ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

27/03/2021

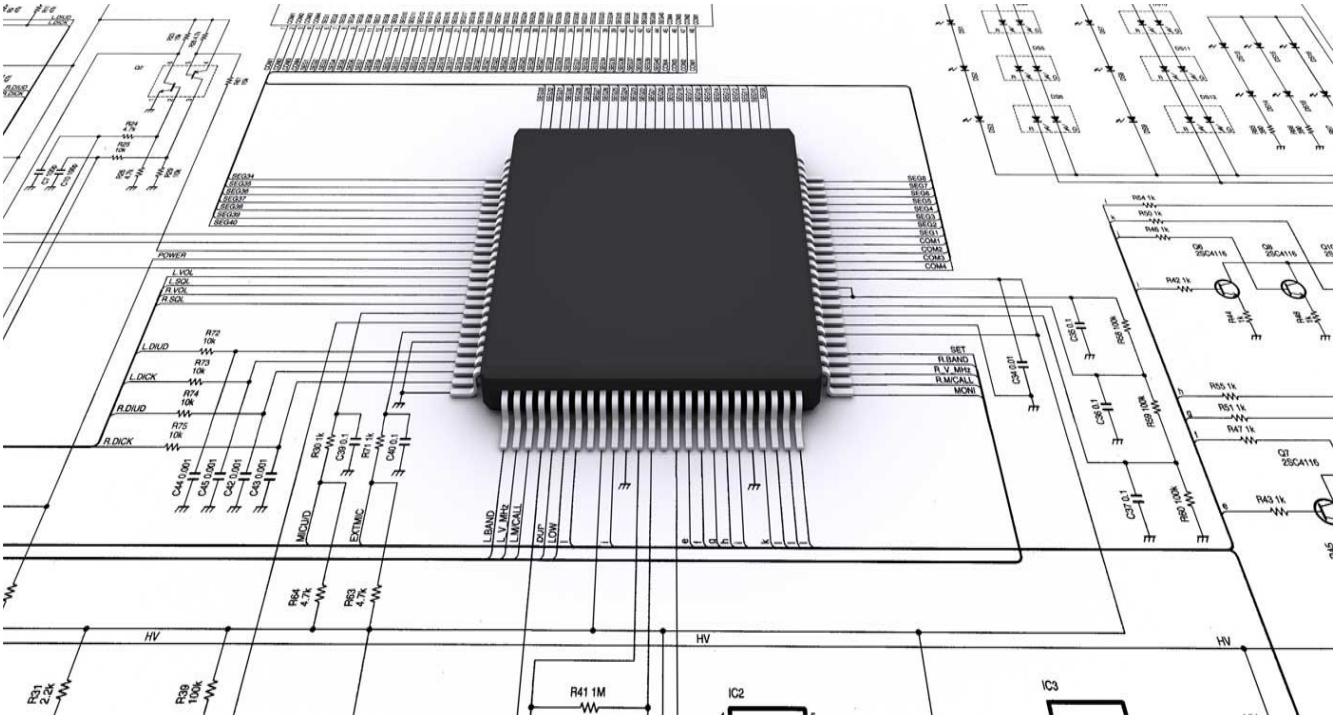
ΗΡΥ608/419-ΑΝΑΠΤΥΞΗ ΕΡΓΑΛΕΙΩΝ CAD ΓΙΑ ΣΧΕΔΙΑΣΗ ΟΛΟΚΛΗΡΩΜΕΝΩΝ ΚΥΚΛΩΜΑΤΩΝ

ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ 2021

ΑΣΚΗΣΗ 2
ΕΝΑΣ ΑΠΛΟΣ ΠΡΟΣΟΜΟΙΩΤΗΣ ON/OFF ΓΙΑ ΚΥΚΛΩΜΑΤΑ CMOS

ΑΝΑΦΟΡΑ

Μυλωνάκης
Εμμανουήλ
2015030079



Σκοπός-Περιγραφή

Σκοπός της δεύτερης εργαστηριακής άσκησης, είναι η υλοποίηση ενός απλού προσομοιωτή τρανζίστορ τύπου CMOS σε ψηφιακό επίπεδο. Η άσκηση πραγματώθηκε στην γλώσσα προγραμματισμού C μοντελοποιώντας το πρόβλημα με graph coloring. Δίνετε ένα αρχείο ως είσοδος του προγράμματος, το οποίο περιέχει την συνδεσμολογία του κυκλώματος υπό την μορφή Netlist, καθώς επίσης και τα αντίστοιχα testbench για την προσομοίωσή του. Οι πληροφορίες από το αρχείο αποθηκεύονται και διαχειρίζονται μέσω δομών δεδομένων, οι οποίες αξιοποιούνται κατά την προσομοίωση για την διεξαγωγή αποτελεσμάτων. Ο αλγόριθμος, πέραν από ορθά αποτελέσματα στην περίπτωση που το κύκλωμα είναι σωστά σχεδιασμένο, είναι ικανός να ανιχνεύει ανοιχτό κύκλωμα ή βραχυκύκλωμα στην έξοδο.

Υλοποίηση-Κώδικας

Αρχικά, θα αναφερθούμε στις δομές δεδομένων που χρησιμοποιήθηκαν και τον τρόπο διαχείρησής τους. Γίνεται χρήση μονοδιάστατων και δισδιάστατων πινάκων με δυναμική παραχώρηση μνήμης. Επιλέχθηκε δυναμική παραχώρηση μνήμης διότι, πρώτον, δεν μπορούμε να προβλέψουμε τον αριθμό των τρανζίστορ του κυκλώματος, επομένως κάνοντας μεγάλες στατικές δομές καταναλώνεται η μνήμη αδίκως, και δεύτερον, κατ'αυτόν τον τρόπο, ο κώδικα είναι ικανός να προσομοιώνει μεγάλα κυκλώματα πολυάριθμων τρανζίστορ. Στους μονοδιάστατους πίνακες αποθηκεύονται οι κόμβοι είσοδου, εξόδου, εισόδου δοκιμής, εξόδου δοκιμής και το διάνυσμα δοκιμής, ενώ σε μία απλή μεταβλητή ο κόμβος τάσης και γείωσης ξεχωριστά. Η δισδιάστατοι πίνακες αφορούν το netlist του κυκλώματος και τον γράφο, αυτά τα όποια παίζουν πρωταρχικό ρόλο στην προσομοίωση. Ο πίνακας για το netlist έχει σταθερό πλήθων στηλών ίσο με τέσσερα, ενώ οι γραμμές του δημιουργούνται δυναμικά ανάλογα με το πλήθος των τρανζίστορ. Στην πρώτη στήλη αποθηκεύεται η πληροφορία για το αν έιναι PMOS ή NMOS, και στις υπόλοιπες τρεις στήλες τα pins με την σειρά που αναγράφονται στο αρχείο. Ο δισδιάστατος πίνακας του γράφου έχει σταθερό πλήθος γραμμών ίσο με τέσσερα και πλήθος στηλών ίσο με τον αριθμό των κόμβων του κυκλώματος. Στην πρώτη γραμμή, αποθηκεύονται οι τιμές των κόμβων του κυκλώματος και στις εναπομείναντες τρεις το χρώμα των κόμβων σε αντιστοιχία κατά στήλες.

Αναλυτικότερα όσον αφορά τον γράφο. Η αρχικοποίησή του έχει μορφή που φαίνεται στον παρακάνω πίνακα.

5	8	4	2	3	9
1	X	X	X	X	0
X	X	X	X	X	X
X	X	X	X	X	X

Η πρώτη σειρά αφορά την τιμή του κόμβου όπως προαναφέρθηκε. Η δεύτερη σειρά έχει τα χρώματα της αρχικής του κατάστασης. Καταλαβαίνουμε από το παράδειγμα αυτό, ότι στον κόμβο '5' εφαρμόζεται η τάση και στον '9' η γείωση. Η δεύτερη γραμμή αποτελεί σημείο αναφοράς για την πρώτη επανάληψη του αλγορίθμου της προσομοίωσης. Οι γραμμές αυτές παραμένουν **απαράλλακτες** καθ' όλη την εκτέλεση του προγράμματος.

Αυτό γιατί, για κάθε νέο διάνυσμα εισόδου το κύκλωμα δεν αλλάζει και κατ'επέκταση ο γράφος παραμένει σταθερός. Έτσι οι πληροφορίες κάθε κόμβου αποθηκεύονται στις δύο τελευταίες σειρές κατά την εκτέλεση της προσομοίωσης, και για καινούργιο διάνυσμα εισόδου, ο γράφος γίνεται reset στην μορφή που απεικονίζεται.

Όταν το διάνυσμα εισόδου διαβαστεί από το αρχείο ξεκινάει αμέσως η προσομοίωση. Δύο δείκτες (με την έννοια του index, όχι pointer) αρχικοποιούνται πριν την πρώτη επανάληψη και δείχνουν στην γραμμή 2 και 3, τους οποίους θα ονομάσουμε info και update αντίστοιχα. Με τον info γνωρίζουμε από πιο γράφο διαβάζουμε πληροφορίες και τον update που τις αποθηκεύουμε. Διασχίζουμε όλο το netlist, αποθηκεύουμε ότι πληροφορία γνωρίζουμε (ο αλγόριθμο που θα αναλυθεί παρακάτω) και για την επόμενη διάσχυση του netlist, αυξάνουμε τους δείκτες αυτούς. Έπειτα, για τις επόμενες επαναλήψεις, οι δείκτες αυτοί γίνονται swap και έτσι πετυχαίνουμε ενημέρωση μόνος της τρίτης και τέταρτης γραμμής. Με αυτήν την τεχνική αποφεύγουμε αντιγραφές μνήμης για το λεγόμενο double buffering, καθώς επίσης πιθανές επανά-δεσμεύσεις στην μνήμη, εάν θέλουμε να επεκτείνουμε τον γράφο κατα γραμμές, μέχρις ότου να συγκλίνει. Πράγματα τα οποία έχουν αρνητική επίδραση στην ταχύτητα της προσομοίωσης.

Ως υπενθύμιση και για μια καλύτερη εικόνα, στον παρακάτω πίνακα δίνουμε ένα απλό παράδειγμα αποθήκευσης του netlist από το αρχείο. Έχουμε τον εξής πίνακα:

'p'	6	5	4
'P'	8	4	1
'N'	6	1	2
'N'	3	4	2

Αυτός ο πίνακας αφορά netlist που θα αναγραφώταν στο αρχείο ως εξής:
U1 PMOS 6 5 4
U2 PMOS 8 4 1
U3 NMOS 6 1 2
U4 NMOS 3 4 2

Έχωντας πλέον κατανοήσει τον τρόπο διαχείρησης των δομών δεδομένων, είμαστε έτοιμοι να εξηγήσουμε την λειτουργικότητα της προσωμοίωσης παρέα με ψήγματα κώδικα.

Εικόνα 1.

```
91   void simulate()
92   {
93       int info_index = 1; //From which graph
94       int update_index = 2; //In whicn graph
95       int converges = 0; //To terminate when
96       //First of all, we are using Graph[1] (
97       reset_graph(); //Reset graph for new si
98
99       while (converges < 2) // 2 stable state
100      {
101          for (int i = 0; i < numOfMos; i++)/
102          {
```

Αρχικά και όπως προαναφέρθηκε, με το που κληθεί η συνάρτηση simulate(), αρχικοποιούμαι τους δείκτες-indexes info_index και update_index, μια μεταβλητή converges ώστε να τερματίσει όταν η διαδικασία όταν ο γραφός έχει συγκλίνει και κάνουμε reset τον γράφο.

Εικόνα 2.

```
101  for (int i = 0; i < numOfMos; i++)//We are crossing netlist f
102  {
103      int gate = gate_value(MOS[i][1], info_index);//Info about
104
105      //Note: Sumbol S/D shorthand for Source/Drain.
106      int pin1_info = get_info_node(MOS[i][2], info_index);//La
107      int pin1_update = get_info_node(MOS[i][2], update_index);
108      int node1 = MOS[i][2];//Value of node.
109
110      //Same as above, but now, for the other transistor's pin
111      int pin2_info = get_info_node(MOS[i][3], info_index);
112      int pin2_update = get_info_node(MOS[i][3], update_index);
113      int node2 = MOS[i][3];
```

Διασχίζουμε μία φορά όλο το netlist, το οποίο είναι αποθηκευμένο στην μεταβλήτη **MOS**. Για κάθε τρανζίστορ του netlist αποθηκεύουμε την τιμή-χρώμα της πύλης, και για τα δύο pins τους, τις τιμές που είχαν στον προηγούμενο γράφο(pinX_info), τις τιμες που έχουν στον τωρινό γράφο και πρόκειται να αλλάξουν (pinX_update), καθώς επίσης και τους αριθμούς των αντίστοιχων κόμβων για τα pins αυτά.

Εικόνα 3.

```
123      if (node1 == VCC && pin1_update == 'X')
124      {
125          update_nodes_info(node1, update_index, 1);
126          pin1_update = 1;
127      }
128
129      if (node2 == VCC && pin2_update == 'X')
130      {
131          update_nodes_info(node2, update_index, 1);
132          pin2_update = 1;
133      }
134
135      if (node1 == GND && pin1_update == 'X')
136      {
137          update_nodes_info(node1, update_index, 0);
138          pin1_update = 0;
139      }
140
141      if (node2 == GND && pin2_update == 'X')
142      {
143          update_nodes_info(node2, update_index, 0);
144          pin2_update = 0;
145      } /*(1)*/
```

Στην συνέχεια ακολουθούν οι βασικοί έλεγχοι. Αν έχουμε κόμβο-τάσης ή κόμβο-γείωσης, τότε χρωματίζουμε με τις τιμές '1' ή '0' αντίστοιχα. Επιπλέον, αυτοί οι κόμβοι πρέπει να έχουν χρώμα 'X'. Αυτό διότι, πρώτον μόνο το 'X' αλλάζει χρώμα σε '1' ή '0', και δεύτερον, αν στο κύκλωμα έχουμε βραχυκύκλωμα στην έξοδο το οποίο οδεύει μέχρι την τάση ή την γείωση, τότε χρωματίζεται ο κόμβος τάσης ή γείωσης με 'SC'. Κατά την διεξαγωγή αποτελέσματος, οι κόμβοι τάσης-γείωσης έχουν χρώμα '1' και '0' αντίστοιχα όπως είναι και το ορθό. Ο χρωματισμός τους ως 'SC' είναι θέμα υλοποίησης του αλγορίθμου και συνεισφέρει στην σύγκλισή του. Έπειτα την ενημέρωση του κόμβου στον γράφο, ενημερώνεται και η αντίστοιχα μεταβλητή με την καινούργια τιμή, καθώς, πρώτον είναι χρήσιμη για την λογική που ακουλουθεί και δεύτερον, ο αλγόριθμος θα συγκλίνει

γρηγορότερα απ' το να έπαιρνε αυτήν την τιμή στην επόμενη επανάληψη.

Οι συναρτήσεις **update_node_info(int Node, int index, int color)** και **get_info_node(int Node, int index, int color)**, αναζητούν στην πρώτη γραμμή του πίνακα-γράφου τον κόμβο Node και στην αντίστοιχη στήλη ενημερώνουν/επιστρέφουν (update/get) το χρώμα στην γραμμή index. Επομένως, έχουμε μία εξωτερική επανάληψη που τρέχει για **n** τρανζίστορ και μία εμφωλευμένη επανάληψη που αναζητά σε **V** γράφους. Πράγμα που μας δίνει μία πολυπλοκότητα **O(n*V)** όπου για **n-->inf και V-->inf** μπορούμε να αποφανθούμε **O(V^2).** Έπειτα της παραπομπής αυτής συνεχίζουμε στον αλγόριθμο.

Εικόνα 4.

```
149  if ((MOS[i][0] == 'P' && gate == 0) || (MOS[i][0] == 'N' && gate == 1))
150  {
151      /* If we have set the output node with value Z and in the upcoming iterations
152         we find out that our transistor transfers the current between its pins,
153         we should restore that value with X.
154      */
155      if (is_node_output(node1) && pin1_update == 'Z')
156          pin1_update = 'X';
157
158      if (is_node_output(node2) && pin2_update == 'Z')
159          pin2_update = 'X';
160
161      /*Case 1 turns to 0. The pin we are about to update has the value 1
162        and for the other pin we have the information about colour '0'. */
163      if ((pin1_update == 1 && pin2_info == 0) || pin2_info == 'S')
164      {
165          update_nodes_info(node1, update_index, 'S');
166          pin1_update = 'S';
167      }
168      //Symmetric with the above one, but for the other pin.
169      if ((pin2_update == 1 && pin1_info == 0) || pin1_info == 'S')
170      {
171          update_nodes_info(node2, update_index, 'S');
172          pin2_update = 'S';
173      }
174      /*We update normally because of transistor's transfer ability.
175        The value we update have to has colour 'X' and nothing else.*/
176      if (pin1_update == 'X' && pin2_info != 'X' && pin2_info != 'S' && pin2_info != 'Z')
177          update_nodes_info(node1, update_index, pin2_info);
178      else if (pin2_update == 'X' && pin1_info != 'X' && pin1_info != 'S' && pin1_info != 'Z')
179          update_nodes_info(node2, update_index, pin1_info);
180  }
181
182  if ((MOS[i][0] == 'P' && gate == 1) || (MOS[i][0] == 'N' && gate == 0))
183  {   /*If the gate of PMOS transistor is 1 and the node is output
184       there is a posibility for OC at output.*/
185      if (is_node_output(node1) && pin1_update == 'X')
186          update_nodes_info(node1, update_index, 'Z');
187
188      if (is_node_output(node2) && pin2_update == 'X')
189          update_nodes_info(node2, update_index, 'Z');
190  }
191
```

Ελέγχουμε αν το τρανζίστορ μεταφέρει ρεύμα μεταξύ Source - Drain. Πρώτος έλεγχος, αν έχουμε κόμβο-εξόδου και έχει χρώμα 'Z' τότε αλλάζουμε το χρώμα που πρόκειται να ενημερωθεί σε 'X' (1). Προς το παρόν τον ξεχνάμε. Έπειτα έχουμε την μη επιθυμητή κατάσταση όπου το '1' γίνεται '0'. Ουσιαστικά ο αλγόριθμος λέει, αν ο κόμβος που ενημερώνεται έχει χρώμα '1' και η πληροφορία με την οποία θα ενημερωθεί είναι '0', τότε έχουμε βραχυκύκλωμα. Απόλυτα συμμετρικός ο επόμενος έλεγχος αλλά για το δεύτερο pin του transistor. Συνεχίζοντας, μεταφέρεται το χρώμα από το ένα pin στο άλλο μόνο στην περίπτωση όπου, αυτό που ενημερώνεται πρέπει να έχει χρώμα 'X' και η πληροφορία που γνωρίζω είναι αποκλειστικά '0' ή '1'.

Στην επόμενη συνθήκη ελέγχου, ο αλγόριθμος αναζητά για πιθανό ανοιχτοκύκλωμα στην έξοδο. Αν το τρανζίστορ δεν μεταφέρει ρεύμα μεταξύ Source-Drain, ο κόμβος αντιπροσωπεύει έξοδο κυκλώματος και δεν έχει ενημερωθεί τότε πρόκειται για **πιθανό** βραχυκύκλωμα. Η λέξη "πιθανό" τονίζεται, διότι πρόκειται για υπόθεση. Για αυτόν τον λόγο υπάρχει ο έλεγχος (1), που φαινόταν να μην βγάζει νοήμα, ώστε αν σε κάποια μελόντική επανάληψη μπορέσει ο αλγόριθμος να του δώσει τιμή, να το κάνει κιόλας. Ουσιαστικά, δεδομένου ότι ο αλγόριθμος βρίσκει σωστά τα βραχυκυκλώματα, αν η

έξοδος δεν πάρει τιμή αυτό οφείλεται σε ανοιχτοκύκλωμα. Επομένως, αφου συγκλίνει ο αλγόριθμος μπορούμε να ελέγξουμε αν η έξοδος έχει πάρει ή όχι τιμή ώστε να αποφανθούμε για ανοιχτοκύκλωμα. Κάνοντας όμως λίγο πιο πολύπλοκη την λογική μας και τις συνθήκες, γλιτώνουμε αυτό το έξτρα "πέρασμα" στο τέλος και κερδίζουμε σε χρόνο. Κλείνοντας, γίνεται εναλλαγή δεικτών όπως αναφέρθηκε, ελέγχεται αν ο γράφος συγκλίνει και τερματίζει.

Αποτελέσματα.

Στον φάκελο με την εργασία, περιέχονται αρχεία X_Correct.txt και X_Wrong.txt. Τα πρώτα αφορούν κυκλώματα με ορθή συνδεσμολογία και τα δεύτερα με λανθασμένη. Τα αρχεία που στο όνομά τους εχουν την συμβολοσειρά "ABC" αφορούν το κύκλωμα της εκφώνησης (AB+C)΄. Τα αρχεία NAND αφορούν την αντίστοιχη πύλη τρειών εισόδων, τέλος έχουμε αρχεία για την πολυ-αγαπημένη μου πύλη XOR.

Παρακάτω φαίνονται κάποια από τα αποτελέσματα που αφορούν το αρχείο XOR_Wrong.txt.

```
Input Vector = <1, 0>
NODE : Color
1 : 1 VCC
2 : X
3 : SC
4 : SC OUTPUT
5 : X
6 : SC
7 : 0 GND
8 : SC
9 : SC
10 : SC
11 : SC
```

```
Input Vector = <1, 1>
NODE : Color
1 : 1 VCC
2 : X
3 : 1
4 : Z OUTPUT
5 : 0
6 : X
7 : 0 GND
8 : 0
9 : 0
10 : 0
11 : 0
```

Παρατηρώντας το netlist βλέπουμε την εξής επικοινωνία μεταξύ των κόμβων: **(3)<-->(4)<-->(5)**.
Άρα σύμφωνα με τα αποτελέσματα, στην αριστερή εικόνα αναγνωρίσμουμε βραχυκύκλωμα μεταξύ κόμβων 4,5 που αφορά την συγκεκριμένη ακμή, και στην δεξιά φαίνεται η ασύνδετη έξοδος, δηλαδή ανοιχτοκύκλωμα στις ακμές 3,4 και 4,5.

Ανατρέξτε στα αρχεία Result για περισσότερες λεπτομέρειες.