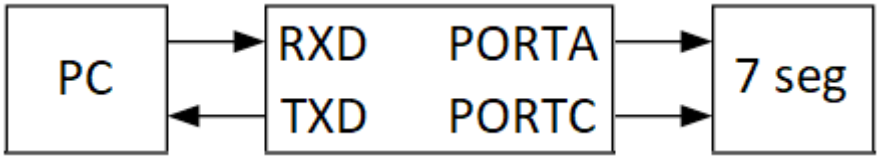
HPY411- Ενσωματωμένα Συστήματα Μικροεπεξεργαστών

Εργαστήριο 5

LAB41145851 **10/11/2020**

Εμμανουήλ Πετράκος ΑΜ 2014030009

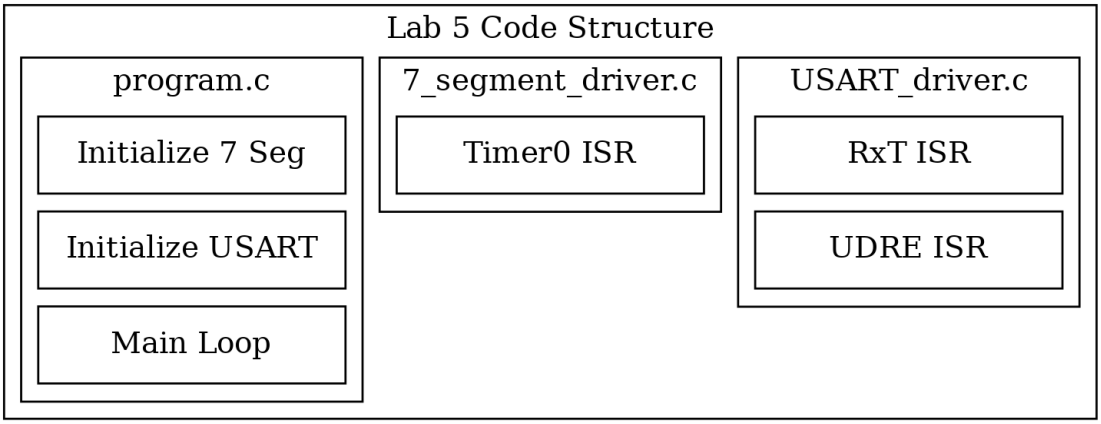
Στο πέμπτο εργαστήριο οι προδιαγραφές και η λειτουργικότητα του συστήματος παραμένουν ως έχει, αλλά ο κώδικας εξυπηρέτησης των interrupts υλοποιείται σε C. Και πάλι, ως είσοδος θεωρούνται τα δεδομένα που φτάνουν στον δέκτη του USART, ενώ ως έξοδος οι απαντήσεις στον πομπό του USART και τα 7 segment LEDs που ελέγχονται από τις θύρες A και C.



Top level schematic

Επεξήγηση προσέγγισης

Ο κώδικας του εργαστηρίου είναι υλοποιημένος εξολοκλήρου σε C και αποτελείται από τρία αρχεία. Στο πρώτο υλοποιούνται οι αρχικοποιήσεις και το main loop ενώ στα άλλα δύο οι συναρτήσεις εξυπηρέτησης των interrupts(ISR). Η δομή του κώδικα παρουσιάζεται στο παρακάτω διάγραμμα.



Για να υλοποιηθούν οι ISR σε C, χρησιμοποιείται το macro του avr-gcc ISR() με το κατάλληλο vector. Για παράδειγμα, η ISR του χρονομετρητή υλοποιείται ως ”`ISR( TIMER0_COMP_vect )`”.Πέρα από αυτό, δεν υπάρχει κάποιο αξιοσημείωτο κομμάτι του νέου κώδικα.

Η λειτουργικότητα του 7 segments driver παραμένει ίδια και υλοποιείται στην ISR του TIMER0 compare match. Οι θύρες A και C χρησιμοποιούνται ως έξοδοι των κωδικοποιήσεων και του

μετρητή ολίσθησης, ενώ ο χρονιστής του driver ρυθμίζεται στα 2ms ώστε η συχνότητα ανανέωσης των 7 segments να είναι 60Hz. Οι αναγκαίες ρυθμίσεις φαίνονται στον παρακάτω πίνακα.

Διεύθυνση	Τιμή	Ρύθμιση
DDRA	0xFF	Χρήση θύρας ως έξοδος
DDRC	0xFF	Χρήση θύρας ως έξοδος
PORTA	0xFF	Όλα τα LED off
PORTC	0x80	Εκκίνηση λειτουργίας από AN0
OCR0	77	60Hz/segment @ 10MHz F_CPU
TCCR0		
CS02:0	100	$clk_{I/O}$/256 (From prescaler)
WGM01:0	10	Compare mode: Clear Counter on match
TIMSK		
OCIE0	1	Ενεργοποίηση interrupt Compare Match

Σε κάθε interrupt, ο μετρητής ολίσθησης (PORTC) γίνεται shift κατά ένα ψηφίο και στην έξοδο δεδομένων (PORTA) φαίνεται η 7 segment κωδικοποίηση του αριθμού στην αντίστοιχη θέση μνήμης. Οι κωδικοποιήσεις είναι αποθηκευμένες στην SRAM και η πρόσβαση σε αυτές γίνεται με τον παρακάτω τρόπο.

Υπολογισμός κωδικοποίησης , όπου x το ενεργοποιημένο ψηφίο

$$\text{bcd\_number} = data[x]$$

$$PORTA = \text{segments\_encoding}[\text{bcd\_number}]$$

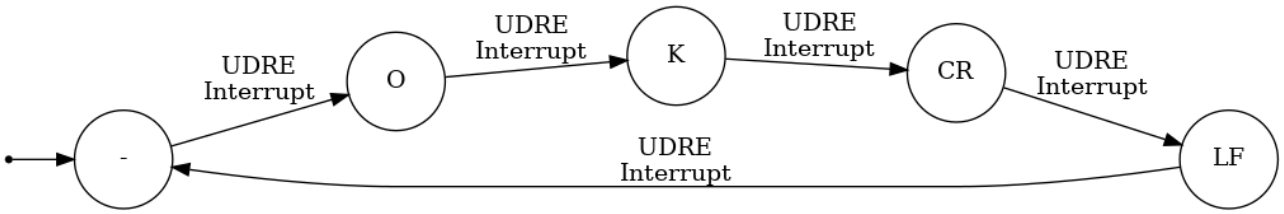
Κατά την αρχικοποίηση του USART ενεργοποιούνται ο receiver,ο transmitter και τα interrupts του receiver. Επίσης, ρυθμίζονται το baud-rate και η μορφή του frame σύμφωνα με τις προδιαγραφές. Οι απαραίτητες ρυθμίσεις φαίνονται στον παρακάτω πίνακα.

Διεύθυνση	Τιμή	Ρύθμιση
UBRR	64	Baudrate = 9600bps @ 10MHz F_CPU
UCSRB		
RXEN	1	Ενεργοποίηση receiver
RXCIE	1	Ενεργοποίηση receiver interrupts
TXEN	1	Ενεργοποίηση transmitter
UCSZ2	0	8 bit frame
UCSRC		
UMSEL	0	Ασύγχρονη λειτουργία
UPM1:0	00	0 bit Parity
UCSZ1:0	11	8 bit frame
USBS	0	1 stop bit

Όποτε ολοκληρώνεται η μετάδοση ενός χαρακτήρα προς τον μικροελεγκτή, ενεργοποιείται ένα interrupt. Η εξυπηρέτηση του γίνεται στην ρουτίνα ISR_URXC. Επειδή οι προδιαγραφές έχουν ξεκαθαρίσει ότι τα μηνύματα είναι ορθά και η αντιμετώπιση των χαρακτήρων είναι ανεξάρτητη των προηγούμενων/επόμενων, δεν χρειάζεται μια μηχανή πεπερασμένων καταστάσεων. Η λειτουργία της ρουτίνας είναι αντιδραστική και φαίνεται στον παρακάτω πίνακα.

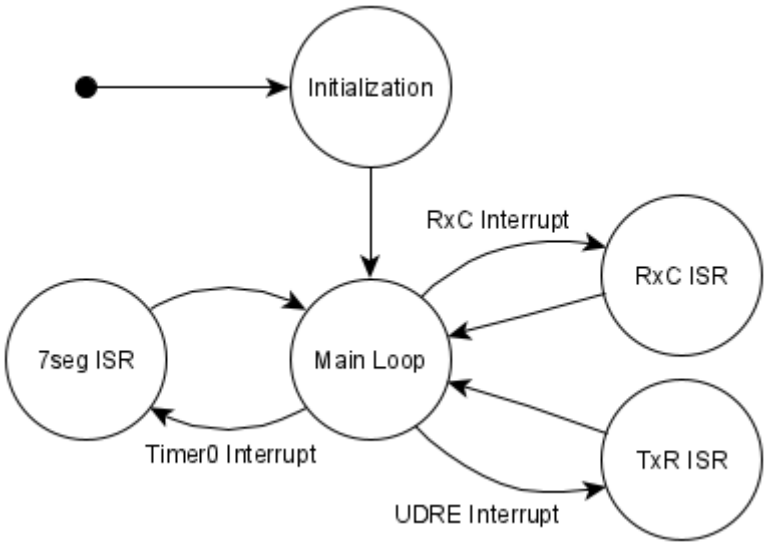
Είσοδος	Ενέργεια
C, N	Καθαρισμός δεδομένων μέσω της ρουτίνας clr_7seg_data
A, T, <CR>	Τίποτα
<LF>	Ενεργοποίηση Interrupt απάντησης(UDRIE), αύξηση μετρητή εκκρεμών απαντήσεων
0-9	ASCII → bcd, αποθήκευση αριθμού μέσω της ρουτίνας save_to_7seg_data

Η αποστολή των απαντήσεων γίνεται μέσω των interrupts που ενεργοποιεί ο transmitter όταν ο buffer του έχει διαθέσιμο χώρο (UDRE). Οι απαντήσεις αποτελούνται από 4 χαρακτήρες που πρέπει να σταλθούν σειριακά, για αυτό χρειάζεται η μηχανή πεπερασμένων καταστάσεων της παρακάτω εικόνας. Η απάντηση αποτελείται από 4 frames ενώ η μικρότερη εντολή από 3, άρα μπορεί να ενεργοποιηθεί νέα απάντηση πριν η FSM φτάσει σε ηρεμία (κατάσταση -) και να χαθεί. Για αυτό, υπάρχει ο μετρητής εκκρεμών απαντήσεων. Όταν ξεκινάει η αποστολή μια απάντησης μειώνεται κατά 1, ενώ αν στο τέλος της έχει την τιμή 0 απενεργοποιείται το interrupt.



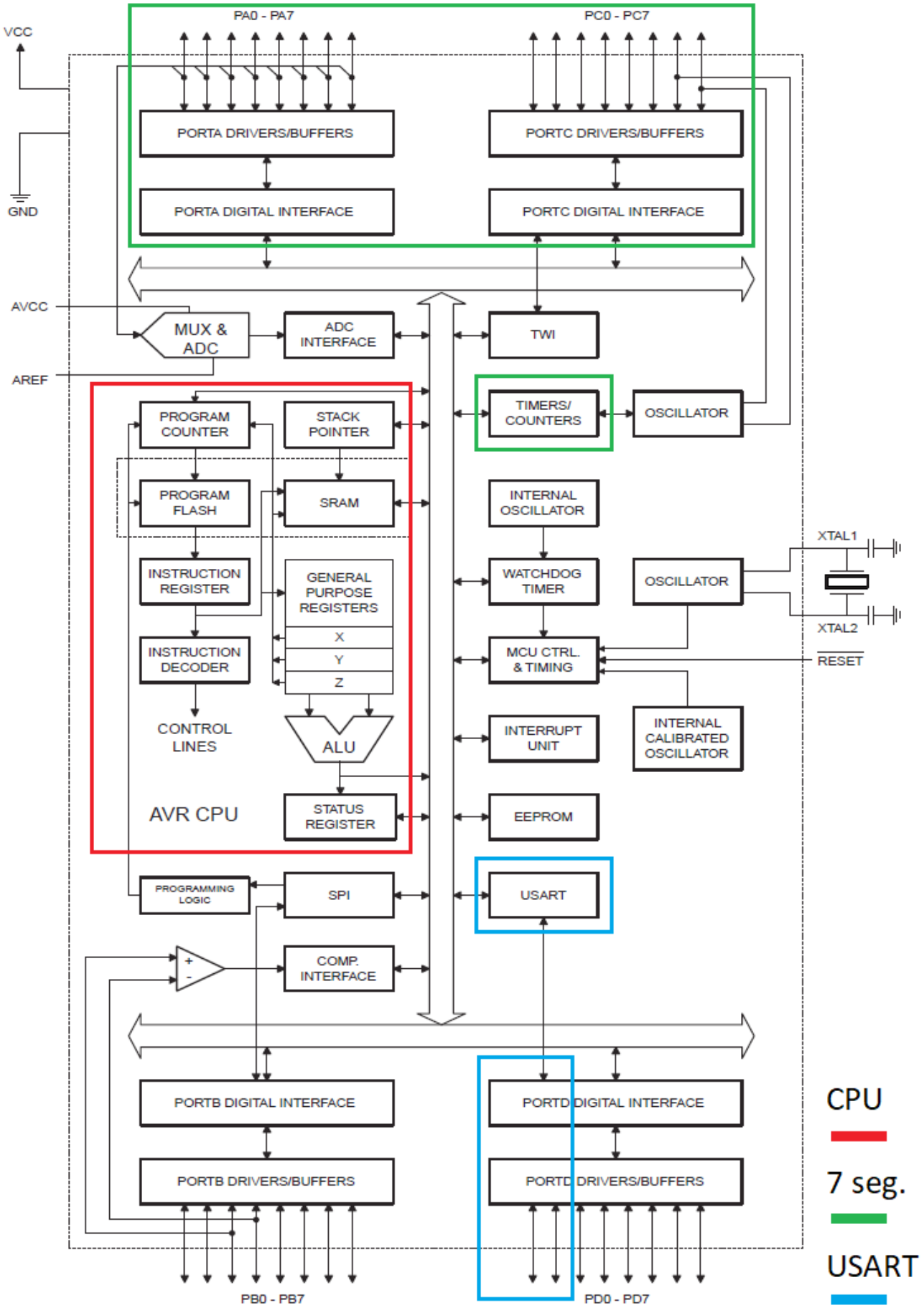
FSM αποστολής απαντήσεων

Η λειτουργικότητα του συστήματος εξυπηρετείται μέσω των interrupts και περιγράφεται στο παρακάτω διάγραμμα.



Top level FSM

Τέλος, παρουσιάζεται ο χάρτης χρησιμοποιούμενων πόρων.



Πειραματική Διαδικασία

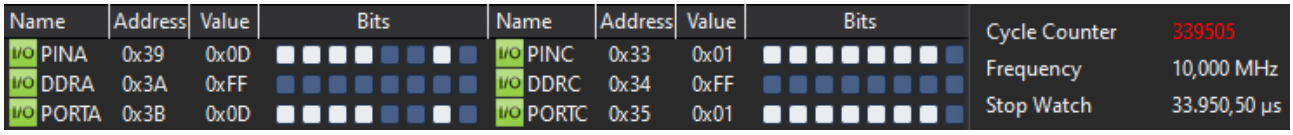
Για τον έλεγχο του USART driver παρακολουθούνται η μνήμη και οι απαντήσεις μετά από τις εντολές που βρίσκονται στο αρχείο usart.stim, ενώ για τον έλεγχο του 7 segment driver παρακολουθούνται οι PORTA και PORTC.

Βάζοντας ένα break point στην ISR του receiver, στην αντιμετώπιση των <CR> frames, φαίνεται η μνήμη μετά το τέλος ενός μηνύματος.

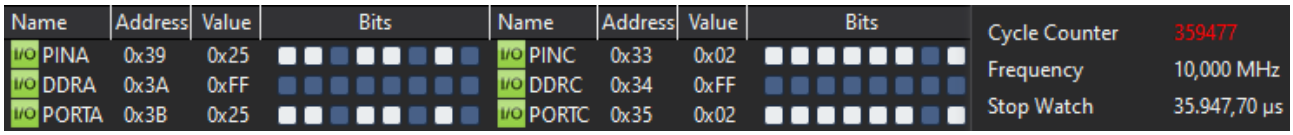
Εντολή	Κατάσταση μνήμης δεδομένων, 0x006d – 0x0074 (LSB → MSB)							
-	0a	0a	0a	0a	0a	0a	0a	0a
AT<CR><LF>	0a	0a	0a	0a	0a	0a	0a	0a
N1<CR><LF>	01	0a	0a	0a	0a	0a	0a	0a
N123<CR><LF>	03	02	01	0a	0a	0a	0a	0a

Στο αρχείο lab.log καταγράφονται οι απαντήσεις και εμφανίζεται η ακολουθία 0x4f, 0x4b, 0x0d, 0x0a τρεις φορές, δηλαδή όσα ήταν και τα εισερχόμενα μηνύματα.

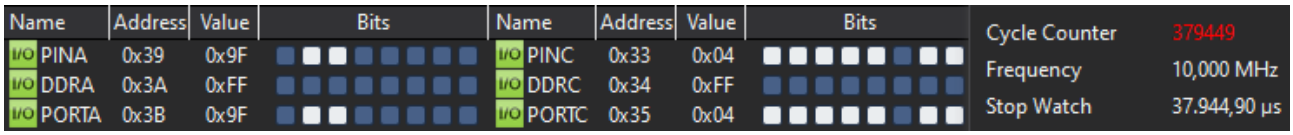
Χρησιμοποιώντας ένα breakpoint στο τέλος της ISR του χρονιστή, φαίνονται οι θύρες A και C μετά από κάθε αλλαγή. Στο παρακάτω παράδειγμα έχει ενεργοποιηθεί αμέσως μετά το τελευταίο εισερχόμενο μήνυμα από το USART.



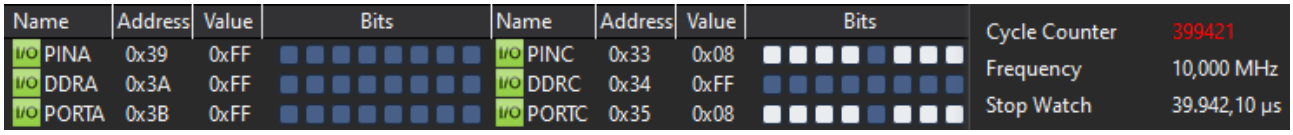
PORTA = 0b00001101 (3), PORTC = AN0



PORTA = 0b00100101 (2), PORTC = AN1



PORTA = 0b10011111 (1), PORTC = AN2



PORTA = 0b11111111 (τίποτα), PORTC = AN3

Ανάλυση & Παρατηρήσεις

Παρατηρούνται μερικές διαφορές στον παραγόμενο κώδικα assembly σε σχέση με τον κώδικα των προηγούμενων εργαστηρίων. Καταρχάς, όταν οι ISR ήταν υλοποιημένες σε assembly, αποτελούνταν από υπορουτίνες για να είναι ο κώδικας καλύτερα δομημένος και ευανάγνωστος. Τώρα που είναι υλοποιημένες σε C, ο κώδικας είναι συνοπτικός και δεν υπάρχει τέτοια ανάγκη. Σαν αποτέλεσμα, οι κλήσεις και επιστροφές υπορουτινών είναι λιγότερες στον παραγόμενο κώδικα. Επίσης, ο compiler εισάγει στις ISR κώδικα για push/pop των καταχωρητών που χρησιμοποιούνται (cpu, program counter, status). Τέλος, παρατηρήθηκε ότι ο compiler δεν βελτιστοποιεί τις προσβάσεις σε εξωτερικούς καταχωρητές. Πχ, στη παρακάτω εντολή παράγει δύο IN.

```
ring_counter = ( PORTC >> 7 ) | ( PORTC << 1 );
```

Αυτό αντιμετωπίζεται με τοπικές μεταβλητές.

Παρακολουθώντας την SRAM μέσω του simulator, φαίνεται ότι ο compiler χαρτογραφεί την μνήμη με τον παρακάτω τρόπο και ότι χρησιμοποιεί μέχρι 16 θέσεις μνήμης για το stack. Το stack χρησιμοποιείται εξολοκλήρου από τον compiler για push/pop καταχωρητών κατά την κλήση συναρτήσεων και interrupt, καθώς το πρόγραμμα δεν το χρησιμοποιεί απευθείας και οι συναρτήσεις που καλούνται δεν έχουν ορίσματα.

0x0000 ... 0x005F	Register address space
0x0060	Remaining transmits
0x0061	Transmitter state
0x0062 ... 0x006c	7 segment format 0 ... 7 segment format A
0x006d ... 0x0074	Data LBS ... Data MSB
...	Unused memory
0x0450 ... 0x045f	Stack

Χάρτης μνήμης

Αφού έχουν τροποποιηθεί οι υλοποιήσεις των ISR, η κατανάλωση των υπολογιστικών πόρων είναι διαφορετική. Ο 7 segment driver αντιμετωπίζει ένα interrupt κάθε 19968 κύκλους. Κατά μέσο όρο χρειάζονται 69 για τον υπολογισμό των εξόδων (55 με 83 κύκλους ανάλογα το AN). Άρα, ο driver καταναλώνει περίπου 0,345% της διαθέσιμης υπολογιστικής ισχύς.

Η κατανάλωση των ISR του USART δεν είναι σταθερή, καθώς ενεργοποιούνται από εξωτερικά μηνύματα. Για αυτό μελετάται η χειρότερη περίπτωση, η οποία είναι να έρχονται συνέχεια μηνύματα 8 αριθμών. Ένα τέτοιο μήνυμα αποτελείται από 11 frames, όπου ο χρόνος αποστολής ενός σε κύκλους υπολογίζεται ως εξής:

$$t_{frame} = \frac{\text{size}_{frame} * baudrate}{clk} = \frac{10\,bits * 9600\,bps}{10\,Mhz} = 10417\ \kappa\acute{\upsilon}\kappa\lambda o\iota\ \rho o\lambda o\gamma\iota o\acute{\upsilon}$$

Στους παρακάτω πίνακες φαίνεται πόσους κύκλους διαρκεί η κάθε ISR ανάλογα το frame.

Διάρκεια RxT ISR ανά frame (Σε κύκλους)						
C	N	A	T	<CR>	<LF>	Αριθμός<X>
102	104	54	56	58	68	151

Διάρκεια UDRE ISR ανά frame (Σε κύκλους)				
O	K	<CR>	<LF>	-
38	36	39	42	47

Λαμβάνοντας υπόψιν ότι ένα μήνυμα μπορεί να λαμβάνεται παράλληλα με την αποστολή της απάντησης για το προηγούμενο, το ποσοστό του υπολογιστικού χρόνου της CPU που χρειάζεται ο driver υπολογίζεται ως εξής:

$$\frac{N + 8*\text{<X>} + \text{<CR>} + \text{<LF>} + O + K + \text{<CR>}_{\alpha\pi\alpha\nu\tau} + \text{<LF>}_{\alpha\pi\alpha\nu\tau} + -}{frames * t_{frame}} * 100 = 1{,}43\,\%$$

Πρακτικά, δε θα βρεθεί σε τέτοια κατάσταση και υπό κανονική λειτουργία η απασχόληση του CPU θα είναι τάξης μεγέθους μικρότερη.

Τέλος, μπορούν να συγκριθούν οι υλοποιήσεις των drivers σε assembly και C. Στην περίπτωση του 7 segment driver δεν υπάρχει σημαντική διαφορά (~10%). Αντίθετα, στον USART driver η υλοποίηση σε C είναι περίπου 50% πιο αργή (0.9% μέγιστη κατανάλωση CPU στην assembly υλοποίηση).

Πηγές

ATmega16 manual
http://ww1.microchip.com/downloads/en/devicedoc/doc2466.pdf

AVR Libc User Manual
http://savannah.nongnu.org/download/avr-libc/avr-libc-user-manual-2.0.0.pdf.bz2