



ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ  
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ  
ΥΠΟΛΟΓΙΣΤΩΝ

[HRY 419]-Ανάπτυξη Εργαλείων CAD για  
Σχεδίαση Ολοκληρωμένων Κυκλωμάτων  
Αναφορά 3<sup>ου</sup> εργαστηρίου

Ιωάννης Περίδης

A.M. 2018030069

13 Απριλίου 2021

*ΠΡΟΣΟΧΗ: Η εργασία αυτή είχε υποβληθεί κανονικά έγκαιρα πριν την λήξη της προθεσμίας. Απλά κάνοντας το 4<sup>ο</sup> εργαστήριο κατάλαβα ότι είχα ξεχάσει έναν counter να αυξάνει σε έναν βρόγχο εκεί που δεν έπρεπε και επηρέαζε ελαφρώς τα αποτελέσματα. Άλλαξα μονάχα 1 γραμμή κώδικα (έσβησα την αύξηση του counter) και ξαναέστειλα μαζί με την 4<sup>η</sup> και την 3<sup>η</sup> εργασία με αυτήν την μικρή διόρθωση. Μου είχατε πεί να σας το γράψω εδώ για να το ξέρετε για την διόρθωση ότι ήταν έγκυρη η αποστολή της εργασίας μου. Δεν άλλαξε κάτι άλλο στην εργασία.*

## 1 Εισαγωγή:

Σκοπός της εργαστηριακής άσκησης αυτής, είναι να σπάσουμε ένα μεγάλο πρόβλημα σε υπο προβλήματα, κάνοντας μια αρχή για την ολοκληρωμένη διαδικασία place & route (P&R). Συγκεκριμένα, μας ζητήθηκε να γίνει routing και compaction σε μια σειρά από standard shells, με την χρήση κάποιου σχετικά απλοϊκού αλγόριθμου ο οποίος εξισορροπεί την αποτελεσματικότητα της λύσης σε σχέση με την πολυπλοκότητα της. Φυσικά αυτό έγινε χρησιμοποιώντας τις παραδοχές της εκφώνησης με σκοπό την απλοποίηση του προβλήματος σε έναν μικρό βαθμό.

## 2 Διαχείριση netlist εισόδου:

Ξεκινώντας, το πρόγραμμα δέχεται σαν είσοδο ένα netlist με μια πολύ συγκεκριμένη μορφή και με την παραδοχή πως δεν θα υπάρξουν λάθη σε αυτό. Το netlist εισόδου, χωρίζεται σε 3 τομείς το number of gates, placement και routing.

Αρχικά, με την χρήση της συνάρτησης fseek μετακινούμε τον κέρσορα στο αρχείο στο σημείο που θέλουμε για να διαβάσουμε με την fscanf τον αριθμό πυλών σαν έναν ακέραιο και να τον αποθηκεύσουμε σε μια μεταβλητή numOfShells.

Ύστερα, δημιουργούμε το array από ακεραίους placement[] που έχει μήκος το numOfShells. Ξανά με την βοήθεια της συνάρτησης fseek (και αλλάζοντας σειρά στο αρχείο με την fgets, θα δείτε στον κώδικα αν θέλετε πως), πηγαίνουμε τον κέρσορα στο σημείο που θέλουμε και από εκεί υλοποιούμε έναν βρόγχο επανάληψης για να διαβάσουμε όλους τους αριθμούς πυλών. Αυτό γίνεται, χρησιμοποιώντας την συνάρτηση fgets για να διαβαστεί ο αριθμός πύλης σαν string 2 χαρακτήρων και ύστερα μέσω της συνάρτησης atoi, μετατρέπεται σε ακέραιος και εισάγεται στο array placement[]. Εδώ θα πρέπει να αναφερθεί πως από όσο γίνεται κατανοητό το placement[], πρακτικά μας δείχνει την σειρά με την οποία θα έχω τα standard shells (δηλαδή τις πύλες) στο κύκλωμά μου.

Τέλος, δημιουργούμε έναν πίνακα routng[][] με 3 στήλες και numOfShells γραμμές από ακεραίους. Με τον ίδιο τρόπο με πριν γεμίζω την 1<sup>η</sup> στήλη του πίνακα με τον αριθμό της πύλης, την 2<sup>η</sup> στήλη του πίνακα με την 1<sup>η</sup> είσοδο της πύλης και την 3<sup>η</sup> στήλη με την 3<sup>η</sup>

είσοδο. Στο σημείο αυτό πρέπει να αναφερθεί πως η σειρά των πυλών τώρα, είναι κατά αύξοντα αριθμό και όχι στοιχισμένη έτσι όπως το placement. Ακόμη, με σκοπό να διαφοροποιηθούν οι εξωτερικές εισοδοί του κυκλώματος από τις εσωτερικές, έγινε το εξής. Για εξωτερική είσοδο μορφής INXX, ο αριθμός που αποθηκεύεται στον πίνακα είναι XX+100 (αυτό διότι υπάρχουν μέχρι 99 shells ,άρα σίγουρα δεν θα γίνει κάποιο μπερδεμα με άλλη εσωτερική είσοδο), ενώ για μορφής AXX, ο αριθμός που αποθηκεύεται είναι XX+200 και τέλος για BXX, αποθηκεύεται XX+300 με σκοπό οι εισοδοί A,B και IN να ξεχωρίζουν και μεταξύ τους.

### 3 Πίνακας απεικόνισης κυκλώματος:

Για την απεικόνιση και εκτύπωση του κυκλώματος σε μια ολοκληρωμένη μορφή, η διαδικασία έγινε βήμα βήμα , σπάζοντας το πρόβλημα σε υπο προβλήματα και έπειτα συνενώνοντας τις λύσεις τους.

Αρχικά, συνδυάζοντας τις τιμές του πίνακα routing[][] και την σειρά όπου θα έχω τα std shells στο κύκλωμα, δηλαδή το array placement[], δημιουργήθηκε ο πίνακας inputs[][] με τον αριθμό πύλης και τις αντίστοιχες εισόδους στην σωστή σειρά αυτή τη φορά.

Ύστερα, υλοποιήθηκε ο πίνακας outpus[][] των εξόδων, ο οποίος δεν φαίνεται στο netlist καθώς και μια έξοδος μπορεί να πηγαίνει σε πολλές διαφορετικές εισόδους. Η δημιουργία του πίνακα αυτού, ήταν λίγο πιο σύνθετη , διότι ενώ οι γραμμές παραμένουν σταθερές και ίσες με numOfShells. Οι στήλες του θα έχουν μέγιστο μήκος NumOfShells επίσης, αλλά θα είναι διάφορες του 0 μόνο όσες εξοδοί πηγαίνουν σε κάποια είσοδο (θα γίνει πιο κατανοητό με την εικόνα παρακάτω). Στο σημείο αυτό, πρέπει να αναφερθεί ότι κρατιέται στην μεταβλητή maxOuts, ο αριθμός εξόδων της πύλης που κατευθύνει τις περισσότερες εξόδους.

Ενώνοντας τους δύο παραπάνω πίνακες , δημιουργήθηκε ο πίνακας total[][] ο οποίος έχει numOfShells γραμμές και 3+maxOuts στήλες. Αυτό διότι οι πρώτες 3 στήλες έχουν τον αριθμό πύλης και τις εισόδους, ενώ οι επόμενες στήλες έχουν τον αριθμό των πυλών που παίρνουν την έξοδο της.

**Παρακάτω φαίνεται η απεικόνιση του κυκλώματος για το netlist της εκφώνησης:**

**Netlist:**

```
** NUMBER OF GATES
6
** PLACEMENT
U21 GATE_2
U53 GATE_2
U06 GATE_2
U29 GATE_2
U44 GATE_2
U78 GATE_2
** ROUTING
U06 IN01 U44
U21 U29 IN35
U29 U53 U06
U44 U21 U78
U53 U44 IN02
U78 U53 U06
```

**Έξοδος προγράμματος:**

GATES	IN1	IN2	OUTS	...
21	29	135	44	0
53	44	102	29	78
6	101	44	29	78
29	53	6	21	0
44	21	78	53	6
78	53	6	44	0

### 4 Preprocessing:

Συνεχίζοντας, πριν προχωρήσουμε στην διαδικασία του routing πρέπει να γίνει κάποια προ επεξεργασία του κυκλώματος με σκοπό το routing και μεταγενέστερα ακόμα και το

placement να δώσουν καλύτερα ή γρηγορότερα αποτελέσματα.

Η ιδέα είναι πως θέλουμε να τοποθετήσουμε τα σύρματα με το μεγαλύτερο μήκος στις υψηλότερες γραμμές στο routing , ενώ τα σύρματα με το μικρότερο μήκος σε μεταγενέστερες γραμμές πιο κάτω. Αυτό διότι έτσι θα γίνει πιο εύκολα το compaction και επίσης όσο προχωράμε γραμμές δεν θα γίνονται τόσο μεγάλες δραστικές αλλαγές.

Η υλοποίηση αυτού, έγινε μέσω ενός πολύπλοκου βρόγχου που ήταν υπεύθυνος να δημιουργήσει έναν πίνακα length[] με το μήκος κάθε σύρματος (δηλαδή κάθε μιας ένωσης μεταξύ μιας εισόδου και μιας εξόδου) , χρησιμοποιώντας τα στοιχεία του παραπάνω πίνακα total[][]. Το μήκος φυσικά, δεν είναι το πραγματικό μήκος των καλωδίων, αλλά η απόσταση μεταξύ της εισόδου και τις εξόδου ,μετρημένη σε standard shells.

Στην συνέχεια με την χρήση του πίνακα length[] και ενός βρόγχου που βρίσκει το μέγιστο μήκος του array length, δημιουργήθηκε με πολύ απλό τρόπο (maxLength-length+1) ο πίνακας order[] που μας δείχνει με ποια σειρά πρέπει να τοποθετηθούν τα σύρματα στο routing ( οι αριθμοί του order δεν είναι σειριακοί, απλά εκτελούνται κατά αύξουσα σειρά). Προφανώς δίνει προτεραιότητα στα μεγαλύτερα και μετά στα μικρότερα.

Τέλος, με την χρήση των στοιχείων του πίνακα total[][], δημιουργήθηκε ένας πίνακας ακεραίων totalConnections[][] , ο οποίος έχει γραμμές ίσες με numOfShells\*maxOuts και 2 στήλες , που συμβολίζουν τα 2 στοιχεία που ενώνονται, μια είσοδο με μία έξοδο.

Οι 3 πίνακες αυτοί, εκτυπώθηκαν με σκοπό να φαίνεται για κάθε σύνδεση , δηλαδή για κάθε σύρμα το μήκος του και η σειρά που θα τοποθετηθεί.

***Παρακάτω φαίνεται η έξοδος PREPROCESSING του προγράμματος για το netlist της εκφώνησης:***

```
*****PREPROCESSING*****
=====
(U44_IN,U21_OUT) | wire length:4 | order of placement:1 |
(U29_IN,U53_OUT) | wire length:2 | order of placement:3 |
(U78_IN,U53_OUT) | wire length:4 | order of placement:1 |
(U29_IN,U06_OUT) | wire length:1 | order of placement:4 |
(U78_IN,U06_OUT) | wire length:3 | order of placement:2 |
(U21_IN,U29_OUT) | wire length:3 | order of placement:2 |
(U53_IN,U44_OUT) | wire length:3 | order of placement:2 |
(U06_IN,U44_OUT) | wire length:2 | order of placement:3 |
(U44_IN,U78_OUT) | wire length:1 | order of placement:4 |
```

## 5 Routing:

Πλέον, είμαστε έτοιμοι να προχωρήσουμε στο routing που δεν είναι άλλο από το να τοποθετήσουμε στην σωστή σειρά τα σύρματα μας ένα σε κάθε μια γραμμή (για αρχή).

Το παραπάνω, γίνεται ταξινομώντας τον πίνακα των συρμάτων totalConnections[][] , μέσω της συνάρτησης sawpRows() η οποία ανταλλάσσει 2 από τις γραμμές σε έναν πίνακα Nx2. Η ταξινόμηση γίνεται με κριτήριο την σειρά προτεραιότητας τοποθέτησης , δηλαδή μέσω του πίνακα order[]. Εννοείται πως όταν ανταλλάσσουμε μια σειρά στον πίνακα totalConnections, τα αντίστοιχα στοιχεία ανταλλάσσουμε από τα arrays order και length, έτσι ώστε τα μήκη και η σειρά των συρμάτων να παραμείνει ίδια στο καθένα και να μην μπερδευτούνε.

***Παρακάτω φαίνεται η έξοδος ROUTING του προγράμματος για το netlist της εκφώνησης:***

```

*****ROUTING*****
=====
(U44_IN,U21_OUT) | wire length:4 | order of placement:1 |
(U78_IN,U53_OUT) | wire length:4 | order of placement:1 |
(U78_IN,U06_OUT) | wire length:3 | order of placement:2 |
(U21_IN,U29_OUT) | wire length:3 | order of placement:2 |
(U53_IN,U44_OUT) | wire length:3 | order of placement:2 |
(U29_IN,U53_OUT) | wire length:2 | order of placement:3 |
(U06_IN,U44_OUT) | wire length:2 | order of placement:3 |
(U29_IN,U06_OUT) | wire length:1 | order of placement:4 |
(U44_IN,U78_OUT) | wire length:1 | order of placement:4 |

```

Έπειτα, εκτυπώνεται ο αριθμός γραμμών που περιέχει η λύση πριν την διαδικασία του compaction.

*Παρακάτω φαίνεται η έξοδος ROWS IN SOLUTION του προγράμματος για το netlist της εκφώνησης:*

```

*****ROWS IN SOLUTION*****
=====
Number of rows in the solution before compaction:9

```

## 6 Compaction:

Μετά το routing ακολουθεί ο αλγόριθμος του compaction ο οποίος έχει σκοπό να βελτιώσει την λύση μας ,δηλαδή να μειωθεί ο αριθμός των γραμμών στο τελικό κύκλωμα μέσα σε ένα λογικό χρόνο και με μια λογική πολυπλοκότητα.

Η ιδέα πίσω από τον αλγόριθμο, χωρίζεται σε 2 βήματα.

### *Βήμα πρώτο:*

Βρίσκουμε τις εξόδους οι οποίες πηγαίνουν σε παραπάνω από 1 εισόδους και πρακτικά, απλά χρειάζεται να επεκτείνουμε ένα σύρμα από την έξοδο το οποίο να ενώνεται κατευθείαν με όλες τις εισόδους. Με αυτόν τον τρόπο, γλιτώνουμε για κάθε μια έξοδο που κατευθύνεται σε N εισόδους, N-1 περιττά σύρματα που θα συνδέονταν με την ίδια έξοδο σε διαφορετική γραμμή.

Το παραπάνω υλοποιείται αρκετά εύκολα με έναν βρόγχο ο οποίος περνώντας τον πίνακα totalConnections όταν βρίσκει το παραπάνω σενάριο αφαιρεί τον κατάλληλο αριθμό γραμμών από την τελική λύση και μας λέει ποιες συγκεκριμένες έξοδοι κατευθύνονται σε παραπάνω από 1 εισόδους. Στην περίπτωση μας, όπως θα δείτε και παρακάτω, υπάρχουν 3 έξοδοι οι οποίες κατευθύνονται σε 2 διαφορετικές εισόδους. Είναι φανερό άρα, πως μέσω του βήματος αυτού οι γραμμές στην τελική μας λύση θα μειωθούν από 9 στο 6.

*Παρακάτω φαίνεται η έξοδος COMPACTION FIRST STEP του προγράμματος για το netlist της εκφώνησης:*

```

*****COMPACTION FIRST STEP*****
=====
wire (U78_IN,U53_OUT) is extended. Because U53_OUT goes to more than 1 inputs
wire (U78_IN,U6_OUT) is extended. Because U6_OUT goes to more than 1 inputs
wire (U53_IN,U44_OUT) is extended. Because U44_OUT goes to more than 1 inputs

```

### *Βήμα δεύτερο:*

Το δεύτερο βήμα , είναι λιγάκι πιο πολύπλοκο. Έχουμε ήδη βρει για κάθε σύρμα το μήκος του. Τώρα θέλουμε να βρούμε συγκεκριμένα το standard shell που αρχίζει το σύρμα , όπως

και το shell στο οποίο τελειώνει το σύρμα, με σκοπό να βρούμε μεταξύ ποιων κελιών είναι τοποθετημένα στο κύκλωμα μας. Οι θέσεις αυτές αποθηκεύονται στις μεταβλητές start και end. Στην συνέχεια , αφού γνωρίζουμε το μέγιστο μήκος των κελιών μας, είναι πολύ απλό να βρούμε για κάθε ένα σύρμα πόσο κενό χώρο έχει από τα δεξιά του και πόσο κενό χώρο από τα αριστερά του. Οι τιμές αυτές αποθηκεύονται στα arrays spaceRigth[] και spaceLeft[].

Στο σημείο αυτό, έχουμε ότι χρειαζόμαστε για να επιτευχθεί η βασική λειτουργία του αλγόριθμου. Για αρχή για κάθε γραμμή ελέγχουμε αν υπάρχει αρκετός κενός χώρος στα δεξιά ή στα αριστερά στην γραμμή από πάνω έτσι ώστε να τοποθετηθεί το τωρινό σύρμα. Δηλαδή αν το length του τωρινού < spaceLeft ή spaceRigth του προηγούμενου. Φυσικά δεν αρκεί μονάχα αυτή η συνθήκη για να χωρέσει το σύρμα να μπει στην ίδια γραμμή με το προηγούμενο. Πρέπει να γίνει έλεγχος των άκρων τους και να είναι συμβατά. Συγκεκριμένα, αν θέλω να τοποθετήσω το νέο σύρμα αριστερά του προηγούμενου πρέπει το τέλος του τωρινού μου σύρματος να είναι στο ίδιο shell ή σε προηγούμενα shells από αυτό που βρίσκεται η αρχή του προηγούμενου σύρματος. Αντίστοιχα λειτουργεί και για αριστερά απλά ελέγγω η αρχή του τωρινού να είναι στο ίδιο ή σε επόμενα shells από το τέλος του προηγούμενου.

Για κάθε σύρμα που ισχύουν οι 2 συνθήκες, γλιτώνουμε από την τελική λύση, μια γραμμή και εκτυπώνεται στην οθόνη, συγκεκριμένα ποιο σύρμα θα μετακινηθεί και αν θα πάει αριστερά ή δεξιά του προηγούμενου από αυτό σύρματος. Στην περίπτωση μας, το τελικό σύρμα του routing , χωράει δεξιά από το προηγούμενο έτσι από 6 γραμμές στην λύση που είχαμε ,μετά το 1° βήμα τώρα έχουμε 5.

### ***Προσοχή λεπτά σημεία στο βήμα 2:***

Γίνεται ένας έλεγχος πριν από κάθε γραμμή. Αν η γραμμή ανήκει στην ομάδα των συρμάτων τα οποία επεκτάθηκαν στο βήμα 1, τότε ούτε προσπαθούμε να χωρέσουμε αυτήν την γραμμή στην προηγούμενη, ούτε επίσης προσπαθούμε να χωρέσουμε σε αυτήν την γραμμή κάτι από την επόμενη. Αυτό διότι, εφόσον έγινε η συνένωση με τα σύρματα που έπαιρναν την ίδια έξοδο, προφανώς θα άλλαζε το μήκος της γραμμής θα μεγάλωνε και δεν έχει υπολογιστεί το νέο αυτό μήκος ,ούτε οι νέες θέσεις αρχής και τέλους γιατί θα αυξανόταν πολύ η πολυπλοκότητα του προγράμματος. Οπότε εφόσον έτσι και αλλιώς θα είναι ένα αρκετά μεγάλο μήκος (αφού έχει πολλές συνδέσεις) και η πιθανότητα να χωρούσε άλλο σύρμα ή το ίδιο να χωρούσε κάπου αλλού ήταν μικρή , απλά το παραλείπουμε και το αφήνουμε όπως έχει για να γλιτώσουμε από πολυπλοκότητα.

Ακόμη, είναι φανερό πως αν γίνει επιτυχώς μια μετακίνηση σύρματος στην πάνω σειρά, αφού υπάρχει αρχικά μόνο ένα σύρμα σε κάθε γραμμή, τότε θα μείνει η γραμμή στην οποία βρισκόταν άδεια. Άρα στα σίγουρα το επόμενο σύρμα θα χωρέσει να μετακινηθεί επίσης.

### ***Παρακάτω φαίνεται η έξοδος COMPACTION SECOND STEP του προγράμματος για το netlist της εκφώνησης:***

```
*****COMPACTION SECOND STEP*****
=====
start of wire: 1| end of wire: 5| wire length: 4| empty space left: 0| empty space right: 1|
start of wire: 2| end of wire: 6| wire length: 4| empty space left: 1| empty space right: 0|
start of wire: 3| end of wire: 6| wire length: 3| empty space left: 2| empty space right: 0|
start of wire: 1| end of wire: 4| wire length: 3| empty space left: 0| empty space right: 2|
start of wire: 2| end of wire: 5| wire length: 3| empty space left: 1| empty space right: 1|
start of wire: 2| end of wire: 4| wire length: 2| empty space left: 1| empty space right: 2|
start of wire: 3| end of wire: 5| wire length: 2| empty space left: 2| empty space right: 1|
start of wire: 3| end of wire: 4| wire length: 1| empty space left: 2| empty space right: 2|
start of wire: 5| end of wire: 6| wire length: 1| empty space left: 4| empty space right: 0|

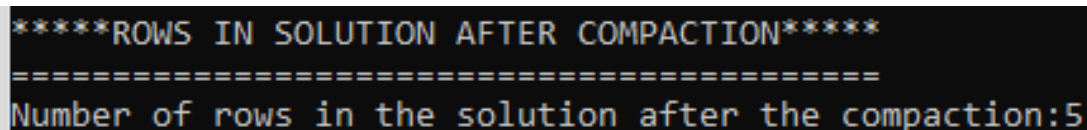
**wire (U44_IN,U78_OUT) is moved 1 row up, at the right of wire (U29_IN,U6_OUT)**
```



Τελικά, εκτυπώνω τον νέο μειωμένο αριθμό γραμμών της λύσης μου μετά τον αλγόριθμο του compaction.

*Παρακάτω φαίνεται η έξοδος ROWS IN SOLUTION AFTER COMPACTION του προγράμματος για το netlist της εκφώνησης:*

```
*****ROWS IN SOLUTION AFTER COMPACTION*****  
=====
```



```
Number of rows in the solution after the compaction:5
```

## 7 Αποτελέσματα & Netlist εξόδου:

Συμπερασματικά, γίνεται κατανοητό πως ο αλγόριθμος μας για routing του κυκλώματος, δεν μπορεί να είναι σε όλες τις καταστάσεις βελτίστως και ταυτόχρονα να έχει μικρή χρονική πολυπλοκότητα. Εμείς πρέπει να βρούμε μια ισορροπία μεταξύ των 2 αυτών που να δίνει ένα αρκετά καλό αποτέλεσμα σε έναν αρκετά μικρό χρόνο.

Συγκεκριμένα, στην περίπτωση μας θα έδινε αρκετά καλύτερες λύσεις, δηλαδή με πολύ λιγότερες γραμμές, αν προσπαθούσαμε στο βήμα 2 να χωρέσουμε το νέο σύρμα σε οποιοδήποτε κενό υπάρχει από τις παραπάνω σειρές και όχι μόνο στην ακριβώς παραπάνω. Γεγονός που αποφεύχθηκε για να έχουμε λιγότερο CPU time, καθώς και δεν θα είχε τόσο μεγάλη σημασία στο τελικό αποτέλεσμα μετά το placement.

Τέλος εκτός από όλες τις παραπάνω γραφικές εξόδους που δίνει το πρόγραμμά μου, δημιουργεί και ένα netlist εξόδου που περιέχει 3 τομείς. Το gates in the circuit, rows in the solution και routing.

*Παρακάτω φαίνεται το netlist εξόδου του προγράμματος για το netlist της εκφώνησης:*

```
**GATES IN THE CIRCUIT  
U21 U53 U06 U29 U44 U78  
**ROWS IN THE SOLUTION  
5  
**ROUTING  
(U44_IN,U21_OUT)  
(U78_IN,U53_OUT)  
(U78_IN,U06_OUT)  
(U21_IN,U29_OUT)  
(U53_IN,U44_OUT)  
(U29_IN,U53_OUT)  
(U06_IN,U44_OUT)  
(U29_IN,U06_OUT)  
(U44_IN,U78_OUT)
```

*Παρακάτω φαίνονται όλα τα αποτελέσματα από τα 7 δοκιμαστικά netlists που μας δόθηκαν:*

Στο σημείο αυτό, πρέπει να αναφερθεί πως όλα τα netlists δούλεψαν σωστά και παράχθηκε λύση routing για το κάθε ένα.

Παρακάτω φαίνεται ο πίνακας με τα αποτελέσματα της κάθε λύσης. Αρχικά φαίνεται το πλήθος των συρμάτων πριν το compaction, δηλαδή πρακτικά το πλήθος των συνδέσεων που

υπάρχουν. Έπειτα φαίνεται το πλήθος των γραμμών που αφαιρείται σε κάθε ένα από τα βήματα του compaction.

Παρατηρείται από τα αποτελέσματα πως ο αλγόριθμος συμπεριφέρεται διαφορετικά, αναλόγως με την σειρά που έχουν τοποθετηθεί τα standard shells. Προφανώς, αν έχουν τοποθετηθεί βολικά (πιο κοντά στην βέλτιστη λύση) , τότε θα έχει λιγότερες γραμμές η λύση. Αλλιώς, αν έχουν τοποθετηθεί μη βολικά η λύση θα χρειάζεται παραπάνω γραμμές, αφού θα γίνουν λιγότερες μεταθέσεις στο βήμα 2 του compaction.

<i>Netlist</i>		<i>Rows after compaction step 1</i>	<i>Rows after compaction step 2</i>
<i>Askhsh_2_FA_1bit_v1</i>	4	3	2
<i>Askhsh_2_FA_1bit_v2</i>	4	3	2
<i>Askhsh_2_FA_4bit_v1</i>	22	15	8
<i>Askhsh_2_FA_4bit_v2</i>	22	15	4
<i>Askhsh_2_FA_4bit_v3</i>	22	15	8
<i>Askhsh_2_FA_8bit_v1</i>	46	31	16
<i>Askhsh_2_FA_8bit_v2</i>	46	31	19

*\*ΠΡΟΣΟΧΗ : Για να τρέξει σωστά το πρόγραμμα πρέπει να τοποθετήσετε τα paths που βρίσκεται το netlist εισόδου, αλλά και το path που θέλετε να σας δημιουργήσει το netlist εξόδου\**