
Ενσωματωμένα Συστήματα Μικροεπεξεργαστών

Project 3

Ημερομηνία Παράδοσης: 7 Νοεμβρίου 2021

Αμπλιανίτης Κωνσταντίνος

2017030014

Σκοπός της Άσκησης:

Σκοπός του εργαστηρίου είναι η περαιτέρω εξοικείωση με τον AVR μικροελεγκτή. Πιο συγκεκριμένα σκοπός του εργαστηρίου είναι η διαχείριση δύο διεργασιών με την μορφή Interrupts, μία που αφορά τον Timer και μία που αφορά την σειριακή θύρα RS-232.

Περιγραφή της τεχνολογίας που χρησιμοποιήθηκε:

Για την υλοποίηση της συγκεκριμένης εργαστηριακής άσκησης χρησιμοποιήθηκε το Microchip Studio (version 7.0.2542) της Microchip Technology.

Περιγραφή επίλυσης της άσκησης:

Για την επίλυση της άσκησης κρατήθηκε σε μεγάλο βαθμό ο κώδικας που χρησιμοποιήθηκε για το Project 2. Οι αλλαγές που επέστησαν στον κώδικα του αφορούν κυρίως αντικατάσταση ορισμένων `rjmp` με `subroutines`, αρχικοποίηση της στοίβας και μικρές αλλαγές στη δομή. Για την χρήση της σειριακής θύρας RS-232, χρειάζεται η ενεργοποίηση του πρωτοκόλλου UART. Για την χρήση λοιπόν της RS-232 υλοποιήθηκαν οι εξής συναρτήσεις.

- **UARTInit** Υπέθυνη για την αρχικοποίηση-ενεργοποίηση του UART.
- **getc** Interrupt handler για την διαχείριση του interrupt εισόδου από την σειριακή θύρα.

καθώς και κάποια `subroutines` τα οποία καλούνται στην `getc`.

UARTInit Subroutine υπέθυνη για την αρχικοποίηση-ενεργοποίηση του UART.

Για την αρχικοποίηση-ενεργοποίηση του UART το πρώτο πράγμα που χρειάστηκε ήταν ο υπολογισμός του `baud-prescaler` που ανατίθεται στην θέση `UBRR`. Ο υπολογισμός της τιμής έγινε μέσω του παρακάτω τύπου.

$$\frac{F_{CPU}}{16USART_{BAUDRATE}} - 1$$

F_{CPU} συχνότητα ρολογιού

$USART_{BAUDRATE}$ Το επιθυμητό Baud rate(9600)

Επιπλέον για την σωστή αρχικοποίηση τίθενται το `UCSRB` και το `UCSRC` για την ενεργοποίηση του `recieve\transmit` και την αρχικοποίηση 8 data-bits 1 stop-bit αντίστοιχα.

getc Interrupt handler για την διαχείριση του interrupt εισόδου από την σειριακή θύρα

Η **getc** ήταν επιθυμητό να λειτουργεί ως interrupts εισόδου και όχι με polling. Για να γίνει αυτό, αρχικά βρέθηκαν οι τιμές ASCII των χαρακτήρων A, T, C, N, <CR>, <LF> καθώς και των αριθμών 0-9. Δεδομένων των παραπάνω, γίνονται διαδοχικές συγκρίσεις του χαρακτήρα εισόδου με τους χαρακτήρες ASCII και ανάλογα καλούνται τα subroutines που χρειάζεται.

```
; interrupt reciever. We recieve one char at the time. We have a switch like function
; to check what is the recieved character on ascii. the check is getting done with branches.
; in the end it returns.
getc:
    in r16, UDR
    in r16, UDR // need it to be there two times to make sure udr does not hangup
    mov r16, r15 // take the input from rs-232

    ; basically a switch
    ;a and t
    cpi r16, 0x41
    breq end_processing

    cpi r16, 0x54
    breq end_processing

    ;c and n
    cpi r16, 0x43
    breq empty_seven_seg

    cpi r16, 0x4E
    breq empty_seven_seg

    ;lf
    cpi r16, 0x0A
    breq lf_int

    ; for the numbers i have to cover an array of ascii chars.
    cpi r16, 0x30
    brlo end_processing
    ; i put this in after cheking the below values on purpose. I don't mention cr cause either way i skip the function
    cpi r16, 0x3A
    brlo data_processing

;end of the interrupt routine
end_processing:
    reti
```

Όπως φαίνεται στην εικόνα η συγκριση με τον ASCII της <CR> παραλείπεται καθώς η τιμή του <CR> περιλαμβάνεται σε αυτές κάτω του 0X30 και η διαδικασία χειρισμού του CR είναι η παράλειψη του έτσι και αλλιώς.

Subroutines της **getc**

empty_seven_seg Περιπτώσεις C και N.

Στην περίπτωση κλίσης της υπορουτίνας **empty_seven_seg**, γίνεται πρόσβαση στην sram στο σημείο που βρίσκονται τα byte υπεύθυνα για το PORTC. Στην συνέχεια γίνεται αρχικοποίηση κάθε byte στο 0X0A. Αυτό γίνεται γιατί όταν γίνει η πρόσβασή στο συγκεκριμένο byte και χρησιμοποιηθεί ως offset για την απεικόνιση της κατάστασης σε seven segment στην διεύθυνση 0X0A υπάρχει το 0XFF που σηματοδοτεί όλα σβηστά(λόγω κοινής ανόδου). Ο λόγος που αυτή η ρουτίνα καλείται τόσο στο C όσο και στο N είναι ότι και στις δύο περιπτώσεις είναι επιθυμητό να καθαρίζουμε τις καταστάσεις από προηγούμενες εισόδους για να μην υπάρχουν ανεπιθύμητες περιπτώσεις.

data_processing Περίπτωση αριθμού ως Input.

Σε περίπτωση που το input είναι αριθμός ακολουθείται η εξής διαδικασία. Αρχικά αφαιρείται το 0X30 ώστε η ASCII κωδικοποίηση να μετατραπεί σε αριθμός. Στην συνέχεια, μέσω διαδοχικών εντολών move και store (και θεωρώντας πάντα ότι το input μας είναι valid), γίνεται μετακίνηση των δεδομένων στην μνήμη μία θέση προς τα κάτω και το most significant bit χάνεται(δεν έχει σημασία για valid εισόδους γιατί θα είναι 0X0A δηλαδή σβηστό).

```
;data processing is responsible for the numbers.
;the basic thing i need to do is to practically push every
;stored bit in the next memory slot.
data_processing:

    ;I have the r16 which has the data input. I have to always put the input at the last spot(least significant bit)

    ;first thing i need to do is load both the sram locations that i keep

    ;portC segments that we will be storing the numbers
    ldi YH, HIGH(port_C_segment)
    ldi YL, LOW(port_C_segment)

    ; next thing to do is to find a way to be able to process the r16 as a number and not as an ascii code.
    ;The digits 0 to 9 are ASCII 30 hex to 39 hex, so i have just to subtract the 0x30 from r16 to have the number
    ; that i will use as offset in the memory with the Z register.
    subi r16, 0x30 ; subi uses one less cycle so we use that. Now we can use the r16 as offset.

    ldi r18, 0x00
    ldi r19, 0x07
store_loop:
    ld r17, Y
    st Y+, r16
    mov r16, r17
    inc r18
    cp r18, r19
    brne store_loop
    rjmp end_processing
```


lf_int Περίπτωση <LF> ως input

Στην συγκεκριμένη περίπτωση σημαίνει ότι το μήνυμα του input τελείωσε και θα πρέπει να σταλεί το OK<CR><LF>. Για αυτό το λόγο το συγκεκριμένο τμήμα κώδικα καλεί την συνάρτηση putc με ορίσματα τους ASCII characters που αντιστοιχούν στο παραπάνω μήνυμα. Η putc, η οποία λειτουργεί με polling, χρησιμοποιεί τον καταχωρητή TCNT2 (καθώς δεν θα χρησιμοποιηθεί αλλού και ο UDR θα κρεμάσει) και στέλνει τον αντίστοιχο ASCII character που έχει δοθεί από την lf_int.

Αποτελέσματα

Αρχικά παρατηρούμε ότι οι εντολές που δέχεται ο μικροελεγκτής λειτουργούν κανονικά αφού στο αρχείο που κρατάει τα outputs, για τέσσερις εισόδους έχουμε το εξής αποτέλεσμα:

```
#16028
TCNT2 = 0x47
#12
TCNT2 = 0x4b
#12
TCNT2 = 0x0d
#12
TCNT2 = 0x0a
#13964
TCNT2 = 0x47
#12
TCNT2 = 0x4b
#12
TCNT2 = 0x0d
#12
TCNT2 = 0x0a
#19964
TCNT2 = 0x47
#12
TCNT2 = 0x4b
#12
TCNT2 = 0x0d
#12
TCNT2 = 0x0a
#29964
TCNT2 = 0x47
#12
TCNT2 = 0x4b
#12
TCNT2 = 0x0d
#12
TCNT2 = 0x0a
```



OK<CR><LF>

4 επαναλήψεις OK<CR><LF>, όσες και οι φορές που κλήθηκε για input.

Ενδεικτικά Αποτελέσματα

Κατάσταση μνήμης μετά την εντολή C<CR><LF>

```
data 0x0060 0a 0a 0a 0a 0a 0a 0a 03 9f 25 0d 99 25 41 1d 01 09 ff 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x0082 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x00A4 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x00C6 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x00E8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x010A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Όπως φαίνεται μετά το C οι τιμές που είναι υπεύθυνες για τα seven segment του PORTC παίρνουν ως offset το 0X0A που αντιστοιχεί το 0XFF των καταστάσεων που θα δώσει κλειστή οθόνη.

Ενδεικτικά αποτελέσματα μετά τον δεύτερο αριθμό

[illegible]

Κατάσταση της Ram μετά την είσοδο του δεύτερου αριθμού.

Απόδειξη λειτουργίας του προγράμματος μέσω ενδεικτικών καταστάσεων (2,3,4)

PORTC		PORTD		PORTB							
I/O	PINC	0x33	0x02	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
I/O	DDRC	0x34	0xFF	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
I/O	PORTC	0x35	0x02	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

I/O	PINA	0x39	0x41	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
I/O	DDRA	0x3A	0xFF	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
I/O	PORTA	0x3B	0x41	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Όπως φαίνεται το offset είναι σωστό και μας δίνεται το σωστό situation από την μνήμη. Ομοίως και για τις άλλες καταστάσεις (παρουσιάζονται ενδεικτικά οι επόμενες δύο).

I/O	PINC	0x33	0x04								
I/O	DDRC	0x34	0xFF								
I/O	PORTC	0x35	0x04								
I/O	PINA	0x39	0x1D								
I/O	DDRA	0x3A	0xFF								
I/O	PORTA	0x3B	0x1D								

I/O	PINC	0x33	0x08	
I/O	DDRC	0x34	0xFF	
I/O	PORTC	0x35	0x08	
I/O	PINA	0x39	0x09	
I/O	DDRA	0x3A	0xFF	
I/O	PORTA	0x3B	0x09	