Αναλυτική Περιγραφή του CLEFIA Block Cipher

Detailed Analysis of CLEFIA VHDL Implementation

Hardware Cryptography Analysis

January 17, 2026

Contents

1 Εισαγωγή (Introduction)

Ο CLEFIA είναι ένας 128-bit block cipher που αναπτύχθηκε από τη Sony Corporation το 2007. Υποστηρίζει κλειδιά μεγέθους 128, 192, και 256 bits και είναι συμβατός με το πρότυπο AES. Η αρχιτεκτονική του βασίζεται σε μια γενικευμένη δομή Feistel με τέσσερις γραμμές δεδομένων (4-branch generalized Feistel network).

The CLEFIA cipher consists of two main components:

- **Data Processing Part**: Εκτελεί την κρυπτογράφηση/αποκρυπτογράφηση
- **Key Scheduling Part**: Παράγει τα whitening keys και round keys

2 Βασικά Χαρακτηριστικά της Υλοποίησης

2.1 Αρχιτεκτονική Επισκόπηση

Η υλοποίηση σε VHDL ακολουθεί μια πλήρως σύγχρονη (fully synchronous) αρχιτεκτονική με τα εξής χαρακτηριστικά:

1. **Pipeline Architecture**: Χρησιμοποιεί δύο στάδια για τις F-functions
2. **Register-based Design**: Όλες οι είσοδοι και έξοδοι καταχωρούνται
3. **State Machine Control**: Finite State Machine (FSM) για έλεγχο ροής
4. **Inline Matrix Operations**: Οι πολλαπλασιασμοί πινάκων γίνονται inline

2.2 Κύριες Καταστάσεις (Main States)

Η μηχανή καταστάσεων έχει 7 κύριες καταστάσεις:

```
type state_type is (IDLE, KEY_GFN, KEY_EXPAND,
                    INIT_WHITE, ROUNDS, FINAL_WHITE, OUTPUT);
```

- **IDLE**: Αναμονή για σήμα εκκίνησης (start signal)
- **KEY_GFN**: Εκτέλεση του GFN4,12 για παραγωγή του ενδιάμεσου κλειδιού L
- **KEY_EXPAND**: Επέκταση κλειδιών για παραγωγή WK και RK
- **INIT_WHITE**: Αρχικό whitening με WK0 και WK1
- **ROUNDS**: Εκτέλεση των 18 γύρων κρυπτογράφησης
- **FINAL_WHITE**: Τελικό whitening με WK2 και WK3
- **OUTPUT**: Έξοδος αποτελέσματος

3 Δομές Δεδομένων (Data Structures)

3.1 S-Boxes

Οι πίνακες αντικατάστασης (S-boxes) αποτελούν κρίσιμο στοιχείο του CLEFIA. Υπάρχουν δύο S-boxes: S0 και S1.

S0 Structure: Ο S0 κατασκευάζεται από τέσσερις 4-bit S-boxes (SS0, SS1, SS2, SS3):

$$\mathrm{S0}(x) : \{0,1\}^8 \to \{0,1\}^8 \tag{1}$$
$$x = x_0|x_1, \quad x_i \in \{0,1\}^4 \tag{2}$$
$$t_0 \leftarrow \mathrm{SS0}(x_0), \quad t_1 \leftarrow \mathrm{SS1}(x_1) \tag{3}$$
$$u_0 \leftarrow t_0 \oplus 2 \cdot t_1, \quad u_1 \leftarrow 2 \cdot t_0 \oplus t_1 \tag{4}$$
$$y_0 \leftarrow \mathrm{SS2}(u_0), \quad y_1 \leftarrow \mathrm{SS3}(u_1) \tag{5}$$
$$y = y_0|y_1 \tag{6}$$

S1 Structure: Ο S1 βασίζεται σε αντίστροφη συνάρτηση στο $GF(2^8)$:

$$\mathrm{S1}(x) = \begin{cases} g(f(x)^{-1}) & \text{if } f(x) \neq 0 \\ g(0) & \text{if } f(x) = 0 \end{cases} \tag{7}$$

όπου οι f και g είναι affine transformations στο $GF(2)$.

3.2 Diffusion Matrices

Δύο 4×4 πίνακες χρησιμοποιούνται για διάχυση (diffusion):

$$M_0 = \begin{bmatrix} 0x01 & 0x02 & 0x04 & 0x06 \\ 0x02 & 0x01 & 0x06 & 0x04 \\ 0x04 & 0x06 & 0x01 & 0x02 \\ 0x06 & 0x04 & 0x02 & 0x01 \end{bmatrix}, \quad M_1 = \begin{bmatrix} 0x01 & 0x08 & 0x02 & 0x0a \\ 0x08 & 0x01 & 0x0a & 0x02 \\ 0x02 & 0x0a & 0x01 & 0x08 \\ 0x0a & 0x02 & 0x08 & 0x01 \end{bmatrix} \tag{8}$$

Οι πολλαπλασιασμοί εκτελούνται στο $GF(2^8)$ με primitive polynomial $z^8+z^4+z^3+z^2+1$.

4 Key Scheduling - Αναλυτική Περιγραφή

Το Key Scheduling είναι η πιο κρίσιμη διαδικασία του CLEFIA. Για κλειδί 128-bit, η διαδικασία έχει δύο στάδια:

4.1 Φάση 1: Παραγωγή Ενδιάμεσου Κλειδιού L

Αυτό το στάδιο εκτελείται στην κατάσταση `KEY_GFN` και διαρκεί 12 κύκλους (gfn_cnt = 0 έως 12).

4.1.1 Κύκλος 0 - Αρχικοποίηση

```
if gfn_cnt = 0 then
    L_reg <= reg_key;
```

Το αρχικό κλειδί $K = K_0|K_1|K_2|K_3$ φορτώνεται στο καταχωρητή L_reg όπου:

$$K_0 = \text{reg\_key}[127:96] \quad (32 \text{ bits}) \tag{9}$$
$$K_1 = \text{reg\_key}[95:64] \tag{10}$$
$$K_2 = \text{reg\_key}[63:32] \tag{11}$$
$$K_3 = \text{reg\_key}[31:0] \tag{12}$$

4.1.2 Κύκλοι 1-12: GFN4,12 Transformation

Σε κάθε κύκλο i (όπου $i \in [1, 12]$), εκτελούνται τα εξής βήματα:

Βήμα 1: Υπολογισμός F0 Function

Η F0 function λαμβάνει ως είσοδο το πιο σημαντικό 32-bit τμήμα του L_reg:

```
v_in_data := L_reg(127 downto 96);  --                  32 bits
v_in_rk   := CON_128((gfn_cnt-1)*2); --                 CON
```

Ο υπολογισμός της F0 περιλαμβάνει:

a) **XOR με Round Key**:

$$v\_tmp = v\_in\_data \oplus v\_in\_rk \tag{13}$$

b) **Διαχωρισμός σε 4 bytes**:

$$b_0 = v\_tmp[31:24] \tag{14}$$
$$b_1 = v\_tmp[23:16] \tag{15}$$
$$b_2 = v\_tmp[15:8] \tag{16}$$
$$b_3 = v\_tmp[7:0] \tag{17}$$

c) **S-Box Substitution** (S0, S1, S0, S1):

```
r := to_integer(unsigned(b0(7 downto 4))); --
c := to_integer(unsigned(b0(3 downto 0))); --
f0_pipe(31 downto 24) <= S0(r,c);

r := to_integer(unsigned(b1(7 downto 4)));
c := to_integer(unsigned(b1(3 downto 0)));
f0_pipe(23 downto 16) <= S1(r,c);

r := to_integer(unsigned(b2(7 downto 4)));
c := to_integer(unsigned(b2(3 downto 0)));
f0_pipe(15 downto 8) <= S0(r,c);

r := to_integer(unsigned(b3(7 downto 4)));
c := to_integer(unsigned(b3(3 downto 0)));
f0_pipe(7 downto 0) <= S1(r,c);

```

Κάθε byte μετατρέπεται σε δύο nibbles (4-bit): το ανώτερο nibble δίνει τη γραμμή του S-box, το κατώτερο τη στήλη.

d) **Matrix Multiplication M0**:

Ο πολλαπλασιασμός με τον πίνακα M_0 εκτελείται inline στο Process 6:

```
--                  4 bytes                 S-box
b(0) := f0_pipe(31 downto 24);
b(1) := f0_pipe(23 downto 16);
b(2) := f0_pipe(15 downto 8);
b(3) := f0_pipe(7 downto 0);

--                                                                  GF
    (2^8)
for i in 0 to 3 loop
    --                                          2
    tmp_byte := b(i)(6 downto 0) & '0';  -- Shift left
    if b(i)(7) = '1' then
        tmp_byte := tmp_byte xor x"1d";  -- Reduction
    end if;
    b_x2(i) := tmp_byte;

    --                                          4 = (x2) * 2
    tmp_byte := b_x2(i)(6 downto 0) & '0';
    if b_x2(i)(7) = '1' then
        tmp_byte := tmp_byte xor x"1d";
    end if;
    b_x4(i) := tmp_byte;

    --                                          6 = x4 XOR x2
    b_x6(i) := b_x4(i) xor b_x2(i);
end loop;

```

Οι πολλαπλασιασμοί στο $GF(2^8)$ γίνονται με shift και XOR. Για πολλαπλασιασμό επί 2:

$$2 \cdot b = \begin{cases} b \ll 1 & \text{if MSB}(b) = 0 \\ (b \ll 1) \oplus 0x1d & \text{if MSB}(b) = 1 \end{cases} \tag{18}$$

Το αποτέλεσμα του πίνακα M_0 υπολογίζεται:

```
res0(0) := b(0) xor b_x2(1) xor b_x4(2) xor b_x6(3);
res0(1) := b_x2(0) xor b(1) xor b_x6(2) xor b_x4(3);
res0(2) := b_x4(0) xor b_x6(1) xor b(2) xor b_x2(3);
res0(3) := b_x6(0) xor b_x4(1) xor b_x2(2) xor b(3);
f0_result := res0(0) & res0(1) & res0(2) & res0(3);

```

Αυτό αντιστοιχεί στον πολλαπλασιασμό:

$$\begin{bmatrix} res_0(0) \\ res_0(1) \\ res_0(2) \\ res_0(3) \end{bmatrix} = M_0 \cdot \begin{bmatrix} b(0) \\ b(1) \\ b(2) \\ b(3) \end{bmatrix} \tag{19}$$

Βήμα 2: Υπολογισμός F1 Function

Παρόμοια με την F0, η F1 λαμβάνει το δεύτερο 32-bit τμήμα:

```
v_in_data := L_reg(63 downto 32);  -- Bits [63:32]
v_in_rk   := CON_128((gfn_cnt-1)*2 + 1); --

```

Η διαφορά με την F0 είναι στη σειρά των S-boxes (S1, S0, S1, S0) και στη χρήση του πίνακα M_1:

```
-- S-box substitution
f1_pipe(31 downto 24) <= S1(r,c);  -- S1          S0
f1_pipe(23 downto 16) <= S0(r,c);  -- S0          S1
f1_pipe(15 downto 8)  <= S1(r,c);
f1_pipe(7 downto 0)   <= S0(r,c);
```

Ο πίνακας M_1 υπολογίζεται με επιπλέον πολλαπλασιασμούς:

```
--                                          M1:
    x8, x10 (xA)
for i in 0 to 3 loop
    tmp_byte := b(i)(6 downto 0) & '0';
    if b(i)(7) = '1' then tmp_byte := tmp_byte xor x"1d"; end if;
    b_x2(i) := tmp_byte;

    tmp_byte := b_x2(i)(6 downto 0) & '0';
    if b_x2(i)(7) = '1' then tmp_byte := tmp_byte xor x"1d"; end if;
    b_x4(i) := tmp_byte;

    tmp_byte := b_x4(i)(6 downto 0) & '0';
    if b_x4(i)(7) = '1' then tmp_byte := tmp_byte xor x"1d"; end if;
    b_x8(i) := tmp_byte;

    b_xA(i) := b_x8(i) xor b_x2(i);  -- x10 = x8 XOR x2
end loop;

res1(0) := b(0) xor b_x8(1) xor b_x2(2) xor b_xA(3);
res1(1) := b_x8(0) xor b(1) xor b_xA(2) xor b_x2(3);
res1(2) := b_x2(0) xor b_xA(1) xor b(2) xor b_x8(3);
res1(3) := b_xA(0) xor b_x2(1) xor b_x8(2) xor b(3);
f1_result := res1(0) & res1(1) & res1(2) & res1(3);
```

Βήμα 3: Ενημέρωση του L_reg (GFN Structure)

Μετά τον υπολογισμό των F0 και F1, το L_reg ενημερώνεται σύμφωνα με τη δομή GFN4:

```
L0 := L_reg(127 downto 96);  --
L1 := L_reg(95 downto 64);   --
L2 := L_reg(63 downto 32);   --
L3 := L_reg(31 downto 0);    --

-- GFN transformation
L_reg <= (L1 xor f0_result) & L2 & (L3 xor f1_result) & L0;
```

Αυτό αντιστοιχεί στο μαθηματικό μοντέλο:

$$T_1' \leftarrow T_1 \oplus F_0(RK_{2i}, T_0) \tag{20}$$

$$T_3' \leftarrow T_3 \oplus F_1(RK_{2i+1}, T_2) \tag{21}$$

$$(T_0, T_1, T_2, T_3) \leftarrow (T_1', T_2, T_3', T_0) \tag{22}$$

Η περιστροφή (rotation) είναι εμφανής: $L_0 \to L_3$ position, $L_1 \to L_0$, κ.ο.κ.

4.2 Φάση 2: Key Expansion

Μετά την ολοκλήρωση του GFN4,12, η κατάσταση αλλάζει σε `KEY_EXPAND`. Αυτή η φάση διαρκεί 10 κύκλους (exp_cnt = 0 έως 9, συν 1 για τον έλεγχο ολοκλήρωσης).

4.2.1 Κύκλος 0 - Αρχικοποίηση

```
if exp_cnt = 0 then
    --                                  Whitening Keys
    WK0 <= reg_key(127 downto 96);
    WK1 <= reg_key(95 downto 64);
    WK2 <= reg_key(63 downto 32);
    WK3 <= reg_key(31 downto 0);

    --                                                  L
    L_reg <= L_reg(31 downto 0) & L_reg(127 downto 96) &
             L_reg(95 downto 64) & L_reg(63 downto 32);
```

Τα Whitening Keys είναι απλά το αρχικό κλειδί K:

$$(WK_0, WK_1, WK_2, WK_3) = (K_0, K_1, K_2, K_3) \tag{23}$$

Το L περιστρέφεται μία θέση δεξιά (στην ουσία, μετακινείται το L_3 στη θέση του L_0).

4.2.2 Κύκλοι 1-9: Παραγωγή Round Keys

Για κάθε κύκλο $j \in [1, 9]$:

Βήμα 1: Υπολογισμός Temporary Keys με XOR

```
idx := (exp_cnt-1)*4;  --                                RK array

-- XOR        L                        CON
T0_k := L_reg(127 downto 96) xor CON_128(24 + idx);
T1_k := L_reg(95 downto 64)  xor CON_128(24 + idx + 1);
T2_k := L_reg(63 downto 32)  xor CON_128(24 + idx + 2);
T3_k := L_reg(31 downto 0)   xor CON_128(24 + idx + 3);
```

Για παράδειγμα, στον πρώτο κύκλο (exp_cnt $= 1$):

$$idx = 0 \tag{24}$$
$$T_{0k} = L_0 \oplus CON_{128}[24] \tag{25}$$
$$T_{1k} = L_1 \oplus CON_{128}[25] \tag{26}$$
$$T_{2k} = L_2 \oplus CON_{128}[26] \tag{27}$$
$$T_{3k} = L_3 \oplus CON_{128}[27] \tag{28}$$

Βήμα 2: DoubleSwap Function Σ

Το L μετασχηματίζεται με τη συνάρτηση DoubleSwap:

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

Η DoubleSwap Σ ορίζεται ως:

$$Y = X[7:63] | X[121:127] | X[0:6] | X[64:120] \tag{29}$$

όπου $X[a:b]$ σημαίνει bits από τη θέση a έως b.
Αναλυτικά:

- Παίρνει bits [120:64] (57 bits)

- Συνενώνει με bits [6:0] (7 bits)
- Συνενώνει με bits [127:121] (7 bits)
- Συνενώνει με bits [63:7] (57 bits)

Το αποτέλεσμα: $7 + 57 + 7 + 57 = 128$ bits με διαφορετική διάταξη.

Βήμα 3: Conditional XOR με το Key

Σε περιττούς κύκλους (exp_cnt = 2, 4, 6, 8), τα temporary keys κάνουν XOR με το αρχικό κλειδί:

```
if (exp_cnt-1) mod 2 = 1 then  --
   expansion
    T0_k := T0_k xor reg_key(127 downto 96);
    T1_k := T1_k xor reg_key(95 downto 64);
    T2_k := T2_k xor reg_key(63 downto 32);
    T3_k := T3_k xor reg_key(31 downto 0);
end if;
```

Δηλαδή:

$$\text{Για περιττό } i: \quad RK_i \leftarrow (L \oplus CON) \oplus K \tag{30}$$

$$\text{Για άρτιο } i: \quad RK_i \leftarrow L \oplus CON \tag{31}$$

Βήμα 4: Αποθήκευση Round Keys

```
RK(idx)   <= T0_k;
RK(idx+1) <= T1_k;
RK(idx+2) <= T2_k;
RK(idx+3) <= T3_k;
```

Κάθε κύκλος παράγει 4 round keys (128 bits συνολικά). Μετά από 9 κύκλους, έχουμε:

$$9 \times 4 = 36 \text{ round keys } (\text{RK}_0 \text{ έως } \text{RK}_{35}) \tag{32}$$

4.2.3 Πίνακας Key Expansion

Ο πλήρης πίνακας για 128-bit key (από το specification):

$$RK_0|RK_1|RK_2|RK_3 \leftarrow L \oplus (CON_{24}|CON_{25}|CON_{26}|CON_{27}) \tag{33}$$

$$RK_4|RK_5|RK_6|RK_7 \leftarrow \Sigma(L) \oplus K \oplus (CON_{28}|CON_{29}|CON_{30}|CON_{31}) \tag{34}$$

$$RK_8|RK_9|RK_{10}|RK_{11} \leftarrow \Sigma^2(L) \oplus (CON_{32}|CON_{33}|CON_{34}|CON_{35}) \tag{35}$$

$$\vdots \tag{36}$$

$$RK_{32}|RK_{33}|RK_{34}|RK_{35} \leftarrow \Sigma^8(L) \oplus (CON_{56}|CON_{57}|CON_{58}|CON_{59}) \tag{37}$$

4.3 Παράδειγμα Υπολογισμού

Έστω το test vector από το specification:

$$\text{key} = \texttt{ffeeddcc bbaa9988 77665544 33221100}\ L\ = \texttt{8f89a61b 9db9d0f3 93e65627 da0d027e} \tag{38}$$

Κύκλος 1 $(\mathbf{exp}_cnt = 1, idx = 0)$: $Αυταποθηκεονταιως$: $(RK_0, RK_1, RK_2, RK_3) =$ $(\texttt{f3e6cef9}, \texttt{8df75e38}, \texttt{41c06256}, \texttt{640ac51b})$(39) Το L μετασχηματίζεται με DoubleSwap και η διαδικασία συνεχίζεται.

5 Data Processing - Αναλυτική Περιγραφή

Το Data Processing εκτελείται μετά την ολοκλήρωση του Key Scheduling και περιλαμβάνει τρία στάδια:

5.1 Initial Whitening (INIT$_W HITE$)

Το plaintext υφίσταται XOR με τα whitening keys:

```
if reg_mode = '0' then  --
T0 <= reg_plaintext(127 downto 96);           -- P0
T1 <= reg_plaintext(95 downto 64) xor WK0;    -- P1 XOR WK0
T2 <= reg_plaintext(63 downto 32);            -- P2
T3 <= reg_plaintext(31 downto 0) xor WK1;     -- P3 XOR WK1
```

Για κρυπτογράφηση (encrypt mode):

$$T_0 \leftarrow P_0 \; T_1 \qquad \leftarrow P_1 \oplus WK_0 \; T_2 \leftarrow P_2 \; T_3 \qquad \leftarrow P_3 \oplus WK_1 \tag{40}$$

Για αποκρυπτογράφηση (decrypt mode), χρησιμοποιούνται τα WK2 και WK3:

```
else  --
T0 <= reg_plaintext(95 downto 64) xor WK2;    -- C1 XOR WK2
T1 <= reg_plaintext(63 downto 32);            -- C2
T2 <= reg_plaintext(31 downto 0) xor WK3;     -- C3 XOR WK3
T3 <= reg_plaintext(127 downto 96);           -- C0
```

5.2 Round Processing (ROUNDS)

Η κατάσταση ROUNDS εκτελεί 18 γύρους (round$_c nt = 0 ως 17). Κθεγρος λειτουργεσεδοφσεις λγωτουpipe$

5.2.1 Φάση 0: Υπολογισμός F-Functions

Όταν phase = '0', οι F0 και F1 υπολογίζονται στους καταχωρητές pipeline (Processes 4 και 5). **Για Κρυπτογράφηση (mode = '0'):**

```
-- F0 input
v_in_data := T0;
v_in_rk   := RK(round_cnt*2);
-- F1 input
v_in_data := T2;
v_in_rk   := RK(round_cnt*2 + 1);
```

Οι είσοδοι για τον γύρο ii i είναι:

$$F_0 : \quad T_0, \quad RK_{2i} \tag{41}$$

$$F_1 : \quad T_2, \quad RK_{2i+1} \tag{42}$$

Για Αποκρυπτογράφηση (mode = '1'): Τα round keys χρησιμοποιούνται σε αντίστροφη σειρά:

```
v_in_data := T3;  --          T0
rk_idx := (18 - 1 - round_cnt) * 2;
v_in_rk := RK(rk_idx);
```

Για παράδειγμα, στον πρώτο γύρο αποκρυπτογράφησης (round$_c nt = 0$) : $rk_i dx = (17-0) \times 2 = 34 \Rightarrow RK_{34} (43)$

5.2.2 Φάση 1: Ενημέρωση State

Όταν phase = '1' και $\text{step}_en =' 1', ταα ποτελσματατων νF-functions(f0_pipe, f1_pipe) χρησιμοποιονται γ$
Για Κρυπτογράφηση:

5.3 Final Whitening (FINAL$_W HITE$)

Μετά τον τελευταίο γύρο, εφαρμόζεται το τελικό whitening:

```
if reg_mode = '0' then  --
T0 <= T3;            -- Swap
T1 <= T0 xor WK2;    -- XOR      WK2
T2 <= T1;            -- Pass through
T3 <= T2 xor WK3;    -- XOR      WK3
```

Για κρυπτογράφηση:

$$C_0 \leftarrow T_3 \; C_1 \qquad \leftarrow T_0 \oplus WK_2 \; C_2 \leftarrow T_1 \; C_3 \qquad \leftarrow T_2 \oplus WK_3 \tag{44}$$

Για αποκρυπτογράφηση, χρησιμοποιούνται WK0 και WK1:

```
else  --
T0 <= T0;            --            swap
T1 <= T1 xor WK0;    -- XOR      WK0
T2 <= T2;
T3 <= T3 xor WK1;    -- XOR      WK1
```

5.4 Output (OUTPUT)

Στην κατάσταση OUTPUT, το αποτέλεσμα καταχωρείται στην έξοδο:

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

Το ciphertext είναι η συνένωση των τεσσάρων 32-bit τμημάτων:

$$C = T_0 | T_1 | T_2 | T_3 \quad (128 \text{ bits}) \tag{45}$$

6 Παράδειγμα Πλήρους Εκτέλεσης

Ας εξετάσουμε το test vector:

```
key:       ffeeddcc bbaa9988 77665544 33221100
plaintext: 00010203 04050607 08090a0b 0c0d0e0f
ciphertext: de2bf2fd 9b74aacd f1298555 459494fd
```

6.1 Key Schedule Results

Από το specification, το L παράγεται ως:

$$L = \texttt{8f89a61b 9db9d0f3 93e65627 da0d027e} \tag{46}$$

Whitening keys:

$$WK_0 = \texttt{ffeeddcc}\ WK_1 \quad = \texttt{bbaa9988}\ WK_2 = \texttt{77665544}\ WK_3 \quad = \texttt{33221100} \tag{47}$$

Τα πρώτα round keys:

$$RK_0 = \texttt{f3e6cef9}\ RK_1 \quad = \texttt{8df75e38}\ RK_2 = \texttt{41c06256}\ RK_3 \quad = \texttt{640ac51b} \tag{48}$$

6.2 Round 1 Execution

Initial State (μετά Initial Whitening):

$$T_0 = \texttt{00010203}\ T_1 \quad = \texttt{04050607} \oplus \texttt{ffeeddcc} = \texttt{fbebdbcb}\ T_2 = \texttt{08090a0b}\ T_3 \quad = \texttt{0c0d0e0f} \oplus \texttt{bbaa9988} \tag{49}$$

F0 Calculation:

$$\text{Input} = T_0 = \texttt{00010203}\ \text{RK} \quad = RK_0 = \texttt{f3e6cef9}\ \text{After XOR} = \texttt{f3e7ccfa} \tag{50}$$

Bytes: `f3`, `e7`, `cc`, `fa` S-box outputs (από Table 1):

$$S_0(\texttt{f3}) = \texttt{29}\ S_1(\texttt{e7}) \quad = \texttt{02}\ S_0(\texttt{cc}) = \texttt{46}\ S_1(\texttt{fa}) \quad = \texttt{e1} \tag{51}$$

Μετά το Matrix M0M$_0M0$: `547a3193`

F1 Calculation:

$$\text{Input} = T_2 = \texttt{08090a0b}\ \text{RK} \quad = RK_1 = \texttt{8df75e38}\ \text{After XOR} = \texttt{85fe5433} \tag{52}$$

Μετά S-boxes και M1M$_1M1$: `abf12070`

State Update:

$$T_1' = T_1 \oplus F_0 = \texttt{fbebdbcb} \oplus \texttt{547a3193} = \texttt{af91ea58}\ T_3' \quad = T_3 \oplus F_1 = \texttt{b7a79787} \oplus \texttt{abf12070} = \texttt{1c56b7f} \tag{53}$$

After Rotation:

$$T_0 \leftarrow T_1' = \texttt{af91ea58}\ T_1 \quad \leftarrow T_2 = \texttt{08090a0b}\ T_2 \leftarrow T_3' = \texttt{1c56b7f7}\ T_3 \quad \leftarrow T_0 = \texttt{00010203} \tag{54}$$

Αυτό συμφωνεί με το intermediate value στο specification (Round 2 input).

7 Συμπεράσματα

7.1 Κρίσιμα Σημεία Υλοποίησης

1. **Πλήρης Σύγχρονη Λογική**: Όλα τα signals ενημερώνονται σε rising$_edge(clk)$**Pipeline για F-**
   $\Delta οξεχωριστοprocesses(4και5)υπολογζουνF0καιF1παρλληλα$
2.
3. **Inline Matrix Operations**: Οι πολλαπλασιασμοί M_0 και M_1 δεν είναι functions αλλά inline code για ταχύτητα
4. **Two-Phase Round Execution**: Κάθε γύρος χρειάζεται 2 cycles (phase 0 και 1)
5. **Mode-Dependent Logic**: Διαφορετική λογική για encrypt/decrypt modes

Φάση	**Clock Cycles**
KEY_GFN (12 rounds $\times\times 2phases$)	24
KEY_EXPAND (10 iterations)	10
INIT_WHITE	1
ROUNDS (18 rounds $\times\times 2phases$)	36
FINAL_WHITE	1
OUTPUT	1
Σύνολο	**73 cycles**

Table 1: Χρονική ανάλυση CLEFIA-128

7.2 Χρονική Ανάλυση

Συνολικός χρόνος για μία κρυπτογράφηση (128-bit key):
Για συχνότητα λειτουργίας f=100f = 100 f=100 MHz:

$$\text{Latency} = \frac{73}{100 \times 10^6} = 730 \text{ ns} \tag{55}$$

$$\text{Throughput} = \frac{128 \text{ bits}}{730 \text{ ns}} \approx 175.3 \text{ Mbps} \tag{56}$$

7.3 Πλεονεκτήματα Υλοποίησης

- Πλήρως σύγχρονη σχεδίαση - εύκολη για timing closure
- Χαμηλή κατανάλωση λόγω register-based design
- Επεκτάσιμη για 192/256-bit keys (χρειάζονται μόνο αλλαγές στο key scheduling)
- Μικρό footprint - inline operations αντί για functions

7.4 Πιθανές Βελτιστοποιήσεις

1. **Loop Unrolling**: Υπολογισμός πολλών rounds παράλληλα
2. **Pipelining**: Επεξεργασία πολλών blocks ταυτόχρονα
3. **LUT-based S-boxes**: Χρήση Block RAM για γρηγορότερη πρόσβαση
4. **Separate Key Schedule Unit**: Παράλληλη εκτέλεση key schedule και data processing