ΕΝΣΩΜΑΤΩΜΕΝΑ ΣΥΣΤΗΜΑΤΑ ΜΙΚΡΟΕΠΕΞΕΡΓΑΣΤΩΝ ΗΡΥ 411

LAB41140566

Milestone 3: Othello Game Project

Φλέγγας Γεώργιος 2014030161

Χατζηπέτρος Αλέξανδρος 2013030151

24/05/2019

ΗΜΜΥ

Εισαγωγή:

Για το πρότζεκτ του μαθήματος Ενσωματωμένα Συστήματα Μικροεπεξεργαστών, μας ζητήθηκε η υλοποίηση ενός παιχνιδιού Othello με την χρήση του αναπτυξιακού STK500 το οποίο διαθέτει τον μικροελεγκτή ATMega16L. Το παιχνίδι θα διαθέτει τόσο την ικανότητα να παίξει ένας άνθρωπος εναντίον του AVR, καθώς και να αντιμετωπίσει κάποιο άλλο STK500. Το παιχνίδι θα παίζεται σε σκακιέρα 8x8 με τον παίχτη που έχει τα μαύρα να ξεκινάει πρώτος. Οι συντεταγμένες στον οριζόντιο άξονα θα δίνονται από αριθμούς 1-8, ενώ στον κάθετο από γράμματα A-H. Το παιχνίδι τελειώνει όταν δεν υπάρχουν κενές θέσεις στην σκακιέρα ή όταν υπάρχουν μεν κενές θέσεις αλλά δεν μπορούν να καλυφθούν από κανένα παίκτη. Νικητής θα είναι ο παίχτης με τα περισσότερα πούλια στο ταμπλό όταν λήξει το παιχνίδι.

Πρωτόκολλο επικοινωνίας:

Για να εξασφαλιστεί η δυνατότητα επικοινωνίας τόσο με τον χρήστη, όσο και με άλλα AVR, ήταν αναγκαίο να οριστεί ένα κοινό πρωτόκολλο επικοινωνίας. Για τον λόγο αυτό, εφαρμόστηκαν οι ακόλουθες ρυθμίσεις: χρήση κρυστάλλου 10MHz, ταχύτητα και το πρωτόκολλο της σειριακής θύρας 9600baud, 8Bits, 1Stop Bit, No Parity , default χρόνος κίνησης κάθε παίχτη 2 sec.

Για να είναι εφικτή η επικοινωνία μεταξύ AVR και αντιπάλου δημιουργήθηκαν οι ακόλουθες συναρτήσεις:

- `void UART_Init()` για την αρχικοποίηση της θύρας επικοινωνίας
- `void USART_Transmit( unsigned char data )` για την μετάδοση πληροφορίας
- `void USART_Transmit_Str(char data[])` για την μετάδοση string πληροφορίας
- `void AVR_Reciever(char Data[])` για την λήψη πληροφορίας από το pc

Τον κύριο ρόλο για την αποκωδικοποίηση των μηνυμάτων, παίζει η συνάρτηση `void AVR_Reciever(char Data[])`, η οποία θα επεξεργάζεται τα μηνύματα που δέχεται το STK500 από τον αντίπαλο χάρις στο USART_RXC_vect και θα συνεχίζει στις απαραίτητες ενέργειες . Ο τρόπος με τον οποίο αναγνωρίζει κάθε εντολή, είναι να συγκρίνει κάθε χαρακτήρα που θα αποθηκεύει στο Data[] κατά την ανάγνωση, με την ανάλογη ακολουθία σε ASCII. Ιδιαίτερη προσοχή δόθηκε στην περίπτωση που γίνεται λήψη του OK<CR>. Με σκοπό την ορθή, αντιμετώπιση της εντολή αυτής, αξιοποιήθηκε το `char LastTrasmit[10]`, το οποίο αποθηκεύει την τελευταία εντολή που μετέδωσε το STK500. Όταν ανιχνευθεί το OK<CR>, ελέγχει το περιεχόμενο του LastTrasmit[] και συγκινεί με τις πιθανές εντολές που μπορεί να μετέδωσε τελευταία και όταν βρει την σωστή περίπτωση προχωρά στην εκτέλεση των απαραίτητων ενεργειών.

Υλοποίηση Timer:

Για τον timer που θα κρατάει τον χρόνο μέσα στον οποίο πρέπει ο κάθε παίχτης να κάνει την κίνηση του χρησιμοποιήθηκε αρχικά ο 8-bit TIMER0, όμως λόγο προβλημάτων που προέκυψαν μετά την προσαρμογή του κρυστάλλου, αξιοποιήθηκε ο TIMER1 . Με βάση την χρήση του Overflow Interrupt TIMER1_OVF_vect και χρησιμοποιώντας prescaler 1024, προχωρήσαμε στην εφαρμογή της ακόλουθης φόρμουλας:

$$Ftimer = CPU\frac{freq}{Prescaler} = 10000000/1024 = 9765.625$$

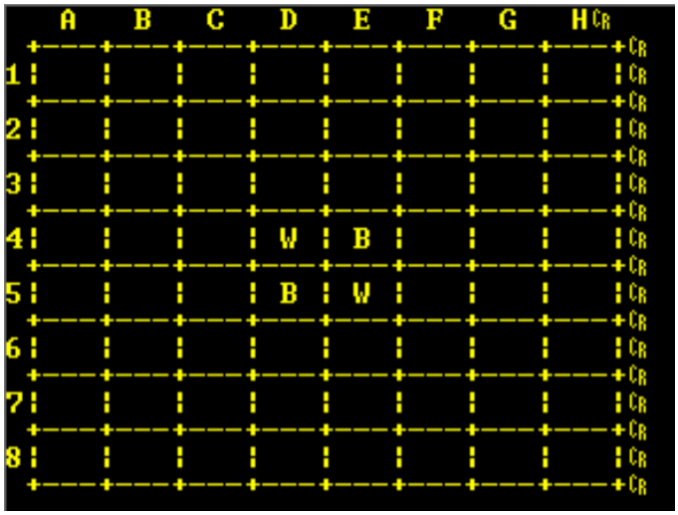
$$Ttick = \frac{1}{Ftimer} = 0.0001024$$

$$Targer = \frac{Time\,wewant}{Ttick}$$

Το παραπάνω για την default τιμή 2 sec, δίνει 19531. Επειδή ο TIMER1 κάνει overflow όταν φτάσει στην τιμή 65535, δίνεται στο t_count η τιμή 65535-Target, δηλαδή 46004 το οποίο θα ανατίθεται στο TCNT1 κάθε φορά που ο χρόνος ξεκινάει από την αρχή. Σε περίπτωση που δίνεται άλλη τιμή μέσα από την εντολή ST<SP>[1-9]<CR>, αφού ανιχνευθεί η περίπτωση από την AVR_Reciever και ελεγχθεί ότι η τιμή που δόθηκε είναι αποδεκτή, τότε μέσω της φόρμουλας θα τεθεί η κατάλληλη τιμή στο t_count. Για τις περιπτώσεις που το Target θα είναι μεγαλύτερο του 65535, θα ορίζεται αρχικά το t_count ως 65535 και παράλληλα θα υπολογίζεται το t_extend=target-65535. Όταν θα πραγματοποιηθεί υπερχείλιση, θα δοθεί η τιμή 65535-t_extend στο TCNT1, ώστε να μετρηθεί ο έξτρα χρόνος και μετά να δοθεί timeout=1. Το timeout είναι η μεταβλητή που θα ελέγχει το AVR, ώστε να διαγνώσει εάν υπήρξε παράβαση χρόνου ή όχι.

Πυρήνας παιχνιδιού:

Όπως ειπώθηκε κατά την εισαγωγή, το παιχνίδι θα παίζεται σε ένα ταμπλό 8x8.Για την υλοποίηση του, δημιουργήθηκαν οι συναρτήσεις `void BoardInit().` Το board θα είναι ουσιαστικά ένας δισδιάστατος πίνακας 8x8, `volatile uint8_t board[8][8]`. Με την χρήση της BoardInit, όλες του οι θέσεις αρχικοποιούνται ως *<SP>(32)* και στην συνέχεια οι θέσεις [3,3] και [4,4] παίρνουν την τιμή *W(87)* συμβολίζοντας το White, ενώ οι [3,4] και [4,3] την τιμή *B(66)* για το Black. Το αρχικό Board φαίνεται παρακάτω:



Η υλοποίηση της πλήρης λειτουργίας του παιχνιδιού βασίστηκε πάνω σε 2 κυρίως διαφορετικά κομμάτια : α) Διεπαφή επικοινωνίας χρήστη-AVR, β) Κεντρικό do-while loop το οποίο φροντίζει την ομαλή εκτέλεση των κινήσεων των παιχτών.

Με το α) ασχοληθήκαμε κατά την διεκπεραίωση του Milestone 1 και προσθέσαμε τα απαραίτητα κομμάτια κώδικα, ώστε να βεβαιώνουμε την ορθή επικοινωνία των 2 παιχτών. Αυτό επιτεύχθηκε κυρίως με την χρήση while-loops, τα οποία “κολλάνε” τον κώδικα μέχρι η αντίστοιχη wait μεταβλητή πάρει την τιμή 0, αφού το AVR λάβει το κατάλληλο μήνυμα Ιδιαίτερο ενδιαφέρον έχει το πώς θα αντιμετωπίσει το avr, την κίνηση του αντίπαλου παίχτη όταν λάβει το μήνυμα MV<SP>{[A-H][1,8]}<CR>, κάτι στο οποίο θα αναφερθούμε παρακάτω.
Το β) αποτελεί τον πυρήνα του παιχνιδιού και σε μορφή ψευδοκώδικα η κύρια ιδέα είναι η εξής:

```
board_init;
Get Players Color
do:
   if (black player):
        if (enemy's turn):
            if (valid_moves('W')):
                Passes = 0;
                PrintBoard(moves)
                read player's moves
            else:
                passes++
                if(passes<2):
                    Ask player to pass
                else:
                    Neither Player got a move, Game over
        if (avr's turn):
            if
                (valid_moves('W')
                ): Passes = 0;
                avr_move('w')
                Moves_Done++
            else:
                passes++
                if(passes<2):
                    Avr passes
                else:
                    Neither Player got a move, Game
    over black player next round
    if (white player):
       (. . .)
while((Moves_Done<64)&&(Passes<2)&&(End_Game!=1)&&(New_Game!=1))
calculate_score()
announce winner
```

Κάθε φορά θα παίζει πρώτος ο μαύρος παίχτης. Αν αυτός είναι ο αντίπαλος, τότε με την προϋπόθεση ότι έχει διαθέσιμες κινήσεις, οι οποίες ανιχνεύονται μέσω της συνάρτησης `int valid_moves(char turn)` και αποθηκεύονται στον πίνακα moves, το avr περιμένει από τον παίχτη να του στείλει την κίνηση του, ενώ σε περίπτωση που που δεν έχει κίνηση, θα περιμένει PASS. Αν και οι 2 παίχτες κάνουν PASS ο ένας μετά τον άλλον σημαίνει ότι δεν υπάρχουν άλλες διαθέσιμες κινήσεις και το παιχνίδι τερματίζει. Όταν το avr ανιχνεύσει την κίνηση θα την επεξεργαστεί μέσω της ακόλουθης διαδικασίας:

Αρχικά θα διαβάσει τις συντεταγμένες σε ascii (μετατρέποντας το γράμμα σε αριθμό) και θα τις μεταφέρει σε δεκαδική μορφή ώστε να εξυπηρετούν την υλοποίηση μας. Στην συνέχεια, αν ο χρήστης δεν έχει υπερβεί τον επιτρεπόμενο χρόνο και αν σε αυτές τις συντεταγμένες του πίνακα moves, υπάρχει η τιμή V(86 σε ascii), τότε προχωράει στην εκτέλεση της πράξης. Με τον τρόπο αυτό εξασφαλίζεται ότι έχουμε valid κίνηση και χρόνο. Αν υπάρχει παραβίαση σε μια από αυτές τις προϋποθέσεις, το avr στέλνει το κατάλληλο μήνυμα και περιμένει την απάντηση του χρήστη.

Για την εκτέλεση κάποιας κίνησης υλοποιήθηκε την συνάρτηση void make_move(int row, int col, char turn), η οποία ανάλογα με τις συντεταγμένες και το χρώμα του παίχτη κάνει τις κατάλληλες αλλαγές και τις αποθηκεύει στο board και αυξάνει το moves_done. Εν τέλη στέλνει Ok, τερματίζει το loop στο οποίο είχαμε κολλήσει περιμένοντας την κίνηση του παίχτη και επανεκκινεί τον χρονομετρητή.
Αν το avr είναι ο άσπρος παίχτης, και έχει διαθέσιμες κινήσεις τότε μέσω της συνάρτησης int avr_move(char turn), θα εκτελέσει την κίνηση του.

Παρακάτω ακολουθεί από μια σύντομη περιγραφή για τις συναρτήσεις valid_moves και make_move:
int valid_moves(char turn)
Η συνάρτηση αυτή, βοηθάει στο να ανιχνεύσουμε τις διαθέσιμες κινήσεις του παίχτη που παίζει στον εκάστοτε γύρο. Αρχικά αδειάζει τον πίνακα moves, γεμίζοντας τον με κενά(Space=32 ascii). Στην συνέχεια ξεκινάει αναζήτηση μέσα στον πίνακα board. Ελέγχει ένα-ένα τα κουτιά, σε περίπτωση που έχει ένα κουτί κάποιο πούλι, τότε τοποθετεί στον moves στην ίδια θέση το πούλι αυτό. Αν όμως το κουτί είναι κενό, ελέγχει εάν κάποιο από τα γειτονικά του κουτιά είναι πούλι του αντιπάλου. Σε περίπτωση που βρει αντίπαλο, ξεκινάει αναζήτηση προς κάθε κατεύθυνση "πατώντας" πάνω σε αντίπαλα πούλια, μέχρι να βρει κενό ή πούλι του παίχτη. Αν βρει κενό, τότε τερματίζει την αναζήτηση προς την κατεύθυνση αυτή, ενώ αν βρει πούλι του παίχτη θέτει το κουτί από το οποίο ξεκίνησε ως διαθέσιμη κίνηση του παίχτη, μαρκάροντας το στον πίνακα moves με V.
void make_move(int row, int col, char turn)
Η συνάρτηση αυτή, τοποθετεί το πούλι του παίχτη στις δοσμένες συντεταγμένες και ξεκινάει αναζήτηση προς κάθε κατεύθυνση "πατώντας" πάνω σε αντίπαλα πούλια, μέχρι να βρει πούλι του παίχτη. Όταν βρει ένα, ξεκινάει να προχωράει ανάποδα μέχρι να βρει την αρχική θέση και αλλάζει ένα-ένα τα πούλια του αντιπάλου.

Το παιχνίδι θα τερματίσει όταν: α) κάποιος από τους 2 παίχτης στείλει end game ή quit game, β) δεν υπάρχουν διαθέσιμες θέσεις στην σκακιέρα, γ) κανένας παίχτης δεν έχει διαθέσιμη κίνηση. Στην συνέχεια θα καταμετρηθούν τα πούλια και το AVR θα ανακοινώσει τον νικητή τόσο μέσα από κατάλληλο μήνυμα, καθώς και ανάβοντας ένα από τα 3 leds που είναι συνδεδεμένα μέσω του PORTB: Led1 για νίκη του AVR, Led2 για νίκη του παίχτη, Led3 για ισοπαλία.

Τακτική AVR:

Όσον αφορά την τακτική που θα ακολουθεί το AVR, αφού μελετήθηκαν οι προτεινόμενοι αλγόριθμοι (AlphaBeta, Monte Carlo) και επιλέχθηκε η υλοποίηση μια παραλλαγή του αλγορίθμου AlphaBeta. Η επιλογή αυτή έγινε λόγο του μικρού μεγέθους του ταμπλό, το οποίο δίνει την δυνατότητα να εξερευνήσουμε όλες τις δυνατές επιλογές μέσα στον μικρό διαθέσιμο χρόνο που έχουμε.

Στο προηγούμενο μέρος, αναφέρθηκε η συνάρτηση `avr_move`, η οποία εκτελεί την κίνηση του AVR. Το τρόπος σκέψης είναι ο εξής: Για κάθε διαθέσιμη κίνηση του avr, ελέγχει ποια θα είναι η καλύτερη δυνατή απάντηση από τον αντίπαλο και επιλέγει την κίνηση που θα έχει τα μικρότερα οφέλη για τον αντίπαλο. Για να το πετύχει αυτό αποθηκεύει τον board και τον moves σε προσωρινούς πίνακες και στην συνέχεια επιλέγει μια μια τις διαθέσιμες κινήσεις και τις εκτελεί. Όταν εκτελέσει μια κίνηση, καλεί την συνάρτηση `valid_moves(Opponent)` ώστε να αποκτήσεις τις διαθέσιμες κινήσεις του αντιπάλου. Στην συνέχεια. υπολογίζει την κίνηση του αντιπάλου, η οποία θα του επιφέρει το καλύτερο δυνατό score μέσω της συνάρτησης `int best_move(char player)` και επιστρέφει το μέγιστο score που θα επιτεύξει ο αντίπαλος δεδομένο της κίνησης του AVR. Το score αυτό συγκρίνεται με τα προηγούμενα score και σε περίπτωση που είναι μικρότερο, αποθηκεύονται οι συντεταγμένες της κίνησης του avr, αφού αυτή η κίνηση είναι καλύτερη από κάποια προηγούμενη. Εν συνεχεία, επαναφέρει το board στην κατάσταση πριν εκτελέσει την κίνηση και δοκιμάζει την επόμενη διαθέσιμη μέχρι να μην έχει άλλη. Εν τέλει, επαναφέρει το board για μια τελευταία φορά και εκτελεί την καλύτερη κίνηση που βρήκε.

Εκτέλεση:

Το αποτέλεσμα της εκτέλεσης του κώδικα μας φαίνεται στις παρακάτω εικόνες:

```
   A   B   C   D   E   F   G   H
 +---+---+---+---+---+---+---+---+
1| W |   | W |   |   | W |   |   |
 +---+---+---+---+---+---+---+---+
2| W | W | W |   | W | W |   |   |
 +---+---+---+---+---+---+---+---+
3| W | W | W | W | W |   |   |   |
 +---+---+---+---+---+---+---+---+
4| W |   | W | W | W |   |   |   |
 +---+---+---+---+---+---+---+---+
5| W | W | W | W | W | W |   |   |
 +---+---+---+---+---+---+---+---+
6| B |   | W |   | W | W | W |   |
 +---+---+---+---+---+---+---+---+
7|   |   |   | W |   |   | W |   |
 +---+---+---+---+---+---+---+---+
8|   |   |   |   |   |   |   |   |
 +---+---+---+---+---+---+---+---+
The final score is:AVR: 29 User: 1
 WN
```