# AES ENCRYPTION OF IMAGE ON FPGA

Sreyas Janamanchi (IMT2022554)

Pradyumna G (IMT2022555)

Chinmay Krishna R (IMT2022561)

## BACKGROUND

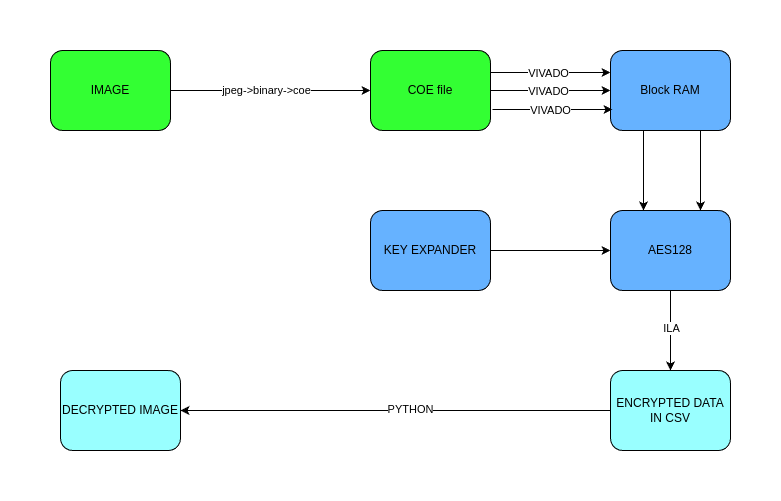
The Advanced Encryption Standard (AES) is a trusted encryption algorithm widely used for securing sensitive data, offering key lengths of 128, 192, or 256 bits to prevent unauthorized access. In this project, we implement AES-128 encryption, a method extensively used to protect internet communications and sensitive files.

Implementing AES encryption on an FPGA enables secure, real-time processing of sensitive visual data, leveraging parallel processing for low latency and efficient resource utilization. This approach is ideal for time-sensitive applications such as surveillance and medical imaging, ensuring robust data security and adaptability in embedded systems.

## METHODOLOGY

In this project we perform acceleration of AES-128 encryption on an image using FPGA. Here’s the link to our code: <https://github.com/Sreyas-J/ImageAES_FPGA>

Reference: <https://github.com/michaelehab/AES-Verilog,> this gave us inspiration for a basic non-accelerated AES-128 encryption. The code implements composite arithmetic algorithm for acceleration.



* The process begins with converting an image into a .coe file format. The .coe file is a series of 128-bit plaintexts.
* A round-robin algorithm was implemented in python to split the image data into three .coe files. Round-robin was chosen to make the project easily extendable to real-time image transfer.
* Each .coe file is used to create a block RAM (BRAM).
* We have instantiated six AES-128 modules. Each BRAM then provides input to two AES-128 modules using another round-robin to distribute data.
* The data is exported from ILA to a csv file.
* The csv is processed through Python code to reassemble the data in the correct order, accounting for a twelve-clock cycle delay. Multiple adjustments were required to synchronize the round-robin approach with the clock delay for accurate data retrieval.

To accelerate AES-128 encryption, the following optimization techniques were implemented: -

* **Parallel Computation**: Multiple AES encryption instances were instantiated, enabling parallel processing and significantly increasing throughput.
* **Single Key Expansion**: Key expansion is executed only once in the main module, and the expanded key is shared across all instances, eliminating redundant calculations.
* **Optimized MixColumn Operation**: Composite arithmetic using Galois fields was applied to optimize the MixColumn step, a core operation that is repeatedly performed during encryption.
* **Increased Clock Frequency**: The clock frequency was maximized to further boost the output rate, enhancing the overall speed of encryption.

## REPORTS

### BASIC IMPLEMENTATION

Clock frequency: 30.303MHz

Latency= 12 clock cycles

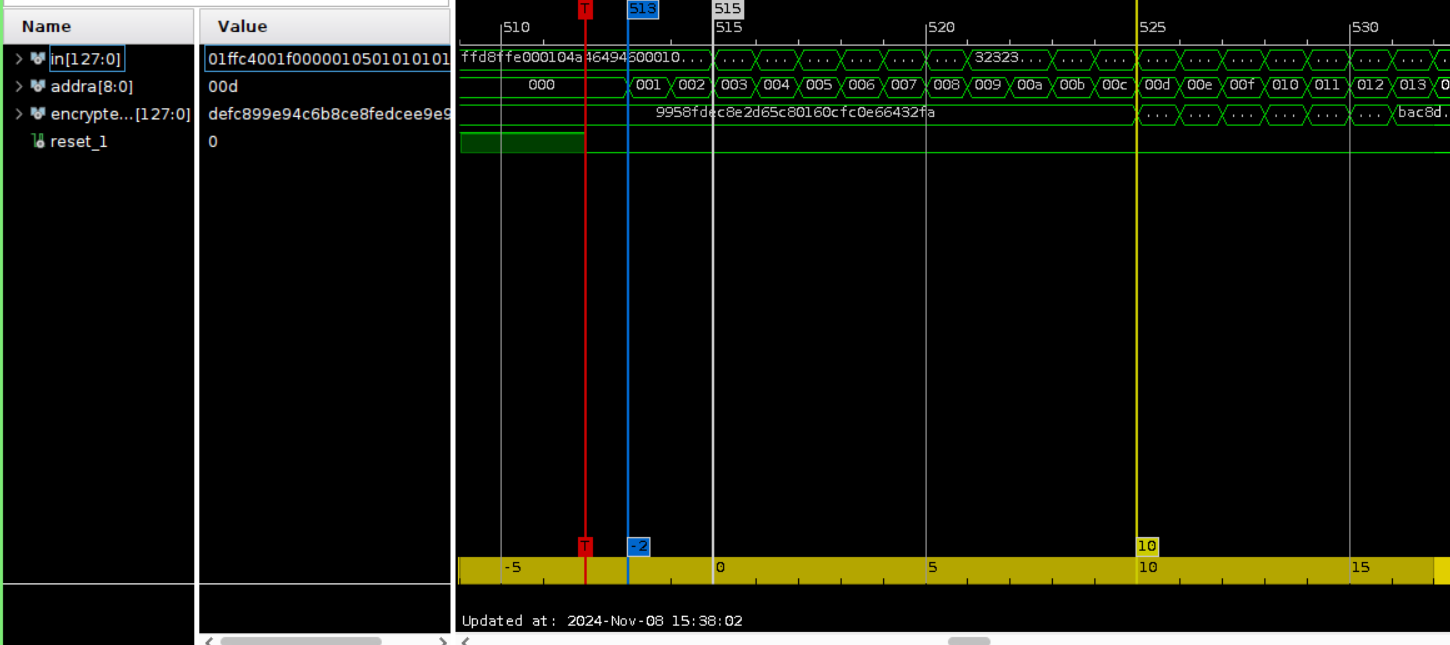
Latency: 396ns

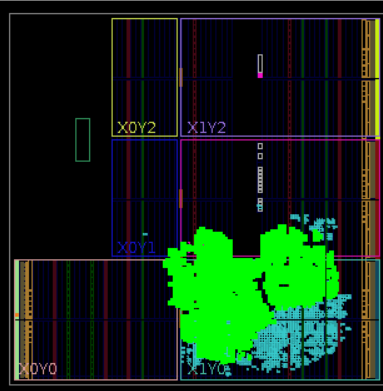
Throughput= Clock frequency

Throughput: 30.303\*106 ops/sec

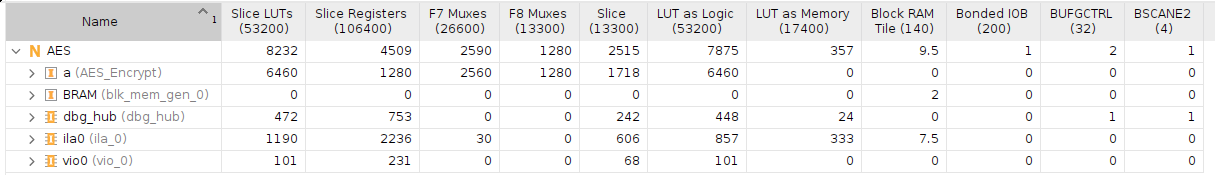
Time taken for encrypting a 128px\*128px jpeg image: 9504ns



We can see here that it has a 12-clock cycle delay from BRAM address update to encrypted value output (2 clock cycles to read the value and 10 clock cycles for AES encryption)



Implemented Layout

Resource Utilisation

### ACCELERATED IMPLEMENTATION

Clock frequency: 119.403MHz

Worst negative slack: 0.795ns

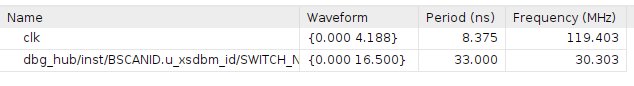
Latency= 12 clock cycles

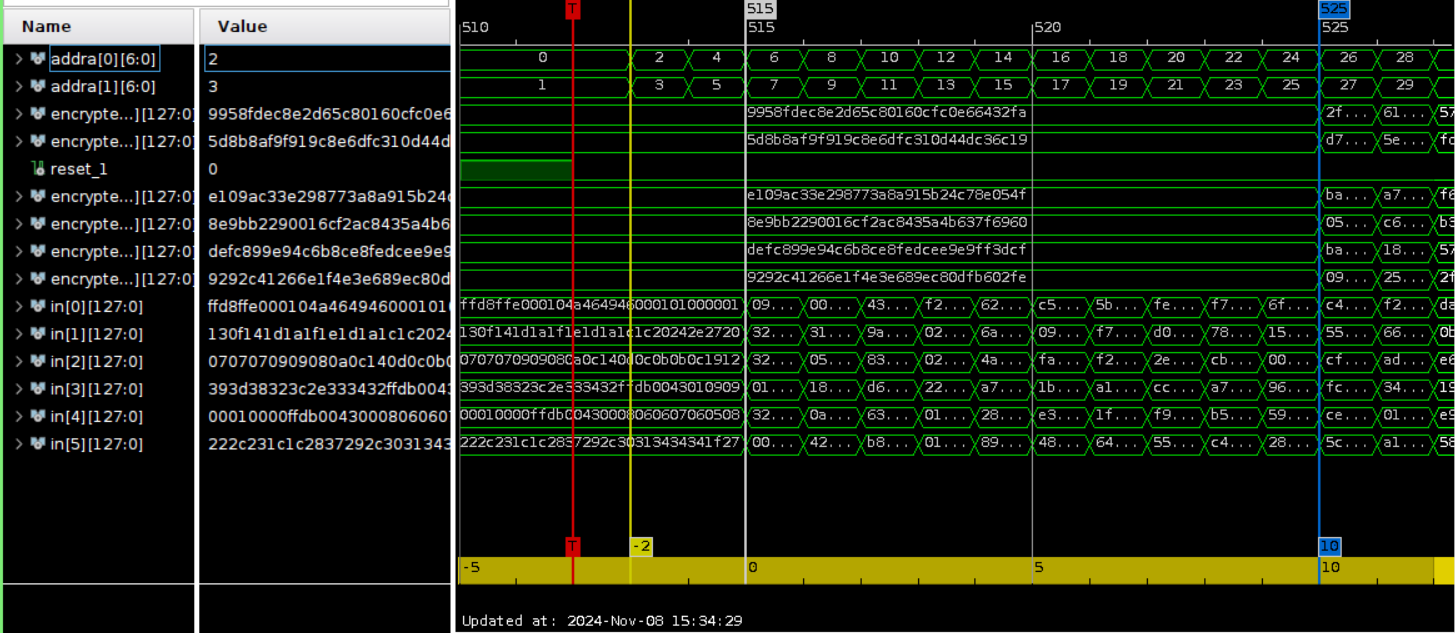
Latency: 100.5ns

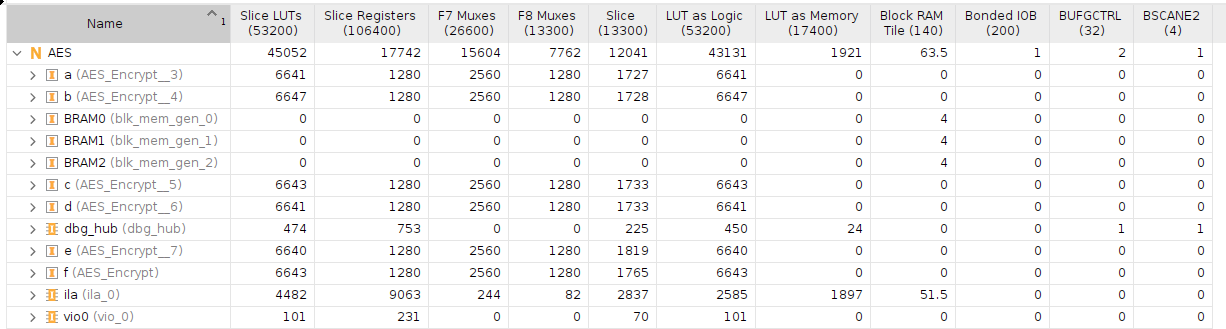
Throughput= Clock\_frequency\*6

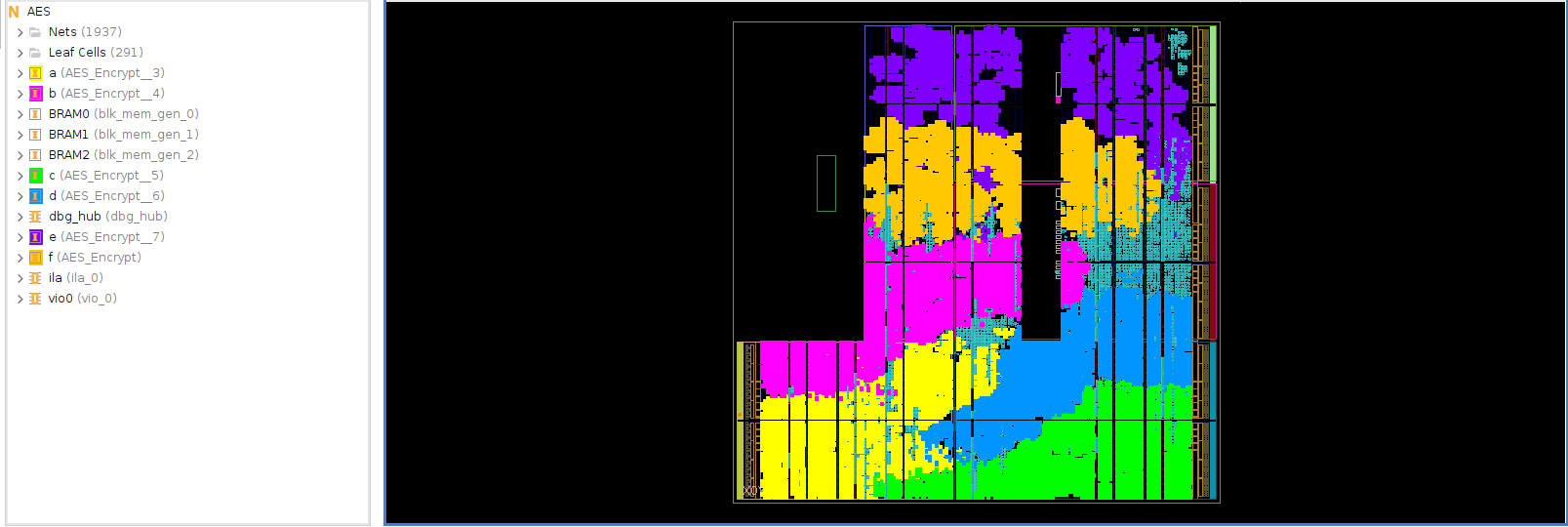
Throughput: 716.418\*106 ops/sec

Time taken for encrypting a 128px\*128px jpeg image: 485.75ns



We can see here that it has a 12-clock cycle delay from BRAM address update to encrypted value output (2 clock cycles to read the value and 10 clock cycles for AES encryption)

Resource Utilisation

Implemented Layout

## RESULTS

## FUTURE ASPECTS

* Further acceleration can be achieved through pipelining. The encryption pipeline consists of nine instances, each handling rounds 2 through 10. When a plaintext is being processed in a specific round, the remaining instances are free to begin encrypting additional plaintexts, maximizing throughput and efficiency.
* The image can be dynamically transmitted from a microcontroller or processing system (PS) via a communication protocol such as UART or AXI. The encrypted image is then sent back to the microcontroller/PS, enabling automation and easing implementation for embedded systems.
* Further security can be improved by accelerating authentication algorithms like **RSA** and **AES-CMAC** on FPGA to verify data authenticity and integrity, in addition to confidentiality provided by AES encryption.