Μάθημα: ΣΧΕΔΙΑΣΜΟΣ ΣΥΣΤΗΜΑΤΩΝ VLSI

Εξάμηνο: Εαρινό

Στοιχεία Μελών Ομάδας :

Σωτηρόπουλος Διονύσης-Οδυσσέας, Γεώργιος, 5661, 6ο έτος

Εισαγωγή :

Στα πλαίσια αυτής της εργασίας καλλούμαστε να υλοποιήσουμε το σύστημα ασφαλείας Minalpher v1.1 σε κώδικα VHDL.

Περιγραφή Συστήματος

Το Minalpher είναι ένα σύστημα κρυπτογράφησης και αποκρυπτογράφησης που υποστηρίζει δύο ξεχωριστές βασικές λειτουργίες κρύπο/αποκρυπτογράφησης.

1. Authenticated Encryption with Associated Data (AEAD)
2. Message Authentication Code (MAC)

Και οι δύο αυτές λειτουργίες βασίζονται στον αλγόρυθμο κρυπτογράφησης Tweakable Even-Mansour του οποίου η λειτουργία θα αναλυθέι παρακάτω.

Εργαλεία και Σημειώσεις

Για την υλοποίηση του Minalpher χρησιμοποιήσαμε το πρόγραμμα Modelsim v6.4 SE για την μεταγλώτηση των αρχείων .vhd και την προσομοίωση.

Για την σύνθεση χρησιμοποιήθηκε το πρόγραμμα Vivado v2018.1 (αναμένεται)

Η εργασία βασίζεται στο paper Minalpher v1.1 των σχεδιαστών Yu Sasaki, Yosuke Todo, Kazumaro Aoki, Yusuke Naito, Takeshi Sugawara, Yumiko Murakami, Mitsuru Matsui και Shoichi Hirose.

Η ολοκλήρωση του συστήματος του Minalpher είναι ακόμα ημιτελής. Σε αυτή την αναφορά περιλαμβάνεται η βασική συνάρτηση κρυπτογράφησης P, η αντίστροφή της P^{-1} καθώς και επιπλέον ολοκληρωμένα για την υλοποίηση των αλγόρυθμων κρυπτογράφησης (TEM_ENC) και αποκρυπτογράφησης (TEM_DEC).

Σημειώνεται οτι οι μέσα στα χρονικά πλαίσια για την ολοκλήρωση της εργασίας και του γνωσιακού επιπέδου οι παραπάνω αλγόρυθμοι ολοκληρώθηκαν ανεπιτυχώς και (προφανώς) χωρίς βελτιστοποιήσεις.

Υλοποιήσεις :

Σε αυτό το μέρος της αναφοράς θα περιγράψουμε τα ολοκληρωμένα προς σχεδίαση και τα βήματα που λήφθηκαν. Οι υλοποιήσεις απλών συστημάτων πρώτου και (μερικών) δευτέρου επιπέδου δεν περιλαμβάνονται. Θα εστιάσουμε στα ολοκληρωμένα υψηλότερου επιπέδου (τουλάχιστον 4ου) καθώς και ορισμένα αξιοσημείωτα συστήματα/εξαρτήματα των προαναφερόμενων.

TWEAKABLE EVEN MANSOUR (TEM)

Οι αλγόρυθμοι TEM βασίζονται σε μία συνάρτηση παραλλαγής (permutation) Minalpher_P (P).

Η συνάρτηση αυτή αποτελείται από 17+1 γύρους ανάμιξης της πληροφορίας εισόδου μέσα από ένα σύστημα Substitution-Permutation-Xor.

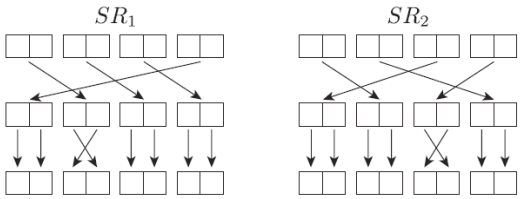
Συνοπτικά η συνάρτηση P δουλεύει ως εξής.

Η είσοδος X χωρίζεται σε δύο πίνακες 4x32 bits A , B.

1. Οι τετράδες bits (nibbles) αντικαθιστούνται ανάλογα με την αντιστρέψιμη substitution συνάρτηση substitute Nibbles (SN).

x	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xA	0xB	0xC	0xD	0xE	0xF
$s(x)$	0xB	0x3	0x4	0x1	0x2	0x8	0xC	0xF	0x5	0xD	0xE	0x0	0x6	0x9	0xA	0x7

2. Οι γραμμές κάθε πίνακα αναμιγνύονται σύμφωνα με την συνάρτηση Shuffle Rows (SR)



3. Εναλλαγή των δύο πινάκων (SM)
4. Xor του (καινούριου) πίνακα B με τον A (XM)
5. Ανάμειξη στηλών του κάθε πίνακα σύμφωνα με τη συνάρτηση Mix Columns (MC)

$$A^{MC}[0][j] \leftarrow A[0][j] \oplus A[1][j] \oplus A[3][j], \quad 0 \le j < 8,$$
$$A^{MC}[1][j] \leftarrow A[1][j] \oplus A[2][j] \oplus A[0][j], \quad 0 \le j < 8,$$
$$A^{MC}[2][j] \leftarrow A[2][j] \oplus A[3][j] \oplus A[1][j], \quad 0 \le j < 8,$$
$$A^{MC}[3][j] \leftarrow A[3][j] \oplus A[0][j] \oplus A[2][j], \quad 0 \le j < 8,$$

$$B^{MC}[0][j] \leftarrow B[0][j] \oplus B[1][j] \oplus B[3][j], \quad 0 \le j < 8,$$
$$B^{MC}[1][j] \leftarrow B[1][j] \oplus B[2][j] \oplus B[0][j], \quad 0 \le j < 8,$$
$$B^{MC}[2][j] \leftarrow B[2][j] \oplus B[3][j] \oplus B[1][j], \quad 0 \le j < 8,$$
$$B^{MC}[3][j] \leftarrow B[3][j] \oplus B[0][j] \oplus B[2][j], \quad 0 \le j < 8.$$

6. Xor του πίνακα B με τη σταθερά γύρου. (E)

$r \oplus 0$	$r \oplus 1$	$r \oplus 2$	$r \oplus 3$	0	0	0	0
$r \oplus 1$	$r \oplus 0$	$r \oplus 3$	$r \oplus 2$	0	0	0	0
$r \oplus 2$	$r \oplus 3$	$r \oplus 0$	$r \oplus 1$	0	0	0	0
$r \oplus 3$	$r \oplus 2$	$r \oplus 1$	$r \oplus 0$	0	0	0	0

Η διαδικασία αυτή συνεχίζεται με τους δύο πίνακες στην έξοδο του γύρου να μπαίνουν ως είσοδοι στον επόμενο γύρο.

Υλοποίηση της συνάρτησης P

Components

1. SN – Nibble Substitution

   Το εξάρτημα αυτό αντικαθιστά τετράδες bit εισόδου ανάλογα με την δεκαεξαδική τους τιμή.

   > Για την υλοποίηση του χρησιμοποιήθηκαν πύλες and και or που προέκυψαν από την τεχνική Karnaugh. Ευκολότερη υλοποίηση behavioral δοκιμάστηκε αλλά επέφερε προβλήματα κατά το simulation.

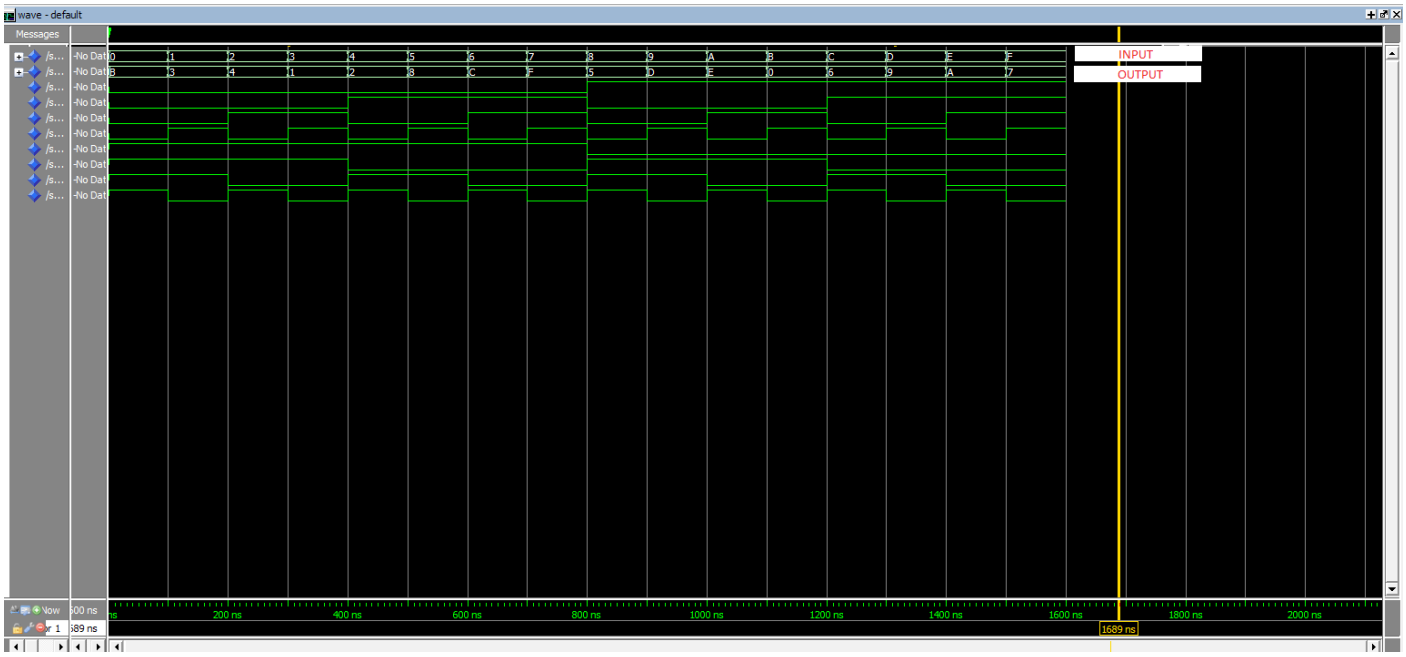
2. SR, MC, SM – Shuffle Rows, Mix Columns, Swap Matrices

   Η υλοποίηση των συγκεκριμένων έγινε απλά με οδήγηση των εισόδων στις σωστές εξόδους.

3. XM και E(RC) υλοποιήθηκαν με πύλες xor.

4. RC – Round Constant

   Η υλοποίηση της σταθεράς γύρου υλοποιήθηκε με xor πύλες.



(Προσ. 1. SN: **Προσομοίωση λειτουργίας SN**)

Τα παραπάνω εξαρτήματα συνδέονται με τη σειρά SR→SR→ SM→ XM → MC→ RC για να συνθέσουν το σύστημα που υλοποιεί τη συνάρτηση γύρου της P και δέχεται ως είσοδο 256 bits και μία σταθερά γύρου r και βγάζει στην έξοδο “μπερδεμμένα” 256 bits.

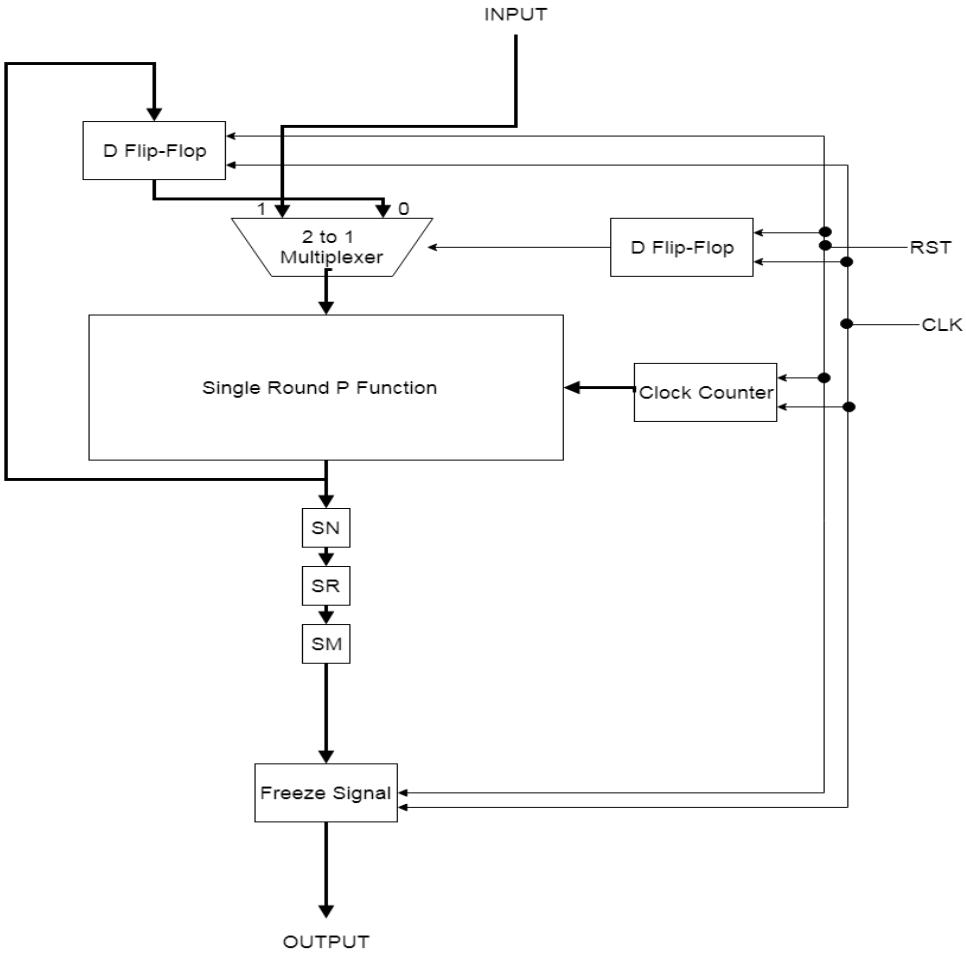
Έχοντας ολοκληρώσει το κομμάτι του γύρου και προκειμένου να υλοποιήσουμε ολόκληρη τη συνάρτηση P θα πρέπει να κατασκευάσουμε ένα κύκλωμα που θα εκτελεί την ίδια διαδικασία 17 φορές με την έξοδο κάθε γύρου ώς είσοδο του επόμενου. Σημειώνεται οτι κατα την ολοκλήρωση των 17 γύρων η συνάρτηση εκτελέι έναν επιπλέον “μισό” γύρο ώς εξής:

$$X_{r+1} \leftarrow T \circ S(X_r),$$

Όπου X_r η έξοδος του 17^{ου} γύρου, S η SN και T η SR και SM

Για την υλοποίηση της Συνάρτησης P για 17+1 γύρους (17+1 P Procedure) χρησιμοποιήθηκαν επιπλέον

- κυκλώματα πολυπλέχτη 2 σε 1
- Μετρητής ρολογιού
- D flip-flops καθώς
- τα προηγούμενα εξαρτήματα SN και SR.
- Επιπλέον, προκειμένου να “παγώσουμε” την έξοδο στους 17+1 γύρους σχεδιάσαμε ένα επιπλέον ολοκληρωμένο freeze_signal.



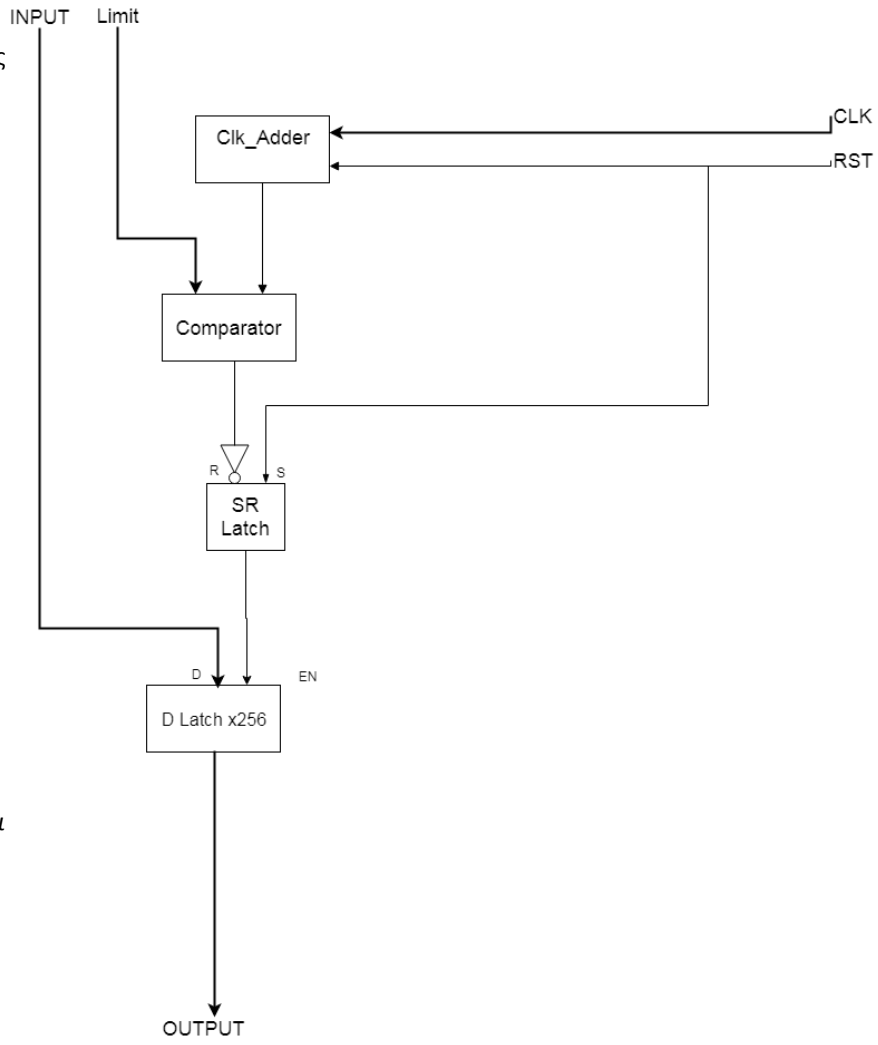
Όπως φαινεται στο διπλανό σχήμα. Η είσοδος INPUT επιλέγεται κατα το reset να εισέλθει στην συνάρτηση γύρου P η οποία είναι ασύγρονη και δέχεται επιπλέον είσοδο τον αριθμο του γύρου από τον μετρητή clock_counter. Αφού το σύστημα ενεργοποιηθεί. (rst→ 0) σε κάθε κλυκλο ρολογιού η έξοδος της P ξαναμπαίνει στην είσοδο. Τέλος όταν ενεργοποιηθεί το εσωτερικό σήμα του Freeze_Signal η τελική έξοδος του συστήματος παγώνει στην 17 επανάληψη.

(Σχ. 1. P_Process_17+1_Rounds)

Clock Counter και Freeze Signal (Σχ. 2. Freeze_Signal)

Ιδιαίτερα προκλητική ήταν η προσπάθεια να κατασκευάσουμε κυκλώματα άθροισης ρολογιού και “παγώματος” σημάτων. Έγιναν διάφορες structural προσπάθειες με d-ff και πύλες and, με σειριακά jk-ff καθώς και με αθροιστές αλλά τελικά ο μετριτής ρολογιού (clock_adder) υλοποιήθηκε με behavioural κώδικα.

Το ολοκληρωμένο Freeze Signal που φαίνεται στην εικόνα δέχεται εισόδους 256 bits INPUT και ένα 6 bit Limit και επιστρέφει την τιμή που έιχε το INPUT όταν ο εσωτερικός αθροιστής ρολογιού έφτασε την τιμή limit.



Περεταίρω επεξήγηση: Όπως φαίνεται στο σχήμα, ο συγκριτής (Comparator) συγκρίνει την είσοδο limit με την έξοδο του μετρητή ρολογιού και επιστρέφει “0” όταν η τιμές ταυτίζονται. *Σημειώνεται οτι αυτό θα διαρκέσει για ένα μόνο κύκλο ρολογιού καθώς ο Clk_Adder θα συνεχίσει να αυξάνει το ρολόι.*

Η έξοδος του συγκριτή αντιστρέφεται και μπαίνει στην έισοδο R ένός SR latch με το RST στην S.

Με αυτόν τον τρόπο το Latch αρχικοποιείται με είσόδους “01” επιστρέφοντας “1” μέχρι τον κύκλο ρολογιού όπου ο μετρητής φτάνει το Limit όπου και θα πάρει την τιμή “0” στην έξοδο. Επειδή ο μετριτής θα συνεχίσει να μετράει οι είσοδοι, από τον επόμενο κύκλο ρολογιού και μετά θα είναι πάντα* “00” άρα η έξοδος θα έχει παγώσει στο “0”!!!

Το μόνο που μένει είναι να συνδέσουμε την έισοδο INPUT με d Latches και να οδηγήσουμε την έξοδο του SR στην είσοδο EN. Τώρα η έξοδος OUTPUT του κυκλώματος θα αλλάζει ανάλογα με το INPUT μέχρι η έξοδος του SR Latch να μηδενιστεί απ’ οπου και στο εξής η έξοδος θα παγώσει.

(ΣΗΜΕΪΩΣΗ, ο μετριτής μετράει από το 0 οπότε για 17 γύρους, Limit = 00010000)

**(επειδή ο μετρητής ρολογιού επιστρέφει 6 bits θα μετρήσει μέχρι το 2^6-1 από όπου και θα ξαναρχίσει να μετράει από το 0. Επομένως το OUTPUT θα ξανα αλλάξει αφού παγώσει για πρώτη φορά.)*

TEM Encode/Decode

Algorithm 1 Encryption Algorithm of Tweakable Even-Mansour

procedure TEM_ENC$(K, \mathsf{flag}, N, i, j, M)$
$L \leftarrow (K\|\mathsf{flag}\|N) \oplus P(K\|\mathsf{flag}\|N)$
$C \leftarrow \mathrm{y}^i(\mathrm{y}+1)^j L \oplus P(M \oplus \mathrm{y}^i(\mathrm{y}+1)^j L)$
return C
end procedure

Algorithm 2 Decryption Algorithm of Tweakable Even-Mansour

procedure TEM_DEC$(K, \mathsf{flag}, N, i, j, C)$
$L \leftarrow (K\|\mathsf{flag}\|N) \oplus P(K\|\mathsf{flag}\|N)$
$M \leftarrow \mathrm{y}^i(\mathrm{y}+1)^j L \oplus P^{-1}(C \oplus \mathrm{y}^i(\mathrm{y}+1)^j L)$
return M
end procedure

Οι διαδικασιες κρυπτογράφησης και αποκρυπτογράφησης του TEM, πέρα από το μήνυμα M ή κρυπτογράφημα C δέχονται ως είσοδο ένα κλειδι K, ένα flag ειδικό για τις λειτουργίες AEAD και MAC που προαναφέρθηκαν. Ένα N ανάλογο του padding των blocks των M και C καθώς και αριθμούς i,j που προσδιορίζουν το offset των blocks των M και C.

Η διαδικασία απο-/κρυπτογράφησης αποτελείται από δύο στάδια όπως φαίνεται στην εικόνα.

L που είναι παραγωγή κλειδιού και M ή C που είναι η αποκρυπτογράφηση ή η κρυπτογράφηση του αρχικού μηνύματος αντίστοιχα.

Προκειμένου να υλοποιήσουμε αυτές τις διαδικασίες πρέπει να σχεδιάσουμε έναν πολλαπλασιαστή που θα εκτελεί την πράξη $y^i(y+1)^jA$.

Η σχεδίαση ενός τέτοιου πολλαπλασιαστή ήταν άλλη μία πρόκληση. Η υλοποίηση του y*A δώθηκε σχηματικά από τους σχεδιαστές του Minalpher ως εξής

Using the polynomials, we can compute a multiplication by $\mathtt{y}$ which is used in an offset as follows.

$$(A_{31}\mathtt{y}^{31} + A_{30}\mathtt{y}^{30} + A_{29}\mathtt{y}^{29} + \cdots + A_3\mathtt{y}^3 + A_2\mathtt{y}^2 + A_1\mathtt{y} + A_0) \times \mathtt{y}$$
$$= A_{30}\mathtt{y}^{31} + A_{29}\mathtt{y}^{30} + \cdots + A_3\mathtt{y}^4 + (A_2 + A_{31})\mathtt{y}^3 + (A_1 + A_{31})\mathtt{y}^2 + A_0\mathtt{y} + A_{31}\mathtt{x}$$

As we can see, this multiplication can be done using a left shift with one byte and two byte-wise XOR and one multiplication by $\mathtt{x}$ in $\mathrm{GF}(2^8)$. This is illustrated in Fig. 51. Similar to above computation, we

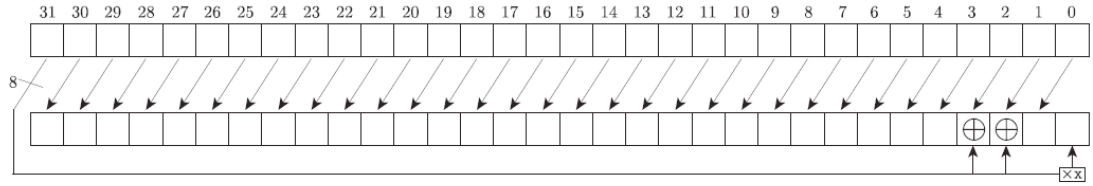


Fig. 51. Multiplication by $\mathtt{y}$

can compute a multiplication by $\mathtt{x}$ as follows.

$$(a_7\mathtt{x}^7 + a_6\mathtt{x}^6 + a_5\mathtt{x}^5 + a_4\mathtt{x}^4 + a_3\mathtt{x}^3 + a_2\mathtt{x}^2 + a_1\mathtt{x} + a_0) \times \mathtt{x}$$
$$= (a_6 + a_7)\mathtt{x}^7 + a_5\mathtt{x}^6 + (a_4 + a_7)\mathtt{x}^5 + a_3\mathtt{x}^4 + a_2\mathtt{x}^3 + a_1\mathtt{x}^2 + (a_0 + a_7)\mathtt{x} + a_7$$

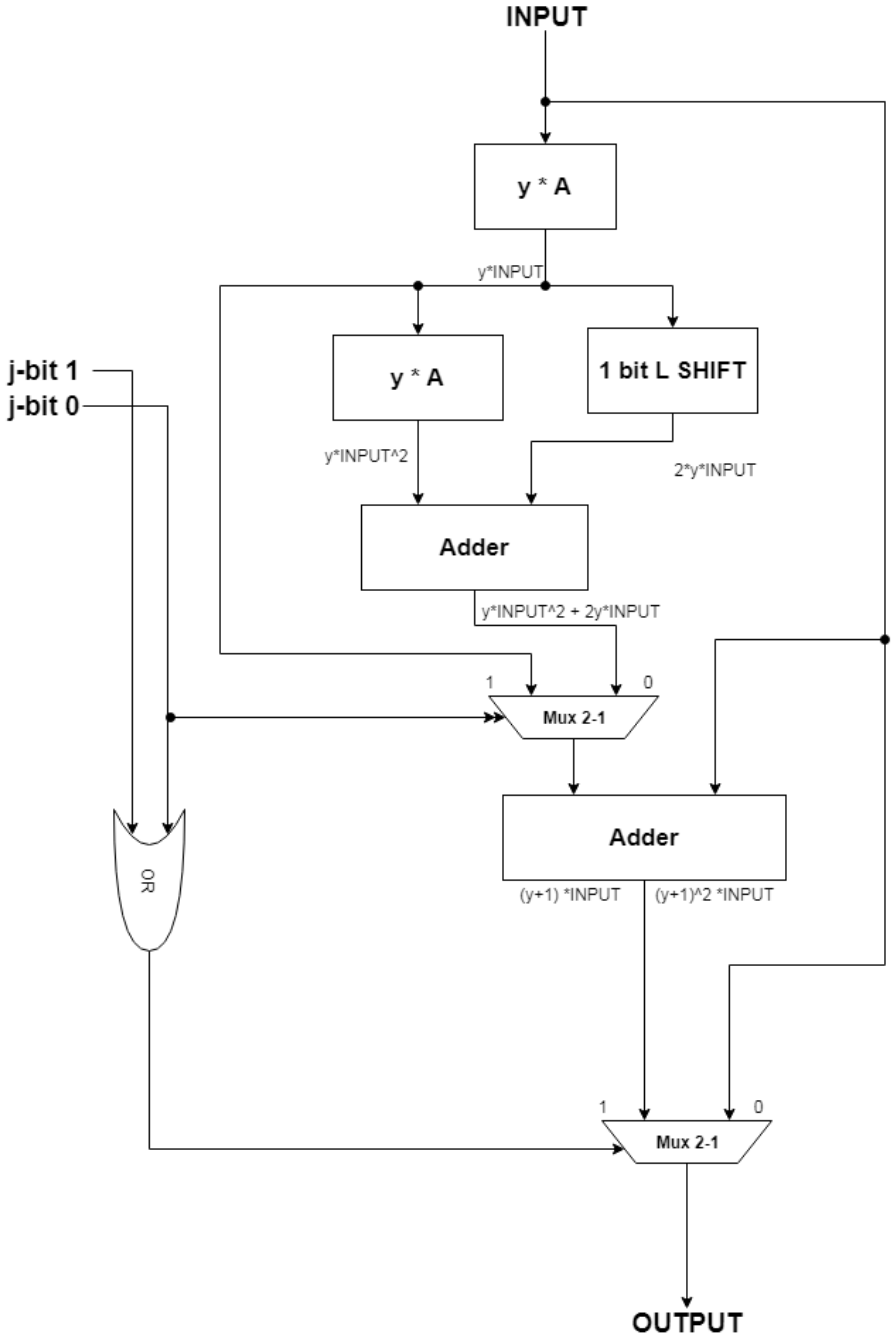
Assume A as an 8-bit string, there is a well-known equation to compute a multiplication by $\mathtt{x}$.

$$A \times \mathtt{x} = (A^{\ll 1}) \oplus (\mathrm{msb}(A)\mathtt{0xA3})$$

Όπως φαίνεται στο σχήμα θα χρειαστεί να κάνουμε shift κατα ένα κελί αριστερά, 2 byte-wise Xor και να κατασκευάσουμε έναν πολλαπλασιαστή 8bits για την πράξη A*x στο LSByte του αποτελέσματος. Για την δύναμη “i” αρκεί να οδηγήσουμε την έξοδο στην είσοδο i-1 φορές.

Η υλοποίηση της πράξης $(y+1)^j * A$ είναι πολύ πιο περίπλοκη. Ωστόσο, ελέγχοντας τις λειτουργίες του Minalpher (AEAD και MAC) βλέπουμε πως το j παίρνει τιμές {0,1,2}. Έτσι αποφεύγουμε την κατασκευή ενός γενικού πολλαπλασιαστή τέτοιου τύπου και αντίθετα υλοποιήσαμε ένα ασύγχρονο κύκλωμα που δέχεται σαν είσοδο A των 256 bits και j των 2 bits βγάζοντας στην έξοδο το αποτέλεσμα:

$Ay^2 + 2Ay + A$ για j=2, $Ay + A$ για j=1 ή A για j=0.

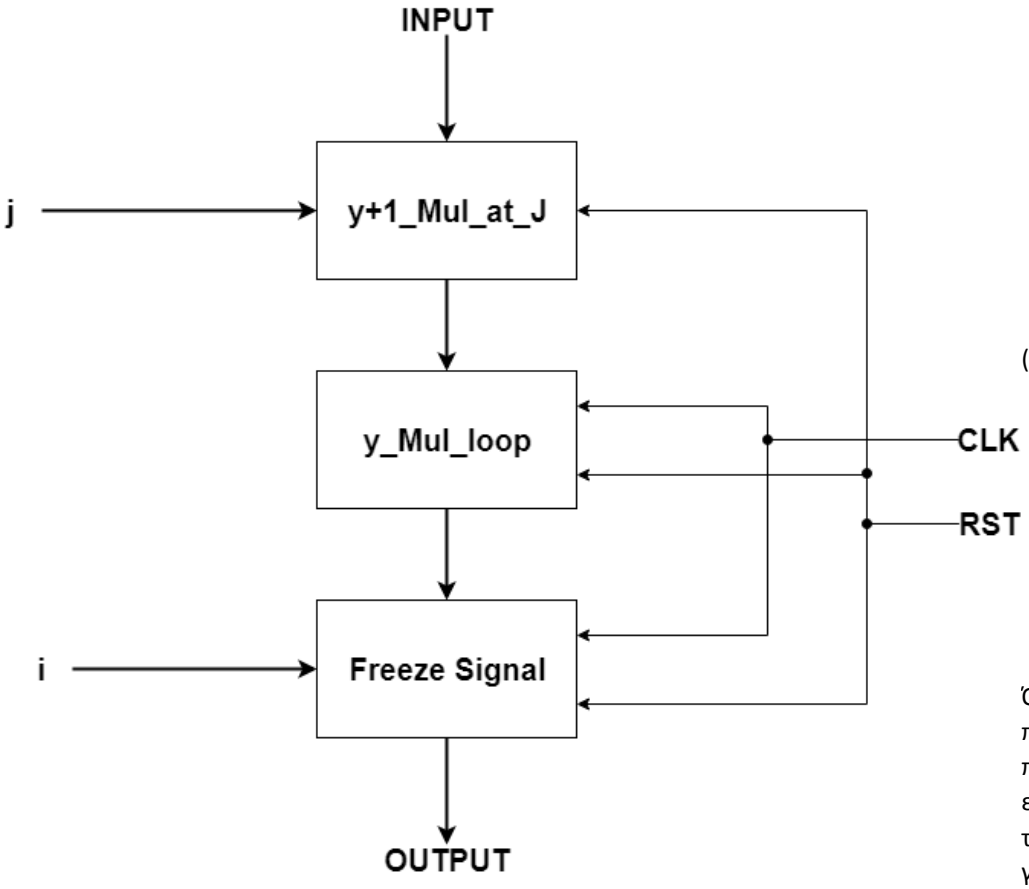


Αξίζει να αναφέρουμε πως τα 2 bit εισόδου υποστηρίζουν την τιμή “3” (11) η οποία στην συγκεκριμένη περίπτωση είναι αδιάφορη/λανθασμένη. Το σύστημα που υλοποιήσαμε θα βγάλει στην έξοδο **Ay + A** και για j=3.

(Σχ. 3. y+1_mul_A_at_j)

(Τα σήματα reset παραλήφθηκαν στο σχήμα)

Έχοντας κατασκευάσει τα δύο components για τους πολλαπλασιασμούς y^i*A και $(y+1)^j$*A, σχεδιάζουμε το ολοκληρωμένο σύστημα πολλαπλασιασμού ως εξής:



(Σχ. 4. Mans_Multiplier)

Όπως φαίνεται στο σχήμα, πέρα από τα 2 προαναφερόμενα εξαρτήματα χρησιμοποιούμε το σύστημα "signal_freeze" για να παγώσουμε την έξοδο μετά από "i" επαναλήψεις της y_Mul_loop καθώς το εξάρτημα δεν φράσσει την ανατροφοδότηση από μόνο του και επαναλαμβάνει όσο μετράει το ρολόι.

Υλοποίηση Κωδικοποιητή Tweakabe Even-Mansour

Έχοντας όλα τα απαραίτητα εξαρτήματα, σε αυτό το σημείο μπορούμε να κατασκευάσουμε τον Κωδικοποιητή όπως περιγράφεται στο paper.

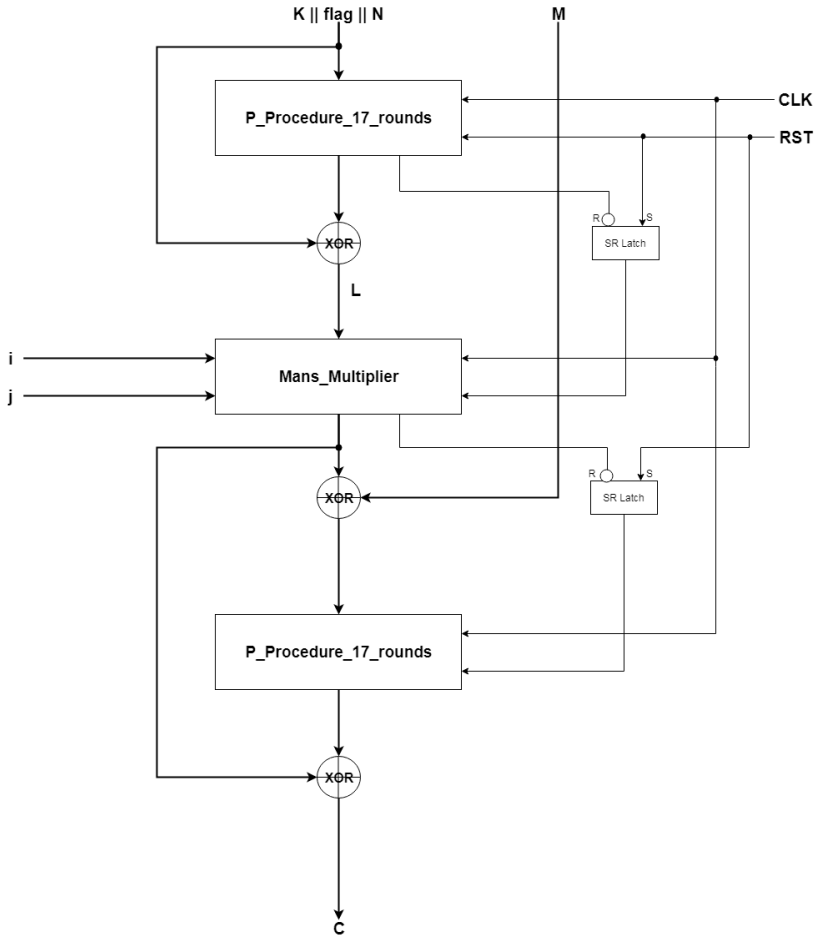
Algorithm 1 Encryption Algorithm of Tweakable Even-Mansour

procedure TEM_Enc$(K, \mathsf{flag}, N, i, j, M)$
  $L \leftarrow (K \| \mathsf{flag} \| N) \oplus P(K \| \mathsf{flag} \| N)$
  $C \leftarrow \mathrm{y}^i(\mathrm{y}+1)^j L \oplus P(M \oplus \mathrm{y}^i(\mathrm{y}+1)^j L)$
  return C
end procedure

Θα χρησιμοποιήσουμε τα εξαρτήματα P_Process_17+1_Rounds, Mans_Multiplier καθώς και πύλες xor.

Ωστόσο το σύστημα θα χρειαστεί επιπλέον κυκλώματα για τον έλεγχο χρονισμού έτσι ωστε το κάθε εξάρτημα να ενεργοποιείται αφού το προηγούμενο έχει εκτελέσει την λειτουργία του.

Προκειμένου να μη χρησιμοποιήσουμε επιπλέον μετρητές ρολογιού, οδηγούμε επιπλέον εξόδους "maxed" από τα εξαρτήματα Freeze Signal των προαναφερόμενων P_Process_17+1_rounds και Mans_Multiplier τα οποία θα οδηγηθουν στις εισόδους reset των επόμενων, πετυχαίνοντας έτσι σειριακή ενεργοποίηση για σωστά αποτελέσματα (Στη θεωρία τουλάχιστον).



(Σχ. 5. TEM_ENC)

Υλοποίηση Αποκωδικοποιητή Tweakable Even-Mansour

Έχοντας ολοκληρώσει την διαδικασία κωδικοποίησης δεν είναι δύσκολο να σχεδιάσουμε και την αντίστροφη διαδικασία.

Algorithm 2 Decryption Algorithm of Tweakable Even-Mansour

procedure TEM_DEC$(K, \mathsf{flag}, N, i, j, C)$
$L \leftarrow (K\|\mathsf{flag}\|N) \oplus P(K\|\mathsf{flag}\|N)$
$M \leftarrow \mathrm{y}^i(\mathrm{y}+1)^j L \oplus P^{-1}(C \oplus \mathrm{y}^i(\mathrm{y}+1)^j L)$
return M
end procedure

Το κύκλωμα είναι σχεδόν πανομοιότυπο εκτός από την συνάρτηση P, που στην συγκεκριμένη περίπτωση πρέπει να σχεδιάσουμε την αντίστροφή της όπως ορίζει το paper.

1.4.1.2 Specification of Minalpher-P^{-1} (Backward Procedure)

In the backward procedure, first the 256-bit input value IN is copied to a 256-bit value X_0 which is an input to the round function. Then, following operations are performed from $i = 1$ to r,

$$X_i \leftarrow R(X_{i-1}, M \circ E(r-i)),$$

where the function R is the same function as that of the forward procedure. Finally, X_{r+1} is calculated from X_r as follows,

$$X_{r+1} \leftarrow T \circ S(X_r),$$

and X_{r+1} is copied to the 256-bit output value OUT. Minalpher-P uses $r = 17$, then Minalpher-P has 17.5 rounds.

Η αντίστροφη διαδικασία P (P_Back_Proc) επιτυγχάνεται χρησιμοποιόντας το ίδιο σύστημα με την P με τις εξής αλλαγές.

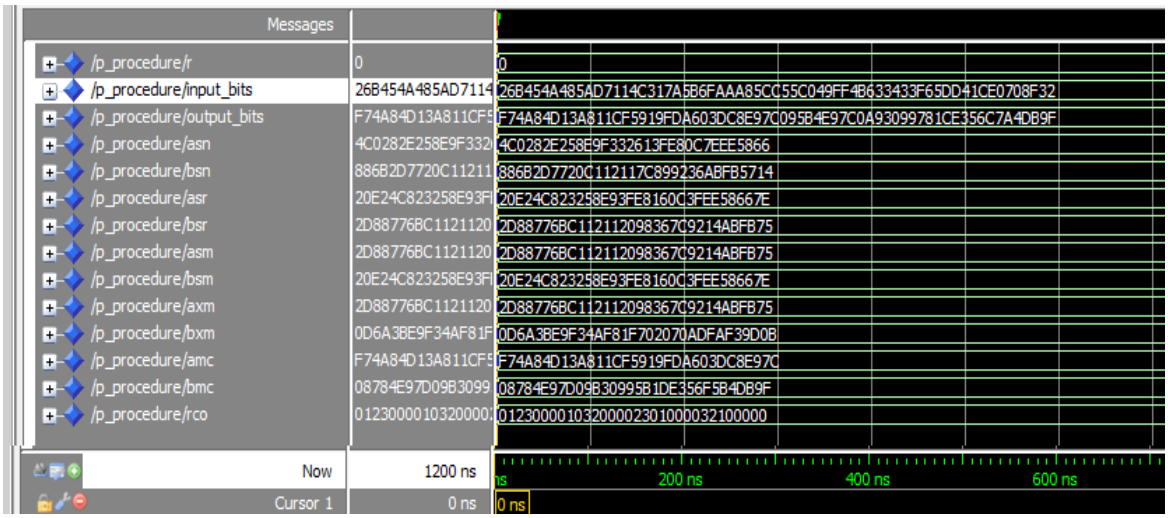
1. Ο μετριτής γύρου μετράει από το 16 στο 0.
2. Η αφού εφαρμοστεί το βήμα E(r-i) οι δύο πίνακες περνάνε μέσα από ένα ακόμη σύστημα M

   (XM→ MC).

   Σημείωση: Η συγκεκριμένες αναπαραστάσεις της 1.4.1.2 καθώς και ανάλογες του paper ήταν δυσανάγνωστες καθώς ήταν πολύ περιεκτικές και χωρίς σχολιασμούς ή επεξηγήσεις. Αρχικά δοκιμάστηκε η RC→M→xor B με λάθος αποτελέσματα.

Αποτελέσματα Προσομοίωσης :

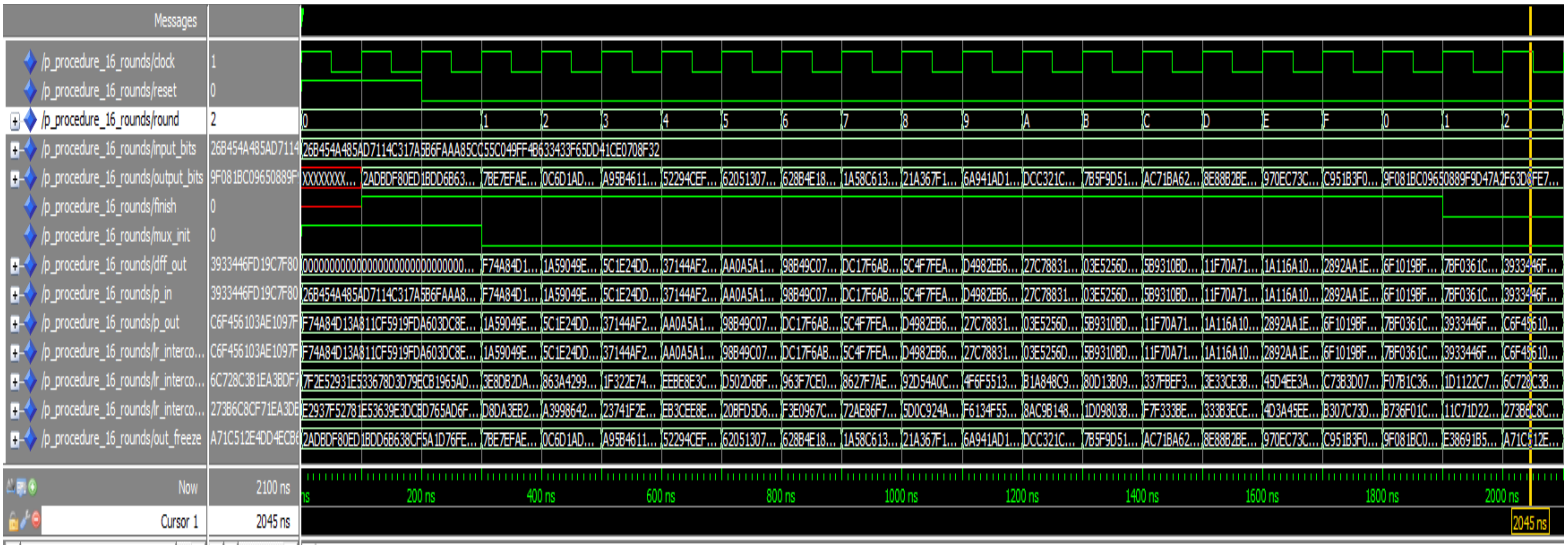
Σε αυτό το μέρος της εργασίας θα παρουσιάσουμε τις κυματομορφές προσομοίωσης των παραπάνω κυκλωμάτων. Στην προσομοίωση, κάθε σύστημα οδηγήθηκε από ένα τυχαίο βέκτορα 256 bits (RAND_INP = 256'h26B454A485AD7114C317A5B6FAAA85CC55C049FF4B633433F65DD41CE0708F32) για κάθε είσοδο 256 bits εκτός αν αναφέρεται κάτι διαφορετικό. Το ρολόι έχει περίοδο 100ns και αλλάζουμε το σήμα reset από το λογικό 1 στο 0 σε t=200ns.



(Προσ. 2. P_Procedure)

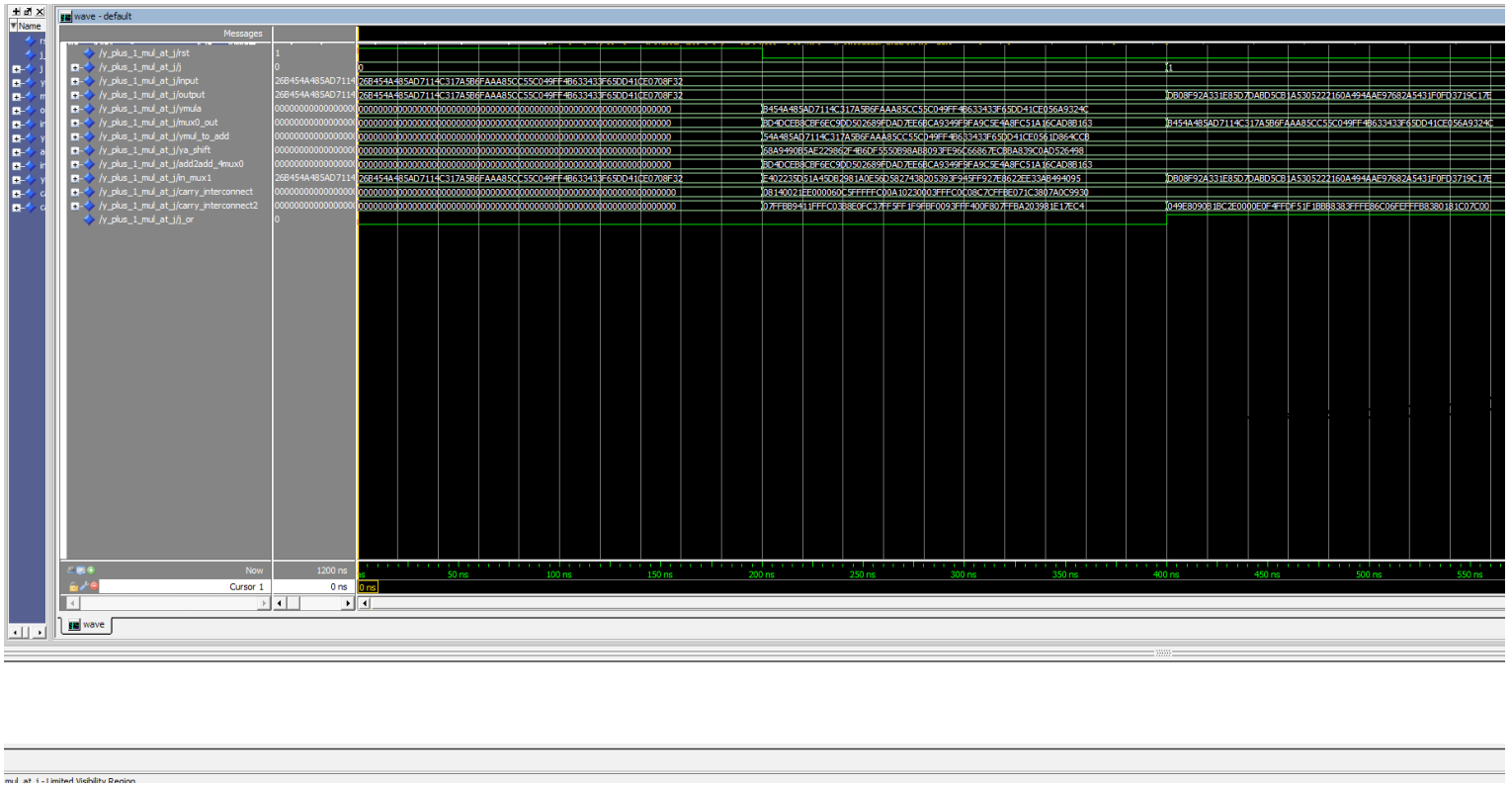
Σχόλια: η P_Procedure ενός γύρου είναι ασύγχρονο κύκλωμα γι αυτό και στην προσομοίωση βγάζει αποτελέσματα από τη χρονική στιγμή 0 καθώς ο simulator δεν λαμβάνει υπόψη το throughput.

Στο σχήμα φαίνονται επίσης οι ενδιάμεσες τιμές τύπου XYZ όπου x={πίνακας A, πίνακας B}, YZ η ενέργεια {Substitute Nibbles, Shuffle Rows κ.ο.κ}.

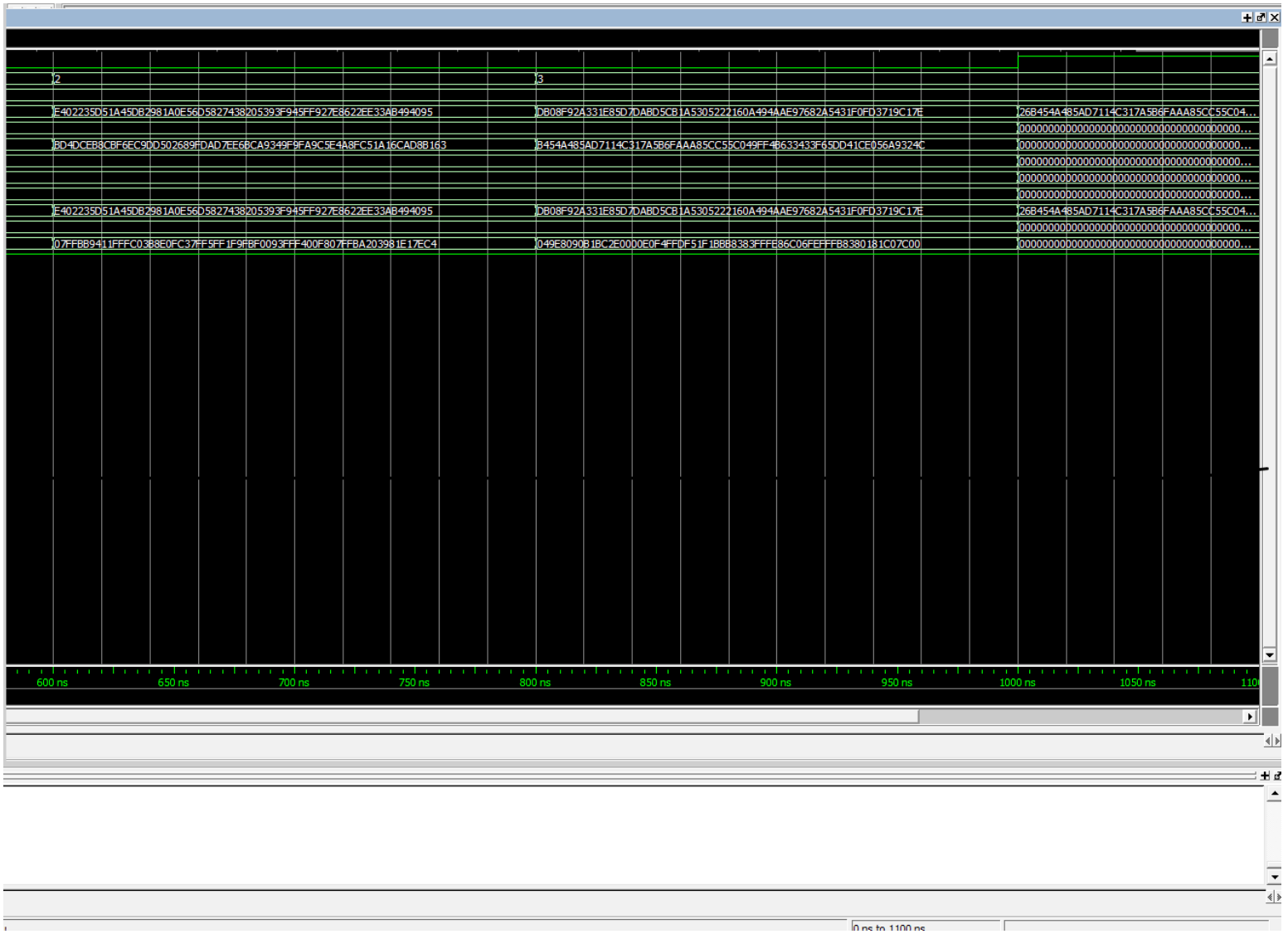


(Προσ. 3. P_Procedure_17+1_Rounds)

Σχόλια: Στο σχήμα φαίνεται ξεκάθαρα η επαναληπτική διαδικασία της P, σε κάθε κύκλο ρολογιού το output αλλάζει τιμή μέρι την στιγμή που ο μετριτής φτάσει την τιμή 16, (στο σχήμα φαίνεται ο μετριτής γύρων με μέγιστη τιμή το F, στον πρώτο μηδενισμό του είναι η τιμή 16) τότε ενεργοποιείται το σήμα freeze και η έξοδος παγώνει ενώ το σύστημα συνεχίζει τη λειτουργία του εσωτερικά.

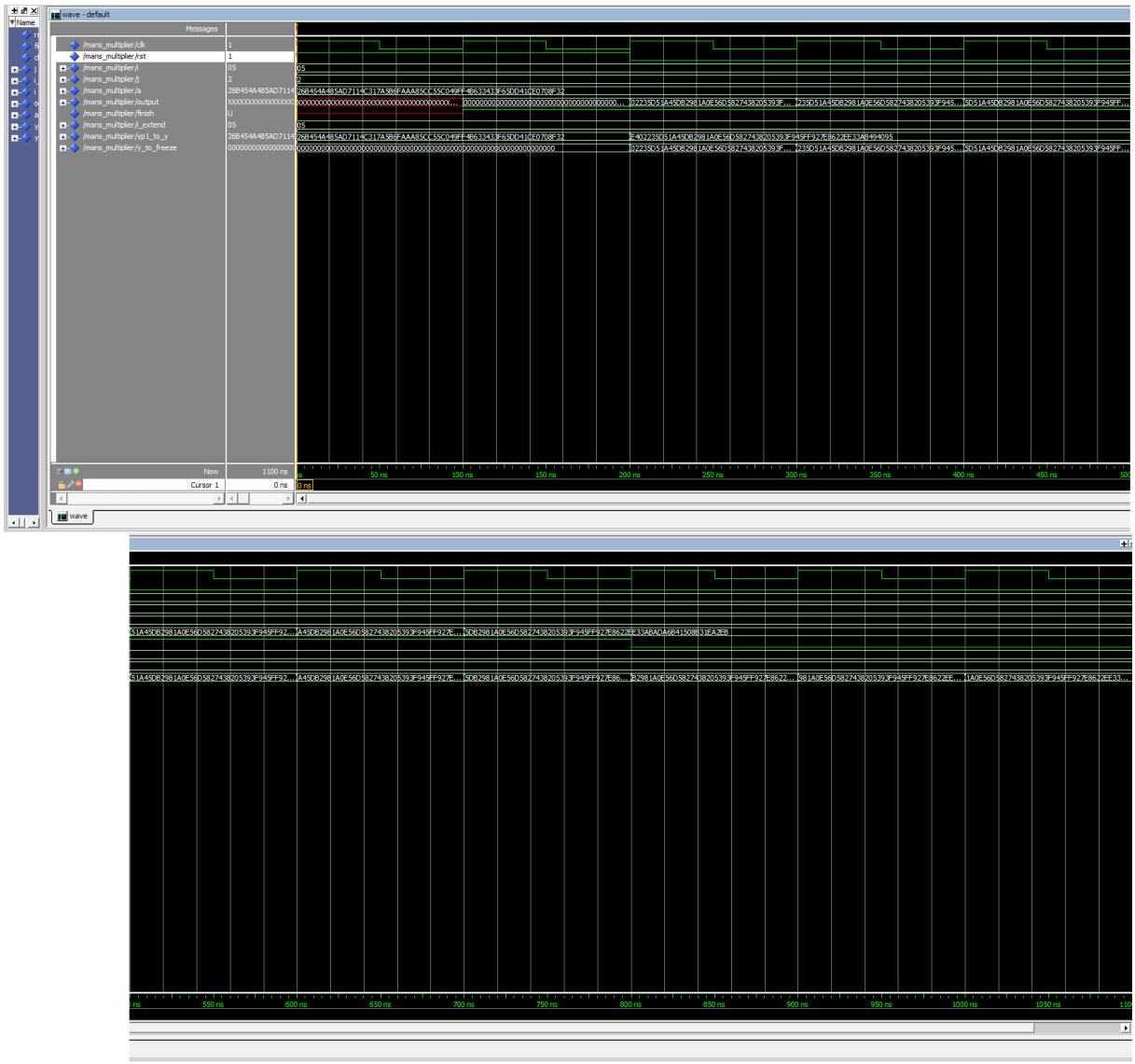


(Προσ. 4.1 y+1_at_j_mul)



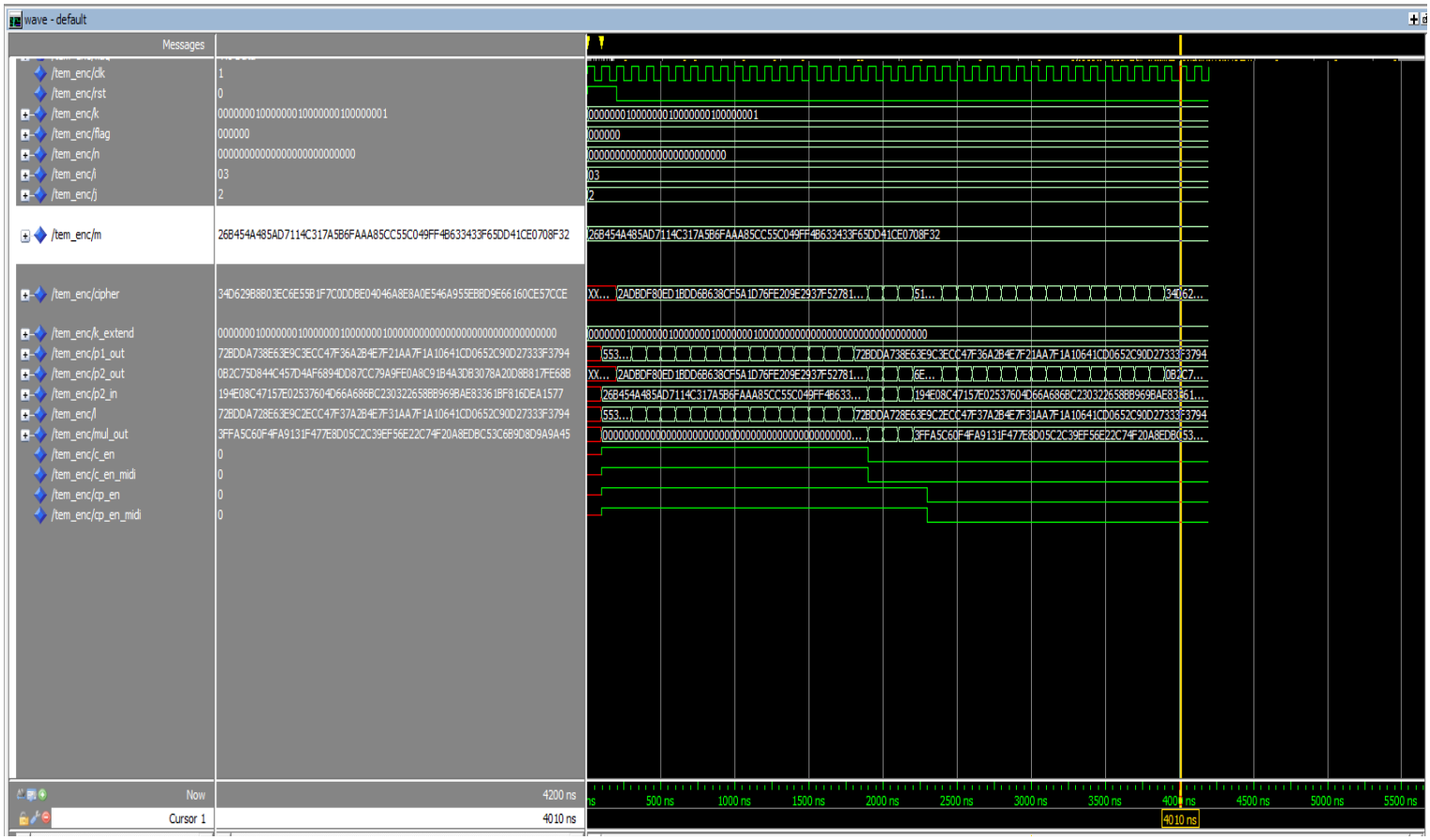
(Προσ. 4.2 y+1_at_j_mul)

Σχόλια: Στα σχήματα 4.1 και 4.2 φαίνεται η λειτουργία του πολλαπλασιαστή $(y+1)^j A$ καθώς αλλάζουμε τις τιμές του j. Επίσης παρατηρήστε ότι η λανθασμένη τιμή 3 επιστρέφει το ίδιο αποτέλεσμα με την 1.



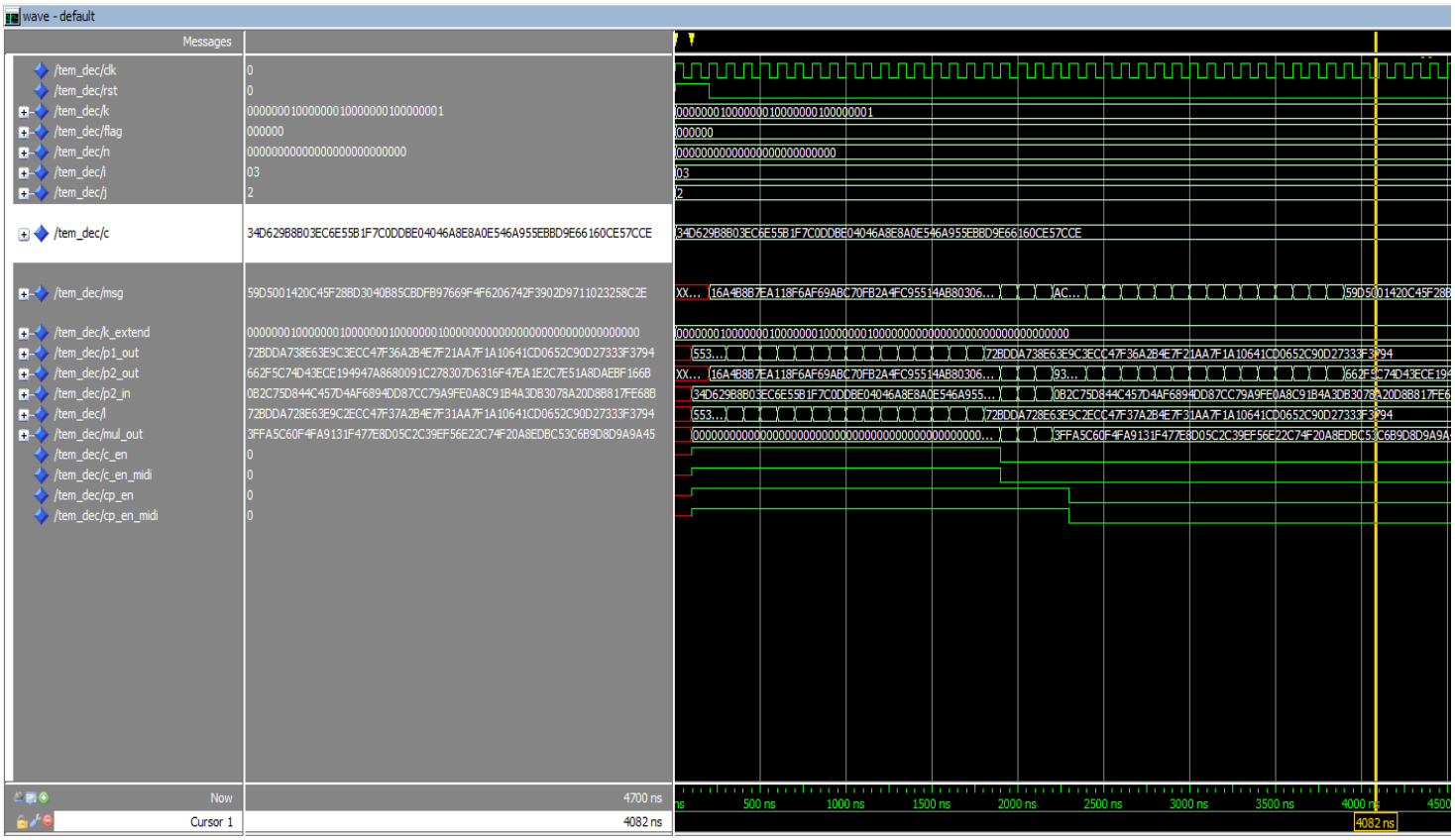
(Προσ. 5. Mans_Multiplier)

Σχόλια: Στην εικόνα 5 παρουσιάζεται η προσομοίωση του πολλαπλασιαστή $(y+1)^j y^i A$ για .i = 5 (δηλαδή 6 επαναλήψεις) και j=2. Το πρώτο κομμάτι του πολλαπλασιασμού είναι ασύγχρονο και βγάζει αποτέλεσμα χωρίς καθυστερήσεις, από εκεί και πέρα βλέπουμε το output να αλλάζει όσο πολλαπλασιάζουμε επι y μέχρι την χρονική στιγμή 800ns, 600ns μετά την ενεργοποίηση της λειτουργίας (rst → 0) οπου το σύστημα έχει πολλαπλασιάσει με y, 6 φορές και η έξοδος παγώνει.



(Προσ. 6. TEM_ENC)

Σχόλια: Στην εικόνα 6 φαίνεται η ολοκληρωμένη λειτουργία του κρυπτογράφου για i=3 και j=1 το κλειδί έχει δωθεί αυθέρετα ως K=128'h00000010000000100000010000001 ενώ τα N και flags έχουν μηδενικές τιμές. Βλέπουμε στο σχήμα πως αρχικά το κύκλωμα αλλάζει τις τιμές p1_out 17 φορές μετά την ενεργοποίηση όπου p1_out είναι η λειτουργία της P που γεννά νέο κλειδί (σήμα L). Στη συνέχεια βλέπουμε οτι για 3 κύκλους ρολογιού αλλάζει το σήμα mul_out που είναι η έξοδος του εσωτερικού mans_multiplier που εκτελεί πολλαπλασιασμούς για i=3 και j=2. Τότε και αυτό το σήμα παγώνει και p2_out αρχίζει να αλλάζει. Εδώ φαίνεται η λειτουργία της 2ης P που εκτελείτε για ακόμα 17 κύκλους ρολογιού. Τη χρονική στιγμή 4000ns, το κυκλωμα έχει ολοκληρώσει τη λειτουργία του και παγώνει την έξοδο στην τιμή 34D629B8B03EC6E55B1F7C0DDBE04046A8E8A0E546A955EBBD9E66160CE57CCE.



(Προσ. 7. TEM_DEC)

Σχόλια: Προσομοιώνοντας την λειτουργία του αποκωδικοποιητή βλέπουμε τις ανάλογες κυματομορφές στην εικόνα 7. Τα διανύσματα είσοδου είναι ίδια με αυτά του κωδικοποιητή εκτός από την έισοδο C που είναι η έξοδος C που παράχθηκε κατα την εκτέλεση του κωδικοποιητή για την default είσοδο. Παρατηρούμε πως η έξοδος δεν ταυτίζεται με την έισοδο οπότε μπορούμε να υποθέσουμε οτι έχει γίνει κάποιο λάθος στο σχεδιασμό.

Παρουσιάζονται οι τιμές M,C,M'

M: `26B454A485AD7114C317A5B6FAAA85CC55C049FF4B633433F65DD41CE0708F32`

C: `34D629B8B03EC6E55B1F7C0DDBE04046A8E8A0E546A955EBBD9E66160CE57CCE`

M': `59D5001420C45F28BD3040B85CBDFB97669F4F6206742F3902D9711023258C2E`

Επίλογος και Επιπλέον Σχόλια

Κατά την συγγραφή της εργασίας έγιναν διάφορες αλλαγές και διορθώσεις στην υλοποίηση του κάθε βασικού κυκλώματος τόσο κατά τη συγγραφή κώδικα VHDL όσο και κατα την σύνθεση της αναφοράς όπου πολλά λάθη βγήκαν στην επιφάνεια.

Σημειώνεται πως στην παρούσα (δεύτερη) έκδοση της αναφοράς ελέχτηκε και διορθώθηκε η συνάρτηση P στην οποία είχα παραβλέψει πολύ σημαντικα/βασικά συστήματα. Επίσης δοκιμάστηκε η

κωδικο/απόκωδικοποίηση θεωρόντας ότι $P^{-1}(P(x)) = x$ αλλά ούτε αυτό φαίνεται να βγάζει σωστό αποτέλεσμα. Υποθέτω πως το λάθος βρίσκεται στην ανίστροφη διαδικασία και πιο συγκεκριμένα στο κομμάτι του τελευταίου “μισού” γύρου.

Βλέποντας μια πιό γενική όψη του συστήματος που σχεδιάσαμε φαίνεται πως (πέρα απο διορθώσεις για την σωστή λειτουργία του) μπορούν να γίνουν πολλές αλλαγές. Σε αυτό το παράρτημα θα αναφέρω μερικές.

- Παύση της συνεχόμενης εσωτερικής λειτουργίας εφόσων τα εξαρτήματα P και Mans_Multiplier έχουν ολοκληρώσει την βασική τους λειτουργία. (Power)
- TEM_ENC/DEC: Υλοποίηση με ένα μόνο σύστημα P. Για την παραγωγή Ciphertext ή Message χρειάζεται το L οπότε θα μπορούσαμε να χρησιμοποιήσουμε το συστημα P που παράγει το L και για την παραγωγή των C/M εφόσων το συνολικό σύστημα τρέχει σειριακά χωρίς επιπλέον χρονικό κόστος. Σημειώνεται οτι θα χρειαστεί ειδικό κύκλωμα που θα αλλάζει τη λειτουργία του P σε P^{-1} (Υλικό/Χώρος)
- Κάθε κύκλωμα freeze_signal χρησιμοποιεί το δικό του μετρητή. Θα μπορούσαμε να υλοποιήσουμε το σύστημα TEM_ENC/DEC με ένα μόνο μετρητή που να κάνει reset κάθε φορά που κάποιο εξάρτημα ολοκληρώνει τη λειτουργία του.(Υλικό/Χώρος)

Προβλεπόμενα προβλήματα

- Πολλά από τα ολοκληρωμένα πρώτου επιπέδου έχουν σχεδιαστεί με behavioral τεχνική και ίσως αποδειχθούν πολύ πιο περίπλοκα στην πραγματική τους σχεδίαση
- Πολλά από τα ολοκληρωμένα κατώτερων επιπέδων έχουν σχεδιαστεί με structural τεχνικές με μόνο κρητήριο να αναγνωρίζονται οι έξοδοι τους κατά το simulation. Υποθέτω πως κάτι τέτοιο θα έχει επιπτώσεις στη σύνθεση, πόσο μάλλον για FPGA.
- Σε αυτή την εργασία έχουμε σχεδιάσει ορισμένα ασύγχρονα κυκλώματα τα οποία έχουν κάποιο throughput που δεν έχουμε υπολογίσει αλλά αντίθετα συνεχίσαμε το σχεδιασμό με την παραδωχή οτι το throughput είναι 0ns.

Σύνθεση VIVADO

Πραγματοποιήθηκε επιτυχής σύνθεση του κυκλώματος TEM_ENC στο Vivado σε device

“xc7k70tfbv676-1 (active)”

Βλέπουμε στο σχηματικό πως το κύκλωμα του TEM_ENC περιέχει ένα P_Process_16_rounds και ένα Mans_Multiplier, να υποθέσουμε πως το εργαλείο βελτιστοποίησε την αρχιτεκτονική μας με λιγότερο υλικό όπως προαναφέρθηκε.(?)

Η επαφή με το πρόγραμμα Vivado είναι ακόμη πολύ επιφανειακή και λόγω αυτού δεν θα επιχειρήσω να συμπεριλάβω επεξήγηση. Το schematic που έχει προκύψει παρουσιάζεται σε μορφή pdf καθώς είναι πολύ λεπτομερές για να τυπωθεί σε χαρτί A4.

Κώδικες VHDL:

Επίπεδο 1

Οι περισσότεροι από τους κώδικες πρώτου επιπέδου αντιγράφτηκαν από το διαδίκτυο είτε ως ήταν είτε υποστήκαν αλλαγές για να ικανοποιήσουν τις απαιτήσεις μας.

Comparator

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- fpga4student.com FPGA projects, Verilog projects, VHDL projects
entity comparator is
port (
      clock: in std_logic;
      A,B: in std_logic_vector(7 downto 0);
      IAB: in std_logic; -- Expansion input ( Active low)
      Output: out std_logic -- Output = 0 when A = B
 );
end comparator;
architecture Behavioral of comparator is
signal AB: std_logic_vector(7 downto 0); -- temporary variables
signal Result: std_logic;
begin
 AB(0) <= (not A(0)) xnor (not B(0));
        -- combinational circuit
 AB(1) <= (not A(1)) xnor (not B(1));
 AB(2) <= (not A(2)) xnor (not B(2));
 AB(3) <= (not A(3)) xnor (not B(3));
 AB(4) <= (not A(4)) xnor (not B(4));
 AB(5) <= (not A(5)) xnor (not B(5));
 AB(6) <= (not A(6)) xnor (not B(6));
 AB(7) <= (not A(7)) xnor (not B(7));
 process(clock)
 begin
 if(rising_edge(clock))then
   if(AB = x"FF" and IAB = '0') then
         -- check whether A = B and IAB =0 or not
            Result <= '0';
    else
     Result <= '1';
    end if;
 end if;
 end process;
 Output <= Result;
end Behavioral;
```

Full Adder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity full_adder_1 is
 Port ( A : in STD_LOGIC;
 B : in STD_LOGIC;
 Cin : in STD_LOGIC;
 S : out STD_LOGIC;
 Cout : out STD_LOGIC);
end full_adder_1;

architecture behaviour of full_adder_1 is

begin

 S <= A XOR B XOR Cin ;
 Cout <= (A AND B) OR (Cin AND A) OR (Cin AND B) ;

end behaviour;
```

Multiplexer 2 to 1

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY mul_2to1 IS
port(I0,I1,S:IN std_logic;Y:OUT std_logic);
end mul_2to1;
architecture arch_mul of mul_2to1 is
  begin
    Y<=((not S) and I0) or (S and I1);

  end arch_mul;
```

D Latch

```
 library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity d_latch_top is
    Port ( D  : in  STD_LOGIC;
           EN : in  STD_LOGIC;
           Q  : out STD_LOGIC);
end d_latch_top;

architecture Behavioral of d_latch_top is
    signal DATA : STD_LOGIC;
begin

    DATA <= D when (EN = '1') else DATA;
    Q <= DATA;

end Behavioral;
```

SR Latch

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity S_R_latch_top is
    Port ( S : in    STD_LOGIC;
           R : in    STD_LOGIC;
           Q : inout STD_LOGIC); -- changed out to inout
end S_R_latch_top;

architecture Behavioral of S_R_latch_top is
signal notQ : STD_LOGIC;
begin

Q    <= R nor notQ;
notQ <= S nor Q;

end Behavioral;
```

D flip-flop

```
library ieee;
use ieee.std_logic_1164.all;
----------------------
entity dff is
port (              d, clk, rst: in std_logic;
                    q: out std_logic    );
end dff;
----------------------
architecture behavior of dff is
begin
          process(rst, clk)
          begin
                    if (rst='1') then
                    q <= '0';
                    else if
                    (clk'event AND clk = '1') then
                    q<=d;
                    end if;
                    end if;
          end process;
end behavior;
```

Counter

```
 library IEEE;
            use ieee.std_logic_1164.all;
            use ieee.std_logic_unsigned.all;
            entity counter is
             port(Clock, CLR : in  std_logic;
             Q : out std_logic_vector(3 downto 0));
             end counter;
             architecture archi of counter is
             signal tmp: std_logic_vector(3 downto 0);
             begin
             process (Clock, CLR)
             begin
                   if (CLR='1') then
                          tmp(3 downto 0) <= (3 downto 0 => '0');
                   elsif (Clock'event and Clock='1') then
                          tmp <= tmp + '1';
                   end if;
             end process;
                   Q <= tmp;--conv_std_logic_vector( --std_logic_vector(to_unsigned(tmp, tmp'length));
             end archi;
```

Reverse Counter

```
 library IEEE;
           use ieee.std_logic_1164.all;
           use ieee.std_logic_unsigned.all;
           entity reverse_counter is
            port(Clock, CLR : in  std_logic;
            Q : out std_logic_vector(3 downto 0));
            end reverse_counter;
            architecture archi of reverse_counter is
            signal tmp: std_logic_vector(3 downto 0);
            begin
            process (Clock, CLR)
            begin
                  if (CLR='1') then
                        tmp(3 downto 1) <= (3 downto 1 => '1');
                                                            tmp(0) <= '1';
                  elsif (Clock'event and Clock='1') then
                        tmp <= tmp - '1';
                  end if;
            end process;
                  Q <= tmp;
            end archi;
```

Substitute Nibbles

```
         library IEEE;
use IEEE.std_logic_1164.all;

entity SubNibbler is
port(
--         clk: in std_logic;
--         rst: in std_logic;
          bytein: in std_logic_vector(3 downto 0);
          byteout: out std_logic_vector(3 downto 0)
          );
end SubNibbler;

architecture rtl of SubNibbler          is

signal A3 : std_logic;
signal A2 : std_logic;
signal A1 : std_logic;
signal A0 : std_logic;
signal A3n : std_logic;
signal A2n : std_logic;
signal A1n : std_logic;
signal A0n : std_logic;
begin


A3 <= bytein(3);
A2 <= bytein(2);
A1 <= bytein(1);
A0 <= bytein(0);
A3n <= "not"(bytein(3));
A2n <= "not"(bytein(2));
A1n <= "not"(bytein(1));
A0n <= "not"(bytein(0));

byteout(3) <= (A3n and A2 and A0)or (A3n and A2 and A1)or (A3 and A1n and A0)or (A3 and A1 and A0n) or (A3n and A2n and A1n
and A0n);
byteout(2) <= (A3n and A1 and A0n) or (A2n and A1 and A0n) or (A2 and A1 and A0)or (A3 and A2n and A1n) or (A3 and A1n and
A0n);
byteout(1) <= (A3n and A2n and A1n) or (A2 and A1n and A0n) or (A2 and A1 and A0) or (A3 and A1 and A0n);
byteout(0) <= (A3n and A2n and A1n) or (A3 and A2n and A1n) or (A3 and A2 and A0) or (A3n and A1 and A0);
```

Shuffle Rows SR1, $SR1^{-1}$, SR2, $SR2^{-1}$

(4 ξεχωριστά αρχεία παρουσιάζονται με την σειρά)

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
------------------------
entity ShuffleRows1 is
port(
          INPUT_NIBS: in std_logic_vector (31 downto 0);
          OUTPUT_NIBS: out std_logic_vector (31 downto 0)
          );
end ShuffleRows1;
----------------------
architecture SR1 of ShuffleRows1 is
begin

          OUTPUT_NIBS(31 downto 28) <= INPUT_NIBS(4*5+3 downto 4*5);                    --7                5
          OUTPUT_NIBS(27 downto 24) <= INPUT_NIBS(4* 4 +3 downto 4* 4);                --6                4
          OUTPUT_NIBS(23 downto 20) <= INPUT_NIBS(4* 3 +3 downto 4* 3);                --5                3
          OUTPUT_NIBS(19 downto 16) <= INPUT_NIBS(4* 2 +3 downto 4* 2);                --4                2
          OUTPUT_NIBS(15 downto 12) <= INPUT_NIBS(4* 0 +3 downto 4* 0);                --3                0
          OUTPUT_NIBS(11 downto 8) <= INPUT_NIBS(4* 1 +3 downto 4* 1);                 --2                1
          OUTPUT_NIBS(7 downto 4) <= INPUT_NIBS(4* 7 +3 downto 4* 7);                  --1                7
          OUTPUT_NIBS(3 downto 0) <= INPUT_NIBS(4* 6 +3 downto 4* 6);                  --0                6
end SR1;

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
------------------------
entity ShuffleRows1_reverse is
port(
          INPUT_NIBS: in std_logic_vector (31 downto 0);
          OUTPUT_NIBS: out std_logic_vector (31 downto 0)
          );
end ShuffleRows1_reverse;
----------------------
architecture SR1 of ShuffleRows1_reverse is
begin
          OUTPUT_NIBS(31 downto 28) <= INPUT_NIBS(4*1+3 downto 4*1);                    --7                1
          OUTPUT_NIBS(27 downto 24) <= INPUT_NIBS(4* 0 +3 downto 4* 0);                --6                0
          OUTPUT_NIBS(23 downto 20) <= INPUT_NIBS(4* 7 +3 downto 4* 7);                --5                7
          OUTPUT_NIBS(19 downto 16) <= INPUT_NIBS(4* 6 +3 downto 4* 6);                --4                6
          OUTPUT_NIBS(15 downto 12) <= INPUT_NIBS(4* 5 +3 downto 4* 5);                --3                5
          OUTPUT_NIBS(11 downto 8) <= INPUT_NIBS(4* 4 +3 downto 4* 4);                 --2                4
          OUTPUT_NIBS(7 downto 4) <= INPUT_NIBS(4* 2 +3 downto 4* 2);                  --1                2
          OUTPUT_NIBS(3 downto 0) <= INPUT_NIBS(4* 3 +3 downto 4* 3);                  --0                3
end SR1;

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
------------------------
entity ShuffleRows2 is
port(
          INPUT_NIBS: in std_logic_vector (31 downto 0);
          OUTPUT_NIBS: out std_logic_vector (31 downto 0)
          );
end ShuffleRows2;
----------------------
architecture SR1 of ShuffleRows2 is
begin
          OUTPUT_NIBS(31 downto 28) <= INPUT_NIBS(4*3+3 downto 4*3);                    --7                3
          OUTPUT_NIBS(27 downto 24) <= INPUT_NIBS(4* 2 +3 downto 4* 2);                --6                2
          OUTPUT_NIBS(23 downto 20) <= INPUT_NIBS(4* 6 +3 downto 4* 6);                --5                6
          OUTPUT_NIBS(19 downto 16) <= INPUT_NIBS(4* 7 +3 downto 4* 7);                --4                7
          OUTPUT_NIBS(15 downto 12) <= INPUT_NIBS(4* 1 +3 downto 4* 1);                --3                1
          OUTPUT_NIBS(11 downto 8) <= INPUT_NIBS(4* 0 +3 downto 4* 0);                 --2                0
          OUTPUT_NIBS(7 downto 4) <= INPUT_NIBS(4* 5 +3 downto 4* 5);                  --1                5
          OUTPUT_NIBS(3 downto 0) <= INPUT_NIBS(4* 4 +3 downto 4* 4);                  --0                4
end SR1;

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
------------------------
entity ShuffleRows2_reverse is
port(
          INPUT_NIBS: in std_logic_vector (31 downto 0);
          OUTPUT_NIBS: out std_logic_vector (31 downto 0)
          );
end ShuffleRows2_reverse;
----------------------
architecture SR1 of ShuffleRows2_reverse is
begin
```

```
          OUTPUT_NIBS(31 downto 28) <= INPUT_NIBS(4*4+3 downto 4*4);                    --7                4
          OUTPUT_NIBS(27 downto 24) <= INPUT_NIBS(4* 5 +3 downto 4* 5);                --6                5
          OUTPUT_NIBS(23 downto 20) <= INPUT_NIBS(4* 1 +3 downto 4* 1);                --5                1
          OUTPUT_NIBS(19 downto 16) <= INPUT_NIBS(4* 0 +3 downto 4* 0);                --4                0
          OUTPUT_NIBS(15 downto 12) <= INPUT_NIBS(4* 7 +3 downto 4* 7);                --3                7
          OUTPUT_NIBS(11 downto 8) <= INPUT_NIBS(4* 6 +3 downto 4* 6);                 --2                6
          OUTPUT_NIBS(7 downto 4) <= INPUT_NIBS(4* 3 +3 downto 4* 3);                  --1                3
          OUTPUT_NIBS(3 downto 0) <= INPUT_NIBS(4* 2 +3 downto 4* 2);                  --0                2
end SR1;
```

Mix Columns

```
-------------------------------------
----COLUMN MIXER-
-------------------------------------
---- BE CAREFUL ON INPUT
---- WE HAVE TO INPUT NIBS BY COLUMNS
---- SO IF bits = 128, COLUMNS = 8, ROWS = 4
---- i = row_ind, j = col_ind
---- FOR THE FIRST COLUMN WE INPUT 4*(i+1)*(j+1)-1 downto 4*(i+1)*(j+1)-4

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
--USE WORK.MINALPHER_PKG.ALL;


------------------------
entity MixColumns is
port(
          INPUT_NIBS: in std_logic_vector (15 downto 0);
          OUTPUT_NIBS: out std_logic_vector (15 downto 0)
          );
end MixColumns;
----------------------
architecture behaviour of MixColumns is
begin

          OUTPUT_NIBS(15 downto 12) <= INPUT_NIBS(15 downto 12) xor INPUT_NIBS(3 downto 0) xor INPUT_NIBS(11 downto 8);
                    --3
          OUTPUT_NIBS(11 downto 8) <= INPUT_NIBS(11 downto 8) xor INPUT_NIBS(15 downto 12) xor INPUT_NIBS(7 downto 4);
          --2
          OUTPUT_NIBS(7 downto 4) <= INPUT_NIBS(7 downto 4) xor INPUT_NIBS(11 downto 8) xor INPUT_NIBS(3 downto 0);
          --1
          OUTPUT_NIBS(3 downto 0) <= INPUT_NIBS(3 downto 0) xor INPUT_NIBS(7 downto 4) xor INPUT_NIBS(15 downto 12);
          --0

end behaviour;
```

x_Multiplier

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
------------------------
entity x_mul is
port(
          rst : in std_logic;
          INPUT: in std_logic_vector (7 downto 0);
          output: out std_logic_vector (7 downto 0)
          );
end x_mul;
----------------------
architecture behvr of x_mul is
signal mul_tmp1 : std_logic_vector(7 downto 0);
begin
------------------------
          mul_tmp1(7) <= INPUT(6) xor INPUT(7);   --mul x
          mul_tmp1(6) <= INPUT(5);
          mul_tmp1(5) <= INPUT(4) xor INPUT(7);
          mul_tmp1(4) <= INPUT(3);
          mul_tmp1(3) <= INPUT(2);
          mul_tmp1(2) <= INPUT(1);
          mul_tmp1(1) <= INPUT(0) xor INPUT(7);
          mul_tmp1(0) <= INPUT(7);
-------------------------
--To avoid clock complexity
--throughout the higher lvl
--design we will generate this
--asyncronically
-------------------------
```

```
process(rst, mul_tmp1)
begin
          if(rst = '1') then
                    OUTPUT <= (7 downto 0 => '0');
          else
                    OUTPUT <= mul_tmp1;

          end if;
end process;
end behvr;
--------------------------
```

Επίπεδο 2

(Στο επίπεδο αυτό έχουμε συμπεριλάβει συστήματα υψηλότερου επιπέδου 2_2 για λόγους ευκολίας)

Limit Counter

```
----------------------------------------
--The limit counter outputs the clock adder
--and a bit that freezes to 0 when the adder
--reaches the limit
----------------------------------------
library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity limit_counter is
           port(
                    Clock, CLR : in  std_logic;
                    Limit: in std_logic_vector(7 downto 0);
                    Q : out std_logic_vector(5 downto 0);
                    Maxd : out std_logic
                    );
 end limit_counter;
----------------------------------------
 architecture archi of limit_counter is
          COMPONENT S_R_latch_top Port
                    (
                    S : in    STD_LOGIC;
                    R : in    STD_LOGIC;
        Q : inout STD_LOGIC); -- changed out to inout
          end COMPONENT S_R_latch_top;
           COMPONENT clk_addr port
                              (
                              Clock, CLR : in  std_logic;
                              Q : out std_logic_vector(5 downto 0)
                              );
           end COMPONENT clk_addr;
           COMPONENT comparator is
                    port (
                                clock: in std_logic;
                                -- clock for synchronization
                                A: in std_logic_vector(7 downto 0);
                                B: in std_logic_vector(7 downto 0);
                                -- Two inputs
                                IAB: in std_logic; -- Expansion input ( Active low)
                                Output: out std_logic -- Output = 0 when A = B
                              );
                    end COMPONENT comparator;

signal out_wire_cnt : std_logic_vector(5 downto 0);
signal out_wire_cmp : std_logic;
signal cnt_8_extens : std_logic_vector(7 downto 0);
signal out_max                : std_logic;
begin
           CA00: clk_addr port map(
                              clock     => clock,
                              clr                 => clr,
                              Q                   => out_wire_cnt
                              );
                              cnt_8_extens(7) <= '0';
                              cnt_8_extens(6) <= '0';
                              cnt_8_extens(5 downto 0) <= out_wire_cnt;
          CM00: comparator port map(
                              clock     => clock,
                              A                   => cnt_8_extens,
                              B                   => Limit,
                              IAB                 => '0',
                              Output    => out_wire_cmp
                              );
          L00: S_R_latch_top port map(
                    S => clr,
                    R => "not"(out_wire_cmp),
        Q => out_max -- changed out to inout
                    );
          process(clr, out_wire_cnt)
          begin
                    if(clr = '1') then
                              Q <= (5 downto 0 => '0');
                    else
                              Q <= out_wire_cnt;
                    end if;
          end process;
          Maxd <= out_max;
 end archi;
```

P_Procedure

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
-----------------------
entity P_Procedure is
port(
          r: in std_logic_vector(3 downto 0);

          INPUT_BITS: in std_logic_vector (255 downto 0);
          OUTPUT_BITS: out std_logic_vector (255 downto 0)
          );
end P_Procedure;
----------------------
architecture structural of P_Procedure is
          signal ASN : std_logic_vector(127 downto 0);
          signal BSN : std_logic_vector(127 downto 0);

          signal ASR : std_logic_vector(127 downto 0);
          signal BSR : std_logic_vector(127 downto 0);

          signal ASM : std_logic_vector(127 downto 0);
          signal BSM : std_logic_vector(127 downto 0);

          signal AXM : std_logic_vector(127 downto 0);
          signal BXM : std_logic_vector(127 downto 0);

          signal AMC : std_logic_vector(127 downto 0);
          signal BMC : std_logic_vector(127 downto 0);
          signal RCO : std_logic_vector(127 downto 0);

          COMPONENT subNibbler PORT(
                                                bytein: in std_logic_vector(3 downto 0);
                                                byteout: out std_logic_vector(3 downto 0)
                                    );
          END COMPONENT subNibbler;
-------------------4 different shuffle rows my god----------------
          COMPONENT ShuffleRows1 PORT(
                                                INPUT_NIBS: in std_logic_vector (31 downto 0);
                                                OUTPUT_NIBS: out std_logic_vector (31 downto 0)
                                    );
          END COMPONENT ShuffleRows1;

          COMPONENT ShuffleRows2 PORT(
                                                INPUT_NIBS: in std_logic_vector (31 downto 0);
                                                OUTPUT_NIBS: out std_logic_vector (31 downto 0)
                                    );
          END COMPONENT ShuffleRows2;

          COMPONENT ShuffleRows1_reverse PORT(
                                                INPUT_NIBS: in std_logic_vector (31 downto 0);
                                                OUTPUT_NIBS: out std_logic_vector (31 downto 0)
                                    );
          END COMPONENT ShuffleRows1_reverse;

          COMPONENT ShuffleRows2_reverse PORT(
                                                INPUT_NIBS: in std_logic_vector (31 downto 0);
                                                OUTPUT_NIBS: out std_logic_vector (31 downto 0)
                                    );
          END COMPONENT ShuffleRows2_reverse;
-----------------------------------------------------------------
          COMPONENT MixColumns PORT(
                                                INPUT_NIBS: in std_logic_vector (15 downto 0);
                                                OUTPUT_NIBS: out std_logic_vector (15 downto 0)
                                    );
          END COMPONENT MixColumns;

          COMPONENT RoundConstant PORT(
                                                r: in std_logic_vector (3 downto 0);
                                                OUTPUT_NIBBS: out std_logic_vector(127 downto 0)
                                    );
          END COMPONENT RoundConstant;


begin
-----------------------------------------------------------------
---SUBSTITUTE NIBBLES
-----------------------------------------------------------------
          G1 : FOR n IN 31 DOWNTO 0 GENERATE
          SN00:subNibbler
                    PORT MAP(           --clk => clock,
                                        --rst => reset,
                                        bytein(3) => INPUT_BITS(n*4+3),
```

```
                                    bytein(2) => INPUT_BITS(n*4+2),
                                    bytein(1) => INPUT_BITS(n*4+1),
                                    bytein(0) => INPUT_BITS(n*4),

                                    byteout(3) => BSN(n*4+3),
                                    byteout(2) => BSN(n*4+2),
                                    byteout(1) => BSN(n*4+1),
                                    byteout(0) => BSN(n*4)

                           );
                  END GENERATE G1;


         G2 : FOR n IN 63 DOWNTO 32 GENERATE
         SN01:subNibbler
                   PORT MAP(
                                    bytein(3) => INPUT_BITS(n*4+3),
                                    bytein(2) => INPUT_BITS(n*4+2),
                                    bytein(1) => INPUT_BITS(n*4+1),
                                    bytein(0) => INPUT_BITS(n*4),

                                    byteout(3) => ASN((n-32)*4+3),
                                    byteout(2) => ASN((n-32)*4+2),
                                    byteout(1) => ASN((n-32)*4+1),
                                    byteout(0) => ASN((n-32)*4)

                           );
                  END GENERATE G2;

-------------------------------------------------------------------
--SHUFFLE ROWS
-------------------------------------------------------------------

         --SHUFFLE A MATRIX's ROWS
                           SRA3:ShuffleRows2_reverse
                    PORT MAP(
                                    INPUT_NIBS => ASN(32*3+31 downto 32*3),
                                    OUTPUT_NIBS => ASR(32*3+31 downto 32*3)
                           );


                  SRA2:ShuffleRows1_reverse
                   PORT MAP(
                                    INPUT_NIBS => ASN(32*2+31 downto 32*2),
                                    OUTPUT_NIBS => ASR(32*2+31 downto 32*2)
                           );

                  SRA1:ShuffleRows2
                   PORT MAP(
                                    INPUT_NIBS => ASN(32*1+31 downto 32*1),
                                    OUTPUT_NIBS => ASR(32*1+31 downto 32*1)
                           );


                  SRA0:ShuffleRows1
                   PORT MAP(
                                    INPUT_NIBS => ASN(32*0+31 downto 32*0),
                                    OUTPUT_NIBS => ASR(32*0+31 downto 32*0)
                           );

         --SHUFFLE B MATRIX's ROWS
                  SRB3:ShuffleRows2
                   PORT MAP(
                                    INPUT_NIBS => BSN(32*3+31 downto 32*3),
                                    OUTPUT_NIBS => BSR(32*3+31 downto 32*3)
                           );


                  SRB2:ShuffleRows1
                   PORT MAP(
                                    INPUT_NIBS => BSN(32*2+31 downto 32*2),
                                    OUTPUT_NIBS => BSR(32*2+31 downto 32*2)
                           );

                  SRB1:ShuffleRows2_reverse
                   PORT MAP(
                                    INPUT_NIBS => BSN(32*1+31 downto 32*1),
                                    OUTPUT_NIBS => BSR(32*1+31 downto 32*1)
                           );


                  SRB0:ShuffleRows1_reverse
                   PORT MAP(
                                    INPUT_NIBS => BSN(32*0+31 downto 32*0),
                                    OUTPUT_NIBS => BSR(32*0+31 downto 32*0)
                           );
```

```
------------------------------------------------------------------
--SWAP MATRICES
------------------------------------------------------------------

          BSM <= ASR;
          ASM <= BSR;

------------------------------------------------------------------
--XOR MATRICES
------------------------------------------------------------------

          AXM <= ASM;
          BXM <= ASM xor BSM;

------------------------------------------------------------------
--MIX COLUMNS
------------------------------------------------------------------
--INDEX FORMULA IS 32 Col_i + 4 Row_i + 3|0

          G5 : FOR n IN 7 DOWNTO 0 GENERATE
          MC00:MixColumns
                    PORT MAP(
                                        INPUT_NIBS(15 downto 12) => AXM(32*3+4*n+3 downto 32*3+4*n),
                                        INPUT_NIBS(11 downto 8) => AXM(32*2+4*n+3 downto 32*2+4*n),
                                        INPUT_NIBS(7 downto 4) => AXM(32*1+4*n+3 downto 32*1+4*n),
                                        INPUT_NIBS(3 downto 0) => AXM(32*0+4*n+3 downto 32*0+4*n),


                                        OUTPUT_NIBS => AMC(16*n+15 downto 16*n)
                              );
                    END GENERATE G5;

          G6 : FOR n IN 7 DOWNTO 0 GENERATE
          MC01:MixColumns
                    PORT MAP(
                                        INPUT_NIBS(15 downto 12) => BXM(32*3+4*n+3 downto 32*3+4*n),
                                        INPUT_NIBS(11 downto 8) => BXM(32*2+4*n+3 downto 32*2+4*n),
                                        INPUT_NIBS(7 downto 4) => BXM(32*1+4*n+3 downto 32*1+4*n),
                                        INPUT_NIBS(3 downto 0) => BXM(32*0+4*n+3 downto 32*0+4*n),


                                        OUTPUT_NIBS => BMC(16*n+15 downto 16*n)
                              );
                    END GENERATE G6;
------------------------------------------------------------------
--ROUND FUNCTION
------------------------------------------------------------------

          RC:RoundConstant
                    PORT MAP(
                                        r => r,
                                        OUTPUT_NIBBS => RCO
                              );
------------------------------------------------------------------
--END ROUND / OUPTUTS
------------------------------------------------------------------

          OUTPUT_BITS(255 downto 128) <= AMC;
          OUTPUT_BITS(127 downto 0) <= RCO xor BMC;

          end structural;
--------------------------
```

P_Back_Proc

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
-----------------------
entity P_Back_Proc is
port(
          r: in std_logic_vector(3 downto 0);

          INPUT_BITS: in std_logic_vector (255 downto 0);
          OUTPUT_BITS: out std_logic_vector (255 downto 0)
          );
end P_Back_Proc;
----------------------
architecture synthesis of P_Back_Proc is
          signal ASN : std_logic_vector(127 downto 0);
          signal BSN : std_logic_vector(127 downto 0);

          signal ASR : std_logic_vector(127 downto 0);
          signal BSR : std_logic_vector(127 downto 0);

          signal ASM : std_logic_vector(127 downto 0);
          signal BSM : std_logic_vector(127 downto 0);

          signal AXM : std_logic_vector(127 downto 0);
          signal BXM : std_logic_vector(127 downto 0);

          signal AMC : std_logic_vector(127 downto 0);
          signal BMC : std_logic_vector(127 downto 0);

          --signal ONB : nibble_matrix;
          signal RCO_TMP : std_logic_vector(127 downto 0);
          signal RCO : std_logic_vector(127 downto 0);

          signal RAXM : std_logic_vector(127 downto 0);
          signal RBXM : std_logic_vector(127 downto 0);

          signal RAMC : std_logic_vector(127 downto 0);
          signal RBMC : std_logic_vector(127 downto 0);

          signal BEX : std_logic_vector(127 downto 0);

          signal AOS : std_logic_vector(127 downto 0);
          signal BOS : std_logic_vector(127 downto 0);

          COMPONENT subNibbler PORT(
                                                bytein: in std_logic_vector(3 downto 0);
                                                byteout: out std_logic_vector(3 downto 0)
                                    );
          END COMPONENT subNibbler;

------------------4 different shuffle rows my god----------------
          COMPONENT ShuffleRows1 PORT(
                                                INPUT_NIBS: in std_logic_vector (31 downto 0);
                                                OUTPUT_NIBS: out std_logic_vector (31 downto 0)
                                    );
          END COMPONENT ShuffleRows1;

          COMPONENT ShuffleRows2 PORT(
                                                INPUT_NIBS: in std_logic_vector (31 downto 0);
                                                OUTPUT_NIBS: out std_logic_vector (31 downto 0)
                                    );
          END COMPONENT ShuffleRows2;

          COMPONENT ShuffleRows1_reverse PORT(
                                                INPUT_NIBS: in std_logic_vector (31 downto 0);
                                                OUTPUT_NIBS: out std_logic_vector (31 downto 0)
                                    );
          END COMPONENT ShuffleRows1_reverse;

          COMPONENT ShuffleRows2_reverse PORT(
                                                INPUT_NIBS: in std_logic_vector (31 downto 0);
                                                OUTPUT_NIBS: out std_logic_vector (31 downto 0)
                                    );
          END COMPONENT ShuffleRows2_reverse;
-----------------------------------------------------------------

          COMPONENT MixColumns PORT(
                                                INPUT_NIBS: in std_logic_vector (15 downto 0);
                                                OUTPUT_NIBS: out std_logic_vector (15 downto 0)
                                    );
          END COMPONENT MixColumns;
```

```
          COMPONENT RoundConstant PORT(
                                                r: in std_logic_vector (3 downto 0);
                                                OUTPUT_NIBBS: out std_logic_vector (127 downto 0)
                                    );
          END COMPONENT RoundConstant;


begin
-----------------------------------------------------------------
---SUBSTITUTE NIBBLES
-----------------------------------------------------------------
          G1 : FOR n IN 31 DOWNTO 0 GENERATE
          SN00:subNibbler
                     PORT MAP(          --clk => clock,
                                        --rst => reset,
                                        bytein(3) => INPUT_BITS(n*4+3),
                                        bytein(2) => INPUT_BITS(n*4+2),
                                        bytein(1) => INPUT_BITS(n*4+1),
                                        bytein(0) => INPUT_BITS(n*4),

                                        byteout(3) => BSN(n*4+3),
                                        byteout(2) => BSN(n*4+2),
                                        byteout(1) => BSN(n*4+1),
                                        byteout(0) => BSN(n*4)

                                );
                    END GENERATE G1;


          G2 : FOR n IN 63 DOWNTO 32 GENERATE
          SN01:subNibbler
                     PORT MAP(          --clk => clock,
                                        --rst => reset,
                                        bytein(3) => INPUT_BITS(n*4+3),
                                        bytein(2) => INPUT_BITS(n*4+2),
                                        bytein(1) => INPUT_BITS(n*4+1),
                                        bytein(0) => INPUT_BITS(n*4),

                                        byteout(3) => ASN((n-32)*4+3),
                                        byteout(2) => ASN((n-32)*4+2),
                                        byteout(1) => ASN((n-32)*4+1),
                                        byteout(0) => ASN((n-32)*4)

                                );
                    END GENERATE G2;

-----------------------------------------------------------------
--SHUFFLE ROWS
-----------------------------------------------------------------

          --SHUFFLE A MATRIX's ROWS
                              SRA3:ShuffleRows2_reverse
                     PORT MAP(
                                        INPUT_NIBS => ASN(32*3+31 downto 32*3),
                                        OUTPUT_NIBS => ASR(32*3+31 downto 32*3)
                                );


                    SRA2:ShuffleRows1_reverse
                     PORT MAP(
                                        INPUT_NIBS => ASN(32*2+31 downto 32*2),
                                        OUTPUT_NIBS => ASR(32*2+31 downto 32*2)
                                );

                    SRA1:ShuffleRows2
                     PORT MAP(
                                        INPUT_NIBS => ASN(32*1+31 downto 32*1),
                                        OUTPUT_NIBS => ASR(32*1+31 downto 32*1)
                                );


                    SRA0:ShuffleRows1
                     PORT MAP(
                                        INPUT_NIBS => ASN(32*0+31 downto 32*0),
                                        OUTPUT_NIBS => ASR(32*0+31 downto 32*0)
                                );

          --SHUFFLE B MATRIX's ROWS
                    SRB3:ShuffleRows2
                     PORT MAP(
                                        INPUT_NIBS => BSN(32*3+31 downto 32*3),
                                        OUTPUT_NIBS => BSR(32*3+31 downto 32*3)
                                );


                    SRB2:ShuffleRows1
```

```
                    PORT MAP(
                                        INPUT_NIBS => BSN(32*2+31 downto 32*2),
                                        OUTPUT_NIBS => BSR(32*2+31 downto 32*2)
                                );

                    SRB1:ShuffleRows2_reverse
                    PORT MAP(
                                        INPUT_NIBS => BSN(32*1+31 downto 32*1),
                                        OUTPUT_NIBS => BSR(32*1+31 downto 32*1)
                                );


                    SRB0:ShuffleRows1_reverse
                    PORT MAP(
                                        INPUT_NIBS => BSN(32*0+31 downto 32*0),
                                        OUTPUT_NIBS => BSR(32*0+31 downto 32*0)
                                );


------------------------------------------------------------------
--SWAP MATRICES
------------------------------------------------------------------

          BSM <= ASR;
          ASM <= BSR;

------------------------------------------------------------------
--XOR MATRICES
------------------------------------------------------------------

          AXM <= ASM;
          BXM <= ASM xor BSM;

------------------------------------------------------------------
--MIX COLUMNS
------------------------------------------------------------------
--INDEX FORMULA IS 32 Col_i + 4 Row_i + 3|0

          G5 : FOR n IN 7 DOWNTO 0 GENERATE
          MC00:MixColumns
                    PORT MAP(
                                        INPUT_NIBS(15 downto 12) => AXM(32*3+4*n+3 downto 32*3+4*n),
                                        INPUT_NIBS(11 downto 8) => AXM(32*2+4*n+3 downto 32*2+4*n),
                                        INPUT_NIBS(7 downto 4) => AXM(32*1+4*n+3 downto 32*1+4*n),
                                        INPUT_NIBS(3 downto 0) => AXM(32*0+4*n+3 downto 32*0+4*n),


                                        OUTPUT_NIBS => AMC(16*n+15 downto 16*n)
                                );
                    END GENERATE G5;

          G6 : FOR n IN 7 DOWNTO 0 GENERATE
          MC01:MixColumns
                    PORT MAP(
                                        INPUT_NIBS(15 downto 12) => BXM(32*3+4*n+3 downto 32*3+4*n),
                                        INPUT_NIBS(11 downto 8) => BXM(32*2+4*n+3 downto 32*2+4*n),
                                        INPUT_NIBS(7 downto 4) => BXM(32*1+4*n+3 downto 32*1+4*n),
                                        INPUT_NIBS(3 downto 0) => BXM(32*0+4*n+3 downto 32*0+4*n),


                                        OUTPUT_NIBS => BMC(16*n+15 downto 16*n)
                                );
                    END GENERATE G6;
------------------------------------------------------------------
--ROUND FUNCTION
------------------------------------------------------------------

          RC:RoundConstant
                    PORT MAP(
                                        r => r,
                                        OUTPUT_NIBBS => RCO--_TMP
                                );

                                        --FIX NIBBS TO BITS
                                        --RCO_TMP <= CONV_NIBBS_TO_LOGIC(ONB);

------------------------------------------------------------------
--XOR AND MIX COLUMNS OF ROUND CONST -- THIS HAS BEEN CHANGED WITH RC xor B and then M
------------------------------------------------------------------
--In this section of the b_proc we are asked to calculate the round constant
--in M E(r-i) but E is a 127 bit vector. Assuming the rest less sig bits are 0
--RAXM = RBXM therefore we will just MC the RC
------------------------------------------------------------------
          --RAXM => RCO;

--        G7 : FOR n IN 7 DOWNTO 0 GENERATE
```

```
--        MC02:MixColumns
--                  PORT MAP(
--                                       INPUT_NIBS(15 downto 12) => RCO_TMP(4*4*(n+1)-1 downto 4*4*(n+1)-4),
--                                       INPUT_NIBS(11 downto 8) => RCO_TMP(4*3*(n+1)-1 downto 4*3*(n+1)-4),
--                                       INPUT_NIBS(7 downto 4) => RCO_TMP(4*2*(n+1)-1 downto 4*2*(n+1)-4),
--                                       INPUT_NIBS(3 downto 0) => RCO_TMP(4*(n+1)-1 downto 4*(n+1)-4),
--
--                                       OUTPUT_NIBS => RCO(16*n+15 downto 16*n)
--                             );
--        END GENERATE G7;
-----------------------------------------------------------------------
BEX <= RCO xor BMC xor AMC;

          G7 : FOR n IN 7 DOWNTO 0 GENERATE
          MC02:MixColumns
                    PORT MAP(
                                       INPUT_NIBS(15 downto 12) => AMC(4*4*(n+1)-1 downto 4*4*(n+1)-4),
                                       INPUT_NIBS(11 downto 8) => AMC(4*3*(n+1)-1 downto 4*3*(n+1)-4),
                                       INPUT_NIBS(7 downto 4) => AMC(4*2*(n+1)-1 downto 4*2*(n+1)-4),
                                       INPUT_NIBS(3 downto 0) => AMC(4*(n+1)-1 downto 4*(n+1)-4),

                                       OUTPUT_NIBS => AOS(16*n+15 downto 16*n)
                             );
          END GENERATE G7;

          G8 : FOR n IN 7 DOWNTO 0 GENERATE
          MC02:MixColumns
                    PORT MAP(
                                       INPUT_NIBS(15 downto 12) => BEX(4*4*(n+1)-1 downto 4*4*(n+1)-4),
                                       INPUT_NIBS(11 downto 8) => BEX(4*3*(n+1)-1 downto 4*3*(n+1)-4),
                                       INPUT_NIBS(7 downto 4) => BEX(4*2*(n+1)-1 downto 4*2*(n+1)-4),
                                       INPUT_NIBS(3 downto 0) => BEX(4*(n+1)-1 downto 4*(n+1)-4),

                                       OUTPUT_NIBS => BOS(16*n+15 downto 16*n)
                             );
          END GENERATE G8;

-----------------------------------------------------------------------
--END ROUND / OUPTUTS
-----------------------------------------------------------------------

          OUTPUT_BITS(255 downto 128) <= AOS;--AMC;
          OUTPUT_BITS(127 downto 0) <= BOS;--RCO xor BMC;

          end synthesis;
--------------------------
```

y_Mul

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
------------------------
entity y_mul is
port(

          rst : in std_logic;
          INPUT: in std_logic_vector (255 downto 0);
          output: out std_logic_vector (255 downto 0)
          );
end y_mul;
----------------------
architecture behvr of y_mul is

          COMPONENT x_mul port
                    (
                    rst : in std_logic;
                    INPUT: in std_logic_vector (7 downto 0);
                    output: out std_logic_vector (7 downto 0)
                    );
          end COMPONENT x_mul;


signal mul_tmp : std_logic_vector(255 downto 0);

-----------------------

begin

 XM00: x_mul port map(
                              rst       => rst,
                              INPUT     => INPUT(255 downto 248),
                              output  => mul_tmp(7 downto 0)--x_out_wire
                              );

          mul_tmp(255 downto 32) <= input(247 downto 24);
          mul_tmp(31 downto 24) <= INPUT(23 downto 16) xor INPUT(255 downto 248);         --byte 3
          mul_tmp(23 downto 16) <= INPUT(15 downto 8) xor INPUT(255 downto 248);          --byte 2
          mul_tmp(15 downto 8) <= INPUT(7 downto 0);

process(rst, mul_tmp)
          begin
                    if (rst = '1') then
                              OUTPUT <= (255 downto 0 =>'0');
                    else
                              OUTPUT <= mul_tmp;
                    end if;
end process;


end behvr;
--------------------------
```

Επίπεδο 3

Freeze Signal

```
library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity freeze_signal is
           port(
            INPUT : in std_logic_vector(255 downto 0);
                    Clock, CLR : in  std_logic;
                    Limit: in std_logic_vector(7 downto 0);
                    OUTPUT : out std_logic_vector(255 downto 0);
                    maxd: out std_logic
                    );

 end freeze_signal;
-------------------------
architecture structural of freeze_signal is

COMPONENT limit_counter
           port(
                    Clock, CLR : in  std_logic;
                    Limit: in std_logic_vector(7 downto 0);
                    Q : out std_logic_vector(5 downto 0);
                    Maxd : out std_logic
                    );
end COMPONENT limit_counter;


COMPONENT d_latch_top
          Port
          (
                    EN  : in std_logic;
                    D         : in std_logic;
                    Q         : out std_logic
          );
END COMPONENT d_latch_top;

signal freeze_signal : std_logic;

-----------------------------
begin
  lcnt: limit_counter Port Map
  (
    clock => clock,
    clr   => clr,
    Limit => Limit,
    Maxd  => Freeze_signal
    );

  G1:FOR n IN 255 downto 0 GENERATE
  dlat: d_latch_top PORT Map
  (
          EN  => Freeze_signal,
          D         => INPUT(n),
          Q         => OUTPUT(n)
  );
  END GENERATE G1;

  maxd <= Freeze_signal;

 end structural;
```

y_Mul_Loop

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
--USE WORK.MINALPHER_PKG.ALL;


------------------------
entity y_mul_loop is
port(
          clk : in std_logic;
          rst : in std_logic;
          INPUT: in std_logic_vector (255 downto 0);
          output: out std_logic_vector (255 downto 0)
          );
end y_mul_loop;
----------------------
architecture behvr of y_mul_loop is


          COMPONENT dff port
          (
                    d, clk, rst: in std_logic;
                    q: out std_logic
          );
          end COMPONENT dff;

          COMPONENT y_mul PORT
                    (
                    rst : in std_logic;
                    --clk : in std_logic;
                    INPUT: in std_logic_vector (255 downto 0);
                    output: out std_logic_vector (255 downto 0)
                    );
          END COMPONENT y_mul;

          COMPONENT mul_2to1 port
                    (
                              I0,I1,S:IN std_logic;Y:OUT std_logic
                    );
          end COMPONENT mul_2to1;

          signal y_to_dff : std_logic_vector(255 downto 0);
          signal dff_to_mux : std_logic_vector(255 downto 0);
          signal mux_to_y : std_logic_vector(255 downto 0);
          signal mux_select : std_logic;
----------------------

------------------------

begin

          G0: FOR n IN 255 downto 0 GENERATE
          dff0: dff PORT MAP
          (
                    d => y_to_dff(n),
                    clk => clk,
                    rst => rst,
                    q => dff_to_mux(n)
          );
          END GENERATE G0;

          G1: FOR n IN 255 downto 0 GENERATE
          mu00: mul_2to1 PORT MAP
          (
                    I0 => dff_to_mux(n),
                    I1 => INPUT(n),
                    S         => mux_select,
                    Y         => mux_to_y(n)
          );
          END GENERATE G1;


          ym00: y_mul PORT MAP
          (
                    rst => rst,
                    --clk => clk,
                    INPUT     => mux_to_y,
                    OUTPUT    => y_to_dff
          );

          dff1 : dff PORT MAP
          (
```

```
                    d => rst,
                    clk => clk,
                    rst => '0',
                    q => mux_select
          );
          OUTPUT <= y_to_dff;
end behvr;
```

y_plus_1_Mul_at_j

```
----------------------
--This is an asyncronous system
--that outputs the multiplication
--of A(y+1)^j for 0<= j <= 2
--Drive 3 subsystems to a 4to1 mux
--2: Ay^2 + 2Ay + A
--1: Ay+A
--0: A
------------------------------


LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
--USE WORK.MINALPHER_PKG.ALL;


------------------------
entity y_plus_1_mul_at_j is
port(
          rst : in std_logic;
          j : in std_logic_vector(1 downto 0);
          INPUT : in std_logic_vector(255 downto 0);
          OUTPUT: out std_logic_vector(255 downto 0)
          );
end y_plus_1_mul_at_j;
----------------------
architecture struct of y_plus_1_mul_at_j is

          COMPONENT mul_2to1 port
          (
          I0,I1,S:IN std_logic;Y:OUT std_logic);
          end COMPONENT mul_2to1;

          COMPONENT y_mul PORT
                    (
                    rst : in std_logic;
                    INPUT: in std_logic_vector (255 downto 0);
                    output: out std_logic_vector (255 downto 0)
                    );
          END COMPONENT y_mul;

          COMPONENT full_adder_1 PORT
          (
          A : in STD_LOGIC;
          B : in STD_LOGIC;
          Cin : in STD_LOGIC;
          S : out STD_LOGIC;
          Cout : out STD_LOGIC);
          end COMPONENT full_adder_1;


          --signal outp_wire: std_logic_vector(255 downto 0);
          signal ymulA : std_logic_vector(255 downto 0);
          signal mux0_out : std_logic_vector(255 downto 0);
          signal ymul_to_add : std_logic_vector(255 downto 0);
          signal ya_shift : std_logic_vector(255 downto 0);
          signal add2add_4mux0 : std_logic_vector(255 downto 0);
          signal in_mux1 : std_logic_vector(255 downto 0);
          --signal mux0_to_mux2 : std_logic_vector(255 downto 0);
          signal carry_interconnect : std_logic_vector(256 downto 0);
          signal carry_interconnect2 : std_logic_vector(256 downto 0);
          signal j_or                                        :std_logic;

------------------------
begin


--First y*A
-------------------------------------
          YM00: y_mul port map(
                              rst       => rst,
                              INPUT     => INPUT,
                              output  => ymulA
                              );

--y^2*A
```

```
--------------------------------------
          YM01: y_mul port map(
          rst       => rst,
          INPUT     => ymulA,
          output  => ymul_to_add
          );

--2y*A
--------------------------------------
                    ya_shift(255 downto 1) <= ymulA(254 downto 0);
                    ya_shift(0) <= '0';
                    carry_interconnect(0) <='0';


--Add y^2*A + 2y*A
--------------------------------------
          GA1 : FOR n IN 255 downto 0 GENERATE
          FA00: full_adder_1 PORT MAP
                              (

                              A => ymul_to_add(n),
                              B => ya_shift(n),
                              Cin => carry_interconnect(n),
                              S => Add2Add_4mux0(n),
                              Cout => carry_interconnect(n+1)--clock=>clock,
                              );
          END GENERATE GA1;

          carry_interconnect2(0) <='0';

--MUX 1 : bit 0 -> j = {2,1}
--------------------------------------
                    GM0: FOR n IN 255 downto 0 GENERATE
          M00: mul_2to1 port map(
                    I0 => Add2Add_4mux0(n),
                    I1 => ymulA(n),
                    S  => j(0),
                    Y  => mux0_out(n)
                    );
          END GENERATE GM0;

--2nd Adder
--------------------------------------
          GA2 : FOR n IN 255 downto 0 GENERATE
          FA00: full_adder_1 PORT MAP
                              (

                              A => mux0_out(n),
                              B => INPUT(n),
                              Cin => carry_interconnect2(n),
                              S => in_mux1(n),
                              Cout => carry_interconnect2(n+1)--clock=>clock,
                              );
          END GENERATE GA2;

          --j Or
          j_or <= j(1) or j(0);

          ---MUX for OR j----
          GM1: FOR n IN 255 downto 0 GENERATE
          M01: mul_2to1 port map(
                    I0 => INPUT(n),
                    I1 => in_mux1(n),
                    S  => j_or,
                    Y  => OUTPUT(n)
                    );
          END GENERATE GM1;


end struct;
--------------------------
```

Επίπεδο 4

P_Procedure_17+1_Rounds

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
--USE WORK.MINALPHER_PKG.ALL;
USE IEEE.NUMERIC_STD.ALL;
------------------------
--This architecture of the Complete P function
--is syncronous and consists of one P process
--running 17 loops + 1 for the last round
--It takes 1 cc for reset
--1 for initialization of the input dff
--16 cc for the P_process loop
--and outputs the result in a total of 17 cc
--after the reset
------------------------
entity P_Procedure_16_Rounds is
port(
          clock: in std_logic;
          reset: in std_logic;
          INPUT_BITS: in std_logic_vector (255 downto 0);
          OUTPUT_BITS: out std_logic_vector (255 downto 0);
          finish : out std_logic
          );
end P_Procedure_16_Rounds;
---------------------------------------
architecture structural of P_Procedure_16_Rounds is

--COMPONENTS

    COMPONENT counter

              port(Clock, CLR : in  std_logic;

              Q : out std_logic_vector(3 downto 0));

    end COMPONENT counter;

          COMPONENT dff PORT
          (
                    d, clk, rst: in std_logic;
                    q: out std_logic
          );
          END COMPONENT dff;

          COMPONENT mul_2to1 PORT
          (
                    I0,I1,S:IN std_logic;Y:OUT std_logic
          );
          END COMPONENT mul_2to1;

          COMPONENT SubNibbler PORT
                              (

                              bytein: in std_logic_vector(3 downto 0);
                              byteout: out std_logic_vector(3 downto 0)
                              );
          end COMPONENT SubNibbler;


-------------------4 different shuffle rows my god----------------
          COMPONENT ShuffleRows1 PORT(
                                                    INPUT_NIBS: in std_logic_vector (31 downto 0);
                                                    OUTPUT_NIBS: out std_logic_vector (31 downto 0)
                                        );
          END COMPONENT ShuffleRows1;

          COMPONENT ShuffleRows2 PORT(
                                                    INPUT_NIBS: in std_logic_vector (31 downto 0);
                                                    OUTPUT_NIBS: out std_logic_vector (31 downto 0)
                                        );
          END COMPONENT ShuffleRows2;

          COMPONENT ShuffleRows1_reverse PORT(
                                                    INPUT_NIBS: in std_logic_vector (31 downto 0);
                                                    OUTPUT_NIBS: out std_logic_vector (31 downto 0)
                                        );
          END COMPONENT ShuffleRows1_reverse;

          COMPONENT ShuffleRows2_reverse PORT(
                                                    INPUT_NIBS: in std_logic_vector (31 downto 0);
                                                    OUTPUT_NIBS: out std_logic_vector (31 downto 0)
                                        );
```

```
          END COMPONENT ShuffleRows2_reverse;
-------------------------------------------------------------------

          COMPONENT P_procedure PORT
                              (
                              r: in std_logic_vector(3 downto 0);
                              --clock: in std_logic;
                              --reset: in std_logic;
                              INPUT_BITS: in std_logic_vector (255 downto 0);
                              OUTPUT_BITS: out std_logic_vector (255 downto 0)
                              );
          END COMPONENT P_procedure;

  COMPONENT freeze_signal is
     port(
      INPUT : in std_logic_vector(255 downto 0);
      Clock, CLR : in  std_logic;
      Limit: in std_logic_vector(7 downto 0);
      OUTPUT : out std_logic_vector(255 downto 0);
            Maxd : out std_logic
      );

   end COMPONENT freeze_signal;
--WIRES
signal mux_init : std_logic;
--signal dff_in     : std_logic_vector(255 downto 0);
signal dff_out  : std_logic_vector(255 downto 0);
signal P_in         : std_logic_vector(255 downto 0);
signal P_out        : std_logic_vector(255 downto 0);
--signal init_sel : std_logic;
signal round        : std_logic_vector(3 downto 0);

signal lr_interconnect1 : std_logic_vector(255 downto 0);
signal lr_interconnect2 : std_logic_vector(255 downto 0);
signal lr_interconnect3 : std_logic_vector(255 downto 0);

signal out_Freeze : std_logic_vector(255 downto 0);

begin
------------------------

CNT: Counter PORT MAP
                              (
                              Clock     => clock,
                              CLR                 => reset,--"not"(init_sel),
             Q                => round(3 downto 0)
                               );

G0: FOR n IN 255 DOWNTO 0 GENERATE
DFF00: dff
          PORT MAP(
                                        d => P_out(n),--dff_in(n),
                                        clk => clock,
                                        rst => reset,
                                        q => dff_out(n)
                              );
END GENERATE G0;


----------------------------
--Mux Initializer WITH dff rst stall
----------------------------
DFFST: dff
          PORT MAP(
                                        d => reset,--dff_in(n),
                                        clk => clock,
                                        rst => '0',
                                        q => mux_init
                              );

GM: FOR n IN 255 downto 0 GENERATE
IMUX: mul_2to1
          PORT MAP(
                                        I0 => dff_out(n),
                                        I1 => INPUT_BITS(n),
                                        S  => mux_init,
                                        Y  => P_in(n)
                              );
END GENERATE GM;
----------------------------
P00:P_procedure
          PORT MAP(
                              r=> round, --round const
                              --clock=>clock,
                              --reset=>reset,
                              INPUT_BITS => P_in,
```

```
                              OUTPUT_BITS=> P_out
                    );

lr_interconnect1 <= P_out;

--LAST ROUND CONSTANT
G2 : FOR n IN 63 DOWNTO 0 GENERATE
SN00:SubNibbler
          PORT MAP(
                              bytein => lr_interconnect1(n*4+3 downto n*4),
                              byteout => lr_interconnect2(n*4+3 downto n*4)
                              );
END GENERATE G2;

--FINAL ROUND SHUFFLE ROWS
          --SHUFFLE A MATRIX's ROWS
                              SRA3:ShuffleRows2_reverse
                     PORT MAP(
                                        INPUT_NIBS => lr_interconnect2(32*7+31 downto 32*7),
                                        OUTPUT_NIBS => lr_interconnect3(32*7+31 downto 32*7)
                              );


                    SRA2:ShuffleRows1_reverse
                     PORT MAP(
                                        INPUT_NIBS => lr_interconnect2(32*6+31 downto 32*6),
                                        OUTPUT_NIBS => lr_interconnect3(32*6+31 downto 32*6)
                              );

                    SRA1:ShuffleRows2
                     PORT MAP(
                                        INPUT_NIBS => lr_interconnect2(32*5+31 downto 32*5),
                                        OUTPUT_NIBS => lr_interconnect3(32*5+31 downto 32*5)
                              );


                    SRA0:ShuffleRows1
                     PORT MAP(
                                        INPUT_NIBS => lr_interconnect2(32*4+31 downto 32*4),
                                        OUTPUT_NIBS => lr_interconnect3(32*4+31 downto 32*4)
                              );

          --SHUFFLE B MATRIX's ROWS
                    SRB3:ShuffleRows2
                     PORT MAP(
                                        INPUT_NIBS => lr_interconnect2(32*3+31 downto 32*3),
                                        OUTPUT_NIBS => lr_interconnect3(32*3+31 downto 32*3)
                              );


                    SRB2:ShuffleRows1
                     PORT MAP(
                                        INPUT_NIBS => lr_interconnect2(32*2+31 downto 32*2),
                                        OUTPUT_NIBS => lr_interconnect3(32*2+31 downto 32*2)
                              );

                    SRB1:ShuffleRows2_reverse
                     PORT MAP(
                                        INPUT_NIBS => lr_interconnect2(32*1+31 downto 32*1),
                                        OUTPUT_NIBS => lr_interconnect3(32*1+31 downto 32*1)
                              );


                    SRB0:ShuffleRows1_reverse
                     PORT MAP(
                                        INPUT_NIBS => lr_interconnect2(32*0+31 downto 32*0),
                                        OUTPUT_NIBS => lr_interconnect3(32*0+31 downto 32*0)
                              );

OUT_FREEZE(255 downto 128) <= lr_interconnect3(127 downto 0);
OUT_FREEZE(127 downto 0) <= lr_interconnect3(255 downto 128);

                    frez:     freeze_signal
     port MAP
     (
      INPUT => OUT_FREEZE,
      Clock => clock,
      CLR   => reset,
      Limit => "00010000", --counter out_bits are 4 so 16 rounds it is--"00010000", --starts counting from zero so 16-> 17
rounds
      OUTPUT => OUTPUT_BITS,
            Maxd    => finish
      );


END structural;
```

P_Back_Proc_17+1_Rounds

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
--USE WORK.MINALPHER_PKG.ALL;
USE IEEE.NUMERIC_STD.ALL;
------------------------
--This architecture of the Complete P function
--is syncronous and consists of one P process
--running 16 loops + 1 for the last round
--It takes 1 cc for reset
--1 for initialization of the input dff
--16 cc for the P_process loop
--and outputs the result in a total of 17 cc
--after the reset
------------------------
entity P_Back_Proc_16_Rounds is
port(
          clock: in std_logic;
          reset: in std_logic;
          INPUT_BITS: in std_logic_vector (255 downto 0);
          OUTPUT_BITS: out std_logic_vector (255 downto 0);
          finish : out std_logic
          );
end P_Back_Proc_16_Rounds;
---------------------------------------
architecture structural of P_Back_Proc_16_Rounds is

--COMPONENTS

    COMPONENT reverse_counter

             port(Clock, CLR : in  std_logic;

             Q : out std_logic_vector(3 downto 0));

    end COMPONENT reverse_counter;

          COMPONENT dff PORT
          (
                    d, clk, rst: in std_logic;
                    q: out std_logic
          );
          END COMPONENT dff;

          COMPONENT mul_2to1 PORT
          (
                    I0,I1,S:IN std_logic;Y:OUT std_logic
          );
          END COMPONENT mul_2to1;

          COMPONENT SubNibbler PORT
                              (

                              bytein: in std_logic_vector(3 downto 0);
                              byteout: out std_logic_vector(3 downto 0)
                              );
          end COMPONENT SubNibbler;


-------------------4 different shuffle rows my god----------------
          COMPONENT ShuffleRows1 PORT(
                                                    INPUT_NIBS: in std_logic_vector (31 downto 0);
                                                    OUTPUT_NIBS: out std_logic_vector (31 downto 0)
                                        );
          END COMPONENT ShuffleRows1;

          COMPONENT ShuffleRows2 PORT(
                                                    INPUT_NIBS: in std_logic_vector (31 downto 0);
                                                    OUTPUT_NIBS: out std_logic_vector (31 downto 0)
                                        );
          END COMPONENT ShuffleRows2;

          COMPONENT ShuffleRows1_reverse PORT(
                                                    INPUT_NIBS: in std_logic_vector (31 downto 0);
                                                    OUTPUT_NIBS: out std_logic_vector (31 downto 0)
                                        );
          END COMPONENT ShuffleRows1_reverse;

          COMPONENT ShuffleRows2_reverse PORT(
                                                    INPUT_NIBS: in std_logic_vector (31 downto 0);
                                                    OUTPUT_NIBS: out std_logic_vector (31 downto 0)
                                        );
          END COMPONENT ShuffleRows2_reverse;
-----------------------------------------------------------------
```

```
          COMPONENT P_Back_Proc PORT
                              (
                              r: in std_logic_vector(3 downto 0);
                              --clock: in std_logic;
                              --reset: in std_logic;
                              INPUT_BITS: in std_logic_vector (255 downto 0);
                              OUTPUT_BITS: out std_logic_vector (255 downto 0)
                              );
          END COMPONENT P_Back_Proc;

  COMPONENT freeze_signal is
     port(
      INPUT : in std_logic_vector(255 downto 0);
      Clock, CLR : in  std_logic;
      Limit: in std_logic_vector(7 downto 0);
      OUTPUT : out std_logic_vector(255 downto 0);
            Maxd : out std_logic
      );

   end COMPONENT freeze_signal;
--WIRES
signal mux_init : std_logic;
--signal dff_in     : std_logic_vector(255 downto 0);
signal dff_out  : std_logic_vector(255 downto 0);
signal P_in         : std_logic_vector(255 downto 0);
signal P_out        : std_logic_vector(255 downto 0);
--signal init_sel : std_logic;
signal round        : std_logic_vector(3 downto 0);

signal lr_interconnect1 : std_logic_vector(255 downto 0);
signal lr_interconnect2 : std_logic_vector(255 downto 0);
signal lr_interconnect3 : std_logic_vector(255 downto 0);

signal out_Freeze : std_logic_vector(255 downto 0);

begin
-------------------------

CNT: reverse_counter PORT MAP
                              (
                              Clock     => clock,
                              CLR                => reset,--"not"(init_sel),
             Q                => round(3 downto 0)
                               );



G0: FOR n IN 255 DOWNTO 0 GENERATE
DFF00: dff
          PORT MAP(
                                        d => P_out(n),--dff_in(n),
                                        clk => clock,
                                        rst => reset,
                                        q => dff_out(n)
                              );
END GENERATE G0;


----------------------------
--Mux Initializer WITH dff rst stall
----------------------------
DFFST: dff
          PORT MAP(
                                        d => reset,--dff_in(n),
                                        clk => clock,
                                        rst => '0',
                                        q => mux_init
                              );


GM: FOR n IN 255 downto 0 GENERATE
IMUX: mul_2to1
          PORT MAP(
                                        I0 => dff_out(n),
                                        I1 => INPUT_BITS(n),
                                        S  => mux_init,
                                        Y  => P_in(n)
                              );
END GENERATE GM;
----------------------------

P00:P_Back_Proc
          PORT MAP(
                              r=> round, --round const
```

```
                            INPUT_BITS => P_in,
                            OUTPUT_BITS=> P_out
                    );


lr_interconnect1 <= P_out;

--LAST ROUND CONSTANT
G2 : FOR n IN 63 DOWNTO 0 GENERATE
SN00:SubNibbler
          PORT MAP(
                            bytein => lr_interconnect1(n*4+3 downto n*4),
                            byteout => lr_interconnect2(n*4+3 downto n*4)
                            );
END GENERATE G2;

          --SHUFFLE A MATRIX's ROWS
                            SRA3:ShuffleRows2_reverse
                     PORT MAP(
                                       INPUT_NIBS => lr_interconnect2(32*7+31 downto 32*7),
                                       OUTPUT_NIBS => lr_interconnect3(32*7+31 downto 32*7)
                            );


                    SRA2:ShuffleRows1_reverse
                     PORT MAP(
                                       INPUT_NIBS => lr_interconnect2(32*6+31 downto 32*6),
                                       OUTPUT_NIBS => lr_interconnect3(32*6+31 downto 32*6)
                            );

                    SRA1:ShuffleRows2
                     PORT MAP(
                                       INPUT_NIBS => lr_interconnect2(32*5+31 downto 32*5),
                                       OUTPUT_NIBS => lr_interconnect3(32*5+31 downto 32*5)
                            );


                    SRA0:ShuffleRows1
                     PORT MAP(
                                       INPUT_NIBS => lr_interconnect2(32*4+31 downto 32*4),
                                       OUTPUT_NIBS => lr_interconnect3(32*4+31 downto 32*4)
                            );

          --SHUFFLE B MATRIX's ROWS
                    SRB3:ShuffleRows2
                     PORT MAP(
                                       INPUT_NIBS => lr_interconnect2(32*3+31 downto 32*3),
                                       OUTPUT_NIBS => lr_interconnect3(32*3+31 downto 32*3)
                            );

                    SRB2:ShuffleRows1
                     PORT MAP(
                                       INPUT_NIBS => lr_interconnect2(32*2+31 downto 32*2),
                                       OUTPUT_NIBS => lr_interconnect3(32*2+31 downto 32*2)
                            );

                    SRB1:ShuffleRows2_reverse
                     PORT MAP(
                                       INPUT_NIBS => lr_interconnect2(32*1+31 downto 32*1),
                                       OUTPUT_NIBS => lr_interconnect3(32*1+31 downto 32*1)
                            );

                    SRB0:ShuffleRows1_reverse
                     PORT MAP(
                                       INPUT_NIBS => lr_interconnect2(32*0+31 downto 32*0),
                                       OUTPUT_NIBS => lr_interconnect3(32*0+31 downto 32*0)
                            );

OUT_FREEZE(255 downto 128) <= lr_interconnect3(127 downto 0);
OUT_FREEZE(127 downto 0) <= lr_interconnect3(255 downto 128);


                    frez:     freeze_signal
     port MAP
     (
      INPUT => OUT_FREEZE,
      Clock => clock,
      CLR   => reset,
      Limit => "00010000", --starts counting from zero so 16-> 17 rounds BUT WE GOT 4 COUNT BITS SO FK IT WE GO WITH 16
ROUNDS
      OUTPUT => OUTPUT_BITS,
            Maxd    => finish
      );

END structural;
```

Mans_Multiplier

```
--NOTES: CHECK OUTPUT TIMERS BECAUSE WE USED TO HAVE I_MAX_SYNC NOW IMAX
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
--USE WORK.MINALPHER_PKG.ALL;


------------------------
entity mans_Multiplier is
port(

          clk : in std_logic;
          rst : in std_logic;

          i: in std_logic_vector (5 downto 0); --64 values
          j: in std_logic_vector (1 downto 0);
          A: in std_logic_vector(255 downto 0);

          output: out std_logic_vector (255 downto 0);
          finish: out std_logic
          );
end mans_Multiplier;
----------------------
architecture EMM of mans_Multiplier is

  COMPONENT freeze_signal is
     port(
      INPUT : in std_logic_vector(255 downto 0);
      Clock, CLR : in  std_logic;
      Limit: in std_logic_vector(7 downto 0);
      OUTPUT : out std_logic_vector(255 downto 0);
            Maxd : out std_logic
      );

   end COMPONENT freeze_signal;

          COMPONENT y_plus_1_mul_at_j PORT
                    (
                    j: in std_logic_vector(1 downto 0);
                    rst : in std_logic;
                    INPUT: in std_logic_vector (255 downto 0);
                    output: out std_logic_vector (255 downto 0)
                    );
          END COMPONENT y_plus_1_mul_at_j;

          COMPONENT y_mul_loop PORT
                    (
                    rst : in std_logic;
                    clk : in std_logic;
                    INPUT: in std_logic_vector (255 downto 0);
                    output: out std_logic_vector (255 downto 0)
                    );
          END COMPONENT y_mul_loop;


          COMPONENT dff port
          (
                    d, clk, rst: in std_logic;
                    q: out std_logic
          );
          end COMPONENT dff;
          --SIGNALS--
          signal i_extend : std_logic_vector(7 downto 0);
          signal i_count  : std_logic_vector(5 downto 0);
          signal i_max        : std_logic;
          signal i_max_sync   : std_logic;

          signal yp1_to_y : std_logic_vector(255 downto 0);
          signal y_to_freeze : std_logic_vector(255 downto 0);

          --signal clk_2      : std_logic;

          --signal rst_or_max : std_logic;

BEGIN

          --Signal Inits--

          i_extend(5 downto 0)          <= i;
          i_extend(6)                                       <= '0';
          i_extend(7)                                       <= '0';


          -- COMPONENTS --
```

```
          YP00:y_plus_1_mul_at_j PORT MAP
                    (
                              j                   => j,
                              rst                 => rst,
                              INPUT     => A,
                              output    => yp1_to_y
                    );

          Y00:y_mul_loop PORT MAP
                    (
                    rst                 => rst,
                    clk       => clk,
                    INPUT     => yp1_to_y,
                    output    => y_to_freeze
                    );

                    FR00: freeze_signal PORT MAP
                    (
                      INPUT => y_to_freeze,
                      Clock => clk,
                      CLR     => rst,
                      Limit => i_extend,
                      OUTPUT=> OUTPUT,
                      Maxd  => finish

                    );

                    --finish <= i_max_sync;

end EMM;


-----------------------------------------------
-----SIMULATION SCRIPT--------------------------

--force -freeze sim:/mans_multiplier/clk 1 0, 0 {50 ns} -r 100
--force -freeze sim:/mans_multiplier/rst 1 0
--force -freeze sim:/mans_multiplier/i 6'h05 0
--force -freeze sim:/mans_multiplier/j 6'h04 0
--force -freeze sim:/mans_multiplier/A 256'h557711111111111111111111111111111111111111111111111111111111111111111111FF 0
-----------------------
--This is a syncronous system
--that outputs the multiplication of
-- A * (y+1)^j*y^i in a total of i+j+1 cc
--
```

Επίπεδο 5

TEM_ENC

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
--USE WORK.MINALPHER_PKG.ALL;
USE IEEE.NUMERIC_STD.ALL;

------------------------
entity TEM_ENC is
port(
          --INPUTS
          K         : in std_logic_vector (127 downto 0);
          flag: in std_logic_vector (23 downto 0);
          N         : in std_logic_vector (103 downto 0);
          i         : in std_logic_vector (5 downto 0);     --NOT YET IMPLEMEMNTED
          j         : in std_logic_vector (1 downto 0);

          M         : in std_logic_vector (255 downto 0);

          clk : in std_logic;
          rst : in std_logic;

          --CIPHER
          CIPHER: out std_logic_vector (255 downto 0)
          );
end TEM_ENC;
---------------------------------------
architecture synthesis of TEM_ENC is

          COMPONENT S_R_latch_top PORT
     (               S : in    STD_LOGIC;
           R : in    STD_LOGIC;
           Q : inout STD_LOGIC); -- changed out to inout
          end COMPONENT S_R_latch_top;


          COMPONENT P_procedure_16_rounds PORT
                              (
                              clock: in std_logic;
                              reset: in std_logic;
                              INPUT_BITS: in std_logic_vector (255 downto 0);
                              OUTPUT_BITS: out std_logic_vector (255 downto 0);
                              finish: out std_logic
                              );
          END COMPONENT P_procedure_16_rounds;

          COMPONENT mans_Multiplier port
          (

          clk : in std_logic;
          rst : in std_logic;

          i: in std_logic_vector (5 downto 0); --64 values
          j: in std_logic_vector (1 downto 0);
          A: in std_logic_vector(255 downto 0);

          output: out std_logic_vector (255 downto 0);
          finish : out std_logic
          );
          end COMPONENT mans_Multiplier;

          signal K_extend : std_logic_vector(255 downto 0);
          signal P1_out       : std_logic_vector(255 downto 0);
          signal P2_out       : std_logic_vector(255 downto 0);
          signal P2_in        : std_logic_vector(255 downto 0);
          signal L            : std_logic_vector(255 downto 0);
          signal Mul_out      : std_logic_vector(255 downto 0);

          signal C_EN                   : std_logic;
          signal C_EN_midi    : std_logic;
          signal CP_EN                  : std_logic;
          signal CP_EN_midi   : std_logic;
-------------------------------------------

begin

K_extend(255 downto 128) <= K;
K_extend(127 downto 104) <= flag;
K_extend(103 downto 0)        <= N;

--The output of P1 will need 17 cc to complete
L00:P_procedure_16_rounds
```

```
          PORT MAP(
                              clock=>clk,
                              reset=>rst,
                              INPUT_BITS => K_extend,
                              OUTPUT_BITS=> P1_out,
                              finish => C_EN_midi
                    );



                    --GENERATE L
L <= K_extend xor P1_out;

-----CIPHERTEXT GENERATION-------------------
SR00:s_r_latch_top PORT MAP
          (
                    R => "not"(C_EN_midi),
                    S => rst,
                    Q => C_EN
          );



M00:mans_Multiplier port MAP
          (
          clk       => clk,
          rst       => C_EN,

          i                   => i,
          j                   => j,

          A                   => L,

          output    => Mul_out,
          finish => CP_EN_midi
          );


P2_in <= M xor Mul_out;

sr01:s_r_latch_top PORT MAP
          (
                    R => "not"(CP_EN_midi),
                    S => rst,
                    Q => CP_EN
          );


L01:P_procedure_16_rounds
          PORT MAP(
                              clock=>clk,
                              reset=>CP_EN,
                              INPUT_BITS => P2_in,
                              OUTPUT_BITS=> P2_out
                    );

CIPHER <= Mul_out xor P2_out;



END SYNTHESIS;
--------------------------------------------------------------------------
--                                      SIMULATION SCRIPT
--force -freeze sim:/tem_enc/K 128'h00000001000000010000000100000001 0
--force -freeze sim:/tem_enc/flag 24'h000000 0
--force -freeze sim:/tem_enc/N 104'h00000000000000000000000000 0
--force -freeze sim:/tem_enc/i 6'h03 0
--force -freeze sim:/tem_enc/j 6'h03 0
--force -freeze sim:/tem_enc/M 256'h1111111111111111111111111111111111111111111111111111111111111111111111111111111111 0
--force -freeze sim:/tem_enc/clk 1 0, 0 {50 ns} -r 100
--force -freeze sim:/tem_enc/rst 1 0
```

TEM_DEC

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
--USE WORK.MINALPHER_PKG.ALL;
USE IEEE.NUMERIC_STD.ALL;

------------------------
entity TEM_DEC is
port(
          --INPUTS
          K         : in std_logic_vector (127 downto 0);
          flag: in std_logic_vector (23 downto 0);
          N         : in std_logic_vector (103 downto 0);
          i         : in std_logic_vector (5 downto 0);     --NOT YET IMPLEMEMNTED
          j         : in std_logic_vector (1 downto 0);

          C         : in std_logic_vector (255 downto 0);

          clk : in std_logic;
          rst : in std_logic;

          --CIPHER
          MSG: out std_logic_vector (255 downto 0)
          );
end TEM_DEC;
---------------------------------------
architecture synthesis of TEM_DEC is

          COMPONENT S_R_latch_top PORT
     (               S : in    STD_LOGIC;
           R : in    STD_LOGIC;
           Q : inout STD_LOGIC); -- changed out to inout
          end COMPONENT S_R_latch_top;


          COMPONENT P_Back_Proc_16_Rounds PORT
                              (
                              clock: in std_logic;
                              reset: in std_logic;
                              INPUT_BITS: in std_logic_vector (255 downto 0);
                              OUTPUT_BITS: out std_logic_vector (255 downto 0);
                              finish: out std_logic
                              );
          END COMPONENT P_Back_Proc_16_Rounds;

          COMPONENT P_Procedure_16_Rounds PORT
                              (
                              clock: in std_logic;
                              reset: in std_logic;
                              INPUT_BITS: in std_logic_vector (255 downto 0);
                              OUTPUT_BITS: out std_logic_vector (255 downto 0);
                              finish: out std_logic
                              );
          END COMPONENT P_Procedure_16_Rounds;

          COMPONENT mans_Multiplier port
          (

          clk : in std_logic;
          rst : in std_logic;

          i: in std_logic_vector (5 downto 0); --64 values
          j: in std_logic_vector (1 downto 0);
          A: in std_logic_vector(255 downto 0);

          output: out std_logic_vector (255 downto 0);
          finish : out std_logic
          );
          end COMPONENT mans_Multiplier;

          signal K_extend : std_logic_vector(255 downto 0);
          signal P1_out       : std_logic_vector(255 downto 0);
          signal P2_out       : std_logic_vector(255 downto 0);
          signal P2_in        : std_logic_vector(255 downto 0);
          signal L            : std_logic_vector(255 downto 0);
          signal Mul_out      : std_logic_vector(255 downto 0);

          signal C_EN                   : std_logic;
          signal C_EN_midi    : std_logic;
          signal CP_EN                  : std_logic;
          signal CP_EN_midi   : std_logic;
-------------------------------------------

begin
```

```
K_extend(255 downto 128) <= K;
K_extend(127 downto 104) <= flag;
K_extend(103 downto 0)        <= N;

--The output of P1 will need 16 cc to complete
L00:P_Procedure_16_Rounds
          PORT MAP(
                              clock=>clk,
                              reset=>rst,
                              INPUT_BITS => K_extend,
                              OUTPUT_BITS=> P1_out,
                              finish => C_EN_midi
                    );



                    --GENERATE L
L <= K_extend xor P1_out;

-----CIPHERTEXT GENERATION-------------------
SR00:s_r_latch_top PORT MAP
          (
                    R => "not"(C_EN_midi),
                    S => rst,
                    Q => C_EN
          );



M00:mans_Multiplier port MAP
          (
          clk       => clk,
          rst       => C_EN,

          i                   => i,
          j                   => j,

          A                   => L,

          output    => Mul_out,
          finish => CP_EN_midi
          );


P2_in <= C xor Mul_out;

sr01:s_r_latch_top PORT MAP
          (
                    R => "not"(CP_EN_midi),
                    S => rst,
                    Q => CP_EN
          );


L01:P_Back_Proc_16_Rounds
          PORT MAP(
                              clock=>clk,
                              reset=>CP_EN,
                              INPUT_BITS => P2_in,
                              OUTPUT_BITS=> P2_out
                    );

MSG <= Mul_out xor P2_out;



END SYNTHESIS;
--------------------------------------------------------------------------------
--                                      SIMULATION SCRIPT
--force -freeze sim:/tem_enc/K 128'h00000001000000010000000100000001 0
--force -freeze sim:/tem_enc/flag 24'h000000 0
--force -freeze sim:/tem_enc/N 104'h00000000000000000000000000 0
--force -freeze sim:/tem_enc/i 6'h03 0
--force -freeze sim:/tem_enc/j 6'h03 0
--force -freeze sim:/tem_enc/M 256'h1111111111111111111111111111111111111111111111111111111111111111111111111111111111 0
--force -freeze sim:/tem_enc/clk 1 0, 0 {50 ns} -r 100
--force -freeze sim:/tem_enc/rst 1 0
```

Τέλος