ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

03/06/2021

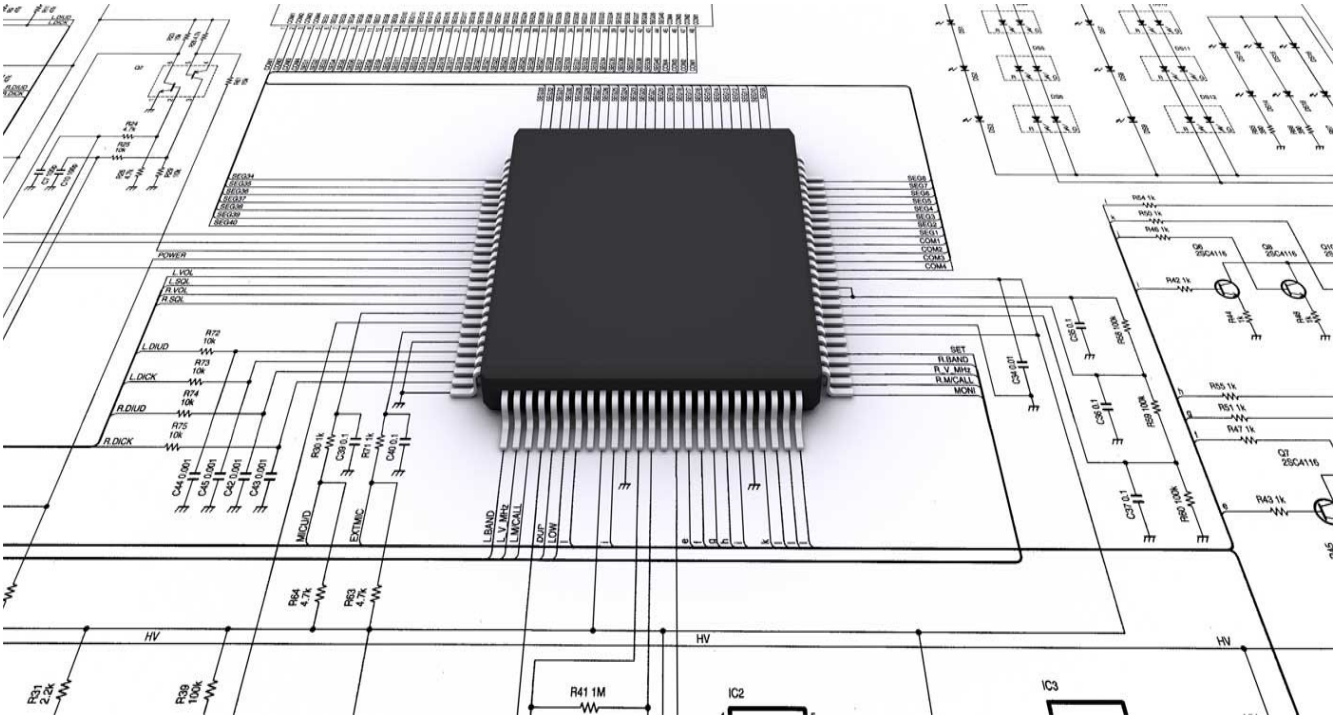
ΗΡΥ608/419-ΑΝΑΠΤΥΞΗ ΕΡΓΑΛΕΙΩΝ CAD ΓΙΑ ΣΧΕΔΙΑΣΗ ΟΛΟΚΛΗΡΩΜΕΝΩΝ ΚΥΚΛΩΜΑΤΩΝ

ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ 2021

ΑΣΚΗΣΗ 6
ΣΤΟΙΧΕΙΑ MODEL CHECKING ΚΑΙ FORMAL VERIFICATION: ΟΜΟΜΟΡΦΙΚΟΙ ΓΡΑΦΟΙ

ΑΝΑΦΟΡΑ

Μυλωνάκης
Εμμανουήλ
2015030079



ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Υλοποίηση - Περιγραφή.

Για την έκτη εργαστηριακή άσκηση κληθήκαμε να υλοποιήσουμε μια απλή μορφή formal verification. Έχοντας την ορθή συνδεσμολογία ενός κυκλώματος αποθηκευμένη σε ένα αρχείο, αυτή, πρέπει να συγκριθεί με την συνδεσμολογία του κυκλώματος υπό εξέταση, ώστε να αποφανθούμε αν τα δύο είναι ισοδύναμα ή όχι. Η βασική ιδέα είναι ότι, εφόσον τα κυκλώματα περιγράφονται υπό μορφή κατευθυντικού άκυκλου γράφου, αναζητόντας μέγιστους υπογράφους δηλαδή μονοπάτια από την πηγή έως το προορισμό και συγκρίνοντας τις αντίστοιχες διαδρομές ως προς τον χρωματισμό των κόμβων (δηλαδή το είδος της πύλης που διασχίζουμε) καταλήγουμε στο επιθυμητό συμπέρασμα.

Ως δομές δεμομένων χρησιμοποιήθηκαν: ένας τρισδιάστατος πίνακας εν ονόματι DAG στον οποίο αποθηκεύονται οι πληροφορίες των δύο κυκλωμάτων προς σύγκριση, ένας τρισδιάστατος πίνακας στον οποίο αποθηκεύεται μία λίστα για τα μονοπάτια (list_of_paths), ένας αντίστοιχος πίνακας με τον list_of_paths με όνομα str_path στο οποίο βρίσκονται τα μονοπάτια υπό κωδικοπιημένη μορφή σε string και τέλος, ο πίνακας equal_nodes για τους ισοδύναμους κόμβους των δύο κυκλωμάτων. Πιο αναλυτικά:

- **int DAG[2][#Gates][5]**

Μπορούμε να φανταστούμε την δομή αυτή πως αποτελείται από δύο στρώματα (layers) πινακών των δύο διαστάσεων. Στο πρώτο layer, δηλαδή στον πρώτο δισδιάστατο πίνακα, έχουμε πληροφορίες του Netlist για την ορθή υλοποίηση του κυκλώματος και στο δεύτερο layer του κυκλώματος υπό εξέταση. Κάθε ένας από τους δισδιάστατους πίνακες αποτελείται από πλήθος γραμμών ίσο με το πλήθος πυλών του κυκλώματος και σε κάθε στήλη αποθηκεύεται:

- **1η Στήλη:** Ο αριθμός Χ της πύλης που αναγράφεται στο netlist ως GX.
- **2η Στήλη:** Οι χαρακτήρες 'O', 'A' και 'N' για το αν πρόκειται για πύλη NOR_2, NAND_2 και NOT αντίστοιχα. Με άλλα λόγια ο χρωματισμός του κόμβου.
- **3η Στήλη:** Ο αριθμός του κόμβου της πρώτης εισόδου της πύλης που αναγράφεται στο netlist.
- **4η Στήλη:** Ο αριθμός του κόμβου της δεύτερης εισόδου της πύλης που αναγράφεται στο netlist. Στην περίπτωση πύλης NOT αποθηκεύεται η τιμή -1.
- **5η Στήλη:** Ο αριθμός του κόμβου για την έξοδο της πύλης που αναγράφεται στο netlist.

Η δεύτερη διάσταση του πίνακα ([#Gates]) δημιουργείται δυναμικά με βάση το πλήθος πυλών σε κάθε αρχείο. Επιπλέον, έχουμε τον πίνακα numOfGates[2] που αποθηκεύεται το πλήθος πυλών του κάθε κυκλώματος.

- **int list_of_paths[2][1024][1024] <=> char str_paths[2][1024][1024].**

Ο ακέραιος πίνακας list_of_paths κρατάει μία λίστα για όλα τα μονοπάτια του γράφου αποθηκεύοντας τις θέσεις των κόμβων ως δείκτες στον πίνακα DAG. Έχοντας του δείκτες αυτούς για τον πίνακα DAG μπορούμε να ανακτήσουμε εύκολα όλες τις πληροφορίες επί των κόμβων, ακμών και χρωμάτων. Ο πίνακας str_path βρίσκεται σε αντιστοιχία με την λίστα των μονοπατιών και έχει αποθηκευμένα υπο μορφή string όλα τα μονοπάτια. Για παράδειγμα, μία όδευση των πυλών NAND_2->NAND_2->NOT->NOR_2, κωδικοποιείται ως "AANO", με τους χαρακτήρες-χρώματα όπως έχουν αποθηκευτεί στον πίνακα DAG. Είναι τρισδιάστατοι καθώς ακολουθούν την ίδια λογική με τον πίνακα DAG όσον αφορά τα layers. Είναι σταστική και αρκετά μεγάλη για τις απαιτήσεις της εργαστηριακής άσκησης. Σε έναν πίνακα num_of_paths[2] γνωρίζουμε το πλήθος των μονοπατιών για κάθε κύκλωμα ενώ, κάθε μονοπάτι τερματίζει με τον κόμβο υπ'αριθμών -1 ώστε να μην αποθηκεύονται και να προσμετρούνται όλα τα μήκοι όλων των μονοπατιών.

- **int equal_nodes[2][1024]**

Δύο παράλληλοι μονοδιάστατοι πίνακες στους οποίους αποθηκεύονται οι ισοδύναμοι κόμβοι των δύο κυκλωμάτων. Πιο αναλυτικά, αποθηκεύονται οι θέσεις των θέσεων στον πίνακα DAG και για παράδειγμα γνωρίζουμε ότι, equal_nodes[0][4]=5 <=> equal_nodes[1][4]=1, δηλάδη ο κόμβος στην θέση 5 του ορθού κυκλώματος είναι ισοδύναμος με τον κόμβο στην θέση 1 του κυκλώματος υπό εξέταση. Οι τιμές 5, 1 είναι θέσεις στον πίνακα DAG, άρα γράφωντας DAG[0][5][X] και DAG[1][1][Y] με X,Y δείκτες στην στήλη που επιθυμούμε, λαμβάνουμε χρώμα κόμβου, ακμές και αριθμό πύλης για τα δύο κυκλώματα, με αυτά να βρίσκονται σε ισοδυναμία.

<u>Διαμόρφωση των παραπάνω δομών σύμφωνα με δύο απλά netlists.</u>

Ορθό κύκλωμα.

```
## LIBRARY
GATES.LIB
## RAILS
## INPUTS
1,2
## OUTPUTS
5
## NETLIST
G1 NOR_2 ; IN 1, 2 ; OUT 3
G2 NOR_2 ; IN 1, 2 ; OUT 4
G3 NAND_2 ; IN 3, 4 ; OUT 5
```

Υπό εξέταση κύκλωμα.

```
## LIBRARY
GATES.LIB
## RAILS
## INPUTS
11,12
## OUTPUTS
15
## NETLIST
G1 NOR_2 ; IN 11, 12 ; OUT 14
G2 NAND_2 ; IN 13, 14 ; OUT 15
G3 NOR_2 ; IN 11, 12 ; OUT 13
```

Τα παραπάνω κυκλώματα είναι λογικά ισοδύναμα. Ως εναρκτήριοι κόμβοι θεωρούνται οι είσοδοι του κυκλώματος και ως τερματικοί κόμβοι οι έξοδοί του.

Πίνακας DAG

Index ↓	Layer 1					Layer 2				
0	1	'O'	1	2	3	1	'O'	11	12	14
1	2	'O'	1	2	4	2	'A'	13	14	15
2	3	'A'	3	4	5	3	'O'	11	12	13

numOfGates[0] = 3 numOfGates[1] = 3

Πίνακας list_of_paths

Index ↓	Layer 1			Layer 2		
0	0	2	-1	0	1	-1
1	0	2	-1	0	1	-1
2	1	2	-1	2	1	-1
3	1	2	-1	2	1	-1

Πίνακας str_paths

Layer 1	Layer 2
"OA"	"OA"
"OA"	"OA"
"OA"	"OA"
"OA"	"OA"

Ο πίνακας **list_of_paths** όπως προαναφέρθηκε αποθηκεύει τις θέσεις για τον πίνακα DAG. Βλέποντας στο layer 1 του πίνακα list_of_paths, η πρώτη διαδρομή είναι (0) -> (2). Χρησιμοποιώντας την ως index για τον πίνακα DAG στο layer 1 αντίστοιχα, η διαδρομή αντιστοιχή σε (G1 NOR_2) -> (G3 NAND_2) και αυτή στον πίνακα str_path έχει κωδικοποιηθεί ως "OA" (για index=0, Layer 1). Όμως, η διαδρομή (0)->(2) εμφανίζεται δύο φορές. Αυτό γιατί και οι δυο είσοδοι της πύλης G1 του ορθού netlist έρχονται από διαφορετική είσοδο κυκλώματος. Άρα, αυτό το μονοπάτι το χρειαζόμαστε δύο φορές, καθώς αν στο αρχείο υπό εξέταση στην αντίστοιχη πύλη μόνο η μία είσοδός της αποτελούσε είσοδο κυκλώματος, τότε, στο layer 2 του list_of_paths θα είχαμε αποτυπώση μία λιγότερη διαδρομή, συνεπώς, θα είμαστε σε θέση να αποφανθούμε ότι δεν πρόκειται για δύο ισοδύναμα κυκλώματα. Στην περίπτωση όπου οι είσοδοι μιας πύλης είναι βραχυκυκλωμένοι (δηλαδή G1 NOR_2; 1,1 ; 5) και αποτελούν είσοδο κυκλώματος τότε δημιουργούμε μόνο ένα μονοπάτι για τον κόμβο αυτό. Αυτή η λεπτομέρεια είναι πολύ σημαντική, διότι αν στο υπο εξέταση αρχείο, αντίστροφα με το προηγούμενο παράδειγμα, έχουμε διαφορετικές εισόδους σε μία πύλη που είναι είσοδοι κυκλώματος, αλλά στο ορθό αρχείο αυτές είναι βραχυκυκλωμένες τότε πάλι θα έχουμε διαφορετικό πλήθος μονοπατιών αρά δεν θα έχουμε ισοδυναμία γράφων. Επι της ουσίας ο έλεγχος των μονοπατιών γίνεται ένα προς ένα, δηλαδή όσο πλήθος μονοπατιών έχουμε στο ένα κύκλωμα αλλά τόσο και στο δεύτερο, και επιπλέον, όσες εμφανίσεις του ίδιου μονοπατιού στο ένα τόσες και στο άλλο.

Πίνακας equal_nodes

Index ↓	Layer 1	Layer 2
0	0	0
1	1	2
2	2	1

Ο πίνακας αυτός ενημερώνεται στον τελικό βήμα του αλγορίθμου όπου ελέγχεται ισότητα μεταξύ των string μονοπατιών. Βρίσκοντας το ισοδύναμο μονοπάτι, χρησιμοποιούντε τα indexes που αναλύθηκαν προηγουμένος για την εύρεση των ισοδύναμων κόμβων. Ενδιαφέρον έχει εδώ ότι, αυτός ο πίνακας μπορεί να

απεικονιστεί με πολλές ισοδύναμες παραλλαγές. Βλέπωντας την σειρά του πίνακα για index=1, έχουμε τις τιμές 1 και 2 αντίστοιχα. Αντικαθιστώντας τις τιμές αυτές για να κατευθυνθούμε στον πίνακα DAG, έχουμε ότι η πύλη G2 του ορθού αρχείου είναι ισοδύναμη με την πύλη G3 του υπό εξέταση αρχείου. Όμως, λογικά ισοδύναμη πύλη με την G2 του ορθού αρχείου είναι και η G1 του υπο εξέταση. Και αυτή η περίπτωση είναι ορθή καθώς, πρόβλημά μας είναι η λογική ισοδυναμία και δεν ενδιαφερόμαστε για ονοματολογία εισόδων/εξόδων ή αντίστροφη συνδεσμολογία επί της ίδιας πύλης. Άρα από την στιγμή που δύο πύλες έχουν το ίδιο χρώμα και ανηκούν στο ίδιο μονοπάτι (ίδια αλληλουχία χρωμάτων) αποτελούν ισοδύναμους κόμβους. Με βάση αυτό, ο τρόπος κωδικοποίησης των μονοπατιών που περιγράφηκε προηγουμένος είναι αρκετός για τις διαστάσεις του προβλήματός μας, αρκεί να αναγνωρίστουμε όλα τα μονοπάτια ορθά, να ξεκινάμε πάντα από είσοδο κυκλώματος και να καταλήγουμε σε έξοδο, ώστε να μην έχουμε μικρούς υπογράφους αλλά μόνος του μέγιστους δυνατούς.

Ψευδοκώδικας

```
MAIN:
    open files and store infos;
    basic_checks();
    find_all_paths();
    final_check();


FUNC basic_check():
    Same number of inputs, otherwise exit(1);
    Same number of outputs, otherwise exit(1);
    Same number of gates, otherwise exit(1);
    Same number of gates per type, otherwise exit(1);
END FUNC

FUNC find_all_paths():
    path[] = 0;
    S[] = find_starting_point();
    for i in every starting_point in S[]
        DFS(S[i], path, p_index); //Depth-First Search;
END FUNC

FUNC DFS(node, path, p_index):

    if(node is destiostation)
    {
        path[p_index] = node; //push last node in the path
        path[p_index+1] = -1; //Terminal value.
        store path in list_of_paths and in str_path;
        return;
    }
```

```
    path[p_index] = node; //push current node in path

    NB[] = find_all_neighbors(node);

    for i in NB[]   // For every neighbor call recursion
    {
        p_index++; // increase path index in order to store the next.
        DFS(NB[i], path, p_index);
        path[p_index] = -1; //Pop last node as recursion returns from its last call.
        p_index--; //Decrease path index.
    }

    return;
END FUNC

FUNC final_check():

    boolean isEqual;
    for i in str_path[0] // This is for the golden standard
    {
        isEqual = false;
        for j in (str_path[1] and not already checked) // This is for file under examination
        {
            if(string_compare(str_path[0],str_path[1]) == EQUALS)
            {
                isEqual = True;
                update equal_nodes array;
                break; // Check for the next
            }
        }
    }
END FUNC
```

Ο αλγόριθμος που υλοποιήθηκε για την περάτωση του εργαστηρίου υπολογίζεται ότι έχει χρονική πολυπλοκότητα **O(n²).** Σειριακά εκτελούνται τα παρακάτω βήματα με την ακόλουθη σειρά:

1. Διάβασμα **2** αρχείων αποτελούμενα από **n** γραμμές (**n** επαναλήψεις) → **O(2*n) = O(n).**
2. Κάποιοι βασική ελέγχοι επί **n** πυλών με χρήση μίας επανάληψης. → **O(n)+Θ(k) = O(n).**
3. Χρήση αλγορίθμου DFS για εύρεση όλων των μονοπατιών για όλους τους αρχικούς κόμβους. Ο αλγόριθμος DFS για την εύρεση ενός μονοπατιού από πηγή σε προορισμό έχει πολυπλοκότητα **O(V+E)** όταν οι κόμβοι δεν επανα-επισκέπτονται, πράγμα που συμβαίνει καθώς έχουμε άκυκλο γράφο. Όμως εμείς κάνουμε εύρεση όλων των μονοπατιών για κάθε εναρκτήριο κόμβο που μας αλλάζει την πολυπλοκότητα σε **O(k*V*(V+E)),** όπου k το πλήθος των εισόδων του κυκλώματος. Ασυμπτωτικά παίρνουμε πολυπλοκότητα **O(V²)** καθώς το k, δηλαδή το πλήθος των εισόδων είναι αμελητέο σε σχέση με το **V²**. → **O(V²).**

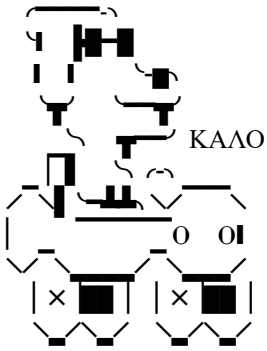
4. Μια διπλή επανάληψη με εμφώλευση για την σύγκρηση **n** μονοπατιών. → **O(n^2)**

Αθροίζοντας τις παραπάνω πολυπλοκότητες καταλήγουμε στο **O(n^2).**

Παραδοτέα.

Να σημειωθεί ότι οι αλλαγές των αρχείων γίνονται πάνω στο Correct.txt. Το Golden Standard παραμένει σταθερό.

- ✔ **FA_original.txt:** Ορθή υλοποίηση του ενός Full Adder. Golden Standard.
- ✔ **Correct.txt:** Υπο εξέταση αρχείο με σωστή υλοποιήση του Full Addar.
- ✔ **Wrong_0.txt:** Λάθος υλοποίηση FA. Μία λιγότερη πύλη από το Golden Standard.
- ✔ **Wrong_1.txt:** Λάθος υλοποίηση FA. Ίσο αριθμό πυλών. Πύλη G12 αλλαγμένοι σε NOR_2.
- ✔ **Wrong_2.txt:** Λάθος υλοποίηση FA. Διαφορετικό πλήθος εισόδων.
- ✔ **Wrong_3.txt:** Λάθος υλοποίηση FA. Διαφορετικό πλήθος εξόδων.
- ✔ **Wrong_4.txt:** Λάθος υλοποίηση FA. Πύλη G6 τερματικός κόμβος 92. Μετανομασία σε 91 που.
- ✔ **Wrong_5.txt:** Λάθος υλοποίηση FA. Αλλαγή στις εισόδους της πύλη G1. Από (12,22) σε (12,12). Οι κόμβοι 12, 22 είναι είσοδοι κυκλώματος.
- ✔ **Wrong_6.txt:** Λάθος υλοποίηση FA. Αλλαγή στην πύλη G7. Αλλάζει η όδευση **G2-NOR_2→G7-NAND_2** σε **G3-NAND_2→G7-NAND_2**. Αλλάζει δηλαδή η αλληλουχία χρωμάτων της όδευσης.
- ✔ **Wrong_7.txt:** Λάθος υλοποίηση FA. Αλλαγή στην πύλη G6. Αλλάζει η όδευση **G12-NAND_2→G6-NAND_2** σε **G7-NAND_2→G6-NAND_2**. Δεν αλλάζει η αλληλουχία χρωμάτων της όδευσης.



ΚΑΛΟ ΚΑΛΟΚΑΙΡΙ !