Timing analysis is crucial to ensure that your design meets the required timing constraints and operates correctly.

- Timing Constraints

Before diving into the analysis, let's define some key timing constraints:

- **Setup Time (Setup to Clk):** The amount of time before the clock edge that data must be stable to be properly captured.
- **Hold Time (Hold to Clk):** The amount of time after the clock edge that data must be stable to be properly captured.
- Setup and Hold Times for Input Pins

The following table presents the setup and hold times for each input pin, each of which is associated with the sys_clk_pin:

Table 7 Setup and Hold Times.

Input Pin	Setup Time (ps)	Hold Time (ps)
sys_clk_pin clk	2.68	-0.08
sys_clk_pin key	0.37	1.59
sys_clk_pin plaintext	0.31	1.66
sys_clk_pin rst	1.16	1.94

Timing Metrics Summary

In the upcoming table, we delve into the essential timing metrics pivotal to our design’s performance. This comprehensive overview encapsulates the critical temporal parameters that our design adheres to, ensuring efficiency and precision. These metrics serve as benchmarks for evaluating the design's responsiveness and are crucial for the subsequent optimization processes. Understanding these metrics is key to grasping how our design achieves its performance goals within the set temporal constraints.

Table 8 Summary of Timing Metrics in Each Implementation.

Metric	Description	Value
WNS	Worst-case timing violation indicating potential issues	2.324 ns
TNS	Cumulative negative slack across all paths	0.000 ns
TNS Failing Endpoints	Number of endpoints failing to meet timing	0
TNS Total Endpoints	Total number of analyzed endpoints	530
WHS	Worst-case timing violation for hold time constraints	0.085 ns
THS	Cumulative hold slack across all paths	0.000 ns
THS Failing Endpoints	Number of endpoints failing to meet hold time constraints	0
THS Total Endpoints	Total number of analyzed endpoints for hold time	530
WPWS	Worst-case timing violation for pulse width constraints	4.500 ns

Timing Analysis

In the standard implementation of the AES encryption algorithm, a comprehensive timing analysis was conducted to assess the critical path delays and overall performance of the design. The timing results are summarized below:

- WNS (Worst Negative Slack): This represents the worst-case timing delay by which a signal in the design fails to meet its required arrival time (setup time) with respect to the clock. In your table, the WNS is 2.324 nanoseconds, indicating that the slowest signal is delayed by this amount compared to what's required for proper operation.

- TNS (Total Negative Slack): This is the sum of all negative slack values across all endpoints (signals) in the design. In your table, the TNS is 0.000 nanoseconds, indicating that the design meets all timing requirements collectively.
- TNS Failing Endpoints: This is the number of endpoints (signals) that have negative slack values, meaning they don't meet their timing requirements. In your table, there are 0 failing endpoints, which is a good sign.
- TNS Total Endpoints: This is the total number of endpoints (signals) in the design that are being analyzed for timing. In your table, there are 530 total endpoints.
- WHS (Worst Hold Slack): Similar to WNS, this represents the worst-case timing delay by which a signal in the design fails to meet its required hold time with respect to the clock. In your table, the WHS is 0.085 nanoseconds, indicating that the slowest signal doesn't meet its hold time requirement by this amount.
- THS (Total Hold Slack): This is the sum of all hold slack values across all endpoints in the design. In your table, the THS is 0.000 nanoseconds, indicating that the design meets all hold time requirements collectively.
- THS Failing Endpoints: This is the number of endpoints that have negative hold slack values, meaning they don't meet their hold time requirements. In your table, there are 0 failing endpoints, which is a good sign.
- THS Total Endpoints: This is the total number of endpoints being analyzed for hold time. In your table, there are 530 total endpoints.

- WPWS (Worst Pulse Width Slack): This represents the worst-case timing delay by which a signal's pulse width (duration) fails to meet its required specification. In your table, the WPWS is 4.500 nanoseconds, indicating that the slowest signal's pulse width is too wide for this amount.
- TPWS (Total Pulse Width Slack): This is the sum of all pulse width slack values across all endpoints in the design. In your table, the TPWS is 0.000 nanoseconds, indicating that the design meets all pulse width requirements collectively.
- TPWS Failing Endpoints: This is the number of endpoints that have negative pulse width slack values, meaning they don't meet their pulse width requirements. In your table, there are 0 failing endpoints.
- TPWS Total Endpoints: This is the total number of endpoints being analyzed for pulse width. In your table, there are 269 total endpoints.
- In summary, this table provides a detailed overview of the timing analysis for your digital design, including information about setup time, hold time, and pulse width requirements. The negative slack values (WNS, WHS, WPWS) indicate areas where the design needs improvement to meet its timing constraints, while positive values or zeros indicate that the design meets the timing requirements for those aspects.

It is important to note that these timing metrics were evaluated against specified timing constraints, and any violations should be carefully addressed to ensure the reliable operation of the AES encryption core.

In summary, the timing analysis demonstrates that the standard implementation of the AES encryption algorithm generally meets its timing requirements, with some specific paths that may require optimization to achieve optimal performance.

The statement "All user specified timing constraints are met" indicates that, based on this summary, your design meets all the specified timing constraints, as there are no failing endpoints for setup, hold, or pulse width timings. This comprehensive timing analysis assures that your AES implementation on the FPGA with the standard technique operates within the defined timing parameters. For the subsequent variants we will not go over the definitions of these terms and parameters again. We will dive directly into the numbers to get a clear picture.

4.4. Pipelined Implementation

4.4.1. Area Utilization

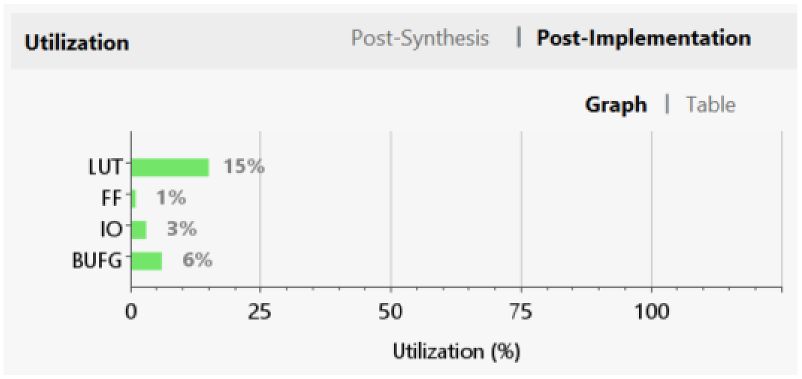
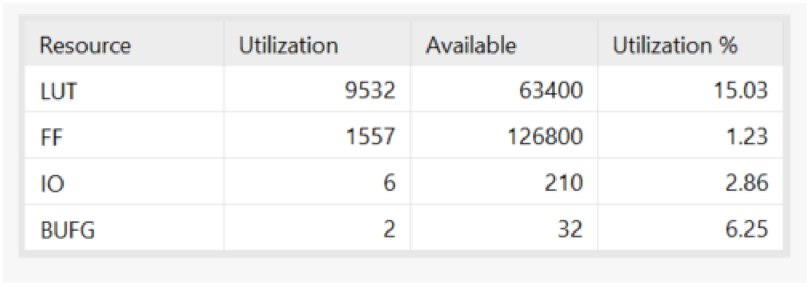


Figure 17 Power Profile for Pipelined Implementation



Resource	Utilization	Available	Utilization %
LUT	9532	63400	15.03
FF	1557	126800	1.23
IO	6	210	2.86
BUFG	2	32	6.25

Figure 18 Hierarchical Power Usage for Pipelined Implementation

In this section, we delve into the area and resource utilization of the pipelined implementation of the "top" design on the XC7A100TCSG324-1 device using Vivado v.2021.2. Understanding the utilization of various hardware resources is crucial in assessing the efficiency and scalability of the design.

Slice Logic

The utilization of slice logic provides insights into the core computational elements of the design. Table 9 summarizes the key statistics related to slice logic utilization:

Table 9 Slice Logic Utilization

Site Type	Used	Fixed	Prohibited	Available	Utilization
Slice LUTs	9532	0	0	63400	15.03%
LUT as Logic	9532	0	0	63400	15.03%
Slice Registers	1557	0	0	126800	1.23%
F7 Muxes	2968	0	0	31700	9.36%
F8 Muxes	1296	0	0	15850	8.18%

- **Slice LUTs**: A significant portion of the available LUTs (Look-Up Tables) is used, constituting approximately 15.03% of the total available.
- **Slice Registers**: Registers, which are essential for pipelining and sequential operations, are utilized at a rate of 1.23%, indicating efficient use of register resources.
- **F7 and F8 Muxes**: These multiplexers are employed in routing and control within the slices, with F7 Muxes at 9.36% utilization and F8 Muxes at 8.18%, showcasing their importance in the design.

Summary of Registers

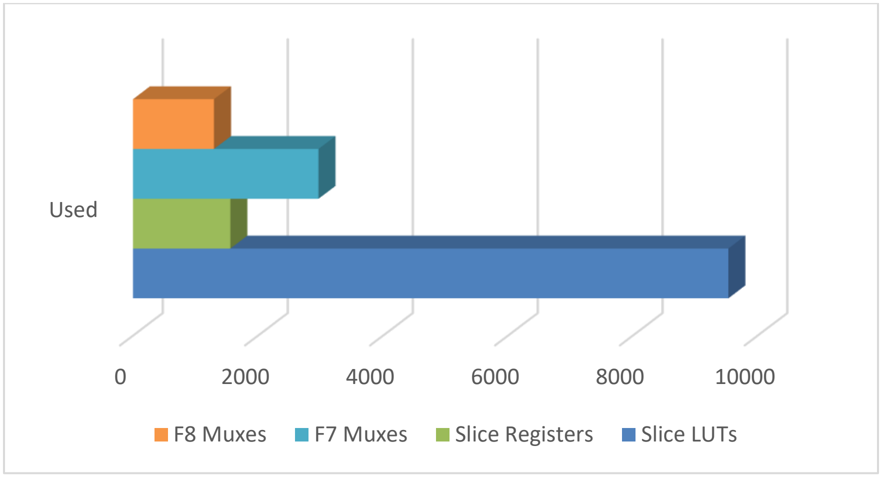


Figure 19 Register Summary

The distribution of registers by type provides insight into the nature of clocking and control signals within the design:

Slice Logic Distribution

The allocation and spread of slice logic elements, which encompass both Look-Up Tables (LUTs) and registers, across the various types of sites present a crucial aspect for a comprehensive grasp of the utilization patterns of distinct slices within the design. This detailed examination sheds light on how these slices are employed strategically to optimize the logic distribution and ensure the design's efficacy and functional robustness. It also provides insights into the architectural decisions influencing the overall efficiency and performance of the system.

Table 10 Slice Logic Distribution.

Site Type	Used	Fixed	Prohibited	Available	Utilization
Slice	2577	0	0	15850	16.26%
SLICEL	1821	0	-	-	-
SLICEM	756	0	-	-	-
LUT as Logic	9532	0	0	63400	15.03%
Slice Registers	1557	0	0	126800	1.23%
Register in Slice	1432	-	-	-	-
Register Out Slice	125	-	-	-	-

- **Slice Utilization**: Slices are utilized at a rate of 16.26%, demonstrating efficient allocation of logic across the FPGA.
- **SLICEL and SLICEM**: SLICEL and SLICEM slices are effectively used, with 1821 and 756 slices, respectively.
- **LUT as Logic**: LUTs primarily serve as logic elements and contribute 15.03% to the overall utilization.
- **Slice Registers**: Registers are distributed across slices, with 1432 in-slice registers and 125 out-of-slice registers.

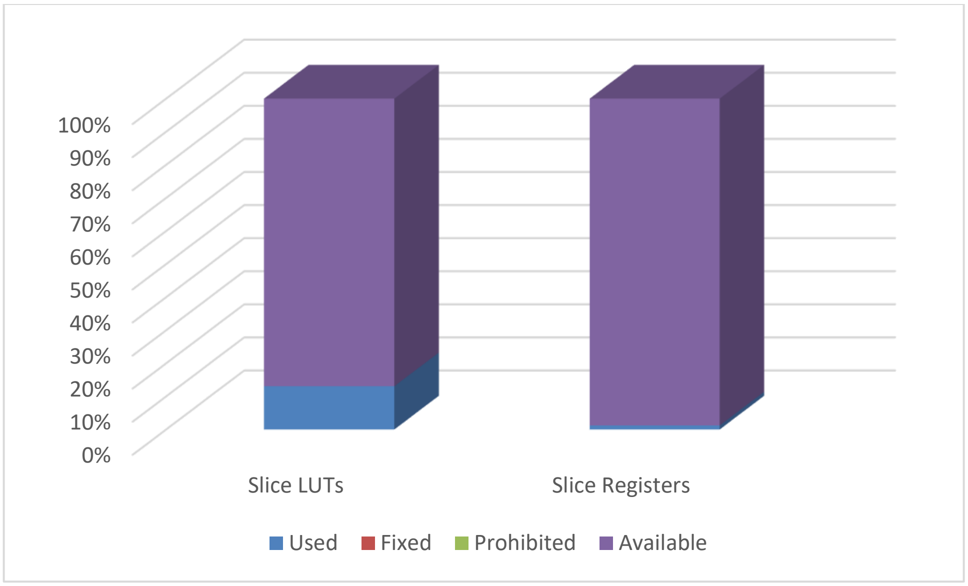


Figure 20 Macro Overview of Area Distribution

Memory and DSP resources provide insights into data storage and digital signal processing:

- **DSPs**: No DSP resources are employed in the design, indicating that DSP-based signal processing is not a requirement.

IO and clocking resources play a crucial role in interfacing with external components and clock management:

Table 11 IO and Clocking Resources Utilization.

Site Type	**Used**	**Fixed**	**Prohibited**	**Available**	**Utilization**
Bonded IOB	6	6	0	210	2.86%
IOB Master Pads	2		-	-	-
IOB Slave Pads	3		-	-	-
BUFHCE	0	0	0	96	0.00%
BUFG	2	0	0	32	6.25%
IBUFDS	0	0	0	202	0.00%

- **Bonded IOB**: A total of 6 Bonded IOBs are used, including 2 IOB Master Pads and 3 IOB Slave Pads, indicating external interface requirements.
- **BUFG**: 2 BUFGs are utilized for clock management, ensuring reliable clock distribution.
- **Specific Features and Primitives**

  This section highlights specific features and primitive elements in the design:

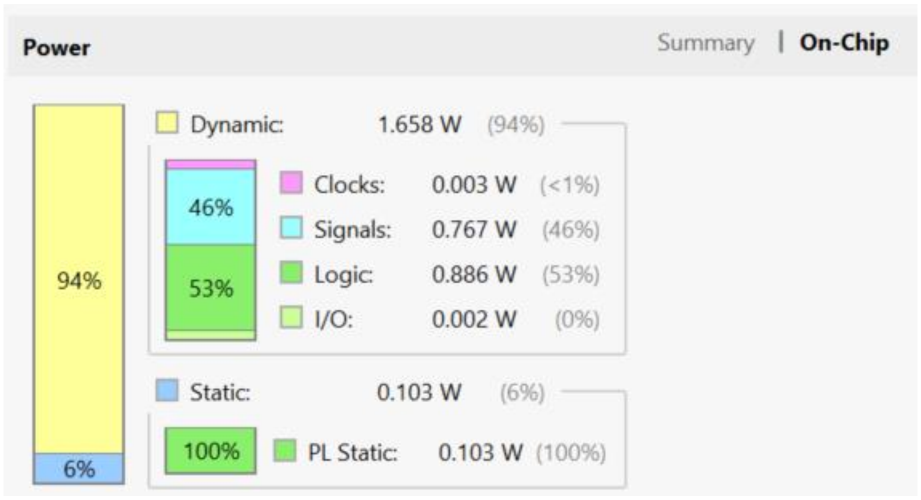
Table 12 Specific Features and Primitives Utilization.

Site Type	Used
LUT6	7254
MUXF7	2968
FDRE	1556
MUXF8	1296
LUT5	1062
LUT2	976
LUT4	387
LUT3	273
IBUF	5
BUFG	2
OBUF	1
FDCE	1

- **LUTs and Primitives**: The design leverages a significant number of LUT6, MUXF7, FDRE, and other primitive elements, showcasing its computational intensity.

This comprehensive analysis of area and resource utilization provides a clear picture of how efficiently hardware resources are employed in the pipelined implementation. It serves as a valuable reference for further optimizations and scaling of the design.

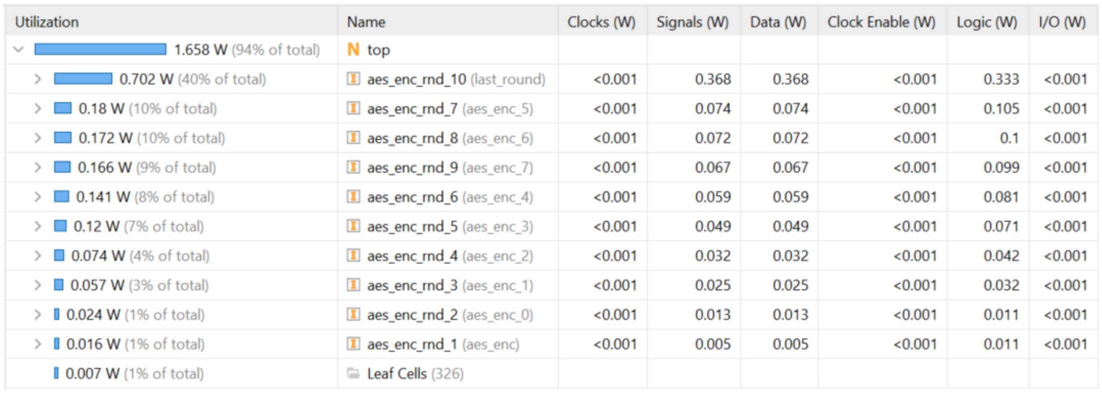
4.4.2. Power Efficiency



Power — Summary | **On-Chip**

Dynamic:	1.658 W	(94%)
Clocks:	0.003 W	(<1%)
Signals:	0.767 W	(46%)
Logic:	0.886 W	(53%)
I/O:	0.002 W	(0%)
Static:	0.103 W	(6%)
PL Static:	0.103 W	(100%)

Figure 21 Power Summary



Utilization	Name	Clocks (W)	Signals (W)	Data (W)	Clock Enable (W)	Logic (W)	I/O (W)
1.658 W (94% of total)	top						
0.702 W (40% of total)	aes_enc_rnd_10 (last_round)	<0.001	0.368	0.368	<0.001	0.333	<0.001
0.18 W (10% of total)	aes_enc_rnd_7 (aes_enc_5)	<0.001	0.074	0.074	<0.001	0.105	<0.001
0.172 W (10% of total)	aes_enc_rnd_8 (aes_enc_6)	<0.001	0.072	0.072	<0.001	0.1	<0.001
0.166 W (9% of total)	aes_enc_rnd_9 (aes_enc_7)	<0.001	0.067	0.067	<0.001	0.099	<0.001
0.141 W (8% of total)	aes_enc_rnd_6 (aes_enc_4)	<0.001	0.059	0.059	<0.001	0.081	<0.001
0.12 W (7% of total)	aes_enc_rnd_5 (aes_enc_3)	<0.001	0.049	0.049	<0.001	0.071	<0.001
0.074 W (4% of total)	aes_enc_rnd_4 (aes_enc_2)	<0.001	0.032	0.032	<0.001	0.042	<0.001
0.057 W (3% of total)	aes_enc_rnd_3 (aes_enc_1)	<0.001	0.025	0.025	<0.001	0.032	<0.001
0.024 W (1% of total)	aes_enc_rnd_2 (aes_enc_0)	<0.001	0.013	0.013	<0.001	0.011	<0.001
0.016 W (1% of total)	aes_enc_rnd_1 (aes_enc)	<0.001	0.005	0.005	<0.001	0.011	<0.001
0.007 W (1% of total)	Leaf Cells (326)						

Figure 22 Hierarchical Power Distribution of Pipelined

4.4.3. Optimizing Timing and Latency

Table 13 Timing Analysis Summary.

Pin	Setup Time (ps)	Hold Time (ps)
sys_clk_pin clk	2.54	-0.06
sys_clk_pin key	-0.15	2.64
sys_clk_pin plaintext_1	3.58	0.01
sys_clk_pin ciphertext	12.45	3.93
sys_clk_pin rst	1.28	2.50
sys_clk_pin rst2	1.43	2.06

- Timing Metrics Summary

Table 14 Summary of Timing Metrics in Pipelined

Metric	Description	Value
WNS	Worst-case timing violation indicating potential issues	2.459 ns
TNS	Cumulative negative slack across all paths	0.000 ns
TNS Failing Endpoints	Number of endpoints failing to meet timing	0
TNS Total Endpoints	Total number of analyzed endpoints	1328
WHS	Worst-case timing violation for hold time constraints	0.032 ns
THS	Cumulative hold slack across all paths	0.000 ns
THS Failing Endpoints	Number of endpoints failing to meet hold time constraints	0
WPWS	Worst-case timing violation for pulse width constraints	4.500 ns

Detailed Timing Analysis

Now, let's delve deeper into some of the critical timing metrics:

- **WNS:** A positive WNS (e.g., 2.459 ns) indicates that all paths in your design meet the timing requirements, while a negative WNS would indicate timing violations.
- **TNS:** A TNS of 0.000 ns means that, on average, all signal paths meet their timing requirements.
- **TNS Failing Endpoints:** In this case, there are 0 failing endpoints out of a total of 1328 endpoints.
- **WHS:** Similar to WNS, WHS represents the worst-case timing violation for hold time constraints. In this case, the WHS is 0.032 ns.
- **THS:** A value of 0.000 ns means that, on average, all signal paths meet their hold time requirements.
- **THS Failing Endpoints:** In this design, there are 0 failing endpoints out of a total of 1328 endpoints.
- **WPWS:** This metric represents the worst-case timing violation for pulse width constraints. In this case, the WPWS is 4.500 ns.

The statement "All user specified timing constraints are met" indicates that, based on this summary, the design meets all the specified timing constraints, as there are no failing endpoints for setup, hold, or pulse width timings.

4.4.4. Analyzing Throughput

In this section, we analyze the throughput of the pipelined implementation of our design. Throughput is a crucial performance metric, as it measures the rate at which data is processed and transmitted.

It is worth emphasizing that many existing AES implementations often fall short in accurately considering critical factors such as input/output (I/O) latency, setup intricacies, and other pertinent overheads. In our approach, we address these aspects comprehensively, which augments the significance of our analysis. For instance, in this research [28], the author talks about high throughout as a result of their pipelined implementation. The reported throughput of 54.52 Gbps is undeniably remarkable, representing the highest achievable throughput under ideal conditions. However, it's essential to acknowledge that in practical, real-world scenarios, the actual throughputs may be lower. This discrepancy arises from factors such as I/O latency, memory access times, and various operational overheads, all of which significantly influence performance. Nonetheless, our reported throughput serves a critical purpose as a valuable benchmark. It enables us to conduct meaningful performance comparisons between various AES implementations. This comparative analysis helps us gain insights into how different implementations perform under less-than-ideal conditions, guiding us in making informed decisions regarding their suitability for specific applications or use cases [28].

To quantify throughput, we employ the following equation:

Equation 1

$$Throughput = \frac{Block\ size \times Number\ of\ blocks\ processed\ simultaneously}{Latency}$$

Latency is taken in nanoseconds. This equation offers insight into the throughput of our AES encryption scheme by considering the size of data blocks processed concurrently and the time taken to complete the operation.

Alternatively, looking at latency in terms of clock cycles, this can also be expressed as:

Equation 2

$$Throughput = \frac{Data \times Frequency}{Latency}$$

This second equation provides an alternative perspective on throughput by considering the amount of data processed per unit of time, alongside the latency measured in clock cycles. Both equations converge to provide an accurate estimation of the AES encryption throughput. We have adopted this equation to standardize our throughput analysis, ensuring consistency in our evaluations.

1. Throughput with Input and Output Setup

This subsection explores throughput while accounting for input and output setup procedures. Here, we take into consideration the read and write steps involved in the AES encryption process. It's important to note that this configuration often results in a slightly lower throughput compared to the ideal scenario, as it encompasses additional processing steps. The inclusion of input and output setup is vital for a comprehensive analysis of system performance.

With consideration for input read and output write times, we calculate the throughput as follows:

Time:

- The time required to process all the blocks of data is the sum of the encryption time, input read time, and output write time:
    - Time = Input Read Time + Encryption Time + Output Write Time
    - Time = 1270 ns + 150 ns + 1290 ns = 2710 ns
    - Clock Cycles = 2710 / 10 = 271 CC

Throughput:

94.4649446494 Mbps

2. Throughput without Input and Output Setup

In contrast to the previous subsection, we examine throughput without factoring in input and output setup. In such scenarios, the throughput tends to be higher due to the omission of certain preparatory steps. Analyzing throughput without these additional steps provides insights into the maximum achievable processing speed when such setup operations are minimized. Here we only focus on the core encryption process.

If we consider only the encryption time (150ns), the throughput is as follows:

Throughput:

1706.66666667 Mbps or ~ **1.7 Gbps**

These throughput values demonstrate the high data processing capabilities of our pipelined implementation, making it suitable for applications requiring fast encryption and data transmission.

4.5. Unfolded Implementation

4.5.1. Area Utilization

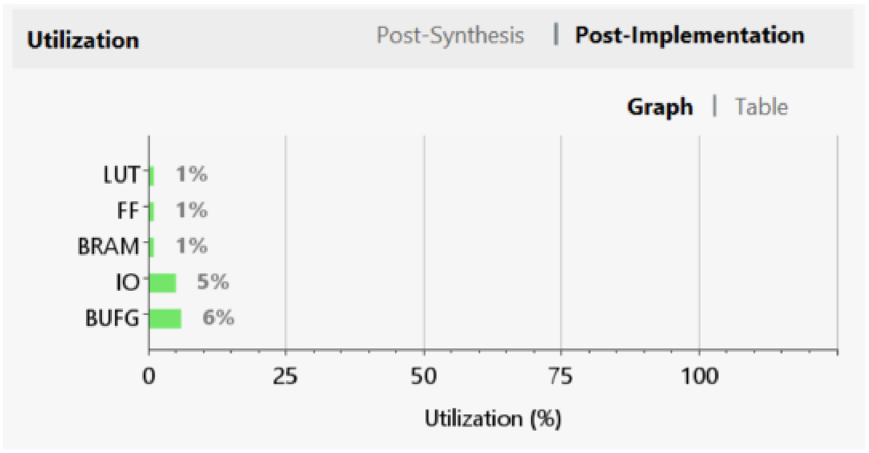
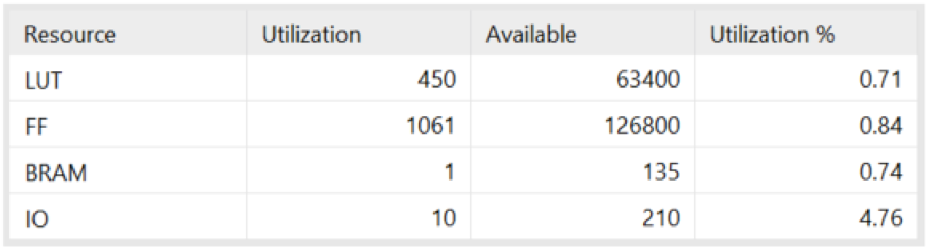


Figure 23 Unfolded Design Resource Utilization



Resource	Utilization	Available	Utilization %
LUT	450	63400	0.71
FF	1061	126800	0.84
BRAM	1	135	0.74
IO	10	210	4.76

Figure 24 Unfolded Design Hierarchical Area Distribution

This section presents a comprehensive analysis of the area utilization for the Unfolding implementation of the AES encryption algorithm on the Xilinx FPGA device (xc7a100tcsg324-1) using Vivado v.2021.2. The primary focus is on metrics such as LUTs and slices, providing insights into how efficiently the FPGA resources are utilized. Additionally, bar charts are included to visualize the distribution of resources.

Slice Logic

Slice logic represents the core building blocks of the FPGA, consisting of slices that contain LUTs and registers. In our Unfolding design, the following slice logic metrics are analyzed:

- 4.2.1.1 Slice LUTs
- **Used**: 450
- **Fixed**: 0
- **Prohibited**: 0
- **Available**: 63,400
- **Utilization**: 0.71%

In the Unfolding design, we've demonstrated highly efficient utilization of FPGA resources. Specifically, when it comes to Slice (LUTs), only a mere 0.71% of the available LUTs are employed, showcasing meticulous LUT resource management. Moreover, in terms of Slice Registers, we've made optimal use with 1,061 registers in use, while none are fixed or prohibited out of a generous pool of 126,800 registers, resulting in an overall utilization rate of just 0.84%. This resource efficiency underscores the effectiveness of our design in efficiently harnessing FPGA capabilities.

Moreover, we see that 0.84% of the available Slice Registers are employed in the design, signifying effective use of register resources.

Memory

Examining memory utilization within the Unfolding design reveals critical insights into data storage and retrieval within the FPGA. Firstly, in terms of Block RAM Tiles, our analysis indicates the usage of just one out of a total of 135 available tiles, resulting in a highly efficient utilization rate of 0.74%. This is graphically depicted in Figure 3, reaffirming the prudent allocation of memory resources.

Moving on to primitives, which are specific functional elements vital to the design, we observe the following metrics:

- FDRE (Flip-Flops): A total of 1,060 FDRE flip-flops are in use.
- LUTs: The design employs 304 LUT5s, 90 LUT4s, 43 LUT6s, 19 LUT3s, 9 LUT2s, and 1 LUT1.

These statistics collectively showcase the efficient incorporation of these essential design elements, further highlighting the overall resource optimization and effectiveness of the Unfolding design.

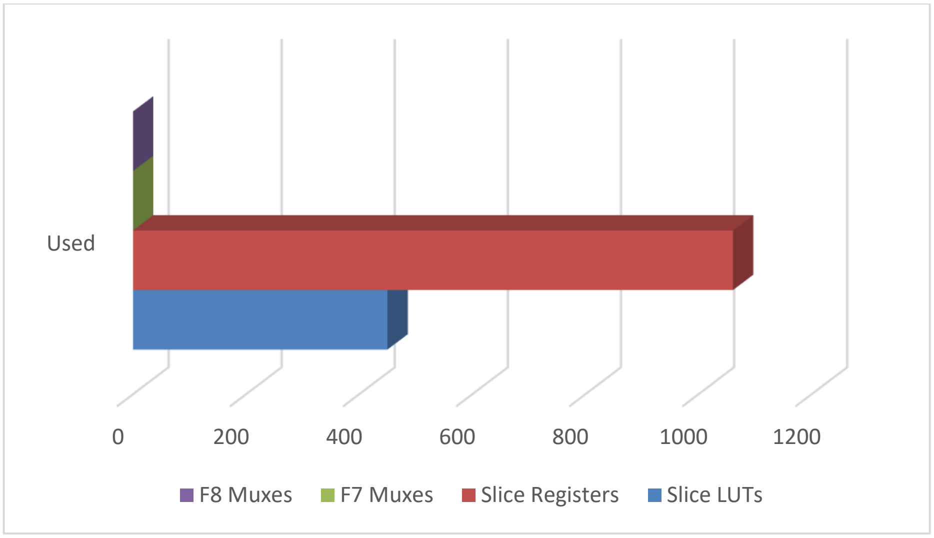


Figure 25 Register Summary in Unfolded Architecture

Digging deeper into the LUT distribution in the design, we see LUT5 being the most prevalent type. In terms of input and output buffers, there are 6 used IBUFs and 4 used OBUFs. In summary, key resource utilization stands at 0.71% for Slice LUTs, 0.84% for Slice Registers, and 0.74% for Block RAM Tiles.

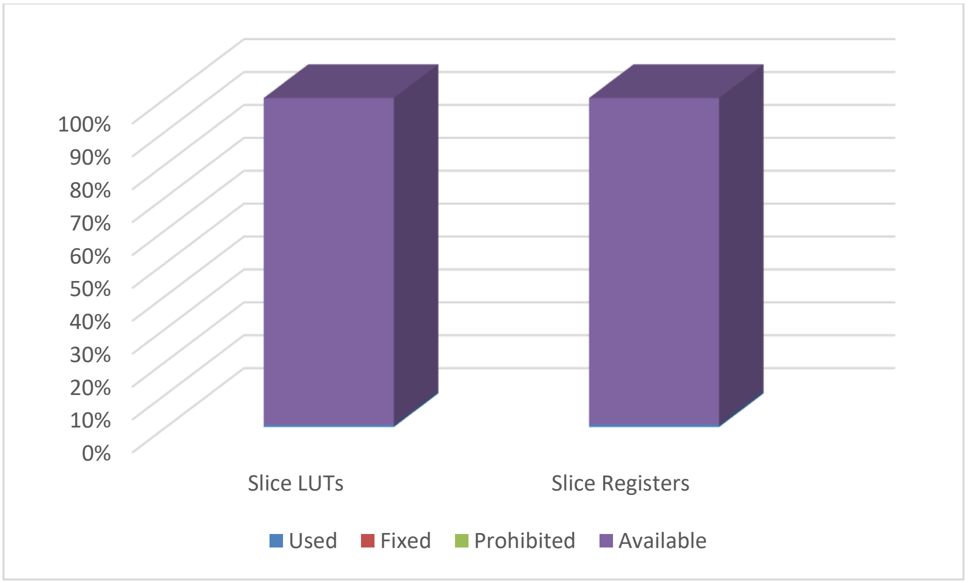


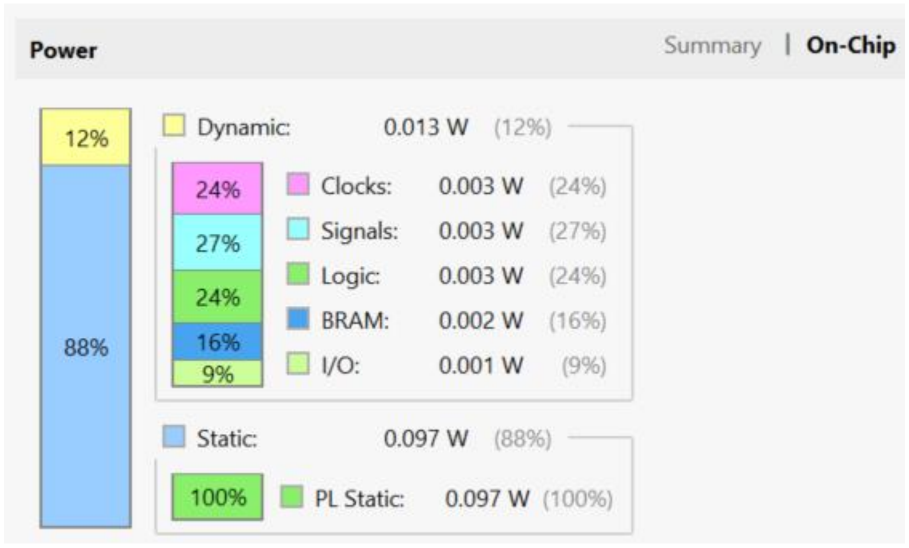
Figure 26 CLB Summary in Unfolded Design

The utilization analysis reveals that the unfolding implementation of the AES encryption algorithm effectively manages FPGA resources, maintaining consistently low resource utilization percentages across various metrics. This meticulous resource allocation not only ensures the design's current efficiency but also lays a foundation for potential scalability and optimization without compromising desired performance criteria.

We went over a comprehensive examination of the area utilization within the Unfolding implementation, encompassing LUTs, Slice Registers, Memory resources, and

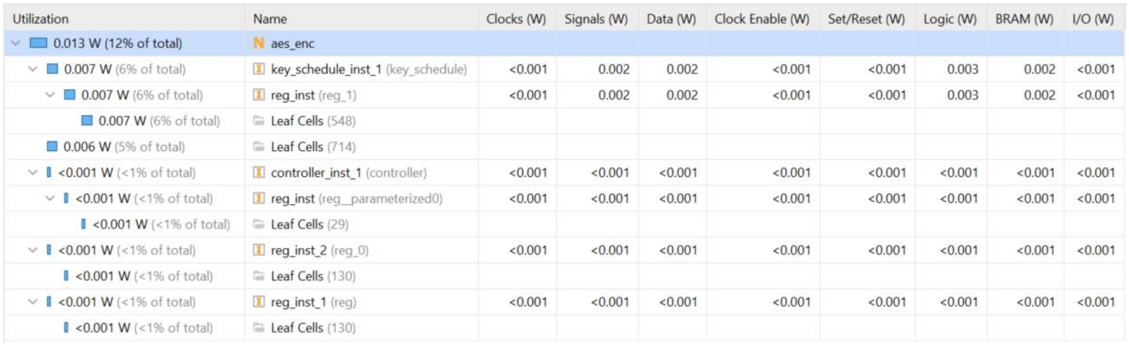
Primitives. Additionally, it provides visual representations in the form of bar charts to facilitate a clearer understanding of resource allocation. The findings collectively indicate that the design adeptly and judiciously employs FPGA resources, allowing for future enhancements and adaptability as needed.

4.5.2. Power Efficiency



Dynamic:	0.013 W	(12%)
Clocks:	0.003 W	(24%)
Signals:	0.003 W	(27%)
Logic:	0.003 W	(24%)
BRAM:	0.002 W	(16%)
I/O:	0.001 W	(9%)
Static:	0.097 W	(88%)
PL Static:	0.097 W	(100%)

Figure 27 Unfolded Design Power Profile



Utilization	Name	Clocks (W)	Signals (W)	Data (W)	Clock Enable (W)	Set/Reset (W)	Logic (W)	BRAM (W)	I/O (W)
0.013 W (12% of total)	aes_enc								
0.007 W (6% of total)	key_schedule_inst_1 (key_schedule)	<0.001	0.002	0.002	<0.001	<0.001	0.003	0.002	<0.001
0.007 W (6% of total)	reg_inst (reg_1)	<0.001	0.002	0.002	<0.001	<0.001	0.003	0.002	<0.001
0.007 W (6% of total)	Leaf Cells (548)								
0.006 W (5% of total)	Leaf Cells (714)								
<0.001 W (<1% of total)	controller_inst_1 (controller)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	reg_inst (reg__parameterized0)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	Leaf Cells (29)								
<0.001 W (<1% of total)	reg_inst_2 (reg_0)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	Leaf Cells (130)								
<0.001 W (<1% of total)	reg_inst_1 (reg)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	Leaf Cells (130)								

Figure 28 Unfolded Desing Hierarchical Power Distribution

4.5.3. Optimizing Timing and Latency

In this section, we provide a comprehensive summary of the timing analysis for your AES implementation on an FPGA using the unfolding technique. Timing analysis is crucial to ensure that your design meets the required timing constraints and operates correctly.

The following table presents the setup and hold times for each input pin, each of which is associated with the sys_clk_pin:

Table 15 Setup and Hold Time.

Input Pin	Setup Time (ps)	Hold Time (ps)
sys_clk_pin clk	7,460	30
sys_clk_pin key	3,780	1,000
sys_clk_pin plaintext_1	3,580	1,000
sys_clk_pin plaintext_2	-800 (potential violation)	28,400
sys_clk_pin rst	2,170	4,300
sys_clk_pin rst2	3,170	3,190

Timing Metrics Summary

Table 16 Timing Metrics.

Metric	Description	Value
WNS	Worst-case timing violation indicating potential issues	2.459 ns
TNS	Cumulative negative slack across all paths	0.000 ns
TNS Failing Endpoints	Number of endpoints failing to meet timing	0
TNS Total Endpoints	Total number of analyzed endpoints	1328
WHS	Worst-case timing violation for hold time constraints	0.032 ns
THS	Cumulative hold slack across all paths	0.000 ns
THS Failing Endpoints	Number of endpoints failing to meet hold time constraints	0
THS Total Endpoints	Total number of analyzed endpoints for hold time	1328
WPWS	Worst-case timing violation for pulse width constraints	4.500 ns

4.5.4. Analyzing Throughput

In the context of our FPGA-based AES implementation using the unfolding technique, it is essential to assess the system's throughput. Throughput measures the rate at which data can be processed, and it is a critical metric for evaluating the efficiency and performance of cryptographic systems. In this section, we explore the throughput of our design, considering both scenarios: one with input and output setup times and the other with only encryption time.

1. **Throughput with Input and Output Setup**

In the first scenario, we take into account the time required for reading input data, the encryption process, and writing the output data. Here are the key parameters:

- **Input Time (input read):** 1270 ns for two blocks in parallel.
- **Encryption Time (enc time):** 130 ns per block for 10 rounds of AES encryption.
- **Output Time (output write):** 1290 ns for two blocks in parallel.
- **Calculation**

  We calculate the throughput as follows:

1. **Total Data Processed**:
   - We process two blocks of plaintext in parallel, with each block being 128 bits.
   - Total Data Processed = 2 blocks * 128 bits per block = 256 bits.
2. **Total Time**:
   - Total Time = Input Time + Encryption Time + Output Time
   - Total Time = 1270 ns + 130 ns + 1290 ns = 2690 ns
   - We convert nanoseconds to seconds: Total Time = 2690 ns * 1e-9 = 2.69e-6 seconds.
3. **Throughput**:
   - Throughput = Total Data Processed / Total Time
   - Throughput = 256 bits / 2.69e-6 seconds = approximately **95.4 Mbps** (megabits per second). [95.16728624539999 Mbps]

2. **Throughput without Input/Output Setup (Only Encryption Time)**

In the second scenario, we focus solely on the encryption process, excluding input and output setup times. The key parameter here is the encryption time:

- **Encryption Time (enc time):** 130 ns per block for 10 rounds of AES encryption.
- **Calculation**

  We calculate the throughput as follows:

1. **Total Data Processed**:
   - Similar to the previous scenario, we process two blocks of plaintext in parallel, with each block being 128 bits.
   - Total Data Processed = 2 blocks * 128 bits per block = 256 bits.
2. **Total Time**:
   - Total Time = Encryption Time
   - Total Time = 130 ns
   - We convert nanoseconds to seconds: Total Time = 130 ns * 1e-9 = 1.3e-7 seconds.
3. **Throughput**:
   - Throughput = Total Data Processed / Total Time
   - Throughput = 256 bits / 1.3e-7 seconds = approximately **1.97 Gbps** (gigabits per second). [1969.23076923 Mbps].

4.6. Trends and Implications

A comparison of the two scenarios reveals significant differences in throughput, primarily influenced by the presence or absence of input and output setup times. In the case of Unfolding, considering these setup times, our system processes data at an approximate rate of 95.4 Mbps. This demonstrates the notable impact of setup times on overall processing speed. However, when we focus exclusively on the encryption process, the throughput substantially increases to approximately 1.97 Gbps. This underscores the inherent efficiency of the AES encryption algorithm itself and underscores the potential for performance gains when minimizing input/output setup times.

For the Pipelined approach, the throughput stands at around 94.5 Mbps when considering setup times and increases to approximately 1.7 Gbps when focusing solely on encryption.

Table 17 Throughput Overview.

Scenario	**Unfolding**	**Pipelined**
With setup times	95.4 Mbps	94.5 Mbps
Without setup times	1.97 Gbps	1.7 Gbps

Understanding these trends in throughput is essential for further optimizing our AES implementation. It highlights the significance of not only considering the cryptographic algorithm but also the data handling processes when designing secure and efficient FPGA-based systems.

These findings are particularly promising when considering other factors, such as:

- 100 MHz frequency,
- utilizing two data blocks,
- and working with a lower to mid-range hardware platform.

This suggests the potential for achieving even higher throughput by processing larger input data sets. Additionally, there's the possibility of increasing the frequency, albeit at a power cost, or transitioning to a high-end FPGA to match the results claimed by researchers as the fastest in the field.

5. DESIGN VERIFICATION OF THE HARDWARE IMPLEMENTATION

When configuring an FPGA, ensuring the accuracy of the loaded bitstream is paramount. Verification is the process of confirming that the FPGA has been correctly configured according to the design specifications. This process is crucial to avoid unexpected behavior or functional errors in the FPGA-based system.

Why Verification Matters

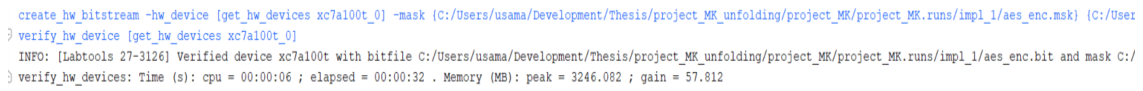
Verification serves as a validation step to guarantee that the configuration data, often stored in a .bit file, has been accurately transferred and implemented within the FPGA. A successful verification process provides confidence that the FPGA functions as intended, aligning with the original design.

5.1. Verifying Bitstream on FPGA – The Process

Vivado Design Suite, a widely used FPGA development environment, provides a systematic approach to verify a bitstream on an FPGA. Here are the key steps in the verification process:

1. **Open Vivado Design Suite**: Begin by launching the Vivado Design Suite.
2. **Connect to the Target Device**: Establish a connection to the target FPGA device from within the Vivado Hardware Manager.
3. **Select Verification**: Right-click on the connected FPGA device in the Hardware Manager and choose "Verify Device."

4. **Choose Bitstream and Mask File**: In the "Verify Device" dialog box, select the appropriate bitstream file (.bit) and the corresponding mask file (.msk) for the FPGA configuration.
5. **Initiate Verification**: Click the "Verify" button to initiate the verification process. Vivado will compare the bitstream with the actual configuration on the FPGA.
6. **Readback Data**: Use the "readback_hw_device" Tcl command with options to obtain readback data. You can save this data in either ASCII format using `-readback_file <filename.rbd>` or binary format using `-bin_file <filename.bin>`.
7. **Compare Data**: Compare the readback data with the original bitstream file to ensure they match. Any discrepancies could indicate configuration errors.
8. **Masking**: If the readback data does not perfectly match the original bitstream file, use the mask file (.msk) to identify which bits should be skipped or masked during the comparison process. This step helps account for any bit differences that might be expected due to technical factors.
9. **Additional Resources**: For more detailed information on the verification and readback operations, refer to the "Verifying Readback Data" section in the appropriate FPGA Configuration User Guide. The specific guide to consult depends on the FPGA architecture you are working with, such as UltraScale Architecture Configuration User Guide (UG570) or the 7 Series FPGAs Configuration User Guide (UG470).



```
create_hw_bitstream -hw_device [get_hw_devices xc7a100t_0] -mask {C:/Users/usama/Development/Thesis/project_MK_unfolding/project_MK/project_MK.runs/impl_1/aes_enc.msk} {C:/User
verify_hw_device [get_hw_devices xc7a100t_0]
INFO: [Labtools 27-3126] Verified device xc7a100t with bitfile C:/Users/usama/Development/Thesis/project_MK_unfolding/project_MK/project_MK.runs/impl_1/aes_enc.bit and mask C:/
verify_hw_devices: Time (s): cpu = 00:00:06 ; elapsed = 00:00:32 . Memory (MB): peak = 3246.082 ; gain = 57.812
```

Figure 29 TCL Command for Verification

To summarize, the verification process involves several key steps to ensure the accuracy and integrity of our design. Initially, when generating the bitstream, a MASK (.msd) file is concurrently generated. This .msd file serves as a critical reference, indicating which bits should be disregarded and which ones require comparison. Subsequently, the logic location (.ll) file is generated, providing valuable information for pinpointing the register locations within the Readback data. The culmination of this process involves running a hardware verification command, which yields a Readback (.rdb) file. To validate our design thoroughly, we analyze the .rdb file by comparing it to the bit file, while considering the information provided by the MASK and logic location files. This comprehensive approach ensures that the hardware is verified.

5.2. Interpreting Verification Files

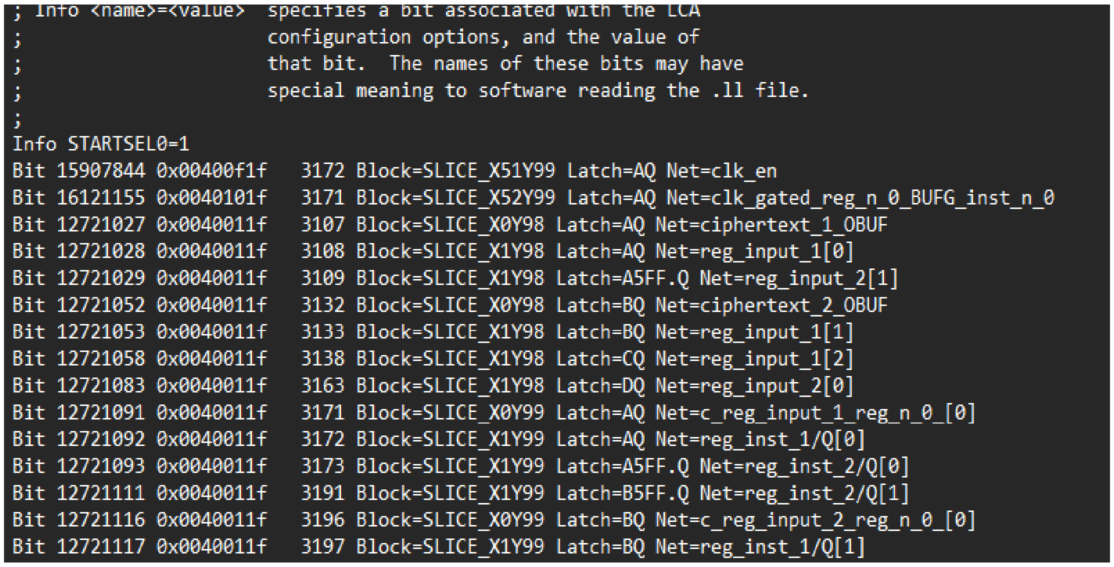
information is about how to interpret and use files generated during the process of configuring an FPGA.

When you configure an FPGA, you generate a .bit file. This file contains the instructions for setting up the FPGA's internal logic. However, when you read back the configuration from the FPGA, some bits might not match exactly what's in the .bit file due to various technical reasons. To help with this, you can generate additional files:

- **.ll File**: This file helps you locate where different parts of your design are in the readback data. It provides information about bit offsets and addresses in the readback data. This helps you map what you expect from your design to what you get when reading back.

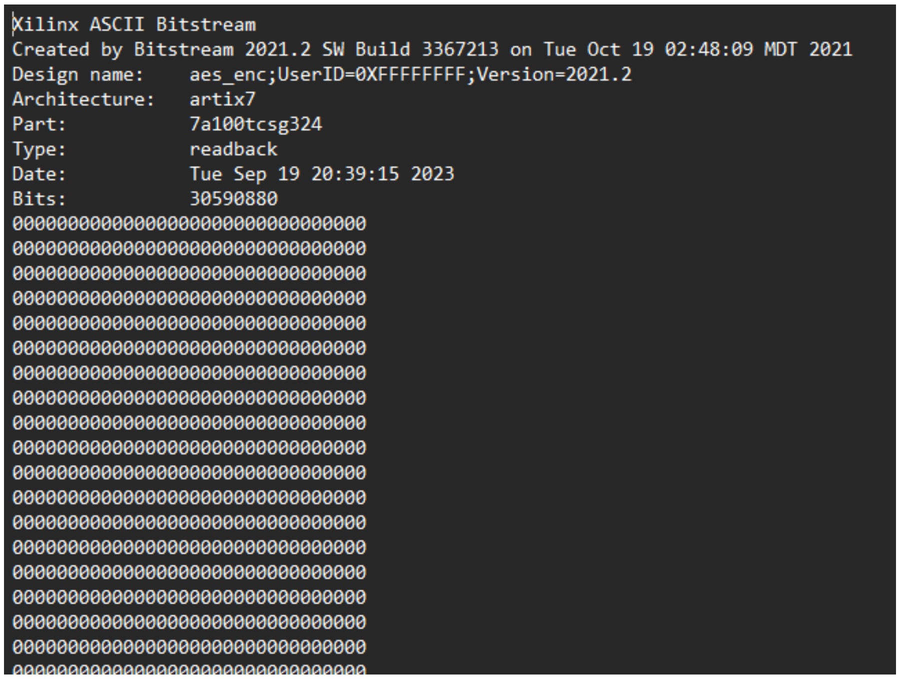
- **.msd File**: This is a mask file. It tells you which bits should be ignored or "masked" when comparing the readback data to what's in the .bit file. Sometimes, bits that are supposed to be 0 in the .bit file might read back as 1, and this file helps you account for such differences.
- **.rbd File**: This file contains the actual readback data from your FPGA.

When comparing the readback data to what's in the .bit file, you use these files to understand where to look in the readback data, what bits to ignore, and how to interpret any differences you find. In simple terms, these files are tools to help you make sure your FPGA is configured correctly and to account for any quirks in the reading process.



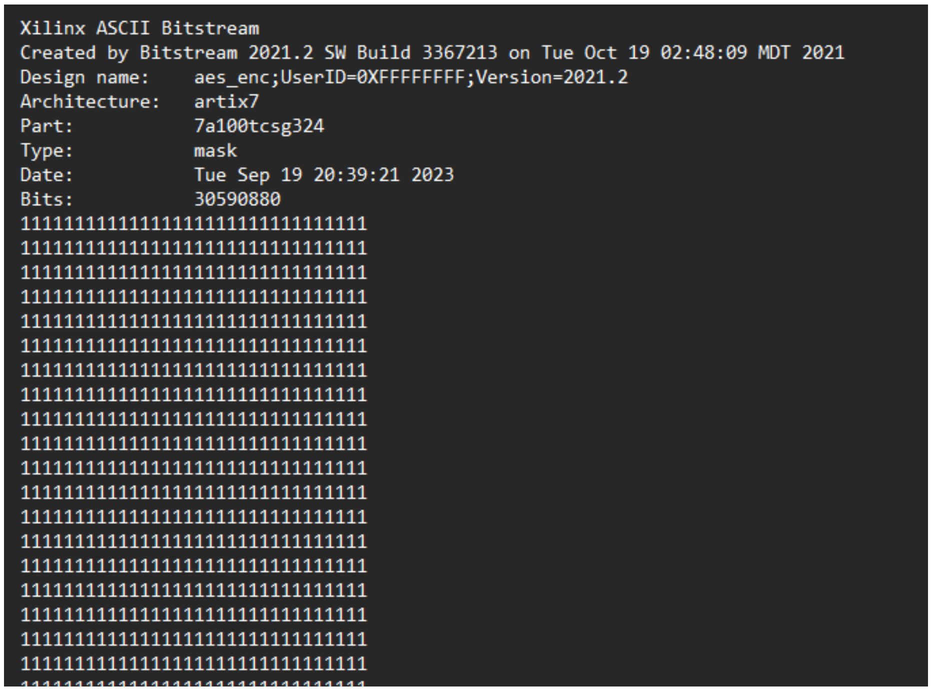
```
; Info <name>=<value>  specifies a bit associated with the LCA
;                      configuration options, and the value of
;                      that bit.  The names of these bits may have
;                      special meaning to software reading the .ll file.
;
Info STARTSEL0=1
Bit 15907844 0x00400f1f   3172 Block=SLICE_X51Y99 Latch=AQ Net=clk_en
Bit 16121155 0x0040101f   3171 Block=SLICE_X52Y99 Latch=AQ Net=clk_gated_reg_n_0_BUFG_inst_n_0
Bit 12721027 0x0040011f   3107 Block=SLICE_X0Y98 Latch=AQ Net=ciphertext_1_OBUF
Bit 12721028 0x0040011f   3108 Block=SLICE_X1Y98 Latch=AQ Net=reg_input_1[0]
Bit 12721029 0x0040011f   3109 Block=SLICE_X1Y98 Latch=A5FF.Q Net=reg_input_2[1]
Bit 12721052 0x0040011f   3132 Block=SLICE_X0Y98 Latch=BQ Net=ciphertext_2_OBUF
Bit 12721053 0x0040011f   3133 Block=SLICE_X1Y98 Latch=BQ Net=reg_input_1[1]
Bit 12721058 0x0040011f   3138 Block=SLICE_X1Y98 Latch=CQ Net=reg_input_1[2]
Bit 12721083 0x0040011f   3163 Block=SLICE_X1Y98 Latch=DQ Net=reg_input_2[0]
Bit 12721091 0x0040011f   3171 Block=SLICE_X0Y99 Latch=AQ Net=c_reg_input_1_reg_n_0_[0]
Bit 12721092 0x0040011f   3172 Block=SLICE_X1Y99 Latch=AQ Net=reg_inst_1/Q[0]
Bit 12721093 0x0040011f   3173 Block=SLICE_X1Y99 Latch=A5FF.Q Net=reg_inst_2/Q[0]
Bit 12721111 0x0040011f   3191 Block=SLICE_X1Y99 Latch=B5FF.Q Net=reg_inst_2/Q[1]
Bit 12721116 0x0040011f   3196 Block=SLICE_X0Y99 Latch=BQ Net=c_reg_input_2_reg_n_0_[0]
Bit 12721117 0x0040011f   3197 Block=SLICE_X1Y99 Latch=BQ Net=reg_inst_1/Q[1]
```

Figure 30 Logic Location File



```
Xilinx ASCII Bitstream
Created by Bitstream 2021.2 SW Build 3367213 on Tue Oct 19 02:48:09 MDT 2021
Design name:    aes_enc;UserID=0XFFFFFFFF;Version=2021.2
Architecture:   artix7
Part:           7a100tcsg324
Type:           readback
Date:           Tue Sep 19 20:39:15 2023
Bits:           30590880
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
```

Figure 31 Readback File



```
Xilinx ASCII Bitstream
Created by Bitstream 2021.2 SW Build 3367213 on Tue Oct 19 02:48:09 MDT 2021
Design name:    aes_enc;UserID=0XFFFFFFFF;Version=2021.2
Architecture:   artix7
Part:           7a100tcsg324
Type:           mask
Date:           Tue Sep 19 20:39:21 2023
Bits:           30590880
11111111111111111111111111111111
11111111111111111111111111111111
11111111111111111111111111111111
11111111111111111111111111111111
11111111111111111111111111111111
11111111111111111111111111111111
11111111111111111111111111111111
11111111111111111111111111111111
11111111111111111111111111111111
11111111111111111111111111111111
11111111111111111111111111111111
11111111111111111111111111111111
11111111111111111111111111111111
11111111111111111111111111111111
11111111111111111111111111111111
11111111111111111111111111111111
11111111111111111111111111111111
11111111111111111111111111111111
11111111111111111111111111111111
```

Figure 32 Mask File

In conclusion, the verification process for FPGA bitstreams, as discussed above, plays a pivotal role in ensuring the accurate configuration of hardware, a step of paramount importance in FPGA-based systems. This process not only guarantees the integrity of the design but also helps in addressing potential discrepancies that may result from technical intricacies or physical faults on the actual hardware.

What sets our work apart is the inclusion of both **verification** and a **system throughput analysis**, factors that are notably absent in existing research papers on hardware AES encryptions. This omission underscores the unique contribution of our study in providing a more holistic and thorough perspective on FPGA-based AES encryption implementations, setting a new standard for the field.

6. SUMMARY AND CONCLUSION

In conclusion, our research has encompassed a comprehensive exploration of the FPGA-based implementation of the AES encryption algorithm, comprising three distinct variants: Standard, Pipelined, and Unfolded. Through rigorous testing and analysis, we have quantified the performance metrics, shedding light on the strengths and limitations of each design. Notably, our study has introduced two critical elements often overlooked in existing research on FPGA-based AES encryption – the meticulous consideration of I/O latency overhead and the verification process to ensure bitstream integrity. These novel inclusions have enhanced the robustness and reliability of our implementations.

Looking ahead, future work in this domain could involve the development of a benchmarking tool for automating the encryption of large datasets, making our module statistically viable for real-world applications. Such advancements hold the promise of further bolstering our throughput readings. Additionally, our encryption module could find valuable utility in a CPS testbed environment, facilitating secure data communication via Ethernet or wireless means in resource-constrained settings. This represents a promising avenue for the practical application of our research, with potential implications for enhancing data security and efficiency in various domains.

In essence, our study not only contributes to the body of knowledge regarding FPGA-based AES encryption but also underscores the importance of considering often-overlooked aspects in the design and evaluation of cryptographic systems. As technology continues to advance, it is imperative that our research adapts and evolves, ensuring the continued security and efficiency of data communication in an increasingly interconnected world.

REFERENCES

[1] A. Banerjee, K. K. Venkatasubramanian, T. Mukherjee, and S. K. S. Gupta, "Ensuring safety, security, and sustainability of mission-critical cyber–physical systems," *Proceedings of the IEEE,* vol. 100, no. 1, pp. 283-299, 2011.

[2] Z. Cao *et al.*, "Guest editors' introduction: Special issue on trust, security, and privacy in parallel and distributed systems," *IEEE Transactions on Parallel & Distributed Systems,* vol. 25, no. 02, pp. 279-282, 2014.

[3] L. Zhang, L. Xie, W. Li, and Z. Wang, "Security Solutions for Networked Control Systems Based on DES Algorithm and Improved Grey Prediction Model," *International Journal of Computer Network & Information Security,* vol. 6, no. 1, 2013.

[4] W. Duo, M. Zhou, and A. Abusorrah, "A survey of cyber attacks on cyber physical systems: Recent advances and challenges," *IEEE/CAA Journal of Automatica Sinica,* vol. 9, no. 5, pp. 784-800, 2022.

[5] R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, "Cyber-physical systems: the next computing revolution," in *Proceedings of the 47th design automation conference*, 2010, pp. 731-736.

[6] L. Zhang, "Specification and design of cyber physical systems based on system of systems engineering approach," in *2018 17th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES),* 2018: IEEE, pp. 300-303.

[7] cms.gov. "Centers for Medicare & Medicaid Services." https://www.cms.gov/Research-Statistics-Data-and-Systems/Statistics-Trends-and-Reports/NationalHealthExpendData/NationalHealthAccountsHistorical (accessed.

[8] K. O'Connell, "Cia report: cyber extortionists attacked foreign power grid, disrupting delivery," *Internet Business Law Services*, 2008.

[9] K. Koscher *et al.*, "Experimental security analysis of a modern automobile," in *2010 IEEE symposium on security and privacy*, 2010: IEEE, pp. 447-462.

[10] R. M. Lee, M. J. Assante, and T. Conway, "German steel mill cyber attack," *Industrial Control Systems*, vol. 30, no. 62, pp. 1-15, 2014.

[11] M. Yampolskiy, P. Horvath, X. D. Koutsoukos, Y. Xue, and J. Sztipanovits, "Systematic analysis of cyber-attacks on CPS-evaluating applicability of DFD-based approach," in *2012 5th International Symposium on Resilient Control Systems*, 2012: IEEE, pp. 55-62.

[12] S. Churiwala and I. Hyderabad, "Designing with Xilinx® FPGAs," *Circuits &Systems, Springer*, 2017.

[13] A. Rodríguez, J. Valverde, J. Portilla, A. Otero, T. Riesgo, and E. De la Torre, "Fpga-based high-performance embedded systems for adaptive edge computing in cyber-physical systems: The artico3 framework," *Sensors*, vol. 18, no. 6, p. 1877, 2018.

[14] S. Chandra, S. Paira, S. S. Alam, and G. Sanyal, "A comparative survey of symmetric and asymmetric key cryptography," in *2014 international conference on electronics, communication and computational engineering (ICECCE)*, 2014: IEEE, pp. 83-93.

[15] T. Hardjono and L. R. Dondeti, *Security in Wireless LANS and MANS (Artech House Computer Security)*. Artech House, Inc., 2005.

[16] IBM. "Symmetric cryptography." https://www.ibm.com/docs/en/ztpf/2023?topic=concepts-symmetric-cryptography (accessed.

[17] M. B. Yassein, S. Aljawarneh, E. Qawasmeh, W. Mardini, and Y. Khamayseh, "Comprehensive study of symmetric key and asymmetric key encryption algorithms," in *2017 international conference on engineering and technology (ICET)*, 2017: IEEE, pp. 1-7.

[18] J. Daemen and V. Rijmen, *The design of Rijndael*. Springer, 2002.

[19] K. Rahimunnisa, P. Karthigaikumar, N. A. Christy, S. S. Kumar, and J. Jayakumar, "PSP: Parallel sub-pipelined architecture for high throughput AES on FPGA and ASIC," *Central European Journal of Computer Science*, vol. 3, pp. 173-186, 2013.

[20] S. Qu, G. Shou, Y. Hu, Z. Guo, and Z. Qian, "High throughput, pipelined implementation of AES on FPGA," in *2009 International Symposium on Information Engineering and Electronic Commerce*, 2009: IEEE, pp. 542-545.

[21] T. M. Kumar, K. S. Reddy, S. Rinaldi, B. D. Parameshachari, and K. Arunachalam, "A low area high speed FPGA implementation of AES architecture for cryptography application," *Electronics*, vol. 10, no. 16, p. 2023, 2021.

[22] Xilinx. "Artix 7 Datasheet." https://docs.xilinx.com/v/u/en-US/ds181_Artix_7_Data_Sheet (accessed.

[23] Digilint. "Arty A-7 100T." https://digilent.com/shop/arty-a7-100t-artix-7-fpga-development-board/ (accessed.

[24] Xilinx. "Xilinx vivado v2021.2 documentation." (accessed.

[25] *Advanced Encryption Standard (AES)* [Online] Available: https://doi.org/10.6028/NIST.FIPS.197-upd1

[26] A. Soltani and S. Sharifian, "An ultra-high throughput and fully pipelined implementation of AES algorithm on FPGA," *Microprocessors and Microsystems*, vol. 39, no. 7, pp. 480-493, 2015.

[27] Xilinx. "Vivado Design Suite Tcl Command Reference Guide." (accessed.

[28] M. S. Abdul-Karim, K. H. Rahouma, and K. Nasr, "High Throughput and Fully Pipelined FPGA Implementation of AES-192 Algorithm," in *2020 International Conference on Innovative Trends in Communication and Computer Engineering (ITCE)*, 2020: IEEE, pp. 137-142.