

Αναφορά:**1. Περιγραφή κώδικα**

Στη δεύτερη εργασία καλούμαστε να δημιουργήσουμε μια ενσωματωμένη συσκευή που θα μετρά το πόσο γρήγορα ένα άτομο πατάει το διακόπτη ως απάντηση σε ένα led που ανάβει. Αρχικά βλέπουμε τη συνάρτηση **void button_press_isr()** η οποία θα καλείται ως απάντηση στο πάτημα του κουμπιού (είναι ο interrupt handler μας) και το μόνο που έχει να κάνει είναι να δίνει σε μια μεταβλητή, με όνομα *signal* (δηλωμένη ως *volatile*, για να μην αγνοηθεί από τον compiler κατά το optimization), την τιμή 1. Αυτή την απόδοση τιμής στο *signal* μετά τη «βλέπει» η *main* και εκτελεί τις ανάλογες ενέργειες που θα αναφερθούν αργότερα.

Ύστερα βλέπουμε τη συνάρτηση **__asm void saveToMemory()** η οποία αναλαμβάνει να περάσει μια τιμή (που θα της δοθεί ως όρισμα) σε μια συγκεκριμένη θέση μνήμης που ορίσαμε (με αντίστοιχο τρόπο που είχαμε κληθεί να κάνουμε και στην πρώτη εργασία).

Μετά ακολουθεί η **main** όπου γίνεται η βασική δουλειά:

- Πρώτα κάνουμε τα απαραίτητα *initializations* στα *leds* και στο *switch* αξιοποιώντας συναρτήσεις έτοιμες (που βρίσκουμε στους *drivers* που ανέβηκαν στο *elearning*)

- Μετά ακολουθεί μια ατέρμονη **while** κατά την οποία πρώτα γίνεται ένα *delay* 5 δευτερολέπτον, ώστε να έχουμε ομαλή μετάβαση στο *response time test* 5 γύρων που ουσιαστικά καλούμαστε να υλοποιήσουμε.

- Μετά το πέρας του *delay* ανάβει το *led* με τη συνάρτηση *leds_set(1,0,0)* και μπαίνουμε σε μια νέα **while** η οποία «τρέχει» μέχρι η μεταβλητή *runCount* να φτάσει στην τιμή 5 (αφού χρειαζόμαστε 5 *response time* πειράματα).

- Μέσα σε αυτή την εσωτερική *while* βλέπουμε μια **if** η οποία περιέχει έναν κώδικα που εκτελείται αφότου πατηθεί το *switch*, ως απάντηση στο άναμμα του *led*, από τον χρήστη και θα εξηγηθεί παρακάτω.

- Αγνοώντας λοιπόν, μέχρι να πατηθεί το κουμπί, η εσωτερική *while* την *if* (τον κωδικα που περιέχει δηλαδή, γιατί τον έλεγχο τον πραγματοποιεί σε κάθε κύκλο της), η δουλειά της είναι να αυξάνει κατά ένα σε κάθε κύκλο της τη μεταβλητή *responseTime* και αμέσως μετά να γίνεται ένα *delay* 10000 κύκλων (*delay_cycles(10000)*).

- Κάποια στιγμή όμως ο χρήστης θα πατήσει το κουμπί, θα κληθεί ο *interrupt handler*, η μεταβλητή *signal*, που περιέχεται ως συνθήκη στην *if*, θα πάρει την τιμή 1 και θα μπορέσει να μπει το πρόγραμμά μας στον κώδικα που περιέχει η *if*:

- Εκεί αρχικά αποθηκεύεται η τιμή της μεταβλητής *responseTime* στην θέση *runCount* (μεταβλητή που τρέχει από το 0 έως το 4 και ουσιαστικά μας λέει σε ποιο από τα 5 πειράματα βρισκόμαστε) ενός πίνακα, μηδενίζεται η τιμή του *responseTime* (για να μετρήσει πάλι από την αρχή στο επόμενο πείραμα (εκ των 5) πόσους κυκλους θα έχουμε στην εσωτερική *while* μέχρι να μπούμε στην *if*), αυξάνουμε κατά ένα τη μεταβλητή *runCount*, επαναφέρουμε την τιμή του *signal* σε μηδέν, σβήνουμε το *led*, ακολουθεί ένα *delay* του οποίου το εύρος είναι από 2000 έως 5000ms (*delay=2000+(rand())%3000*) -> έτσι έχουμε μια τυχαιότητα ως προς το πότε θα ξεκινήσει το εκάστοτε τεστ, για να μην το περιμένει ο χρήστης και να είναι πιο ρεαλιστικό (*response time test*) και τέλος ανάβουμε πάλι το *led* ώστε να ακολουθήσει το επόμενο test.

- Όταν η μεταβλητή *runCount* πάρει την τιμή 5, βγαίνουμε από την εσωτερική *while*, και ακολουθούν οι εξής ενέργειες (μέσα στην ατέρμονη *while*):

- Αθροίζουμε, μέσω μιας *for*, τις τιμές που αποθηκεύτηκαν στον πίνακα 5 θέσεων που αναφέραμε πριν (που διατηρεί τις τιμές της μεταβλητής *responseTime*), στη μεταβλητή *meanResponseTime*, μετα διαιρούμε την *meanResponseTime* με το 5 για να βρούμε τον μέσο όρο της *responseTime*, περνάμε την *meanResponseTime* από κατάλληλες πράξεις για μετατροπή σε χρόνο (ms) και ακολουθεί η κλήση της *assembly* συνάρτησης *saveToMemory(meanResponseTime)*

ώστε να αποθηκευτεί η τιμή της *meanResponseTime* σε μια θέση μνήμης (η *meanResponseTime* καταλήγει να έχει τον μέσο χρόνο αντίδρασης σε ms, άρα στη θέση μνήμης μπαίνουν τα ms που πέρασαν μέχρι να αντιδράσουμε). Ακολουθεί ένα *delay* 5 δευτερολέπτων (ουσιαστικά ανα κύκλο έχουμε 10sec αναμονή ώστε να διακρίνονται τα 5-round tests μεταξύ τους), και ξεκινάει η προηγούμενη διαδικασία απο την αρχή, ώστε να έρθει το επόμενο 5-round response test.

Αντίστοιχη διαδικασία ακολουθείται και με την τροποποίηση που ζητήθηκε, κατά την οποία χωρίς το `#define MODE`, το *led* ξεκινάει αναμένο (πριν ξεκινήσουν τα *test*), και όταν σβήσει ο χρήστης πρέπει να αντιδράσει πατώντας το κουμπί ώστε να το ανάψει.

Όπως αναφέρθηκε, μέχρι να πατηθεί το κουμπί (να αντιδράσει ο χρήστης) το πρόγραμμά μας ουσιαστικά τρέχει συνεχώς σε μια λούπα όπου κάνει έναν έλεγχο μιας συνθήκης (*if(switch)*), αυξάνει μια μεταβλητή (*responseTime++*), περιμένει για 10000 κύκλους και πάλι απο την αρχή. Βάζοντας λοιπόν το *delay* των 10000 κύκλων, καταφέρνουμε ουσιαστικά μέσω της μεταβλητής *responseTime* να μετράμε πόσες φορές εκτελέστηκαν 10000 κύκλοι επεξεργαστή. (Όπως φαίνεται απο τον *debugger*, η εντολή *responseTime++* «μεταφράζεται» σε 5 εντολές *assembly*, και ο έλεγχος της *if* σε 3, άρα οι 10.000 κύκλοι υπερκαλύπτουν τους κύκλους των ενδιάμεσων εντολών και πρακτικά, **μπορούμε να πούμε ότι σε κάθε *responseTime++*, μετράμε 10.000 κύκλους επεξεργαστή.**)

	1 out of 5	2 out of 5	3 out of 5	4 out of 5	5 out of 5	Mean
Round 1	401	384	409	440	435	413
(ms)	250.625	240	255.625	275	271.875	258.125
Round 2	452	419	434	441	397	428
(ms)	282.5	261.875	271.25	275.625	248.125	267.5
Round 3	523	482	493	472	462	486
(ms)	326.875	301.25	308.125	295	288.75	303.75
Round 4	469	440	479	475	414	455
(ms)	293.125	275	299.375	296.875	258.75	284.375

Watch 1		
Name	Value	Type
meanResponseTime	486	int
rtArray[0]	523	int
rtArray[1]	482	int
rtArray[2]	493	int
rtArray[3]	472	int
rtArray[4]	462	int
<Enter expression>		

Στον παραπάνω πίνακα βλέπουμε μερικά αποτελέσματα από 4 γύρους των 5 τεστ:

- Οι τιμές αυτές πάρθηκαν με τη βοήθεια του **watch window** του *keil*.
- Στο **Round 1**, στο 1 out of 5, βλέπουμε την τιμή 401. Αυτό δηλώνει ότι η εσωτερική *while* πρόλαβε να αυξήσει την τιμή του *responseTime* 401 φορές, άρα μετρήσαμε **(401*10000) κύκλους** επεξεργαστή πρακτικά, μέχρι να πατήσουμε το *switch*. Από κάτω αυτό μεταφράζεται σε χρόνο απόκρισης κάνοντας την εξής πράξη: $401 * 10000 * (1/16.000.000) * 1000 = 250.625ms$ Αφού πήραμε 401 φορές το *delay* των 10000 κύκλων, και το ρολόι μας είναι 16Mhz, επί 1000 μας δίνει τον χρόνο αντίδρασης σε ms. Έτσι φαίνεται κάθε φορά **πόση δουλειά μπορεί να κάνει ο επεξεργαστής** μέχρι να αντιδράσουμε εμείς, και με την πράξη αυτή βλέπουμε **τον χρόνο αντίδρασής μας**. Οι τιμές 401, 384, 409 κτλ που βλέπουμε στο Round 1 πάρθηκαν με τη βοήθεια του *watch window* βλέποντας τις 5 θέσεις του πίνακα που αναφέρθηκε προηγουμένως ότι αποθηκεύει την τιμή της μεταβλητής *responseTime*.

1. Testing

Για να διαπιστωθεί η σωστή λειτουργία του προγράμματος, χρησιμοποιήθηκε το *watch window* σε διάφορες περιστάσεις, ώστε να δούμε ότι οι μεταβλητές που ορίστηκαν στο πρόγραμμα έπαιρναν ανά πάσα στιγμή αναμενόμενες/σωστές τιμές. Επίσης βάσει των αποτελεσμάτων των διαφόρων *test* βλέπουμε μέσους χρόνους σε λογικές τιμές ms, μιας και ο χρόνος αντίδρασης του ανθρώπου είναι κοντα στα 300ms. Επίσης με το πέρας της συνάρτησης *saveToMemory(meanResponseTime)*, ελέγχουμε τη θέση μνήμης που μας ενδιαφέρει με τη βοήθεια του παραθύρου Memory 1 του *keil*.

(Η *meanResponseTime* έχει το μέσο όρο των *responseTime*, και περνάει απο την εξίσωση για μετατροπή σε ms, και μετά αποθηκεύεται στη θέση μνήμης.)

Memory 1	
meanResponseTime = 286	
after it was converted to ms	
Address: 0x20000451	
0x20000451: 0000000286 0000000000 00000000	
and then saved to memory address 0x20000451.	
Watch 1	
Name	Value
meanResponseTime	286
rtArray[0]	430
rtArray[1]	448
rtArray[2]	417
rtArray[3]	514
rtArray[4]	481
<Enter expression>	