ΗΥ-120: Ψηφιακή Σχεδίαση
Φθινόπωρο 2003

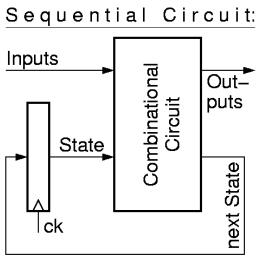
Τμ. Επ. Υπολογιστών
© Πανεπιστήμιο Κρήτης

Εργαστήριο 10:
Μηχανές Πεπερασμένων Καταστάσεων (Finite State Machines - FSM)

12 - 15 Ιανουαρίου 2004

[Βιβλίο: *προαιρετικά* μπορείτε να διαβάσετε μέρος της **§6.4** (σελ. 274-280) και σελίδες 304-305 (από την §6.7).]

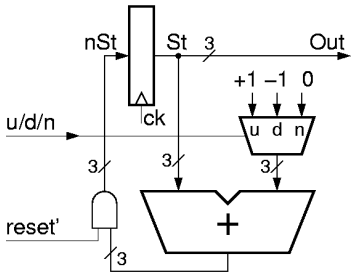
10.1 Ακολουθιακά Κυκλώματα:



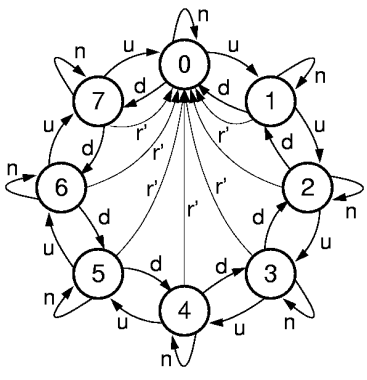
Έχουμε πει (§7.6) ότι "συνδυαστικά" (combinational) κυκλώματα λέμε αυτά που δεν έχουν μνήμη, άρα οι τιμές εξόδου τους εξαρτώνται μόνο από τις παρούσες τιμές των εισόδων τους και όχι από το παρελθόν. Αντίθετα, "ακολουθιακά" (sequential) κυκλώματα λέμε αυτά που έχουν μνήμη, κι έτσι η συμπεριφορά τους εξαρτάται όχι μόνο από το τι "βλέπουν" τώρα στις εισδους τους, αλλά και από την "**κατάσταση**" (state) στην οποία βρίσκονται, και στην οποία κατάσταση έχουν περιέλθει λόγω του παρελθόντος --στα ακολουθιακά κυκλώματα συνηθίζουμε να ονομάζουμε "κατάσταση" τις δυαδικές τιμές που είναι αποθηκευμένες στο σύνολο των στοιχείων μνήμης του κυκλώματος. Στο σχήμα δεξιά φαίνεται μιά γενική αναπαράσταση ενός ακολουθιακού κυκλώματος, η οποία δείχνει ότι αυτό αποτελείται (α) από τα στοιχεία μνήμης (flip-flops) που αποθηκεύουν την κατάσταση του κυκλώματος, και (β) από ένα συνδυαστικό κύκλωμα το οποίο προσδιορίζει: (β1) τις εξόδους σαν συναρτήσεις των εισόδων και της κατάστασης, και (β2) την επόμενη κατάσταση, πάλι σαν συνάρτηση των εισόδων και της παρούσας κατάστασης. Η κατάσταση αποτελεί, τρόπον τινά, μιά "περίληψη" όλου του παρελθόντος --σε κάθε σύστημα, γιά κάθε εφαρμογή, ο σχεδιαστής αποφασίζει πόσο μεγάλη και λεπτομερή τέτοια "περίληψη" χρειάζεται να κρατάμε.

Εδώ θα μελετήσουμε τις "**Μηχανές Πεπερασμένων Καταστάσεων** (Finite State Machines - **FSM**). Από μιάν άποψη, όλα τα ακολουθιακά κυκλώματα είναι μηχανές πεπερασμένων καταστάσεων, διότι, αν έχουν n bits μνήμης (κατάστασης), τότε βρίσκονται ανά πάσα στιγμή σε μιάν από τις 2^n διαφορετικές καταστάσεις τις οποίες διαθέτουν --οι οποίες και είναι πάντα πεπερασμένες. Όμως, συνηθίζουμε να αναφερόμαστε με τον όρο "FSM" κυρίως σε εκείνα τα ακολουθιακά κυκλώματα τα οποία έχουν σχετικά λίγες καταστάσεις, και των οποίων τη λειτουργία μπορούμε να περιγράψουμε και να καταλάβουμε καλύτερα αν μιλήσουμε γιά το πώς η κάθε μιά κατάστασή τους, σε συνδυασμό με τις τιμές εισόδου, καθορίζει τις εξόδους και την επόμενη κατάσταση· συνήθως, η περιγραφή αυτή γίνεται με ένα "**Διάγραμμα (Μετάβασης) Καταστάσεων** " (State (Transition) Diagram), όπως θα δούμε αμέσως παρακάτω. Μεγαλύτερα ακολουθιακά κυκλώματα, με πολλά bits μνήμης, συνήθως τα μελετάμε και τα σχεδιάζουμε κόβοντάς τα σε υποκυκλώματα, μερικά από τα οποία τα περιγράφουμε σαν FSM's ενώ άλλα τα περιγράφουμε και τα μελετάμε με άλλους τρόπους (μνήμες RAM, datapaths --που θα πούμε αργότερα-- κλπ).

10.2 Μετρητές, και το Διάγραμμα Καταστάσεών τους:



Στο παράγραφο 8.2 είδαμε ένα κύκλωμα μετρητή με χρήση αθροιστή. Εδώ δίπλα φαίνεται μιά παραλλαγή αυτού του κυκλώματος: είναι τρίμπιτο αντί τετράμπιτο, γιά απλότητα του σχεδίου, και αντί παράλληλης φόρτωσης (αρχικοποίησης) με αυθαίρετο αριθμό υπάρχει μόνο η δυνατότητα μηδενισμού, μέσω της εισόδου reset'· επίσης, αντί δύο μανταλωτών σε σειρά, φυσικά, εδώ χρησιμοποιούμε έναν ακμοπυροδότητο καταχωρητή. Το ακολουθιακό αυτό κύκλωμα μετρητή αποτελείται από έναν τρίμπιτο καταχωρητή, που κρατά την κατάσταση του κυκλώματος, και από ένα συνδυαστικό κύκλωμα που αποτελείται από τον πολυπλέκτη, τον αθροιστή, και τις πύλες AND. Η μία πύλη AND στο σχήμα παριστάνει στην πραγματικότητα 3 πύλες, με κοινή πρώτη είσοδο "reset'" και με δεύτερη είσοδο από ένα από τα bits του αθροίσματος, καθεμία. Όταν reset'=1, οι 3 έξοδοι των πυλών AND ταυτίζονται με το άθροισμα, ενώ όταν reset'=0, οι 3 αυτές έξοδοι είναι 000.

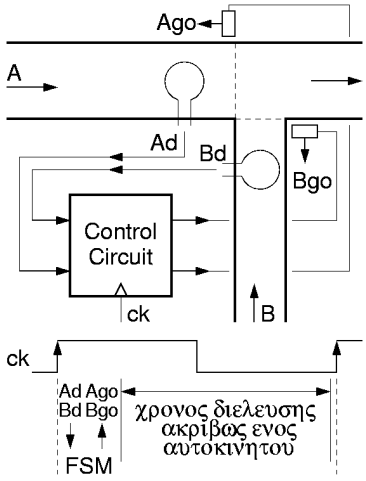


Κάτω από το κύκλωμα φαίνεται το διάγραμμα καταστάσεων αυτού του μετρητή. Οι 8 κύκλοι παριστάνουν τις 8 διαφορετικές καταστάσεις στις οποίες αυτός μπορεί να βρίσκεται, αφού έχει 3 bits κατάστασης. Τα βέλη παριστάνουν τις μεταβάσεις καταστάσεων: εάν σε δεδομένο κύκλο ρολογιού το κύκλωμα βρίσκεται σε δεδομένη κατάσταση, και είναι δεδομένες οι τιμές των εισόδων του, τότε το αντίστοιχο βέλος δείχνει σε ποιάν επόμενη κατάσταση θα πάει το κύκλωμα στον επόμενο κύκλο ρολογιού. Τα μπλέ βέλη, που σημειώνονται με "u" (από "up", δηλ. μέτρηση προς τα επάνω), αντιστοιχούν στην τιμή εισόδου ελέγχου του πολυπλέκτη που επιλέγει την είσοδο +1 (και ενώ ταυτόχρονα η είσοδος reset' είναι 1)· υπ' αυτές τις συνθήκες, η επόμενη κατάσταση είναι εκείνη με κωδικό κατά 1 μεγαλύτερο από την παρούσα. Τα κόκκινα βέλη, που σημειώνονται με "d" (down - κάτω), αντιστοιχούν στην είσοδο που επιλέγει το -1, και

μας μεταφέρουν στην κατάσταση με κωδικό κατά 1 μικρότερο της παρούσας. Τα βέλη που σημειώνονται με "n" (noop - καμιά πράξη) αντιστοιχούν στην είσοδο που επιλέγει το 0, και γι' αυτό η επόμενη κατάσταση ταυτίζεται με την παρούσα.

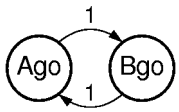
Κάθε ακολουθιακό κύκλωμα πρέπει να έχει έναν τρόπο **αρχικοποίησης** (**reset**) όταν του πρωτοδίνουμε τάση τροφοδοσίας, ή σε περίπτωση (έκτακτης) "ανάγκης" --όταν "χάνουμε το λογαριασμό" σε ποιά κατάσταση βρίσκεται, ή όταν πάει και "κολλάει" σε μιά κατάσταση και δεν ξέρουμε πώς αλλοιώς να το βγάλουμε από εκεί. Όπως έχουμε πει, όταν πρωτοανάβουμε την τροφοδοσία ενός flip-flop, αυτό πηγαίνει τυχαία σε μιάν από τις δύο σταθερές καταστάσεις του, 0 ή 1, και δεν υπάρχει τρόπος να προβλεφθεί σε ποιάν από τις δυό θα πάει (σαν να ρίχνεις ένα κέρμα "κορώνα-γράμματα"). Έτσι, η κατάσταση εκκίνησης ενός ακολουθιακού κυκλώματος είναι **τυχαία** , και πρέπει να υπάρχει τρόπος, στη συνέχεια, αυτό να έλθει σε γνωστή και επιθυμητή αρχική κατάσταση. Στο παραπάνω κύκλωμα, η αρχικοποίηση γίνεται ενεργοποιώντας το σήμα "reset", δηλαδή reset'=0, οπότε η επόμενη κατάσταση είναι πάντα η 0, ανεξαρτήτως προηγούμενης κατάστασης και των άλλων εισόδων. Στο διάγραμμα καταστάσεων, αυτό παριστάνεται με τα μάυρα βέλη που σημειώνονται με "r"' (έχει παραληφθεί ένα ακόμα τέτοιο βέλος, από την κατάσταση 0 στον εαυτό της, που έπρεπε να υπάρχει αλλά δεν χωρούσε στο σχήμα).

10.3 Παράδειγμα FSM: Φώτα Κυκλοφορίας σε Διασταύρωση Οδών



Το πρώτο μας παράδειγμα μηχανής πεπερασμένων καταστάσεων (FSM) θα έχει να κάνει με τον χρονοπρογραμματισμό (scheduling) κοινόχρηστων πόρων. Θα χρησιμοποιήσουμε σαν παράδειγμα μιά διασταύρωση δρόμων, αλλά οι ιδέες είναι ίδιες π.χ. με δρομολόγηση πακέτων πληροφοριών σε δίκτυα υπολογιστών, ή με την εναλλάξ εκτέλεση διεργασιών (προγραμμάτων) σ' έναν υπολογιστή που χρησιμοποιεί πολυπρογραμματισμό. Ας φανταστούμε δύο μονόδρομους, Α και Β, οι οποίοι συμβάλουν, όπως δείχνει το σχήμα δεξιά· ο Α είναι κεντρικός, ενώ ο Β είναι πάροδος. Στη διασταύρωσή τους υπάρχουν φώτα κυκλοφορίας, Ago και Bgo (1=πράσινο, 0=κόκκινο). Θέλουμε να σχεδιάσουμε το κύκλωμα ελέγχου των φαναριών. Στις πιό εξελιγμένες παραλλαγές του, το κύκλωμα αυτό θα χρησιμοποιεί σαν εισόδους του δύο ανιχνευτές αυτοκινήτων (car detectors), Ad και Bd: όταν υπάρχει αυτοκίνητο που περιμένει να περάσει από το φανάρι Α, τότε Ad=1, αλλοιώς Ad=0, και ομοίως γιά το δρόμο Β. Γιά απλοποίηση της συζήτησής μας θα θεωρήσουμε ότι υπάρχει ένα ρολόϊ, ck, που κάθε περίοδος του επαρκεί γιά τη διέλευση ακριβώς ενός αυτοκινήτου από τη διασταύρωση· στην αρχή της περιόδου, η FSM ελέγχου λαμβάνει τις εισόδους Ad και Bd που την ενημερώνουν γιά την ύπαρξη ή μη αυτοκινήτων στον κάθε δρόμο, αμέσως μετά ανάβει πράσινο στο κατάλληλο φανάρι, Ago ή Bgo, και κατά την υπόλοιπη περίοδο του ρολογιού περνάει το αυτοκίνητο που έχει πράσινο. Θα μελετήσουμε πέντε παραλλαγές του κυκλώματος ελέγχου, με σταδιακή βελτίωση των χαρακτηριστικών του.

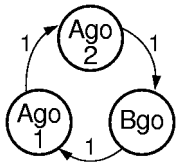
10.3.1 Κλασσικά φανάρια, χωρίς ανιχνευτές, με αναλογία 1 προς 1:



Τα συνηθισμένα φώτα κυκλοφορίας δεν έχουν ανιχνευτές αυτοκινήτων, κι έτσι δεν μπορούν να προσαρμοστούν στις πραγματικές συνθήκες κυκλοφορίας --απλώς αναβοσβήνουν περιοδικά το κόκκινο και το πράσινο με συγκεκριμένη διάρκεια γιά τον κάθε δρόμο. Στην απλούστερη περίπτωση, στο παραπάνω παράδειγμα, το πράσινο θα ανάβει εναλλάξ στους δύο δρόμους, γιά διάστημα ενός κύκλου ρολογιού --δηλαδή επαρκές γιά ένα αυτοκίνητο-- κάθε φορά. Το κύκλωμα ελέγχου σε αυτή την περίπτωση θα είναι η απλούστατη FSM τύπου μετρητή με 2 καταστάσεις που φαίνεται στο σχήμα. Η FSM αυτή δεν έχει εισόδους, και οι μεταβάσεις καταστάσεων γίνονται πάντα με τον ίδιο τρόπο, χωρίς συνθήκη· γιά το λόγο αυτό, στα βέλη σημειώνουμε "1" (πάντα αληθές), δηλαδή η αντίστοιχη μετάβαση γίνεται πάντα. Προφανώς, η FSM αυτή εναλλάσει συνεχώς κατάσταση μεταξύ της "Ago" (πράσινο στον Α) και της "Bgo" (πράσινο στον Β). Το απλούστατο αυτό κύκλωμα ελέγχου έχει μιά σειρά από μειονεκτήματα:

- **Περιττές καθυστερήσεις υπό ελαφρύ φορτίο:** λέμε ότι το σύστημα λειτουργεί υπό ελαφρύ φορτίο όταν περνάνε πολύ λίγα αυτοκίνητα, "αραιά και πού". Υπ' αυτές τις συνθήκες, όταν ένα αυτοκίνητο φτάνει στη διασταύρωση, συνήθως δεν θα υπάρχει αυτοκίνητο στον άλλο δρόμο, άρα θα μπορούσε να περάσει αμέσως. Παρ' όλα αυτά, το φανάρι με το κύκλωμα ελέγχου που μόλις σχεδιάσαμε μπορεί να τύχει να είναι κόκκινο (με πιθανότητα 50%), οπότε αυτό προκαλεί καθυστέρηση χωρίς λόγο.
- **Έλλειψη δικαιοσύνης υπό βαρύ φορτίο:** λέμε ότι το σύστημα λειτουργεί υπό βαρύ φορτίο όταν υπάρχει πολλή κίνηση, άρα σχηματίζεται ουρά πίσω από τα φανάρια, και στους δύο δρόμους. Υπ' αυτές τις συνθήκες, το κύκλωμα ελέγχου που μόλις σχεδιάσαμε αφήνει να περνούν τα αυτοκίνητα Α και τα αυτοκίνητα Β με αναλογία 1 προς 1· όμως, δεδομένου ότι ο Α είναι κεντρικός δρόμος ενώ ο Β είναι πάροδος, η αναλογιά αυτή δεν είναι δίκαια.
- **Ανεπαρκής αξιοποίηση πόρων υπό ασύμμετρο φορτίο:** λέμε ότι το σύστημα λειτουργεί υπό ασύμμετρο φορτίο όταν ο ένας δρόμος έχει πολλή κίνηση και ο άλλος λίγη. Υπ' αυτές τις συνθήκες, το κύκλωμα ελέγχου που μόλις σχεδιάσαμε αφήνει να περνά μόνο ένα αυτοκίνητο κάθε 2 χρόνους από τον βεβαρυμένο δρόμο, δηλαδή ρυθμός ροής μόνο 50 % της μέγιστης εφικτής παροχής, ακόμα και όταν στον άλλο δρόμο δεν υπάρχει κανένα αυτοκίνητο.

10.3.2 Κλασσικά φανάρια, χωρίς ανιχνευτές, με αναλογία 2 προς 1:



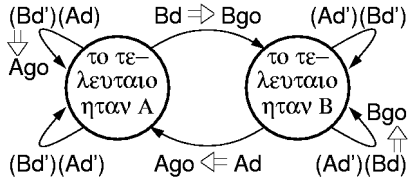
Την αδικία προς τον κεντρικό δρόμο υπό βαρύ φορτίο, λόγω αναλογίας 1:1 στη διάρκεια του πράσινου στους δύο δρόμους, μπορούμε να την διορθώσουμε προσθέτονας καταστάσεις στον παραπάνω μετρητή, όπως φαίνεται στο διάγραμμα καταστάσεων της νέας FSM, δίπλα. Εδώ, το πράσινο στον δρομο Α ανάβει σε δύο από τις τρείς καταστάσεις, ενώ στον δρόμο Β μόνο σε μία· έτσι, υπό βαρύ φορτίο, η αναλογία παροχής A:B είναι 2:1, και η μέγιστη παροχή Α έχει αυξηθεί στα 2/3 ενώ η μέγιστη παροχή Β έχει μειωθεί στο 1/3. Φυσικά, συνεχίζουν να υπάρχουν περιττές καθυστερήσεις υπό ελαφρύ φορτίο και ανεπαρκής αξιοποίηση υπό ασύμμετρο φορτίο --τώρα μάλιστα οι καθυστερήσεις στο δρόμο Β έχουν αυξηθεί και η μέγιστη παροχή του έχει μειωθεί.

10.3.3 Συνδυαστικό κύκλωμα σταθερής προτεραιότητας (απλό "STOP"):

Οι περιττές καθυστερήσεις υπό ελαφρύ φορτίο και η ανεπαρκής αξιοποίηση υπό ασύμμετρο φορτίο είναι αδύνατον να διορθωθούν χωρίς ανιχνευτές αυτοκινήτων, διότι χωρίς αυτούς το κύκλωμα ελέγχου είναι "τυφλό": δεν έχει τρόπο να ξέρει τις πραγματικές συνθήκες κυκλοφορίας. Έστω τώρα ότι έχουμε τις εισόδους Ad και Bd από τους ανιχνευτές. Ας προσπαθήσουμε κατ' αρχήν να φτιάξουμε το κύκλωμα ελέγχου σαν σκέτο συνδυαστικό κύκλωμα, χωρίς μνήμη. Φυσικά, όταν υπάρχει αυτοκίνητο μόνο σε έναν από τους δύο δρόμους, πρέπει να το αφήνουμε να περνάει, τώρα που ξέρουμε τι γίνεται σε κάθε δρόμο την κάθε στιγμή. Το πρόβλημα είναι όταν Ad=1 και Bd=1, δηλαδή υπάρχουν αυτοκίνητα και στους δύο δρόμους. Αφού δεν θυμόμαστε τίποτα από το παρελθόν (συνδυαστικό κύκλωμα), και αφού ο δρόμος Α είναι κεντρικός ενώ ο Β είναι πάροδος, λογικό είναι να δώσουμε προτεραιότητα στον Α, άρα: Ago = Ad, και Bgo = (Ad')·(Bd) --δηλαδή αν υπάρχει αυτοκίνητο στον Α του ανάβουμε πράσινο και το αφήνουμε να περάσει, ενώ αν δεν υπάρχει αυτοκίνητο στον Α και υπάρχει στον Β τότε ανάβουμε πράσινο στον Β. Σαν συνδυαστικό που είναι το κύκλωμα αυτό αποφασίζει χρησιμοποιώντας μόνο τα δεδομένα της στιγμής --χωρίς καμία γνώση από το παρελθόν-- άρα στην πράξη υλοποιείται χωρίς φανάρια, με μιά σκέτη πινακίδα "STOP" στο δευτερεύοντα δρόμο Β.

Το κύκλωμα ελέγχου αυτό δεν εισάγει περιττές καθυστερήσεις υπό ελαφρύ φορτίο, διότι κάθε αυτοκίνητο περνάει αμέσως μόλις έρθει αν δεν υπάρχει αυτοκίνητο στον άλλο δρόμο (ελαφρύ φορτίο). Επίσης, υπό ασύμμετρο φορτίο, προσφέρει πλήρη αξιοποίηση πόρων: όποτε υπάρχει αυτοκίνητο σε κάποιο δρόμο εισόδου, περνάει κάποιο από αυτά στην έξοδο, άρα η έξοδος ποτέ δεν μένει ανεκμετάλλευτη τη στιγμή που υπάρχουν αυτοκίνητα που θέλουν να περάσουν. Όμως, το κύκλωμα αυτό πάσχει από μεγάλη έλλειψη δικαιοσύνης προς τα αυτοκίνητα Β: μόνο όποτε και εάν δεν υπάρχει κανένα αυτοκίνητο Α μπορεί να περάσει ένα αυτοκίνητο Β. Η αδικία αυτή δεν αποκλείεται να φτάσει στην ακραία μορφή του λεγόμενου φαινόμενου "**λιμοκτονίας**" (starvation): εάν συνεχώς υπάρχουν αυτοκίνητα Α, ένα αυτοκίνητο Β δεν θα εξυπηρετηθεί ποτέ.

10.3.4 Προσαρμοστικός Έλεγχος (με αναλογία 1 προς 1):



Η αδικία της τελευταίας λύσης προέρχεται από την έλλειψη μνήμης στο κύκλωμα: όσο πολλά αυτοκίνητα Α και να έχουν περάσει πρόσφατα, ο ελεγκτής δεν το ξέρει, κι έτσι αφήνει ακόμα ένα να περάσει. Η έλλειψη αυτή διορθώνεται με την απλή FSM που φαίνεται στο σχήμα δίπλα. Εδώ έχουμε 1 bit μνήμης (μόνο), μέσω του οποίου θυμόμαστε από ποιόν δρόμο ήλθε το τελευταίο αυτοκίνητο που πέρασε. Εάν το τελευταίο ήταν από τον Α και τώρα υπάρχει αυτοκίνητο στον Β, το αφήνουμε να περάσει· αντίθετα, αν το τελευταίο ήταν από τον Β, τώρα δίνουμε προτεραιότητα στον Α: προτεραιότητα θα έχει ο άλλος δρόμος από αυτόν που εξυπηρετήθηκε την τελευταία φορά. Φυσικά, εάν δεν υπάρχει αυτοκίνητο στο δρόμο με την προτεραιότητα, αφήνουμε να περάσει το αυτοκίνητο από τον άλλο δρόμο, εάν υπάρχει. Η επόμενη κατάσταση είναι ανάλογη με το ποιός τελικά πέρασε, ενώ αν δεν πέρασε κανείς (διότι δεν υπήρχε κανείς) η κατάσταση παραμένει αμετάβλητη. Πάνω στα βέλη του διαγράμματος καταστάσεων σημειώνουμε τη συνθήκη μετάβασης (μαύρο), αλλά και την έξοδο που πρέπει να ενεργοποιηθεί (μπλέ).

Ας δούμε τώρα πώς συνθέτουμε το κύκλωμα μιάς FSM από το διάγραμμα καταστάσεών της. Πρώτα πρέπει να αποφασίσουμε πόσα bits κατάστασης θα χρειαστούμε, και με ποιό συνδυασμό τιμών τους θα παραστήσουμε την κάθε κατάσταση του διαγράμματος. Φυσικά, υπάρχουν πολλές δυνατές επιλογές· η επιλογή μας δεν θα επηρεάσει την εξωτερική συμπεριφορά της FSM, αλλά είναι πολύ πιθανό να επηρεάσει την πολυπλοκότητα του κυκλώματος υλοποίησης --εδώ, πάντως, δεν υπάρχει εύκολος κανόνας γιά τον τρόπο που συμφέρει να γίνει η κωδικοποίηση των καταστάσεων. Στο παρόν παράδειγμά μας, οι καταστάσεις είναι δύο, άρα αρκεί ένα bit γιά την κωδικοποίησή τους· ας το ονομάσουμε S, και ας θεωρήσουμε ότι S=0 σημαίνει ότι "το τελευταίο ήταν Α", ενώ S=1 σημαίνει ότι "το τελευταίο ήταν Β". Εάν ονομάσουμε nS την επόμενη κατάσταση, τότε ο πίνακας αληθείας του συνδυαστικού κυκλώματος μέσα στην FSM είναι ο παρακάτω (αριστερά), από τον οποίον προκύπτουν οι έξοδοι του κυκλώματος σαν συναρτήσεις των εισόδων του (δεξιά):

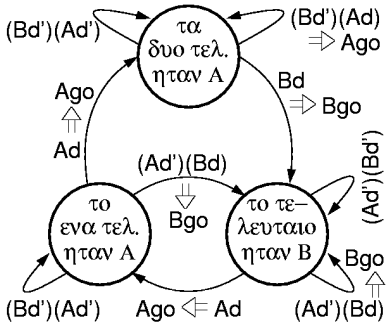
S	Ad	Bd	Ago	Bgo	nS
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	1	0	0
0	1	1	0	1	1
1	0	0	0	0	1
1	0	1	0	1	1
1	1	0	1	0	0
1	1	1	1	0	0

```
Ago = Ad · [ S + (S')·(Bd') ]

Bgo = Bd · [ S' + (S)·(Ad') ]

nS  =  S · Ad'  +  S' · Bd
```

Στην FSM αυτή, ειδική αρχικοποίηση δεν απαιτείται, διότι οιαδήποτε αρχική κατάσταση είναι έγκυρη και αποδεκτή.



10.3.5 Προσαρμοστικός έλεγχος με αναλογία 2 προς 1:
Η τελευταία FSM διατηρεί όλες τις καλές ιδιότητες του προσαρμοστικού ελέγχου (adaptive control), δηλαδή του ελέγχου που παρακολουθεί το φορτίο του συστήματος και προσαρμόζεται σε αυτό: υπό ελαφρύ φορτίο ποτέ δεν προκαλεί περιττές καθυστερήσεις, και υπό ασύμμετρο φορτίο προσφέρει πλήρη χρήση πόρων. Επίσης, αίρει την κατάφορη αδικία στο δρόμο B, δίνοντάς του ισότιμη μεταχείριση με τον δρόμο A. Μένει τώρα να διορθώσουμε την αναλογία παροχής υπό βαρύ φορτίο από 1:1 σε κάτι ορθότερο, δεδομένου ότι η A είναι κύρια οδός και η B είναι πάροδος· αυτό μπορεί να επιτευχθεί με την τροποποιημένη FSM που φαίνεται δίπλα. Εδώ, ξεχωρίζουμε την περίπτωση (κατάσταση) όπου το ένα τελευταίο αυτοκίνητο ήταν A ενώ το προηγούμενό του ήταν B από την περίπτωση που και τα δύο τελευταία αυτοκίνητα ήταν A· στην πρώτη περίπτωση αφήνουμε να περάσει και δεύτερο A κατά προτεραιότητα, ενώ στη δεύτερη περίπτωση δίνουμε προτεραιότητα στο B. Υπό βαρύ φορτίο η FSM αυτή εξυπηρετεί τους δρόμους A και B με αναλογία 2:1, ενώ επίσης διατηρεί και όλα τα άλλα πλεονεκτήματα των δύο τελευταίων λύσεων, δηλαδή τελικά λύνει όλα τα παραπάνω μειονεκτήματα της λύσης της §10.3.1.

Άσκηση 10.4: Σχεδίαση της FSM (5) γιά τη Διασταύρωση των Οδών

[Κάντε την πριν το εργαστήριο και παραδώστε την με την αναφορά σας.]
Σχεδιάστε την τελευταία παραπάνω FSM της περίπτωσης 5 (προσαρμοστικός έλεγχος με αναλογία 2 προς 1), όπως σχεδιάσαμε και την FSM 4, δηλαδή εκφράστε όλες τις εξόδους του συνδυαστικού κυκλώματος σαν συναρτήσεις των εισόδων του. Κωδικοποιήστε τις 3 καταστάσεις της FSM με δύο bits κατάστασης, S1-S0, ως εξής: 00 = "το ένα τελευταίο ήταν A", 01 = "τα δύο τελευταία ήταν A", 10 = "το τελευταίο ήταν B". Μεταφράστε το διάγραμμα καταστάσεων στον πίνακα αληθείας των εξόδων Ago, Bgo, nS1, και nS0 σαν συναρτήσεις των εισόδων S1, S0, Ad, και Bd.

Παρατηρήστε ότι οι γραμμές του πίνακα με S1-S0 = 11 δεν καλύπτονται από το διάγραμμα καταστάσεων, δεδομένου ότι η 11 είναι αχρησιμοποίητη κατάσταση. Αρχικά, αφήστε τις εξόδους, στις γραμμές αυτές, σαν *συνθήκες αδιαφορίας* (§4.3), γιά να διευκολυνθεί η απλοποίηση των συναρτήσεων των εξόδων. Στη συνέχεια όμως, ελέγξτε ποιά σας προέκυψε σαν επόμενη κατάσταση μετά την 11 γιά τις διάφορες τιμές των εισόδων: το κύκλωμα πρέπει να εγκαταλείπει αυτή την "παράνομη" κατάσταση όσο πιό γρήγορα γίνεται, αν τύχει να βρεθεί εκεί όταν πρωτοανάβει η τροφοδοσία. Γιά να είστε εντελώς εντάξει, πρέπει και να σιγουρευτείτε και ότι δεν θα ανάψουν ποτέ και τα δύο πράσινα μαζί (Ago=Bgo=1), ακόμα και όταν το κύκλωμα βρεθεί στην "παράνομη" κατάσταση 11.

10.5 Κώδικες Μεταβλητού Μήκους: Κωδικοποίηση Huffman

Το επόμενο παράδειγμα FSM που θα δούμε είναι από μιάν άλλη ενδιαφέρουσα περιοχή, τους κώδικες μεταβλητού μήκους, οι οποίοι έχουν σημαντικές εφαρμογές στις ψηφιακές επικοινωνίες και ομοιότητες με την αναγνώριση γραμματικών από τους μεταφραστές (compilers). Ξέρουμε ότι με n bits μπορούμε να κωδικοποιήσουμε 2^n διαφορετικά σύμβολα (εργαστήριο 1)· οι κώδικες εκείνοι είναι *σταθερού μήκους* (ή πλάτους), διότι κωδικοποιούν όλα τα σύμβολα με το ίδιο πλήθος bits, καθένα. Οι κώδικες σταθερού πλάτους βολεύουν όταν τα σύμβολα αποθηκεύονται σε μνήμες και όταν μεταφέρουμε τα bits τους όλα μαζί, εν παραλλήλω (π.χ. μέσα στο datapath επεξεργαστών). Όμως, όταν τα bits της κωδικοποίησης μεταδίδονται σειριακά, το ένα μετά το άλλο, μέσα από ένα κανάλι επικοινωνίας, χρησιμοποιούνται και κώδικες **μεταβλητού μήκους** (variable length), όταν αυτοί μπορούν να επιτύχουν μετάδοση της πληροφορίας με λιγότερα συνολικά bits. Οι κώδικες αυτοί πήραν το όνομά τους από τον **Huffman** ο οποίος τους μελέτησε.

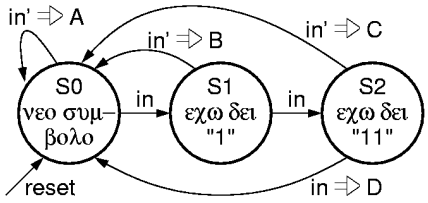
Γιά παράδειγμα, θεωρήστε ότι θέλουμε να στέλνουμε μηνύματα αποτελούμενα από μακρές ακολουθίες τεσσάρων (4) συμβόλων, των A, B, C, και D --δηλαδή το αλφάβητό μας έχει αυτά τα 4 γράμματα/σύμβολα. Ξέρουμε ότι αυτά μπορούμε να τα παραστήσουμε μ' ένα κώδικα σταθερού μήκους 2 bits ανά σύμβολο: 00, 01, 10, και 11. Έστω τώρα ότι στα μηνύματά μας η συχνότητα με την οποία εμφανίζονται τα διάφορα σύμβολα δεν είναι η ίδια: υπάρχουν σύμβολα πολύ συχνά και άλλα πολύ σπάνια. Σ' αυτή την περίπτωση, ένας κώδικας μεταβλητού μήκους μπορεί να μεταδώσει τα μηνύματά μας με λιγότερα bits συνολικά και κατά μέσον όρο. Ας πούμε, στο παραπάνω παράδειγμα, ότι στα μηνύματά μας το 50 % των εμφανιζομένων συμβόλων είναι A, το 25 % είναι B, το 12.5 % είναι C, και το 12.5 % είναι D, και ας θεωρήσουμε τον κώδικα μεταβλητού μήκους:

```
A: 0,   B: 10,   C: 110,   D: 111.
```

Τότε, γιά κάθε π.χ. 1000 σύμβολα που μεταδίδονται, χρειάζονται περίπου 500 bits γιά τη μετάδοση των περίπου 500 A που υπάρχουν κατά μέσον όρο εκεί (1 bit ανά A), συν άλλα 500 bits γιά τα 250 B (2 bits ανά B), σύν 375 bits γιά τα 125 C (3 bits ανά C), σύν 375 bits γιά τα D. Το σύνολο είναι 1750 bits γιά κάθε 1000 μεταδιδόμενα σύμβολα, ή 1.75 bits/σύμβολο κατά μέσον όρο, πράγμα σημαντικά χαμηλότερο από τα 2 bits/σύμβολο του κώδικα σταθερού πλάτους. Το κλειδί γιά την επίτευξη αυτής της οικονομίας είναι το να αφιερώνουμε τους (λίγους) κώδικες μικρού μήκους στα σύμβολα εκείνα που εμφανίζονται περισσότερο συχνά, σε βάρος των σπανιότερων συμβόλων που παρίστανται με περισσότερα bits. Ο γνωστός μας από παλιά κώδικας Μορς είναι φτιαγμένος με παρόμοιο τρόπο, εκμεταλλευόμενος το γεγονός ότι στις ανθρώπινες γλώσσες ορισμένα γράμματα είναι πολύ συχνότερα από άλλα.

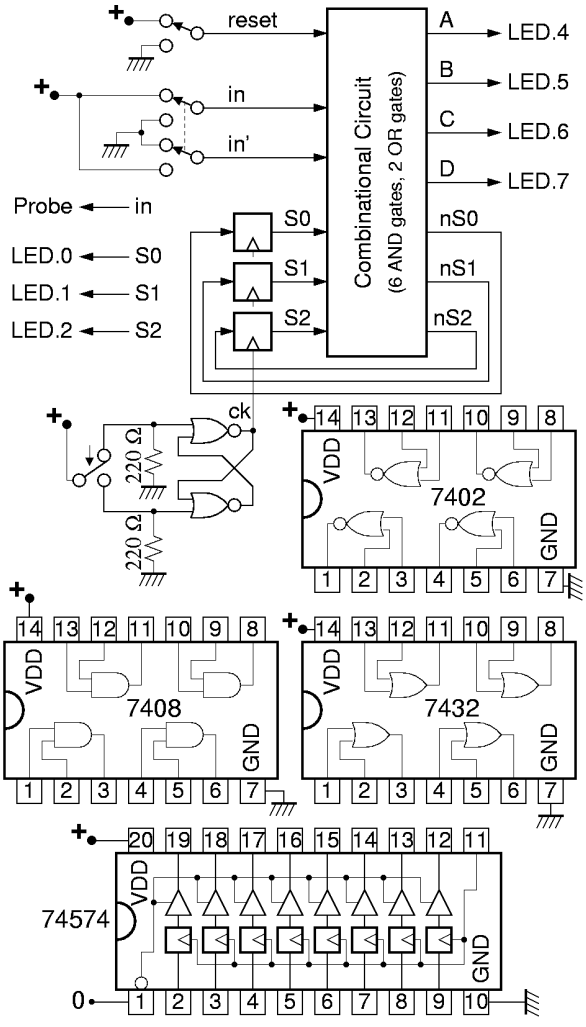
Ο παραπάνω κώδικας έχει φτιαχτεί προσεκτικά ούτως ώστε να είναι εφικτή η κατ' ευθείαν αναγνώριση του κάθε σύμβολου αμέσως μόλις φτάσει το τελευταίο bit της κωδικοποιημένης μορφής του, χωρίς διφορούμενα και χωρίς την ανάγκη να περιμένουμε πρώτα να δούμε τα επόμενα bits του μηνύματος. Η ιδιότητα αυτή υπάρχει επειδή ο κώδικας **κάθε** βραχύτερου σύμβολου **δεν** αποτελεί πρόθεμα του κώδικα **κανενός** από τα μακρύτερα σύμβολα. Παρόμοια ιδιότητα έχουν και οι αριθμοί τηλεφώνου, όπου τα πρώτα ψηφία (πρόθεμα) καθορίζουν μονοσήμαντα το συνολικό μήκος και το είδος του αριθμού. Την ιδιότητα αυτή δεν θα την είχε π.χ. ο κώδικας A=0, B=1, C=10, D=11: με αυτόν τον κώδικα, το μήνυμα 110 μπορεί να σημαίνει είτε BC είτε DA. Παρόμοιο "διφορούμενο" (ambiguous) κώδικα είχε χρησιμοποιήσει και το Μαντείο των Δελφών στον περίφημο εκείνο χρησμό του "ίξεις αφίξεις ουκ εν τω πολέμω θνήξεις", που μπορούσε να ερμηνευτεί είτε σαν "ίξεις, αφίξεις, ουκ εν τω πολέμω θνήξεις" (θα πας, θα γυρίσεις, δεν θα πεθάνεις στον πόλεμο), είτε σαν "ίξεις, αφίξεις ουκ, εν τω πολέμω θνήξεις" (θα πας, δεν θα γυρίσεις, στον πόλεμο θα πεθάνεις).

10.6 FSM Αποκωδικοποίησης Huffman:



Ο παραπάνω κώδικας μεταβλητού μήκους γιά τα 4 σύμβολα A, B, C, D αποκωδικοποιείται με την FSM που φαίνεται δεξιά. Υποθέτουμε ότι τα bits των κωδικοποιημένων συμβόλων έρχονται σειριακά, ένα-ένα, από μιάν είσοδο in, και ότι υπάρχει ένα ρολόϊ ck που σημαδεύει, με την ενεργή ακμή του, το τέλος του κάθε bit εισόδου in. Η FSM έχει 4 εξόδους, A, B, C, D· κάθε μιά τους ανάβει όποτε παραλαμβάνουμε ένα αντίστοιχο σύμβολο, γιά διάρκεια ακριβώς ενός κύκλου ρολογιού, και συγκεκριμένα κατά τον κύκλο ρολογιού παραλαβής του τελευταίου bit του αντίστοιχου συμβόλου.

Η κατάσταση εκκίνησης είναι η S0, οπότε και περιμένουμε να μας έλθει ένα νέο σύμβολο, με όλα του τα bits από την αρχή. Στην ίδια κατάσταση επιστρέφουμε κάθε φορά που τελειώνουν όλα τα bits του προηγούμενου σύμβολου, και επομένως περιμένουμε ένα νέο σύμβολο από την αρχή. Εάν είμαστε στην κατάσταση S0 και μας έλθει είσοδος 0 (in' αληθές), σημαίνει ότι βλέπουμε ένα σύμβολο A (το μοναδικό άρα και το τελευταίο του bit), επομένως πρέπει να ανάψει η έξοδος A και η επόμενη κατάσταση να είναι πάλι η S0, αφού στον επόμενο κύκλο περιμένουμε το πρώτο bit του επομένου συμβόλου. Εάν τώρα είμαστε στην S0 και έλθει είσοδος 1 (in αληθές), τότε ξέρουμε ότι βλέπουμε το πρώτο bit ενός συμβόλου B ή C ή D, αλλά δεν ξέρουμε ακόμα τίνος από τα τρία· έτσι, μεταβαίνουμε στην κατάσταση S1 που σημαίνει ότι έχουμε δεί μέχρι στιγμής ένα "1". Τον επόμενο κύκλο, όντας στην S1, αν δούμε είσοδο 0 σημαίνει ότι βλέπουμε το τέλος ενός συμβόλου B, άρα ανάβουμε την έξοδο B και μεταβαίνουμε στην S0. Αν αντιθέτως δούμε είσοδο 1 σημαίνει ότι βλέπουμε το δεύτερο bit ενός συμβόλου C ή D, αλλά δεν ξέρουμε ακόμα τίνος από τα δύο· έτσι, μεταβαίνουμε στην κατάσταση S2. Τον επόμενο κύκλο, όντας στην S2, είσοδος 0 σημαίνει ότι βλέπουμε το τέλος ενός συμβόλου C, και είσοδος 1 υποδεικνύει το τέλος ενός συμβόλου D.



10.7 Κωδικοποίηση Καταστάσεων *One-Hot:*

Στο εργαστήριο θα κατασκευάσετε την παραπάνω FSM λήψης του κώδικα μεταβλητού μήκους, χρησιμοποιώντας ένα στύλ κωδικοποίησης των καταστάσεών της που λέγεται "**one-hot**" (ένα "ζεστό"): όπως φαίνεται στο σχήμα δίπλα, χρησιμοποιούμε τόσα flip-flops κατάστασης όσες και οι καταστάσεις (αντί όσος ο δυαδικός λογάριθμος του πλήθους των καταστάσεων, που θα αρκούσαν), και θα φροντίσουμε ένα και μόνον ένα από αυτά να είναι αναμένο σε κάθε μιά από τις έγκυρες καταστάσεις της FSM. Το στυλ αυτό είναι σπάταλο σε flip-flops, αλλά συχνά δίνει απλούστερη συνδυαστική λογική γιά την FSM: είναι σαν να είχαμε μιά πιό πυκνά κωδικοποιημένη κατάσταση, και μετά να είχαμε κι έναν συνδυαστικό αποκωδικοποιητή που μας δίνει μία και μόνο μία αναμένη έξοδο, αυτήν που αντιστοιχεί στην παρούσα κατάσταση. Φυσικά, με το να έχουμε 3 (εν προκειμένω) flip-flops κατάστασης, σημαίνει ότι το κύκλωμα στην πραγματικότητα έχει 8 καταστάσεις, από τις οποίες μόνο 3 είναι οι έγκυρες γιά μας, και πρέπει να σιγουρευτούμε ότι το σήμα αρχικοποίησης reset θα φέρνει πάντα την FSM στην επιθυμητή έγκυρη αρχική κατάσταση S0, ανεξαρτήτως σε ποιάν από τις 8 υπαρκτές καταστάσεις (όχι μόνο τις 3 έγκυρες) αυτή ήταν πρίν. Εδώ λοιπόν, την κατάσταση "S0" θα την κωδικοποιούμε με S0=1 και S1=S2=0· την "S1" θα την κωδικοποιούμε με S1=1 και S0=S2=0· και η "S2" θα είναι η S2=1 και S0=S1=0. Με αυτή την κωδικοποίηση καταστάσεων, οι έξοδοι της FSM εκφράζονται πολύ απλά:

```
A = (S0)·(in'),  B=(S1)·(in'),  C=(S2)·(in'),  D=(S2)·(in)
```

διότι (S0) είναι αληθές όταν και μόνον όταν βρισκόμαστε στην κατάσταση S0, κ.ο.κ. γιά κάθε κατάσταση. Στη συνέχεια, εκφράζουμε τις εξόδους επόμενης κατάστασης. Η έξοδος nS1 πρέπει πάντα να έχει την τιμή που πρέπει να πάρει το flip-flop S1 κατά τον επόμενο κύκλο ρολογιού· άρα το nS1 πρέπει να είναι αληθές τότε και μόνο τότε όταν στον επόμενο κύκλο ρολογιού η FSM πρέπει να μεταβεί στην κατάσταση S1. Από το διάγραμμα καταστάσεων της §10.6 βλέπουμε ότι η επόμενη κατάσταση είναι η S1 μόνο σε μία περίπτωση (μόνο ένα βέλος μπαίνει στην S1): όταν τώρα είμαστε στην S0 και in=1· άρα, το nS1 πρέπει να είναι αληθές όταν και μόνον όταν (S0)·(in). Ομοίως, η S2 είναι επόμενη κατάσταση μόνο όταν S1=1 και in=1, άρα τότε και μόνο τότε πρέπει να ανάβει το nS2. Τώρα, η S0 είναι επόμενη κατάσταση σε πέντε διαφορετικές περιπτώσεις: όταν είμαστε στην S0 και in=0, ή στην S1 και in=0, ή είμαστε στην S2 (με οιαδήποτε είσοδο in), ή όταν reset=1. Έτσι, προκύπτουν οι εξής σχέσεις γιά τις εξόδους επόμενης κατάστασης:

```
        nS1 = (S0)·(in)·(reset'),  nS2 = (S1)·(in)·(reset'),
nS0 = (S0)·(in') + (S1)·(in') + (S2) + (reset) = in' + S2 + reset
```

Βλέπουμε ότι η έκφραση γιά το nS0 απλοποιήθηκε ως εξής: όποτε in=0 επόμενη κατάσταση είναι πάντα η S0, και αυτό καλύπτει τα τρία επάνω βέλη στο διάγραμμα καταστάσεων· το βέλος γιά S2 και in=1 μπορούμε να το καλύψουμε με τον όρο S2 (που επίσης καλύπτει την περίπτωση S2 και in=0 που ήδη καλύψαμε με τον όρο in')· έτσι, το μόνο άλλο βέλος προς S0 που έμεινε είναι το reset, άρα συνολικά nS0 = in' + S2 + reset. Όσον αφορά, τώρα, τη συνολική δράση του reset, αυτή πρέπει να σχεδιαστεί προσεκτικά: όταν reset=1, πρέπει η επόμενη κατάσταση να είναι η S0 ανεξαρτήτως του πού βρισκόμασταν, είτε σε μιάν από τις έγκυρες καταστάσεις S0, S1, S2, είτε σε κάποιαν από τις **υπόλοιπες 5 "παράνομες" καταστάσεις** (000, 011, 101, 110, 111). Με τον τρόπο που περιελήφθη το reset στις παραπάνω συναρτήσεις, αυτό εξασφαλίζεται: οιαδήποτε τιμή και αν έχουν τα S0, S1, S2, και in, όταν reset=1, ο όρος "AND (reset')" στις εξόδους nS1 και nS2 εξασφαλίζει ότι αυτές θα είναι 0, και ο όρος "OR (reset)" στην έξοδο nS0 εξασφαλίζει ότι αυτή θα είναι 1.

Πείραμα 10.8: η FSM Αποκωδικοποίησης Huffman

Κατασκευάστε στο εργαστήριο την παραπάνω FSM λήψης του κώδικα μεταβλητού μήκους, σύμφωνα με το κύκλωμα του τελευταίου σχήματος και τις παραπάνω εξισώσεις που περιγράφουν το συνδυαστικό του κομάτι. Όπως δείχνει το σχήμα, μπορείτε να πάρετε και τις δύο πολικότητες, in και in', της εισόδου με έναν διακόπτη DPDT. Εάν θέλετε, μπορείτε να παραλήψετε τον όρο (reset') από τα nS1 και nS2, αρκεί να θυμάστε πάντα όταν κάνετε reset να έχετε ταυτόχρονα και in=0. Γιά το συνδυαστικό κομάτι του κυκλώματος θα χρειαστείτε 2 chips 7408 (πύλες AND) και 1 chip 7432 (πύλες OR). Γιά την κατάσταση, χρησιμοποιήστε 3 από τα 8 ακμοπυροδότητα flip-flops ενός chip 74574. Γιά το ρολόϊ του, θα χρειαστείτε τον *καθαρισμό παλμών* που είπαμε στην παράγραφο 8.5: χρησιμοποιήστε ένα chip 7402 (πύλες NOR), αντιστάσεις, και διακόπτη SPDT. Συνδέστε LED's γιά να παρακολουθείτε την είσοδο in, την κατάσταση της FSM, και τις εξόδους A, B, C, και D, όπως δείχνει το σχήμα.

Πριν φτάσετε στο εργαστήριο, κάντε ένα πλήρες σχεδιάγραμμα συνδεσμολογίας, περιλαμβανομένων και όλων των διακοπτών, αντιστάσεων, LED's, και της τροφοδοσίας. Επίσης, γράψτε στο χαρτί σας την κωδικοποίηση του μηνύματος "**ABACADACABADACAC**" σύμφωνα με τον κώδικά μας μεταβλητού μήκους. Επίσης, αποκωδικοποιήστε το μήνυμα που περιέχουν τα bits "**10010011101110110011 00**" σύμφωνα πάντα με τον ίδιο κώδικα.

Στο εργαστήριο, κατασκευάστε το κύκλωμα, αρχικοποιήστε το σωστά (reset=1 και in=0 και ακμή ρολογιού), και στη συνέχεια ελέγξτε τη σωστή λειτουργία του στέλνοντάς του τα δύο παραπάνω μηνύματα (ή όσο κομάτι από αυτά αντέξετε) και βλέποντας αν τα λαμβάνει σωστά, δηλαδή αν ανάβει η σωστή LED μία φορά γιά κάθε γράμμα-σύμβολο.

Up to the Home Page of CS-120

Last updated: 16 Dec. 2003, by M. Katevenis.