RISC-V Based Secure AES-128 Accelerator SoC with Feedback Architecture for Low-Power Application

Author:

Omar Ashraf Abd El Mongy

Preface

This document presents the design and implementation of an AES encryption module integrated with a RISC-V processor using the AHB-Lite protocol.

The project aims to provide a secure and efficient solution for data encryption in embedded systems. It covers the architecture of the AES module, its integration with the RISC-V core, and the verification process on an FPGA platform.

The report also highlights the challenges encountered during development and outlines future work to enhance the system's capabilities.

Table of Contents

1. Introduction

1.1 Project Overview

The **RISC-V Based Secure AES-128 Accelerator SoC with Feedback Architecture for Low-Power Application** is a custom-designed System-on-Chip (SoC) aimed at providing efficient, secure encryption for embedded systems.

The primary goal of this project is to implement a low-power, high-performance AES-128 encryption module integrated into a RISC-V processor-based system, utilizing an **AHB-Lite** interface for communication.

The AES module is optimized for area and power efficiency by employing a feedback architecture that reduces the number of rounds required for encryption, ensuring minimal resource usage.

1.2 Key Features

The key features of the project include:

- **AES-128 Encryption**: Utilization of the AES-128 encryption standard, offering a high level of security suitable for a wide range of applications.
- **RISC-V Processor Integration**: The AES encryption module is controlled by a RISC-V core, providing flexibility and extensibility in the design.
- **AHB-Lite Interface**: The SoC incorporates an AHB-Lite interface, enabling efficient communication between the AES accelerator and other components, including the RISC-V core and memory.
- **Feedback Architecture**: A feedback mechanism is employed within the AES module to reduce the area and power consumption by reusing one round of the AES encryption process.
- **Low-Power Design**: The project is optimized for low power consumption, making it suitable for battery-powered and embedded systems.

1.3 Motivation

With the growing need for secure communication in embedded systems, encryption plays a critical role in ensuring data privacy and integrity.

However, the constraints of power and area in embedded applications require careful optimization of cryptographic solutions.

The integration of AES-128 with a feedback architecture provides a balanced solution that meets security requirements while maintaining low power consumption and minimal area usage.

This project aims to address these challenges by providing a highly efficient AES encryption accelerator for low-power, embedded applications.

1.4 Applications

This SoC is well-suited for a variety of low-power, secure communication applications, including:

- **IoT Devices**: Ensuring secure data transmission in resource-constrained environments.
- **Embedded Systems**: Providing encryption capabilities for devices with limited power and area budgets.
- **Wearable Technology**: Enabling secure communication for wearable devices, where power efficiency is crucial.
- **Secure Storage**: Providing a secure encryption solution for embedded storage devices in applications requiring data protection.

1.5 Scope of the Document

This document provides a detailed overview of the design and implementation of the **RISC-V Based Secure AES-128 Accelerator SoC**.

It covers the system architecture, design considerations, and optimization techniques, including the feedback architecture and integration with the RISC-V core.

Additionally, the document discusses the testing and verification of the AES module, including simulation and hardware validation.

The overall goal is to showcase the design of a power-efficient AES encryption accelerator suitable for embedded and low-power applications.

2. System Architecture

2.1 Overview

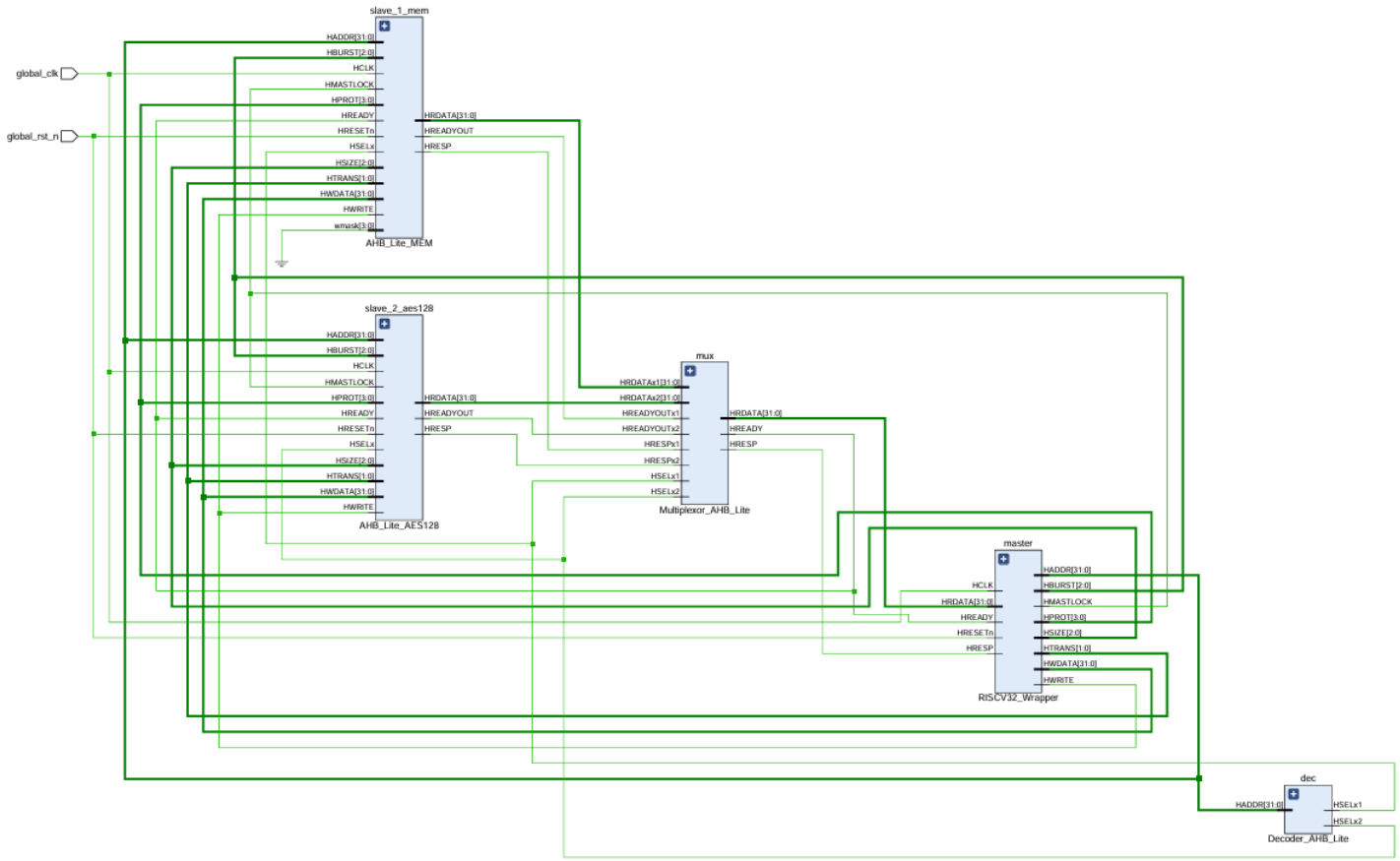
The **RISC-V Based Secure AES-128 Accelerator SoC** is designed with a modular and efficient architecture to enable secure encryption in embedded systems.

The system integrates a **RISC-V processor core** with an **AES-128 encryption accelerator** through an **AHB-Lite interface**.

This allows the RISC-V core to interact with the AES module and other peripheral components, such as memory, in a streamlined manner.

The AES accelerator employs a **feedback architecture** to reduce area and power consumption, ensuring optimal performance for low-power applications.

2.2 High-Level Block Diagram



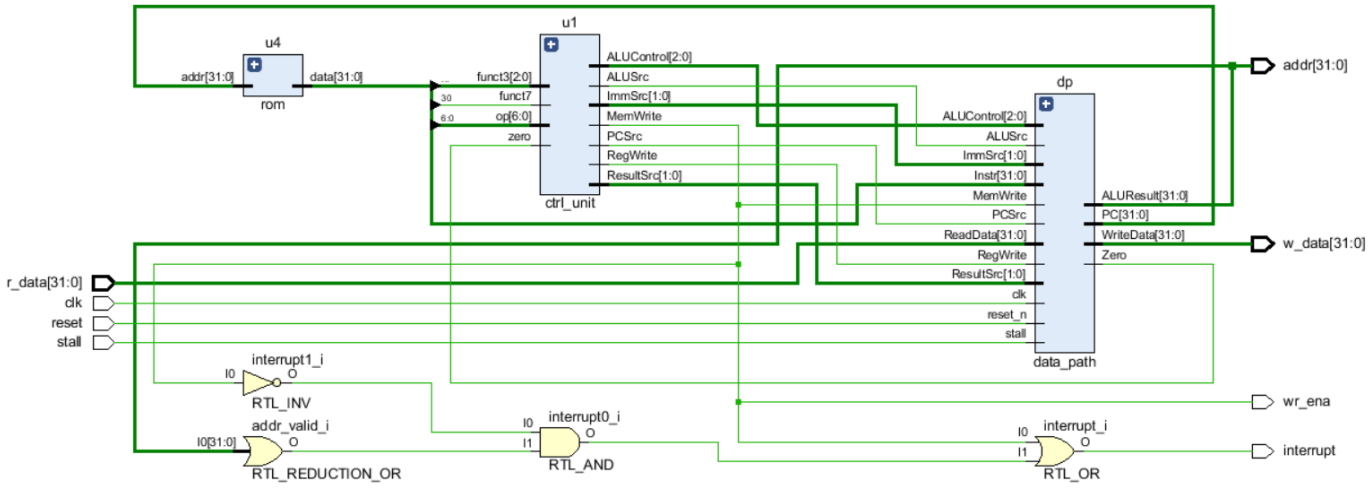
The high-level architecture of the SoC includes the following components:

- **RISC-V Core**: Serves as the master in the AHB-Lite interface, controlling the AES module and memory operations.
- **AES-128 Accelerator**: The AES encryption module is implemented as an AHB-Lite slave, responsible for performing AES-128 encryption on data provided by the RISC-V core.
- **Memory (RAM)**: Also, an AHB-Lite slave to store data and keys for encryption/decryption operations.
- **Decoder and Multiplexer (MUX)**: The decoder handles address decoding and ensures the proper routing of signals between the AHB-Lite master and slaves (AES module and memory).

2.3 Detailed Architecture Description

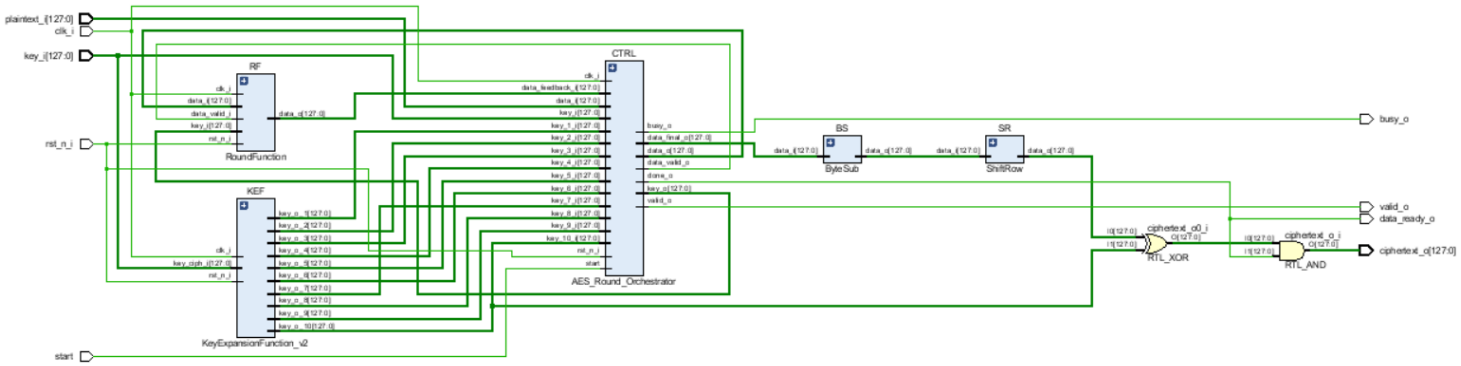
2.3.1 RISC-V Core

- The **RISC- v core** is based on the RV32I instruction set, making it compatible with a wide range of software and flexible for various applications.
- It acts as the **AHB-Lite master**, initiating communication with the AES-128 accelerator and managing memory operations.
- The core interacts with the AES module by writing plaintext and encryption keys to the AES block and triggering encryption operations.
- The core can also retrieve the encrypted data after processing.



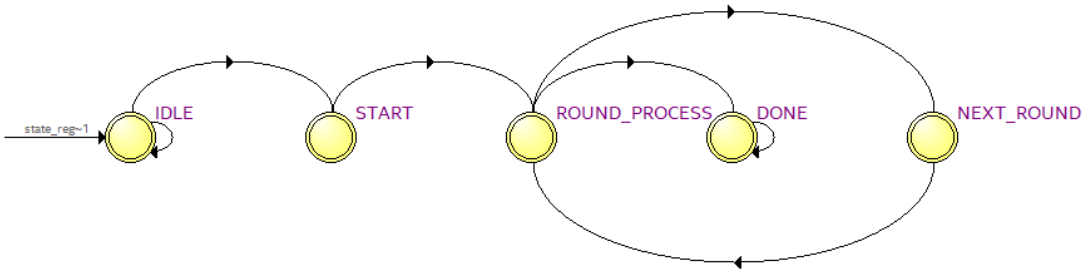
2.3.2 AES-128 Accelerator

- The **AES-128 accelerator** is the heart of the encryption module. It is responsible for carrying out the AES encryption algorithm on 128-bit data blocks.
- The AES module operates on a feedback architecture, where only one round of AES is executed repeatedly to minimize area and power.
- This approach significantly reduces the hardware complexity compared to implementing all 10 rounds of AES in a traditional structure.
- The AES module receives inputs from the RISC-V core (plaintext and encryption keys) and produces the ciphertext output after encryption.



The AES Round Orchestrator is a custom-designed controller (FSM) that facilitates the feedback mechanism in the AES-128 accelerator.

- This controller manages the sequential execution of the single AES round, ensuring that the encryption process is repeated for each block of data.
- It **orchestrates** the flow of control signals and data, enabling the AES accelerator to reuse the round results efficiently and continuously.
- By using the AES Round Orchestrator, the feedback architecture allows the AES module to achieve the necessary encryption results with minimal hardware, reducing both area and power consumption.



	Source State	Destination State	Condition
1	DONE	DONE	
2	IDLE	IDLE	(!start)
3	IDLE	START	(start)
4	NEXT_ROUND	ROUND_PROCESS	
5	ROUND_PROCESS	NEXT_ROUND	(!counter[0]),(!counter[1]) + (!counter[0]),(counter[1]),(!counter[2]),(!counter[3]) + (!counter[0]),(counter[1]),(counter[2]) + (counter[0])
6	ROUND_PROCESS	DONE	(!counter[0]),(counter[1]),(!counter[2]),(counter[3])
7	START	ROUND_PROCESS	

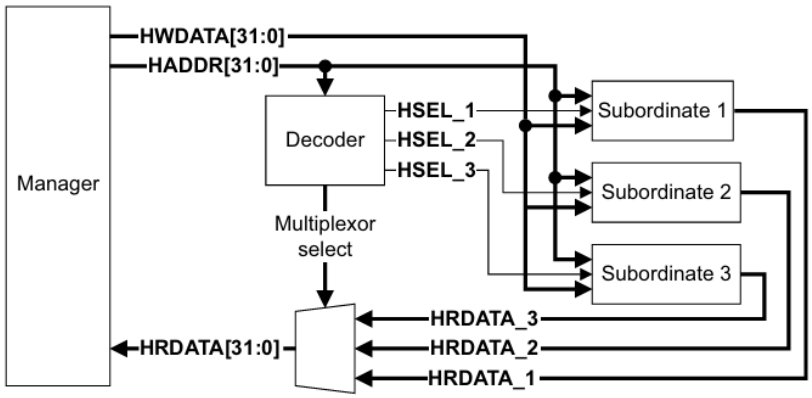
2.3.3 AHB-Lite Interface

The **AHB-Lite interface** is a simplified version of the AHB protocol, used to connect the RISC-V core, AES-128 accelerator, and memory.

This interface is optimized for lower resource usage, making it suitable for embedded and low-power systems.

The interface supports a master-slave relationship where the RISC-V core acts as the master and the AES module and memory act as slaves.

Data transfer across the interface is initiated by the RISC-V core and involves reading/writing data from/to the AES accelerator and memory.

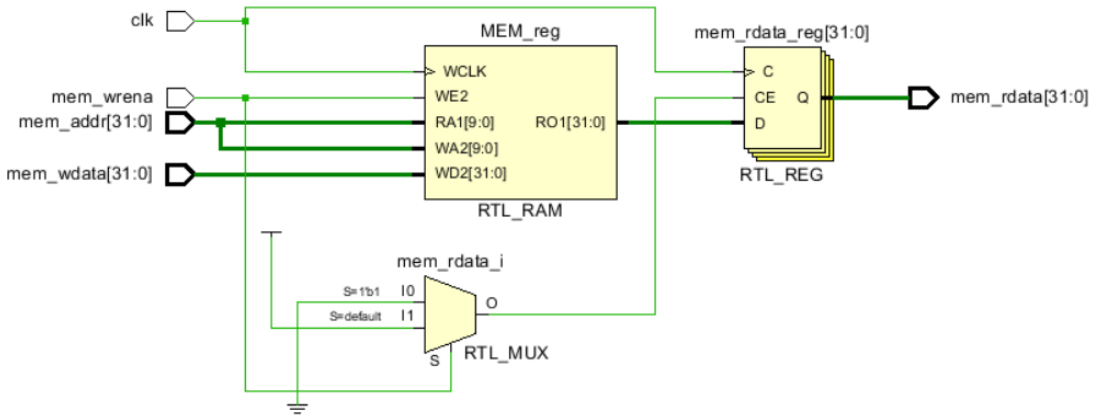


2.3.4 Memory (RAM)

Memory in this design is used to store both plaintext data and encryption keys.

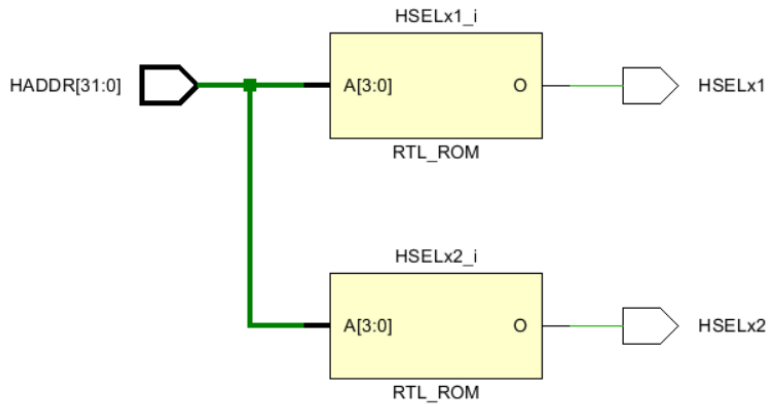
The memory block is mapped as an AHB-Lite slave, allowing the RISC-V core to read and write to it during encryption processes.

The AES module retrieves the keys and data from memory, performs encryption, and outputs the ciphertext back to memory or to the core.



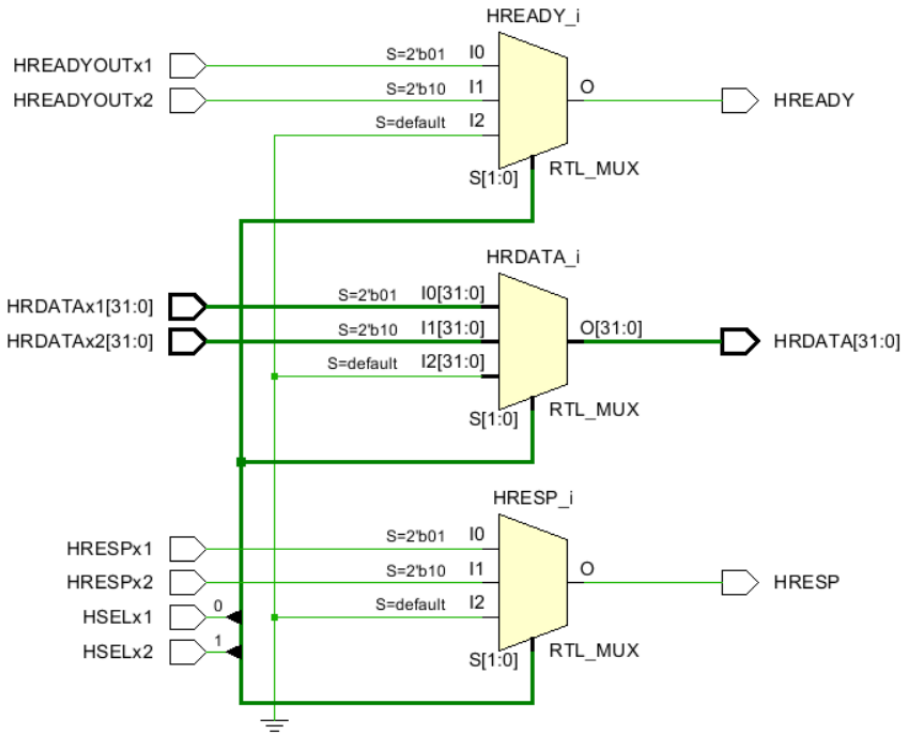
2.3.5 Decoder and Multiplexer

The **decoder** is responsible for decoding the AHB-Lite address and directing signals to the appropriate slave device (either the AES-128 accelerator or the memory).



The **multiplexer (MUX)** ensures the correct data is routed to the appropriate slave based on the AHB-Lite bus signals.

This mechanism ensures efficient and error-free data transfer between the master and the various slaves in the system.



2.4 Data Flow

The data flow within the system is managed by the RISC-V core, which initiates encryption operations by performing the following sequence:

1. The RISC-V core firstly loads the plaintext and encryption key that are initially loaded in the Data Memory (RAM)
2. The RISC-V core writes the plaintext and encryption key to the AES-128 accelerator and memory.
3. The RISC-V core triggers the AES-128 encryption operation.
4. The AES-128 accelerator performs the encryption process, outputting the ciphertext.
5. The RISC-V core reads the encrypted data from memory or the AES module for further processing or output.

This process ensures smooth and secure encryption of data while utilizing minimal system resources.

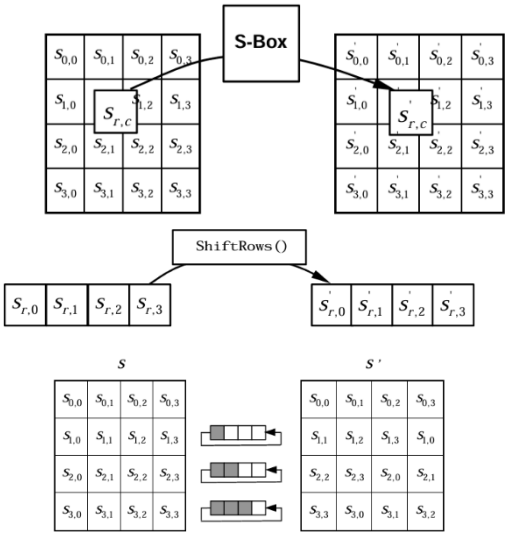
3. AES-128 Module Design

3.1 Algorithm Overview

The AES (Advanced Encryption Standard) is a symmetric encryption algorithm that operates on fixed-size blocks (128 bits) of data using a secret key of varying lengths (128, 192, or 256 bits). For AES-128, the key size is 128 bits, and the encryption process involves 10 rounds, with each round consisting of four primary operations:

SubBytes

- This operation substitutes each byte of the state (the data block being processed) using a substitution table (S-Box). The byte values are replaced according to the S-Box, which provides non-linearity to the encryption process.

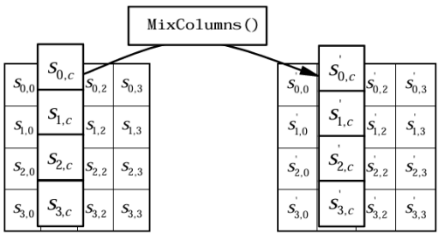


ShiftRows

- The rows of the state are shifted cyclically. The first row remains unchanged, the second row is shifted by one byte, the third row by two bytes, and the fourth row by three bytes. This provides diffusion, spreading the bits of the state across the output.

MixColumns

- This step operates on the columns of the state, mixing the data to provide further diffusion.
- Each column is treated as a polynomial and multiplied by a fixed polynomial modulo x^4 + 1. This operation increases the diffusion of the state.



AddRoundKey

- A round key is generated from the original encryption key by using a key expansion process. The round key is **XORed** with the state, adding confusion to the encryption process.

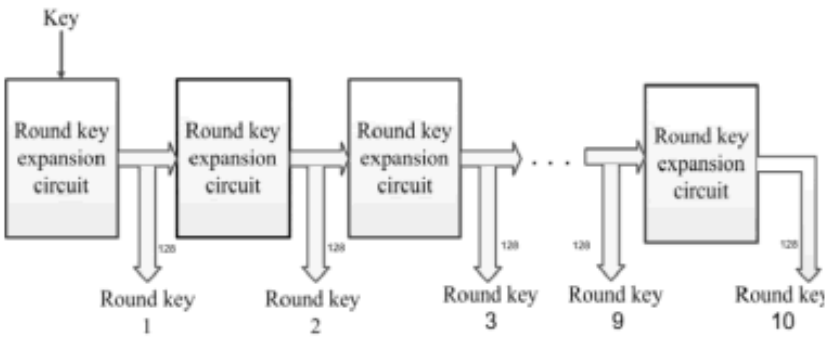
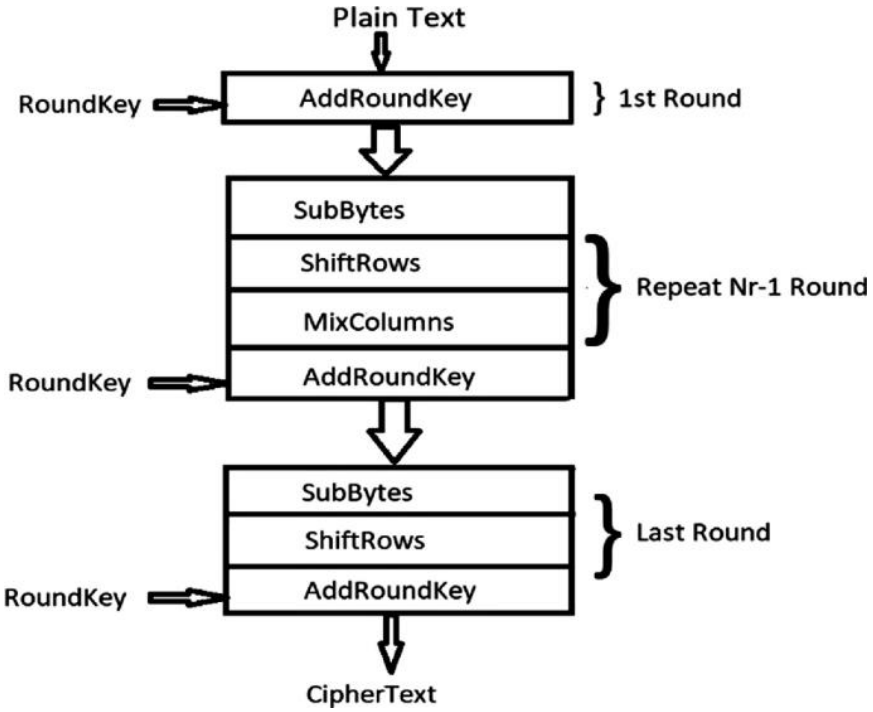


Figure 9. Key expansion module.
Source: The authors



Feedback Architecture

In this design, a feedback architecture is implemented to optimize area and power consumption. Rather than executing all 10 rounds of AES in a traditional approach, only a single round of AES is executed repeatedly to reduce the hardware complexity.

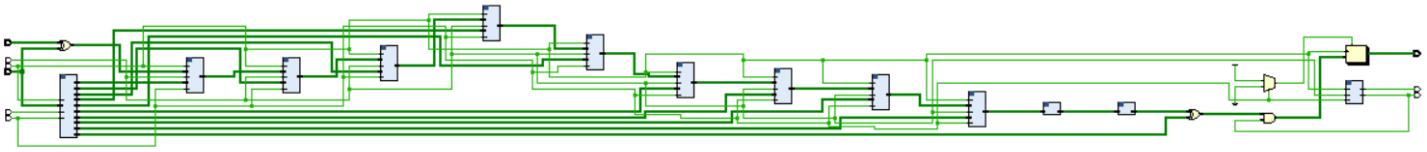
The **AES_Round_Orchestrator** manages the feedback mechanism by storing intermediate results from each round, and feeding them back into the system for the next iteration. This process continues until the entire block has been processed through the necessary 10 rounds.

3.2 Design Optimization

Single-Round Reuse for Area/Power Efficiency

To reduce the area and power consumption of the AES-128 module, we implemented a single-round reuse approach.

Instead of having separate hardware for each of the **10 AES rounds**, the design reuses a single AES round block. The encryption process is optimized by applying the round operations repeatedly, rather than constructing a full 10-round pipeline. This approach leads to a significant reduction in both hardware resource utilization and power consumption.



Resource Utilization Report:

Hierarchy

Name	Slice LUTs (41000)	Slice Registers (82000)	F7 Muxes (20500)	F8 Muxes (10250)	Bonded IOB (300)	BUFGCTRL (32)
AHB_Lite_AES128	32734	4264	6215	2052	103	2
uut (AES128_Encryptor)	32617	3975	6215	2052	0	0
AC (AES_Controller)	6	7	0	0	0	0
BS (ByteSub)	288	0	1984	864	0	0
KEF (KeyExpansion...	1601	2688	640	0	0	0
stage[1].RF (Round...	5462	128	798	264	0	0
stage[2].RF (Round...	3179	128	399	132	0	0
stage[3].RF (Round...	3179	128	399	132	0	0
stage[4].RF (Round...	3179	128	399	132	0	0
stage[5].RF (Round...	3179	128	399	132	0	0
stage[6].RF (Round...	3179	128	399	132	0	0
stage[7].RF (Round...	3179	128	399	132	0	0
stage[8].RF (Round...	3179	128	399	132	0	0
stage[9].RF (Round...	1152	128	0	0	0	0

By leveraging a **feedback mechanism** controlled by the **AES_Round_Orchestrator**, the intermediate data from each round is retained and used as input for the next round, thereby maintaining the flow of the encryption process without duplicating hardware for each round.

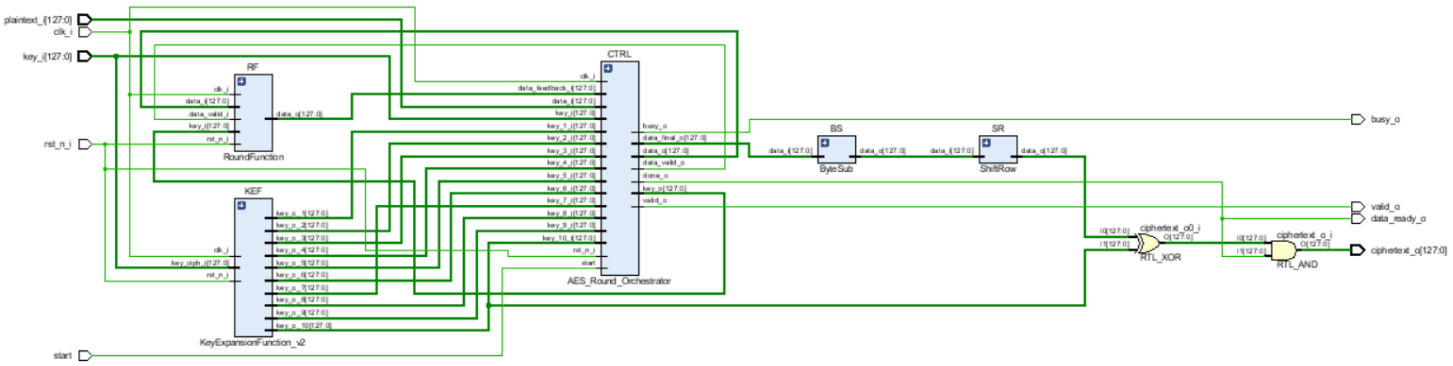
This design choice makes the AES accelerator highly efficient in terms of area and power usage, which is ideal for low-power applications such as embedded systems.

Feedback Mechanism for Iterating Through 10 Rounds

The **AES_Round_Orchestrator** is responsible for managing the feedback loop and iterating through the 10 AES rounds. This mechanism ensures that the same round block is reused for each iteration.

After the first round of AES is completed, the feedback from the round (the intermediate data) is passed back into the system and used as input for the next round.

This is done until all 10 rounds have been processed.



Resource Utilization Report:

Name	Slice LUTs (41000)	Slice Registers (82000)	F7 Muxes (20500)	F8 Muxes (10250)	Bonded IOB (300)	BUFGCTRL (32)
AHB_Lite_AES128	7321	2090	1487	612	103	3
uut (AES128)	7229	1801	1487	612	0	0
BS (ByteSub)	0	0	256	64	0	0
CTRL (AES_Round...	2402	393	0	0	0	0
KEF (KeyExpansion...	2955	1280	640	320	0	0
RF (RoundFunction)	1872	128	591	228	0	0

The feedback architecture operates as follows:

1. The **RISC-V core** initiates the AES encryption process and supplies the plaintext data and the AES key.
2. The **AES-128 accelerator** performs the first AES round using the feedback mechanism. The intermediate results are saved and fed back into the next round.
3. This process continues for all 10 rounds, with the final result being the encrypted ciphertext.
4. The **AES_Round_Orchestrator** ensures that the intermediate data from each round is properly managed, minimizing the need for extensive logic gates that would otherwise be required to handle multiple rounds simultaneously.

This feedback-driven design allows for a significant reduction in the size of the AES module while maintaining the integrity and security of the encryption.

3.3 Simulation Results

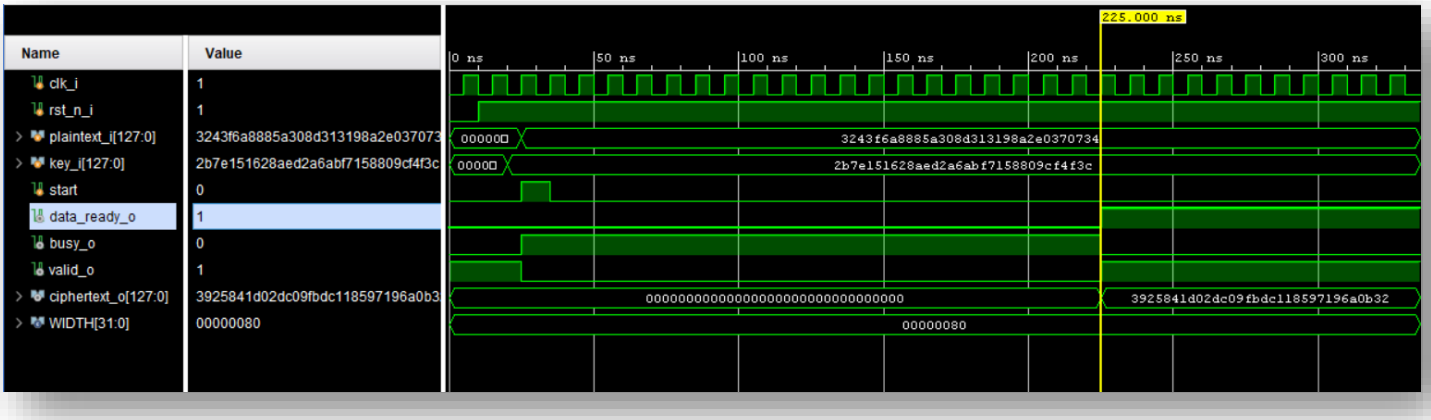
The **AES-128 accelerator** has been tested and verified through simulation using known encryption vectors.

The **NIST-provided example** was used as a reference for validation. The simulation results for AES encryption are consistent with the expected behavior for AES-128 encryption, confirming the correct implementation of the AES algorithm.

Below are some key results from the simulation:

Test Vector:

- **Plaintext**: 0x 32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34
- **Key**: 0x 2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c
- **Expected Ciphertext**: 0x39 25 84 1d 02 dc 09 fb dc 11 85 97 19 6a 0b 32



The test vector encryption was successfully performed by the AES-128 accelerator using the feedback architecture.

The ciphertext generated by the accelerator in **20 clock cycles** matches the expected output, confirming that the module operates correctly in terms of encryption.

These results demonstrate the validity of the **single-round reuse approach** and the effectiveness of the feedback mechanism in achieving the desired encryption output while maintaining area and power efficiency.

4. AHB-Lite Interface

The AES-128 Accelerator System-on-Chip (SoC) employs an **AHB-Lite interface** for seamless communication between the RISC-V core (acting as the master) and the AES-128 module, as well as other peripherals like RAM.

AHB-Lite is a simplified subset of the AMBA AHB protocol, designed to support high-performance communication with reduced complexity.

Key Features of the AHB-Lite Interface in the System

- **Single Sequential Transactions**:
  The system supports **single sequential transactions only**, which is sufficient for this design. This means burst operations are not supported. This design choice simplifies the hardware implementation while meeting the performance requirements of the application.
    - It's worth noting that other systems like ARM Cortex-M0, which are widely used in embedded systems, also support only single transactions, making this a common and acceptable trade-off in low-power designs.
- **Memory-Mapped Access**:
  The AES-128 module and RAM are mapped into the RISC-V core's address space, enabling easy access to these peripherals using standard load and store instructions.
- **Address-Decoding Logic**:
  A simple address decoder is used to route transactions from the AHB master (RISC-V core) to the appropriate slave module (AES accelerator or RAM).
- **AHB Arbitration**:
  Since there is only a **single master** in this system, no arbitration logic is required, further reducing complexity.

AHB-Lite Signals in the System

Below is a **summary** of the **key signals** used in the AHB-Lite interface:

Signal Name	Destination	Width	Description
HCLK	Manager and Subordinate	1	The bus clock times all bus transfers.
HRESETn	Manager and Subordinate	1	The bus reset signal is active LOW and resets the system and the bus.
HADDR	Subordinate, Decoder	ADDR_WIDTH (10–64)	Byte address of the transfer.
HBURST	Subordinate	HBURST_WIDTH (0, 3)	Number of transfers in the burst and address increments.
HMASTLOCK	Subordinate	1	Indicates if the transfer is part of a locked sequence.
HPROT	Subordinate	HPROT_WIDTH (0, 4, 7)	Provides information about the access type.
HSIZE	Subordinate	3	Indicates the size of the transfer.
HREADY	Manager and Subordinate	1	When HIGH, the HREADY signal indicates to the Manager and all Subordinates, that the previous transfer is complete.
HRDATA	Multiplexor, Manager	DATA_WIDTH	During read operations, the read data bus transfers data from the selected Subordinate to the multiplexor.
HREADYOUT	Multiplexor	1	When HIGH, the HREADYOUT signal indicates that a transfer has finished on the bus
HRESP	Multiplexor	1	When LOW, the HRESP signal indicates that the transfer status is OKAY. When HIGH, the HRESP signal indicates that the transfer status is ERROR.
HSELxa	Subordinate	1	Each Subordinate has its own select signal HSELx
HTRANS	Subordinate	2	Indicates the transfer type (IDLE, BUSY, NONSEQ, SEQ).
HWDATA	Subordinate	DATA_WIDTH (8–256)	Data transferred from the Manager during write operations.
HWSTRB	Subordinate	DATA_WIDTH/8	Write strobes indicating active byte lanes of HWDATA.
HWRITE	Subordinate	1	Indicates transfer direction (HIGH for write, LOW for read).

Transaction Flow

1. **Read Operation**:
    - The RISC-V master sends the target address (HADDR) and asserts HSEL for the target slave (AES-128 or RAM).
    - The slave provides the requested data (HRDATA) after processing the request.
2. **Write Operation**:
    - The master provides the target address (HADDR) and the data to be written (HWDATA), along with the HWRITE signal set to 1.
    - The slave writes the data to the appropriate location.

System Address Mapping

The following table summarizes the memory space and address mapping for the AES-128 accelerator and other components in the SoC. The memory space begins at 0x0000_0000 for RAM, while the AES module is mapped starting at 0x4000_0000.

Address (Hex)	Register/Signal Name	Description
0x0000_0000	RAM	Starting address of the system memory (RAM).
...	...	Other RAM addresses.
0x4000_0000	KEY0_ADDR	First 32-bit word of the 128-bit AES encryption key.
0x4000_0004	KEY1_ADDR	Second 32-bit word of the AES encryption key.
0x4000_0008	KEY2_ADDR	Third 32-bit word of the AES encryption key.
0x4000_000C	KEY3_ADDR	Fourth 32-bit word of the AES encryption key.
0x4000_0010	TEXT0_ADDR	First 32-bit word of the plaintext data block.
0x4000_0014	TEXT1_ADDR	Second 32-bit word of the plaintext data block.
0x4000_0018	TEXT2_ADDR	Third 32-bit word of the plaintext data block.
0x4000_001C	TEXT3_ADDR	Fourth 32-bit word of the plaintext data block.
0x4000_0020	CTRL_ADDR	Control register to start and monitor AES operations.
0x4000_0024	CIPHER0_ADDR	First 32-bit word of the encrypted ciphertext output.
0x4000_0028	CIPHER1_ADDR	Second 32-bit word of the encrypted ciphertext output.
0x4000_002C	CIPHER2_ADDR	Third 32-bit word of the encrypted ciphertext output.
0x4000_0030	CIPHER3_ADDR	Fourth 32-bit word of the encrypted ciphertext output.
0x4000_0034	DONE_STATUS	Status register indicating completion of AES encryption.

5. System Integration

Address Map

- **Memory Mapping:** The memory and AES module are mapped into the AHB address space as follows:
    - **0x0000_0000 - 0x0000_FFFF**: General-purpose memory.
    - **0x4000_0000 - 0x4000_FFFF**: AES module registers for plaintext, key, control/status and the output cipher text

Data Flow

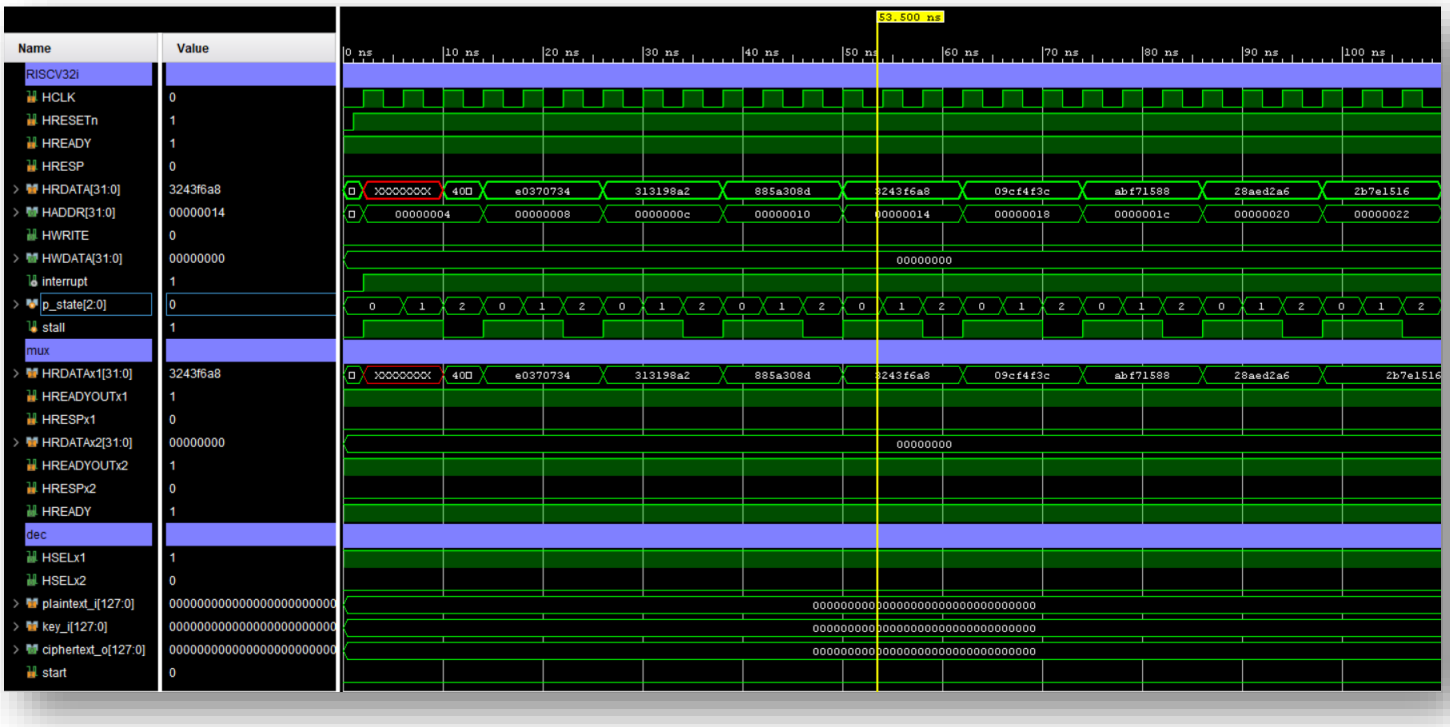
1. **Write Operation**
    - The RISC-V core writes the plaintext and encryption key into the AES module's registers via the AHB interface.
2. **Encryption Trigger**
    - The core sets the control register to initiate the encryption process.
    - The AES module begins the encryption algorithm based on the provided inputs.
3. **Read Operation**
    - Once encryption completes, the core reads the encrypted ciphertext from the AES module’s output register.

6. Testing and Verification

Simulation

- **Testbench Setup:**
    - I wrote an RISC-V assembly program along with the linker script and run a simple Verilog testbench simulates the RISC-V core writing to and reading from the AES module.
    - AHB protocol signals are monitored to ensure compliance.
- **Test Cases Run:**
    - Different plaintexts and keys were provided, and their corresponding expected ciphertexts were verified using a golden AES model provided by NIST documentation.

Reading Operation => The Processor Load The Plain Text and The Key From The Data Memory:



Writing Operation => The Processor Store The Plain Text and The Key To The AES128 Accelerator Along With The Start Signal To Start The Encryption Process:

The Final Correct Cipher Text Output

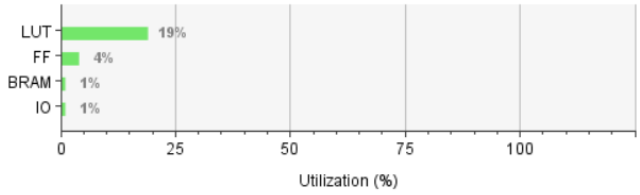
7. Resource Utilization and Power

Full System Resource Utilization Report:

Name	Slice LUTs (41000)	Slice Registers (82000)	F7 Muxes (20500)	F8 Muxes (10250)	Block RAM Tile (135)	Bonded IOB (300)	BUFGCTRL (32)
top_SoC	7807	3298	1647	644	1	2	6
master (RISCV32_Wrapper)	569	1132	128	0	0	0	0
slave_1_mem (AHB_Lite_MEM)	0	76	0	0	1	0	0
slave_2_aes128 (AHB_Lite_AES128)	7238	2090	1519	644	0	0	0

Summary

Resource	Utilization	Available	Utilization %
LUT	7807	41000	19.04
FF	3298	82000	4.02
BRAM	1	135	0.74
IO	2	300	0.67



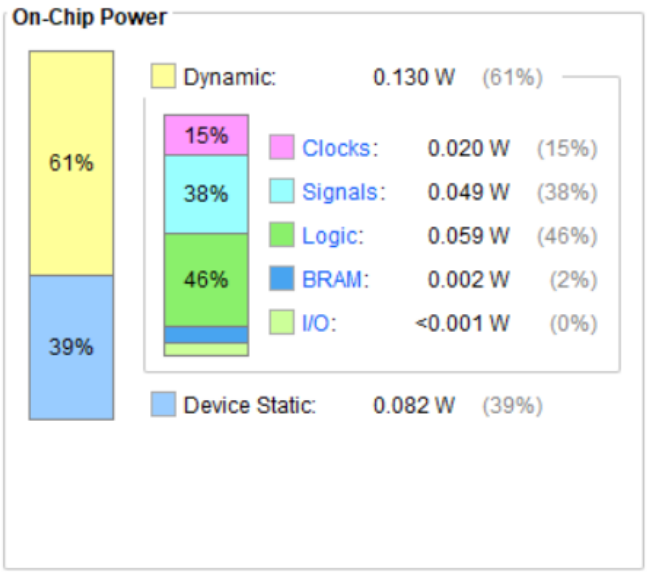
Final Power Report:

Summary

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**0.211 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25.4°C**
Thermal Margin:	59.6°C (31.4 W)
Effective ϑJA:	1.9°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity



8. Challenges and Solutions

Encryption Only

- **Progress:** Currently, only encryption functionality is implemented and verified.
- **Future Plan:** The decryption process will be added next.

9. Future Work

1. **Add Decryption Capability**
    - Extend the current AES module to support decryption using the inverse cipher.
2. **Optimize Timing and Area/Power:**
    - Implement further pipelining and clock gating to reduce timing violations and power usage.
3. **Add DMA Integration:**
    - Enable efficient memory transfers between core, AES module, and memory using a DMA controller.

10. References

1. **AES Algorithm Reference:**
    - National Institute of Standards and Technology (NIST) Federal Information Processing Standards Publication 197 (FIPS-197).
2. **AHB Protocol Specification:**
    - ARM AMBA 3 AHB-Lite Protocol Specification.