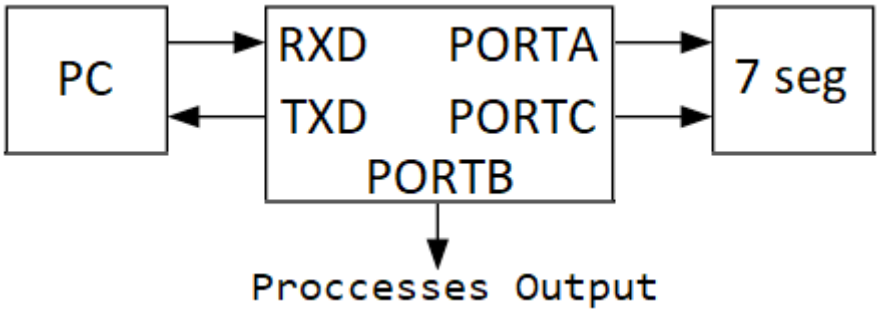
HPY411- Ενσωματωμένα Συστήματα Μικροεπεξεργαστών

Εργαστήριο 8

LAB41145851 **7/12/2020**

Εμμανουήλ Πετράκος ΑΜ 2014030009

Στο 8ο εργαστήριο ολοκληρώνεται η υλοποίηση του δρομολογητή διεργασιών. Η PORTB χρησιμοποιείται ως η κοινή έξοδος όλων των διεργασιών. Επίσης, έχει παραμείνει η λειτουργικότητα του εργαστηρίου 5.

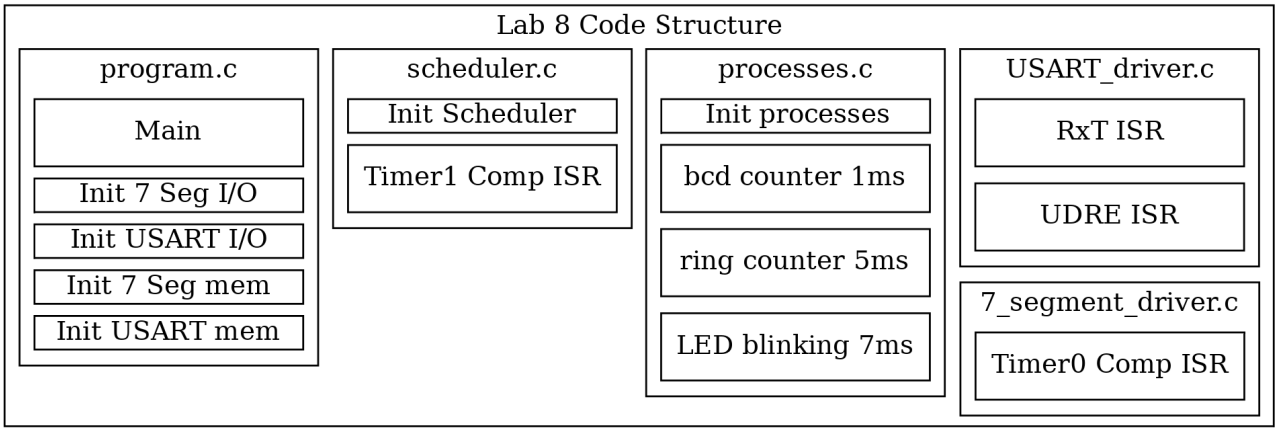


Αλληλεπίδραση μικροελεγκτή με τον έξω κόσμο.

Επεξήγηση προσέγγισης

Στο 8ο εργαστήριο έχουν γίνει οι παρακάτω αλλαγές στον κώδικα:

- Έχει επεκταθεί η συνάρτηση εξυπηρέτησης interrupts του receiver στον USART driver ώστε να δέχεται εντολές ενεργοποίησης/απενεργοποίησης διεργασιών.
- Στο νέο αρχείο scheduler.c, υλοποιείται ο δρομολογητής στην ρουτίνα αντιμετώπισης Interrupts του Timer1 καθώς και η αντίστοιχη συνάρτηση αρχικοποίησης.
- Στο αρχείο processes.c έχουν υλοποιηθεί 3 απλές διεργασίες για την δοκιμή του δρομολογητή, καθώς και μια συνάρτηση για την αρχικοποίηση τους.



Με τις νέες προδιαγραφές, η αντιμετώπιση των αριθμών που λαμβάνει ο receiver εξαρτάται από προηγούμενα frames. Άρα ο driver πρέπει να έχει μνήμη. Για αυτό, χρησιμοποιείται μια θέση μνήμης (receiver status) όπου στο ξεκίνημα ενός μηνύματος, αν αυτό πρόκειται να έχει αριθμούς, κρατάει τον τύπο του. Πέρα από αυτό, η λειτουργία του driver παραμένει ως έχει. Στον παρακάτω πίνακα φαίνεται αναλυτικά η αντιμετώπιση κάθε frame.

Είσοδος	Ενέργεια
C	Καθαρισμός δεδομένων 7 segment
N	Καθαρισμός δεδομένων 7 segment, receiver status = N
S	receiver status = S
Q	receiver status = Q
A, T, <CR>	Καμία
<LF>	Ενεργοποίηση Interrupt απάντησης (UDRIE), αύξηση μετρητή εκκρεμών απαντήσεων, καθαρισμός receiver status
Αριθμός	ASCII → bcd • Αν receiver status = N, αποθήκευση αριθμού στα δεδομένα 7 segment • Αν receiver status = S, ενεργοποίηση αντίστοιχου process • Αν receiver status = Q, απενεργοποίηση αντίστοιχου process

Η κατασκευή των 100ms timeslices επιτυγχάνεται χρησιμοποιώντας τον 16 bit Timer1 σε λειτουργία Clear Timer on Compare Match (CTC). Η ρύθμιση του χρόνου μέτρησης γίνεται μέσω του prescaler και των καταχωρητών σύγκρισης (OCR1A).

Η επιλογή του prescaler μπορεί να γίνει βάση του παρακάτω πίνακα. Με μικρότερο κβάντο χρόνου η μέτρηση είναι πιο ακριβής, αλλά για να γίνει η μέτρηση με ένα 16 bit μετρητή πρέπει να συμπληρώνονται 100ms με πλήθος κβάντων <= 2^{16} = 65536. Για αυτό επιλέγεται ο prescaler /64.

Prescaler	Ανάλυση /Κβάντο χρόνου	Μέγιστος χρόνος μέτρησης
No prescaler	0.1μs	6.5536 ms
/8	0.8μs	52.4288 ms
/64	6.4μs	419.4304 ms
/256	25.6μs	~1.668 s
/1024	102.4μs	~6.711 s

$f_{Clk\_Timer1} = f_{Clk} = 10MHz$

Η τιμή των καταχωρητών σύγκρισης υπολογίζεται διαιρώντας των επιθυμητό χρόνο με το κβάντο χρόνου.

$$OCR1A = \frac{T_{interrupt}}{t_{κβάντου}} - 1^{[1]} = \frac{100\,ms}{6.4\,\mu s} - 1 = 15624$$

[1] Ο μετρητής ξεκινάει από το κβάντο 0.

Οι αναγκαίες ρυθμίσεις για την αρχικοποίηση του δρομολογητή φαίνονται στον παρακάτω πίνακα.

Διεύθυνση	Τιμή	Ρύθμιση
scheduler_control	0x00	Απενεργοποίηση των διεργασιών
OCR1AH	15642 >> 8	100ms/Interrupt @ 10MHz F_CPU
OCR1AL	15642 & 0xFF	
TCCR1B		
CS02:0	100	$clk_{I/O}$/64 (From prescaler)
WGM13:10	0100	CTC, OCR1A
TIMSK		
OCIE1A	1	Ενεργοποίηση interrupt CTC, OCR1A

Η επιλογή της διεργασίας που θα εκτελεστεί σε κάθε timeslice γίνεται μέσω της ρουτίνας αντιμετώπισης των interrupts του TIMER1. Για να γίνει αυτό, πρέπει να γνωρίζει ποια διεργασία έτρεχε στο προηγούμενο και ποιες είναι ενεργοποιημένες. Αυτή η πληροφορία υπάρχει στην θέση μνήμης `scheduler_control` που φαίνεται αναλυτικά παρακάτω. Αν όλες οι διεργασίες είναι απενεργοποιημένες η ρουτίνα μηδενίζει τον shift register και επιστρέφει. Αλλιώς τον κάνει shift μέχρι να φτάσει σε μία ενεργοποιημένη διεργασία. Αν η μόνη ενεργοποιημένη διεργασία είναι αυτή που έτρεχε, γίνονται 3 shift και το επόμενο timeslice δίνεται σε αυτήν.

Bit	7	6	5	4	3	2	1	0	
	–	–	SCPR3	SCPR2	SCPR1	SCPE3	SCPE2	SCPE1	Scheduler Control
Init	0	0	0	0	0	0	0	0	

- **Bits 5..3 – SCPR3, SCPR2, SCPR1: Scheduler Process Running 3, 2, 1**

Αυτές οι τρεις θέσεις δείχνουν ποια διεργασία τρέχει στο παρόν timeslice. Αντιμετωπίζεται ως ένας shift register.

- **Bits 2..0 – SCPE3, SCPE2, SCPE1: Scheduler Process Enable 3, 2, 1**

Τα bits 2 έως 0 καθορίζουν ποιες διεργασίες είναι ενεργοποιημένες. Ενεργοποιούνται / Απενεργοποιούνται μέσω μηνυμάτων στο USART.

Η κλίση των διεργασιών γίνεται στην main. Μέσα σε ένα ατέρμον βρόχο καλείται συνέχεια η διεργασίας όπου το αντίστοιχο SCPR bit είναι ενεργό. Όλη η λογική του συστήματος υλοποιείται στις συναρτήσεις αντιμετώπισης των interrupts και δεν χρειάζεται κάτι παραπάνω.

Για την δοκιμή του δρομολογητή, έχουν δημιουργηθεί 3 απλές διεργασίες με κοινή έξοδο την θύρα B. Και οι τρεις περιέχουν κώδικα καθυστέρησης, ώστε να προσομοιώνουν πιο σύνθετες διεργασίες και να δοκιμάσουν την ανοχή του συστήματος σε απόκλιση των timeslices. Ο scheduler είναι non-preemptive και πρέπει να τελειώσει μια διεργασία για να ξεκινήσει η επόμενη. Είναι υλοποιημένες ως συναρτήσεις σε C, δηλαδή υλοποιούνται με call και return.

- bcd_counter_1ms(): Απλός αύξον μετρητής με καθυστέρηση 1ms. Χρησιμοποιεί την θέση μνήμης Bcd_counter_data.
- bcd_counter_1ms(): Μετρητής δακτυλίου με καθυστέρηση 5ms. Χρησιμοποιεί την θέση μνήμης ring_counter_data.
- LED_blinking_7ms(): Αντιστρέφει την τιμή της θέσης μνήμης LED_blinking_data με καθυστέρηση 7ms.

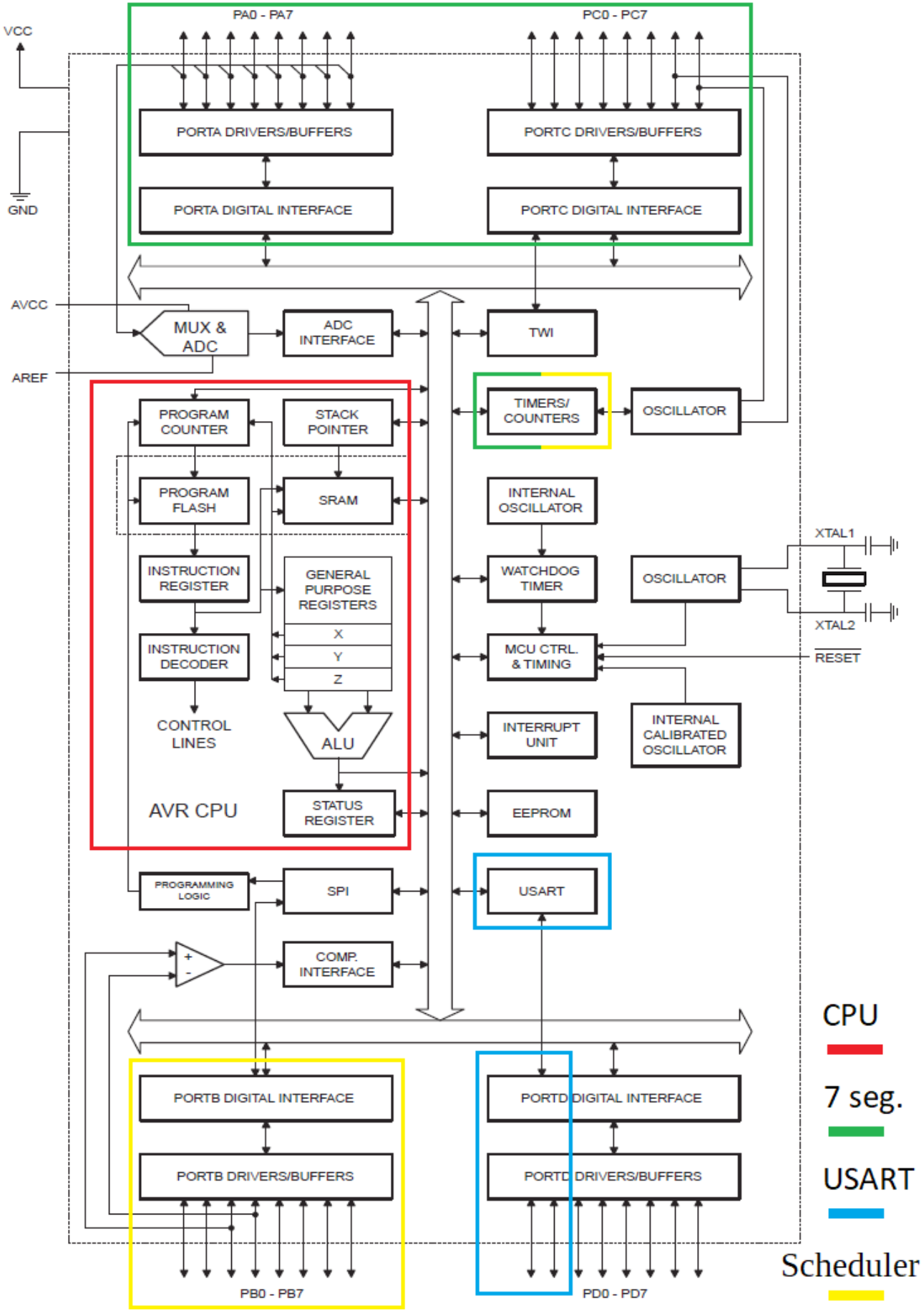
Η υλοποίηση του 7 Segment Driver είναι ίδια με το 5ο εργαστήριο και λειτουργεί συνεχώς.

Παρακάτω φαίνεται πως χαρτογραφεί την μνήμη ο compiler και υπογραμμίζονται οι αλλαγές για την υλοποίηση του δρομολογητή.

0x0000 ... 0x005F	Register address space
0x0060	LED blinking data
0x0061	Ring counter data
0x0062	Bcd counter data
0x0063	OK transmits left
0x0064	Transmitter status
0x0065	Receiver status
0x0066 ... 0x0070	7 segment format 0 ... 7 segment format A
0x0071 ... 0x0078	Data LSB ... Data MSB
0x0079	Scheduler control
...	Unused memory
... ... 0x045f	Stack

Χάρτης μνήμης

Τέλος, παρουσιάζεται ο χάρτης χρησιμοποιούμενων πόρων. Η θύρα B χρησιμοποιείται ως η κοινή έξοδος των διεργασιών που ελέγχονται από τον scheduler



Πειραματική Διαδικασία

Ο έλεγχος του συστήματος είναι σπασμένος σε δύο κομμάτια ώστε να είναι πιο εύκολη η ανάλυση του. Στο πρώτο κομμάτι δοκιμάζονται οι αλλαγές στο USART ενώ στο δεύτερο η λειτουργία των διεργασιών.

Στέλνοντας τα παρακάτω μηνύματα μέσω του stimulus file και θέτοντας ένα breakpoint στο τέλος της συνάρτησης εξυπηρέτησης τους, φαίνεται η κατάσταση της μνήμης του δρομολογητή μετά από αυτά.

Εντολή	Κατάσταση Scheduler Control (MSB→LSB)							
-	0	0	0	0	0	0	0	0
S1<CR><LF>	0	0	0	0	0	0	0	1
S2<CR><LF>	0	0	0	0	0	0	1	1
S3<CR><LF>	0	0	0	0	0	1	1	1
Q1<CR><LF>	0	0	0	0	0	1	1	0
Q2<CR><LF>	0	0	0	0	0	1	0	0
Q3<CR><LF>	0	0	0	0	0	0	0	0

Επίσης, στο αρχείο lab.log εμφανίζονται 6 απαντήσεις OK<CR><LF>.

Έχοντας ενεργοποιημένες και τις τρεις διεργασίες πρέπει να παίρνουν timeslices η μία μετά την άλλη. Θέτοντας από ένα breakpoint στο τέλος κάθε διεργασίας, παρακολουθούνται οι αλλαγές στην κοινή θύρα εξόδου. Επίσης, κοιτώντας την μνήμη κάθε μίας καθώς και το stop watch από το παράθυρο Processor Status, φαίνεται αναλυτικά η λειτουργία του συστήματος.

Χρόνος	SCPR	PORTB	Bcd_counter_data	ring_counter_data	LED_blinking_data
0 ms	000	0x00	0x00	0x80	0x00
...					
100 ms	100	0x00	0x00	0x80	0x00
107 ms	100	0xff	0x00	0x80	0xff
...	15 κλήσεις της LED_blinking_7ms()				
205 ms	010	0xff	0x00	0x80	0xff
210 ms	010	0x01	0x00	0x01	0xff
...	18 κλήσεις της ring_counter_5ms()				
300 ms	001	0x04	0x00	0x04	0xff
301 ms	001	0x01	0x01	0x04	0xff
...	100 κλήσεις της bcd_counter_1ms()				
400 ms	100	0x64	0x64	0x04	0xff
407 ms	100	0x00	0x64	0x04	0x00

Απενεργοποιώντας όλες τις διεργασίες, πρέπει όταν τελειώσει το τρέχον timeslice να μην τρέξει καμία άλλη διεργασία και η PORTB να μείνει με την τελευταία αλλαγή. Με ένα breakpoint στην ISR του TIMER1, παρατηρούνται τα παρακάτω αποτελέσματα.

Χρόνος	SCPR	PORTB	Bcd_counter_data	ring_counter_data	LED_blinking_data
505 ms	000	0x00	0x64	0x04	0x00
600 ms	000	0x00	0x64	0x04	0x00
700 ms	000	0x00	0x64	0x04	0x00

Τέλος, αν είναι ενεργοποιημένη μια διεργασία πρέπει να παίρνει όλα τα timeslices.

Χρόνος	SCPR	PORTB	Bcd_counter_data	ring_counter_data	LED_blinking_data
800 ms	001	0x00	0x64	0x04	0x00
900 ms	001	0xc7	0xc7	0x04	0x00
1000 ms	001	0x2b	0x2b	0x04	0x00

Ανάλυση & Παρατηρήσεις

Από τον πίνακα με τρεις ενεργοποιημένες διεργασίες πηγάζει μια σημαντική παρατήρηση. Στα 205 ms η έξοδος αλλάζει από την διεργασία LED_blinking_7ms() ενώ το timeslice έχει αλλάξει στα 200 ms και έχει δοθεί στην επόμενη. Αυτό συμβαίνει γιατί η τελευταία κλήση της LED_blinking_7ms() έγινε στα 198 ms και ο scheduler είναι non-preemptive, δηλαδή πρέπει να τελειώσει για να δοθεί ο επεξεργαστής στην επόμενη. Σαν αποτέλεσμα μια διεργασία μπορεί να κλέψει επεξεργαστικό χρόνο από την επόμενη.

Αυτό μπορεί να θεωρηθεί ως σημαντικό πρόβλημα ανάλογα την εφαρμογή και τις απαιτήσεις ακρίβειας. Μπορεί να αντιμετωπιστεί αν ο scheduler μετατραπεί σε preemptive, με την αντίστοιχη αύξηση σε πολυπλοκότητα και απαιτήσεις πόρων. Μια άλλη λύση είναι η χρονομέτρηση του timeslice να ξεκινάει αφού δοθεί ο έλεγχος του επεξεργαστή στην επόμενη διεργασία. Έτσι μπορεί οι διεργασίες να μεγαλώνουν τα timeslice τους, αλλά δεν χάνουν χρόνο οι επόμενες και η πολυπλοκότητα του scheduler αυξάνεται ελαφρά. Στην υλοποίηση του εργαστηρίου δεν χρειάζεται να γίνει κάτι τέτοιο, καθώς οι προδιαγραφές δίνουν ανοχή 10% στον χρόνο του timeslice και η χρονικά μεγαλύτερη διεργασία διαρκεί 7 ms (7% στον χρόνο του timeslice).

Οι διεργασίες που δεν εκτελούνται κάποια στιγμή, έχουν αποθηκευμένα τα δεδομένα τους και όταν ξαναέρχεται η σειρά τους συνεχίζουν από εκεί που είχαν μείνει. Δηλαδή δεν έχουν αντίληψη του context switch.

Επίσης, το context switch καταναλώνει πρακτικά μηδενικό χρόνο του επεξεργαστή, καθώς συμβαίνει κάθε 100 ms και χρειάζεται περίπου 100-200 κύκλους. Επίσης, ο κώδικας κλήσης των διεργασιών στην main εκτελείται κάθε μερικά ms και διαρκεί λίγους κύκλους, άρα δεν έχει αξιοσημείωτο αντίκτυπο στην επίδοση του συστήματος. Αυτό δεν θα ίσχυε αν ο χρόνος εκτέλεσης των διεργασιών είχε παρόμοια τάξη μεγέθους με τον χρόνο κλήσης τους (λίγους κύκλους).

Ο 7 segment driver λειτουργεί συνέχεια στο παρασκήνιο και κλέβει υπολογιστικό χρόνο από τις διεργασίες. Όπως έχει αναλυθεί σε προηγούμενα εργαστήρια, απαιτεί το 1.47% του επεξεργαστή. Συνυπολογίζοντας την ανακρίβεια των timeslices, στην χειρότερη περίπτωση μια διεργασία θα εκμεταλλευτεί το ~91.5% του timeslice της.

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