Πλήρης Τεχνική Ανάλυση VHDL: CLEFIA Cipher (128-bit)

Ανάλυση RTL & Μαθηματική Τεκμηρίωση

January 17, 2026

Abstract

This document provides a comprehensive, line-by-line analysis of the VHDL implementation for the CLEFIA block cipher (128-bit). It breaks down the architecture into 8 distinct processes, explaining the mapping between the VHDL RTL logic and the mathematical specifications of the algorithm (Generalized Feistel Network, Galois Field multiplication, and Key Scheduling).

Contents

1 Εισαγωγή

Ο παρών κώδικας περιγράφει την υλοποίηση του αλγορίθμου **CLEFIA** (128-bit key/block) σε επίπεδο RTL. Η σχεδίαση χρησιμοποιεί μια κεντρική Μηχανή Πεπερασμένων Καταστάσεων (FSM) για τον συγχρονισμό των λειτουργιών.

Ο αλγόριθμος βασίζεται σε δομή *Generalized Feistel Network* (GFN) 4 κλάδων και απαιτεί 18 γύρους για κλειδί 128-bit.

2 Ανάλυση Διεργασιών **(Processes)**

2.1 Process 1: Input Registering (Συγχρονισμός Εισόδου)

Σκοπός: Η απομόνωση του κυκλώματος από αστάθειες των σημάτων εισόδου.

```
process(clk)
begin
    if rising_edge(clk) then
        if rst = '1' then
            reg_start <= '0'; ...
        else
            reg_start <= start;
            reg_plaintext <= plaintext;
            reg_key <= key;
        end if;
    end if;
end process;
```

Ανάλυση:

- Η διεργασία είναι πλήρως σύγχρονη (ενεργοποιείται στο `rising_edge`).
- Όλα τα σήματα εισόδου (`plaintext`, `key`, `start`) αποθηκεύονται σε εσωτερικούς καταχωρητές (`reg_*`). Αυτό εξασφαλίζει ότι αν το εξωτερικό σήμα αλλάξει κατά τη διάρκεια ενός κύκλου υπολογισμού, η εσωτερική λογική δεν θα επηρεαστεί (glitch protection).

2.2 Process 2: Phase Control (Έλεγχος Φάσης)

Σκοπός: Η διαίρεση του υπολογιστικού φόρτου ενός γύρου σε δύο κύκλους ρολογιού.

```
process(clk) ...
    if state = KEY_GFN or state = ROUNDS then
        phase <= not phase;
    else
        phase <= '0';
    end if;
...
step_en <= '1' when ... (phase = '1') else '0';
```

Ανάλυση:

- Ο αλγόριθμος CLEFIA περιλαμβάνει S-box lookup και πολλαπλασιασμό πινάκων στον ίδιο γύρο. Αν γίνονταν όλα σε έναν κύκλο, το κρίσιμο μονοπάτι (critical path) θα ήταν μεγάλο, μειώνοντας τη συχνότητα λειτουργίας.
- **Phase 0:** Γίνεται η ανάγνωση των S-boxes (Processes 4 & 5).
- **Phase 1:** Γίνεται ο πολλαπλασιασμός πινάκων και η εγγραφή στους καταχωρητές (Processes 6 & 7).
- Το σήμα `step_en` επιτρέπει στην FSM (Process 3) να προχωρήσει μόνο όταν ολοκληρωθεί η Φάση 1.

2.3 Process 3: FSM (Μηχανή Καταστάσεων)

Σκοπός: Ο κεντρικός έλεγχος της ροής του αλγορίθμου.

```
case state is
    when KEY_GFN =>
        if gfn_cnt >= 12 then ... -- Generating L from K
    when KEY_EXPAND =>
        if ks_done = '1' then ... -- Expanding K and L
    when ROUNDS =>
        if round_cnt >= 17 then ... -- 18 Rounds (0 to 17)
```

Ανάλυση:

- **KEY_GFN:** Υλοποιεί τον βρόχο 12 γύρων για την παραγωγή του ενδιάμεσου κλειδιού L.
- **KEY_EXPAND:** Ελέγχει τη διαδικασία επέκτασης κλειδιού μέχρι να παραχθούν όλα τα Round Keys (RK).
- **ROUNDS:** Εκτελεί τους 18 γύρους επεξεργασίας δεδομένων που απαιτούνται για κλειδί 128-bit.

2.4 Process 4 & 5: F0/F1 Pipelines (Υποκατάσταση)

Σκοπός: Η υλοποίηση του πρώτου μισού των συναρτήσεων F_0 και F_1, δηλαδή το XOR με το κλειδί και η υποκατάσταση S-box.

```
-- Process 4 (F0)
v_tmp := v_in_data xor v_in_rk; [cite_start]-- Step 1: T <- RK xor x [cite:
    102]
[cite_start]-- S-box Lookup Sequence: S0, S1, S0, S1 [cite: 104-107]
f0_pipe(31 downto 24) <= S0(r,c);
f0_pipe(23 downto 16) <= S1(r,c); ...
```

Ανάλυση:

- Επιλογή Εισόδου: Ανάλογα με το αν είμαστε σε διαδικασία Κρυπτογράφησης ή Αποκρυπτογράφησης, επιλέγονται τα κατάλληλα δεδομένα (T_0 ή T_3) και το κατάλληλο κλειδί RK.
- **S-Box Addressing:** Η είσοδος των 32-bit χωρίζεται σε 4 bytes. Κάθε byte χρησιμοποιείται ως συντεταγμένες (r, c) για τους 2D πίνακες S_0, S_1.

- Διαφορά **F0/F1:** Η Process 4 χρησιμοποιεί τη σειρά S_0, S_1, S_0, S_1, ενώ η Process 5 χρησιμοποιεί S_1, S_0, S_1, S_0, ακολουθώντας πιστά τις εξισώσεις του προτύπου .

2.5 Process 6: Key Schedule (Χρονοδρομολόγηση Κλειδιού)

Σκοπός: Η παραγωγή του L και όλων των RK/WK. Αυτή η διεργασία περιέχει σύνθετη λογική.

2.5.1 Τμήμα 1: Παραγωγή L (State: KEY_GFN)

Εδώ υλοποιείται ο μετασχηματισμός $GFN_{4,12}$.

```
-- Inline Matrix Multiplication (M0/M1 logic reused here)
-- ... (Shift/XOR operations identical to Datapath logic)
L_reg <= (L1 xor f0_result) & L2 & (L3 xor f1_result) & L0;
```

- Χρησιμοποιεί τα αποτελέσματα των `f0_pipe/f1_pipe` (που έχουν ήδη περάσει από τα S-boxes στην προηγούμενη φάση).
- Εκτελεί τον πολλαπλασιασμό πινάκων (inline) και ενημερώνει το L_reg κάνοντας κυκλική εναλλαγή (Feistel swap).

2.5.2 Τμήμα 2: Επέκταση Κλειδιού (State: KEY_EXPAND)

Υλοποιεί τον αλγόριθμο της ενότητας 5.2 .

Λογική **DoubleSwap (Σ):**

```
Swap_Out := Swap_In(120 downto 64) & Swap_In(6 downto 0) &
            Swap_In(127 downto 121) & Swap_In(63 downto 7);
```

- Ο κώδικας αυτός υλοποιεί την εξίσωση[cite: 399]:

$$Y = X[7-63]|X[121-127]|X[0-6]|X[64-120]$$

- Προσοχή: Στη VHDL τα bits είναι αριθμημένα $127 \rightarrow 0$ (Big Endian), ενώ στο πρότυπο η αρίθμηση μπορεί να διαφέρει. Η υλοποίηση εδώ αντιστοιχεί στη μετακίνηση των bits όπως φαίνεται στο Σχήμα 4.

XOR με Σταθερές:

```
T0_k := L_reg(...) xor CON_128(24 + idx);
```

- Το ενδιάμεσο κλειδί L γίνεται XOR με τις σταθερές CON που είναι αποθηκευμένες σε πίνακα `constant`.

2.6 Process 7: Datapath (Μονοπάτι Δεδομένων)

Σκοπός: Η εκτέλεση της κρυπτογράφησης (Section 4). Περιλαμβάνει τη μαθηματική καρδιά του CLEFIA: τον πολλαπλασιασμό στο $GF(2^8)$.

2.6.1 Μαθηματικό Υπόβαθρο: Πολλαπλασιασμός Πινάκων

Οι πίνακες M_0 και M_1 [cite: 276-277] απαιτούν πολλαπλασιασμό διανύσματος με πίνακα στο σώμα $GF(2^8)$ με ανάγωγο πολυώνυμο $P(z) = z^8 + z^4 + z^3 + z^2 + 1$ (0x11D).

```
--                          'xtime' (                                    2)
tmp_byte := b(i)(6 downto 0) & '0'; -- Shift Left (x2)
if b(i)(7) = '1' then               -- Check Overflow
    tmp_byte := tmp_byte xor x"1d"; -- Modulo reduction (0x1D)
end if;
b_x2(i) := tmp_byte;
```

Εξήγηση:

- Στο δυαδικό σώμα, ο πολλαπλασιασμός επί x (ή επί 2 δεκαδικό) είναι μια ολίσθηση αριστερά (« 1).
- Αν το MSB είναι 1, τότε το αποτέλεσμα υπερβαίνει το 255. Πρέπει να αφαιρέσουμε (στο GF(2) η αφαίρεση είναι XOR) το ανάγωγο πολυώνυμο. Το 0x11D κόβεται σε 8-bit ως 0x1D.
- Οι τιμές $\times 4, \times 6, \times 8$ υπολογίζονται αναδρομικά (π.χ. $x \cdot 6 = (x \cdot 4) \oplus (x \cdot 2)$).

2.6.2 Υλοποίηση Πινάκων

```
-- Matrix M0 implementation [cite: 276]
-- Row 0: 0x01, 0x02, 0x04, 0x06
res0(0) := b(0) xor b_x2(1) xor b_x4(2) xor b_x6(3);
```

- Η VHDL υλοποιεί τον πολλαπλασιασμό πίνακα-διανύσματος ως άθροισμα (XOR) των επιμέρους γινομένων.

2.6.3 Feistel Structure Update

```
if reg_mode = '0' then -- Encryption
    v_T1 := v_T1 xor f0_result; -- T1 <- T1 ^ F0(T0, RK)
    v_T3 := v_T3 xor f1_result; -- T3 <- T3 ^ F1(T2, RK)
    T0 <= v_T1; T1 <= v_T2; [cite_start]... -- Rotation/Swap [cite: 76]
else -- Decryption
    v_T1 := T0 xor f0_result;   -- Inverse operations
    ...
end if;
```

- Υλοποιεί τα βήματα του GFN. Στην κρυπτογράφηση οι κλάδοι περιστρέφονται αριστερά, ενώ στην αποκρυπτογράφηση η ροή των δεδομένων και των κλειδιών αντιστρέφεται .

2.7 Process 8: Output Registers

Σκοπός: Η σύνθεση της εξόδου.

```
if state = OUTPUT then
    reg_ciphertext <= T0 & T1 & T2 & T3;
    reg_done <= '1';
```

```
4       reg_valid <= '1';
5   end if;
```

- Συνενώνει τα τέσσερα τμήματα των 32-bit (T_0, T_1, T_2, T_3) στο τελικό διάνυσμα 128-bit.
- Ενεργοποιεί τα σήματα ελέγχου (`handshake signals`) για να ενημερώσει το υπόλοιπο σύστημα ότι η διαδικασία ολοκληρώθηκε.