Efficient FPGA Implementation of a Look-Up Table-Based Mix-Column Transform for Lightweight Cryptography

Yasir Amer Abbas[1], Saad Al-Azawi[1], Norziana Jamil[2], Muhammad Reza Z'aba[3]

[1]College of Engineering, University of Diyala, Iraq
[2]College of IT, UAE University, United Arab Emirates
[3]National Center of Technology and Cryptography, Serdang, Selangor, Malaysia

dr.yasiral-zubaidi@uodiyala.edu.iq, saad.alazawi@uodiyala.edu.iq
norziana@uaeu.ac.ae, reza.zaba@outlook.com

Introduction

PHOTON AND LED lightweight algorithms offer low power and high speed.However, the MixColumn Transform (MCT), a core operation in their permutation layer, remains a bottleneck for performance and hardware efficiency in FPGA. Traditional MCTs rely on complex combinational logic or large LUTs, resulting in area inefficiency.

Our Method

We introduce a MixColumn Transform design based on finite field GF(256) multiplication using a logarithmic method optimized for FPGA, which reduces logic complexity by precomputing and storing repeated Mix-Column results in a dedicated LUT. By leveraging FPGA dual-output LUT, the design eliminates the need for on-the-fly calculations and enables faster mix-column operations with more resource-efficient hardware. Specifically, instead of using a full 256 × 256 LUT, we implement:

- A **logarithm LUT** storing $\log(a)$ values for $a \in \{1, \ldots, 255\}$
- An **antilog LUT** for $\exp(i)$ modulo 255

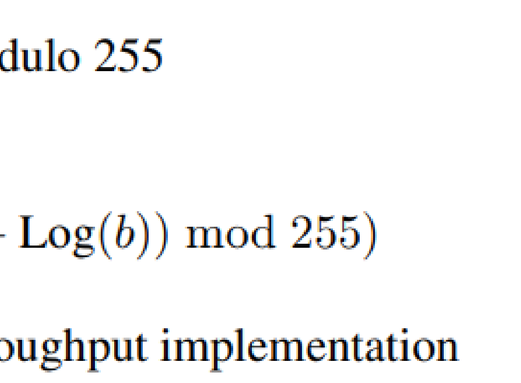
The multiplication is computed as:

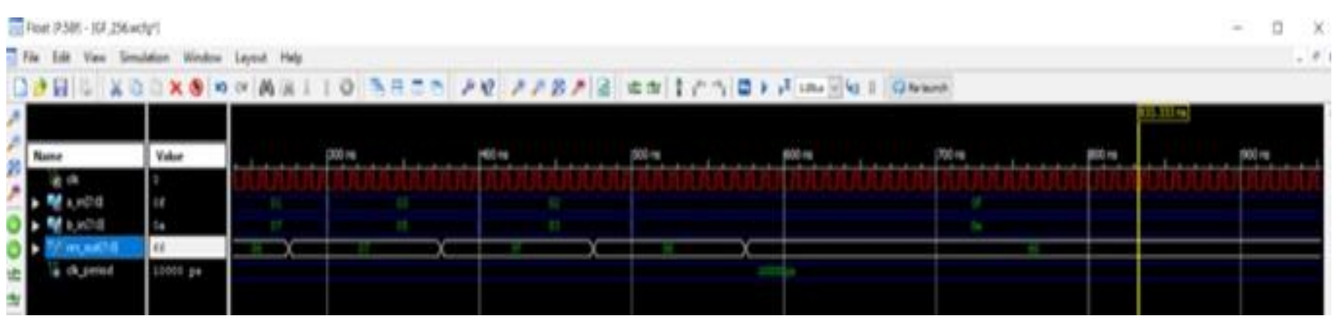
$$GF(a, b) = \text{AntiLog}((\text{Log}(a) + \text{Log}(b)) \bmod 255)$$

This enables a compact and high-throughput implementation while maintaining cryptographic correctness. Overflow is managed via conditional subtraction. The step-by-step computation of the GF(256) multiplication using logarithmic and anti-logarithmic tables is given in Algorithm 1. It implements the core GF(256) multiplication, which can be composed into larger MixColumn matrix transformations for lightweight cipher designs.

Algorithm 1 GF(256) Multiplication using Log-Antilog LUTs

Require: Input bytes a, b
Ensure: Output byte $c = a \cdot b$ in GF(256)
if $a = 0$ or $b = 0$ **then**
  $c \leftarrow 0$
else
  $L_a \leftarrow \text{LogLUT}[a]$
  $L_b \leftarrow \text{LogLUT}[b]$
  $L_c \leftarrow (L_a + L_b) \bmod 255$
  $c \leftarrow \text{AntiLogLUT}[L_c]$
end if
return c =0





Experimental Results

We implemented the design in VHDL using Xilinx ISE 14.5. The architecture uses two ROMs (256×8), a modulo adder, and a control FSM. The design was tested on a Spartan-3 (XC3S500E-5FG676) using ISim. RTL diagrams and signal flow were verified for correctness. The results are given in Table 1.

Metric	GF256_LUT (Proposed)
Block Size	8 bits
Device	Spartan3-5FG676
Clock Cycles	3
Max Frequency	301.66 MHz
Throughput	804.42 Mbps
Total Slices	41
Efficiency	19.62 Mbps/Slice
Power Consumption	0.057 W
Delay	3.315 ns

TABLE 1. Performance of Log-based GF(256) multiplier

Our Findings

The efficiency of our method can be summarized as follows:

1) Addition-based method rather than using a computationally expensive multiplication operation.
2) The multiplication by zero, which produces zero, is eliminated to reduce the computation cost.
3) As the proposed architecture is based on lookup tables, the highest computational speed is achieved by avoiding multiplication operations.
4) Only two 256 × 8 block memories are used to store the precomputed log and antilog values.

TABLE 2. Comparison with other techniques

Methods	Description	Key Features
Log/Antilog Table Method (This work)	Precomputed Log and Antilog tables	• Simple in hardware implementation • No multiplication operations • Fast lookup tables • Low memory size: (2 × 256 × 8)
Polynomial Multiplication	Polynomial multiplications	• Complex • High cost as it requires multiplication operations • Low speed
Lookup Table (Full 256 × 256)	Precomputed all possible multiplications (256 × 256 × 8) bits	• No multiplication • High memory size: 256 × 256 × 8 • Not suitable for larger GF multiplication fields
Bitwise Shift & XOR (Russian Peasant)	Iterative method using shifts and conditional XOR with the irreducible polynomial	• No LUT is required • Low speed • Complex