

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΕΝΣΩΜΑΤΩΜΕΝΑ ΣΥΣΤΗΜΑΤΑ

Κωνσταντίνος Μυλωνάς

2018030151

Νίκος Τογανίδης

2018030085

Αναφορά project εξαμήνου

-

A. Δόλλας

-

22/12/2021

Επισκόπηση:

Στη συγκεκριμένη εργασία προγραμματίζεται ένας μικροεπεξεργαστής ώστε να είναι σε θέση να λύνει το γνωστό Sudoku. Στον μικροεπεξεργαστή δίνονται δεδομένα μέσω σειριακής θύρας από τον χρήστη με βάση το πρωτόκολλο της εκφώνησης. Εκείνος είναι σε θέση να επεξεργαστεί τα δεδομένα και να δώσει την σωστή απάντηση. Με τον τρόπο αυτό δίνεται το άλυτο Sudoku και επιστρέφεται το λυμένο. Ο μικροεπεξεργαστής που χρησιμοποιήθηκε είναι ο AVR ATmega16 που είναι ενσωματωμένος στο STK 500 που μας παραχωρήθηκε από το εργαστήριο.

Αφαιρετικά:

Σε επίπεδο κώδικα το Sudoku προσομοιώνεται με ένα 2D array 9x9 το οποίο ονομάζεται matrix.

Αν θέλαμε κάπως αφαιρετικά να χωρίσουμε το πρόγραμμα σε δομικά blocks θα λέγαμε ότι αποτελείται:

- Block που παραλαμβάνει χαρακτήρες μέσω σειριακής θύρας και θέτει κάποια flags τα οποία σηματοδοτούν την αναμενόμενη λειτουργικότητα
- Block που ανάλογα με τα flags που είναι ενεργά κάνει τις σωστές λειτουργίες για να επιτευχθεί ο εκάστοτε σκοπός, όπως για παράδειγμα να σταλεί OK σε περίπτωση που είχε ληφθεί ΑΤ.

Πίσω από το 2^ο block κρύβονται άλλα μικρότερα sub-blocks που περιέχουν περισσότερες λεπτομέρειες. Ένα από αυτά είναι και ο αλγόριθμος που επιλύει το Sudoku. Πιο συγκεκριμένα ο αλγόριθμος που χρησιμοποιήθηκε είναι ο ακόλουθος backtracking αλγόριθμος:

<https://www.geeksforgeeks.org/sudoku-backtracking-7/>

Ένα άλλο sub-block είναι αυτό που φροντίζει να αποθηκεύονται στη σωστή θέση του matrix τα νούμερα που στέλνει ο χρήστης και τα οποία αποτελούν μέρος της λύσης του Sudoku.

Αναλυτική Περιγραφή

Για να γινόμαστε όμως πιο συγκεκριμένοι ας πάμε ένα βήμα πίσω να δούμε τα flags.

Για τα flags έχει δημιουργηθεί ένα global array flags[9] ώστε να αποφευχθούν τα πολλά ονόματα για τις διάφορες μεταβλητές που θα είχαμε σε άλλη περίπτωση.

Το κάθε στοιχείο του array αντιστοιχεί στο flag που θέτουμε ανάλογα με τον χαρακτήρα που παραλάβαμε. Για παράδειγμα, για τον χαρακτήρα A θέτουμε το flags[0], για T το flags[0] κ.ο.κ

Μπορούμε να συνοψίσουμε στην ακόλουθη δομή:

0	1	2	3	4	5	6	7	8
A	T	C	P	S	B	D	N	<LF>

Για το <CR> δεν θέτουμε κάποιο flag η λήψη αυτού του χαρακτήρα δεν ενεργοποιεί κάποια λειτουργία.

Έχοντας αυτά στο μυαλό, στον handler του RXCI (Receive Complete Interrupt), ανάλογα με τον χαρακτήρα που λαμβάνεται ενεργοποιείται το εκάστοτε flag.

Παρατηρούμε πως στο αλφάβητο οι λειτουργικότητες ενεργοποιούνται μετά το <LF>¹. Έτσι μπάινουμε στην διαδικασία να εξετάσουμε το ποιά flags είναι ενεργά και να προβούμε σε ενέργειες μόνο αφού ληφθεί το <LF>.

Ο έλεγχος για το αν έχει ληφθεί το LF γίνεται στη main όπου ελέγχεται το flags[8]. Αν αυτό είναι 1 καλείται η react() που αναπαριστά το «2^ο block» που περιγράψαμε στην αρχή. Σε αυτή τη συνάρτηση ανάλογα με το ποια flags είναι ενεργά γίνονται τα προβλεπόμενα και στο τέλος επαναφέρονται τα flags στο 0 ώστε ο MCU να είναι έτοιμος να δεχτεί και να «εξυπηρετήσει» την επόμενη εντολή.

Παράκτω παρουσιάζονται αναλυτικά όλες οι περιπτώσεις flag-ενέργεια της react:

1. Δεν είμαστε αυστηροί με αυτό μιας και δεν είναι αρκετά ξεκάθαρο σε όλες τις περιπτώσεις αν πρέπει να περιμένουμε <LF> ή όχι. Για παράδειγμα θα μπορούσε κάποιος να πει πως με το που παραλάβω P ξεκινάω να

λύνω το Sudoku και έτσι γλιτώνω λίγο χρόνο μέχρι ο χρήστης να στείλει το LF. Ωστόσο στη συγκεκριμένη υλοποίηση προχωράμε μόνο αφού λάβουμε LF και θεωρούμε πως έτσι είναι πιο δομημένος ο κώδικας.

AT:

AT σημαίνει ότι έχουμε $\text{flags}[A] = 1$ και $\text{flags}[T] = 1$ και προφανώς $\text{flags}[LF] = 1$. Απο εδώ και στο εξής δεν θα αναφέρουμε το $\text{flags}[LF]$ καθώς είναι δεδομένο πως είναι ενεργό για να έχουμε μπει στην `react`.

Σε αυτή τη περίπτωση σύμφωνα με το αλφάβητο πρέπει απλα να απαντήσει ο MCU `OK<CR><LF>` το οποίο και γίνεται. Στο τέλος γίνονται `reset` τα `flags` – όπως σε κάθε άλλη περίπτωση.

C:

Σε αυτή τη περίπτωση και όντως στη `react` συνεπάγεται πως είναι $\text{flags}[C] = 1$. Έτσι προχωράμε σε μηδενισμό όλων των κελιών του `matrix` και απαντάμε OK. Αυτομάτως συνεπάγεται ότι κανένα κελί δεν είναι συμπληρωμένο. Έτσι λοιπόν η μεταβλητή που κρατάει «την πρόοδο» της λύσης μηδενίζεται και αυτήν. Αυτή η μεταβλητή ονομάζεται *filled* με βάση αυτήν ανάβουμε τα απαραίτητα led όπως ζητείται στην εκφώνηση.

P:

Ο χαρακτήρας P ενεργοποιεί την επίλυση του Sudoku. Έτσι λοιπόν αφού τον παραλάβει ο MCU, απαντάει OK και ξεκινάει να λύνει το Sudoku με τον αναδρομικό αλγόριθμο. Σε αυτή την περίπτωση υπάρχουν δυο ενδεχόμενα. Είτε ο MCU θα συνεχίσει να λύνει μέχρι τέλους (όπου μπορεί να βρει λύση ή και όχι αν το Sudoku δεν λύνεται), είτε ο χρήστης θα στείλει B – `break` και θα διακόψει τον αλγόριθμο. Στην πρώτη περίπτωση ο MCU θα απαντήσει D – `done` ενώ στην δεύτερη θα απαντήσει OK. Στην περίπτωση του `break` η λύση έχει μείνει στο σημείο που την διακόψαμε και ο χρήστης μπορεί να ξαναστείλει P για να συνεχίσει η επίλυση του Sudoku. Το `break` θα αναλυθεί περισσότερο παρακάτω.

S:

Προκειμένου να σταλεί από τον AVR κάποιο κελί πίσω στον PC χρησιμοποιείται η συνάρτηση `send_cell(num)`. Πρωτού προχωρήσουμε στην ανάλυση της συνάρτησης ας δούμε το όρισμα `num`. Αριθμούμε νοητά το κάθε κελί ξεκινώντας από το 0 και πηγαίνοντας από αριστερά προς δεξιά και από πάνω προς τα κάτω. Ενδεικτικά για τις δύο πρώτες γραμμές φαίνονται στον Πίνακα 1 που ακολουθεί:

0 κελί	1 ^ο κελί	2 ^ο κελί	3 ^ο κελί	4 ^ο κελί	5 ^ο κελί	6 ^ο κελί	7 ^ο κελί	8 ^ο κελί
9 ^ο κελί	10 ^ο κελί	11 ^ο κελί	12 ^ο κελί	13 ^ο κελί	14 ^ο κελί	15 ^ο κελί	16 ^ο κελί	17 ^ο κελί

Αυτόν λοιπόν τον αριθμό που αντιστοιχεί το εκάστοτε κελί δίνουμε ως όρισμα *num* στην συνάρτηση *send_cell(num)*. Στόχος είναι να σταλεί απο τον AVR στο PC το κελί αλλά με σωστές συντεταγμένες μιας και . Έχοντας αυτόν τον αριθμό – ταυτότητα του κελιού μπορεί με μαθηματικό τρόπο να υπολογιστούν οι συντεταγμένες με αρίθμηση 0-8. Στη συνέχεια γίνεται fetch της τιμής του matrix που ακολουθεί αρίθμηση 0-8. Η απάντηση όμως που στέλνεται στο PC πρέπει να ακολουθεί αρίθμηση 1-9. Για τον λόγο αυτό προσθέτουμε 1 στις συντεταγμένες που είχαν υπολογιστεί. Τέλος, μετατρέπουμε και τους αριθμούς σε ASCII format προσθέτωντας 48. Το 48 προκύπτει κοιτώντας τον πίνακα ASCII όπου το 0 έχει decimal value 48, το 1 έχει 49 κ.ο.κ. Τέλος γίνεται transmit των χαρακτήρων εναν εναν πχ N125 που συμβολίζει αριθμός 5 στην στήλη 1 και γραμμή 2.

Ο μαθηματικός τρόπος με τον οποίο υπολογίζονται οι συντεταγμένες σε αρίθμηση 0-9 είναι ο εξής:

$$col = num \% 9$$

$$row = \frac{num}{9}$$

Για παράδειγμα για το 11^ο κελί οι παραπάνω τύποι θα δώσουν col = 2, row = 1 το οποίο είναι σωστό όπως φαίνεται στον Πίνακα 1 που ακολουθεί αρίθμηση 0-8. Στη συνέχεια όπως είπαμε προσθέτουμε 1 + 48 = 49 για να μετατρέψουμε σε αρίθμηση 1-9 και ASCII format και στέλνουμε το αποτέλεσμα.

Ετσι, για την περίπτωση του S πρέπει να στείλουμε το πρώτο κελί , δηλαδή το κελί 0. Για τον λόγω αυτόν καλούμε *send_cell(0)*.

T:

Η μεγάλη ανάλυση για την *send_cell* δεν έγινε ανευ λόγου. Την ίδια ακριβώς συνάρτηση χρησιμοποιούμε και στην περίπτωση του T όπου έχουμε εναν αυξανόμενο δείκτη – την μεταβλητή *iterate_index* – που την αυξάνουμε κάθε φορά κατα ενα και καλούμε *send_cell(iterate_index)*. Ετσι στέλνουμε κάθε φορά το επόμενο κελί. Όταν έχει σταλεί και το τελευταίο κελί ο MCU απαντάει με done.

D:

Και σε αυτή την περίπτωση χρησιμοποιήθηκε η *send_cell*. Ωστοσο εδω χρειάζεται μια επιπλέον λειτουργία μιας και πρέπει να υπολογίσουμε το num απο τις συντεταγμένες που παραλάβαμε απο τον χρήστη. Αυτή η διαδικασία θα μπορούσε να αποφευχθεί και

να χρησιμοποιηθεί μια πιο straight forward συνάρτηση που ανάλογα τις συντεταγμένες στέλνει το αντίστοιχο κελί, αλλά για να κρατήσουμε πιο καθαρό τον κώδικα και να μην τον γεμίσουμε με συναρτήσεις και μεταβλητές επιλέξαμε να επαναχρησιμοποιήσουμε την `send_cell` - μιας και το να υπολογιστεί το `num` δεν είναι κάτι ιδιαίτερα δύσκολο.

Ο μαθηματικός τύπος είναι:

$$num = (row - 1) \cdot 9 + col - 1$$

Για παράδειγμα αν ο χρήστης στείλει D32 πρέπει να του στείλουμε το `matrix[1][2]`. Αυτό όπως φαίνεται στον Πίνακα 1 είναι το 11^ο κελί.

$$num = (2 - 1) \cdot 9 + 3 - 1 = 9 + 2 = 11$$

Αρα υπολογίζουμε το `num` και απλά το δίνουμε ως όρισμα στην `send_cell` και τα πράγματα θα δουλέψουμε όπως εξηγήσαμε παραπάνω.

N:

Πρόκειται για την εντολή που αποθηκεύει τα νούμερα που αρχικοποιούν το Sudoku.

Ετσι έχουμε ένα array `temp[3]` όπου στην θέση 0 -> col, 1 -> row, 2-> val. Αυτές οι τιμές εισάγονται στο array στον handler για `receive`, και χρησιμοποιούνται στην συνέχεια μέσα στην `react`. Επειδή οι τιμές που λαμβάνονται είναι ASCII, πρώτου αποθηκευτούν στο `temp[]` μετατρέπονται σε BCD. Αυτό γίνεται μέσω της μάσκας 0x0F όπως είχαμε δει και στα εργαστήρια. Στην `react` και εφόσον είμαστε στην περίπτωση του N – δηλαδή `flags[N] = 1`, αποθηκεύεται ο αριθμός στο σωστό κελί του `matrix` αφαιρώντας 1 από τις συντεταγμένες που παραλάβαμε. Ο λόγος της αφαίρεσης είναι επειδή όπως έχουμε αναφέρει ακολουθούμε αρίθμηση 0-8 ενώ το πρωτόκολλο επιβάλλει αρίθμηση 0-9. Κάθε φορά που εισάγεται μια νέα μη μηδενική τιμή στο `matrix` φροντίζουμε ώστε να διατηρούμε αναμένο το σωστό πλήθος LED.

Αυτό επιτυγχάνεται μέσω της συνάρτησης `led_check` η οποία καλείται όποτε έχουμε τροποποίηση του πίνακα. Η συνάρτηση λειτουργεί ως εξής:

led_check

Τα LED είναι συνδεδεμένα στο PORTB και ενεργά στο 0 επομένως όταν ο πίνακας είναι άδειος έχουμε `PORTB = 0xFF` ώστε να είναι όλα σβηστά. Κάθε 10 μη μηδενικά στοιχεία του πίνακα ανάβει και ένα επιπλέον LED (ή σβήνει αν τα στοιχεία λιγοστεύουν πχ στο

backtracking). Ο αριθμός των στοιχείων αποθηκεύεται πάντα στην μεταβλητή *filled*. Παρατηρούμε πως :

$$\#leds = filled / 10$$

Εχοντας λοιπόν τον αριθμό των leds που πρέπει να είναι αναμένα μπορούμε απλα να κάνουμε `shift left <<` την τιμή `0xFF` και να την θέτουμε στο `PORTB`.

Για παράδειγμα αν θέλουμε 2 αναμένα led δηλαδή $20 \leq filled < 30$ τότε θέλουμε να κάνουμε `shift left` κατά δυο θέσεις το `0xFF = 0b11111111` και να το μετατρέψουμε σε `0b11111100` το οποίο πράγματι θα ανάψει μόνο τα δυο LED.

Αναλυτικότερα για το `break`

Όπως με τις υπόλοιπες εντολές έτσι και με αυτήν θέλουμε να γίνει το `break` αφού παραλάβουμε και το `LF`. Έτσι στην `solve_sudoku` – η συνάρτηση που λύνει αναδρομικά το sudoku – ελέγχουμε τόσο το `flags[B]` όσο και το `flags[LF]` και αν είναι και τα δυο ενεργά τότε κλείνει η αναδρομή και η συνάρτηση επιστρέφει 1. Ο λόγος που επιστρέφει 1 είναι γιατί αν επέστρεφε 0 τότε ο αλγόριθμος θα επαναρχικοποιούσε τις τιμές που είχε θέσει μέχρι στιγμής και θα είχαμε χάσει την έως τώρα πρόοδο.

Αναδρομικός αλγόριθμος επίλυσης Sudoku

Ο αλγόριθμος που χρησιμοποιήθηκε είναι αυτός που αναφέρεται στην αρχή. Η πολυπλοκότητα του είναι $O(9^{n^2})$ που προφανώς δεν είναι κάτι φθηνό. Στον αλγόριθμο έγιναν κάποιες τροποποιήσεις ώστε να είναι συμβατός με το υπόλοιπο πρόγραμμα.

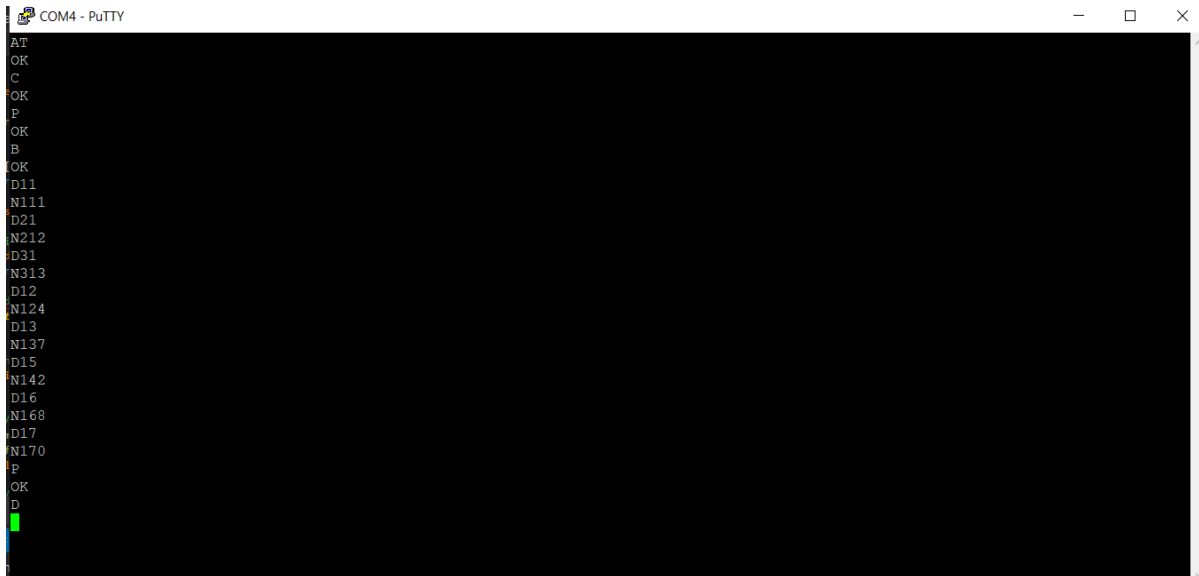
Η γενική ιδέα του αλγορίθμου είναι να εντοπίζει ένα άδειο κελί και να δοκιμάζει κάποια από τις τιμές 1-9. Αν κάποια τιμή δεν παραβιάζει τους κανόνες του παιχνιδιού την θέτει προσωρινά και προχωράει αναδρομικά για το επόμενο κελί. Αν αργότερα καταλήξει πως δεν βρέθηκε κάποια λύση, δεν τερματίζει αλλά επανέρχεται σε προηγούμενο στάδιο και δοκιμάζει μια άλλη τιμή από τις 1-9. Όταν πλέον δεν θα έχει μένει κάποια περίπτωση που να μην έχει δοκιμαστεί τότε και μόνο τότε αποφασίζει πως το Sudoku δεν έχει λύση το οποίο είναι και ορθό. Όσον αφορά τα led, κάθε φορά που τίθεται κάποια τιμή προσωρινά (ή και όχι προσωρινά) και κάθε φορά που επαναφέρεται στο 0, αυξομειώνεται η *filled* και καλείται η *led check* η οποία ανάβει η τον σωστό αριθμό των leds.

Έλεγχος λειτουργικότητας

Για τον έλεγχο της λειτουργικότητας χρησιμοποιήθηκε το interface του κ.Σταύρου. Δοκιμάστηκαν διάφοροι συνδυασμοί και λύθηκαν πολλά Sudoku για διαφορετικά επίπεδα δυσκολίας. Όλα λειτούργησαν όπως περιμέναμε. Το μόνο που δεν επιβεβαιώσαμε με το interface ήταν η εντολή break

Πως ελέγχθηκε η λειτουργικότητα της break

Για την break δεν μπορέσαμε να χρησιμοποιήσουμε το interface του κ.Σταύρου καθώς κατά την επίλυση του Sudoku δεν είμαστε σε θέση να στείλουμε κάποια εντολή. Ετσι προχωρήσαμε με το PUTTY. Επειδή η διαδικασία να αρχικοποιηθεί ένα Sudoku είναι κάπως χρονοβόρα επιλέξαμε να βάλουμε τον AVR να λύσει το εντελώς κενό Sudoku δηλαδή παντού 0. Ετσι γλιτώσαμε τον χρόνο αρχικοποίησης. Ωστόσο ο AVR λύνει τόσο γρήγορα (σε σχέση με ανθρώπινους χρόνους) το Sudoku που ήταν πρακτικά αδύνατον να προλάβουμε να δώσουμε B<CR><LF> πρώτου λυθεί το Sudoku και παραλάβουμε Done. Για τον λόγο αυτό, βάλαμε ένα delay στον αλγόριθμο που λύνει το Sudoku `_delay_ms(50)` για 50 ms το οποίο καθυστερεί αρκετά τον αλγόριθμο και έχουμε τον απαραίτητο χρόνο. Ετσι παρατηρώντας τα LED και αφού ανάψει το 2^ο που ξέρουμε ότι έχουν συμπληρωθεί από 20 έως 30 κελιά (και μάλιστα τα πρώτα 20-30 κελιά γιατί ο πίνακας ήταν αρχικά κενός) τότε στέλνουμε break και παίρνουμε OK. Στη συνέχεια περιμένουμε λίγο και βλέπουμε πως όντως παύουν να αναβουν άλλα led που δείχνει πως έχει σταματήσει η επίλυση του Sudoku. Με Debug κοιτάμε όντως ενδεικτικά κάποια κελιά και επιβεβαιώνουμε ότι κάποια είναι συμπληρωμένα και κάποια όχι όπως είχαμε προβλέψει. Επειτα πατάμε πάλι P και η επίλυση συνεχίζει από εκεί που είχε μείνει και μόλις λυθεί παίρνουμε Done. Στην παρακάτω εικόνα φαίνεται η διαδικασία ελέγχου του break όπως περιγράφηκε.



Το delay είναι σε σχόλια ώστε να μην επηρεάζεται η απόδοση του αλγορίθμου ωστόσο σε περίπτωση που χρειάζεται να ελεγχθεί η λειτουργικότητα του break μπορεί να βγει από σχόλια. Η F_CPU είναι defined στην αρχή του προγράμματος για να λειτουργήσει σωστά η βιβλιοθήκη delay.

Αρχικοποιήσεις:

Οι αρχικοποιήσεις που γίνονται στην αρχή αφορούν την σειριακή θύρα, τον πίνακα και την PORTB. . Σύμφωνα με τις απαιτήσεις πρέπει να έχουμε baud rate 9600. Για να το πετύχουμε αυτό, δεδομένου ότι το ρολόι μας είναι 10 MHz, πρέπει να θέσουμε στον UBRR την τιμή 64₁₀ ή ισοδύναμα 0x40. Ο υπολογισμός αυτός γίνεται μέσω του τύπου που παρουσιάζεται πολύ ωραία στον ακόλουθο σύνδεσμο:

<https://maxembedded.com/2013/09/the-usart-of-the-avr/>

$$UBRR = \frac{f_{osc}}{16 \cdot BAUD} - 1$$

$$UBRR = \frac{10 \cdot 10^6}{16 \cdot 9600} - 1$$

$$UBRR = \frac{10 \cdot 10^6}{153600} - 1$$

$$UBRR = 64,104$$

