**Ανάλυση κυκλωμάτων**

Σε αυτό το Report, κάνουμε ανάλυση των κυκλωμάτων.

Αυτή η ανάλυση περιλαμβάνει τα resources που χρησιμοποιούν το κάθε κύκλωμα, καθώς το κύκλο ρολογιών που κάνουν για να εκτελεστούν.

Επιπλέον περιλαμβάνεται ο κώδικας του κάθε κυκλώματος διαιρεμένο στις συναρτήσεις που μπορεί να εμπεριέχονται καθώς και στους βρόγχους που μπορούν υπάρχουν.

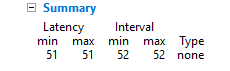
Αναλύουμε όλα τα παραπάνω με βάση τις εντολές που χρησιμοποιούν καθώς και τους κύκλους του ρολογιού που κάνουν μέχρι να εκτελεστούν.

Στο τέλος, με λίγα λόγια προσπαθώ να εξηγήσω την λόγο που έχουν την απόδοση που εμφανίζει το εργαλείο τα κυκλώματα.

**1ο κύκλωμα:**

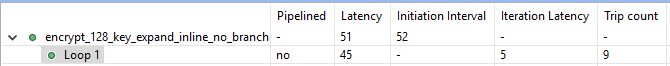
**Συνολικό Latency**

Το 1ο κύκλωμα έχει βγάλει το παρακάτω Latency από την σύνθεση:



**Ανάλυση**

Το 1ο κύκλωμα εμπεριέχει 1 Loop με latency :45 , iteration Latency :5 καθώς και Trip count :9



**Resources:**

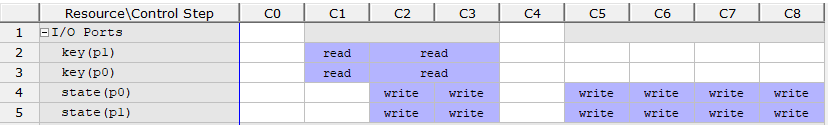
**I/O Ports:**

Σύμφωνα με το παρακάτω πίνακα χρησιμοποιεί 2 I/O Ports το key για read, καθώς το state για write.



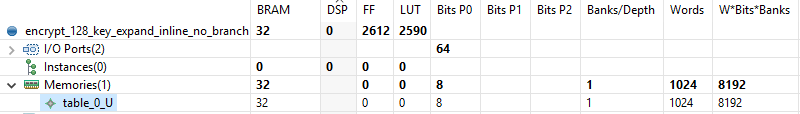
Καθώς βλέπουμε ότι έχουν μήκος 32 bit.

Επιπλέον βλέπουμε ότι κάνει τόσες προσπελάσεις στα Ports όσα βλέπουμε στον πίνακα:

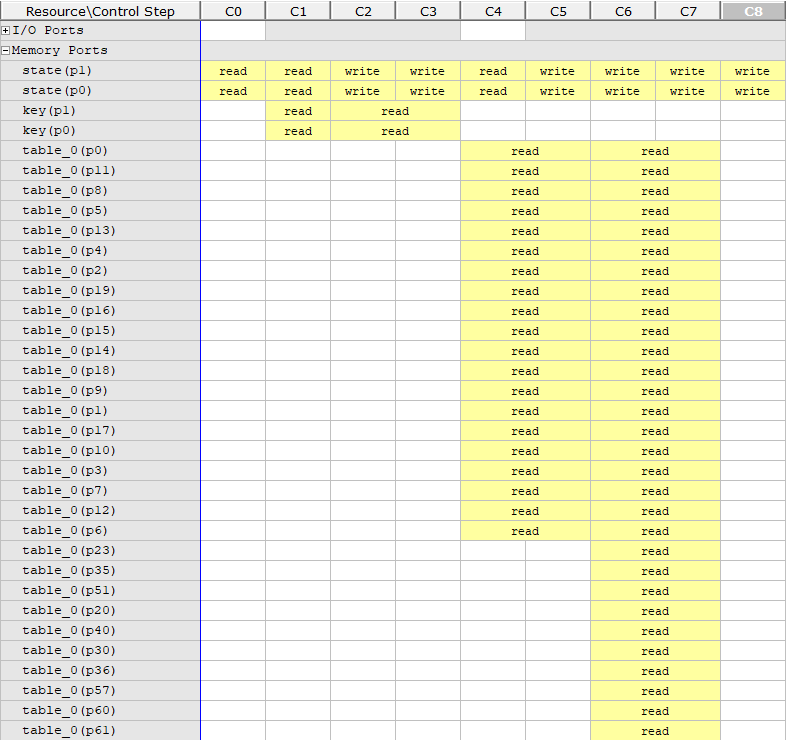


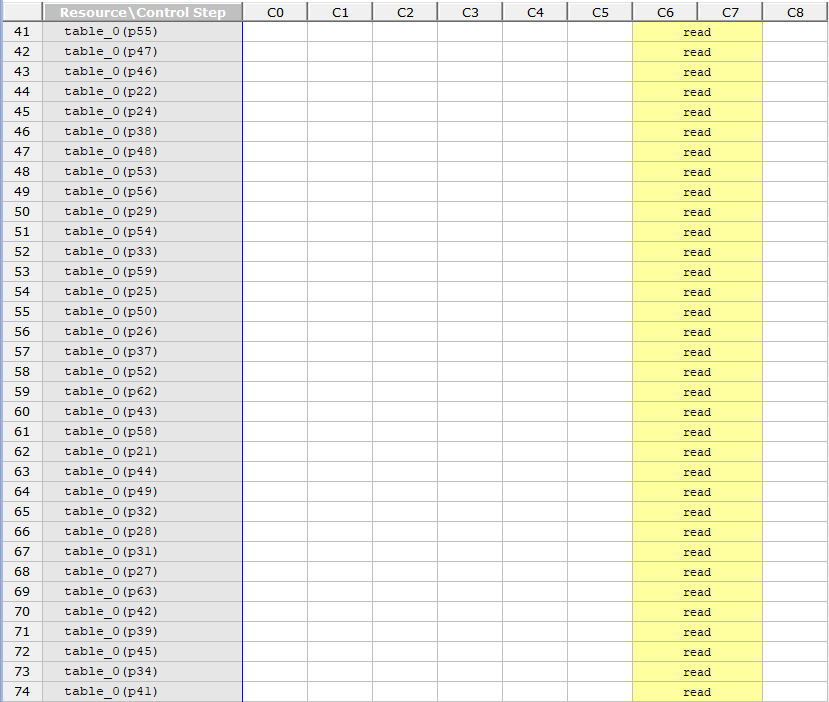
**Mνήμες:**

Σύμφωνα με τον πίνακα χρησιμοποιεί μια μνήμη εν’ ονόματι table\_0\_U που είναι BRAM , Block RAM 32, με μέγεθος 1024\*8=8192



Βλέπουμε ότι ξεκίνάει από το το 4 κύκλο να διαβάζει στην μνήμη του table\_0 ενώ πρίν περίμενε να περάσουν 3 κύκλοι ώστε διαβάσει τα I/O Ports key και state.

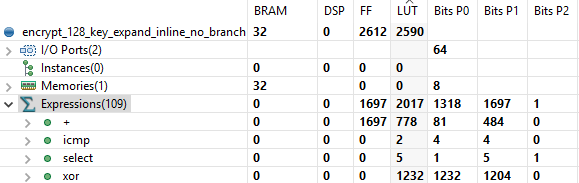




Και επίσης βλέπουμε τις προσπελάσεις που γίνονται στην μνήμη table\_0 που τελειώνουν στο 7ο κύκλο του ρολογιού και στον 8ο κάνει write στο state.

**Expressions:**

Από τι βλέπουμε από τον πίνακα χρησιμοποιεί το κύκλωμα 109 expressions:

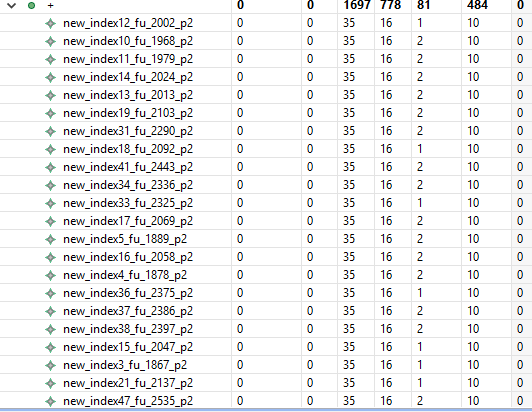


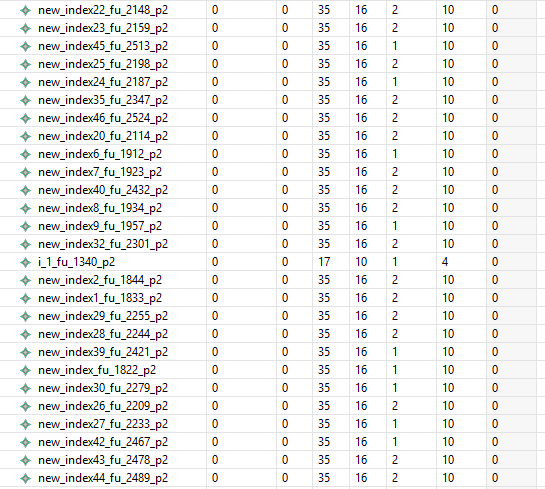
Στα οποία περιλαμβάνουν εντολές πρόσθεσης(+),σύγκρισης(icmp),επιλογής(select),xor(^).

**Προσθέσεις:**

Όλες οι προσθέσεις περιλαμβάνουν 1697 κυκλώματα flip-flop, 778 Look-Up tables καθώς 81+ 484 προσθέσεις bits αριθμών.

Με τόσες εντολές:



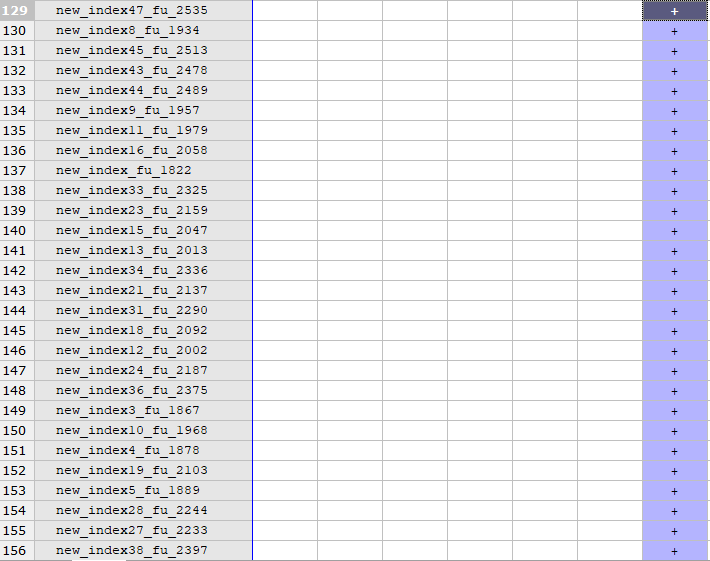


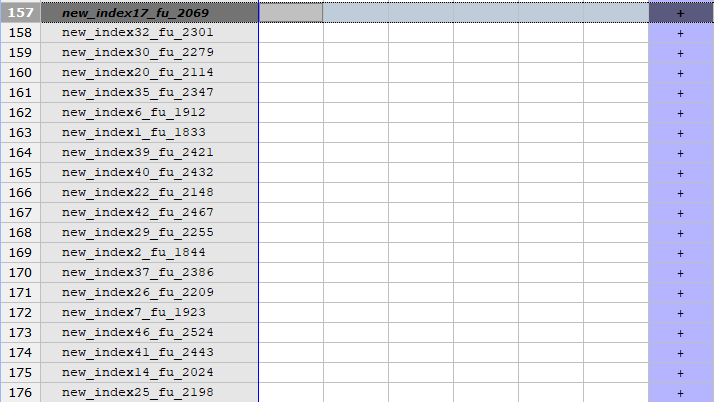
Οι προσθέσεις που γίνονται στις εντολές:

5ο κύκλο ρολογιού:



7ο κύκλο ρολογιού:

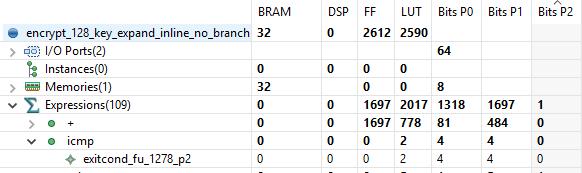




Συνολικές προσθέσεις 176-129+1=48

**Συγκρίσεις:**

Γίνεται στο κύκλωμα μια σύγκριση καθώς έχει 2 Loop-Up tables ,καθώς οι συγκρίσεις γίνονται των αριθμών 4 bit.

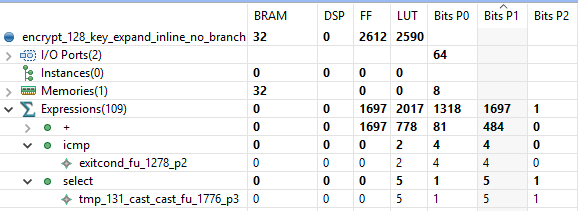


Η σύγκριση γίνεται στο 5ο κύκλο του ρολογιού:



**Select:**

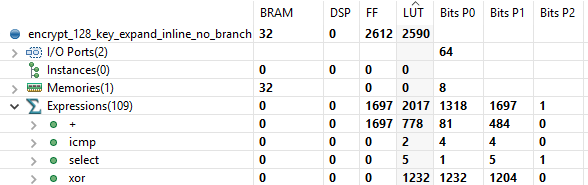
Στο κύκλωμα γίνεται ένα select που χρησιμοποιεί 5 Loop-Up Tables καθώς 1+5+1 bits αριθμών.



Η select γίνεται στο 6ο κύκλου του ρολογιού:



**XOR:**



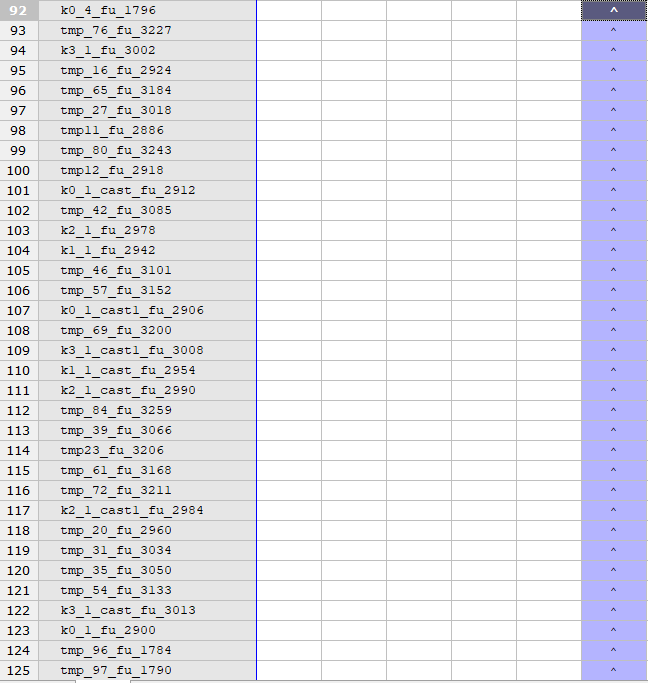
Βλέπουμε ότι όλες οι εντολές xor χρησιμοποιούνε 1232 Look-Up Table , καθώς γίνεται xor μεταξύ αριθμών 1232+1024 συνολικών bit αριθμών.

Τα xor είναι τα παρακάτω:

Αυτά που γίνονται στο 3ο και 4ο κύκλο του ρολογιού:

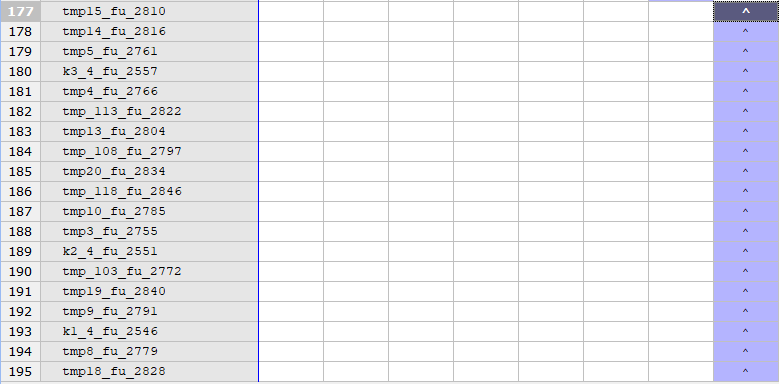


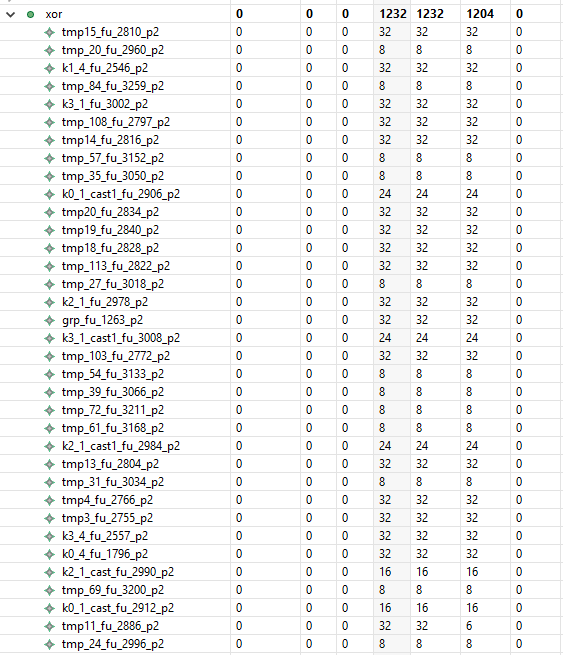
Αυτά που γίνονται στο 6ο κύκλο του ρολογιού:

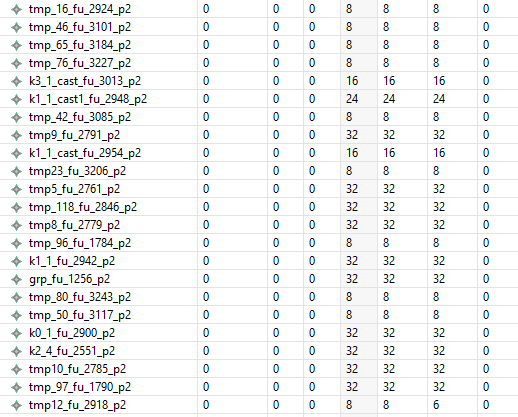




Αυτά που γίνονται στο 8ο κύκλο του ρολογιού:

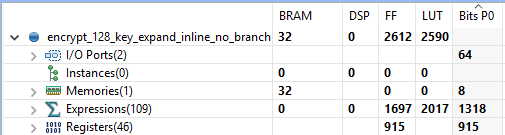






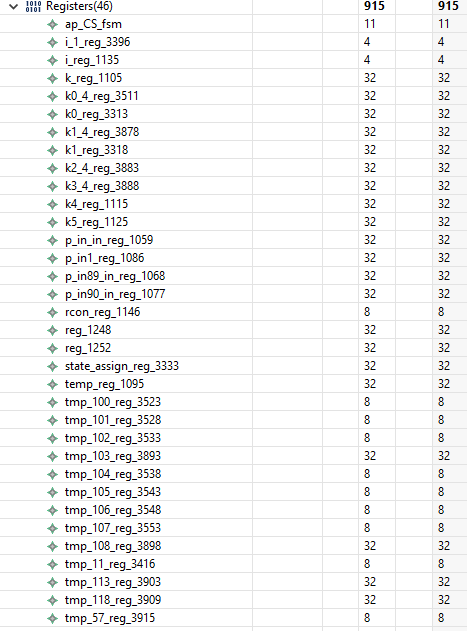
**Καταχωρητές:**

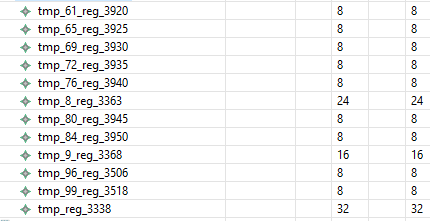
Βλέπουμε από τον πίνακα ότι ο κύκλωμα χρησιμοποιεί 46 καταχωρητές.



Τα οποία χρησιμοποιούν 915 Flip Flop καθώς το άθροισμα των bit όλων των καταχωρητών είναι 915 bits.

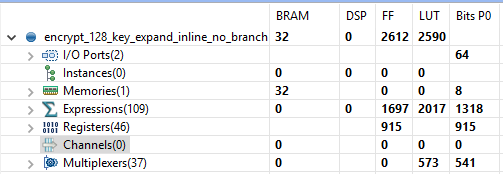
Παρακάτω είναι όλοι οι καταχωρητές:



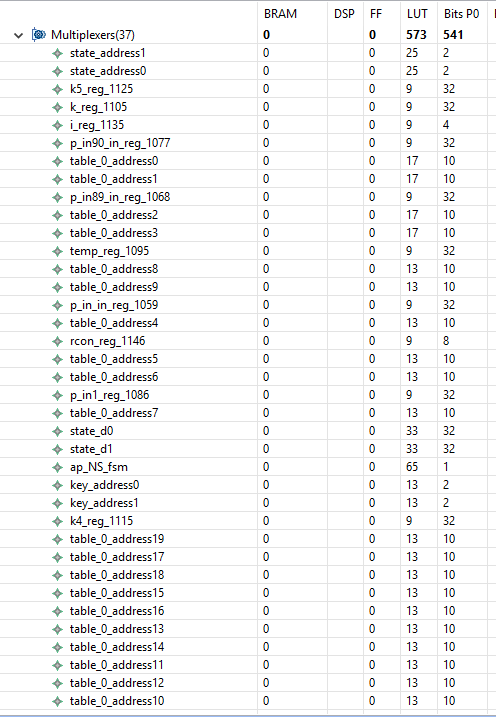


**Πολυπλέκτες:**

Το κύκλωμα χρησιμοποιεί 37 πολυπλέκτες εκ των οποίων έχουμε συνολικό άθροισμα Look Up table 573 καθώς και συνολικό αριθμό bit 541.



Βλέπουμε στην παρακάτω εικόνα όλους τους πολυπλέκτες του κυκλώματος:



**Instances:**

Σύμφωνα με τον πίνακα έχουμε 0 instances.Το 1ο κύκλωμα δεν εμπεριέχει συναρτήσεις που έχουν μεταφραστεί σε RTL Block.



Αν προσθέσουμε όλα τα resources που αναλύσαμε πιο πάνω, διαπιστώνουμε ότι το συνολικό κύκλωμα εμπεριέχει 32 BRAM, 2612 κυκλώματα Flip-Flop καθώς 2590 Look Up Tables.

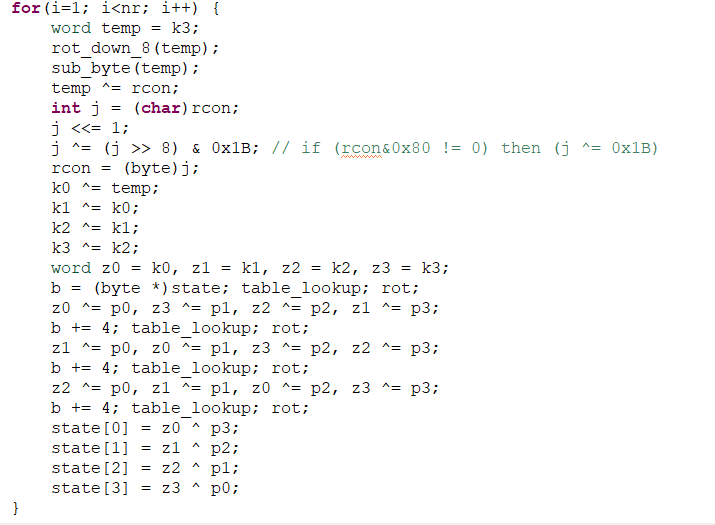
Έχει 51 Latency και 52 Interval.

**Loops:**



Βλέπουμε παραπάνω ότι εμπεριέχει ένα loop, το οποίο έχει καθυστερεί το κύκλωμα 5 που είναι ο αριθμός της καθυστέρησης ανά επανάληψη \* αριθμό των επαναλήψεων 9 = 45 latency.

Οι εντολές του που είναι στο loop είναι οι εξής:

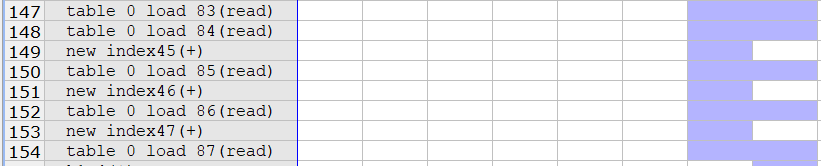


Απο τι βλέπουμε, στο βρόγχο ανήκουν εντολές για την δημιουργία διαφόρων καταχωρητών όπως είναι word, int. Επιπλέον, βλέπουμε οτι γίνονται πράξεις πολλές πράξεις xor ( ^ ) καθώς και shift left ( << ).

Και παρακάτω θα δούμε τους κύκλους:

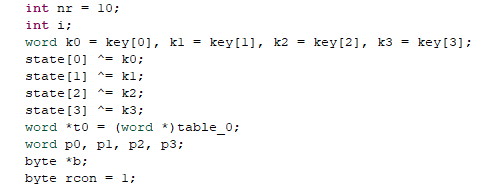


Βλέπουμε απο οτι αρχίζει απο τον 5ο κύκλο και τελειώνει στον 9ο.



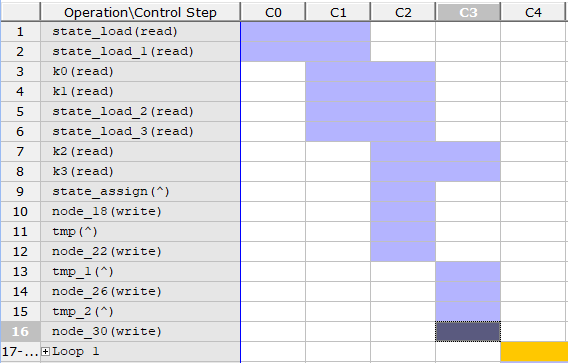
Βλέπουμε οτι τα πολλα read καθυστερούν το κύκλωμα.

Ο κώδικας που είναι εκτός loop:



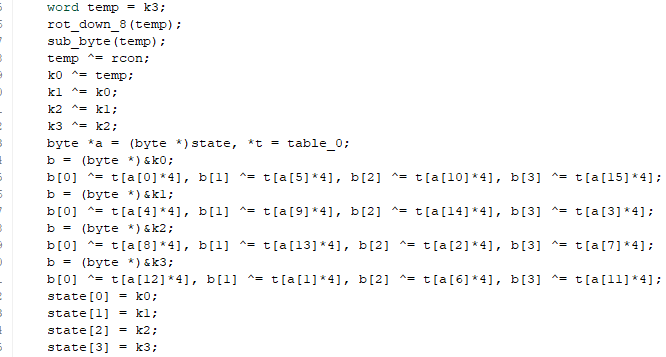
Στον παραπάνω κώδικα βλέπουμε εκτός απο εντολές εκχώρησης, πράξεις xor.

Στα οποία περιλαμβάνει:

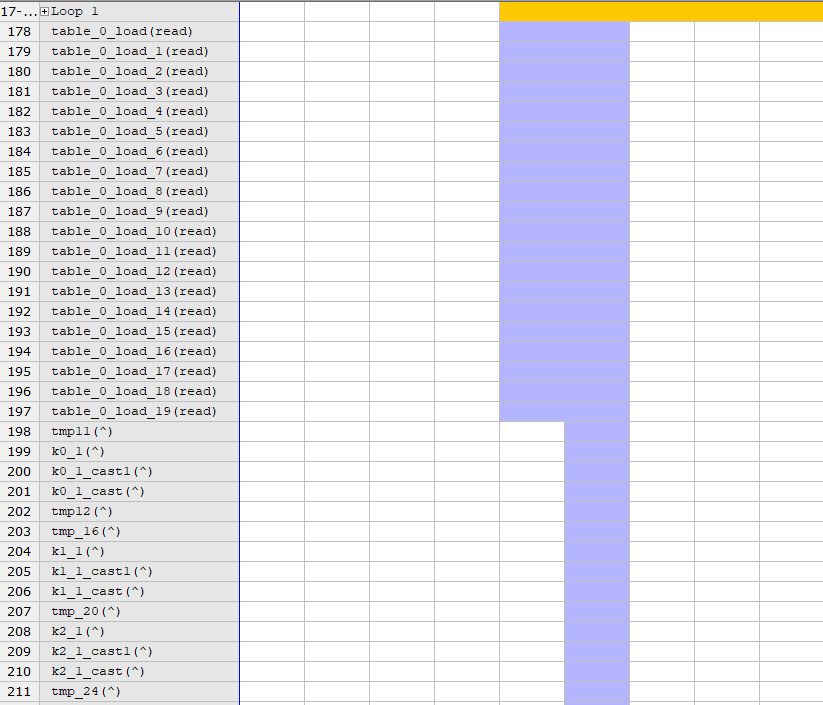
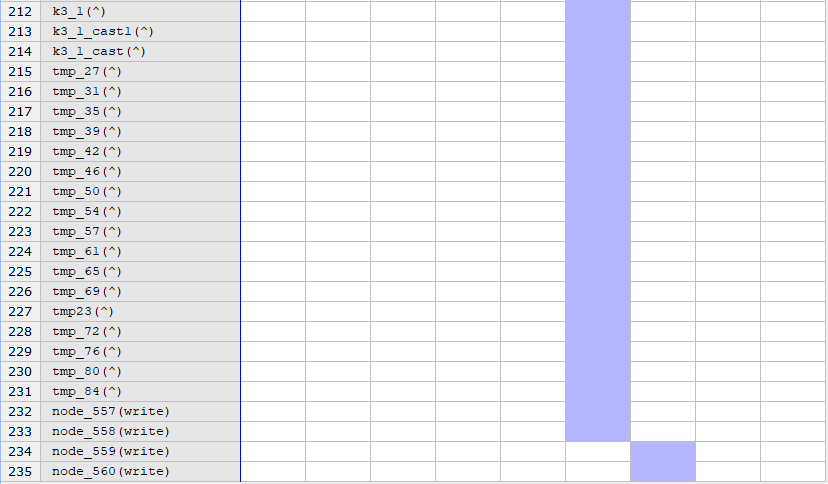


Σύμφωνα με τον πίνακα, κώδικα εντολές read, xor καθώς και write. Επιπλέον βλέπουμε οτι οι πιο «βαριές εντολές» είναι τα read αφού κάνουν 2 κύκλους.

Με τα το τέλος του Loop συναντάμε τις εντολές:



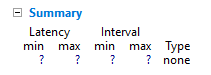
Βλέπουμε απο τον κώδικα και απο τον παρακάτω πίνακα:

Έχουμε εντολές τύπου read,xor,write εκτός απο εντολές εκχώρησης.

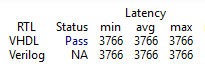
**2ο κύκλωμα:**

**Συνολικό Latency:**



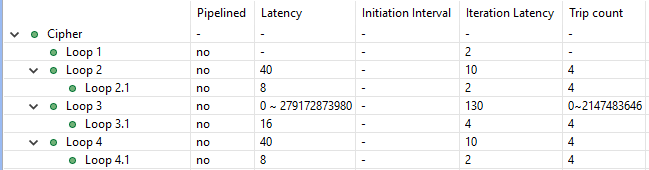
Για να βρούμε το Latency θα χρησιμοποιήσουμε το Latency από το Co-Simulation.

Ο λόγος είναι ότι σε ένα loop που εμπεριέχεται στο κύκλωμα, περιλαμβάνει μεταβλητές που τις εισάγει ο χρήστης. Αυτές οι μεταβλητές καθορίζουν το πότε θα τελειώσει το loop.



**Ανάλυση:**

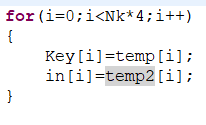
Το 2ο κύκλωμα εμπεριέχει τα παρακάτω loop:



Δηλαδή 4 loop.

To 1ο loop έχει καθυστέρηση εκτέλεσης 2 με άγνωστο αριθμό επαναλήψεων.

Κώδικας 1ο loop:



Απο τι βλέπουμε, στο παρακάτω βρόγχο υπάρχουν μόνο εντολές εκχώρησης. Λογικό είναι να έχει τοσο χαμηλό iteration latency. Εφόσον έχουμε loop, θα βρούμε και εντολές πρόσθεσης και σύγκρισης.

Κύκλους πιο αναλυτικά:

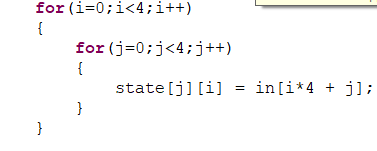


Βλέπουμε οτι αρχίζει στον 2ο κύκλο και τελειώνει στο 3ο.

Το 2ο loop έχει συνολική καθυστέρηση 40 καθώς εκτελεί 4 επαναλήψεις με κάθε επανάληψη να έχει 10 κύκλους καθυστέρηση.

Επιπλέον περιλαμβάνει εμφωλευμένο με αριθμό επαναλήψεων 4 καθώς με καθυστέρηση 8.

Κώδικας 2ο loop:



Απο τι βλέπουμε και εδώ υπάρχουν απλώς 4 \*4 καταχωρήσεις. Λογικό και εδώ να έχουμε χαμηλό iteration latency.

**Εφόσον έχουμε loop, θα βρούμε και εντολές πρόσθεσης και σύγκρισης.**

Κύκλους πιο αναλυτικά:



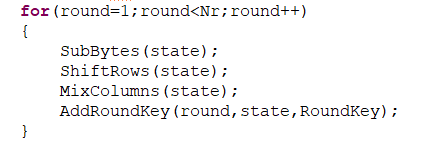
Βλέπουμε οτι αρχίζει στο 4ο κύκλο και τελειώνει στον 6ο κύκλο.

Το 3ο loop έχει συνολική καθυστέρηση από 0 ~ 279172873980 καθώς 130 καθυστέρηση ανά loop.

O αριθμός των επαναλήψεων είναι από 0 ~ 2147483646. Ο λόγος είναι επειδή περιλαμβάνεται στο loop ο αριθμός των επαναλήψεων που δίνεται από τον χρήστη που κανονικά θα είναι 10.

Επιπλέον περιλαμβάνεται στο 3ο , εμφωλευμένο με συνολική καθυστέρηση 16, που προκύπτει από τον αριθμό των επαναλήψεων \* την καθυστέρηση ανά επανάληψη που είναι 4\*4, το οποίο θα βρίσκεται σε καμία συνάρτηση.

Κώδικας 3ο loop:



Αυτός ο κώδικας θα αποτελέσει τον πιο βαρύ για το κύκλωμα.

Μέσα σε Nr γύρους θα πρέπει να εκτελέσει 4 \* Nr συναρτήσεις που έχουν το δικό τους χρόνο εκτέλεσης, χρόνους κύκλους ρολογιού καθώς και καθυστέρηση που θα προσθέσουν το κύκλωμα.

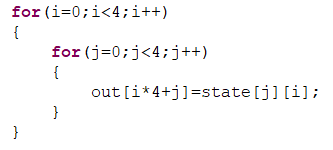
Κύκλους πιο αναλυτικά:



Βλέπουμε οτι αρχίζει στο 6ο κύκλο και τελειώνει στον 23ο.

Το 4ο loop έχει συνολική καθυστέρηση 40 που προκύπτει από το αριθμό των επαναλήψεων \* καθυστέρηση που είναι 10\*4. Εμπεριέχει και αυτό εμφωλευμένο με συνολική καθυστέρηση 2\*4=8.

Κώδικας 4ο loop:



Aπο τι βλέπουμε και εδώ, συναντάμε εντολές εκχώηρησης καθώς και γινομένου.

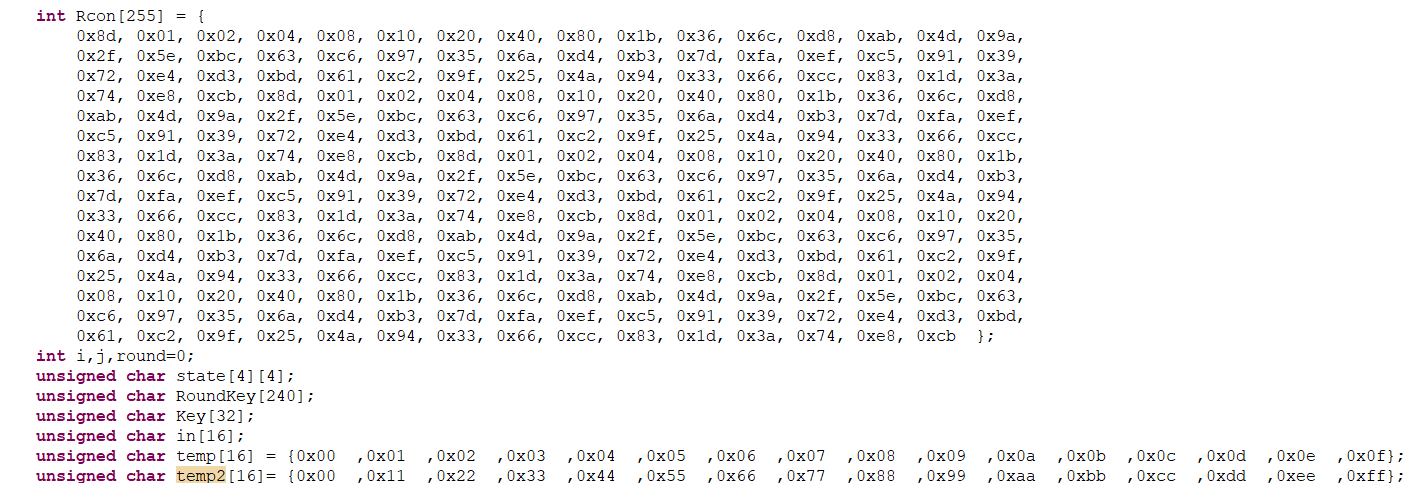
Κύκλους πιο αναλυτικά:



Βλέπουμε οτι αρχίζει στο 22ο και τελειώνει στο 24ο κύκλο.

**Εφόσον έχουμε loop, θα βρούμε και εντολές πρόσθεσης και σύγκρισης.**

Εντολές εκτος loop:



Βλέπουμε καταχωρητές.



Και εδώ βλέπουμε και συναρτήσεις που τις καλούμε.

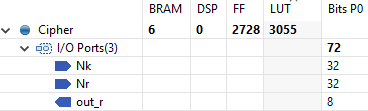
Σύμφωνα με το παρακάτω πίνακα χρησιμοποιεί 6 BRAM, Block Ram, 2728 Flip-Flop και 3055 Look Up Tables.



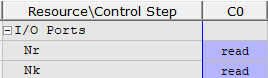
Πιο συγκεκριμένα, έχει τα παρακάτω:

**Resources:**

**I/O Ports:**

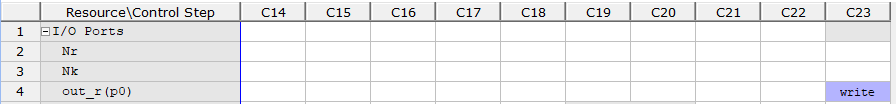


3 Ι/Ο Ports

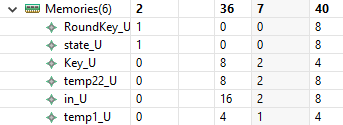


Τα οποία σύμφωνα με τους παραπάνω πίνακες το Nr,Nk είναι για read, ενώ το our για write.

Επιπλέον, διαβάζονται στο πρώτο κύκλο ρολογιού ενώ το out στο τελευταίο, αφού βγάζει το αποτέλεσμα.

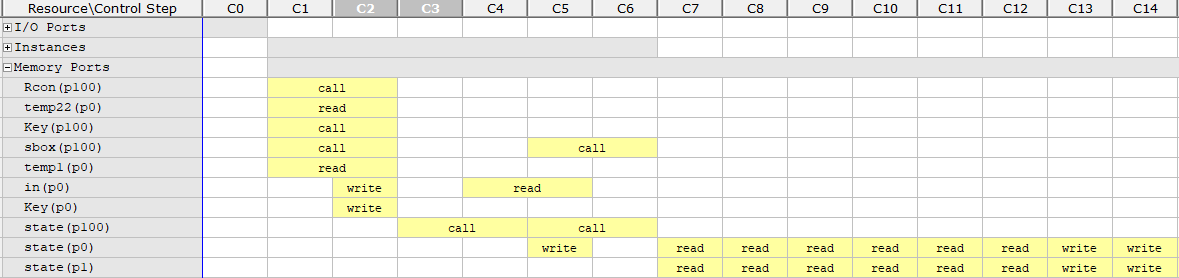


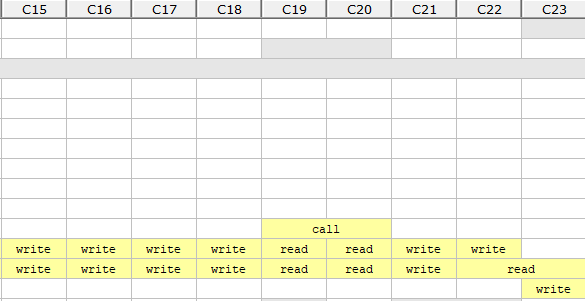
**Μνήμες:**



Σύμφωνα με τον παραπάνω πίνακα, εμπεριέχει το κύκλωμα 6 μνήμες, από τις οποίες 2 από τις 6 αποτελούν μνήμες BRAM και χρησιμοποιεί η κάθε μία από αυτές τις 2 , 8 bit. Επιπλέον, χρησιμοποιούνε 36 Flip-Flop, 7 Look Up Tables καθώς συνολικό αριθμών bits 40.

Τις βλέπουμε παρακάτω τις προσπελάσεις:

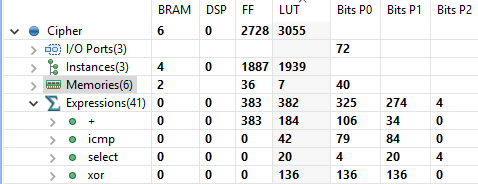




Βλέπουμε ότι ξεκινάνε στο 2ο κύκλο επειδή περιμένουν τα I/O Ports να διαβαστούν, και επιπλέον, βλέπουμε ότι κάποιες κάνουν 23 κύκλους ρολογιού μέχρι να ολοκληρωθούν.

**Expressions:**

Από τι βλέπουμε από τον πίνακα χρησιμοποιεί το κύκλωμα 41 expressions:

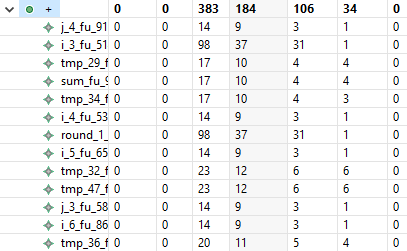


Στα οποία περιλαμβάνονται εντολές πρόσθεσης(+),σύγκρισης(icmp),select,xor(^).

**Προσθέσεις:**

Όλες οι προσθέσεις περιλαμβάνουν 383 Flip-Flop, 184 Look Up Tables καθώς 106+34 συνολικά bit αριθμών.

Με τόσες εντολές:



Οι προσθέσεις που γίνονται στις εντολές:

Στο 2ο κύκλο:



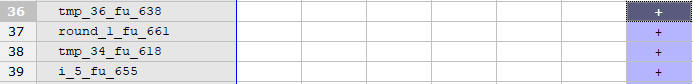
Στο 4ο κύκλο:



Στο 5ο κύκλο:



Στο 20ο κύκλο:



Στο 22ο κύκλο:



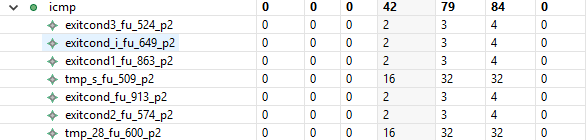
Στο 23ο κύκλο:



Συνολικές προσθέσεις:13

**Συγκρίσεις:**

Στο κύκλωμα οι συγκρίσεις χρησιμοποιούν 32 Look Up Tables καθώς 79+84 συνολικό αριθμό bits:



Οι συγκρίσεις που γίνονται στο κύκλωμα:

Στο 2ο κύκλο:



Στο 4ο κύκλο



Στο 5ο κύκλο:



Στο 6ο κύκλο:



Στο 20ο κύκλο:



Στο 22ο κύκλο:



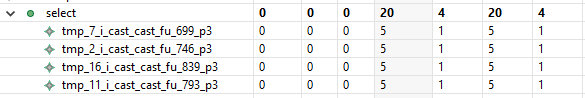
Στο 23 κύκλο:



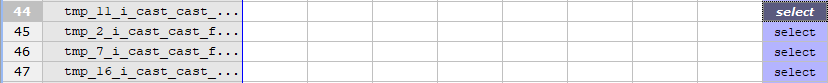
Συνολικές συγκρίσεις: 7

**Select:**

Στο κύκλωμα τα select που γίνονται, χρησιμοποιούν 20 Look Up Tables καθώς 4+20+4=28 συνολικό αριθμών bits.

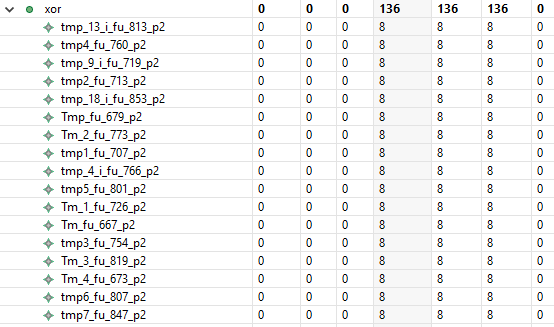


Τα select που γίνονται στο κύκλο 22

.

Ο συνολικός αριθμός των select είναι 4.

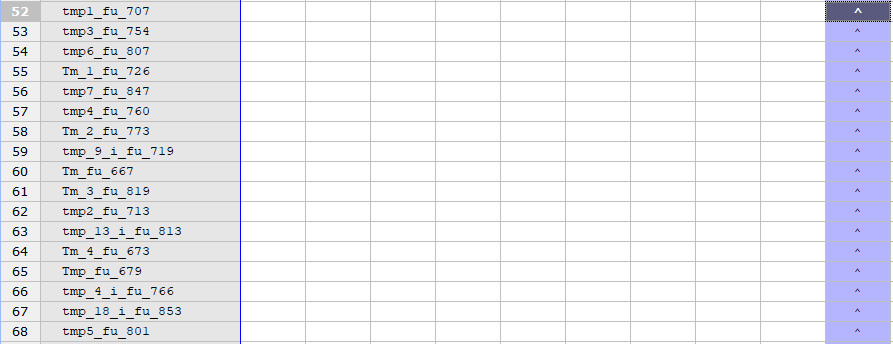
**XOR:**



Στο κύκλωμα όλα τα xor χρησιμοποιούνε συνολικά 136 Look Up Tables καθώς συνολικό αριθμό bits 136+136=272.

Τα xor τα συναντάμε στο κύκλωμα:

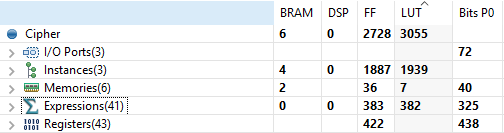
Στο 22ο κύκλο ρολογιού:



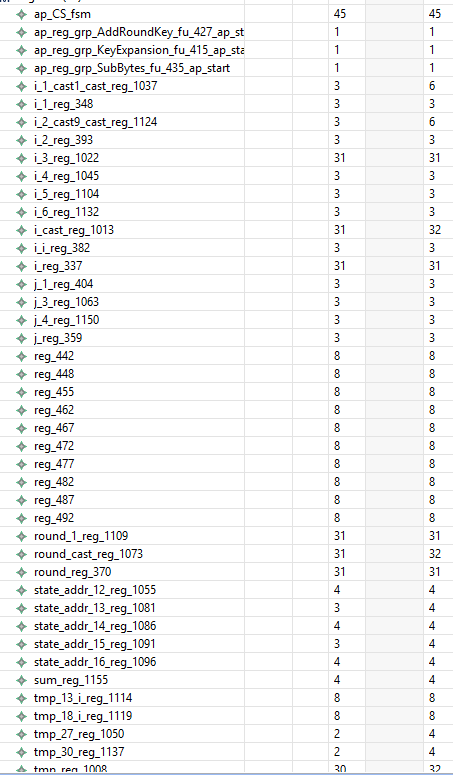
Συνολικός αριθμός xor: 16

**Καταχωρητές:**

Βλέπουμε από τον πίνακα ότι ο κύκλωμα χρησιμοποιεί 43 καταχωρητές, στα οποία εμπεριέχουν κυκλώματα 422 Flip-Flop καθώς 438 συνολικό αριθμών bits.

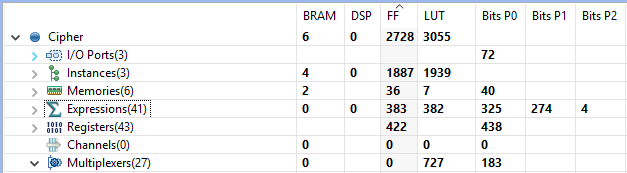


Οι καταχωρητές είναι οι παρακάτω, με αριθμό των Flip-flop και αριθμό bits.

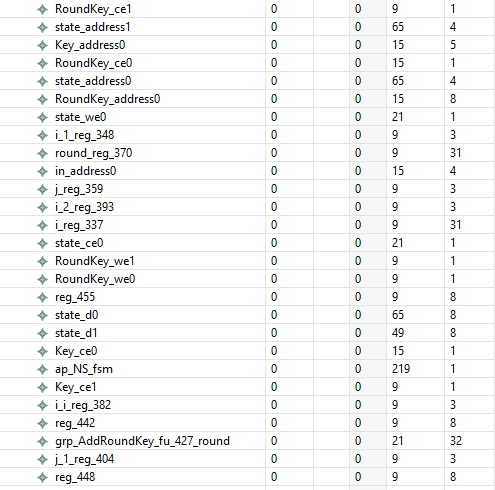


**Πολυπλέκτες:**

Το κύκλωμα χρησιμοποιεί 27 πολυπλέκτες εκ των οποίων έχουμε συνολικό άθροισμα Look Up table 727 καθώς και συνολικό αριθμό bit 183.

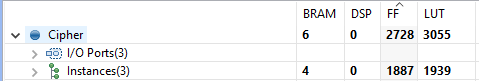


Και είναι οι παρακάτω:



**Instances:**

Τα instances αποτελούν functions, τα οποία έχουν RTL hierarchy.



Σύμφωνα με το παραπάνω πίνακα, έχουμε 3 instances τα οποία χρησιμοποιούνε 4 BRAM, 1887 κυκλώματα Flip-Flop καθώς 1939 Look Up Tables.

Τα instances είναι τα παρακάτω:



Βλέπουμε παραπάνω ότι μόνο το instance KeyExpansion έχει 1704 Flip-Flop, 1722 look Up tables, και 3 BRAM. Ενώ τα αλλά δύο έχουν 1/0 BRAM, 75/108 Flip-Flop και 100/117 Look Up Tables.

Τα instances στο κύκλωμα τα συναντάμε στο κύκλο:

2ο κύκλο:



4ο κύκλο:



6ο κύκλο:



Βλέπουμε ότι μόνο το call κρατάει 2 κύκλους ρολογιού.

**Ανάλυση KeyExpansion:**

Κώδικας:

**void** **KeyExpansion**(**unsigned** **char** RoundKey[240],**unsigned** **char** Key[32],**int** Rcon[255],**int** Nk,**int** Nr)

{

**int** i,j;

**unsigned** **char** temp[4],k;

// The first round key is the key itself.

**for**(i=0;i<Nk;i++)

{

RoundKey[i\*4]=Key[i\*4];

RoundKey[i\*4+1]=Key[i\*4+1];

RoundKey[i\*4+2]=Key[i\*4+2];

RoundKey[i\*4+3]=Key[i\*4+3];

}

// All other round keys are found from the previous round keys.

**while** (i < (Nb \* (Nr+1)))

{

**for**(j=0;j<4;j++)

{

temp[j]=RoundKey[(i-1) \* 4 + j];

}

**if** (i % Nk == 0)

{

// This function rotates the 4 bytes in a word to the left once.

// [a0,a1,a2,a3] becomes [a1,a2,a3,a0]

// Function RotWord()

{

k = temp[0];

temp[0] = temp[1];

temp[1] = temp[2];

temp[2] = temp[3];

temp[3] = k;

}

// SubWord() is a function that takes a four-byte input word and

// applies the S-box to each of the four bytes to produce an output word.

// Function Subword()

{

temp[0]=getSBoxValue(temp[0]);

temp[1]=getSBoxValue(temp[1]);

temp[2]=getSBoxValue(temp[2]);

temp[3]=getSBoxValue(temp[3]);

}

temp[0] = temp[0] ^ Rcon[i/Nk];

}

**else** **if** (Nk > 6 && i % Nk == 4)

{

// Function Subword()

{

temp[0]=getSBoxValue(temp[0]);

temp[1]=getSBoxValue(temp[1]);

temp[2]=getSBoxValue(temp[2]);

temp[3]=getSBoxValue(temp[3]);

}

}

RoundKey[i\*4+0] = RoundKey[(i-Nk)\*4+0] ^ temp[0];

RoundKey[i\*4+1] = RoundKey[(i-Nk)\*4+1] ^ temp[1];

RoundKey[i\*4+2] = RoundKey[(i-Nk)\*4+2] ^ temp[2];

RoundKey[i\*4+3] = RoundKey[(i-Nk)\*4+3] ^ temp[3];

i++;

}

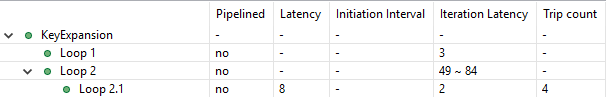
}

Απο τι βλέπουμε στον κώδικα , έχουμε κάποιους καταχωρητές ,2 loop με το 2ο να είναι εμφωλευμένο.

Απο πράξεις βλέπουμε προσθέσεις, συγκρίσεις για τα loop και για τα if.

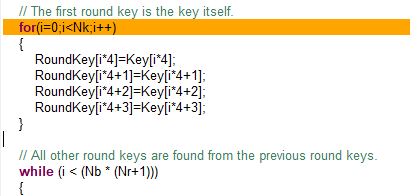
Εκτός απο αυτά βλέπουμε γίνομενα, xor καθώς και κάλεσμα συναρτήσεων όπως η getSBoxVlaue.

Για τα loop:

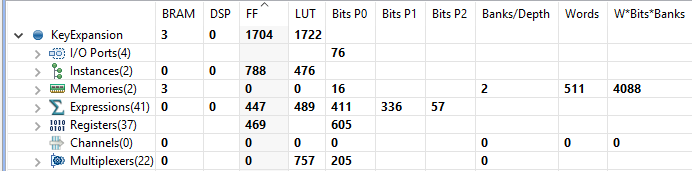


Βλέπουμε ότι το εμφωλευμένο έχει καθυστέρηση 8 , το οποίο βγήκε από αριθμό των επαναλήψεων \* καθυστέρηση της κάθε επανάληψης δηλαδή 2 \* 4.

Τα άλλα δύο περιμένουν την τιμή από την είσοδο συγκεκριμένα το Nk, Nb.

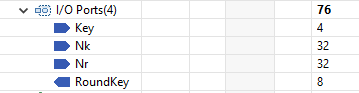


Από τα resources μπορούμε να δούμε πώς μόνο αυτή η συνάρτηση χρησιμοποιεί: 4 I/O Ports,2 instances , 2 memories , 41 expressions, 37 Registers και 22 Multiplexers.



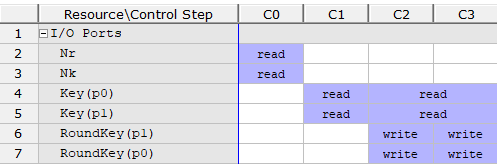
**I/O Ports:**

Από τι βλέπουμε χρησιμοποιεί 4 εκ των οποίων είναι:



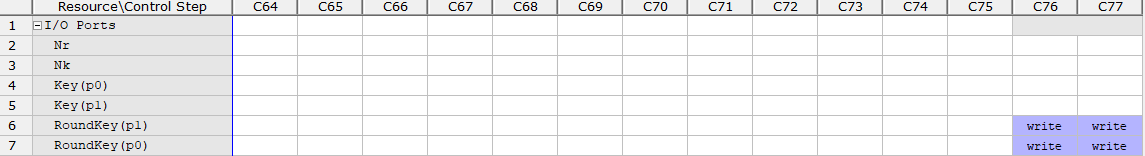
Key με 4 bits, Νκ-Νr με 32 bits καθώς RoundKey με 8 bits.

Bλέπουμε από τον παρακάτω πίνακα :



Ότι πρέπει να περάσουν 4 κύκλοι ρολογιού για να φορτώσει τα δεδομένα.

Και βλέπουμε ότι τελειώνουν στο 78 κύκλο ρολογιού.



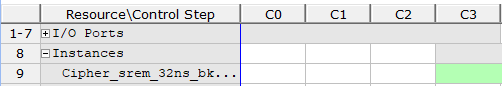
**Instances:**

Από τι είδαμε στο παραπάνω πίνακα, η συνάρτηση αυτή εμπεριέχει 2 instances, οι οποίες είναι :

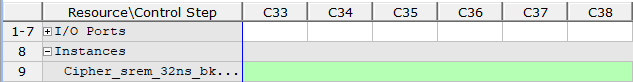


Η κάθε μια χρησιμοποιεί 394 Flip-Flop καθώς και 238 Look Up Tables.

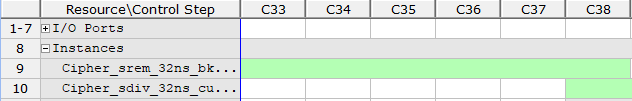
Το πρώτο instance ξεκινάει από τι βλέπουμε στο 4ο κύκλο του ρολογιού δηλαδή μετά την φόρτωση των δεδομένων.



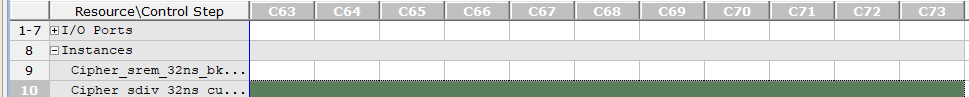
Και τελειώνει στο 39ο κύκλο.



Ενώ το 2ο instance ξεκινάει στο 39ο κύκλο



Και τελειώνει στο 74ο κύκλο.



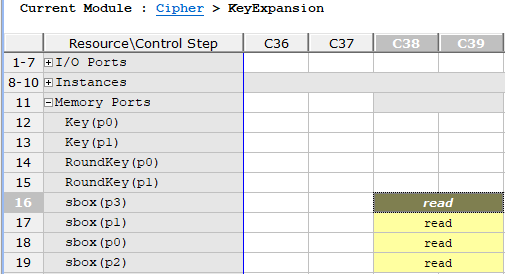
Βλέπουμε πόσο επιβαρύνουν το κύκλωμα τα srem,div.

**Μνήμες:**

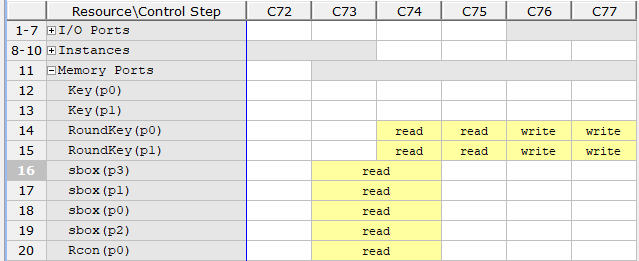
Από τι είδαμε παραπάνω, χρησιμοποιεί 2 μνήμες, οι οποίες έχουν 3 BRAM, καθώς συνολικά 16 bits αριθμών. Οι μνήμες αυτές είναι:



Και βρίσκονται στο 39ο κύκλο ρολογιού και κρατάνε 2 κύκλους.



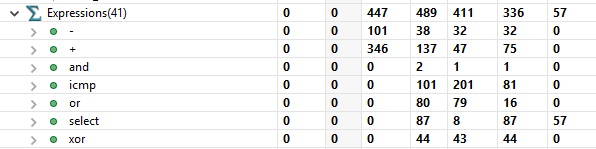
Καθώς και στο 74ο κύκλο:



To Round Key επειδή είναι input , το αποθηκεύουμε στην μνήμη.

**Expressions:**

Βλέποντας τον αρχικό πίνακα της συνάρτησης θα δούμε ότι έχει και expressions, τα οποία είναι 41 , χρησιμοποιούνε 447 FF, 489 LUT καθώς 411+336+77 συνολικό bit αριθμών. Από τι βλέπουμε παρακάτω εκτός από προσθέσεις, έχουμε αφαιρέσεις , and ,συγκρίσεις ,or , select καθώς και xor.



**Αφαιρέσεις:**

Από τι βλέπουμε είναι μόνο μια καθώς εκτελείται στο 75ο κύκλο:



**Προσθέσεις:**

Στο 2ο κύκλο:



Στο 4ο κύκλο:



Στο 75ο κύκλο:



Συνολικές προσθέσεις :5

**AND:**

Από τι βλέπουμε είναι μόνο μια καθώς εκτελείται στο 39ο κύκλο:



**Compare:**

Στο 1ο κύκλο:



Στο 2ο κύκλο:



Στο 3ο κύκλο:



Στο 4ο κύκλο:



Στο 5ο κύκλο:



Στο 39ο κύκλο:



Συνολικός αριθμός compare:9

**OR:**

2ο κύκλο:



3ο κύκλο:



5ο κύκλο:



75ο κύκλο:



76ο κύκλο:



77ο κύκλο:



78ο κύκλο:



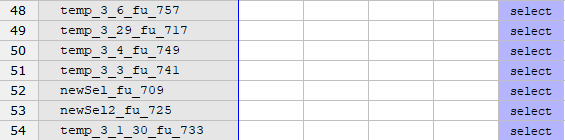
Συνολικός αριθμός OR : 10

**Select:**

1ο Κύκλο:



5ο κύκλο:



Συνολικός αριθμός select:8

**XOR:**

4ο κύκλο:



75ο κύκλο:



76ο κύκλο:



77ο κύκλο:

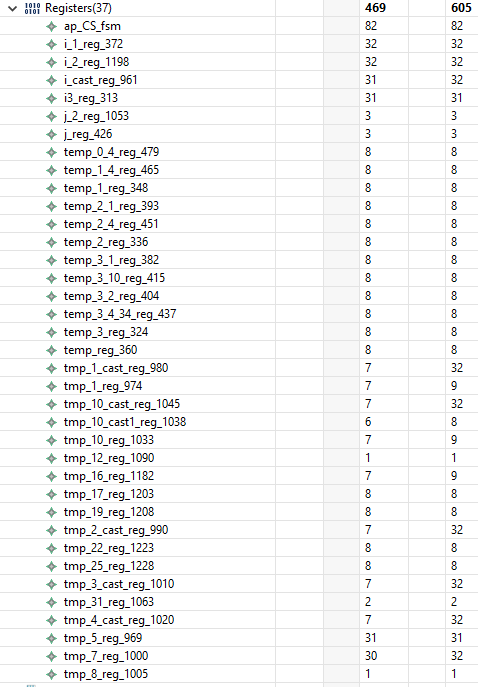


**Καταχωρητές:**



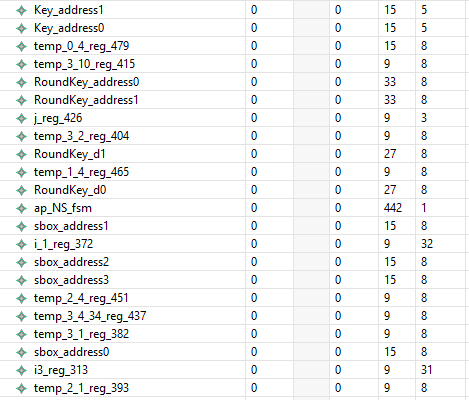
Από τι βλέπουμε εχουμε 37 καταχωρητές, καθώς χρησιμοποιούνε 469 Flip-Flop και 605 συνολικό αριθμό bit.

Οι καταχωρητές είναι οι παρακάτω:

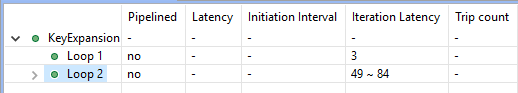


**Πολυπλέκτες:**

Είναι οι ακόλουθοι:



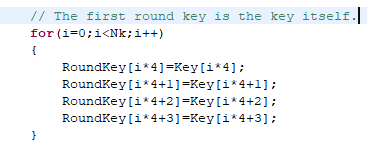
**LOOP:**

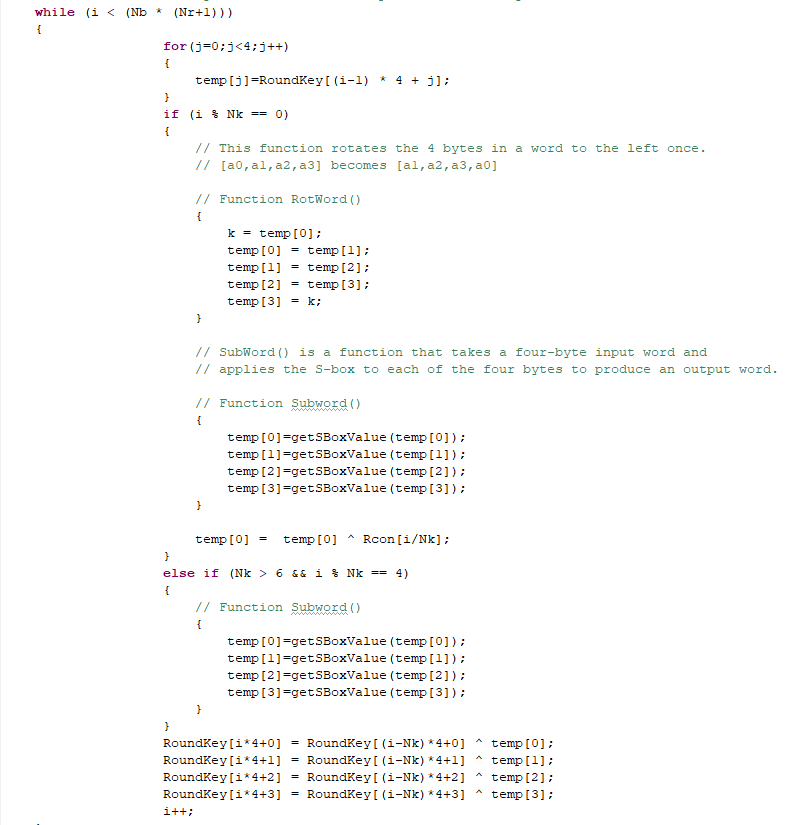
Θα εξετάσουμε εδώ τα loop:  


Όπως είδαμε έχουμε 2 loop, το 1ο τελειώνει μετά από 3 κύκλους ενώ το άλλο ποικίλλει από 49 μέχρι 84 λόγων συνθήκων (if).

Θα δούμε τους κώδικες των loop οι οποίοι αποτελούν όλο τον κώδικα της συνάρτησης:

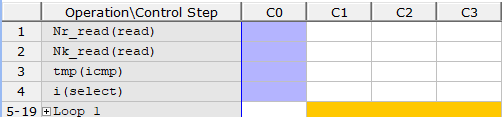
1ο loop:



2ο loop μαζί με το εμφωλευμένο.

Και τώρα θα δούμε την απόδοση τους:

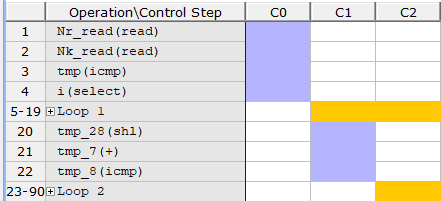
1ο loop:



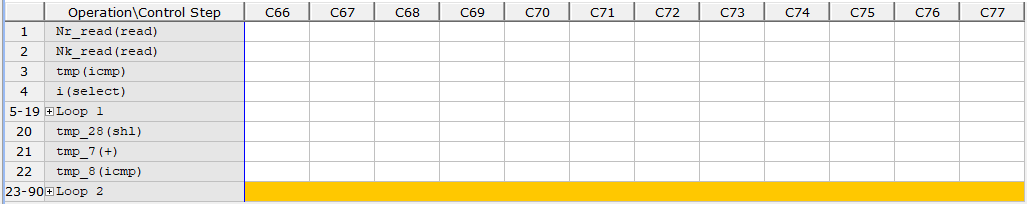
Αρχίζει στο 2ο κύκλο και τελειώνει στο 4ο.

2ο loop:

Αρχίζει στο 3ο κύκλο:

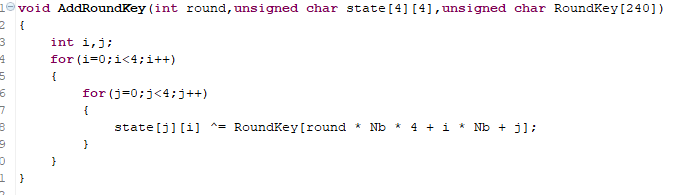


Και τελειώνει στο 78ο.



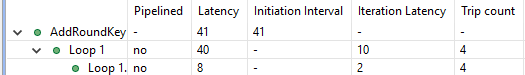
**Ανάλυση AddRoundKey:**

Κώδικας:



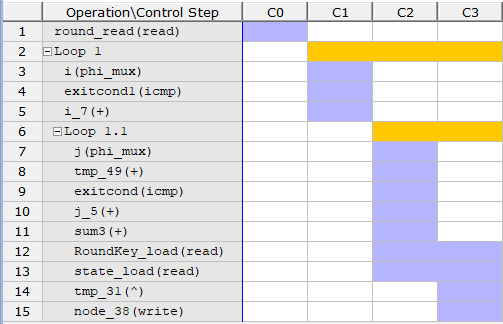
Από τι βλέπουμε αποτελείται 1 loop καθώς κα ένα εμφωλευμένο.

Οι βασικές πράξεις που βλέπουμε εκτός απο πρόσθεση, σύγκριση είναι και xor και γίνομενο.



Με Latency 40, τα οποία τα 8 προέρχονται από το εμφωλεύμενο loop.

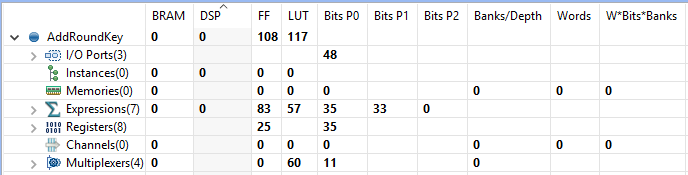
Πρώτα τελειώνει το εμφωλευμένο, άρα θα δούμε τους κύκλους που κάνει αυτό.



Βλέπουμε ότι ξεκινάει στο 3ο κύκλο και τελειώνει στο 4ο.

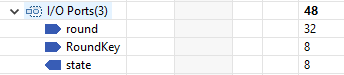
Ενώ το εξωτερικό loop ξεκινάει στο 2ο και τελειώνει στο 4ο.

Από τα resources μπορούμε να δούμε ότι έχει:



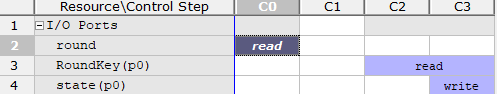
**I/O Ports:**

Συνολικός αριθμός :3



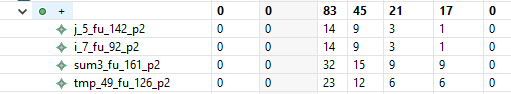
Και χρησιμοποιούνε συνολικά 48 bits αριθμών.

Και τα βλέπουμε στους συγκεκριμένους κύκλους:



**Expressions:**

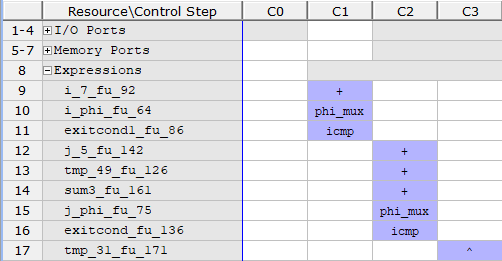
**Προσθέσεις:**



Συνολικός αριθμός :4

Με χρήση 83 FF , 45 LUT, 21 + 17 αριθμούς bits.

Και τα συναντάμε στους κύκλους:



**Συγκρίσεις:**



Αριθμός συγκρίσεων: 2

Συνολικός αριθμός LUT:4

Συνολικός αριθμός Bits:6+8

Τα συναντάμε στους κύκλους στο παραπάνω πίνακα.

**XOR:**

Συνολικός αριθμός xor:1



Συνολικός αριθμός LUT:8

Συνολικός αριθμός Bits:8+8

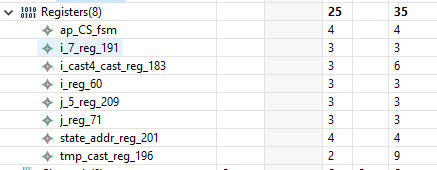
Τoν συναντάμε στον κύκλο στο παραπάνω πίνακα.

**Καταχωρητές:**

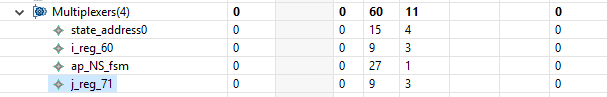
Συνολικός αριθμός:8

Χρήση 25 FF

Χρήση 35 Bits



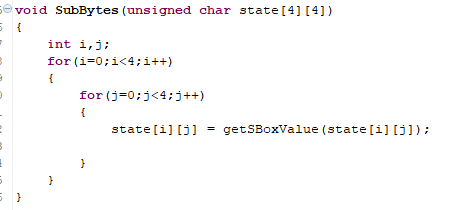
**Πολυπλέκτες:**



Συνολικός αριθμός 4 με χρήση 60 LUT , 11 bits.

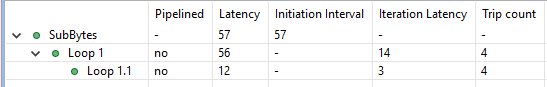
**Ανάλυση SubBytes:**

Κώδικας:



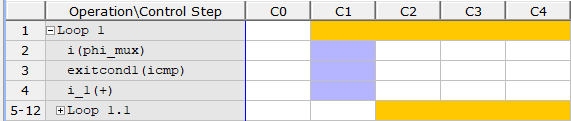
Από τι βλέπουμε αποτελείται 1 loop καθώς και ένα εμφωλευμένο.

Εκτός απο πρόσθεση, σύγκριση βλέπουμε και κάλεσμα συνάρτησης.



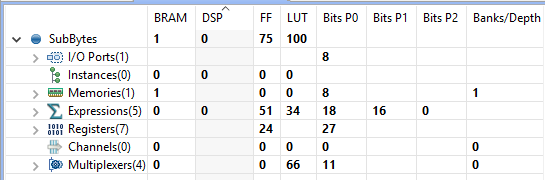
Και σύμφωνα με τον πίνακα το εμφωλευμένο έχει συνολικό latency 3\*4=12

Θα δούμε τώρα τους κύκλους, πρώτα του nested loop, γιατί πρέπει να τελειώσει, για να τελειώσει και το εξωτερικό.



Βλέπουμε ότι αρχίζει στο 3ο κύκλο και τελειώνει στο 5ο κύκλο, ενώ το εξωτερικό αρχίζει στο 2ο κύκλο και τελειώνει στο ίδιο.

Από τα resources μπορούμε να δούμε ότι έχει:



**I/O Ports:**

Συνολικός αριθμός:1



Χρήση Bits: 8

Τον συναντάμε στον 5ο κύκλο:



**Μνήμες:**

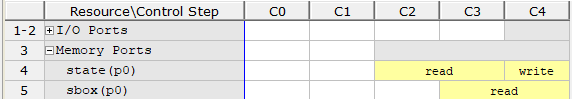
Συνολικός αριθμός:1



Χρήση BRAM:1

Χρήση Bits:8

Τον συναντάμε στον 3ο κύκλο:



**Expressions:**

Συνολικός αριθμός :5

Συνολικός Αριθμός FF:51

Συνολικός Αριθμός LUT:34

Συνολικός Aριθμός BIT:18+16



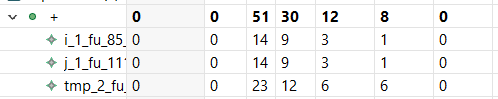
**ADD:**

Αριθμός Add:3

Αριθμός FF:51

Αριθμός LUT:30

Aριθμός BIT:12+8

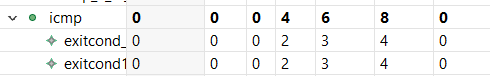


**COMPARE:**

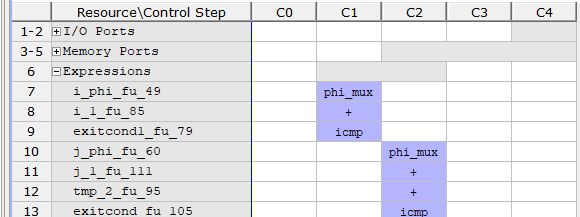
Aριθμός compare:2

Αριθμός FF:4

Αριθμός BIT: 6+8



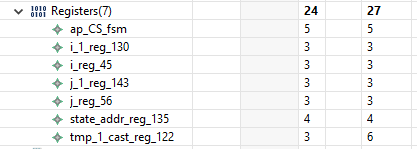
Τα βρίσκουμε στους κύκλους:



**Καταχωρητές:**

Αριθμός FF:24

Aριθμός BIT:27



**Πολυπλέκτες:**

Αριθμός LUT:66

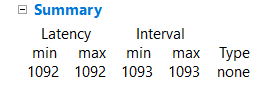
Aριθμός BIT:11



**3ο κύκλωμα:**

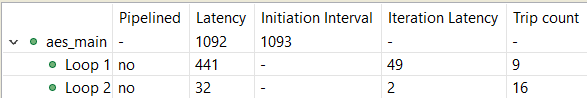
**Συνολικό Latency:**

Το 3ο κύκλωμα έχει βγάλει το παρακάτω Latency από την σύνθεση:



**Ανάλυση:**

Το 3ο κύκλωμα απο τι βλέπουμε απο την εικόνα έχει 2 loop.



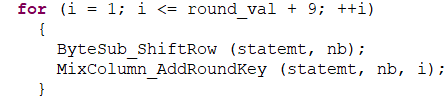
Απο τι βλέπουμε αυτά επιβαρύνουν το κυκλωμα κατά 441+32=473 latency.

Επίσης βλέπουμε οτι το 1ο loop επαναλαμβάνεται 9 φορές με καθυστέρηση ανά loop 49 ενώ το δεύτερο επαναλαμβάνεται κατα 16 φορές με καθυστέρηση 2.

Το περισευόμενο latency προκύπτει απο τον υπόλοιπο κώδικα που είναι εκτός των 2 loop.

Ο κώδικας των loop είναι ο παρακάτω:

1ο loop:

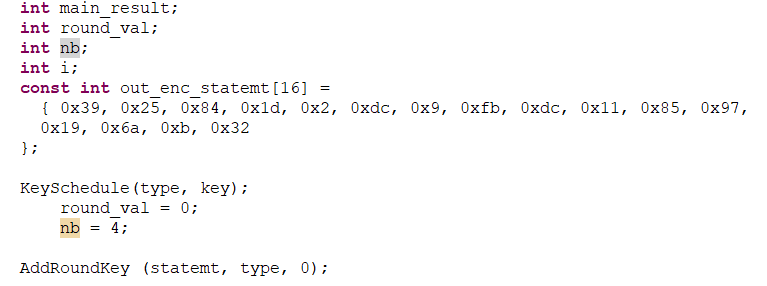


2ο loop:

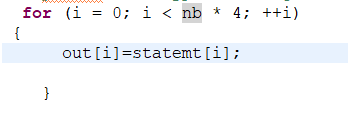
΄

Όπως βλέπουμε, το 1ο loop καλεί 2 συναρτήσεις, άρα θα επιρρεάσει εξίσου το latency , ο χρόνος εκτέλεσης των συναρτήσεων. Αυτό ισχύει και στο 2ο loop που κάνει πρόσθεση και αμέσως καλεί την συνάρτηση out\_enc\_statemt[i].

Ο υπόλοιπος κώδικας είναι ο αυτός:



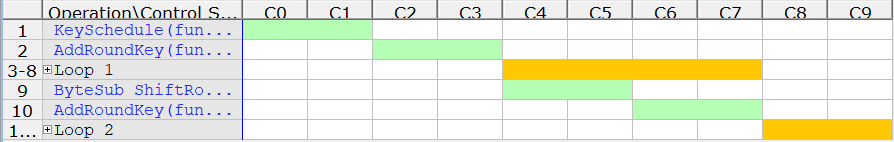




Που βλέπουμε κάποιους καταχωρητές, καθώς και κάλεσμα συνάρτησεων.

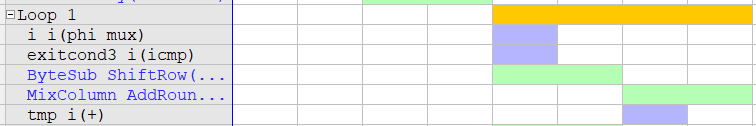
Ενώ βλέπουμε και ένα loop ,το οποίο αυτο που κάνει είναι κάποιες καταχωρήσεις χωρίς να επιβαρύνει σημαντικά το κύκλωμα και για αυτον τον λόγο δεν το βάζει το εργάλειο στον πίνακα με τα loops.

Με την παρακάτω εικόνα θα δούμε τους κύκλους που κάνουν περίπου οι εντολές που είναι στο κύκλωμα.

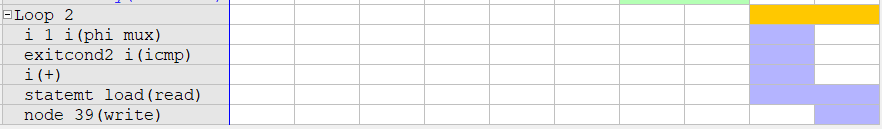


Βλέπουμε οτι στην αρχή καλείται η συνάρτηση KeySchedule στον 1ο κύκλο.

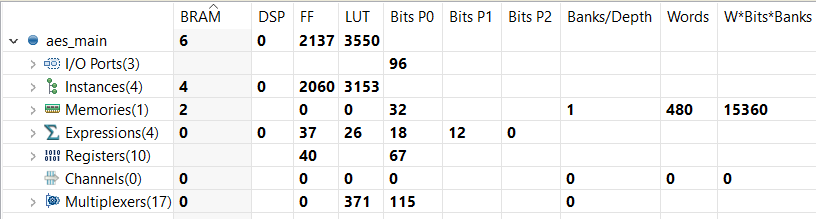
Έπειτα καλείται η AddRoundKey, και στην συνέχεια το 1ο loop.

Το 1ο loop εμπεριέχει όπως θα δούμε παρακάτω κάποιες πράξεις, καθώς και δύο συναρτήσεις .

Μετά τις 2 συναρτήσεις που καλούνται μετα το πέρας του 1ου loop, καλείται και το 2ο loop που εμπεριέχει τους εξής κύκλους και πράξεις:



Τα resources που χρησιμοποιεί το συνολικό κύκλωμα είναι τα παρακάτω:

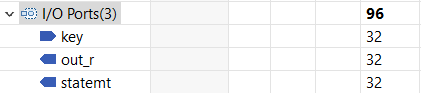


6 ΒRAM, 66 Flip-Flop καθώς και 3550 Look Up Tables.

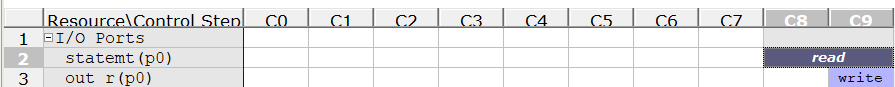
**I/O Ports:**

Συνολικός αριθμός: 3

Από τι βλέπουμε χρησιμοποιούν συνολικό αριθμό bits 96 και είναι τα παρακάτω:



Επιπλέον, τα συνάνταμε στους παρακάτω κύκλους:



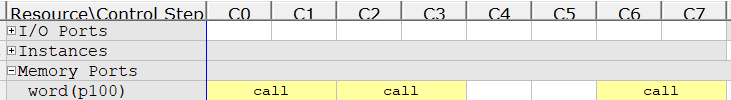
Δηλαδή στο 8ο για read καθώς και στο 9ο για write.

**Μνήμες:**

Απο τι βλέπουμε, απο τον πίνακα χρησιμοποιεί μια μνήμη η οποία έχει 2 BRAM, καθώς συνολικό αριθμό bits 32. Το συνολικό μέγεθος που καταναλώνει είναι 15360.



Και την βρίσκουμε στους κύκλους:

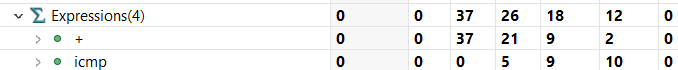


Στο 1ο ,3ο και στο 7ο κύκλο.

**Expressions:**

Συνολικός αριθμός: 4

Απο τον πίνακα βλέπουμε οτι υπάρχουν 2 είδη expressions.



Ο Συνολικός αριθμός FF είναι 37 ,26 LUT καθώς 18+12 συνολικά bits αριθμών.

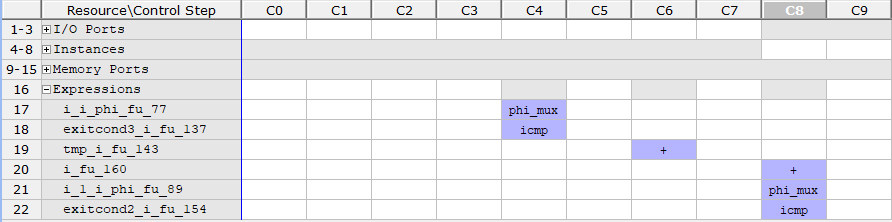
**Προσθέσεις:**

Συνολικός αριθμός: 2

Χρησιμοποιούν 37 FF ,26 LUT και συνολικό bits αριθμών 9+2=11.



Και τους συναντάμε στους παρακάτω κύκλους:



Δηλαδή στο 7ο και 9 κύκλο.

**Συγκρίσεις:**

Συνολικός αριθμός: 2

Χρησιμοποιούν 5 LUT και συνολικό bits αριθμών 9+10=19.

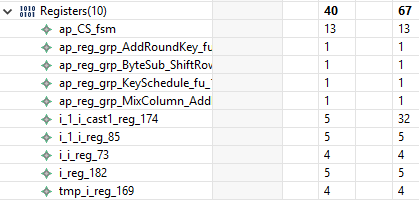


Και τους συναντάμε στους κύκλους που αναφέρει η παραπάνω εικόνα, δηλαδή στον 5ο και 9ο κύκλο.

**Καταχωρητές:**

Συνολικός αριθμός: 10

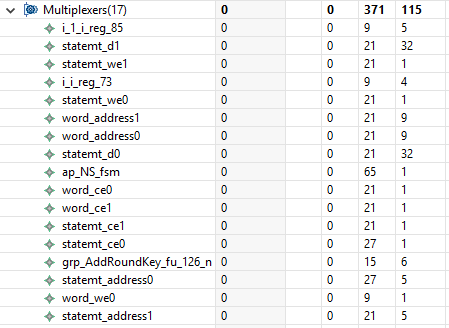
Χρησιμοποιούν 40 FF και συνολικό bits αριθμών 67 και είναι οι παρακάτω:



**Πολυπλέκτες:**

Συνολικός αριθμός: 17

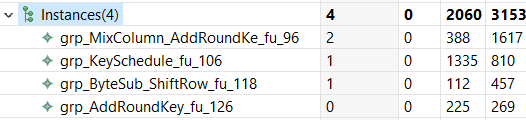
Χρησιμοποιούν 371 LUT και συνολικό bits αριθμών 115 και είναι οι παρακάτω:



**Instances:**

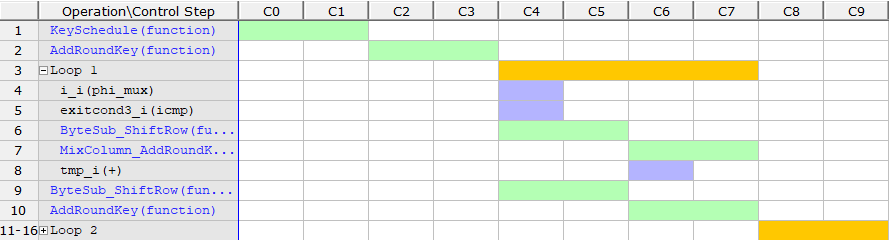
Συνολικός Αριθμός: 4

Τα instances είναι τα παρακάτω:



Βλέπουμε οτι χρησιμοποιούνε συνολικό αριθμό FF 2060, καθώς και 3153 look Up Tables.

Εκτός απο αυτά βλέπουμε οτι χρησιμοποιούνε και συνολικό αριθμό BRAM 4.



Από την παραπάνω εικόνα βλέπουμε ότι τα instances:

MixColumn\_AddRoundKey αρχίζει να εκτελείται στο 7ο κύκλο και είναι μέσα στο 1ο loop.

KeySchedule αρχίζει να εκτελείται στο 1ο κύκλο.

ByteSub\_ShiftRow αρχίζει να εκτελείται στο 5ο κύκλο. Εκτελείται 2 φορές , μια εντός loop και μία εκτός

AddRoundKey εκτελείται στο 3ο και 7ο κύκλο.

**Ανάλυση MixColumn\_AddRoundKey:**

Κώδικας:

**int** **MixColumn\_AddRoundKey** (**int** statemt[32], **int** nb, **int** n)

{

**int** ret[8 \* 4], j;

**register** **int** x;

**for** (j = 0; j < nb; ++j)

{

ret[j \* 4] = (statemt[j \* 4] << 1);

**if** ((ret[j \* 4] >> 8) == 1)

ret[j \* 4] ^= 283;

x = statemt[1 + j \* 4];

x ^= (x << 1);

**if** ((x >> 8) == 1)

ret[j \* 4] ^= (x ^ 283);

**else**

ret[j \* 4] ^= x;

ret[j \* 4] ^=

statemt[2 + j \* 4] ^ statemt[3 + j \* 4] ^ word[0][j + nb \* n];

ret[1 + j \* 4] = (statemt[1 + j \* 4] << 1);

**if** ((ret[1 + j \* 4] >> 8) == 1)

ret[1 + j \* 4] ^= 283;

x = statemt[2 + j \* 4];

x ^= (x << 1);

**if** ((x >> 8) == 1)

ret[1 + j \* 4] ^= (x ^ 283);

**else**

ret[1 + j \* 4] ^= x;

ret[1 + j \* 4] ^=

statemt[3 + j \* 4] ^ statemt[j \* 4] ^ word[1][j + nb \* n];

ret[2 + j \* 4] = (statemt[2 + j \* 4] << 1);

**if** ((ret[2 + j \* 4] >> 8) == 1)

ret[2 + j \* 4] ^= 283;

x = statemt[3 + j \* 4];

x ^= (x << 1);

**if** ((x >> 8) == 1)

ret[2 + j \* 4] ^= (x ^ 283);

**else**

ret[2 + j \* 4] ^= x;

ret[2 + j \* 4] ^=

statemt[j \* 4] ^ statemt[1 + j \* 4] ^ word[2][j + nb \* n];

ret[3 + j \* 4] = (statemt[3 + j \* 4] << 1);

**if** ((ret[3 + j \* 4] >> 8) == 1)

ret[3 + j \* 4] ^= 283;

x = statemt[j \* 4];

x ^= (x << 1);

**if** ((x >> 8) == 1)

ret[3 + j \* 4] ^= (x ^ 283);

**else**

ret[3 + j \* 4] ^= x;

ret[3 + j \* 4] ^=

statemt[1 + j \* 4] ^ statemt[2 + j \* 4] ^ word[3][j + nb \* n];

}

**for** (j = 0; j < nb; ++j)

{

statemt[j \* 4] = ret[j \* 4];

statemt[1 + j \* 4] = ret[1 + j \* 4];

statemt[2 + j \* 4] = ret[2 + j \* 4];

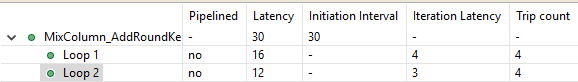
statemt[3 + j \* 4] = ret[3 + j \* 4];

}

**return** 0;

}

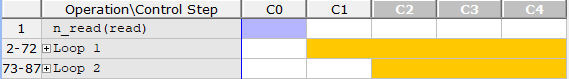
Από τι βλέπουμε αποτελείται από 2 loop, τα οποία σύμφωνα από τον παρακάτω πίνακα εκτελούνται 4 φορές το κάθε ένα με την διαφορά ότι το 1ο loop έχει κατά 1 μεγαλύτερο latency από το 2ο.



Και αυτό φαίνεται λογικό από τις πράξεις και από τον όγκο των δεδομένων που υπάρχει στο 1ο loop.

Το υπόλοιπο 30-28=2 latency, προκύπτει από το υπόλοιπο κώδικα που βρίσκεται εκτός των βρόγχων και θα είναι η δημιουργία, φόρτωση των καταχωρητών για να ξεκινήσουν τα loop, εφόσον χρείαζονται το nb που το παίρνει σαν όρισμα η συνάρτηση.

Το βλέπουμε και παρακάτω:



**Resources:**

Η παραπάνω συνάρτηση από τι βλέπουμε από τον πίνακα έχει συνολικά:

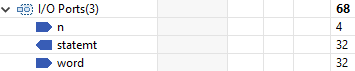
2 BRAM, 388 FF , 1617 LUT.



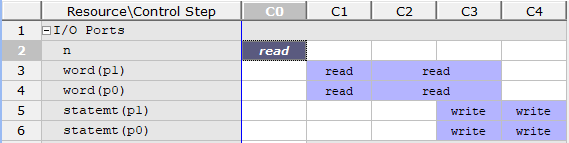
**I/O Ports:**

Συνολικός αριθμός: 3

Καθώς χρησιμοποιούν τα ports 68 συνολικά bits.



Τα συναντάμε στους εξής κύκλους:



1ο , 2ο και 3 κύκλο.

Τα word/ statement πιάνουν 2 κύκλους για το μέγεθος τους που είναι 32 bit.

**Μνήμες:**

Συνολικός αριθμός: 1

Βλέπουμε ότι δεσμεύει 2 BRAM ,32 Bits P0 καθώς τον χώρο 32\*32=1024 λόγω BRAM.



Την συναντάμε στον παρακάτω κύκλο:



Στον 3ο καθώς τελειώνει στο 5ο κύκλο.

**Expressions:**

Συνολικός αριθμός: 58

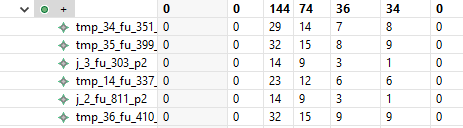
Ο Συνολικός αριθμός FF είναι 144 ,1387 LUT καθώς 1226+1092+256 συνολικά bits αριθμών.



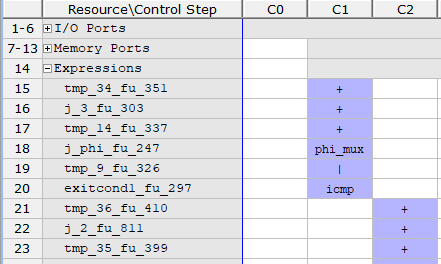
**Προσθέσεις:**

Συνολικός αριθμός: 6

Ο Συνολικός αριθμός FF είναι 144 ,1387 LUT καθώς 36+34 συνολικά bits αριθμών.



Και τα συναντάμε στους παρακάτω κύκλους:

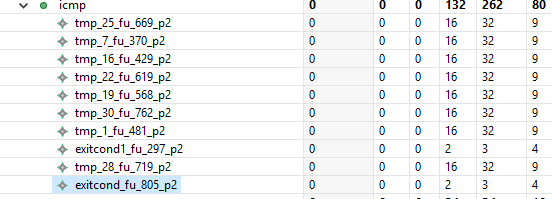


2ο και 3ο κύκλο.

**Συγκρίσεις:**

Συνολικός αριθμός: 10

Ο Συνολικός αριθμός LUT 132 καθώς 262+80 συνολικά bits αριθμών.



Και τα συναντάμε στους παρακάτω κύκλους:

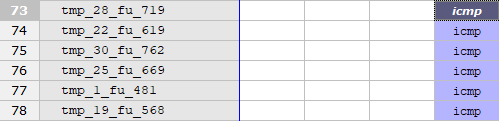
2ο κύκλο:



3ο κύκλο:



4ο κύκλο:



**OR:**

Συνολικός αριθμός: 6

Ο Συνολικός αριθμός LUT 24 καθώς 34 συνολικά bits αριθμών.

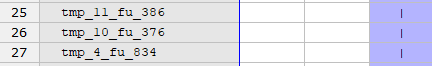


Και τα συναντάμε στους παρακάτω κύκλους:

2ο κύκλο:



3ο κύκλο:



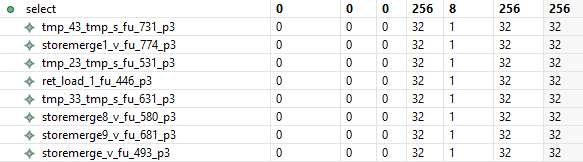
4ο κύκλο:



**Select:**

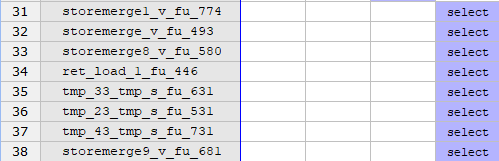
Συνολικός αριθμός: 8

Ο Συνολικός αριθμός LUT 256 καθώς 8+256+256 συνολικά bits αριθμών.



Και τα συναντάμε στους παρακάτω κύκλους:

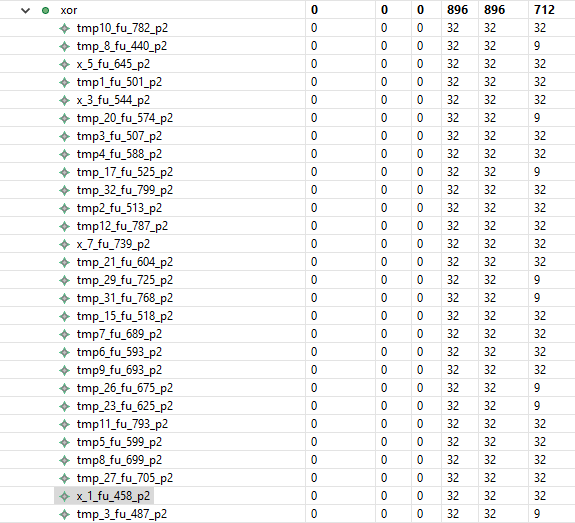
4ο κύκλο:



**XOR:**

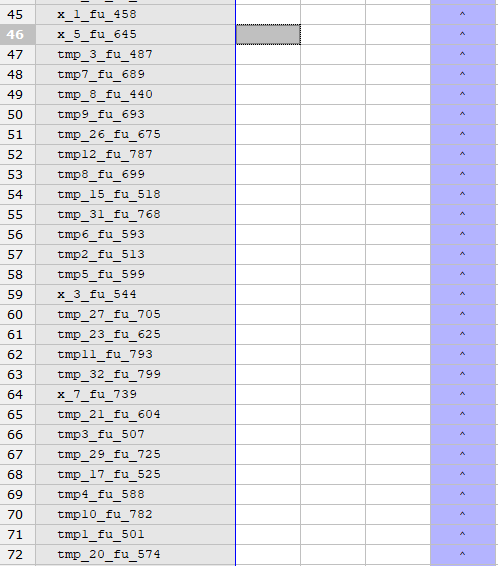
Συνολικός αριθμός: 28

Ο Συνολικός αριθμός LUT 896 καθώς 896+712 συνολικά bits αριθμών.



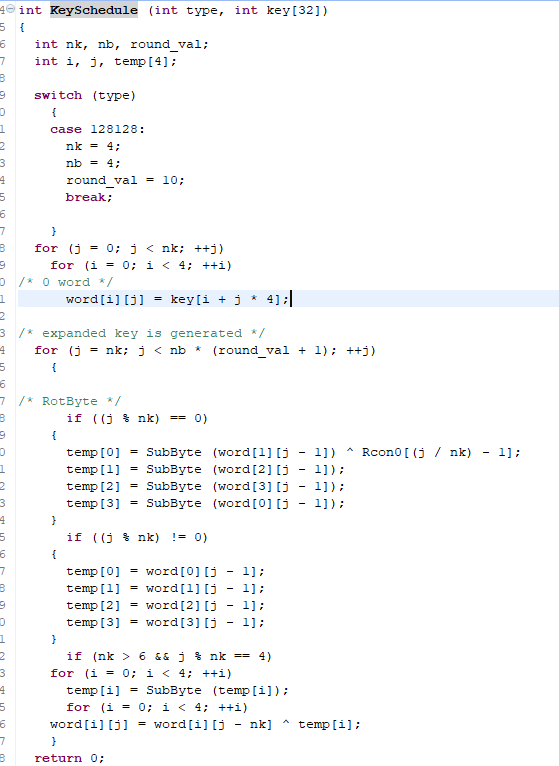
Και τα συναντάμε στους παρακάτω κύκλους:

4ο κύκλο:



**Ανάλυση KeySchedule:**

Κώδικας:

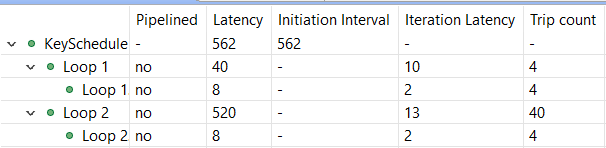


Βλέπουμε ότι αποτελείται από 1 switch, 2 loop τα όποια περιλαμβάνουν εμφωλευμένους βρόγχους.

Στο πρώτο loop βλέπουμε οτι περιλαμβάνει ενα άλλο loop , στο οποίο υπάρχει μια εκχώρηση.

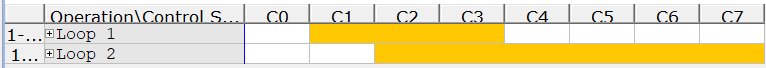
Στο δεύτερο loop βλέπουμε οτι περιλαμβάνει διάφορες συνθήκες στις οποίες αν ισχύσουν οι συνθήκες είτε οχι θα γίνουν καλέσθουν συναρτήσεις είτε απλώς θα έχουμε νέες εκχωρήσεις σε μεταβλητές. Επιπλέον, υπάρχει και ένας εμφωλευμένος βρόγχος στον οποίο γίνεται πράξη xor.

Βλέπουμε παρακάτω τις χρόνους των βρόγχων που επιβραδύνουν το κύκλωμα (καθυστερήσεις).



Λογικό να επιβαρύνει το 2ο loop πιο πολύ το κύκλωμα απο το πλήθος των γραμμών που έχει σε σύγκριση με το 1ο.

Και τώρα θα δούμε τους κύκλους που κάνει το καθέ ενα:



Είναι λογικό να ξεκινήσουν απο το 2ο κύκλο αφου περιμένουν τα δεδομένα απο την είσοδο που αποτελεί argument στην συνάρτηση.

**Resources:**

Απο τι βλέπουμε απο τον πίνακα έχουμε 1 BRAM , 1335 FF και 810 LUT.



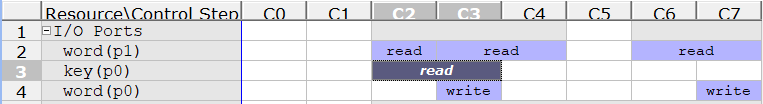
**I/O Ports:**

Συνολικός αριθμός: 2



Καθώς δεσμεύουν 64 bit.

Τα συναντάμε στους κύκλους ρολογιού:

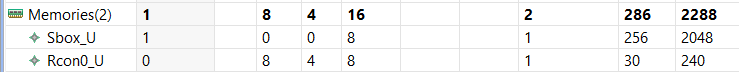


3ο ,4ο, 6ο και 7ο κύκλο.

**Μνήμες:**

Συνολικός αριθμός: 2

Ο Συνολικός αριθμός FF είναι 85 ,4 LUT ,16 συνολικά bits αριθμών καθώς και 1 ΒRAM μεγέθους 2288.



Και τα βρίσκουμε στους κύκλους:



4ο και 5ο κύκλο.

**Expressions:**

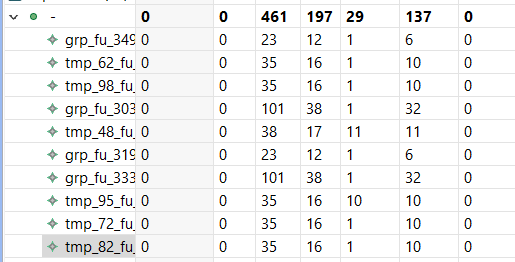
Συνολικός αριθμός: 43

Ο Συνολικός αριθμός FF είναι 909 ,537 LUT καθώς 206+373+64 συνολικά bits αριθμών. 

**Αφαιρέσεις:**

Συνολικός αριθμός: 10

Ο Συνολικός αριθμός FF είναι 461 ,197 LUT καθώς 29+137 συνολικά bits αριθμών.

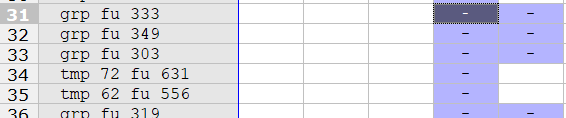


Και τα βρίσκουμε στους κύκλους:

3ο κύκλο:



4ο κύκλο:



5ο κύκλο:



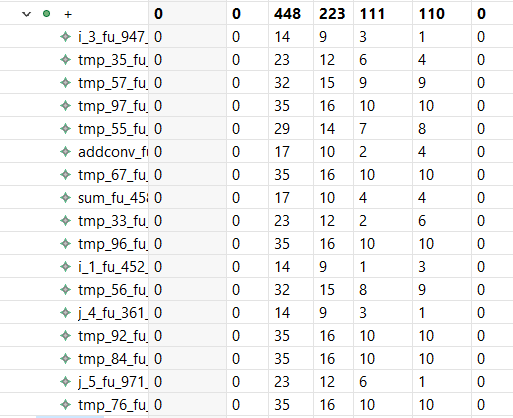
6ο κύκλο:



**Προσθέσεις:**

Συνολικός αριθμός: 18

Ο Συνολικός αριθμός FF είναι 448 ,223 LUT καθώς 111+110 συνολικά bits αριθμών.

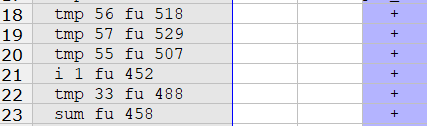


Τα συναντάμε στους κύκλους ρολογιού:

2ο κύκλο:



3ο κύκλο:



4ο κύκλο:



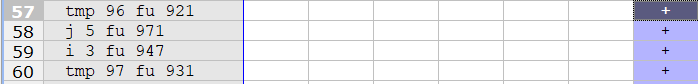
5ο κύκλο:



6ο κύκλο:



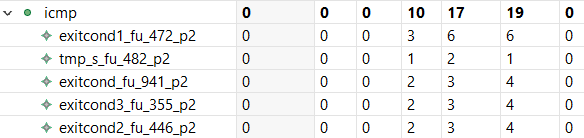
7ο κύκλο:



**Συγκρίσεις:**

Συνολικός αριθμός: 5

Ο Συνολικός αριθμός FF είναι 10 ,17 LUT καθώς 19 συνολικά bits αριθμών.

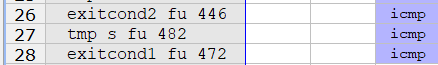


Τα συναντάμε στους κύκλους ρολογιού:

2ο κύκλο:



3ο κύκλο:



7ο κύκλο:



**OR:**

Συνολικός αριθμός: 1

Ο Συνολικός αριθμός LUT 3 καθώς 4 συνολικά bits αριθμών.

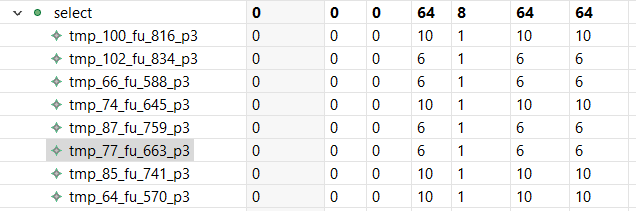


Το συναντάμε στον 3ο κύκλο:



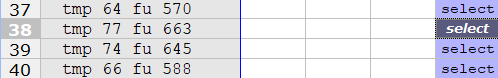
**Select:**

Συνολικός αριθμός: 8

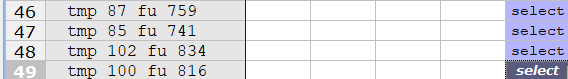
Ο Συνολικός αριθμός LUT είναι 64 καθώς 136 συνολικά bits αριθμών. 

Τα συναντάμε στους κύκλους ρολογιού:

4ο κύκλο:



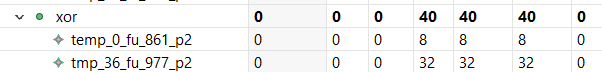
5ο κύκλο:



**XOR:**

Συνολικός αριθμός: 2

Ο Συνολικός αριθμός LUT είναι 40 καθώς 80συνολικά bits αριθμών.



Τα συναντάμε στους κύκλους ρολογιού:

6ο κύκλο:



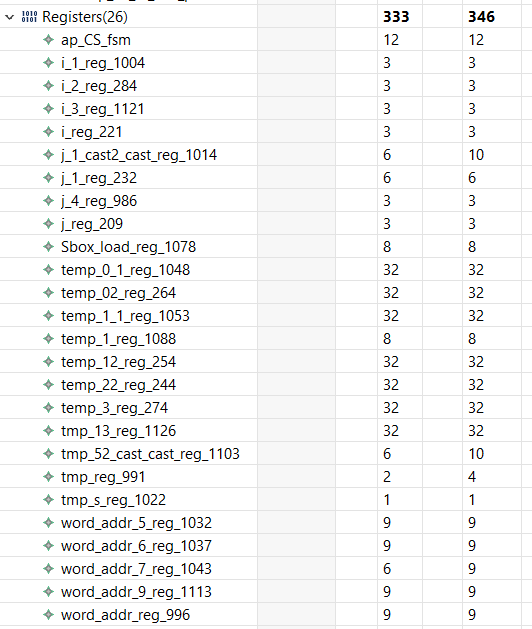
8ο κύκλο:

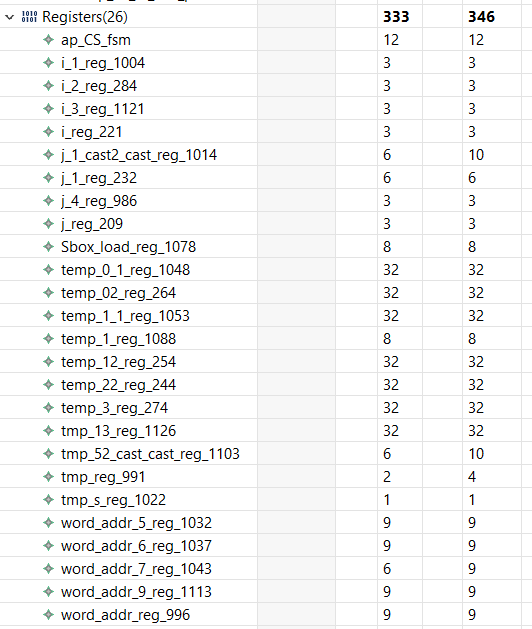


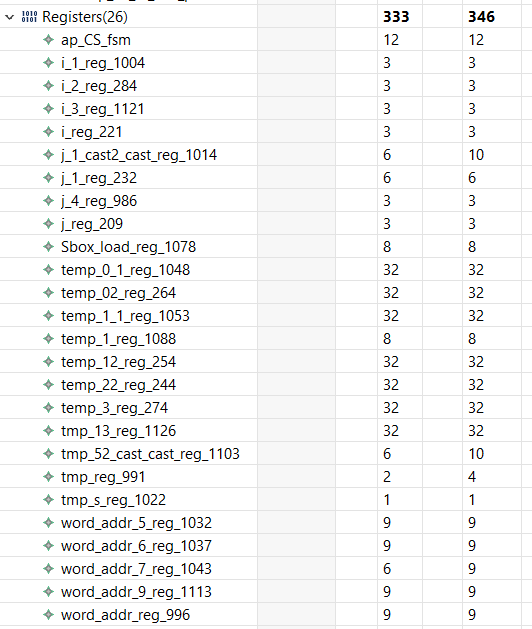
**Καταχωρητές:**

Συνολικός αριθμός: 26

Ο Συνολικός αριθμός FF είναι 333 καθώς 346 συνολικά bits αριθμών.



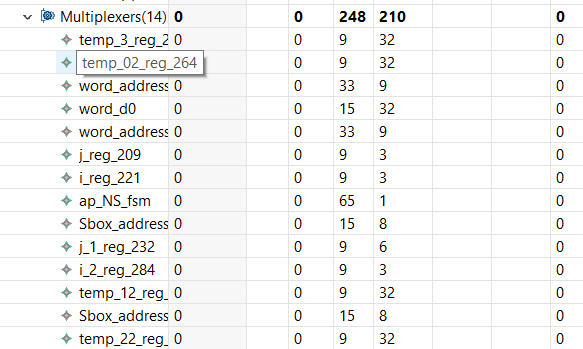




**Πολυπλέκτες:**

Συνολικός αριθμός: 14

Καθώς 248 LUT και 210 Bits.



**INSTANCES:**

Απο τι βλέπουμε έχει υλοποιήσει ένα instance χρησιμοποιούντας 85 FF και 21 LUT.



**Ανάλυση ByteSub\_ShiftRow:**

Κώδικας:

**void** **ByteSub\_ShiftRow** (**int** statemt[32], **int** nb)

{

**int** temp;

**switch** (nb)

{

**case** 4:

temp = Sbox[statemt[1] >> 4][statemt[1] & 0xf];

statemt[1] = Sbox[statemt[5] >> 4][statemt[5] & 0xf];

statemt[5] = Sbox[statemt[9] >> 4][statemt[9] & 0xf];

statemt[9] = Sbox[statemt[13] >> 4][statemt[13] & 0xf];

statemt[13] = temp;

temp = Sbox[statemt[2] >> 4][statemt[2] & 0xf];

statemt[2] = Sbox[statemt[10] >> 4][statemt[10] & 0xf];

statemt[10] = temp;

temp = Sbox[statemt[6] >> 4][statemt[6] & 0xf];

statemt[6] = Sbox[statemt[14] >> 4][statemt[14] & 0xf];

statemt[14] = temp;

temp = Sbox[statemt[3] >> 4][statemt[3] & 0xf];

statemt[3] = Sbox[statemt[15] >> 4][statemt[15] & 0xf];

statemt[15] = Sbox[statemt[11] >> 4][statemt[11] & 0xf];

statemt[11] = Sbox[statemt[7] >> 4][statemt[7] & 0xf];

statemt[7] = temp;

statemt[0] = Sbox[statemt[0] >> 4][statemt[0] & 0xf];

statemt[4] = Sbox[statemt[4] >> 4][statemt[4] & 0xf];

statemt[8] = Sbox[statemt[8] >> 4][statemt[8] & 0xf];

statemt[12] = Sbox[statemt[12] >> 4][statemt[12] & 0xf];

**break**;

**case** 6:

temp = Sbox[statemt[1] >> 4][statemt[1] & 0xf];

statemt[1] = Sbox[statemt[5] >> 4][statemt[5] & 0xf];

statemt[5] = Sbox[statemt[9] >> 4][statemt[9] & 0xf];

statemt[9] = Sbox[statemt[13] >> 4][statemt[13] & 0xf];

statemt[13] = Sbox[statemt[17] >> 4][statemt[17] & 0xf];

statemt[17] = Sbox[statemt[21] >> 4][statemt[21] & 0xf];

statemt[21] = temp;

temp = Sbox[statemt[2] >> 4][statemt[2] & 0xf];

statemt[2] = Sbox[statemt[10] >> 4][statemt[10] & 0xf];

statemt[10] = Sbox[statemt[18] >> 4][statemt[18] & 0xf];

statemt[18] = temp;

temp = Sbox[statemt[6] >> 4][statemt[6] & 0xf];

statemt[6] = Sbox[statemt[14] >> 4][statemt[14] & 0xf];

statemt[14] = Sbox[statemt[22] >> 4][statemt[22] & 0xf];

statemt[22] = temp;

temp = Sbox[statemt[3] >> 4][statemt[3] & 0xf];

statemt[3] = Sbox[statemt[15] >> 4][statemt[15] & 0xf];

statemt[15] = temp;

temp = Sbox[statemt[7] >> 4][statemt[7] & 0xf];

statemt[7] = Sbox[statemt[19] >> 4][statemt[19] & 0xf];

statemt[19] = temp;

temp = Sbox[statemt[11] >> 4][statemt[11] & 0xf];

statemt[11] = Sbox[statemt[23] >> 4][statemt[23] & 0xf];

statemt[23] = temp;

statemt[0] = Sbox[statemt[0] >> 4][statemt[0] & 0xf];

statemt[4] = Sbox[statemt[4] >> 4][statemt[4] & 0xf];

statemt[8] = Sbox[statemt[8] >> 4][statemt[8] & 0xf];

statemt[12] = Sbox[statemt[12] >> 4][statemt[12] & 0xf];

statemt[16] = Sbox[statemt[16] >> 4][statemt[16] & 0xf];

statemt[20] = Sbox[statemt[20] >> 4][statemt[20] & 0xf];

**break**;

**case** 8:

temp = Sbox[statemt[1] >> 4][statemt[1] & 0xf];

statemt[1] = Sbox[statemt[5] >> 4][statemt[5] & 0xf];

statemt[5] = Sbox[statemt[9] >> 4][statemt[9] & 0xf];

statemt[9] = Sbox[statemt[13] >> 4][statemt[13] & 0xf];

statemt[13] = Sbox[statemt[17] >> 4][statemt[17] & 0xf];

statemt[17] = Sbox[statemt[21] >> 4][statemt[21] & 0xf];

statemt[21] = Sbox[statemt[25] >> 4][statemt[25] & 0xf];

statemt[25] = Sbox[statemt[29] >> 4][statemt[29] & 0xf];

statemt[29] = temp;

temp = Sbox[statemt[2] >> 4][statemt[2] & 0xf];

statemt[2] = Sbox[statemt[14] >> 4][statemt[14] & 0xf];

statemt[14] = Sbox[statemt[26] >> 4][statemt[26] & 0xf];

statemt[26] = Sbox[statemt[6] >> 4][statemt[6] & 0xf];

statemt[6] = Sbox[statemt[18] >> 4][statemt[18] & 0xf];

statemt[18] = Sbox[statemt[30] >> 4][statemt[30] & 0xf];

statemt[30] = Sbox[statemt[10] >> 4][statemt[10] & 0xf];

statemt[10] = Sbox[statemt[22] >> 4][statemt[22] & 0xf];

statemt[22] = temp;

temp = Sbox[statemt[3] >> 4][statemt[3] & 0xf];

statemt[3] = Sbox[statemt[19] >> 4][statemt[19] & 0xf];

statemt[19] = temp;

temp = Sbox[statemt[7] >> 4][statemt[7] & 0xf];

statemt[7] = Sbox[statemt[23] >> 4][statemt[23] & 0xf];

statemt[23] = temp;

temp = Sbox[statemt[11] >> 4][statemt[11] & 0xf];

statemt[11] = Sbox[statemt[27] >> 4][statemt[27] & 0xf];

statemt[27] = temp;

temp = Sbox[statemt[15] >> 4][statemt[15] & 0xf];

statemt[15] = Sbox[statemt[31] >> 4][statemt[31] & 0xf];

statemt[31] = temp;

statemt[0] = Sbox[statemt[0] >> 4][statemt[0] & 0xf];

statemt[4] = Sbox[statemt[4] >> 4][statemt[4] & 0xf];

statemt[8] = Sbox[statemt[8] >> 4][statemt[8] & 0xf];

statemt[12] = Sbox[statemt[12] >> 4][statemt[12] & 0xf];

statemt[16] = Sbox[statemt[16] >> 4][statemt[16] & 0xf];

statemt[20] = Sbox[statemt[20] >> 4][statemt[20] & 0xf];

statemt[24] = Sbox[statemt[24] >> 4][statemt[24] & 0xf];

statemt[28] = Sbox[statemt[28] >> 4][statemt[28] & 0xf];

**break**;

}

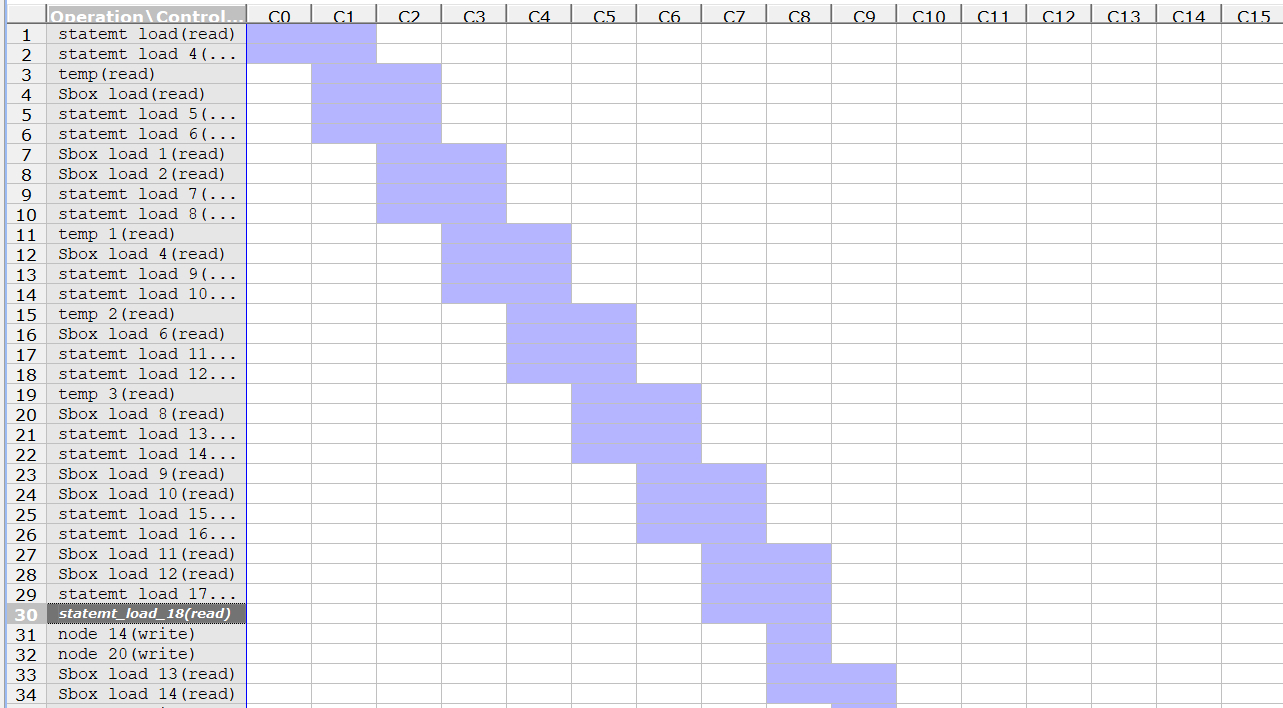
}

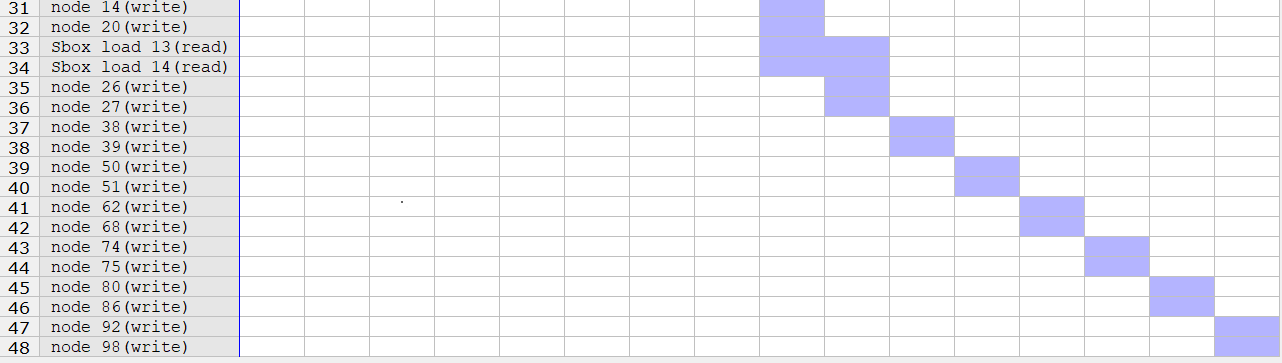
Απο τι βλέπουμε έχει switch καθώς και τις περιπτώσεις που μπορεί να έχει. Επιπλέον, βλέπουμε οτι χρησιμοποιεί την shift left.

Δεν συναντάμε loop, καθώς βλέπουμε οτι καθυστέρηση είναι 15 απο τον πίνακα:



Και σε κύκλους απο τι βλέπουμε τελειώνει όπως είδαμε στο πίνακα δηλαδη στο 15.





**Resources:**

Απο τι βλέπουμε χρησιμοποιεί 1 BRAM , 114 FF καθώς και 457 LUT.



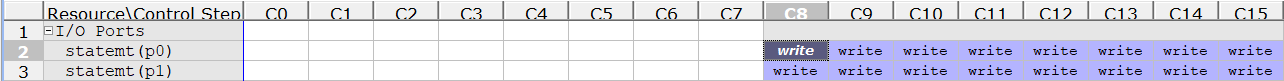
**I/O Ports:**

Συνολικός αριθμός: 1



Καθώς δεσμεύει 32 bit.

Το συναντάμε στον 9ο κύκλο:



**Μνήμες:**

Συνολικός αριθμός: 1

Χρησιμοποιεί 2048 bits για την ΒRAM.



Τα συναντάμε στον 2ο κύκλο ρολογιού.



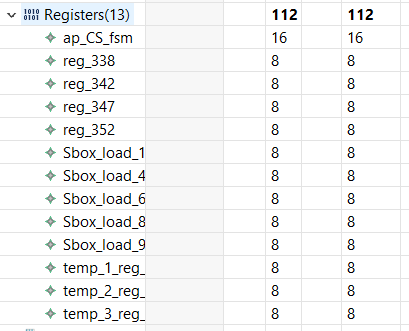
**Expressions:**

To shift left δεν το θεωρεί ως expression, ενώ είναι η μοναδική πράξη που γίνεται στην συνάρτηση.

**Καταχωρητές:**

Συνολικός αριθμός: 13

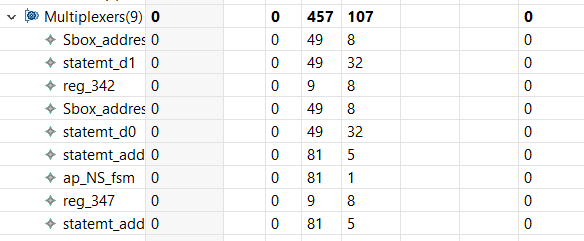
Ο Συνολικός αριθμός FF είναι 112 καθώς 112 συνολικά bits αριθμών.



**Πολυπλέκτες:**

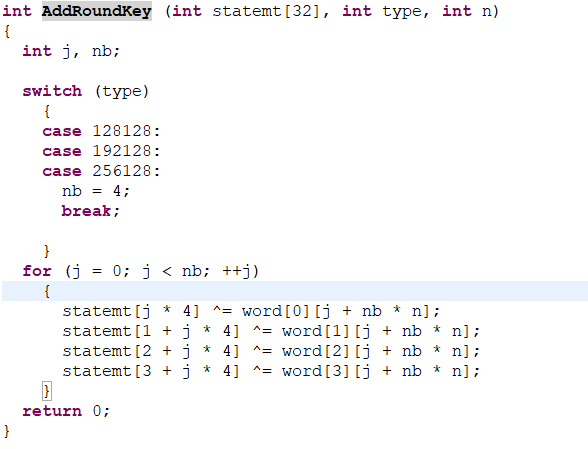
Συνολικός αριθμός: 9

Καθώς χρησιμοποιεί 457 FF και 107 LUT.



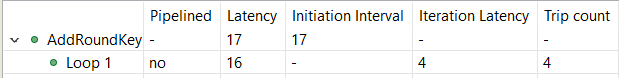
**Ανάλυση ADDRoundKey:**

Κώδικας:

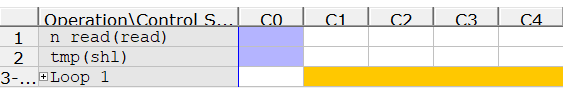


Aπο τον κώδικα βλέπουμε σε αυτη την συνάρτηση, έχουμε ενα switch καθώς και ένα loop. Στο loop βλέπουμε οτι κάνει πράξη xor καθώς προσθέσεις, συγκρίσεις για το loop.

Το latency που προσφέρει η συνάρτηση στο κύκλωμα είναι σύμφωνα με το παρακάτω πίνακα είναι 17 , τα οποία το 16 προέρχονται απο το loop.



Στο άλλο διάγραμμα:



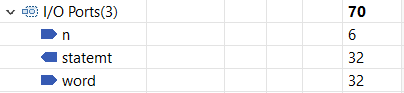
**Resources:**

Απο τι βλέπουμε απο τον πίνακα η συνάρτηση αυτή χρησιμοποιεί 225 FF και 269 LUT.

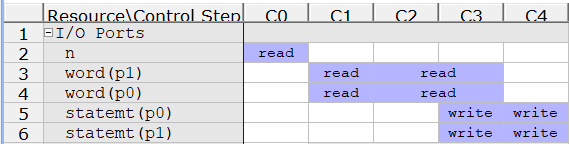


**I/O Ports:**

Συνολικός αριθμός: 3



Καθώς δεσμεύουν 70 bit.

Τα συναντάμε στους κύκλους ρολογιού: 

1ο ,2ο και 4ο κύκλο.

**Μνήμες:**

Δεν χρησιμοποιεί απο τι βλέπουμε.

**Expressions:**

Συνολικός αριθμός: 11

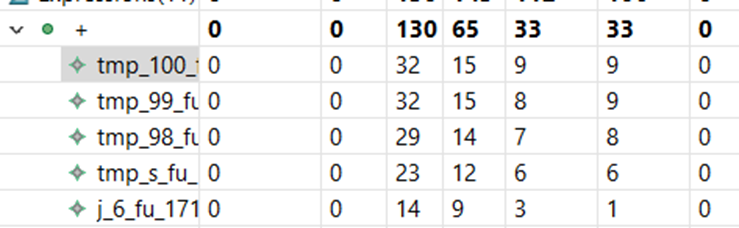
Ο Συνολικός αριθμός FF είναι 130 ,143 LUT καθώς 112+106 συνολικά bits αριθμών.



**Προσθέσεις:**

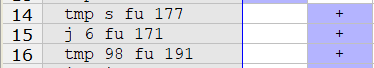
Συνολικός αριθμός: 5

Ο Συνολικός αριθμός FF είναι 130 ,65 LUT καθώς 66 συνολικά bits αριθμών.



Τα συναντάμε στους κύκλους ρολογιού:

2ο κύκλο:



3ο κύκλο:



**Συγκρίσεις:**

Συνολικός αριθμός: 1

Ο Συνολικός αριθμός FF είναι 2 ,3 LUT καθώς 4συνολικά bits αριθμών.



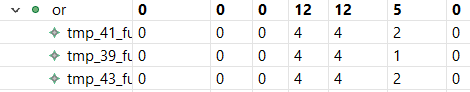
Το συναντάμε στο 2ο κύκλο:



**OR:**

Συνολικός αριθμός: 3

Ο Συνολικός αριθμός LUT είναι 12 καθώς 17 συνολικά bits αριθμών.



Τα συναντάμε στους κύκλους ρολογιού:

2ο κύκλο:



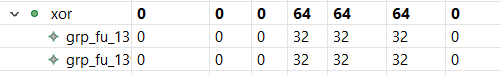
3ο κύκλο:



**XOR:**

Συνολικός αριθμός: 2

Ο Συνολικός αριθμός LUT είναι 64 καθώς 128 συνολικά bits αριθμών.



Τα συναντάμε στους κύκλους ρολογιού:

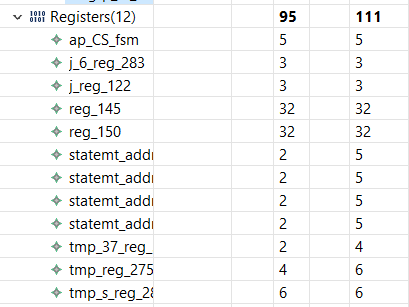
2ο κύκλο:



**Καταχωρητές:**

Συνολικός αριθμός: 12

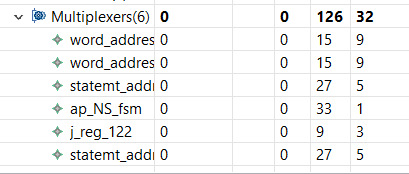
Ο Συνολικός αριθμός FF είναι 95 καθώς 111 συνολικά bits αριθμών.



**Πολυπλέκτες:**

Συνολικός αριθμός: 6

Καθώς χρησιμοποιεί 126 LUT και 32 Bits.



**Συνοπτικά στατιστικά κυκλωμάτων:**

Resources:

|  |  |  |  |
| --- | --- | --- | --- |
| Κύκλωμα | 1ο κύκλωμα | 2ο κύκλωμα | 3ο κύκλωμα |
| I/O Ports | 2 | 3 | 3 |
| Instances | 0 | 3 | 4 |
| Memories | 1 | 6 | 1 |
| Expressions | 109 | 41 | 4 |
| Registers | 46 | 43 | 10 |
| Channels | 0 | 0 | 0 |
| Multiplexers | 37 | 27 | 17 |

|  |  |  |  |
| --- | --- | --- | --- |
| Κύκλωμα | 1ο κύκλωμα | 2ο κύκλωμα | 3ο κύκλωμα |
| BRAM | 32 | 6 | 6 |
| FF | 2612 | 2728 | 2137 |
| LUT | 2590 | 3055 | 3550 |

Αποδόσεις κυκλωμάτων Latency:

|  |  |  |  |
| --- | --- | --- | --- |
| Κύκλωμα | 1ο κύκλωμα | 2ο κύκλωμα | 3ο κύκλωμα |
| Latency | 51 | 3766 | 1092 |

Κύκλωμα 1:

Αποδόσεις για loop:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Latency | Iteration Latency | Trip Count |
| Loop 1 | 45 | 5 | 9 |
| Συνολικό | 51 | - | - |

Κύκλωμα 2:

Αποδόσεις για loop:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Latency | Iteration Latency | Trip Count |
| Loop 1 | - | 2 | - |
| Loop 2 | 40 | 10 | 4 |
| Loop 3 | 0~279172873980 | 130 | 0~2147483646 |
| Loop 4 | 40 | 10 | 4 |
| Συνολικό | - | - | - |

Κύκλωμα 3:

Αποδόσεις για loop:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Latency | Iteration Latency | Trip Count |
| Loop 1 | 441 | 49 | 9 |
| Loop 2 | 32 | 2 | 16 |
| Συνολικό | 1092 | - | - |

Το συνολικό Latency αποτελεί το συνολικό Latency του κυκλώματος και όχι της πρόσθεσης των latency των loop.

**Συμπεράσματα:**

Μια πολυ σημαντική διαφορά των κυκλωμάτων είναι οτι το 1ο κύκλωμα σε σύγκριση με τα άλλα δεν περιέχει instances. Αυτό επιφέρει τεράστια αλλαγή όσον αφορά την απόδοση του κυκλώματος,

Επιπλέον ο αριθμός των βρόγχων αποτελεί πάρα πολυ σημαντικός παράγοντας για την απόδοση του κυκλώματος, ειδικά αν εμπεριέχει και εμφωλευμένα. Να υπενθυμίσουμε οτι κάθε loop , πρέπει να κάνει πράξεις σύγκρισης και πρόσθεσης και οτι κάθε εξωτερικό πρέπει να περιμένει τα τελείωσει το εσωτερικό δηλαδή το εμφωλευμένο.

Παρ όλου που 1ο κύκλωμα χρησιμοποιεί πολλά resources σε σύγκριση με το 3ο και περίπου με το δεύτερο, αυτό που υστερεί είναι τα instances. Βλέπουμε ότι το 1ο δεσμεύει αρκετό χώρο στο κύκλωμα όσον αφορά τις μνήμες, Flip-Flop , Look Up Tables αλλά η στέρηση άλλων function αποτελεί ο κυριότερος παράγοντας για την απόδοση του. Φυσικά και ο τρόπος με τον οποίο έχει φτιαχτεί αποτελεί σημαντικός παράγοντας.

Από τους κώδικες που έχουμε ο πιο μη φιλικός προς τον χρήστη αποτελεί ο 1ος  καθώς έχει φτιαχτεί με τέτοιο τρόπο ώστε να έχει καλή απόδοση μετά την σύνθεση ενώ οι άλλοι είναι πιο κατανοητοί αν και χάνουν στην απόδοση.