

ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΕΠΕΞΕΡΓΑΣΤΩΝ

4 η ΑΣΚΗΣΗ

07/05/2021

**Παπαβασιλείου Μάριος 1059649**

**Ντάκος Γεώργιος 1059569**

**Ανάλυση κώδικα:**

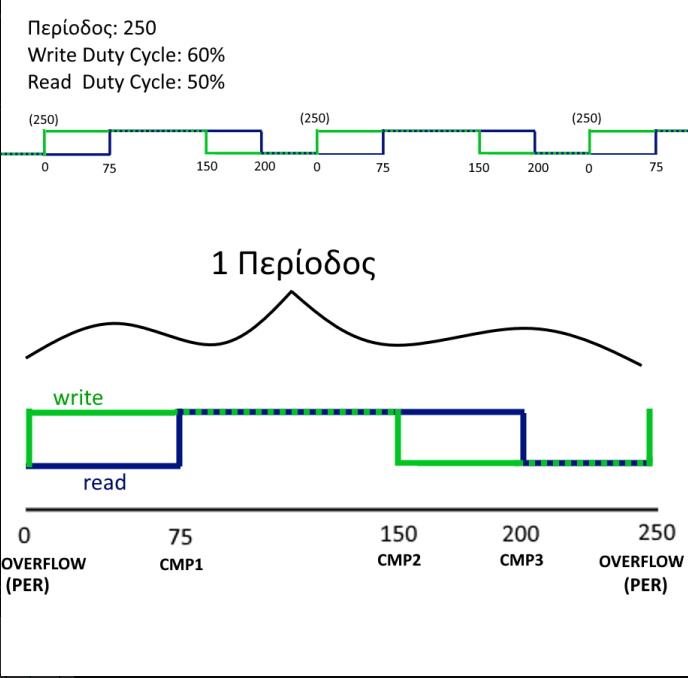
Στην συγκεκριμένη άσκηση μας ζητείται να φτιάξουμε μια μνήμη RAM των 32 θέσεων των 4 bit η κάθε μια και θα κάνει τις εξής λειτουργίες:

1. READ δεδομένων όταν sw6 = 1
2. WRITE δεδομένων όταν sw5= 1

Τώρα εμείς θέλουμε να αποφύγουμε κυρίως την κατάσταση να βρεθούμε και σε κατάσταση READ και σε WRITE διότι δεν γίνεται να κάνουμε ταυτόχρονα ανάγνωση και εγγραφή στο σύστημα μας. Αυτό θα το πέτυχουμε με την χρήση και ενεργοποίηση των αντίστοιχων interrupts, 2 διαφορετικών CMP καταχωριστών του ιδίου ρολογιού οπότε έτσι μπορούμε να διαμορφώσουμε 2 ειδών PWM:

1. Με 60% duty cycle
2. Me 50% duty cycle

Οι παραπάνω τιμές προφανώς θα μπορούσαν να ήταν και διαφορετικές με το κριτήριο μόνο να μην γίνεται ταυτόχρονα read και write. Τώρα οι παραπάνω τιμές έχουν προκύψει με το ότι έχουμε θέσει CMP1 = 150 και CMP2 = 200 (το CMP0 = 75 αν και το έχουμε βάλει μέσα στον κώδικα δεν έχουμε ενεργοποιήσει κάποιο από τα interrupt οπότε στην ουσία δεν έχει κάποια χρησιμότητα πάρα μόνο για διευκόλυνση και μελλοντική χρησιμότητα ) και έχοντας θέσει το PER = 250 τότε προκύπτει duty cycle για το write 150/250 = 0,6 δηλαδή 60% και για το read 125/250 = 0,5 δηλαδή 50% . Δηλαδή με λίγα λογία η θετική ακμή του read είναι στο 75 και πηγαίνει σε αρνητική στην τιμή 200 ενώ η θετική ακμή για το write είναι στο 0 και πηγαίνει σε αρνητική στην 150 όπως φαίνεται και στην παρακάτω εικόνα



Οπότε τώρα σύμφωνα με τα παραπάνω δεν πρόκειται να έχουμε read και write ταυτόχρονα. Τώρα για δικιά μας διευκόλυνση χρησιμοποιήσαμε 2 διαφορετικά port για τα switches αντί για 1 port με διαφορετικά PINS. Οπότε όταν PORTF.PIN5CTRL = 1 έχουμε read και όταν PORTC.PIN5CTRL = 1 έχουμε write. Αρά αναγκαστικά θα χρειαστούμε 2 διαφορετικές συναρτήσεις για τα interrupt αυτά ενώ είχαμε 1 PORT με 2 PINS για τα switch θα είχαμε μια συνάρτηση interrupt απλά πιο εμπλουτισμένη για την σωστή λειτουργία του προγράμματος. Τώρα όταν κάνουμε το PORTC.PIN5CTRL = 1 τότε έχουμε write μπαίνει μέσα στο interrupt του switch5 και κάνει μια μεταβλητή write ίση με ένα οπού μας δηλώνει ότι αναμένουμε εγγραφή σε μια θέση της μνήμης στην θετική ακμή του WRITE(δηλαδή OVF INTERRUPT) ή την αρνητική ακμή WRITE(δηλαδή CMP1 INTERRUPT) παλμού. Οπότε όταν βρει μια από τις 2 ακμές αυξάνουμε τον pointer που δείχνει σε κάποια θέση της μνήμης και αποθηκεύει τα δεδομένα της RandResult που πηρέ από την συνάρτηση rand στην θέση μνήμης στην οποία δείχνει ο rampointer εκείνη την στιγμή.

Τώρα αν έχουμε PORTF.PIN5CTRL = 1 τότε έχουμε read μπαίνει μέσα στο interrupt του switch6 και κάνει μια μεταβλητή read ίση με 1 οπού μας δηλώνει ότι αναμένουμε ανάγνωση σε μια θέση της μνήμης στην αρνητική ακμή του READ(δηλαδή CMP2 INTERRUPT) παλμού. Οπότε όταν βρει την αρνητική ακμή του READ παλμού τότε διαβάζει τα δεδομένα που έχει εκείνη θέση του πίνακα την οποία δείχνει ο rampointer εκείνη την στιγμή και ανάλογα με τα δεδομένα ανάβει καταλληλά τα 4 πρώτα bit του PORTD. Τώρα όταν τελειώσει η ανάγνωση η μεταβλητή NowReading γίνεται 1 οπότε και σβήνει όλα τα λαμπάκια. Αυτή η μεταβλητή NowReading έχει και μια άλλη χρησιμότητα σε περίπτωση που είχαμε READ ξανά κάνουμε READ να μην μπει στο 2ο READ αφού αυτή θέλουμε να είναι η λειτουργία της RAM μας. Με λίγα λογία αν θέλαμε κάποια άλλη λειτουργία για την RAM θα αλλάζαμε και αντίστοιχα τον κώδικα μας. Τώρα όσο αφορά ακόμα τις λειτουργίες της μνήμης όταν έχουμε WRITE και πατήσουμε και READ περιμένουμε πρώτα να τελειώσει το WRITE και μετά να κάνουμε READ. Το ίδιο ισχύει και για το ανάποδο.

**BONUS 1:**

Τώρα για το σορταρισμα της μνήμης χρησιμοποιούμε την συνάρτηση insertSortedValue(int NewElement). Όταν έχουμε write η συνάρτηση αυτή παίρνει ως όρισμα τον αριθμό που μου επέστρεψε η συνάρτηση rand() στην μεταβλητή RandResult. Στην αρχή θέτω έναν index ίσο με το 0 και κάνω την εξής διαδικασία: Στην αρχή μπαίνω σε μια λούπα οπού δεν θα κάνει τίποτα αφού υποτίθεται δεν έχω γράψει ακόμα κάτι στην μνήμη οπότε ο αριθμός θα πάει να γραφτεί στην 1η θέση της μνήμης και μετά θα αυξήσω το ποσά στοιχεία έχουν γραφτεί μέσα στην μνήμη και θα θέσω τον rampointer = i έτσι ώστε αν κάποια στιγμή γράψω και στις 32 θέσεις του και πάω να γράψω ακόμα μια τιμή σε μια παραπάνω θέση το πρόγραμμα να τερματίσει σύμφωνα με την while στην main.Τώρα αν κάποια τιμή που πάμε να γράψουμε είναι πιο μικρή από εκείνη που μας δείχνει ο index i τότε μπαίνω σε μια νέα συνάρτηση PushValue με όρισμα την τιμή του index i και μέσα σε μια λούπα σπρώχνω όλα τα στοιχεία της μνήμης από τη θεση που δείχνει ο index i μια θέση μπροστά οπότε μετρά πάω και γραφώ το νέο στοιχείο στην θέση i και έπειτα ξανά ανανεώνω το ποσά στοιχεία έχουν γραφτεί στην μνήμη και το που να δείχνει ο rampointer. Με λίγα λογία η PushValues μου γλιτώνει χρόνο διότι όλες οι τιμές που είναι μεγαλύτερης από την νέα τιμή που πάει να γραφτεί στην μνήμη θα ανέβουν μια θέση όπως πρέπει και δεν θα χρειάζεται να πάμε να ξανά ελέγξουμε αν είναι μεγαλύτερες ή μικρότερες από την αυτή την τιμή που πάμε να γράψουμε

**BONUS 2:**

**Μπλα μπλα**

