ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ, ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

Αλγορίθμοι λογικής σύνθεσης

ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Δρασίδης Γεώργιος

Βόλος, Απρίλιος 2010

Ευχαριστίες

Η εργασία αυτή είναι αφιερωμένη στους γονείς Ισμήνη και Τάσο, και στην αδερφή Μαρίλεια που υπάρχουν και με στηρίζουν σε κάθε βήμα της ζωής μου. Επίσης ευχαριστώ τον κ.Σταμούλη για την εμπιστοσύνη που μου έδειξε από την πρώτη στιγμή και μου έδωσε την ευκαιρία να δουλέψω μαζί του και με την ομάδα του, όπως επίσης και για τις ώρες καθοδήγησης και συμβουλών που ξόδεψε.

Τον Μιχάλη για την συμπαράσταση του όλα αυτά τα χρόνια, την υποστήριξη και τις ατελείωτες συζητήσεις αποκωδικοποίησης καταστάσεων και κυρίως για το ότι είναι αφοσιωμένος φίλος. Τέλος τον Δ.Καραμπατζάκη για το ξεκίνημα και την καθοδήγηση.

Table of Contents

1. Εισαγωγή

Τα Κυκλώματα Πολύ Μεγάλης Κλίμακας Ολοκλήρωσης (Very Large Scale Integration-VLSI) κυκλώματα έχουν στις μέρες μας τον βασικό ρόλο στην ανάπτυξη πολύπλοκων ηλεκτρονικών συστημάτων. Τα κυκλώματα αυτά αποτελούνται πλέον από πολλά εκατομμύρια και πολλές φόρες δισεκατομμύρια τρανζίστορ γεγονός που καθιστά την ανάπτυξη τους πλέον μια επίπονη και χρονοβόρα διαδικασία. Η σχεδίαση τέτοιων κυκλωμάτων κάνει εκτεταμένη χρήση εργαλείων αυτοματοποιημένης σχεδίασης και σύνθεσης λογικής, καθιστώντας δυνατή τη ανάπτυξη κυκλωμάτων αυξημένης πολυπλοκότητας. Βέβαια, υπάρχουν και εξαιρέσεις που επιβεβαιώνουν τον κανόνα καθώς κάποια δομικά μπλοκ κυκλωμάτων, όπως τα SRAM κελιά σχεδιάζονται έτσι ώστε να επιτευχθεί μεγαλύτερη αποδοτικότητα, καταστρατηγώντας πολλές φορές εδραιωμένους κανόνες σχεδίασης.

Η σχεδίαση συστημάτων πολλών εκατομμυρίων τρανζίστορ είναι πολύ απαιτητική εάν λάβουμε υπόψιν και την πίεση της αγοράς για γρήγορα αποτελέσματα και ορθή λειτουργία. Ενώ παλαιότερα ο χρόνος εισαγωγής στην αγορά ήταν 3-5 χρόνια, τώρα πλέον πολλά προϊόντα έχούν περιθώριο μόνο ένα χρόνο, γεγονός που καθιστά την καθυστέρηση κυκλοφορίας ενός νέου προϊόντος απαγορευτική, καθώς ο αντίκτυπος στα κέρδη μπορεί να είναι τεράστιος [1].

Εξαιτίας των παραπάνω είναι απαραίτητο να αυτοματοποιήσουμε όσο γίνεται περισσότερο την σχεδίαση, ώστε να αυξηθεί η παραγωγικότητα του σχεδιαστή, να εξαλειφθούν τυχόν λάθη, και περιπτώσεις εσφαλμένης λειτουργίας του κυκλώματος. Σε αυτή την προσπάθεια συμβάλλει τα μέγιστα η ανάπτυξη των εργαλείων αυτοματοποιημένου σχεδιασμού με την βοήθεια υπολογιστή (*Computer Aided Design*-**CAD**).

Οι τεχνικές που χρησιμοποιούνται από τα εργαλεία CAD παίζουν πολύ σημαντικό ρόλο στη μείωση του χρόνου σχεδιασμού και βελτιστοποίησης του κυκλώματος. Όσο αυξάνεται η πολυπλοκότητα και το μέγεθος του κυκλώματος τόσο αυξάνεται και η δυσκολία στο να πετύχουμε μια σχεδίαση χωρίς λάθη

ακόμα και για μια ομάδα σχεδιαστών. Επιπλέον, η βελτιστοποίηση κυκλωμάτων μεγάλης κλίμακας γίνεται ένα ακόμα πιο πολύπλοκο πρόβλημα επειδή ο αριθμός των επιλογών υλοποίησης αυξάνει δραματικά, ανάλογα με το μέγεθος του κυκλωμάτων.

Τα εργαλεία CAD παρέχουν ένα αποτελεσματικό μέσο για την σχεδίαση κυκλωμάτων. Για αυτό το λόγο τα εργαλεία CAD αν και έχουν φτάσει σε ένα αρκετά καλό επίπεδο ωριμότητας σε πολλούς τομείς, παραμένουν ένας τομέας συνεχούς έρευνας και εξέλιξης, με έμφαση στη βελτιστοποίηση των κυκλωμάτων.

Ένας πολύ βασικός τομέας των εργαλείων CAD είναι αυτός της **αυτόματης σύνθεσης (synthesis)** και βελτιστοποίησης των ψηφιακών κυκλωμάτων. Οι τεχνικές synthesis επιταχύνουν το κύκλο ολοκλήρωσης της σχεδίασης και διευκολύνουν την δουλειά του σχεδιαστή ψηφιακών κυκλωμάτων. Τεχνικές βελτιστοποίησης χρησιμοποιούνται για να βελτιώσουν την ποιότητα του κυκλώματος. Αν και χρησιμοποιούνται τεχνικές synthesis και βελτιστοποιήσεων για τις περισσότερες σχεδιάσεις κυκλωμάτων, οι δυνατότητες τους δεν έχουν αξιοποιηθεί στο μέγιστο.

Στα πλαίσια της μεταπτυχιακής αυτής εργασίας θα ασχοληθούμε με τον τομέα της σύνθεσης σε ψηφιακά κυκλώματα και πιο συγκεκριμένα με μία συγκεκριμένη κατηγορία σύνθεσης, την λογική σύνθεση ψηφιακών κυκλωμάτων και των βασικών αλγορίθμων που χρησιμοποιούνται από την τα εργαλεία CAD για την διαδικασία αυτή. Στο κεφάλαιο 2 θα δούμε κάποιους θεμελιώδεις ορισμούς, πάνω στους οποίους στηρίζονται πολλοί αλγόριθμοι από αυτούς που χρησιμοποιούνται αλλά και ορισμένες βασικές έννοιες. Στο κεφάλαιο 3 θα δοθεί ο ορισμός της λογικής σύνθεσης και μια θεωρητική προσέγγιση. Στα κεφάλαια 4 και 5 θα περιγραφούν οι βασικοί αλγόριθμοι, για τα δύο βασικά είδη λογικής σύνθεσης, το exact και τα heuristic. Τέλος στο κεφάλαιο 6 θα δοθούν τα αποτελέσματα και οι μετρήσεις πάνω σε κάποια επιλεγμένα κυκλώματα αναφοράς.

2. Βασικοί Ορισμοί

Στο κεφάλαιο αυτό θα δώσουμε κάποιους βασικούς ορισμούς εννοιών που θα χρησιμοποιούνται από εδώ και πέρα για να υπάρχει ένα σημείο αναφοράς για τις έννοιες που θα αναλυθούν παρακάτω.

Έστω B={0,1}, Y={0,1,2}. Μια **λογική (logic ή Boolean ή switching) συνάρτηση** με ν μεταβλητές εισόδου, $x_1,x_2,\ldots,x_\nu$, και μ μεταβλητές εξόδου, $y_1,y_2,\ldots,y_\mu$ είναι μια συνάρτηση [2] :

$$B^\nu \rightarrow Y^\mu$$

Όπου το $x = [x_1,x_2,\ldots,x_\nu] \in B^\nu$ είναι η είσοδος και $y = [y_1,y_2,\ldots,y_\mu]$ είναι η έξοδος συνάρτησης. Εκτός από τις τιμές 0 και 1, θεωρούμε ότι με 2 απεικονίζουμε την συνθήκη αδιαφορίας don't care (DC). Μια λογική συνάρτηση με μ=1 ονομάζεται συνάρτηση μιας εξόδου (**single-output**), ενώ για μ>1, ονομάζεται πολλαπλών εξόδων (**multiple-output**).

Η πιο συχνή αναπαράσταση μιας λογικής συνάρτησης είναι αυτή του *πίνακα αληθείας* (**truth table** ή **tabular form**). Σε αυτή την μορφή, προσδιορίζεται η τιμή της εξόδου για κάθε δυνατό συνδυασμό των εισόδων. Στο σχήμα που ακολουθεί έχουμε ένα παράδειγμα:

X_1	X_2	Y_1	Y_2
0	0	1	1
0	1	0	2
1	0	0	1
1	1	1	0

Σχήμα 2.1. Πίνακας αληθείας

Ως **ON-set**, **OFF-set** και **DC-set** ορίζουμε τα υποσύνολα του B^ν, των οποίων η τιμή για κάθε έξοδο της f είναι 1, 0 ή 2 αντίστοιχα. Μια πλήρως ορισμένη συνάρτηση f ορίζεται ως ο συνδυασμός των τριών αυτών συνιστωσών.

Όρος γινομένου (**product term**) είναι ένα Boolean γινόμενο (AND) κάποιων literals. Εάν κάποιο product term αποτιμάται σε 1 για κάποιο ελαχιστόρο, τότε λέμε ότι το product term περιέχει τον ελαχιστόρο αυτό.

Implicant μιας συνάρτηση f είναι ένας όρος γινομένου (product term) p, ο οποίος περιλαμβάνεται στην συνάρτηση ($p \leq f$), δηλαδή δεν περιέχει κάποιον

ελαχιστόρο στο OFF-set της συνάρτησης. Για παράδειγμα το xy' και xyz είναι implicants της f = xy'+yz.

Prime Implicant είναι ένας implicant της f που δεν περιέχεται σε κανένα άλλο implicant της f. Για παράδειγμα σε συνέχεια από το προηγούμενο το xy' είναι prime implicant, ενώ το xyz δεν είναι αφού με απαλοιφή του x παίρνουμε το yz.

Εάν ένας prime implicant περιέχει έναν ελαχιστόρο που δεν περιέχεται σε κανένα άλλο prime implicant, τότε ονομάζεται ***essential.*** Στο προηγούμενο παράδειγμα, τα xy' και yz είναι essentials, ενώ το xz δεν είναι.

2.1. Κύβοι (Cubes) και Καλύμματα (Covers)

Ένα **cube** [1] είναι μια κωδικοποιημένη αναπαράσταση μιας λογικής συνάρτησης. Έστω ότι το p είναι ένας όρος γινομένου που σχετίζεται με ένα αλγεβρικό άθροισμα γινομένων με ν εισόδους και μ εξόδους. Σύμφωνα με αυτά ένα cube p προσδιορίζεται από ένα διάνυσμα $\boldsymbol{c = [c_1,\ldots, c_v, c_{v+1},\ldots, c_{v+\mu}]}$, όπου τα πρώτα ν στοιχεία του διανύσματος αναπαριστούν τις εισόδους και τα υπόλοιπα μ-ν απεικονίζουν τις εξόδους.

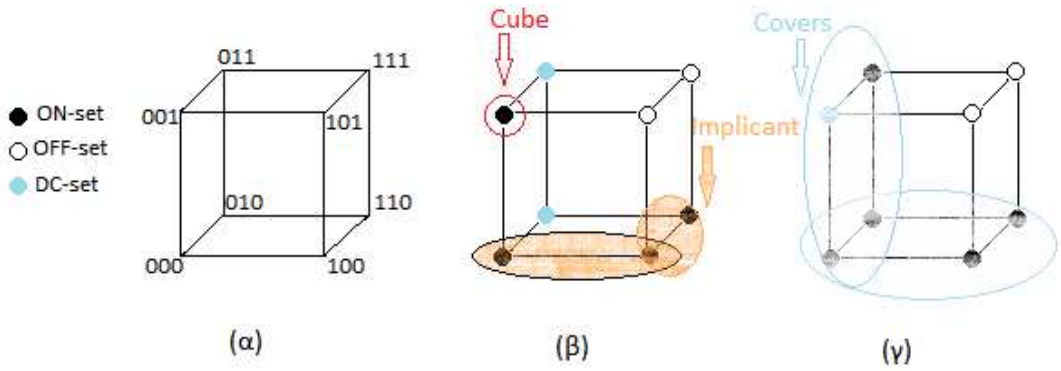
Ένα **cover [1]** μιας λογικής συνάρτησης είναι ένα σύνολο από cubes C = $[c^1,c^2,\ldots,c^k]$. Το cover αναπαριστά την ένωση του ON-set και κάποιου απαραίτητου μέρους του DC-set.

X_1	X_2	X_3	F_1	F_2
0	0	0	1	1
0	0	1	1	2
0	1	0	2	1
0	1	1	2	1
1	0	0	1	1
1	0	1	0	0
1	1	0	1	1
1	1	1	0	0

Σχήμα 2.2. Πίνακας αληθείας

Για παράδειγμα για την συνάρτηση τριών μεταβλητών εισόδου και 2 εξόδων του σχήματος 2.2 μπορούμε να δούμε σε γραφική αναπαράσταση (*Σχήμα 2.3*)

την έννοια του ON-, OFF-, DC-set . Κάθε κορυφή είναι ένα cube και κάθε συνδυασμός από κορυφές που ανήκουν στο ON- ή DC-set είναι ένα cover της αντίστοιχης εξόδου της συνάρτησης. Ο συνδυασμός των δύο cubes στο σχήμα 2.3.β αποτελεί ένα implicant.



Σχήμα 2.3.α) αναπαράσταση συνάρτησης τριών μεταβλητών, β) ON-, OFF-, DC-set covers της f1, γ) ON-, OFF-, DC-set covers της f2

Λέμε ότι ένα cube S **περιέχει (contains) [3]** ένα cube T, T⊆S, εάν $T_i \subseteq S_i$ για κάθε i = 1…n. Εάν επιπλέον S≠T, τότε το S λέμε ότι **περιέχει αυστηρά (strictly contains)** το T , T ⊂ S. Το S περιέχει (αυστηρά) το T εάν το S περιέχει (αυστηρά) όλους τους ελαχιστόρους του T.

Η **τομή** [1]των cubes S και T, **S∩T**, είναι το *μεγαλύτερο cube* που περιέχεται και στο S και το T. Εάν η τομή δύο cubes είναι το κενό σύνολο, τότε τα cubes ονομάζονται ορθογώνια. Ομοίως, τομή δύο cover F και G ορίζεται ως η ένωση της τομής ζευγών cubes από κάθε τα οποία το πρώτο περιέχεται στο ένα cover και το δεύτερο στο άλλο.

Η **ένωση** ή αλλιώς **υπερκύβος (supercube)[1][4]** δύο cubes S και T, S+T, είναι το ελάχιστο cube που μπορεί να περιέχει τα άλλα δύο cubes.

Η **απόσταση (distance)[1]** μεταξύ δύο cubes S και T είναι ο αριθμός των συγκρούσεων στην είσοδο, όπου συγκρούσεις ορίζουμε τις διαφορετικές τιμές για δύο cubes στις αντίστοιχες θέσεις της εισόδου, ή διαφορετικά τον αριθμό των κενών literals της τομής των δυο cubes. Αν είναι 0 τέμνονται, διαφορετικά δεν τέμνονται.

Το **sharp product [1][4]** ενός cube S και T είναι το μηδενικό product term, εάν τα S και T έχουν μηδενική τομή, διαφορετικά ορίζεται ως: ***S#T = S∩T'.***

Δηλαδή το S#T περιέχει όλους τους ελαχιστόρους του S που δεν περιέχονται στο T.

$$\alpha\#\beta = \begin{cases} a_1 \cdot b_1' & a_2 & \dots & a_n \\ a_1 & a_2 \cdot b_2' & \dots & a_n \\ \dots & \dots & \dots & \dots \\ a_1 & & \dots & a_n \cdot b_n' \end{cases}$$

Το **consensus [1]**δύο cubes S και T είναι ένα cube που ορίζεται όπως φαίνεται παρακάτω:

$$CONSENSUS(\alpha, \beta) = \begin{cases} a_1 + b_1 & a_2 \cdot b_2 & \dots & a_n \cdot b_n \\ a_1 \cdot b_1 & a_2 + b_2 & \dots & a_n \cdot b_n \\ \dots & \dots & \dots & \dots \\ a_1 \cdot b_1 & a_2 \cdot b_2 & \dots & a_n + b_n \end{cases}$$

Το consensus είναι κενό όταν δύο cubes έχουν απόσταση μεγαλύτερη ή ίση με 2. Εάν η απόσταση είναι 1 τότε το consensus δίνει ένα μεμονωμένο product term. Το consensus δύο cover F και G ορίζεται ως η ένωση του ανά ζεύγη consensus των product terms για κάθε cover. Για παράδειγμα, έστω οι παρακάτω implicants:

α	01	10	01
β	01	11	10
γ	10	01	01

Οι πρώτοι 2 implicants έχουν απόσταση 1 => *Consensus(α,β)=01 10 11*. Η απόσταση μεταξύ των υπολοίπων συνδυασμών είναι 2 => το consensus τους είναι το κενό.

Το **cofactor [1]** ενός cube C ως προς p, δεδομένου ενός συνόλου από cubes $C=\{c^1,\dots c^n\}$ και ενός cube p, είναι ένα νέο σύνολο από cubes C_p, που υπολογίζεται από το cofactor του κάθε cube του C. Cofactor του c^i ως προς p, είναι κενό όταν το c δεν τέμνεται με το p. Διαφορετικά δίνεται από τον τύπο:

$$C_p = C_1+p_1' \quad C_2+p_2' \quad \dots \quad C_n+p_n'$$

Ένα cover C τέτοιο ώστε κανένα subset του C δεν είναι επίσης cover χαρακτηρίζεται ελάχιστο (**minimal** ή **irredundant**)

2.2. Positional Cube Notation (PCN)

Το Positional Cube Notation (PCN)[1] είναι μια δυαδική κωδικοποίηση των implicants. Ένα cover μπορεί να αναπαρασταθεί μόνο με το μέρος που

αντιστοιχεί στο input για single-output συναρτήσεις. Το PCN κωδικοποιεί κάθε μεταβλητή με 2 bit. Αν και αυτός ο τρόπος διπλασιάζει των αριθμό των στηλών, διευκολύνει τον χειρισμό των implicants. Για παράδειγμα, η τομή 2 implicants σε PCN είναι το bitwise γινόμενο τους.

Κενό	**00**
0	**01**
1	**10**
DC	**11**

Σχήμα 2.4. PCN κωδικοποίηση

Για multiple-output functions πρέπει να προσθέσουμε και το μέρος της συνάρτησης που αναπαριστά τις εξόδους. Αυτό μας δίνει ένα διάνυσμα με δυαδικές τιμές, με τόσες πεδία όσες είναι οι είσοδοι και μια επιπλέον για τις εξόδους. Τα τελευταίο πεδίο έχει τόσα bit όσες είναι οι έξοδοι. Άρα η PCN αναπαράσταση για κάθε cube απαιτεί 2ν+μ bits.

10	10	100
10	01	001
01	10	001
01	01	110

Σχήμα 2.5. PCN αναπαράσταση του $f_1=a'b'+ab$, $f_2=ab$, $f_3=ab'+a'b$

2.3. Shannon expansion και unate functions

Οι βασικοί ευριστικοί (heuristic) αλγόριθμοι που χρησιμοποιούνται, εκτός από τον expand, βασίζονται στην στρατηγική του «διαίρει και βασίλευε» (divide and conquer). Η αποδόμηση ενός προβλήματος σε μικρότερα γίνεται με τη βοήθεια του θεωρήματος του Shannon[2][4], το οποίο με την σειρά του κάνει χρήση των cofactors λογικών συναρτήσεων.

Έχει αποδειχτεί ότι το πρόβλημα εύρεσης του μικρότερου cube που περιέχει το συμπλήρωμα μιας συνάρτησης, το οποίο είναι σημαντικό βήμα του αλγορίθμου Reduce, και το πρόβλημα προσδιορισμού αν μια συνάρτηση είναι ταυτολογία, σημαντικό μέρος για τους αλγορίθμους Irredundant και Essential, μπορεί να απαντηθεί γρήγορα με χρήση unate functions[4]. Συνδυάζοντας τα

αποτελέσματα αυτά με το θεώρημα του Shannon και τον τελεστή cofactor, προκύπτουν αποδοτικοί αναδρομικοί αλγόριθμοι που προσπαθούν να χωρίσουν την συνάρτηση με τέτοιο τρόπο ώστε να καταλήξουμε σε ένα φύλλο του αναδρομικού δέντρου, στο οποίο η συνάρτηση είναι unate, και μπορεί να υπολογιστεί γρήγορα.

Έστω g μια αλγεβρική αναπαράσταση μιας λογικής συνάρτησης G και g_{xj} και $g_{x'j}$ οι αλγεβρικές αναπαραστάσεις των cofactors ως προς x_j και x'_j τότε το Shannon expansion του g είναι :

$$\mathbf{g = x_j g_{xj} + x'_j gx'_j}$$

Το Shannon expansion με όρους αλγεβρικής αναπαράστασης αντιστοιχεί στη ¨συλλογή¨ κοινών όρων από τον όρο γινομένου.

Μια πράξη μπορεί υλοποιηθεί συγχωνεύοντας τα αποτελέσματα από την εφαρμογή της πράξης στους cofactors ως προς μια επιλεγμένη μεταβλητή. Δηλαδή, πράξεις σε σύνολα από Implicants μπορούν να υπολογισθούν αναδρομικά. Σε κάθε επίπεδο αναδρομής προκύπτει ένα απλούστερο πρόβλημα, το οποίο και πρέπει να λυθεί.

Ενώ οι περισσότερες λογικές συναρτήσεις δεν είναι unate, η αναδρομική αποδόμηση του προβλήματος μπορεί να οδηγήσει σε cofactors που είναι unate και των οποίων η επεξεργασία είναι πολύ αποδοτική. Η τεχνική του Unate Recursive Paradigm από τον Brayton εκμεταλλεύεται τις ιδιότητες των unate functions. Με την χρήση των unate functions μπορούμε να κάνουμε πιο αποδοτικές τις διαδικασίες του tautology και του reduction.

Μια λογική συνάρτηση f είναι **μονότονα αύξουσα (μονότονα φθίνουσα)** σε μια μεταβλητή x_j, εάν αλλάζοντας την τιμή του x_j από 0 σε 1, προκαλεί αλλαγές σε όλες τις εξόδους της f από 0 σε 1 (1 σε 0). Μια f η οποία είναι μονότονα αύξουσα ή φθίνουσα ως προς μια μεταβλητή ονομάζεται **unate** για την συγκεκριμένη μεταβλητή. Ένα cover C είναι μονότονα αύξον ως προς μια μεταβλητή x_j, αν όλα τα cubes του C έχουν είτε 1 είτε 2 (αδιάφορο) στη θέση j. Ένα cover λέγεται unate εάν είναι μονότονο ως προς όλες τις μεταβλητές εισόδου.

Μια συνάρτηση f είναι ***weakly unate[2]][4]*** ως προς μια μεταβλητή x_i εάν υπάρχει μια τιμή j τέτοια ώστε αν αλλάξουμε την τιμή του x_i από j σε

οποιαδήποτε άλλη τιμή k, η συνάρτηση δεν μειώνεται, δηλαδή εάν αλλάξει τιμή, αλλάζει από 0 σε 1. Ή αλγεβρικά: F(…, Xi = j, …) ≤ F(…, Xi = k, …), k≠j.

Μια συνάρτηση ή ένα cover είναι strongly/weakly unate if it is strongly/weakly unate σε όλες τις μεταβλητές του.

Οι ιδιότητες που ακολουθούν των weakly unate συναρτήσεων είναι σημαντικές για να προσδιορίσουμε αν μια συνάρτηση που αναπαριστάται από ένα cover είναι ταυτολογία και άρα να γίνει πιο αποδοτική η επίλυση του προβλήματος. Στις παρακάτω περιπτώσεις λέμε ότι το cover είναι ταυτολογία.

- **Θεώρημα:** Έστω F ένα weakly unate cover ως προς τη μεταβλητή x και G υποσύνολο του F που δεν εξαρτάται από το x. Τότε το F είναι μια ταυτολογία αν και μόνο αν η G είναι ταυτολογία.
- **Θεώρημα:** Ένα weakly unate cover είναι ταυτολογία αν και μόνο αν ένας από τους implicants είναι ο καθολικός cube U.

2.4. Ταυτολογία

Η ταυτολογία[1][5] παίζει πολύ σημαντικό ρόλο σε όλους τους αλγορίθμους λογικής βελτιστοποίησης. Χρησιμοποιείται από βασικούς αλγορίθμους του Espresso, όπως είναι οι reduce, irredundant και last gasp τους οποίους θα εξετάσουμε στη συνέχεια. Συνεπώς ένας αποδοτικός αλγόριθμος ταυτολογίας είναι ζωτικής σημασίας. Παρόλο που ως πρόβλημα θεωρείται intractable, η ερώτηση αν μια συνάρτηση είναι ταυτολογία μπορεί να απαντηθεί αποδοτικά χρησιμοποιώντας το recursive paradigm.

Για συναρτήσεις με δυαδικές τιμές, μια συνάρτηση είναι ταυτολογία αν και μόνο αν οι cofactors του ως προς οποιαδήποτε μεταβλητή αλλά και ως προς το συμπλήρωμα είναι και οι δύο ταυτολογίες.

Η μεθοδολογία που ακολουθούμε για την επίλυση του προβλήματος είναι να εφαρμόζουμε τον τελεστή expand ως προς μια μεταβλητή στην συνάρτηση και να ελέγχουμε εάν οι cofactors της είναι ταυτολογίες. Όταν μια συνάρτηση είναι δυαδική και unate ως προς μια μεταβλητή, χρειάζεται να ελέγξουμε μόνο τον ένα cofactor για ταυτολογία, γιατί εάν είναι ταυτολογία ο ένας cofactor συνεπάγεται ότι θα είναι ταυτολογία και ο άλλος. Για παράδειγμα, εάν η συνάρτηση f είναι θετικά unate για την μεταβλητή x, τότε $f_{x'} \subseteq f_x$ και επιπλέον f_x είναι ταυτολογία όταν είναι και η $f_{x'}$.

Για τον τερματισμό της αναδρομικής διαδικασίας επίλυσης του προβλήματος ισχύουν έξι κανόνες:

i. Ένα cover είναι ταυτολογία όταν έχει μια σειρά μόνο από 1
ii. Ένα cover δεν είναι ταυτολογία όταν έχει μια στήλη μόνο από 0
iii. Ένα cover είναι ταυτολογία όταν εξαρτάται από μια μεταβλητή και δεν υπάρχει στήλη μόνο από 0 σε αυτό το πεδίο.
iv. Ένα cover δεν είναι ταυτολογία όταν είναι weakly unate και δεν υπάρχει μια σειρά μόνο από 1, αφού ουσιαστικά δεν αναγνωρίζεται ο πρώτος κανόνας.
v. Το πρόβλημα μπορεί να απλοποιηθεί όταν ένα cover είναι weakly unate ως προς κάποια μεταβλητή. Ο έλεγχος για ταυτολογία μπορεί να εφαρμοστεί στο υποσύνολο των γραμμών που εξαρτώνται από την weakly unate μεταβλητή.
vi. Όταν ένα cover C μπορεί να εκφραστεί ως AUB, όπου τα A και B ορίζονται από μη κοινές μεταβλητές, τότε το C είναι ταυτολογία αν και μόνο αν οποιοδήποτε από τα A ή B είναι ταυτολογία.

Όταν δεν μπορεί να αποδειχθεί μια ταυτολογία με κάποιο από τους παραπάνω κανόνες τερματισμού, τότε στην συνάρτηση εφαρμόζεται το expand ως προς μια μεταβλητή. Πολύ σημαντικό βήμα είναι σε αυτή την περίπτωση η επιλογή της splitting μεταβλητής για να είναι αποδοτικός ο αλγόριθμος.

Για την εύρεση της κατάλληλης splitting μεταβλητής χρησιμοποιείται ένας heuristic αλγόριθμος που προτάθηκε αρχικά από τον Brayton. Ο αλγόριθμος προσπαθεί να επιλέξει μια μεταβλητή που είναι πιθανότερο να δημιουργήσει 2 unate υπό-προβλήματα. Γι αυτό το λόγο επιλέγεται η μεταβλητή από την οποία εξαρτώνται οι περισσότεροι implicants. Η γενικότερη στρατηγική είναι να κρατάμε το δέντρο της αναδρομής όσο πιο ισορροπημένο γίνεται.

Παράδειγμα: έστω η συνάρτηση f = ab + ac + ab'c' + a'. Θέλουμε να ελέγξουμε αν είναι ταυτολογία:

01 01 11
01 11 01
01 10 10
10 11 11

Σχήμα 2.6. PCN αναπαράσταση[1]

Από τις μεταβλητές επιλέγουμε την α που επηρεάζει τους περισσότερους implicants, ως splitting μεταβλητή.

Παίρνουμε το cofactor ως προς α' => *C(α') = 10 11 11* και ως προς α => C(α)=01 11 11. Ο πρώτος cofactor έχει μια μόνο γραμμή η οποία είναι 11 11 11, άρα και είναι ταυτολογία. Ο δεύτερος cofactor είναι :

11 01 11
11 11 01
11 10 10

Αφού δεν ισχύει καμία από τις συνθήκες τερματισμού, βρίσκουμε την δεύτερη splitting variable και υπολογίζουμε τους cofactors ως προς C(b') = 11 10 11 και ως προς C(b) = 11 01 11. Ο πρώτος cofactor είναι:

11 11 01
11 11 10

το οποίο είναι ταυτολογία και δεύτερος cofactor 11 11 11 ο οποίος είναι και αυτός ταυτολογία. Άρα και η f είναι ταυτολογία.

2.5. Containment

Είναι η διαδικασία που απαντάει στην ερώτηση εάν μια F περιέχει/καλύπτει[1] έναν implicant α. Το πρόβλημα αυτό ανάγεται σε πρόβλημα εύρεσης ταυτολογίας χρησιμοποιώντας το παρακάτω θεώρημα.

Θεώρημα : Μια συνάρτηση F περιέχει έναν implicant α αν και μόνον αν ο F_α είναι ταυτολογία.

Για παράδειγμα, αν θέλουμε να εξετάσουμε στην συνάρτηση f = ab + ac +a', αν το bc καλύπτεται. Η PCN αναπαράσταση του bc είναι 11 01 01. Το cofactor είναι:

01 11 11
01 11 11
10 11 11

το οποίο είναι ταυτολογία αφού εξαρτάται μόνο από μια μεταβλητή και δεν υπάρχει στήλη μόνο με μηδενικά, άρα το bc καλύπτεται από την f.

3. CAD σύνθεση και βελτιστοποιήσεις

Στο κεφάλαιο αυτό θα εξετάσουμε μια θεωρητική προσέγγιση της σύνθεσης γενικά ως έννοιας και των επιπέδων έως ότου μία σχεδίαση φτάσει στο επίπεδο του πυριτίου. Ακολούθως θα επικεντρώσουμε στο επίπεδο της λογικής σύνθεσης, όπου και θα εξετάσουμε τα είδη της και τους αλγόριθμους που χρησιμοποιούνται για την σύνθεση σε αυτό το επιπεδο.

3.1. Είδη σύνθεσης

Ένα μοντέλο ενός κυκλώματος είναι μια αφαίρεση, δηλ. μια αναπαράσταση σχετικών χαρακτηριστικών χωρίς επιμέρους λεπτομέρειες. Ως ***σύνθεση[8]*** ορίζουμε την δημιουργία ενός κυκλωματικού μοντέλου, ξεκινώντας από ένα λιγότερο λεπτομερές. Τα μοντέλα μπορούν να κατηγοριοποιηθούν ανάλογα με το επίπεδο της αφαίρεσης. Υπάρχουν 3 βασικά επίπεδα, δηλαδή:

1) επίπεδο αρχιτεκτονικής (**behavioral** ή **architectural** synthesis)
2) επίπεδο λογικής (**logic synthesis**)
3) επίπεδο φυσικό (**layout synthesis**)

Στο επίπεδο της αρχιτεκτονικής ασχολούμαστε με τις λειτουργίες που επιτελεί το κύκλωμα, σε λογικό επίπεδο έχουμε ένα σύνολο από λογικές συναρτήσεις και τέλος στο φυσικό επίπεδο έχουμε ένα σύνολο από γεωμετρικές οντότητες (μέταλλα, πολυπυρίτιο, κτλ.).

Η κατηγοριοποίηση των κυκλωματικών μοντέλων σχετίζεται με την κατηγοριοποίηση των διαδικασιών της σύνθεσης. Συγκεκριμένα μπορούμε να ξεχωρίσουμε τις επιμέρους διαδικασίες της σύνθεσης ανάλογα με τα διαφορετικά επίπεδα του κυκλωματικού μοντέλου σε:

Σύνθεση αρχιτεκτονικού επιπέδου ή ***Σύνθεση Υψηλού επιπέδου [9],*** όπου παράγεται μια δομική αναπαράσταση του κυκλώματος. Αυτό ισοδυναμεί με μία αυτοματοποιημένη διαδικασία σχεδιασμού που μετατρέπει μία αλγοριθμική περιγραφή του κυκλώματος στο αντίστοιχο υλικό (*hardware*) που υλοποιεί την ζητούμενη συμπεριφορά για το κύκλωμα. Ο κώδικας αναλύεται, εισάγονται αρχιτεκτονικοί περιορισμοί και παράγεται ακολούθως μια RTL γλώσσα περιγραφής υλικού (HDL), η οποία με την σειρά της εισάγεται στο εργαλείο της λογικής σύνθεσης, όπου γίνεται η σύνθεση σε επίπεδο πυλών. Στόχος της σύνθεσης υψηλού επιπέδου είναι να δώσει την δυνατότητα στους

σχεδιαστές να δημιουργούν αποδοτικές σχεδιάσεις κυκλωμάτων, χρησιμοποιώντας ένα υψηλότερο επίπεδο περιγραφής και άρα πιο κοντά στην ανθρώπινη σκέψη. Το εργαλείο αναλαμβάνει την μετατροπή σε RTL υλοποίηση.

Σύνθεση λογικού επιπέδου[8] είναι η διαδικασία με την οποία μια αφηρημένη μορφή του επιθυμητού κυκλώματος , συνήθως σε RTL μορφή, μετατρέπεται σε μια σχεδίαση σε επίπεδο λογικών πυλών. Το μοντέλο λογικού επιπέδου μπορεί να παρέχεται από μία μηχανή πεπερασμένων καταστάσεων (FSM), εάν έχουμε να κάνουμε με ένα ακολουθιακό κύκλωμα, από ένα σχηματικό του κυκλώματος ή ισοδύναμα από ένα HDL μοντέλο, εάν πρόκειται για συνδυαστικό κύκλωμα. Το τελικό αποτέλεσμα της λογικής σύνθεσης είναι μια αναπαράσταση σε επίπεδο δομικών μονάδων, όπως είναι το netlist επιπέδου πυλών.

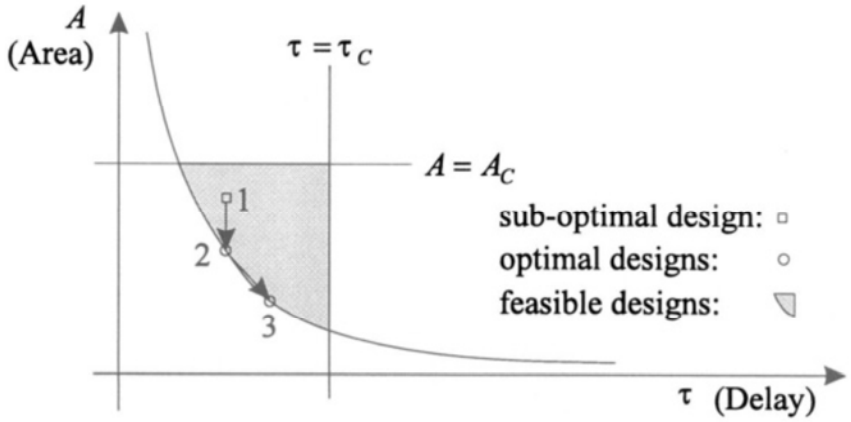
Σύνθεση φυσικού επιπέδου είναι η διαδικασία παραγωγής layout ενός κυκλώματος. Το γεωμετρικό layout είναι ο τελική μορφή της σχεδίασης. Αν και πάλι χρησιμοποιούνται αυτοματοποιημένες μέθοδοι, πολλές φορές χρειάζεται ανθρωπινή παρέμβαση για βελτιώσεις στο τελικό αποτέλεσμα. Οι βασικές εργασίες στη σχεδίαση φυσικού επιπέδου είναι το placement και το wiring η αλλιώς routing.

Μαζί με την σύνθεση του κυκλώματος γίνονται και κάποιες βελτιστοποιήσεις. Οι βελτιστοποιήσεις δεν γίνονται μόνο για την μεγιστοποίηση της ποιότητας του κυκλώματος, αλλά και για να υλοποιηθούν ανταγωνιστικά κυκλώματα, καθώς σε αντίθετη περίπτωση το κέρδος θα ήταν οριακό. Υπάρχουν 2 βασικές κατηγορίες βελτιστοποίησης, ως προς την ταχύτητα (καθυστέρηση) και ως προς την επιφάνεια που καταλαμβάνει το κύκλωμα.

Ως βελτιστοποίηση στο κύκλωμα ορίζουμε την εύρεση της καλύτερης σχεδίασης ως προς κάποιους περιορισμούς στα βασικά χαρακτηριστικά της σχεδίασης. Τα εργαλεία σύνθεσης δίνουν στους σχεδιαστές το μέσο για να ερευνήσουν και να βρουν την βέλτιστη καμπύλη επιφάνεια προς καθυστέρηση ανάμεσα στη βέλτιστη επιφάνεια και την βέλτιστη καθυστέρηση. Η βέλτιστη καμπύλη είναι μια υπερβολή με άξονες την επιφάνεια που καταλαμβάνει το κύκλωμα και την καθυστέρηση του (*Σχήμα 3.1*). Οι περιορισμοί καθυστέρησης προσδιορίζονται από την τάση της αγοράς προς μεγαλύτερες ταχύτητες και οι περιορισμοί στην επιφάνειας εξαρτώνται από το μέγεθος του ολοκληρωμένου

πάνω στο οποίο θα τοποθετηθεί το κύκλωμα και του προϋπολογισμού για την παραγωγή του.

Δεδομένων κάποιων προδιαγραφών καθυστέρησης , η σύνθεση για βέλτιστη επιφάνεια παράγει μια σχεδίαση που βρίσκεται στην τομή της γραμμής των περιορισμών καθυστέρησης $\tau=\tau_c$ και της βέλτιστης καμπύλης. Σχεδιάσεις που καταλαμβάνουν επιφάνεια μικρότερη από αυτή συνεπάγονται καθυστέρηση που υπερβαίνει τις προδιαγραφές.



Σχήμα 3.1. Βέλτιστη καμπύλη περιοχής-καθυστέρησης[2]

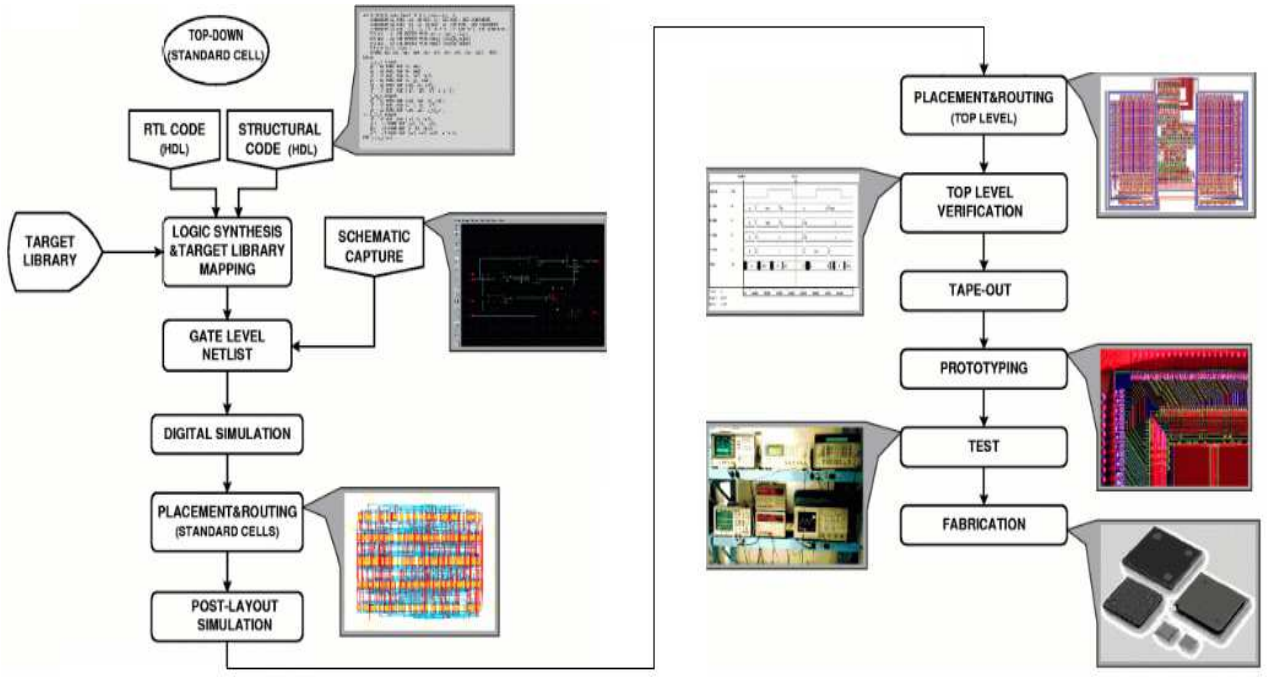
Αντίστοιχα, δεδομένων προδιαγραφών επιφάνειας, η σύνθεση για βέλτιστη καθυστέρηση θα παράγει μια σχεδίαση που βρίσκεται στην τομή της γραμμής περιορισμού επιφάνειας $A=A_c$ και της βέλτιστης καμπύλης. Σχεδιάσεις με καθυστερήσεις μικρότερες από αυτό συνεπάγονται επιφάνεια που υπερβαίνει τις προδιαγραφές.

Στο σχήμα 3.1, εφικτά κυκλώματα, αλλά μη-βέλτιστα είναι κυκλώματα τα οποία βρίσκονται μέσα στην σκιασμένη περιοχή, ενώ βέλτιστα είναι αυτά τα οποία βρίσκονται πάνω στην καμπύλη. Δηλαδή η Σχεδίαση1 είναι εφικτή αλλά μη βέλτιστη, ενώ οι Σχεδιάση2 και Σχεδιαση3 είναι και εφικτά και βέλτιστα καθώς βρίσκονται πάνω στην βέλτιστη καμπύλη. Γενικά, υπάρχουν πολλές διαφορετικές σχεδιάσεις που μπορεί να είναι βέλτιστες για διαφορετικούς περιορισμούς και εξαρτάται από τον σχεδιαστή ποια θα επιλέξει ανάλογα με το βελτιστοποίηση που θέλει να επιτύχει. Για παράδειγμα μια σχεδιαστική μεταφορά από την Σχεδίαση1 στην Σχεδιαση2 μειώνει την επιφάνεια που καταλαμβάνει το κύκλωμα χωρίς να αυξάνει την καθυστέρηση, ενώ από την Σχεδίαση2 στην Σχεδίαση3 είναι μια κίνηση πάνω στη βέλτιστη καμπύλη, όπου

θυσιάζουμε καθυστέρηση για να επιτύχουμε μείωση στην επιφάνεια που καταλαμβάνει το κύκλωμα. Η δυνατότητα να κάνουμε τέτοιες μετατροπές μεταξύ των σχεδιάσεων αποτελεί ένα βασικό παράγοντα που εκμεταλλεύονται τα εργαλεία σύνθεσης. Αν και συνήθως επιδιώκουμε μεγάλη απόδοση, μερικές φορές μπορεί να επιδιώξουμε να ελαχιστοποιήσουμε την επιφάνεια ώστε να χωρέσει το κύκλωμα σε ένα ολοκληρωμένο.

3.2. Ο "δρόμος" προς το πολυπυρίτιο

Μια σειρά από εργαλεία CAD έχούν αναπτυχθεί με σκοπό να βοηθήσουν τον σχεδιαστή στα διάφορα στάδια της σύνθεσης μέχρι το τελικό επίπεδο του πυριτίου. Στο σχήμα 3.2 μπορούμε να δούμε ολόκληρη την διαδικασία της από την σχεδίαση μέχρι την παραγωγή του τσιπ. Τα εργαλεία που χρησιμοποιούνται για την διαδικασία αυτή, κατηγοριοποιούνται ανάλογα με τα επίπεδα τις αφαίρεσης όπως περιγράφονται παρακάτω:



Σχήμα 3.2. Ροή Σχεδίασης για ένα κύκλωμα

Behavioral Synthesis

Resource Allocation
Pipelining
Control Flow Parallelization
Communicating Sequential Processes
Partitioning

Sequential Synthesis

Register Movement and Retiming
State Minimization
State Assignment (Encoding)
Synthesis for Testable FSM's
State Machine Verification

Logic Synthesis

Extraction of combinational logic from HDLs
Two-level (PLA) minimization
Algebraic Decomposition
Multilevel Logic Minimization
Synthesis for Multi-fault Testability
Test Generation via Minimization
Technology Mapping
Timing Optimization

Technology Mapping

Mapping to Library of Logic Gates
Timing Optimization

Physical Design Synthesis

Cell Placement
Routing
Fabrication
Engineering Changes

Τα εργαλεία λογικής σύνθεσης παρέχουν στους σχεδιαστές συστημάτων VLSI νέες δυνατότητες. Η μια είναι η αυτόματη μετάφραση γλωσσών υψηλού-επιπέδου (C, C++ κτλ) σε λογικές σχεδιάσεις, ενώ η δεύτερη είναι η αυτόματη βελτιστοποίηση, χωρίς την παρέμβαση του σχεδιαστή, στην επιφάνεια που καταλαμβάνεται στο ολοκληρωμένο, στην ταχύτητα και στην ορθή λειτουργία του κυκλώματος. Τα εργαλεία δηλαδή δίνουν στους σχεδιαστές είτε ένα γρήγορο και πιο εύκολο δρόμο για μια εξ ολοκλήρου *top-down* σχεδίαση, είτε απλά χρησιμοποιούνται για βελτιστοποιήσεις σε μία σχεδίαση που έχει γίνει χωρίς παρέμβαση εργαλείων σύνθεσης. Τα πλεονεκτήματα αυτής της αυτοματοποιημένης μεθοδολογίας σύνθεσης για σχεδιάσεις VLSI είναι προφανή. Παρέχουν μειωμένο χρόνο σχεδιασμού, ελαχιστοποιούν την πιθανότητα σχεδιαστικού λάθους και δίνουν σχεδιάσεις καλύτερης ποιότητας.

Στην εργασία αυτή όπως είπαμε θα ασχοληθούμε με ένα συγκριμένο τομέα της σύνθεσης, ο οποίος είναι αυτός της λογικής σύνθεσης και των αλγορίθμων που χρησιμοποιούνται κατά την διαδικασία αυτή από τα διάφορα εργαλεία CAD.

3.3. Λογική σύνθεση

Λογική σύνθεση είναι η διαδικασία κατά την οποία μια αφηρημένη μορφή του επιθυμητού κυκλώματος, συνήθως ένα RTL, μετατρέπεται σε μια υλοποίηση λογικών πυλών. Οι αρχές της λογικής σύνθεσης εντοπίζονται στην συνεισφορά του George Boole για τον χειρισμό της λογικής με την χρήση της άλγεβρας που ονομάστηκε άλγεβρα Boole. Τα κυκλώματα μπορούν να χωριστούν σε δύο κατηγορίες :

1) Κυκλώματα δύο επιπέδων
2) Κυκλώματα πολλαπλών επιπέδων

Αρχικά έγιναν γνωστά τα κυκλώματα δύο επιπέδων, διότι ήταν τα μόνα κυκλώματα για τα οποία υπήρχαν αποδοτικές διαδικασίες σχεδιασμού. Αρκετοί αλγόριθμοι είχαν δημιουργηθεί για κυκλώματα δύο επιπέδων από την δεκαετία του 50 ακόμη. Επιπλέον η χρήση προγραμματιζόμενων λογικών διατάξεων (Programmable Logic Arrays - PLA) [2] έκανε εντονότερη την ανάγκη για αποδοτικούς αλγόριθμους ελαχιστοποίησης δύο επιπέδων, διότι κάτι τέτοιο συνεπάγεται δέσμευση λιγότερης περιοχής σε PLA.

Φυσικά τα κυκλώματα πολλαπλών επιπέδων είναι μεγάλης σημασίας για τα VLSI κυκλώματα, καθώς οι περισσότερες σχεδιάσεις χρησιμοποιούν πολλαπλά επίπεδα λογικής. Οι αλγόριθμοι για την βελτιστοποίηση σύνθεσης κυκλωμάτων πολλαπλών επιπέδων είναι αντικείμενο προς έρευνα, και δεν θα ασχοληθούμε με αυτό σε αυτή την εργασία.

3.3.1. Σύνθεση κυκλωμάτων δύο επιπέδων (Two-level Synthesis)

Οι αλγόριθμοι σύνθεσης ουσιαστικά δεν αντιμετωπίζουν το ίδιο το πρόβλημα, αλλά μια μοντελοποίηση του προβλήματος. Συνεπώς για να χρησιμοποιήσουμε αυτούς τους αλγόριθμους, πρέπει αρχικά να μοντελοποιήσουμε το πρόβλημα σε ένα μαθηματικό μοντέλο. Θα περιορίσουμε

την εργασία αυτή σε αλγορίθμους για ένα συγκεκριμένο στυλ σχεδιάσεων αυτό των συνδυαστικών κυκλωμάτων και πιο ειδικά των κυκλωμάτων δύο επιπέδων.

Τα δύο επίπεδα λογικής είναι το ελάχιστο που απαιτείται για την υλοποίηση μιας Boolean συνάρτησης. Υπάρχουν δυο βασικοί λόγοι για τους οποίους θέλουμε να υλοποιήσουμε ένα κύκλωμα σε δύο επίπεδα αντί πολλαπλών επιπέδων, τα οποία είναι *1) η ταχύτητα* και *2) η απλότητα* .

Η καθυστέρηση ενός κυκλώματος μπορεί να εξαρτάται από πολλούς παράγοντες. Ο αριθμός των επιπέδων λογικής από τα οποία πρέπει να περάσει ένα σήμα είναι από τους πιο σημαντικούς παράγοντες. Συνεπώς, οι υλοποιήσεις δύο επιπέδων είναι γενικά γρήγορες, αν και ο μικρός αριθμός επιπέδων συνεπάγεται πολλές φορές μεγάλο fan-in και fan-out των πυλών, το οποίο επηρεάζει αρνητικά την ταχύτητα. Επιπλέον, τα κυκλώματα δύο επιπέδων είναι ευκολότερο να σχεδιαστούν και να αναλυθούν, αφού το εύρος των λύσεων είναι σαφώς περιορισμένο.

Ένας βασικός τομέας του synthesis δύο επιπέδων είναι η βελτιστοποίηση δύο επιπέδων (***two-level optimization***) [2]. Η βελτιστοποίηση δύο επιπέδων είναι σημαντική για τρείς βασικούς λόγους. Αρχικά, παρέχει ένα τρόπο για βελτιστοποίηση της υλοποίησης κυκλωμάτων που είναι ακριβείς αναπαραστάσεις από two-level tabular forms και έχουν άμεση επίδραση στον τρόπο σχεδιασμού των macro-cells χρησιμοποιώντας PLA’s. Επιπλέον, το two-level optimization μας επιτρέπει να μειώσουμε την πληροφορία που είναι απαραίτητη για την αναπαράσταση οποιασδήποτε λογικής συνάρτησης που αντιπροσωπεύει μια συνιστώσα μιας σχεδίασης πολλαπλών επιπέδων. Δηλαδή, ένα κομμάτι ενός δικτύου λογικής μπορεί να σχετίζεται με μια λογική συνάρτηση σε μορφή δύο επιπέδων, της οποίας η βελτιστοποίηση να επιφέρει γενικότερη βελτίωση σε ένα πολυ-επίπεδο κύκλωμα. Συνεπώς το two-level optimization παίζει ένα πολύ βασικό ρόλο στη σχεδίαση πολλαπλών επιπέδων.

3.3.2. Λογική Βελτιστοποίηση (Logic Optimization)

Logic optimization, είναι η διαδικασία εύρεσης ισοδύναμων αναπαραστάσεων για ένα συγκεκριμένο λογικό κύκλωμα που υπόκειται σε ένα ή περισσότερους περιορισμούς. Γενικά, ένα κύκλωμα περιορίζεται κυρίως ως προς την επιφάνεια που καταλαμβάνει σε ένα ολοκληρωμένο, ικανοποιώντας ταυτόχρονα και έναν περιορισμό καθυστέρησης. Μια από τις προκλήσεις είναι

να βρούμε την καλύτερη αναπαράσταση για την δεδομένη περιγραφή του κυκλώματος.

Στη διαδικασία της βελτιστοποίησης δύο επιπέδων χρησιμοποιούνται δύο διαφορετικά είδη αλγορίθμων, οι ακριβείς αλγόριθμοι (**exact**) και οι ευριστικοί (**heuristic**). Οι exact έχουν την βάση τους στην πρώιμη έρευνα που έγινε πάνω στη λογική σύνθεση από τους Quine και McCluskey, οι οποίοι αν και θεωρητικά μπορούν να δώσουν το ακριβές αποτέλεσμα, στην πράξη για μεγάλα κυκλώματα παρουσιάζουν απαγορευτική πολυπλοκότητα. Το πρόβλημα που καλούνται να λύσουν κατηγοριοποιείται στα NP-hard: δηλαδή ο χρόνος επίλυσης με τον αλγόριθμο αυτό αυξάνεται εκθετικά ανάλογα με τον αριθμό των μεταβλητών. Από την άλλη μεριά, οι ευριστικοί αλγόριθμοι με πιο σημαντικούς αυτούς που χρησιμοποιήθηκαν από προγράμματα όπως τα Mini, Presto και το de-facto standard για λογικής βελτιστοποίηση Espresso minimizer, αν και δεν δίνουν την ακριβή λύση καταφέρνουν μια πάρα πολύ καλή προσέγγιση χωρίς να απαιτούν τεράστιους πόρους από το σύστημα, ακόμα και για μεγάλα κυκλώματα. Οι βασικές αρχές των exact και heuristic αλγορίθμων για logic optimization θα περιγραφούν σε επόμενα κεφάλαια.

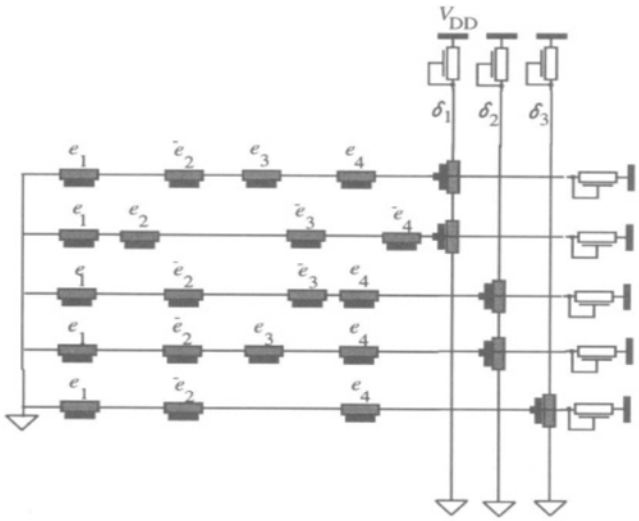
Ένα two-level κύκλωμα δύο επιπέδων μοντελοποιείται από εκφράσεις «αθροισμάτων γινομένων» ή ισοδύναμα από tabular forms όπως είναι οι πίνακες ελαχιστόρων. Ο στόχος της βελτιστοποίησης δυο επιπέδων είναι να μειώσει όσο γίνεται το μέγεθος των λογικών συναρτήσεων είτε σε μορφή αθροίσματος γινομένων (**Sum Of Products-SOP**) είτε σε μορφή γινομένου αθροισμάτων (**Product Of Sums-POS**). Αφού οποιαδήποτε από τις δύο μορφές μπορεί να παραχθεί από την άλλη χρησιμοποιώντας των νόμο De Morgan, διατηρώντας των αριθμό των όρων και των literals, θεωρούμε ότι μπορούμε να χρησιμοποιούμε την μορφή του αθροίσματος γινομένων για την ελαχιστοποίηση, χωρίς απώλεια της γενικότητας.

Μια διάταξη, η οποία χρησιμοποιείται για την υλοποίηση λογικών συναρτήσεων είναι τα PLA's. Μια διάταξη PLA (*Σχήμα 3.3*) είναι ένα ορθογώνιο macro-cell που αποτελείται από μια διάταξη από τρανζίστορ ευθυγραμμισμένα για να σχηματίζουν γραμμές σε αντιστοιχία με τους όρους της συνάρτησης και στήλες σε αντιστοιχία με τις εισόδους και τις εξόδους του κυκλώματος.

Κάθε γραμμή μιας PLA είναι σε αντιστοιχία με ένα όρο της SOP αναπαράστασης. Κάθε τρανζίστορ είναι σε αντιστοιχία με ένα literal της SOP.

Συνεπώς, ο βασικός στόχος του logic minimization είναι η μείωση των όρων και επιπλέον η μείωση των literals.

Η ελαχιστοποίηση της λογικής για συναρτήσεις με μια έξοδο ή πολλαπλές εξόδους ακολουθούν τις ίδιες αρχές, αλλά προφανώς η πολυπλοκότητα για συναρτήσεις πολλών εξόδων είναι μεγαλύτερη.



Σχήμα 3.3. NAND-NAND PLA[2]

3.3.3. Ελαχιστοποίηση με exact αλγόριθμους

Οι exact αλγόριθμοι αντιμετωπίζουν με ακρίβεια το πρόβλημα της εύρεσης ενός ελάχιστου καλύμματος (cover) της λογικής συνάρτησης. Ο αλγόριθμος που επινοήθηκε από τους **Quine και McCluskey** και πήρε το όνομα τους, ήταν ο πρώτος που έδινε λύση στο πρόβλημα. Ο αλγόριθμος βασίζεται στο θεώρημα του Quine, το οποίο οριοθετεί τον χώρο αναζήτησης βέλτιστης λύσης.

Θεώρημα : *Υπάρχει ένα ελάχιστο cover το οποίο είναι και prime[7]*

Το θεώρημα μας δίνει την δυνατότητα να περιορίσουμε την αναζήτηση για ένα ελάχιστο cover σε αυτά τα covers τα οποία αποτελούνται αποκλειστικά από prime implicants.

Ο McCluskey μετέτρεψε την αναζήτηση για ένα ελάχιστο cover σε ένα covering πρόβλημα[6] με την βοήθεια ενός prime implicants πίνακα. Ένας prime implicants πίνακας είναι ένας αλγεβρικός πίνακας δυαδικών τιμών **A,** του οποίου οι στήλες είναι σε μια ένα-προς-ένα αντιστοιχία με τους prime implicants της συνάρτησης και του οποίου οι γραμμές είναι σε μια ένα-προς-ένα αντιστοιχία με τους ελαχιστόρους της. Ένα στοιχείο του πίνακα $a_{ij} \in A \Leftrightarrow$ το j-

οστός prime καλύπτει τον i-οστό ελαχιστόρο. Ένα ελάχιστο cover είναι ένα ελάχιστο σύνολο από στήλες που καλύπτουν όλες τις γραμμές, ή ισοδύναμα ένα ελάχιστο σύνολο από primes που καλύπτουν όλους τους ελαχιστόρους.

Συνεπώς το covering πρόβλημα μπορεί να θεωρηθεί ως το πρόβλημα εύρεσης ενός δυαδικού διανύσματος **x** που αναπαριστά ένα σύνολο από primes με ελάχιστη πληθικότητα τέτοια ώστε:

$$\mathbf{Ax \geq 1}$$

Όπου οι διαστάσεις του πίνακα A και του διανύσματος x αλλά και του 1 ταιριάζουν με τον αριθμό των ελαχιστόρων και των primes. Παρατηρούμε επίσης ότι ο πίνακας A μπορεί να αναπαρασταθεί σε μορφή γράφου, όπου οι κορυφές αντιστοιχούν στους ελαχιστόρους και οι ακμές στους prime implicants και άρα το covering πρόβλημα ανάγεται σε πρόβλημα κάλυψης των ακμών του γράφου.

Η ελαχιστοποίηση με exact αλγορίθμους επιλύεται υπολογίζοντας αρχικά τον πίνακα των prime implicants και λύνοντας ακολούθως το covering πρόβλημα που προέκυψε. Η δυσκολία σε αυτή την προσέγγιση βρίσκεται στην intractability του covering προβλήματος και στο μέγεθος του πίνακα των implicants. Για παράδειγμα, μια λογική συνάρτηση ν εισόδων και μιας εξόδου μπορεί να έχει $3^{\nu}/\nu$ prime implicants και 2^{ν} ελαχιστόρους [9]. Δηλαδή, έχουμε έναν εκθετικό αλγόριθμο σε ένα πρόβλημα εκθετικού μεγέθους, το οποίο είναι πιθανό να απαιτεί απαγορευτικό μέγεθος μνήμης και υπολογιστικού χρόνου.

Ο πίνακας των implicants μπορεί να μειωθεί με τεχνικές παρόμοιες με αυτές με χρησιμοποιούνται γενικά στα covering προβλήματα. Συγκεκριμένα, κάθε ουσιώδης στήλη αντιστοιχεί σε ένα essential prime, που πρέπει να αποτελεί κομμάτι κάθε λύσης. Επίσης μπορεί να χρησιμοποιηθεί η τεχνική του row/column dominance για την απαλοιφή γραμμών ή στηλών αντίστοιχα και την μειώσει έτσι του A. Η εξαγωγή των essential implicants μαζί με την απαλοιφή γραμμών και στηλών μπορούν να επαναλαμβάνονται για να δώσουν ένα μειωμένο prime implicant πίνακα. Εάν ο πίνακας αυτός είναι κενός, τότε έχει κατασκευαστεί ένα ελάχιστο cover έχοντας αφαιρέσει τους essentials. Εάν δεν είναι κενός ο πίνακας τότε ονομάζεται *κυκλικός*. Η μεθοδολογία για την επίλυση τότε που δόθηκε από τον McCluskey καταλήγει σε Branching αλγορίθμους, για παράδειγμα επιλογή των διαφορετικών συνδυασμών των

στηλών (primes) και αποτίμηση της επόμενης κίνησης με μία συνάρτηση κόστους. Αν και η επιλογή μιας στήλης μπορεί να οδηγήσει σε απλοποίηση, η διαδικασία είναι εκθετικά ανάλογη με το μέγεθος του μειωμένου πίνακα implicant στην χειρότερη περίπτωση.

Μια άλλη προσέγγιση για την επίλυση του προβλήματος είναι αυτή της **μεθόδου Petrick[2][10]**, που περιλαμβάνει την καταγραφή των ελαχιστόρων του πίνακα implicant σε μορφή POS. Η μορφή POS μετατρέπεται μετά σε μορφή SOP κάνοντας τους πολλαπλασιασμούς. Η SOP έκφραση που προκύπτει ικανοποιείται όταν οποιοσδήποτε από τους όρους είναι αληθής. Σε αυτή την περίπτωση, οι όροι αντιστοιχούν στους primes που έχουν επιλεγεί. Το κόστος του cover σχετίζεται έτσι με τον αριθμό των literals στο γινόμενο. Ως αποτέλεσμα, ένα ελάχιστο cover προσδιορίζεται από οποιοδήποτε γινόμενο του SOP έχει τα λιγότερα literals.

Αν και η μετατροπή από POS σε SOP είναι απλή, η μετατροπή του POS σε SOP είναι δυσκολότερη από ότι φαίνεται αρχικά, καθώς οι πολλαπλασιασμοί κρύβουν εκθετικό αριθμό πράξεων. Το γεγονός αυτό καθιστά τον αλγόριθμο του Petrick απαγορευτικό ακόμα και για πίνακα μεσαίου μεγέθους.

Αν και οι exact αλγόριθμοι δύο επιπέδων είναι γνωστοί εδώ και πολλά χρόνια, έχουν γίνει πολλές προσπάθειες για την βελτίωση τους και ιδιαίτερα της διαδικασίας του Quine-McCluskey ώστε να έχουν εφαρμογή σε συναρτήσεις που μοντελοποιούν ρεαλιστικά σχεδιαστικά προβλήματα. Το βασικό πρόβλημα της λογικής ελαχιστοποίησης δύο επιπέδων πηγάζει από το μέγεθος του covering προβλήματος και την δυσκολία συνεπώς επίλυσης του.

Μια από τις καλύτερες προσπάθειες για την βελτίωση του Quine-McCluskey έγινε από τους Rudell και Sangiovanni με την υλοποίηση του **Espresso-Exact[5].** Αν και οι αρχές της exact ελαχιστοποίησης στο Espresso-Exact είναι ίδιες με αυτές του Quine-McCluskey, οι αλγόριθμοι που χρησιμοποιούνται είναι διαφορετικοί. Το Espresso-Exact καταφέρνει να ελαχιστοποίηση αποδοτικά 114 από τις 134 συναρτήσεις αναφοράς.

Οι βασικές βελτιώσεις του Espresso-Exact συνοψίζονται στην κατασκευή ενός μικρότερου πίνακα prime implicant μετά την διαδικασία της μείωσης και στην χρήση ενός αποδοτικού αλγορίθμου Branch-and-Bound για covering. Εδώ οι prime implicants χωρίζονται σε τρία σύνολο: ***essentials, partially redundant***

και ***totally redundant***. Totally redundant είναι οι prime implicants που καλύπτονται από τους essentials και το σύνολο αδιαφορίας (Don't Cares-**DC**), ενώ το partially redundant σύνολο καλύπτει του υπόλοιπους και αντιστοιχεί στις στήλες του μειωμένου πίνακα implicant. Οι γραμμές αντιστοιχούν σε σύνολα ελαχιστόρων, αντί για να αντιστοιχούν σε μεμονωμένους ελαχιστόρους όπως στον Quine-McCluskey. Συγκεκριμένα κάθε γραμμή αντιστοιχεί σε όλους τους ελαχιστόρους που καλύπτονται από το την ίδια ομάδα prime implicants.

Μια ακόμη προσέγγιση προτάθηκε από τον Dagenais που βασίζεται στον Quine-McCluskey αλλά δεν κατασκευάζει τον πίνακα των prime implicants. Η υλοποίηση του αλγορίθμου είναι το πρόγραμμα που ονομάστηκε McBoole [11]. Το McBoole παράγει τους prime implicants και τους αποθηκεύει σε 2 λίστες, μια λίστα για τους undecided και μια για τους retained implicants, αποφεύγοντας την χρήση πίνακα, το οποίο αποδείχτηκε πιο αποδοτικό. Εκμεταλλευόμενοι κάποιες ιδιότητες dominance και essential, κάποιοι implicants μεταφέρονται από την undecided στην retained λίστα. Κάθε Implicant στην undecided λίστα έχει ένα μέρος το οποίο δεν μπορεί να γίνει cover από τους implicants στην retained λίστα ή από τις DC συνθήκες. Όταν όλα τα uncovered μέρη είναι κενά, τότε έχει βρεθεί ένα ελάχιστο cover.

Στο McBoole χρησιμοποιείται η τεχνική του branching που αν και είναι από την φύση της εκθετικής πολυπλοκότητας στην χειρότερη περίπτωση, υλοποιείται αποδοτικά από το πρόγραμμα με την μορφή γράφου που αποθηκεύει την σχέση μεταξύ όλων των prime implicants. Το McBoole θεωρείται επιτυχημένο, αν και μπόρεσε να ελαχιστοποιήσει ακριβώς 86/134 από τις συναρτήσεις αναφοράς.

Μια εναλλακτική προσέγγιση [12] προτάθηκε από τον McGeer, η οποία αποστασιοποιείται από τον Quine-McCluskey. Αντί να υπολογίζει όλους τους prime implicants, ο αλγόριθμος παράγει το covering πρόβλημα και υπολογίζει μόνο τους prime implicants που συμμετέχουν στο πρόβλημα, αποφεύγοντας έτσι τον υπολογισμό όλων των prime implicant το οποίο μπορεί να είναι πολύ μεγάλο σε πολυπλοκότητα.

Ο αλγόριθμος βασίζεται στην ιδέα των ***signature cubes*** που αντιστοιχούν σε ομάδες από primes. Συγκεκριμένα, ένας signature cube αντιστοιχεί αποκλειστικά στην ομάδα των primes που καλύπτουν κάθε ελαχιστόρο, αφού

είναι ο μεγαλύτερος κύβος της τομής των αντίστοιχων primes. Το σύνολο των μέγιστων signature cubes, δηλαδή αυτών που δεν περιέχονται μεμονωμένα σε κάποιον άλλο, ορίζουν ένα ελάχιστο canonical cover της λογικής συνάρτησης που ελαχιστοποιείται και αντιπροσωπεύει τον πίνακα των prime implicants. Το ελάχιστο canonical cover μιας λογικής συνάρτησης είναι μοναδικό και μη περιττό(***irredundant***). Συνεπώς η exact ελαχιστοποίηση πρέπει να προσδιορίσει αρχικά ένα ελάχιστο canonical cover, να υπολογίσει τους primes που ανήκουν σε κάθε signature cube και να επιλύσουν το αντίστοιχο covering πρόβλημα που προκύπτει. Η μεθοδολογία αυτή εφαρμόζεται στο Espresso-Signature και έχει αποδειχτεί ότι είναι δύο φορές γρηγορότερο από το Espresso-Exact σε όλες εκτός από τρεις συναρτήσεις αναφοράς.

3.3.4. Ελαχιστοποίηση με heuristic αλγόριθμους

Το βασικό πρόβλημα στην exact ελαχιστοποίηση είναι ο τεράστιος αριθμός prime implicants που προκύπτει κατά την διαδικασία της ελαχιστοποίησης. Για τις περισσότερες συναρτήσεις με περισσότερες από 16 εισόδους για παράδειγμα υπάρχουν πάρα πολλοί primes και ακόμη περισσότεροι ελαχιστόροι. Οι heuristic μέθοδοι αποφεύγουν τον υπολογισμό όλων των prime implicants χρησιμοποιώντας κριτήρια βελτίωσης. Συγκριμένα, σε ένα αρχικό cover εφαρμόζονται επαναληπτικά αλγόριθμοι οι οποίοι επιφέρουν σταδιακή βελτίωση έως ότου ικανοποιηθούν κάποια κριτήρια ώστε να σταματήσει η διαδικασία δίνοντας κάποιο ικανοποιητικό αποτέλεσμα.

Το τίμημα για να μειώσουμε την πολυπλοκότητα εντοπίζεται στο ότι δεν μπορούμε να εγγυηθούμε την ελάχιστη λύση, αν και με προσεκτική επιλογή των αλγορίθμων μπορεί να φτάσουμε σε μια λύση πολύ κοντά στην βέλτιστη σε ένα λογικό χρονικό διάστημα.

Τα πρώτα εργαλεία που χρησιμοποιούσαν heuristic αλγορίθμους υπολόγιζαν όλους τους prime implicants και χρησιμοποιούσαν μετά τους heuristic αλγορίθμους για να βρουν κάποιο cover για το prime implicant table. Τα εργαλεία που χρησιμοποιήθηκαν αργότερα σταμάτησαν να υπολογίζουν όλους τους prime implicants, αποφεύγοντας έτσι το bottleneck του να αποθηκεύουν ένα τόσο μεγάλο μέγεθος πληροφορίας.

Το heuristic minimization μπορεί να θεωρηθεί ως η εφαρμογή κάποιων λογικών τελεστών στο cover, που παρέχεται στο πρόγραμμα μαζί με το DC-set. Αυτό το cover θεωρείται το τελικό αποτέλεσμα της ελαχιστοποίησης όταν οι διαθέσιμοι τελεστές δεν μπορούν πλέον να μειώσουν το μέγεθος του cover και άρα δεν επιτυγχάνεται επιπλέον βελτίωση.

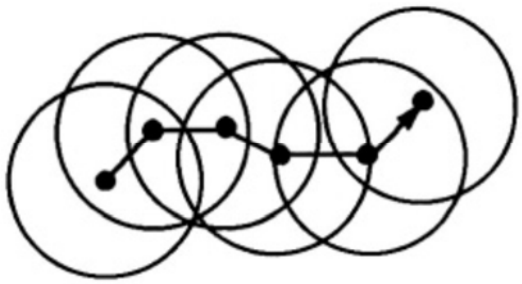
Στα επόμενα κεφάλαια θα ασχοληθούμε ως επί το πλείστον με το εργαλείο Espresso, που είναι το μέχρι τώρα πιο πετυχημένο εργαλείο για την ελαχιστοποίηση λογικών συναρτήσεων με χρήση heuristic μεθόδων. Σε αυτή την ενότητα θα γίνει μια εισαγωγή στο Espresso, άλλα θα αναφερθούμε και σε άλλους heuristic minimizers όπως είναι τα Mini και Presto.

Οι heuristic αλγόριθμοι βασίζονται στην λογική της τοπικής αναζήτησης (Local Search)[2]. Οι αλγόριθμοι που ξεκινούν από μια αρχική λύση και προσπαθούν να βρουν μια καλύτερη εφαρμόζοντας μικρές τροποποιήσεις ονομάζονται local search αλγόριθμοι. Οι αλγόριθμοι local search χρησιμοποιούνται μέσα σε ένα *χώρο αναζήτησης (search space)* που είναι όλες οι δυνατές τιμές που μπορούν να πάρουν οι μεταβλητές του προβλήματος.

Το local search βασίζεται στον ορισμό της *απόστασης* μεταξύ 2 σημείων στο search space. Για παράδειγμα για δύο σύνολα στηλών σε ένα covering πρόβλημα, η απόσταση είναι ο αριθμός των στηλών που εμφανίζονται στο πρώτο σύνολο αλλά δεν εμφανίζονται στο δεύτερο. Προφανώς για διαφορετικά προβλήματα ο ορισμός της απόστασης αλλάζει.

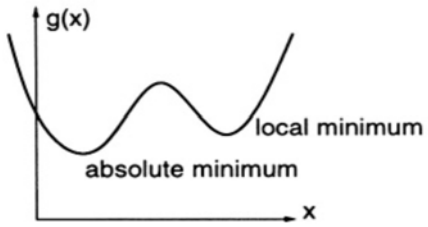
Ένας επιπλέον ορισμός τον οποίο χρειαζόμαστε είναι αυτός της *γειτονιάς* ενός σημείου p ακτίνας r στο search space. Ως γειτονιά ορίζουμε μια ομάδα σημείων του search space, των οποίων η απόσταση από το s είναι λιγότερο από το r.

Ξεκινώντας από μια αρχική λύση, ο αλγόριθμος εξετάζει την γειτονιά για ένα σημείο του οποίου το κόστος είναι μικρότερο από το τρέχον κόστος. Αν βρεθεί σημείο, το θεωρούμε ως νέα αρχική κατάσταση και η διαδικασία συνεχίζεται μέχρι να ικανοποιηθούν κάποια κατάλληλα κριτήρια ώστε να σταματήσει την διαδικασία.



Σχήμα 3.4. Αναπαράσταση του Local Search [2]

Υπάρχουν διαφορετικοί βαθμοί για το πόσο κοντά στη βέλτιστη λύση μπορούμε να φτάσουμε, ανάλογα με το πρόβλημα και τον ορισμό της γειτονιάς. Στο σχήμα 3.5 μπορούμε να δούμε δύο διαφορετικές περιπτώσεις. Στην πρώτη περίπτωση αριστερά έχουμε το ελάχιστο της συνάρτησης, ενώ δεξιά έχουμε ένα τοπικό ελάχιστο.



Σχήμα 3.5. Γραφική απεικόνιση των λύσεων ενός προβλήματος βελτιστοποίησης[2]

Τοπικά βέλτιστη λύση είναι μια λύση στην γειτονιά της οποίας δεν μπορεί να βρεθεί άλλη λύση χαμηλότερου κόστους.

Αυτή την λογική του local search μπορούμε να την εφαρμόσουμε στο πρόβλημα της λογικής ελαχιστοποίησης. Ξεκινάμε από ένα cover μιας συνάρτησης πολλαπλών εξόδων και ως κόστος ορίζεται ο αριθμός των cubes (δηλαδή των όρων γινομένου). Ως γειτονιά θεωρούμε αρχικά το σύνολο των covers που λαμβάνονται από το αρχικό cover προσθέτοντας ένα literal σε ένα από τα cubes. Ένα νέο cover είναι εφικτό στο search space εάν είναι ισοδύναμο με το αρχικό, δηλαδή αναπαριστά την ίδια συνάρτηση. Όταν ένα cube έχει μηδενιστεί στο μέρος των εξόδων, μπορεί να απορριφθεί και άρα συμφώνα με τον ορισμό της γειτονιάς μπορούμε να πάρουμε λύσεις με λιγότερα cube σε σχέση με το αρχικό cover. Αντίθετα λύσεις με περισσότερα cubes σε σχέση με το αρχικό cover δεν ανήκουν στο search space.

Όπως είπαμε heuristic minimization μπορεί να θεωρηθεί η εφαρμογή κάποιων τελεστών σε ένα cover. Οι πιο συχνοί τελεστές[1] είναι οι ακόλουθοι:

i. **Expand:** μετατρέπει ένα cover σε prime και ελάχιστο ως προς ένα μεμονωμένο implicant. Κάθε implicant αντιμετωπίζεται ξεχωριστά. Έτσι κάθε non-prime implicant επεκτείνεται σε prime, δηλαδή αντικαθίσταται από έναν prime implicant που τον περιέχει. Όλοι οι implicants που καλύπτονται από έναν expanded implicant διαγράφονται.

ii. **Reduce:** μετατρέπει ένα cover σε ένα non-prime cover του ίδιου βαθμού. Κάθε implicant αντιμετωπίζεται ξεχωριστά. Το reduce προσπαθεί να αντικαταστήσει κάθε implicant με έναν άλλο που περιέχεται σε αυτόν και τηρεί την συνθήκη, ότι οι reduced implicants μαζί με αυτούς που έχουν απομείνει, εξακολουθούν να αποτελούν cover της συνάρτησης.

iii. **Reshape:** μετατρέπει το cover διατηρώντας τον βαθμό του. Η επεξεργασία των implicant γίνεται σε ζευγάρια. Ένας implicant γίνεται expand, ενώ ο άλλος γίνεται reduced, διατηρώντας την συνθήκη ότι οι reshaped implicants, μαζί με αυτούς που έχουν απομείνει εξακολουθούν να είναι cover της συνάρτησης.

iv. **Irredundant:** μετατρέπει ένα cover σε cover χωρίς περιττούς implicants. Ένα ελάχιστο υποσύνολο implicants επιλέγεται έτσι ώστε να μην υπάρχει ένας μεμονωμένος implicant σε αυτό το υποσύνολο που να καλύπτεται από του υπόλοιπους implicants του υποσυνόλου αυτού.

Στο σχήμα 3.8 βλέπουμε ένα αρχικό cover της λογικής συνάρτησης *f=x'y'z'+x'y+yz*, το οποίο σε μορφή πίνακα αληθείας έχει την παρακάτω μορφή.

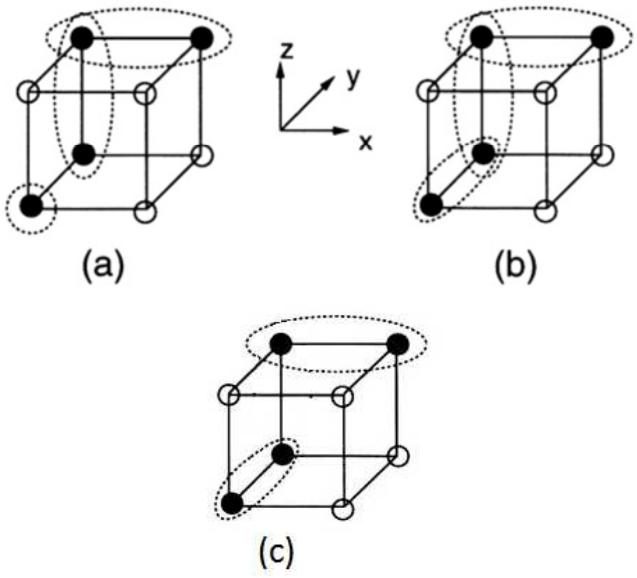
xyz	*f*
000	1
01–	1
–11	1

Σχήμα 3.6. Πίνακας αληθείας της συνάρτησης *f=x'y'z'+x'y+yz*

Αν τώρα αφαιρέσουμε ένα literal από την είσοδο, δηλαδή εφαρμόσουμε τον τελεστή expand στο cube, ουσιαστικά μετατρέπουμε το 000|1 σε 0-0|1 και έχουμε ως αποτέλεσμα το cube του σχήματος 3.8 (b), όπου έχουμε βελτιώσει το cover ως προς ένα literal. Μετά από αυτή την βελτίωση εφαρμόζουμε και τον τελεστή reduce, αφαιρώντας το literal εξόδου 01-|1, αφού μετά το expand έχουμε καλύψει και αυτή την περίπτωση. Ουσιαστικά αφαιρούμε ένα cube από το τελικό cover, παίρνοντας έτσι την βέλτιστη λύση. Διαπιστώνουμε ότι η εφαρμογή των τελεστών οδηγεί σε ένα cover μικρότερου κόστους.

xyz	f
0–0	1
–11	1

Σχήμα 3.7. Εφαρμογή του τελεστή Expand



Σχήμα 3.8. (a) Αρχικό cover συνάρτησης, (b)μετά από εφαρμογή expand σε implicant, (c) μετά την εφαρμογή του reduce, βέλτιστο cover [1]

Μια διάκριση των heuristic minimizers μπορεί να γίνει με βάση του τελεστές που αυτοί χρησιμοποιούν και την σειρά με την οποία τους εφαρμόζουν. Το **Presto** είναι ένα πολύ απλό heuristic minimizer που εφαρμόζει τον τελεστή expand μια φορά. Με αυτή τη μέθοδο μπορούμε να επιτύχουμε ένα prime και ελάχιστο αποτέλεσμα. Όμως η πληθικότητα του τελικού cover μπορεί να είναι πολύ μεγαλύτερη από την ελάχιστη πληθικότητα.

Το MINI είναι ένα πρόγραμμα το οποίο επαναλαμβάνει τους τελεστές expand, reduce και reshape για την αποφυγή λύσεων χαμηλής ποιότητας. Το prime cover που υπολογίζεται από το expand, επεξεργάζεται από τους άλλους δύο τελεστές ώστε να αυξηθούν οι πιθανότητες να μειωθεί σε μέγεθος το νέο cover. Ο αλγόριθμος τερματίζει όταν η επανάληψη των τριών αυτών τελεστών δεν μειώνει άλλο την πληθικότητα του cover. Αυτή η προσέγγιση δεν διασφαλίζει την irredundancy του τελικού cover.

Το Espresso χρησιμοποιεί επιπλέον τον τελεστή *irredundant* για να διασφαλίσει την irredundancy του cover. Ο βασικός αλγόριθμος του Espresso υπολογίζει αρχικά ένα prime και irredundant cover, εφαρμόζοντας τους τελεστές expand και irredundant. Ακολούθως βελτιώνει το cover όποτε είναι δυνατόν, υπολογίζοντας μια ακολουθία από prime και irredundant covers φθίνουσας πληθικότητας. Αυτό επιτυγχάνεται με την επαναληπτική εφαρμογή των reduce, expand και irredundant.

Προφανώς για διαφορετικούς minimizers η υλοποίηση των τελεστών ελαχιστοποίησης μπορεί να είναι διαφορετική. Για παράδειγμα, η σειρά με την οποία ελέγχονται οι implicants κατά την εφαρμογή του expand επηρεάζει το cover που θα προκύψει. Δηλαδή δύο διαφορετικές υλοποιήσεις του expand θα παράγουν cover, τα οποία είναι prime και minimal, αλλά με διαφορετική πληθικότητα της λύσης. Μερικοί αλγόριθμοι που υλοποιούν αυτούς τους τελεστές έχουν πολυπλοκότητα μεγαλύτερη από την πολυωνυμίκη. Από τα heuristic που χρησιμοποιούνται εξαρτώνται και ο χρόνος εκτέλεσης και οι απαιτήσεις μνήμη, που είναι και ο βασικός παράγοντας για να είναι βιώσιμη η εφαρμογή για μεγάλα cover.

4. Exact Αλγόριθμοι

Στο κεφάλαιο αυτό θα ασχοληθούμε εκτενέστερα με τους exact αλγορίθμους που χρησιμοποιούνται και κυρίως με τον αλγόριθμο Quine-McCluskey. Όπως έχουμε ήδη αναφέρει οι exact αλγόριθμοι είναι αυτοί που μας δίνουν την βέλτιστη λύση στο πρόβλημα της ελαχιστοποίησης λογικών συναρτήσεων. Το μεγάλο μειονέκτημα τους όμως είναι ότι έχουν μεγάλες απαιτήσεις σε πόρους του συστήματος (μνήμη) και χρόνο. Εκτός από τον αλγόριθμο Quine-McCluskey υπάρχουν και μερικές ακόμη προσεγγίσεις , όπως η μέθοδος Petrick, Espresso-Exact και Espresso-Signature.

4.1 Αλγόριθμος Quine-McCluskey

Ο αλγόριθμος που προτάθηκε από τους Quine και McCluskey [2][6][7][13] ήταν η πρώτη εξελιγμένη και αυτοματοποιημένη μέθοδος για να ελαχιστοποιήσουμε σε αριθμό πυλών την υλοποίηση δύο επιπέδων ενός συνδυαστικού κυκλώματος. Η μέθοδος ήρθε να αντικαταστήσει τους χάρτες Karnaugh που χρησιμοποιούνταν μέχρι τότε για την ελαχιστοποίηση. Η διαδικασία είναι λειτουργικά ισοδύναμη με τους χάρτες Karnaugh, αλλά η χρήση πίνακα κάνει την διαδικασία ακόμα πιο αποδοτική για τους αλγορίθμους σε Η/Υ και παρέχει επίσης ένα ντετερμινιστικό τρόπο για να ελέγξουμε ότι έχουμε πετύχει βέλτιστη ελαχιστοποιημένη μορφή της συνάρτησης Boole.

Τα δύο βασικά βήματα του αλγορίθμου είναι:

1. Η εύρεση όλων των prime implicants της συνάρτησης.
2. Η χρησιμοποίηση αυτών των prime implicants σε ένα πίνακα prime implicant για εύρεση των essential prime implicants τη σ συνάρτησης, αλλά και των υπολοίπων απαραίτητων για να κατασκευάσουμε ένα cover της συνάρτησης.

Το βασικό μειονέκτημα που αντιμετωπίζουμε με τον αλγόριθμο Quine-McCluskey είναι ότι δεν μπορεί αν εφαρμοστεί σε μεγάλα κυκλώματα με επιτυχία. Το πρόβλημα της ελαχιστοποίησης που έχει να επιλύσει ο Quine-McCluskey θεωρείται NP-hard σε υπολογιστική πολυπλοκότητα (και θεωρούνται τα δυσκολότερα από τα NP προβλήματα). Αυτό σημαίνει ότι ο χρόνος τρεξίματος του αλγορίθμου αυξάνεται εκθετικά ανάλογα με το αριθμό

των μεταβλητών. Δηλαδή για ν μεταβλητές τα άνω όριο για τον αριθμό των prime implicants είναι $3^ν/ν$, το οποίο σημαίνει ότι για ν = 32 για παράδειγμα μπορεί να υπάρχουν πάνω από $5,7*10^{13}$ implicants. Εξαιτίας αυτής της υπολογιστικής έκρηξης ο αλγόριθμος περιορίζεται στην αντιμετώπιση προβλημάτων με 11-15 μεταβλητές ανάλογα με την υλοποίηση και την εκμετάλλευση διάφορων μαθηματικών απλοποιήσεων. Αυτός είναι και ο λόγος που καταφεύγουμε σε heuristic αλγορίθμους που μπορεί να μην είναι βέλτιστοι, αλλά καταφέρνουν να δώσουν μια πολύ καλή προσέγγιση της ελάχιστης λύσης.

Στο υπόλοιπο του κεφαλαίου θα επεκταθούμε στους επιμέρους αλγορίθμους του Quine-McCluskey που χρησιμοποιήθηκαν για την υλοποίηση.

4.2. Υλοποίηση Quine McCluskey σε βήματα

Τα βασικά μέρη για την υλοποίηση του Quine-McCluskey είναι:

- Παραγωγή των Prime Implicants
- Κατασκευή του πίνακα Prime Implicants
- Ελαχιστοποίηση του πίνακα
    - Αφαίρεση των Essential Prime Implicants
    - Εφαρμογή κανόνων κυριαρχίας γραμμής (Row Dominance)
    - Εφαρμογή κανόνων κυριαρχίας στήλης (Column Dominance)
- Επίλυση πίνακα Prime Implicants

4.2.1. Παραγωγή Prime Implicants

Ως είσοδος χρησιμοποιείται ένα αρχείο blif (Berkeley Logic Interchange Format)[16], που είναι μια γλώσσα περιγραφής κυκλωμάτων από το Berkeley, και στην οποία περιγράφεται η λογική συνάρτηση. Ουσιαστικά περιγράφεται μια λίστα όλων των ελαχιστόρων της συνάρτησης για τους οποίους η συνάρτηση F αποτιμάται σε 1, όπως και των ελαχιστόρων για τους οποίους η συνάρτηση F μας δίνει συνθήκη αδιαφορίας. Δηλαδή απεικονίζεται το ON-set όπως και το DC-set.

Οι ελαχιστόροι είναι στην δυαδική τους αναπαράσταση, όπου το 0 αντιστοιχεί στο συμπλήρωμα της μεταβλητής, το 1 στο ορθό της μεταβλητής και με '–' απεικονίζεται το DC ή η απουσία μεταβλητής. Ακολούθως γίνεται αποτίμηση του αριθμού των άσσων που περιέχει καθένας από τους

ελαχιστόρους. Οι ελαχιστόροι κατατάσσονται σε αύξουσα σειρά ανάλογα με τον αριθμό άσσων που περιέχουν και χωρίζονται σε ομάδες. Στην ίδια ομάδα ανήκουν ελαχιστόροι με τον ίδιο αριθμό άσσων.

Το βήμα που ακολουθεί είναι να δημιουργήσουμε ζευγάρια των implicants που διαφέρουν κατά μια μεταβλητή. Γι' αυτό το λόγο συγκρίνουμε κάθε ελαχιστόρο με ν αριθμό άσσων με όλους τους ελαχιστόρους που ανήκουν στην αμέσως επόμενη ομάδα και άρα έχουν ν+1 όρους στην δυαδική αναπαράσταση τους. Αν οι δύο ελαχιστόροι που συγκρίνονται διαφέρουν μόνο σε μία μεταβλητή σημειώνονται ως επιλεγμένοι *(checked)* , τότε δημιουργούμε ένα νέο ελαχιστόρο ο οποίος στη θέση που διαφέρουν οι δύο ¨γονείς¨ του δεν έχει τιμή. Δηλαδή, αφαιρούμε μια μεταβλητή και ο νέος ελαχιστόρος που δημιουργείται εξαρτάται από μια μεταβλητή λιγότερη. Επιπλέον, όλοι οι ελαχιστόροι που δημιουργήθηκαν κατά την σύγκριση μεταξύ των ελαχιστόρων δύο ομάδων που διαφέρουν κατά ένα άσσο, ανήκουν σε μια νέα ομάδα. Φυσικά δεν έχει νόημα να συγκρίνουμε ελαχιστόρους ομάδων που διαφέρουν κατά δύο θέσεις αφού εξ ορισμού δεν θα έχουν διαφορά σε μια μόνο μεταβλητή.

Μόλις ολοκληρωθεί η διαδικασία σύγκρισης μεταξύ των όλων των ν και ν+1 ομάδων και δημιουργίας νέων ελαχιστόρων, συνεχίζουμε με μια παρόμοια διαδικασία, που αποτελεί το επόμενο επίπεδο εξάλειψης μεταβλητών. Στο επίπεδο αυτό ασχολούμαστε με τους νέους ελαχιστόρους που δημιουργήθηκαν από το προηγούμενο βήμα. Ομοίως με πριν συνεχίζουμε τις συγκρίσεις μεταξύ των ελαχιστόρων μεταξύ των ν και ν+1 ομάδων, αλλά τώρα πρέπει να λάβουμε υπόψιν και τις μεταβλητές που έχουμε εξαλείψει, με την έννοια ότι οι δύο ελαχιστόροι μπορούν να συνδυαστούν μόνο αν απουσιάζει η ίδια μεταβλητή (έχουν «παύλα» στην ίδια μεταβλητή) και διαφέρουν κατά μια θέση από τις υπόλοιπες. Οι ελαχιστόροι που μπορούν να συνδυαστούν σημειώνονται και πάλι ως checked.

Η διαδικασία αυτή συνεχίζεται μέχρι να μην μπορούν να δημιουργηθούν νέοι συνδυασμοί από ελαχιστόρους που να διαφέρουν κατά μια θέση. Τότε όλοι οι ελαχιστόροι που έχουν απομείνει από όλη την διαδικασία χωρίς να έχουν συνδυαστεί με κάποιον άλλο(δεν έχουν check) θεωρούνται prime implicants.

4.2.2. Κατασκευή πίνακα Prime Implicants

Τώρα μπορούμε να κατασκευάσουμε τον πίνακα των prime implicants, τον οποίο και θα χρησιμοποιήσουμε για να εφαρμόσουμε τεχνικές row dominance και column dominance για την απλοποίηση του πίνακα. Οι γραμμές του πίνακα των prime implicants είναι όλα τα terms που καλύπτουν το ON-set της συνάρτησης (χωρίς το DC-set). Ενώ στις στήλες τοποθετούμε όλους τους prime implicants που δημιουργήθηκαν από την διαδικασία και δεν έχουν γίνει check.. Το ¨X¨ στον πίνακα στην γραμμή i και στήλη j δείχνει ότι ο prime implicanttης στήλης j καλύπτει τον αντίστοιχο ελαχιστόρο της γραμμής i. Έτσι τώρα δεδομένου του πίνακα prime implicants, πρέπει να επιλέξουμε ένα ελάχιστο σύνολο από primes, τέτοιο ώστε να καλύπτονται όλοι οι ελαχιστόροι του ON-set, δηλαδή να υπάρχει ένα X σε κάθε γραμμή, το οποίο είναι ένα κλασσικό πρόβλημα ελάχιστου καλύμματος.

4.2.3. Ελαχιστοποίηση πίνακα Prime Implicants

Αφού έχει δημιουργηθεί ο πίνακας των prime implicants, το βήμα που έπεται είναι να ελαχιστοποιήσουμε όσο γίνεται τον πίνακα, ώστε να είναι ευκολότερη η επίλυση του. Το βήμα αυτό αποτελείται ουσιαστικά από 3 μέρη, εύρεση essential prime implicants, εφαρμογή κανόνων row dominance και εφαρμογή κανόνων column dominance. Τα μέρη αυτά εφαρμόζονται μέχρι να μην μπορεί να γίνει άλλη ελαχιστοποίηση.

Essential Prime Implicants

Όταν μια γραμμή έχει ένα και μοναδικό X, τότε σημαίνει ότι ο αντίστοιχος prime implicant που καλύπτει τον ελαχιστόρο είναι essential. Αυτό σημαίνει ότι με άλλα λόγια ότι υπάρχει μόνο ένας prime implicant που να καλύπτει τον ελαχιστόρο του ON-set της συνάρτησης. Άρα οποιαδήποτε εφικτή υλοποίηση-λύση θα πρέπει να περιέχει οπωσδήποτε το συγκεκριμένο essential prime implicant. Στην υλοποίηση του αλγορίθμου, αποθηκεύουμε τους essentials ως μέρος της τελικής λύσης και αφαιρούμε από τον πίνακα όλες τους ελαχιστόρους που καλύπτονται (έχουν X στην αντίστοιχη γραμμή) από τους essentials.

Στο σχήμα 4.1 μπορούμε να δούμε ένα παράδειγμα ενός πίνακα prime implicant, όπου έχουμε δυο essential prime implicants. Υπάρχουν 5 prime implicants που καλύπτουν ο καθένας 2 ελαχιστόρους. Οι A'C'D' και ACD είναι

essential prime implicants και ανήκουν στην τελική λύση. Επιπλέον, αφαιρούμε από τον πίνακα όλες τις γραμμές οι οποίες διασταυρώνονται με την στήλη του essential prime implicant, καθώς οι αντίστοιχοι ελαχιστόροι καλύπτονται από τον ίδιο essential. Στο παράδειγμα, οι ελαχιστόροι 0,4,11,15 καλύπτονται από τους essentials.

	$A'C'D'$ (0,4)	$A'BC'$ (4,5)	$BC'D$ (5,13)	ABD (13,15)	ACD (11,15)
0	X				
4	X	X			
5		X	X		
11					X
13			X	X	
15				X	X

Σχήμα 4.1. Πίνακας Prime Implicants

Row Dominance

Εάν ο πίνακας δεν έχει essentials μπορούμε να εκμεταλλευτούμε την ιδιότητα του row dominance[2][14][15] για περαιτέρω απλοποίηση του πίνακα. Μια γραμμή U ενός πίνακα prime implicants λέμε ότι ***κυριαρχεί*** μιας γραμμής V, εάν η U περιέχει τουλάχιστον όλες τις στήλες που περιέχει η V.

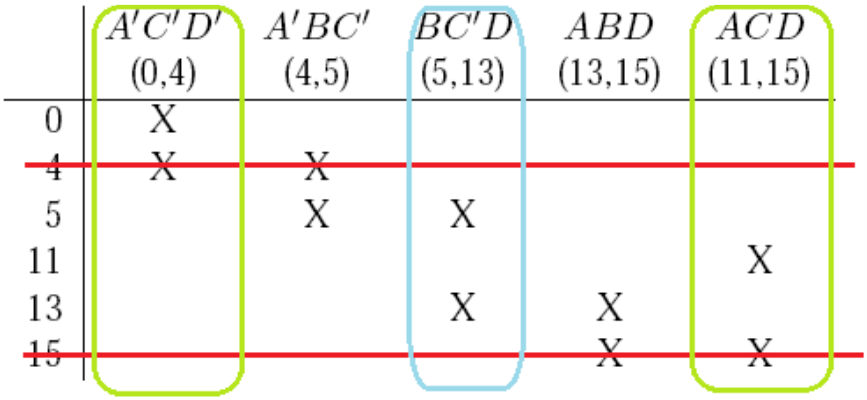
Για παράδειγμα έστω ο πίνακας τους σχήματος 4.2. Παρατηρούμε ότι η γραμμή 3 περιέχει όλους τους implicants που περιέχει και η γραμμή 2, όπως και όλους τους implicants που περιέχει και η γραμμή 7. Δηλαδή, η γραμμή 3 ***κυριαρχεί*** της γραμμής 2 και της γραμμής 7. Μπορούμε ως συνέπεια της παραπάνω παρατήρησης να αγνοήσουμε την γραμμή 3 καθώς θα καλύπτεται πάντα αρκεί να καλύψουμε τις σειρές 2 και 7. *Άρα σε περίπτωση row dominance,* ***μπορούμε να διαγράψουμε την γραμμή που κυριαρχεί των άλλων***. Στο παράδειγμα, εκτός από την γραμμή 3 μπορούμε να διαγράψουμε και την γραμμή 1 που κυριαρχεί της γραμμής 5.

	$A'B'$ (1,2,3)	$C'D$ (1,5)	$A'D$ (1,3,5,7)	$A'C$ (2,3,7)
1	X	X	X	
2	X			X
3	X		X	X
5		X	X	
7			X	X

Σχήμα 4.2. Row Dominance

Column Dominance

Μια στήλη U ενός πίνακα prime implicants λέμε ότι ***κυριαρχεί[2]*** μιας στήλης V, εάν η U περιέχει τουλάχιστον όλες τις γραμμές που περιέχει η V. ***Μπορούμε να διαγράψουμε τις κυριαρχούμενες στήλες***, αφού το να επιλέξουμε την στήλη που κυριαρχεί θα έχει ως αποτέλεσμα να καλύψουμε περισσότερες γραμμές σε σχέση με μια κυριαρχούμενη στήλη.



Σχήμα 4.3. Column Dominance

Στο σχήμα 4.3 διαπιστώνουμε ότι μετά την εύρεση του essential prime implicant και την αφαίρεση των ελαχιστόρων 4 και 15 που καλύπτονται, η στήλη BC'D κυριαρχεί της A'BC' και της ABD. Οι κυριαρχούμενες στήλες διαγράφονται και απομένει η BC'D.

4.2.4. Επίλυση πίνακα prime implicants

Αφού εφαρμόσουμε τους κανόνες ελαχιστοποίησης του πίνακα των prime implicants, ψάχνουμε για νέους essential prime implicants που δημιουργήθηκαν μετά τις τροποποιήσεις του πίνακα. Οι essentials αυτοί ονομάζονται δευτερεύοντες essential prime implicants και μπορούν και αυτοί να αφαιρεθούν από τον πίνακα και προστεθούν στο τελικό cover, καθώς αποτελούν εγγυημένα μέρος οποιασδήποτε λύσης του covering προβλήματος της συνάρτησης.

Πολλές φορές ακόμα και μετά την εφαρμογή των κανόνων ελαχιστοποίησης και την εύρεση των δευτερευόντων essentials παραμένει πληροφορία στον πίνακα, την οποία και δεν μπορούμε να χειριστούμε με τους κανόνες row/column dominance. Το πρόβλημα αυτό ονομάζεται κυκλικό πρόβλημα covering. Λύση τότε επιτυγχάνεται με τρεις μεθόδους:

i. Μέθοδος Petrick
ii. Μέθοδος Branching
iii. Ευριστική (Heuristic) μέθοδος

Μέθοδος Petrick

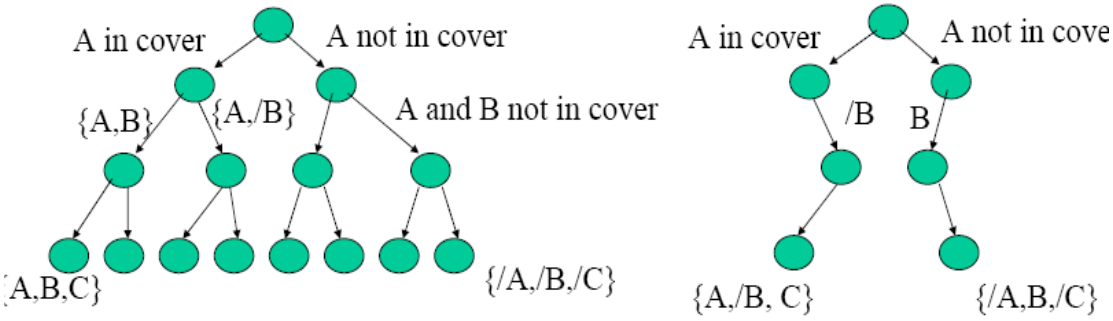
Η μέθοδος Petrick είναι μια τεχνική για τον προσδιορισμό όλων των ελάχιστων SOP λύσεων για πίνακες prime implicants. Η μέθοδος είναι κατάλληλη για υλοποίηση σε Η/Υ. Τα βήματα που ακολουθούμε, αφού έχουμε εφαρμόσει στον πίνακα prime implicants τις μεθόδους ελαχιστοποίησης, είναι:

1. Ορίζουμε ως Pi (P_1,P_2,P_3 κτλ.), την λογική προϋπόθεση (αληθής/ψευδής) να είναι ο prime implicant p_i στο τελικό cover. Pi = 0, ο p_i δεν ανήκει στο τελικό cover. Pi = 1, ο p_i ανήκει στο τελικό cover
2. Ορίζουμε μια λογική συνάρτηση P, η οποία είναι αληθής όταν καλύπτονται όλες οι στήλες. Κάθε όρος του P είναι OR πολλών πιθανών στηλών που καλούνται να καλύψουν κάθε ελαχιστόρο. Το AND όλων των όρων είναι η P.
3. Ελαχιστοποιούμε το P σε ένα SOP εφαρμόζοντας τους πολλαπλασιασμούς και την ιδιότητα X+XY=X
4. Κάθε όρος του αποτελέσματος αναπαριστά μια λύση του P, δηλαδή ένα σύνολο από implicants που καλύπτουν όλες τις γραμμές. Για τον προσδιορισμό της ελάχιστης λύσης ψάχνουμε τους όρους με τον μικρότερο αριθμό prime implicants (P_i).
5. Για κάθε όρο από την προηγούμενη επιλογή, μετράμε τον αριθμό των literals σε κάθε prime implicant και βρίσκουμε τον συνολικό αριθμό
6. Επιλέγουμε τον όρο με τα λιγότερα literals και παίρνουμε ως λύση το αντίστοιχο άθροισμα των prime implicants.

Μέθοδος Branching

Μια άλλη προσέγγιση όταν το πρόβλημα είναι κυκλικό δίνεται από την μέθοδο του branching[2]. Στην τεχνική αυτή επιλέγουμε μια στήλη/prime implicant και έτσι χωρίζουμε το πρόβλημα σε δύο υποπροβλήματα. Στο ένα υποπρόβλημα δεχόμαστε τον prime ως μέρος της τελικής λύσης, ενώ στο άλλο απορρίπτουμε τον prime.

Τα υποπροβλήματα που προκύπτουν μπορεί με την σειρά τους να είναι κυκλικά και άρα να χρειαστεί να επιλέξουμε ένα νέο implicant για τον χωρισμό εκ νέου σε υποπροβλήματα. Αυτό ουσιαστικά καταλήγει σε ένα αναδρομικό πρόβλημα, στο οποίο μπορούμε να δώσουμε λύση μόλις βρούμε απάντηση σε κάποιο από τα υποπροβλήματα. Σε αυτού του είδους τα προβλήματα πρέπει να απαριθμήσουμε όλες τις λύσεις ώστε να φτάσουμε στην λύση με το ελάχιστο κόστος. Τεχνικές βελτιστοποίησης της λύσης χρησιμοποιούνται για την αποφυγή της εκθετικής πολυπλοκότητας, όπως ¨κλάδεμα¨ των προβλημάτων με χρήση μεθόδων εύρεσης του χαμηλότερου ορίου κόστους



Σχήμα 4.4. Κλάδεμα δέντρου branch-and-bound

Heuristic Μέθοδος

Εφαρμόζουμε μια greedy προσέγγιση στην προσπάθεια να βρούμε implicants που καλύπτουν όσο το δυνατόν περισσότερους ελαχιστόρους. Η διαδικασία σε βήματα:

1. Ψάχνουμε στον πίνακα των prime implicants rowwise για να βρούμε τον πρώτο prime που καλύπτει την πρώτη σειρά. Υποθέτουμε ότι ο prime implicant αυτός ανήκει στο τελικό cover. Τον αφαιρούμε από τον πίνακα διαγράφοντας επίσης και τις γραμμές που καλύπτονται και προσθέτω τον implicant στην τελική λύση.
2. Ακολούθως συνεχίζουμε την διαδικασία με μια μικρή διαφορά. Μόλις βρούμε κάποιο implicant, μετράμε τον αριθμό των γραμμών που αυτός καλύπτει και αν είναι μεγαλύτερος από τον αριθμό των γραμμών που καλύπτει ο επόμενος implicant που υπάρχει στην ίδια γραμμή, τον επιλέγουμε για το τελικό cover και αφαιρούμε από τον

πίνακα τις γραμμές που καλύπτει. Σε περίπτωση ισοπαλίας διαλέγουμε τον πρώτο που εξετάσαμε.

4.2.4. Παράδειγμα αλγορίθμου Quine-McCluskey

Έστω η λογική συνάρτηση F(A, B, C, D)=∑m(2, 3, 7, 9, 11, 13)+ ∑d(1, 10, 15). Αρχικά τοποθετούμε τους ελαχιστόρους ομάδες ανάλογα με τον αριθμό των άσσων στην δυαδική αναπαράσταση του καθενός, σχήμα 5.4.α. Στο 5.4.β βλέπουμε την διαδικασία σχηματισμού ζευγαριών ελαχιστόρων που διαφέρουν κατά μια μεταβλητή μόνο (το οποίο σημειώνεται με παύλα). Κάθε ελαχιστόρος που έχει συνδυαστεί σε ζευγάρι σημειώνεται. Η διαδικασία συνεχίζεται μέχρι να μην μπορούν να δημιουργηθούν νέα ζευγάρια (σχήμα 4.4.γ).

Οι implicants που δεν έχουν σχηματίσει ομάδα και άρα δεν είναι checked είναι primes και θα χρησιμοποιηθούν σε ένα πίνακα prime implicants, όπως αυτός του σχήματος 4.5. Οι διπλοεγγραφές αφαιρούνται για ευκολία

α)

Column I	
0	0000
2	0010
8	1000
5	0101
6	0110
10	1010
12	1100
7	0111
13	1101
14	1110
15	1111

β)

Column I			*Column II*	
0	0000	√	(0,2)	00-0
2	0010	√	(0,8)	-000
8	1000	√	(2,6)	0-10
5	0101	√	(2,10)	-010
6	0110	√	(8,10)	10-0
10	1010	√	(8,12)	1-00
12	1100	√	(5,7)	01-1
7	0111	√	(5,13)	-101
13	1101	√	(6,7)	011-
14	1110	√	(6,14)	-110
15	1111	√	(10,14)	1-10
			(12,13)	110-
			(12,14)	11-0
			(7,15)	-111
			(13,15)	11-1
			(14,15)	111-

γ)

Column I			*Column II*			*Column III*	
0	0000	√	(0,2)	00-0	√	(0,2,8,10)	-0-0
2	0010	√	(0,8)	-000	√	(0,8,2,10)	-0-0
8	1000	√	(2,6)	0-10	√	(2,6,10,14)	–10
5	0101	√	(2,10)	-010	√	(2,10,6,14)	–10
6	0110	√	(8,10)	10-0	√	(8,10,12,14)	1–0
10	1010	√	(8,12)	1-00	√	(8,12,10,14)	1–0
12	1100	√	(5,7)	01-1	√	(5,7,13,15)	-1-1
7	0111	√	(5,13)	-101	√	(5,13,7,15)	-1-1
13	1101	√	(6,7)	011-	√	(6,7,14,15)	-11-
14	1110	√	(6,14)	-110	√	(6,14,7,15)	-11-
15	1111	√	(10,14)	1-10	√	(12,13,14,15)	11–
			(12,13)	110-	√	(12,14,13,15)	11–
			(12,14)	11-0	√		
			(7,15)	-111	√		
			(13,15)	11-1	√		
			(14,15)	111-	√		

Σχήμα 4.5. α)Ομαδοποίηση σύμφωνα με αριθμό άσσων, β)-γ) σχηματισμός ζευγαριών,

	$B'D'$ (0,2,8,10)	CD' (2,6,10,14)	BD (5,7,13,15)	BC (6,7,14,15)	AD' (8,10,12,14)	AB (12,13,14,15)
0	X					
2	X	X				
5			X			
6		X		X		
7			X	X		
8	X				X	
10	X	X			X	
12					X	X
13			X			X
14		X		X	X	X
15			X	X		X

Σχήμα4.6. Πίνακας prime implicants

Στην κατασκευή του πίνακα δεν συμμετέχουν οι ελαχιστόροι του DC-set, αφού δεν είμαστε υποχρεωμένοι να τους καλύψουμε. Από τον πίνακα αφαιρούμε τους essential prime implicants B'D' και BD σχήμα 4.6.α και τους προσθέτουμε στο τελικό cover. Επιπλέον, αφαιρούμε και τις γραμμές που καλύπτονται από τους essentials.

	$B'D'$ (0,2,8,10)	CD' (2,6,10,14)	BD (5,7,13,15)	BC (6,7,14,15)	AD' (8,10,12,14)	AB (12,13,14,15)
0	X					
2	X	X				
5			X			
6		X		X		
7			X	X		
8	X				X	
10	X	X			X	
12					X	X
13			X			X
14		X		X	X	X
15			X	X		X

Σχήμα 4.7. Essential primes

Στη συνέχεια βλέπουμε ότι ο ελαχιστόρος 14 κυριαρχεί των άλλων δυο ελαχιστόρων. Η γραμμή που αντιπροσωπεύει τον ελαχιστόρο 14 διαγράφεται, επειδή αν κάποιος implicant καλύπτει τον ελαχιστόρο 6, τότε καλύπτει εγγυημένα και τον 14. Το ίδιο ισχύει και για τον ελαχιστόρο 12.

	CD' (2,6,10,14)	BC (6,7,14,15)	AD' (8,10,12,14)	AB (12,13,14,15)
6	X	X		
12			X	X
14	X	X	X	X

Σχήμα 4.8. Εφαρμογή κανόνα row domination

Το επόμενο βήμα είναι να εφαρμόσουμε και κανόνες column domination. Στο σχήμα 4.9 παρατηρούμε ότι η στήλη του CD' κυριαρχεί την BC και vice versa. Το ίδιο ισχύει και με τις AD' και AB. Η στήλη που κυριαρχείται όπως έχουμε πει διαγράφεται και άρα στην δικιά μας περίπτωση μπορούμε να διαγράψουμε τις στήλες BC και AB(αν και άλλοι συνδυασμοί είναι εφικτοί).

	CD' (2,6,10,14)	BC (6,7,14,15)	AD' (8,10,12,14)	AB (12,13,14,15)
6	X	X		
12			X	X

Σχήμα 4.9. Εφαρμογή κανόνα column domination

Η διαδικασία συνεχίζει επαναλαμβάνοντας τα βήματα της ελαχιστοποίηση του πίνακα μέχρι να μην μπορεί να γίνει άλλη βελτίωση ή να έχει βρεθεί μια λύση στο covering πρόβλημα. Στο παράδειγμα μας συνεχίζουμε ελέγχοντας για τυχόν ύπαρξη δευτερευόντων essentials μετά την διαδικασία της ελαχιστοποίησης, που θα είναι και αυτοί μέρος οποιασδήποτε λύσης. Στο σχήμα 4.10 στον πίνακα έχουν απομείνει τα CD' και AD' που είναι δευτερεύοντες essential prime implicants.

	CD' (2,6,10,14)	AD' (8,10,12,14)
6	X	
12		X

Σχήμα 4.10. Δευτερεύοντες essential prime implicants

Αφαιρώντας και τους δευτερεύοντες essential, ο πίνακας έχει επιλυθεί και το αποτέλεσμα είναι το **F = B'D'+BD+CD'+AD'**.

4.2.5 Quine-McCluskey για συναρτήσεις πολλαπλών εξόδων

Ένα βασικό πλεονέκτημα της μεθόδου Quine-McCluskey είναι ότι μπορεί να επεκταθεί για συναρτήσεις με πολλαπλές εξόδους[2][14], $f = \{f_1,\ldots,f_m\}$. Κάθε όρος του f έχει ένα κομμάτι input και ένα κομμάτι output. Το output κομμάτι δείχνει σε ποια έξοδο, f_j, αντιστοιχεί ένας όρος του ON-set. Εάν ο όρος αντιστοιχεί στην j^{th} έξοδο τότε το output κομμάτι έχει έναν άσσο. Για παράδειγμα, η συνάρτηση δύο εξόδων *f = {f1,f2}*, όπου *f1 = AND(x1,x2)* και *f2 = AND(x2,x3)* μπορεί να αναπαρασταθεί ως εξής: **11- 10**

-11 01

Ισοδύναμα μπορεί να αναπαρασταθεί ως:

110	**10**	***ή***	**011 01**
111	**10**		**110 10**
011	**01**		**111 11**
111	**01**		

Η επίλυση ενός τέτοιου συστήματος μπορεί να γίνει είτε αντιμετωπίζοντας την κάθε έξοδο ξεχωριστά είτε επεκτείνοντας την διαδικασία που ακολουθήσαμε έως τώρα και αντιμετωπίζοντας τις πολλές εξόδους ως μια οντότητα, το οποίο είναι λιγότερο ακριβή διαδικασία.

Οι prime implicants παράγονται και σε αυτή την διαδικασία με τρόπο παρόμοιο με την διαδικασία για τις συναρτήσεις μιας εξόδου. Δηλαδή, οι ελαχιστόροι κατατάσσονται σε αύξουσα σειρά ανάλογα με τον αριθμό των άσσων που περιέχουν στην δυαδική αναπαράσταση τους. Στη συνέχεια προσπαθούμε να συνδυάσουμε ελαχιστόρους που διαφέρουν σε μια θέση σε ένα νέο πιο γενικό ελαχιστόρο. Η διαφορά σε σχέση με τις συναρτήσεις μιας εξόδου είναι ότι τώρα πρέπει να λαμβάνουμε υπόψιν και τις εξόδους. Για να έχει νόημα ο νέος ελαχιστόρος θα πρέπει η τομή της αναπαράσταση των εξόδων των ελαχιστόρων που συνδυάζονται να μην είναι το κενό, δηλαδή να έχούν τουλάχιστον μια έξοδο κοινή. Άρα ο νέος ελαχιστόρος θα έχει '-' στη θέση που διαφέρουν οι δύο ελαχιστόροι που συνδυάζονται και έξοδο που ισούται με το bitwise-AND των εξόδων τους. Εάν η τομή των εξόδων είναι ίδια με οποιοδήποτε από τις εξόδους των αρχικών ελαχιστόρων τότε ο ελαχιστόρος αυτός θεωρείται ότι έχει ελεγχθεί και αφαιρείται από περαιτέρω επεξεργασία. Η διαδικασία επεκτείνεται και για την παραγωγή k-ελαχιστόρων από (k-1)-ελαχιστόρους.

Εφόσον δημιουργηθούν όλοι οι prime implicants συνεχίζουμε έχοντας δημιουργήσει ένα covering πρόβλημα. Τοποθετούμε τους prime implicants στον πίνακα των prime implicants και προσπαθούμε να βρούμε τον ελάχιστο αριθμό prime implicants που καλύπτουν όλους τους ελαχιστόρους της συνάρτησης (χωρίς να περιλαμβάνονται σε αυτούς το DC-set). Το πρόβλημα αυτό μπορεί να επιλυθεί με τις τεχνικές που αναλύσαμε για την περίπτωση μιας εξόδου, στις υποπαραγράφους 4.2.2 και 4.2.3.

Στο σχήμα 4.11. μπορούμε να δούμε ένα παράδειγμα εφαρμογής του αλγορίθμου Quine-McCluskey σε συνάρτηση πολλαπλών εξόδων. Η αλλαγή που παρατηρούμε σε σχέση με την τις συναρτήσεις μιας εξόδου είναι ότι κατά την εφαρμογή του αλγορίθμου χρειαζόμαστε και πληροφορία για τις εξόδους, που συμμετέχουν στην διαδικασία. Για παράδειγμα, ενώ στη δεύτερη στήλη ο implicant ***010- : 0110*** μπορεί να συνδυαστεί με τον ελαχιστόρο ***011- : 0001*** αφού οι το κομμάτι της εισόδου πληροί τις προϋποθέσεις για να συνδυαστεί σε ένα νέο implicant, αυτό δεν συμβαίνει επειδή το κομμάτι εξόδου μας δίνει τομή ίση με το 0. Επιπλέον, μπορούμε να δούμε άλλη μια διαφορά σε σχέση με το αλγόριθμο για μια έξοδο. Αν και ο ελαχιστόρος 5 συνδυάζεται με τον ελαχιστόρο 7, δεν μπορούμε να αφαιρέσουμε από την διαδικασία και τους δυο ελαχιστόρους, παρά μόνο τον 7, αφού η έξοδος του 5 δεν ταυτίζεται με την τομή των εξόδων των ελαχιστόρων που συνδυάστηκαν.

Καταλήγουμε στους prime implicant A, B, C, D, E, F, G, H, Οι οποίοι και αποτελούν τις στήλες του πίνακα των prime implicants.

	Column 1		Column 2		Column 3	
1	0001 : 1101	X	00_1 : 1101 m1 & m3	B	0_ _1 : 0001	F
4	0100 : 0110	X	0_01 : 0101 m1 & m5	C	_0_1 : 1100	G
			_001 : 1100 m1 & m9	X	_11_ : 0001	H
3	0011 : 1101	X	010_ : 0110 m4 & m5	D		
5	0101 : 0111	A	_100 : 0110 m4 & m12	E		
6	0110 : 0001	X				
9	1001 : 1100	X	0_11 : 0001 m3 & m7	X		
12	1100 : 0110	X	_011 : 1100 m3 & m11	X		
			01_1 : 0001 m5 & m7	X		
7	0111 : 0001	X	011_ : 0001 m6 & m7	X		
11	1011 : 1100	X	_110 : 0001 m6 & m14	X		
14	1110 : 0001	X	10_1 : 1100 m9 & m11	X		
15	1111 : 0001	X	_111 : 0001 m7 & m15	X		
			111_ : 0001 m14 & m15	X		

Σχήμα 4.11. Εφαρμογή Quine-McCluskey σε συνάρτηση πολλαπλών εξόδων

5. Heuristic Αλγόριθμοι

Οι ευριστικοί αλγόριθμοι θυσιάζουν την ακρίβεια της ελάχιστης λύσης του covering προβλήματος για ταχύτητα επίλυσης και δυνατότητα αντιμετώπισης μεγαλύτερων προβλημάτων. Δηλαδή, βρίσκουμε μια πολύ καλή προσέγγιση της λύσης σε αποδεκτό χρόνο. Στο κεφάλαιο αυτό θα ασχοληθούμε κυρίως με τον αλγόριθμο που χρησιμοποιεί το εργαλείο Espresso.

5.1. Αλγόριθμος Espresso

Ο αλγόριθμος του Espresso[1][3][4][5] αναπτύχθηκε στο πανεπιστήμιο του Berkeley και κατάφερε να γίνει σημείο αναφοράς στον τομέα του Logic Synthesis. Αντί να αναπτύσσει μια λογική συνάρτηση σε ελαχιστόρους, το πρόγραμμα χειρίζεται κύβους (*cubes*), που αναπαριστούν ένα όρο γινομένου στο ON-,OFF- και DC-set. Αν και δεν είναι εγγυημένη η καθολικά ελάχιστη λύση, ουσιαστικά προσεγγίζεται με μεγάλη επιτυχία.

Συγκρίνοντας το Espresso με άλλες μεθόδους, είναι σαφώς πιο αποδοτικό, μειώνοντας την χρήση μνήμης και υπολογιστικού χρόνου κατά πολλές τάξεις μεγέθους. Θεωρητικά δεν έχει κάποιον περιορισμό στον αριθμό μεταβλητών που μπορεί να χειριστεί.

Για τον αλγόριθμο θα πρέπει να δώσουμε ως είσοδο τα ON-set και DC-set της λογικής συνάρτησης που θέλουμε να ελαχιστοποιήσουμε. Το βασικό μέρος του Espresso είναι μια επανάληψη που “μειώνει” (***reduce***) τους implicants σε μη-prime cubes, τους “επεκτείνει” (***expand***) σε prime implicants και εξάγει ένα ελάχιστο υποσύνολο των prime implicants, έχοντας αφαιρέσει τους περιττούς primes (***irredundant***). Το κομμάτι αυτό επαναλαμβάνεται μέχρι να μην υπάρχει βελτίωση. Μόλις σταθεροποιείται η λύση, εφαρμόζουμε το reduce και το expand με λίγο διαφορετικό τρόπο ώστε να βρεθεί κάποιο τοπικό ελάχιστο (***last_gasp***). Επιπλέον υπάρχει και η μέθοδος super-gasp που χρησιμοποιείται προαιρετικά αντί του last_gasp για να αφιερώσει περισσότερη προσπάθεια για την εύρεση καλύτερης λύσης.

Ακολούθως δίνεται μια σύντομη περιγραφή των επιμέρους αλγορίθμων που χρησιμοποιούνται από το Espresso[5]:

Complement: Επιστρέφει την αναπαράσταση του συμπληρώματος μιας λογικής συνάρτησης. Χρησιμοποιείται κατά την διαδικασία της αρχικοποίησης του αλγορίθμου. Το Expand είναι η διαδικασία που απαιτεί το OFF-set ως είσοδο. Οι υπόλοιποι αλγόριθμοι απαιτούν ως είσοδο το ON-set και το DC-set.

Expand: Αντικαθιστά κάθε cube του cover της συνάρτησης με ένα prime cube. Heuristics χρησιμοποιούνται για την επιλογή ενός prime από όλους τους primes που καλύπτουν το cube

Irredundant: Βρίσκει από το cover της συνάρτησης F ένα ελάχιστο subcover το οποίο είναι και πάλι αρκετό για να αναπαριστά την συνάρτηση. Βασικό συστατικό της irredundant είναι η διαδικασία εύρεσης ταυτολογίας που εξετάζει εάν μια συνάρτηση είναι 1 για όλες τις δυνατές εισόδους.

Essential: Αναγνωρίζει ποιοι από τους prime implicants είναι essentials, δηλαδή πρέπει να βρίσκεται στην τελική λύση. Οι essential prime implicants αφαιρούνται πριν την είσοδο του αλγορίθμου στην βασική επανάληψη.

Reduce: Αντικαθιστά κάθε cube του cover της συνάρτησης F με τον μικρότερο cube που περιέχεται στο απαραίτητο για την αναπαράσταση της ίδιας συνάρτησης. Οι cubes επεξεργάζονται ένας κάθε φορά και άρα η διαδικασία είναι ευαίσθητη ως προς την σειρά επεξεργασίας των cubes.

Last_Gasp: Μια διαφορετική προσέγγιση της επανάληψης Reduce, Expand, Irredundant με σκοπό να πετύχουμε καλύτερη λύση. Κάθε cube αντικαθίσταται με το μέγιστο reduce του cube και ακολούθως στους cubes εφαρμόζεται το expand ώστε να καλύψουν άλλους μέγιστα reduced cubes. Εάν κάποιοι από τους μέγιστα reduced cubes καλύπτουν άλλους, οι primes που προκύπτουν προστίθενται στο cover. Στη συνέχεια εφαρμόζεται το irredundant για την επιλέξουμε αυτούς τους primes που μπορούν να μειώσουν το μέγεθος της συνάρτησης.

Super_Gasp: Παρόμοιο με το Last_Gasp, αλλά αντί να χρησιμοποιούμε το Expand για να επεκτείνουμε τους μέγιστα reduced cubes, χρησιμοποιούνται όλοι οι prime implicants που περιέχουν μέγιστα reduced cube. Το irredundant χρησιμοποιείται για να διαλέξει ένα ελάχιστο subcover από αυτό το μεγάλο cover από prime implicants.

Make_Sparse: Προσπαθεί επαναληπτικά να μειώσει τον συνολικό αριθμό τρανζίστορ που χρειάζονται στην PLA μορφή της συνάρτησης.

Οι διαδικασίες που είδαμε περιληπτικά παραπάνω συνδυάζονται σε έναν αλγόριθμο που χρησιμοποιείται από το Espresso. Η έξοδος του Espresso είναι ένα irredundant cover, συχνά ελάχιστο σε πληθικότητα. Ο Espresso υπολογίζει αρχικά το complement ενός cover, ώστε να χρησιμοποιηθεί από την διαδικασία του expand. Μετά εφαρμόζει την διαδικασία του expand και του irredundant για αφαίρεση περιττών prime implicants, που κάνει το cover prime και irredundant. Το επόμενο βήμα είναι η αφαίρεση των essentials, και η επανάληψη των τελεστών reduce, expand και irredundant για να επιτύχουμε irredundant covers μειωμένης πληθικότητας. Όταν δεν υπάρχει άλλη βελτίωση, το Espresso επιχειρεί να κάνει reduce και expand το cover με κάποια εναλλακτικούς heuristics, που αποτελούν το last_gasp. Παρακάτω, βλέπουμε τον αλγόριθμο[3] που χρησιμοποιείται.

```
ESPRESSO (ON-set, DC-set)
{
    OFF = complement (ON ∪ DC);     /* Compute complement */
    F = expand (ON, OFF);           /* Initial expansion */
    F = irredundant (F, DC);        /* Initial irredundant cover */
    E = essentials (F, DC);         /* Detecting essential primes */
    F = F – E;                      /* Remove essential primes from F */
    DC = DC ∪ E;                    /* Add essential primes to D */
    do
    {
        Φ2 =  cost (F);
        do
        {
            Φ1 = |F|;
            F = reduce (F, DC);          /* Perform reduction */
            F = expand (F, OFF);         /* Perform expansion*/
            F = irredundant (F, DC);     /* Perform irredundant cover */
        } while (|F|< Φ1);
        F = last_gasp (F, DC, OFF);
    } while (cost (F) < Φ2)
    F = F ∪ E;
    DC = DC – E;
    F = make_sparse (F, DC, OFF);        /* Make solution Sparse*/
}
```

5.2. Complement

Η διαδικασία του complement[1] χρησιμοποιείται για να βρίσκουμε το OFF-set δεδομένου του ON-set και του DC-set. Η υλοποίηση του με την χρήση του unate recursive paradigm έχει αποδειχθεί πολύ αποδοτική. Ο αλγόριθμος που χρησιμοποιεί το Espresso αρχικά σχεδιάσθηκε για συναρτήσεις μιας εξόδου. Η χρήση για πολλές εξόδους επιτυγχάνεται με επαναληπτική κλήση του complement για την κάθε έξοδο ξεχωριστά.

Η διαδικασία στηρίζεται στο θεώρημα του Shannon: $f' = xf'_x + x'f'_{x'}$ και άρα ο υπολογισμός μπορεί να γίνει αναδρομικά με τα cofactors της f. Η αναδρομή τερματίζει αν ικανοποιείται κάποιος από τους παρακάτω κανόνες.

- Το cover F είναι κενό. Άρα το συμπλήρωμα είναι το universe
- Το cover F έχει μια γραμμή από 1. Άρα το F είναι ταυτολογία και το complement είναι κενό.
- Το cover F αποτελείται από έναν implicant. Άρα το complement υπολογίζεται από το νόμο De Morgan.
- Όλοι οι implicants του F εξαρτώνται από μία μεταβλητή και δεν υπάρχει στήλη μόνο με 0. Άρα η συνάρτηση είναι ταυτολογία και το complement είναι κενό.
- Το cover έχει μια στήλη με 0 στην θέση j. Έστω α ένα cube μόνο με 1, εκτός από την θέση j. Τότε το $F = \alpha \cap F_\alpha$ και $F' = \alpha' \cup F'_\alpha$. Άρα υπολογίζουμε αναδρομικά το complement του F_α και του α και επιστρέφουμε την ένωση τους.

Δύο σημεία είναι βασικά στην διαδικασία αυτή . Η κατασκευή του cover του F' από τα μέρη που το απαρτίζουν σε κάθε βήμα του αλγόριθμου και η επιλογή της splitting μεταβλητής.

Η κατασκευή του cover του F' πρέπει να γίνει έτσι ώστε να κρατήσουμε το μέγεθος του μικρό, δηλαδή, $F' = (C(x) \cap F'_x) \cup (C(x') \cap F'_{x'})$. Μια απλή βελτίωση είναι να συγκρίνουμε κάθε cube του F'_x με κάθε περίπτωση του $F'_{x'}$. Εάν το cube καλύπτεται, αφαιρείται από το σύνολο και προστίθεται σε μια νέα λίστα G. Στο επόμενο βήμα χρησιμοποιείται ένας merge αλγόριθμος που επιστρέφει $F' = (C(x) \cap \underline{\mathbf{F}}'_x) \cup (C(x') \cap \underline{\mathbf{F}}'_{x'}) \cup G$, όπου $\underline{\mathbf{F}}'_x$ και $\underline{\mathbf{F}}'_{x'}$ είναι τροποποιημένα cover των cofactors.

Η επιλογή της splitting μεταβλητής βασίζεται στο URP. Σύμφωνα με το παρακάτω θεώρημα, ο υπολογισμός ενός cover του complement μιας unate συνάρτησης είναι απλούστερος σε σχέση με μια binate συνάρτηση. Εφόσον γενικά οι συναρτήσεις είναι binate, η επιλογή της splitting μεταβλητής γίνεται για να βελτιώσει την πιθανότητα να παράγει unate cofactors

Παράδειγμα. Έστω η συνάρτηση F = ab + ac + a' :

01 01 11
01 11 01
10 11 11

Η μόνη binate μεταβλητή, η οποία επιλέγεται ως splitting είναι η α. Άρα ο υπολογισμός του complement γίνεται με το F' = (C(α') ∩ F'$_{α'}$) ∪ (C(α) ∩ F'$_{α}$). Ο υπολογισμός του F'$_{α}$ είναι ταυτολογία, αφού το F'$_{α'}$ είναι κενό. Ο υπολογισμός του $F_{α}$ δίνει:

11 01 11
11 11 01

το οποίο είναι θετικό unate για τις μεταβλητές δύο και τρία. Επιλέγοντας την b, έχουμε F'$_{α}$ = $F_{αb'}$ ∪ (C(b') ∩ F'$_{αb'}$). F_{ab} είναι ταυτολογία και το complement είναι κενό. Άρα η $F_{ab'}$ = 11 11 01 και το complement 11 11 10, η τομή του οποίου με το C(b') = 11 10 11 δίνει F'$_{a}$ = 11 10 10. Το complement της f είναι F' = C(a) ∩ F'$_{a}$ = 01 10 10, επειδή C(a) = 01 11 11 και η F'$_{a'}$ είναι κενή. Άρα το complement είναι f= ab'c'

5.3. Expand

Ο τελεστής **expand** [3][5] χρησιμοποιείται από τα περισσότερα εργαλεία ελαχιστοποίησης. Σκοπός του είναι να αυξήσει το μέγεθος ενός implicant ενός δεδομένου cover της F, ώστε να μπορέσουν να καλυφθούν implicants μικρότερου μεγέθους και να διαγραφούν. Implicants που έχουν γίνει όσο γίνεται expanded είναι primes της συνάρτησης. Ως αποτέλεσμα το expand κάνει ένα cover prime και minimal.

Το expand ενός implicant γίνεται αλλάζοντας ένα ή περισσότερα από τα 0 του σε 1. Αυτό αντιστοιχεί σε αύξηση του μεγέθους του implicant και άρα στην κάλυψη περισσότερων ελαχιστόρων. Το βασικό ερώτημα είναι εάν ο expanded cube είναι έγκυρος, δηλαδή παραμένει implicant της συνάρτησης F. Η προσέγγιση για να απαντήσουμε στο ερώτημα αυτό είναι να ελέγχουμε την τομή

expanded implicant με το OFF-set της συνάρτησης. Η τομή μπορεί να υπολογισθεί πολύ αποδοτικά. Πολλές φορές το OFF-set μπορεί να είναι πολύ μεγάλο και να καταναλώνει αρκετούς πόρους του συστήματος.

Η διαδικασία του expand αντιμετωπίζει του implicants ένα την φορά και αλλάζει τις 0 εισόδους σε 1 εξετάζοντας ο implicant που δημιουργείται να είναι έγκυρος. Η υπολογιστική αποδοτικότητα και η ποιότητα του αποτελέσματος εξαρτάται από 2 παράγοντες. 1) Την σειρά με την οποία επιλέγονται οι implicants και 2) την σειρά με την οποία τα 0 αλλάζουν σε 1. Και για τις 2 περιπτώσεις χρησιμοποιούνται heuristics.

Για την επιλογή των κύβων η λογική είναι να κάνουμε expand στα cubes τα οποία δεν είναι πιθανό να καλυφθούν από άλλα cubes. Η τεχνική είναι : ένα διάνυσμα υπολογίζεται , τα στοιχεία του οποίου είναι το άθροισμα των στηλών του πίνακα που απεικονίζει το F. Κάθε cube έχει ένα βάρος που είναι το εσωτερικό γινόμενο του ίδιου του cube και του διανύσματος που υπολογίσθηκε. Τα cubes κατατάσσονται σε αύξουσα σειρά ανάλογα με τα βάρη. Προφανώς αν ένας Implicant είναι prime δεν χρειάζεται expand, αφού κάτι τέτοιο έχει γίνει ήδη.

Η επιλογή των όρων που θα γίνουν expand είναι πιο περίπλοκη. Προφανώς μια αλλαγή με οποιαδήποτε σειρά από 0 σε 1 θα οδηγήσει σε prime, όμως μπορούμε να οδηγηθούμε σε διαφορετικούς primes ανάλογα με την κατεύθυνση του expand που θα επιλέξουμε. Σκοπός είναι να μετατρέψουμε έναν implicant σε prime, καλύπτοντας ταυτόχρονα το μέγιστο αριθμό από άλλους implicants. Με χρήση ενός βοηθητικού πίνακα του Blocking Matrix, μπορούμε να προσδιορίσουμε κάποιες εφικτές και κάποιες μη-εφικτές κατευθύνσεις για expand. Το επόμενο βήμα είναι να ψάξουμε για cubes στο cover που καλύπτονται από κάποια συγκεκριμένο και εφικτό expand. Όταν εξαντληθεί αυτή η αναζήτηση, ψάχνουμε για κατευθύνσεις που επιτρέπουν στον expanded implicant να καλύψει ένα μέγιστο αριθμό άλλων cubes.

Κάποιοι ορισμοί που θα μας βοηθήσουν να ορίσουμε καλύτερα την διαδικασία είναι:

a) *Free* είναι ένα σύνολο των υποψήφιων προς expand εισόδων. Αρχικά, αποτελείται από καταχωρήσεις που είναι όλα μηδενικά, αλλά όπως

προχωράει η διαδικασία, αφαιρούνται στοιχεία και η διαδικασία τερματίζει μόλις το free αδειάσει.

b) *Overexpanded cube* είναι ένα cube του οποίου οι είσοδοι που είναι στο free γίνονται raise. Για οποιοδήποτε implicant β του F, ο μικρότερος cube που περιέχει το α και το β είναι ο supercube τους. Ο cube β είναι δυνατόν να καλυφθεί, εάν το εάν το supercube είναι implicant του f, δηλαδή, εάν δεν τέμνει το OFF-set.

Τα βασικά βήματα στην διαδικασία του expand είναι:

- ***Προσδιορισμός των essentials***
    - Προσδιορισμός των καταχωρήσεων που δεν μπορούν να γίνουν ποτέ raise και αφαίρεση τους από το free. Αναζήτηση cube στο OFF-set που έχει απόσταση 1 από το α.
    - Προσδιορισμός αυτών των κομματιών που μπορούν πάντα να γίνουν raise. Εφαρμογή raise και αφαίρεση τους από το free. Αναζήτηση στήλης που έχει μόνο μηδενικά στο OFF-set
- ***Εύρεση cubes που είναι εφικτό να καλυφθούν***

  Αν υπάρχει implicant β, του οποίου το supercube με το α είναι εφικτή πράξη, επανάλαβε τα επόμενα βήματα.
    - Raise την κατάλληλη καταχώρηση του α και αφαίρεση από το free
    - Αφαίρεση από το free καταχωρήσεων που δεν μπορούν να γίνουν raise, ή που γίνονται πάντα raise και ενημέρωση του α.
- ***Expand κατευθυνόμενο από το overexpanded cube***

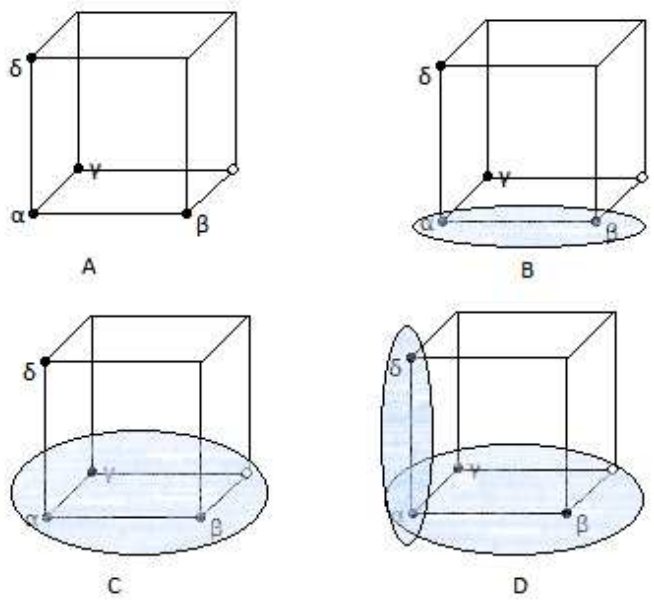
  Όσο το overexpanded cube του α καλύπτει άλλα cubes του F, επανάλαβε τα ακόλουθα βήματα:
    - Raise μια καταχώρηση του α ώστε να επικαλύψει ένα μέγιστο αριθμό από αυτά τα cubes
    - Αφαίρεση από το free καταχωρήσεων που δεν μπορούν να γίνουν raise, ή που γίνονται πάντα raise και ενημέρωση του α.
- ***Εύρεση του μεγαλύτερου prime implicant***
    - Σχηματισμός ενός προβλήματος covering και επίλυση με heuristic μέθοδο. Στο covering πρόβλημα, σκοπός είναι να βρούμε ένα μέγιστο υποσύνολο του free για να κάνουμε raise, χωρίς όμως να υπάρχει τομή του expanded implicant με το OFF-set. Δηλαδή, εύρεση του μεγαλύτερου prime που καλύπτει το α.

Για παράδειγμα, θα θεωρήσουμε την συνάρτηση *F = a'b'c' + ab'c' + a'bc'+ a'b'c* και το abc' ως DC. Στο σχήμα 5.1.α βλέπουμε την αναπαράσταση της F. Οι implicants είναι οι α, β, γ, δ. Αρχικά επιλέγεται ο β = 01 10 10 για expand. Το free περιέχει τις στήλες 1,4,6.

Entries που δεν μπορούν να γίνουν raise. Το cube 01 11 01 του OFF-set έχει απόσταση 1 από τον implicant β και διαφέρει στο τρίτο πεδίο. Άρα η στήλη 6 δε μπορεί να γίνει raise και αφαιρείται από το free.Γνωρίζοντας το OFF-set δεν μπορούμε να προσδιορίσουμε entries που μπορούν πάντα να γίνονται raise.

Το supercube του β με το α είναι έγκυρο και άρα μπορούμε να κάνουμε raise το πρώτο πεδίο του β => β = 11 10 10. Ομοίως και το supercube του β με το γ είναι έγκυρο => β = 11 11 10. Το supercube του β με το δ δεν είναι έγκυρο. Ο expanded implicant β είναι ο 11 11 10, το free είναι άδειο και δεν μπορεί να γίνει άλλο expand. Επιπλέον οι implicants α,γ διαγράφονται από τη F, καθώς καλύπτονται από το expanded β.

Στη συνέχεια ασχολούμαστε με τον implicant δ = 10 10 01, με free 2,4,5. Το δ έχει απόσταση 1 και από τα δύο στοιχεία του OFF-set, άρα οι στήλες 2, 4 δεν μπορούν να γίνουν raise και αφαιρούνται από το free. Η στήλη 5 του OFF-set μπορεί να γίνει raise => δ =10 10 11. Άρα το τελικό cover είναι το expanded α και το expanded β.(σχήμα 5.1.δ)



Σχήμα 5.1.[1] α)Αναπαράσταση της F, β)Expand, γ)Prime cover

5.4. Reduce

Ο τελεστής reduce[3][5] έχει ως σκοπό να μειώσει το μέγεθος του κάθε implicant ενός cover F, έτσι ώστε ένα επιτυχές expand να οδηγήσει σε ένα cover μικρότερης πληθικότητας. Ένας reduced implicant είναι έγκυρος, αν μαζί με τους υπόλοιπους implicants καλύπτουν την F. Το reduced cover έχει την ίδια πληθικότητα με το αρχικό cover και δεν είναι prime (εκτός αν δεν μπορεί να γίνει κανένας implicant reduced). Το reduce βασίζεται στην σειρά με την οποία αντιμετωπίζονται οι implicants, συνεπώς δεν εγγυάται cover χωρίς περιττούς implicants.

Διαισθητικά, για να κάνουμε reduce έναν implicant α μιας συνάρτησης F , πρέπει να αφαιρέσουμε από τον α τους ελαχιστόρους που καλύπτονται από το F-{α}. Κάτι τέτοιο μπορεί να γίνει παίρνοντας την τομή του α με το συμπλήρωμα του F-{α}, πρέπει όμως να διασφαλίσουμε ότι ως αποτέλεσμα παράγεται ένας implicant. Αφού το συμπλήρωμα του F-{α} είναι μια ομάδα από implicants, μπορούμε να το αντικαταστήσουμε από το supercube, δηλαδή τον μικρότερο cube που περιέχει όλους τους cubes, και τομή του οποίου με το α δίνει το αποτέλεσμα. Τυπικά η διαδικασία στηρίζεται στο παρακάτω θεώρημα:

Θεώρημα: Έστω το α ∈ F είναι implicant και Q = F ∪ DC – {α}. Τότε, ο μέγιστα reduced cube είναι ο **a = α ∩ supercube(Q_α')**

Η διαδικασία reduce αποτελείται από 2 βήματα: α) ταξινομεί τους implicants, β) αντικαθιστά κάθε implicant με τον μέγιστα reduced implicant. Το Espresso ταξινομεί τους implicant ανάλογα με τα βάρη τους, όπως κα με τον τελεστή expand. Οι implicants ταξινομούνται σε φθίνουσα σειρά ανάλογα με το βάρος τους, ώστε να επεξεργαστούμε πρώτα αυτούς που είναι μεγάλοι και επικαλύπτουν πολλούς άλλους implicants.

Ο υπολογισμός μέγιστα reduced cube μπορεί να γίνει με διάφορους τρόπους. Το Espresso κάνει χρήση του Unate Recursive Paradigm για τον υπολογισμό του supercube του $\mathbf{Q_\alpha'}$.

supercube(F') = supercube($U^{p-1}_{k=0}\, C(x^{\{k\}}) \cap supercube(F'_{x^{\{k\}}})$ ***)***

Στο τέλος της αναδρομής, υπολογίζουμε τον supercube του συμπληρώματος ενός cube που είναι:

- Άδειος, όταν ο cube δεν εξαρτάται από καμία μεταβλητή
- Το συμπλήρωμα του cube, όταν εξαρτάται από μια μεταβλητή
- Το universe, όταν εξαρτάται από δύο ή περισσότερες μεταβλητές.

Για παράδειγμα, έστω και πάλι η συνάρτηση *F = a'b'c' + ab'c' + a'bc'+ a'b'c* και το *abc'* όρος αδιαφορίας. Υποθέτουμε ότι έχει γίνει expand ως εξής:

α 11 11 10
β 10 10 11

Το διάνυσμα που αναπαριστά το άθροισμα των στηλών είναι το $[21\ 21\ 21]^T$ και άρα τα βάρη είναι 2*1 + 1*1 + 2*1 + 1*1 + 2*1 + 1*0 = 8 για το α, και 2*1 + 1*0 + 2*1 + 1*0 + 2*1 + 1*1 = 7 για το β. Άρα ο implicant α επιλέγεται πρώτος για reduce. Το συμπλήρωμα του β είναι **β'** :

01 11 11
11 01 11

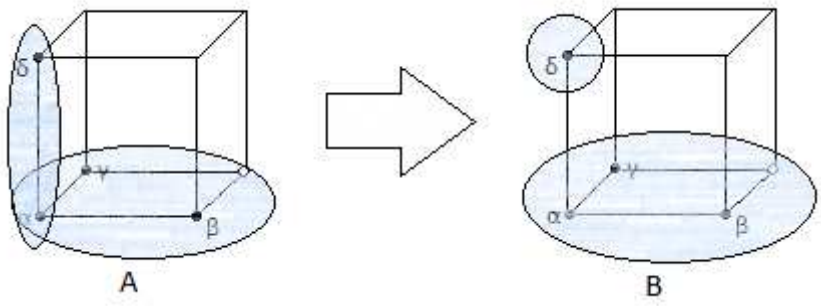
Η τομή του με το α δίνει:

01 11 10
11 01 10

Το οποίο σημαίνει ότι ο α πρέπει να αντικατασταθεί με 2 cubes, το οποίο όμως δεν επιτρέπεται. Άρα ο α δεν μπορεί να γίνει reduced.

Με την σειρά επεξεργαζόμαστε τον β. Ο μέγιστα reduced cube είναι ο b = **α ∩ supercube(Q_β'),** όπου Q = F ∪ DC –{β}, δηλαδή 11 11 10. Άρα Q' = 11 11 01, Q_β' = Q' και supercube(Q'_β) = Q'. Άρα b = β ∩ Q' = 10 10 01 και το reduced cover είναι (σχήμα 5.2.):

11 11 10
10 10 01



Σχήμα 5.2. [1] α) Expanded Cover, β)Μετά από εφαρμογή του reduce

5.5. Irredundant

Ένα cover το οποίο είναι irredundant[3][5] είναι ελάχιστο ως προς τον αριθμό των implicant που περιέχει. Δηλαδή, η πληθικότητα του είναι τοπικό ελάχιστο. Ο τελεστής irredundant κάνει αφαιρεί από το cover τους περιττούς implicants, διαγράφοντας τον μέγιστο αριθμό από περιττούς implicants που μπορεί να διαγράψει.

Υποθέτουμε ότι το cover που έχουμε να επεξεργαστούμε είναι prime, αφού κατασκευάστηκε από την διαδικασία του expand. Το cover της F χωρίζεται σε τρεις κατηγορίες:

- Το relatively essential σύνολο E^r, το οποίο περιέχει τους Implicants Που καλύπτουν μερικούς ελαχιστόρους της συνάρτησης που δεν καλύπτονται από άλλους implicants της F.
- Το totally redundant σύνολο R^t, το οποίο περιέχει τους implicants που καλύπτονται από το relatively essential σύνολο.
- Το partially redundant σύνολο R^p, το οποίο περιέχει τους υπόλοιπους implicants.

Ο υπολογισμός του relatively essential σύνολο βασίζεται στον έλεγχο αν ένας implicant $\alpha \in F$ δεν καλύπτεται από το $F \cup DC-\{\alpha\}$. Ομοίως, ο υπολογισμός του totally redundant σύνολο βασίζεται στον έλεγχο αν ένας implicant $\alpha \in F$ καλύπτεται από το $E^r \cup DC$ και άρα και το $R^p = F-\{ E^r \cup R^t \}$. Η βασική δουλειά του irredundant είναι να βρίσκει ένα υποσύνολο του R^p που μαζί με το E^r καλύπτουν το ON-set της F.

Ένας απλοϊκός αλγόριθμος για την εύρεση περιττών implicants είναι να επιλέγουμε έναν τους implicants $\alpha \in R^p$ και να ελέγχουμε αν περιέχεται στο $H = E^r \cup R^p \cup DC-\{\alpha\}$. Σε περίπτωση που ανήκει μπορούμε να τον διαγράψουμε και επαναλαμβάνοντας αυτή την διαδικασία να πάρουμε ένα irredundant cover, το οποίο όμως τελικά να μην είναι το καλύτερο δυνατό. Το αποτέλεσμα που παίρνουμε δεν είναι βέλτιστο γιατί αντιμετωπίζει τους partially redundant cubes με σειριακό τρόπο. Για να βελτιώσουμε τον αλγόριθμο θα πρέπει λάβουμε υπόψιν και τις σχέσεις επικάλυψης μεταξύ των cubes.

Ο αλγόριθμος που χρησιμοποιείται από το Espresso για την εύρεση των περιττών implicants περιλαμβάνει μια τροποποίηση του αλγορίθμου ελέγχου ταυτολογίας. Αντί να ελέγχουμε για ταυτολογία, προσδιορίζουμε ένα σύνολο

cubes, οι οποίοι όταν αφαιρεθούν, αποτρέπουν την F από το να είναι ταυτολογία.

Έστω $\alpha \in R^p$, εξετάζουμε αν περιέχεται από το $H = E^r \cup R^p \cup DC\text{-}\{\alpha\}$, το οποίο ισοδυναμεί με τον έλεγχο του cofactor H_α για ταυτολογία. Ο έλεγχος δίνει θετική απάντηση, σύμφωνα με τον ορισμό του partially redundant prime. Ο έλεγχος για ταυτολογία ισοδυναμεί με τον συνδυασμό των ελέγχων σε κάθε φύλλο του δέντρου που παράγεται από τον αλγόριθμο ταυτολογίας. Κάθε φύλλο αντιπροσωπεύει ένα σύνολο από συνθήκες για ταυτολογία, ή ισοδύναμα για covering. Όταν ο αλγόριθμος ταυτολογίας φτάνει σε ένα φύλλο, σημειώνουμε τα cubes του H_α που κάνουν την F ταυτολογία. Εάν το φύλλο συσχετίζεται με κάποιο cube στο $E^r \cup DC$, σταματάμε να ελέγχουμε το φύλλο, καθώς το αντίστοιχο κομμάτι του α καλύπτεται είτε από ένα relatively essential prime ή μέρος του DC-set. Διαφορετικά σχηματίζουμε ένα δυαδικό διάνυσμα του οποίου τα entries είναι 1 σε αντιστοιχία με αυτά τα cubes του R^p, των οποίων η συνδυασμένη αφαίρεση παύει να καλύπτει μέρος του α. Επαναλαμβάνοντας τα βήματα για κάθε φύλλο και κάθε cube του R^p και μαζεύοντας τα vectors, καταλήγουμε σε ένα covering πίνακα **A**.

Κάθε στήλη του A αντιστοιχεί ένα στοιχείο του R^p και κάθε γραμμή αντιστοιχεί σε φύλλα του δέντρου ταυτολογίας. Κάθε γραμμή αντιπροσωπεύει ένα υποσύνολο cubes του R^p, των οποίων η αφαίρεση παύει να καλύπτει μέρος του α. Έτσι το πρόβλημα εύρεσης ενός irredundant set του R^p είναι ισοδύναμο με την εύρεση ενός ελάχιστου υποσυνόλου στηλών τέτοιο ώστε όλες οι γραμμές να έχουν τουλάχιστον έναν 1. Αυτό το unate covering πρόβλημα λύνεται με exact ή heuristic αλγόριθμο. Από το Espresso χρησιμοποιείται ένας greedy αλγόριθμος.

Για παράδειγμα έστω το cover (σχήμα 5.3.α) :

α 10 10 11
β 11 10 01
γ 01 11 01
δ 01 01 11
ε 11 01 10

όπου $E^r = \{\alpha,\varepsilon\}$, $R^t = \{\ \}$ και $R^p = \{\beta,\gamma,\delta\}$. Το πρόβλημα είναι να βρούμε ένα ελάχιστο υποσύνολο του R^p, ώστε μαζί με το E^r να καλύπτουν το ON-set της F. Έστω ότι επιλέχθηκε για έλεγχο ο implicant γ από τον απλοϊκό αλγόριθμο.

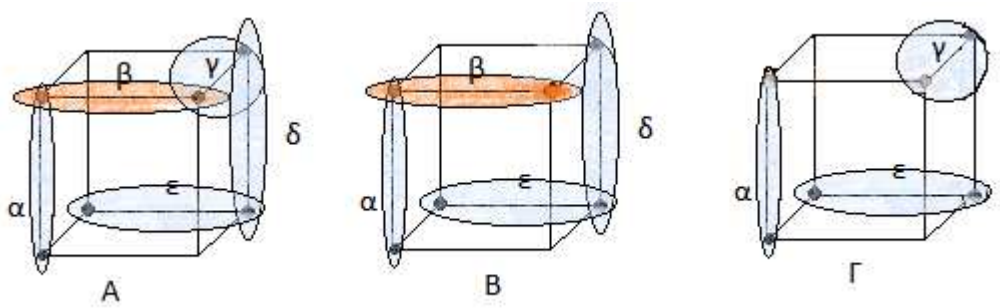
Επειδή ο γ καλύπτεται από τους β, δ μπορούμε να τον θεωρήσουμε περιττό και να τον αφαιρέσουμε από το cover. Τώρα κανείς άλλος implicant δεν μπορεί να αφαιρεθεί από το cover και άρα το cover α, β, δ, ε είναι irredundant (σχήμα 5.3.β). Το πρόβλημα είναι ότι λαμβάνοντας υπόψιν την αλληλεπίδραση των σχέσεων κάλυψης μεταξύ των implicants βρίσκουμε ένα irredundant cover μικρότερου βαθμού (σχήμα 5.3.γ). Στο αποτέλεσμα του σχήματος 5.3.γ. καταλήγουμε με την ακόλουθη διαδικασία. Επιλέγουμε από το R^p τον β για έλεγχο. Ο $\mathbf{H_\beta}$ είναι:

10 11 11
01 11 11

Η splitting μεταβλητή είναι προφανώς η a αφού δεν εξαρτάται από καμία άλλη και δεν υπάρχει στήλη από μηδενικά. Τα 2 cubes στο H_β σχετίζονται με το α και το γ. Αφού το α είναι essential και πρέπει να είναι στο τελικό cover, καταλαβαίνουμε ότι η αφαίρεση του α θα αφήσει ακάλυπτο το β. Άρα το διάνυσμα που προκύπτει είναι [110] για τα β, γ, δ αντίστοιχα(εφόσον το β, καλύπτει τον εαυτό του). Συνεχίζουμε με το γ. Το $\mathbf{H_\gamma}$ είναι :

11 10 11
11 01 11

Έχουμε ταυτολογία και τα cubes σχετίζονται με τα β και δ, που είναι partially redundant. Άρα το διάνυσμα που προκύπτει είναι [111]. Τέλος για την δ έχουμε, ομοίως το vector [011]. Ο covering πίνακας που προκύπτει είναι ο A= {110, 111, 011} από τον οποίο ένα ελάχιστο cover των γραμμών του παίρνουμε από την στήλη 2, δηλαδή τον implicant γ που προστίθεται στους essential και καταλήγουμε στο σχήμα 5.3.γ.



Σχήμα 5.3.α) [1]Αρχικό cover, β) Irredundant cover απλοϊκός αλγόριθμος, γ) Ελάχιστο irredundant cover

5.6. Essentials

Η διαδικασία αυτή είναι σημαντική και για τους exact και για τους heuristic αλγορίθμους, καθώς οι essential[5] prime implicants αποτελούν μέρος οποιουδήποτε ελάχιστου cover και άρα μπορούν να αφαιρεθούν κατά την διάρκεια επεξεργασίας των υπολοίπων implicants και να προστεθούν κατευθείαν στο τελικό cover.

Διαισθητικά για το πρόβλημα. Έστω ένα cover {α, β, γ, δ}. Θέλουμε να εξετάσουμε αν το α είναι essential. Αφαιρούμε από το cover τους implicants που καλύπτονται από το α, δηλαδή τον β, και άρα το cover μειώνεται σε G={γ, δ} (σχήμα 5.4.β). Αν υπάρχουν cubes στο G που αν επεκταθούν καλύπτουν τους ελαχιστόρους του α , τότε δεν είναι essential. Για να μπορεί να συμβεί αυτό πρέπει να υπάρχει 1-1 αντιστοιχία των ελαχιστόρων του α με αυτούς του cover και να έχουν απόσταση ίση με 1. Η συνθήκη αυτή είναι αληθής όταν το CONSENSUS(G,α) (σχήμα 5.4.γ) καλύπτει το α. Αν αυτό δεν ισχύει τότε ο α είναι essential.

Η ανίχνευση essentials μπορεί να γίνει στηριζόμενη στο θεώρημα του **Sasao**.

Θεώρημα: Έστω F = G ∪ α, όπου η τομή του α με το G είναι κενή. Τότε το α είναι ένας essential prime implicant αν και μόνο αν το CONSENSUS(G,α) δεν καλύπτει το α.

Επιστέγασμα: Ο α είναι essential αν και μόνο αν H ∪ DC δεν καλύπτει το α, όπου:

$$\mathbf{H= CONSENSUS\ (((ON \cup DC) \#\ \alpha),\ \alpha)}$$

Το παραπάνω μας επιτρέπει να μειώσουμε τον έλεγχο για essential σε πρόβλημα που μπορεί να λυθεί με ταυτολογία.

Για παράδειγμα έστω το cover:

α 10 10 11
β 11 10 01
γ 01 11 01
δ 01 01 11

F#α:

01 11 01
01 01 11

Και H = 11 10 01. Τότε H_α = 11 11 01, το οποίο δεν είναι ταυτολογία και άρα το α δεν περιέχεται στο H και είναι essential. Τώρα εξετάζοντας το β.

F#β είναι: **10 10 10**
01 01 11

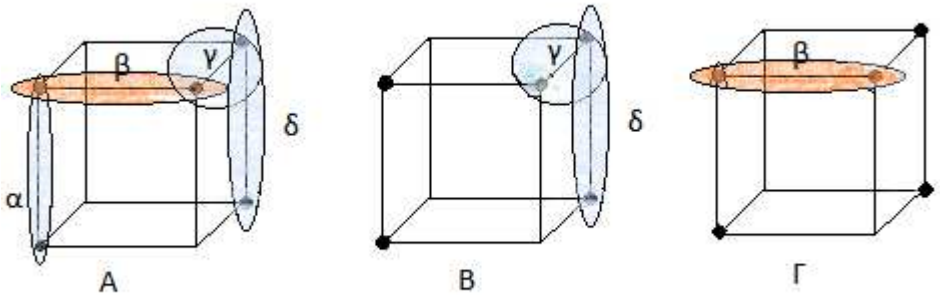
Και το **H** είναι :

10 10 11
01 11 01

Και το **Hβ** είναι:

10 11 11
01 11 11

το οποίο είναι ταυτολογία και άρα ο β περιέχεται στο H και δεν είναι essential.



Σχήμα 5.4.α) Αρχικό cover β)cover G = F#α γ)CONSENSUS(G,α)

5.7. Last_Gasp και Super_Gasp

Το Last_Gasp [3][5] αρχικά υπολογίζει το μέγιστο reduction για κάθε cube του ON-set της συνάρτησης F και δημιουργεί ένα νέο cover G. Εάν ένα cube δεν μπορεί να γίνει reduced αγνοείται. Η διαδικασία expand τροποποιείται ώστε : α) το expand κάποιου cube σταματάει μόλις προσδιοριστεί ότι δεν μπορεί να καλύψει άλλους cubes και αφαιρείται από το G, β) όλα τα cubes γίνονται expand ακόμη και αν έχουν καλυφθεί από την επέκταση άλλου cube. Αυτά τα cubes τα οποία καταφέρνουν να καλύψουν κάποιον reduced cube είναι πιθανόν χρήσιμοι primes για την μείωση του βαθμού του cover. Αυτοί οι νέοι primes προστίθενται απλά στο cover F, και ακολούθως η διαδικασία του irredundant αφαιρεί τους περιττούς.

Ένας εναλλακτικός αλγόριθμος που χρησιμοποιείται από το Espresso είναι αυτός του Super_Gasp. Το Super_Gasp υπολογίζει το μέγιστο reduction για κάθε cube της F και δημιουργεί όλους τους prime implicants που καλύπτουν το cube, αντί για να δημιουργεί ένα μόνο prime implicant. Για να παράγει όλους τους prime implicant που καλύπτουν ένα cube χρησιμοποιείται το Expand. Ταξινομώντας αυτους τους prime implicants, μπορούμε να βρουμε prime που

είναι διπλοί. Το irredundant τότε αφαιρεί τους περιττούς από τους υπόλοιπους primes.

Φυσικά η διαδικασία παραγωγής όλων των primes που καλύπτουν τους μέγιστα reduced cubes αυξάνει πάρα πολύ το μέγεθος του cover. Γι αυτό το Espresso σταματάει την παραγωγή primes στην περίπτωση που είναι πάρα πολλοι και εφαρμόζει το Last_Gasp.

6. Πειραματικά αποτελέσματα

Στο κεφάλαιο αυτό θα παρουσιαστούν τα πειραματικά δεδομένα, τα οποία προέκυψαν από την εφαρμογή exact αλγορίθμων σε κάποια κυκλώματα, και η σύγκρισή τους με τα αποτελέσματα που προκύπτουν από το εργαλείο Espresso. Η σύγκριση έγινε με σημείο αναφοράς τα πειραματικά αποτελέσματα τα οποία έχουν δημοσιοποιηθεί από τον Rudell στην διδακτορική του διατριβή, αλλά και σε σχέση με τα αποτελέσματα που προκύπτουν από την εφαρμογή του εργαλείου Espresso exact μέσα από την σουίτα εργαλειών σύνθεσης του MVSIS.

6.1. Το σύνολο κυκλωμάτων

Τα κυκλώματα-παραδείγματα που χρησιμοποιήθηκαν για τις δοκιμές των αλγορίθμων είναι μια συλλογή από διαφορετικές πηγές και βιβλιογραφίες με κύρια προέλευση όμως τα παραδείγματα τα οποία συμπεριλαμβάνονται στις διάφορες εκδόσεις του εργαλείου Espresso. Τα παραδειγματα αυτά είναι τα ίδια που χρησιμοποιήθηκαν και για την δοκιμη του Espresso. Επιλέχθηκε ένα υποσυνόλο των παραδειγμάτων αυτών, τέτοιο ώστε να είναι αντιπροσωπευτικό του συνόλου των κατηγοριών των κυκλωμάτων και να υπάρχει άμεση σύγκριση με τα αποτελέσματα του εργαλείου Espresso.

Χρησιμοποιήθηκαν συνολικά 60 κυκλώματα, από διαφορετικές πηγές. Από αυτά, 51 παραδείγματα προέρχονται από τα παραδείγματα του Espresso, ενώ 9 προέρχονται από άλλες πηγές, όπως βιβλία, διαλέξεις πάνω στο θέμα της σύνθεσης. Από αυτά τα 51 παραδείγματα του Espresso, 35 είναι κυκλώματα τα οποία είναι ορισμένα ως βιομηχανικά παραδείγματα, δηλαδή η προέλευση τους είναι από την σχεδιασμό ολοκληρωμένου είτε από την βιομηχανία είτε από κάποιο πανεπιστήμιο, και τα υπόλοιπα προέρχονται από μαθηματικές συναρτήσεις. Για να γίνουν ευκολότερα αντιληπτά τα αποτελέσματα θα γίνει παρακάτω μια κατηγοριοποίηση των κυκλωμάτων ανάλογη με αυτή που έγινε για το εργαλείο EspressoMV.

Το πρώτο επίπεδο διαχωρισμού των κυκλώματων μπορεί να γίνει ως προς τον αριθμό των εξόδων τους. Έτσι προκύπτουν δύο κατηγορίες κυκλωμάτων, αυτά τα οποία έχουν μία έξοδο (**single-output**), και τα οποία θα χαρακτηρίζονται από την λέξη *single* στους επόμενους πίνακες, και αυτά που

έχουν περισσότερες της μιας εξόδου (**multiple-output**), και τα οποία θα χαρακτηρίζονται με την λέξη *multi*.

Ένα δεύτερο επίπεδο διαχωρισμού των κυκλωμάτων μπορεί να οριστεί ανάλογα με το γεγονός, εάν αυτά είναι πληρως ορισμένα παραδειγμάτα, δηλαδή η είσοδος τους δεν είναι ομαδοποιημένη, αλλά πλήρως ορισμένη όπως φαίνεται στο σχήμα 6.1.α, και τα οποία συμβολίζονται από εδώ και στο εξής με την λέξη *full.* Αντίθετα, εάν η είσοδος είναι ομαδοποιημένη όπως φαίνεται στο σχήμα 6.1.β χαρακτηρίζονται στο εξής ως *packed.*

Ο τελευταίος διαχωρισμός των παραδειγμάτων αυτών μπορεί να γίνει ως προς την δυσκολία ελαχιστοποίησης του καθενός, το οποίο και θα εξετάσουμε παρακάτω.

```
#Example circuit
#fully described
.i 5
.o 8
.p 8
00001 00000001
00011 00000010
00101 00000100
00111 00001000
01001 00010000
01011 00100000
01101 01000000
01111 10000000
.e
```

```
#Example circuit
#packed description
.i 4
.o 5
.p 11
101- 10001
110- 10001
111- 00011
-001 10010
--0- 00100
-011 01000
000- 10001
-101 01000
-000 01010
-110 11000
011- 10010
.e
```

Σχήμα 6.1. α)Παραδείγμα κυκλώματος που περιγράφεται πλήρως (full) β)Παράδειγμα κυκλώματος, συνάρτηση που δεν είναι πλήρως ορισμένη. (packed)

6.2. Αποτελέσματα

Στους πίνακες όπου παρουσιάζονται τα αποτελέσματα παρακάτω σημειώνεται κατά σειρά :

- το όνομα του κυκλώματος (Circuit),
- το είδος του προβλήματος και το επίπεδο δυσκολίας (Type),
- ο αριθμός εισόδων-εξόδων (In/Out)
- οι αρχικοί όροι τις συνάρτησης (Terms)
- τα αρχικά cubes (Cubes)

- ο αριθμός τον prime implicants που παρήχθησαν κατά την εφαρμογή των αλγορίθμων (Primes)
- ο αριθμός των essential prime implicants (Essentials),
- το αποτέλεσμα της ελαχιστοποίησης εκφρασμένο σε cubes (Minimized)
- το αποτέλεσμα της ελαχιστοποίησης όπως έχει δοθεί από το EspressoMV-heuristic (EspressoMV-H)
- το αποτέλεσμα της ελαχιστοποίησης όπως έχει δοθεί από το EspressoMV-exact (EspressoMV-E)

6.2.1. Βαθμολόγηση δυσκολίας ελαχιστοποίησης κυκλωμάτων

Εφόσον υπάρχει ένας μεγάλος αριθμός από διαφορετικά κυκλώματα προς δοκιμή, έχει γίνει μια διαβάθμιση των προβλημάτων ως προς την δυσκολία που παρουσιάζουν στην ελαχιστοποίηση τους, ώστε οι αριθμοί να αποκτούν μεγαλύτερο νόημα και να βγάλουμε σωστά συμπεράσματα. Επιπλέον η διαβάθμιση έχει γίνει σε αντιστοιχία με την κλίμακα δυσκολίας που επικαλείται το Espresso MV.

Τα προβλήματα, χωρίζονται σε 3 βασικές κατηγορίες (*πίνακας 1*), οι οποίες είναι:

- Μη-κυκλικά προβλήματα με εύκολη επίλυση, και τα οποία αναφέρονται από δω και στο εξής ως *trivial*
- Μη-κυκλικά προβλήματα μεγαλύτερης δυσκολίας, τα οποία αναφέρονται στο εξής ως *non-cyclic*
- Κυκλικά προβλήματα, τα οποία αναφέρονται στο εξής ως *cyclic*

Στην πρώτη κατηγορία ανήκουν κυκλώματα, τα οποία ελαχιστοποιούνται χρησιμοποιώντας μόνο τους essential prime implicants και για τα οποία δεν απαιτείται ιδιαίτερη επεξεργασία για την επίλυση τους.

Στη δεύτερη κατηγορία ανήκουν και πάλι κυκλώματα τα οποία ομοίως ελαχιστοποιούνται επιλέγοντας μόνο τους essential prime implicants, είναι παρόλα αυτά πιο δύσκολα στην εύρεση ελάχιστου από την προηγούμενη κατηγορία.

Τέλος, ως cyclic αναφέρονται τα κυκλώματα στα οποία η επιλογή και αφαίρεση των prime implicants δεν συνεπάγεται ελάχιστη λύση, αλλά μας αφήνει με ένα κυκλικό πίνακα προς επίλυση, όπως είδαμε στο κεφάλαιο 4.3. Τα cyclic αυτά κυκλώματα χωρίζονται σε απλά και δύσκολα, τα οποία και θα συμβολίζουμε με το αρχικό της λέξης (h)ard, δίπλα στο όνομα του προβλήματος. Στα hard κατατάσσονται τα προβλήματα, τα όποια από το EspressoMV θεωρούνται ως cyclic και "χωρίς λύση" (δεν μπορούν να επιλυθούν από το EspressoMV)

Κατηγοριοποίηση	Περιγραφή
trivial	Η ελάχιστη λύση επιτυγχάνεται εύκολα και περιλαμβάνει μόνο τους essential prime implicants
non-cyclic	Δεν υπάρχει κύκλος στον πίνακα των prime implicants και η λύση αποτελείται μόνο από essentials
cyclic	Η ελάχιστη λύση περιλαμβάνει κύκλο και είναι δυσκόλοτερη η επίλυση

Πίνακας 1. Κατηγοριοποίηση των κυκλωμάτων ως προς την δυσκολία ελαχιστοποίησης.

6.2.2. Σύγκριση αποτελεσμάτων exact αλγορίθμων

Η σύγκριση που ακολουθεί έχει γίνει μεταξύ των αποτελεσμάτων που παίρνουμε με την εφαρμογή του κώδικα μας στα υπο δοκιμή κυκλώματα και των αποτελεσμάτων που παίρνουμε με χρήση του εργαλείου EspressoMV σε λειτουργία α)exact και β)heuristic.

Η διαδικασία που ακολουθείται είναι η ίδια και στις τρεις περιπτώσεις. Αρχικά, γίνεται προσπάθεια εύρεσης όλων των prime implicants και ακολούθως να βρούμε ένα ελάχιστο υποσύνολο του συνόλου των prime implicants. Για την διαδικασία αυτή αφού εφαρμοστούν κάποιοι κανόνες ελαχιστοποίησης και καταλήξουμε σε κάποιο κυκλικό πίνακα prime implicants εφαρμόζεται ένας greedy αλγόριθμος επιλογής του prime implicant που καλύπτει τους περισσότερους όρους.

Το προβλήμα εύρεσης ελάχιστου καλύμματος είναι ένα NP-complete πρόβλημα. Έτσι, σε τέτοιου είδους προβλήματα ένας greedy αλγόριθμος δεν μπορεί να δώσει την βέλτιστη λύση, παρά μόνο μια τοπικά βέλτιστη λύση. Αντίθετα, το EspressoMV χρησιμοποιεί πιο πολύπλοκους αναδρομικούς Branch-and-Bound αλγόριθμους που εξετάζουν διεξοδικά ολόκληρο το χώρο των λύσεων καταλήγοντας έτσι σε καλύτερη λύση σε σχέση με κάποιο greedy αλγόριθμο, καταναλώνοντας όμως περισσότερο χρόνο.

Στους πίνακες παρακάτω μπορούμε να δούμε τα αποτελέσματα συγκεντρωτικά ώστε να γίνουν οι απαραίτητες συγκρίσεις. Ο πίνακας 4.α. αφορά σε κυκλώματα, τα οποία έχουν μία έξοδο και η συνάρτηση τους είναι πλήρως ορισμένη. Στον πίνακα 4.β. έχουμε κυκλώματα, τα οποία έχουν πολλαπλή έξοδο και είναι πλήρως ορισμένα, ενώ στους 5.α. και 5.β. έχουμε packed κυκλώματα, τα οποια είναι μιας και πολλαπλών εξόδων αντίστοιχα.

Στον πίνακα 2 παρατηρούμε ότι το Exact Minimizer που χρησιμοποιήσαμε κατάφερε με επιτυχία να λύσει όλα τα παραδειγμάτα τα οποία λύνει το EspressoMV για όλες τις κατηγορίες κυκλωμάτων.

Είδος προβλήματος	**Αριθμός Παραδειγμάτων**	**Exact Minimizer (solved)**	**EspressoMV (solved)**
trivial	5	5	5
non-cyclic	34	34	34
cyclic	17	17	17
cyclic (hard)	4	4	4
Συνολικά	**60**	**60**	**60**

Πίνακας 2. Αριθμός παραδειγμάτων που ελαχιστοποιήθηκαν ανα κατηγορία

Ένα πρώτο συμπέρασμα που μπορούμε να βγάλουμε από τους πίνακες είναι ότι η πολυπλοκότητα επίλυσης ως προς τον χρόνο των προβλημάτων είναι

ανεξάρτητη του αριθμού εισόδων και του αριθμού εξόδων του κυκλώματος, αλλά εξαρτάται άμεσα από τον αρχικό αριθμό των cubes και αυτό γιατί για τα αρχικά cubes έχουμε εξαντλητική επεξεργασία των cubes ανα δύο ώστε να συνδυαστούν σε ομάδες και να δημιουργήσουν prime implicants.

Αναλύοντας τους πίνακες των αποτελεσμάτων παρατηρούμε ότι το Exact Minimizer δίνει ακριβώς το ίδιο αποτελέσμα σε αριθμό cubes όπως και το EspressoMV, το οποίο είναι και το ελάχιστο που μπορεί να επιτευχθεί, για όλα τα προβλήματα τύπου trivial και non-cyclic. Η λύση που προκύπτει βασίζεται αποκλειστικά στους ίδιους κανόνες εξαγωγής των essential prime implicants και δεν έχει περιθώρια να δώσει διαφορετικά αποτελέσματα.

Για τα προβλήματα τύπου cyclic, η διαδικασία εξαρτάται σε μεγάλο βαθμό από τον αλγόριθμο εύρεσης ελάχιστου καλύμματος για την επίλυση του κύκλου. Εξαιτίας της φύσης των greedy αλγορίθμων ένα ολικά ελάχιστο αποτέλεσμα δεν είναι εφικτό, σε αντίθεση με τον Branch-and-Bound αλγόριθμο που χρησιμοποιεί το EspressoMV. Σε αυτό το κομμάτι λοιπόν οφείλονται οι μικρές διαφοροποιήσεις στα αποτελέσματα.

Σε κάποια κυκλώματα, όπως το **m2** δεν υπάρχει ουσιαστική διαφορά αφού και τα δύο εργαλεία δίνουν το ίδιο ελαχιστοποιημένο αποτελέσμα, 47 cubes που είναι και το ολικά ελάχιστο. Επίσης, για το single-output κύκλωμα **sym10**, το αποτέλεσμα που παίρνουμε είναι 210 cubes το οποίο και πάλι ταυτίζεται με τα το αποτέλεσμα του EspressoMV αλλά και με το ολικά ελάχιστο. Στο κύκλωμα **dist** παρατηρουμε ότι η ελάχιστη λύση είναι 120 cubes την οποία και το EspressoMV επιτυγχάνει, ενώ το Exact Minimizer δίνει μια λύση με 121 cubes, δηλαδή έχουμε μια απόκλιση της τάξης του 0,8%. Παρόμοια είναι και τα αποτελέσματα και για το **root** όπου η απόκλιση είναι της τάξης του 1,7% από το βέλτιστο. Στο άλλο άκρο έχουμε τα κυκλώματα **max128, lin.rom** και **prom2**, τα οποία ανήκουν στα cyclic hard προβλήματα και αντίστοιχα από την βιβλιογραφία του EspressoMV στα cyclic unsolved, δηλαδή αυτά που δεν επιλύθηκαν. Στα κυκλώματα η απόκλιση από την βέλτιστη λύση φτάνει στο 10,2%, 12,9% και 15,5% αντίστοιχα, με το τελευταίο να αποτελεί την χειροτερη περίπτωση με διαφορά. Σημειώνεται ότι το EspressoMV σε λειτουργία exact δεν κατάφερε να δώσει αποτέλεσμα σε 10 ώρες λειτουργίας για τα παραπάνω hard παραδείγματα, την στιγμη που το Exact Minimizer έδωσε.

Η μέση απόκλιση (*πίνακας 2*) για τα cyclic κυκλώματα είναι 3,22% σε σχέση με τα αποτελέσματα του EspressoMV και της τάξης του 4,4% από την βέλτιστη ολικά λύση. Αντίστοιχα για το σύνολο των cyclic και hard είναι 10% και 10,6%, ενώ λαμβανοντας υπόψιν τα cyclic μαζί με τα hard έχουμε απόκλιση 4,3% και 5,25%. Στο σύνολο όμων των κυκλωμάτων που χρησιμοποιήθηκαν για τον έλεγχο η απόκλιση από τα αποτελέσματα του Espresso MV έφτασε το 1,8% και για από το ολικά βέλτιστο το 2,27%.Σημειώνεται ότι το EspressoMV σε λειτουργία exact δεν κατάφερε να δώσει αποτέλεσμα σε 10 ώρες λειτουργίας, ενώ το Exact Minimizer έδωσε.

Στον πίνακα 3 μπορούμε να δούμε κάποια στατιστικά σε σχέση με τον αριθμό τον κυκλωμάτων τα οποία δίνουν την ίδια ελαχιστοποίηση ανά κατηγορία και τις αποκλίσεις σε περίπτωση που διαφέρουν. Έτσι στην 2η στήλη έχουμε τον αριθμό των κυκλωμάτων της κατηγορίας και στην 3η τον αριθμό των κυκλωμάτων που δίνουν ακριβως τον ίδιο αριθμό cubes με το EspressoMV, ενώ στην 4η στήλη έχουμε την μέση απόκλιση σε αριθμό cubes από το αποτέλεσμα του EspressoMV και από την βέλτιστη ελαχιστοποίηση[5].

Είδος Προβλήματος	**Συνολικός Αριθμός Κυκλωμάτων**	**Αριθμός Κυκλωμάτων ταυτίζονται**	**% Απόκλιση**	
			EspressoMV	**Βέλτιστο**
trivial	3	3	0	0
non-cyclic	21	21	0	0
cyclic	15	3	3,22	4,4
cyclic (hard)	4	0	10	10,6
Συνολικα cyclic			4,3	6,6
Συνολικά	43	26	1,8	2,27

Πίνακας 3. Αποκλίση από το το ελάχιστο και το EspressoMV

Τα αποτελέσματα όλων των κυκλωμάτων επαληθεύτηκαν με ανατροφοδότηση του ελαχιστοποιημένου κυκλώματος στο εργαλείο fraig_verify και sat_verify του MVSIS και σύγκριση τους με το αρχικό κύκλωμα. Η επαλήθευση ήταν επιτυχής για όλα τα κυκλώματα που δοκιμάστηκαν.

Multiple Output									
Circuit	Problem Type	In/Out	Terms	Init Cubes	Primes	V1 Min	MV E	MV H	Min
br1	Indust-noncyclic	(12/8)	34	116	29	19	19	19	19
br2	Indust-noncyclic	(12/8)	35	125	27	13	13	13	13
dist	Math-cyclic	(8/5)	255	591	401	121	120	120	120
lin.rom	Indust_cyclic(h)	(7/36)	128	2306	1087	147	128	128	
m1	Indust-noncyclic	(6/12)	32	218	59	19	19	19	19
m2	Indust-cyclic	(8/16)	96	831	243	47	47	47	47
m3	Indust-cyclic	(8/16)	128	1105	344	64	63	65	62
m4	Indust-cyclic	(8/16)	256	2183	687	109	104	107	101
max128	Indust-cyclic	(7/24)	128	1616	469	88	79	82	78
max512	Indust-cyclic	(9/6)	512	1616	535	141	137	142	133
max1024	Indust-cyclic(h)	(10/6)	1024	3232	1278	288	267	274	267
mlp4	Math-cyclic	(8/8)	225	678	606	131	127	128	121
myMulti3_2	noncyclic	(3/2)	6	8	5	3	3	3	3
myMulti4_4	noncyclic	(4/4)	11	21	8	5	5	5	5
newapla2	Indust-noncyclic	(6/7)	7	7	7	7	7	7	7
newbyte	Indust-noncyclic	(5/8)	8	8	8	8	8	8	
newcwpExp	Indust-noncyclic	(4/5)	28	44	23	11	11	11	11
p82	Indust-noncyclic	(5/14)	24	81	48	21	21	21	21
pope.rom	Indust-cyclic	(6/48)	64	1616	593	62	59	62	59
prom1	Indust-noncyclic	(9/40)	502	8306	9326	472	472	472	472
prom2	Indust-cyclic(h)	(9/21)	287	3027	2635	340	287	287	287
rd53	Math-noncyclic	(5/3)	31	42	51	31	31	31	31
rd84	Math-cyclic	(8/4)	255	411	633	255			
root	Math-cyclic	(8/5)	255	615	152	58	57	57	57

sqn	Indust-noncyclic	(6/3)	84	144	75	**38**	38	38	38
sqr6	Math-cyclic	(6/12)	63	259	205	**52**	49	49	47
squar5	Math-cyclic	(5/8)	30	85	71	**25**			
tcheck	noncyclic	(3/3)	3	6	5	**3**	3	3	
z4	Math-noncyclic	(7/4)	127	256	167	**59**	59	59	59
z5xp1	Math-cyclic	(7/10)	128	576	390	**66**	63	64	63
2bit Adder	Indust-noncyclic	(4/3)	15	22	17	11	11	11	11

Πίνακας 4.α. Αποτελέσματα κυκλωμάτων πολλαπλών εξόδων (full)

Multiple Output									
Circuit	**Problem Type**	**In/Out**	**Terms**	**Init Cubes**	**Primes**	**V1_Min**	**MV_E**	**MV_H**	**Min**
b12	Indust-cyclic	(15/9)	431	454	125	**86**	41	41/42	41
clpl	Indust-trivial	(11/5)	20	20	20	**20**	20	20/20	20
con1	noncyclic	(7/2)	9	9	9	**9**			
dc1	Indust-noncyclic	(4/7)	15	25	15	**25**	9	9	9
dc2	Indust-noncyclic	(4/7)	58	85	49	**67**	39	39	39
f51m	Math-cyclic	(8/8)	255	1024	561	**78**	76	76/77	76
newcwp	Indust-noncyclic	(4/5)	11	19	15	**19**	11	11	11
newtpla	Indust-noncyclic		4	4	5	**4**	4	4	4
newxcpla1	Indust-noncyclic	(9/23)	43	97		**97**	39	39/39	39
risc	Indust-noncyclic	(8/31)	74	130		**87**	28	28/28	28
t3	Indust-noncyclic	(12/8)	148		42	**33**	33	33	33
vtx1	Indust-noncyclic	(27/6)	110	110	110	**110**	110	110/110	110
x9dn	Indust-noncyclic	(27/7)	120	120	120	120	120	120/120	120

Πίνακας 4.β. Αποτελέσματα κυκλωμάτων πολλαπλών εξόδων (packed)

Single Output									
Circuit	Problem Type	In/Out	Terms	Primes	Ess.	V1_Min	MV_E	MV_H	Min
check	noncyclic	(4/1)	7	2	1	1			1
check2	noncyclic	(4/1)	10	2	1	1			1
life	Math-noncyclic	(9/1)	140	224	84	84	84	84	84
max46	Indust-trivial	(9/1)	46	49	46	46	46	46	46
sym10	Math-cyclic	(10/1)	837	3150	0	210	210	240	210
test_1	noncyclic	(4/1)	10			2	2		2
test_2	noncyclic	(4/1)	12			3	3		3
test_3	noncyclic	(4/1)	13			4	4		4
test_4	noncyclic	(4/1)	9			4	4		4
test_5	noncyclic	(5/1)	20			8	8		8
xor5	Indust-trivial	(5/1)	16	16	16	16	16	16	16
z9sym	Math-cyclic(h)	(9/1)	420	1680	0	90	85	84	84

Πίνακας 5.α. Αποτελέσματα κυκλωμάτων απλής εξόδου (full)

Single Output									
Circuit	Problem Type	In/Out	Terms	Primes	Ess.	V1_Min	MV_E	MV_H	Min
max46	Indust-trivial	(9/1)	46	49	46	46	46	46	46
ryy6	Indust-trivial	(15/1)	112	112	112	112	112	112	112
newill	Indust-cyclic	(8/1)	8	8	8	8	8	8	8

Πίνακας 5.β. Αποτελέσματα κυκλωμάτων απλής εξόδου (packed)

Βιβλιογραφία

[1] Giovanni De Micheli, *Synthesis and optimization of digital circuits*, McGraw-Hill, 1994

[2]G.Hachtel, F. Somenzi, *Logic Synthesis and Verification algorithms*, Springer, 2006

[3] R. Brayton, G. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli., *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, 1984.

[4] Robert K. Brayton, *Logic Synthesis for Hardware Systems*, Lecture Notes, Lecture 3, Berkeley, USA (1998)

[5] R. L. Rudell., *Logic Synthesis for VLSI Design*, PhD thesis, Department of Electrical Engineering and Computer Science, University of California at Berkeley, 1989.

[6] E. J. McCluskey, *Minimization of Boolean functions*, *Bell System Technical Journal*, 1956.

[7] W. Quine. The problem of simplifying truth functions, *American Mathematical Monthly*, 1952.

[8] http://en.wikipedia.org/wiki/Logic_synthesis

[9]http://en.wikipedia.org/wiki/High-level_synthesis

[10] http://en.wikipedia.org/wiki/Petrick%27s_method

[11] M. Dagenais, V. Agarwal, and N. Rumin. *Mcboole: a new procedure for exact logic minimization, IEEE Transactions on Computer-Aided Design*, 5(1):229–238, January 1986.

[12]McGeer, P. et al.: *ESPRESSO-SIGNATURE: A new exact minimizer for logic functions*. Proc. DAC'93

[13] Soha Hassoun and Tsutomu Sasao, *Logic Synthesis And Verification*, Kluwer Academic Publishers, USA, 2002

[14]Zvi Kohavi, *Switching and finite automata theory*, McGraw-Hill, 1978

[15]Hill F., Peterson G., *Switching theory and logical design*, Wiley&Sons, 1976

[16]Berkeley Logic Interchange Format (BLIF),University of California Berkeley, 2005