

Εργαστήριο 3

ΟΝΟΜΑ: Μαυρογιώργης Δημήτρης

ΑΜ: 2016030016

ΗΡΥ 411 - Ενσωματωμένα Συστήματα Μικροεπεξεργατών

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

April 19, 2021

Σκοπός

Σκοπός του τρίτου εργαστηρίου είναι η εξοικείωσή μας με τη σειριακή θύρα RS-232. Στο συγκεκριμένο εργαστήριο έχουμε δύο ανεξάρτητες διεργασίες που θα χρησιμοποιούν διαφορετικά Interrupts η κάθε μία. Η πρώτη διεργασία είναι αυτή που θα κάνει την ανανέωση των δεδομένων που εμφανίζουμε στα 7-segment LED, ενώ η δεύτερη θα επεξεργάζεται τα δεδομένα που λαμβάνονται και αποστέλονται από και προς τη σειριακή θύρα RS-232.

Περιγραφή της υλοποίησης

Στο συγκεκριμένο εργαστήριο εκτός από τις θύρες A και C του προηγούμενου εργαστηρίου χρησιμοποιήθηκαν και οι RXD, TXD για receive και transmit μέσω της σειριακής θύρα αντιστοιχα. Αρχικά, όπως λέει στο εγχειρίδιο του ATmega16, θα πρέπει να διασφαλίσουμε ότι τα Interrupts είναι απενεργοποιημένα και μόλις τελειώσει η αρχικοποίηση να ενεργοποιηθούν. Επιπλέον, για τη λειτουργία της σειριακής θύρας έπρεπε να ενεργοποιήσουμε το receive και transmit κάνοντας "1" τα bit RXEN και TXEN του καταχωρητή UCSRB. Για την ενεργοποίηση του receive interrupt αρχικοποιήθηκε σε "1" και το bit RXCIE.

Κατόπιν αρχικοποιήθηκε ο καταχωρητής UBRRL με την τιμή "0x41" (65₁₀) που είναι η τιμή BAUD για baud rate 9600 και συχνότητα 10 MHz. Τέλος, αφού έγινε "1" το bit URSEL του καταχωρητή UCSRC, για να μπορέσουμε να αρχικοποιήσουμε το συγκεκριμένο καταχωρητή και όχι το UBRR, τέθηκαν σε "1" τα bit UCSZ0 και UCSZ1 για να δηλώσουμε ότι έχουμε 8-bit κατά τη μετάδοση και λήψη. Τα bit UPM0 και UPM1 έμειναν "0", καθώς δεν έχουμε parity mode, ενώ το bit USBS έμεινε "0", επειδή έχουμε 1 stop bit.

Επεξήγηση Κώδικα

Αρχικά, στο data segment δεσμεύουμε χώρο 4 byte για το OK<CR><LF>. Έπειτα δηλώθηκαν κάποιες σταθερές που είναι οι HEX τιμές των γραμμάτων εντολών A, T, C, N, O, K και ο χαρακτήρας X που χρησιμοποιήθηκε για να ξέρουμε κάθε φορά ότι ήρθε κάποιος αριθμός από τη θύρα. Παράλληλα, δημιουργήθηκε μια ρουτίνα για clear της μνήμης, με την οποία αποθηκεύουμε την τιμή "0x0A" που σημαίνει ότι τα LED είναι σβηστά. Επιπλέον, επειδή κατά την ανανέωση των 7-segment διβάζουμε από τη διεύθυνση "0x80" προς την "0x87" και η ανανέωση γίνεται από το LS Digit προς το MS Digit, χρειάζεται μια ρουτίνα για να κάνει shift τα δεδομένα, ώστε να εμφανίζονται σωστά.

Όσον αφορά το handler για τη σειριακή θύρα και το receive των δεδομένων, η διεύθυνση μνήμης που βρίσκεται είναι η "0x16". Η υλοποίησή του είναι πολύ απλή, καθώς το μόνο που χρειάζεται να γίνει είναι να διαβάσουμε την τιμή του UDR σε ένα καταχωρητή και να ελέγχουμε με τον καταχωρητή r21, που έγινε define ως read_char, ότι διαβάζονται σωστά οι εντολές και ανάλογα με την εντολή να κάνουμε την κατάλληλη ενέργεια και να στείλουμε OK<CR><LF>, μόλις λάβουμε και το <LF> της εντολής που εκτελέστηκε.

```
.dseg
.org 0x60
    seg_dec_addr: .byte 11    //Decoded bcd to 7-seg
.org 0x80
    data_addr: .byte 8       //8 numbers to display
.org 0x90
    ok_msg_addr: .byte 4     //Here we store ok in memory
```

(a) Data segment

```
.cseg
.org 0x00 //Address of the beginning of the program
rjmp MEM_TCHT_INIT
.org 0x12 // Address of TIMER0_OVF interrupt handler
rjmp TIMER0_OVF
.org 0x16 // Address of USART_RXC interrupt handler
rjmp USART_RXC

.equ F_CPU = 1000000

.equ A_HEX = 0x41
.equ C_HEX = 0x43
.equ E_HEX = 0x45
.equ N_HEX = 0x4E
.equ O_HEX = 0x4F
.equ T_HEX = 0x54
.equ X_HEX = 0x5B // Used to indicate that we have stored a number - we don't care what number
.equ IF_HEX = 0x5A
.equ CR_HEX = 0x0D

.def digit_cnt = r20
.def read_char = r21
```

(b) Code segment - Definitions

```
USART_INIT:
cli // Disable Global Interrupt before initialization

ldi r16, (1<<EXCIE1)|(1<<RXEN)|(1<<TXEN) //Enable RXC interrupts, Receive and Transait
out UCSRB, r16

ldi r16, (1<<URSEL) // We make it "1" in order to write to UCSRC
out UCSRC, r16

ldi r16, (1<<UCSZ2)|(1<<UCSZ1) //8-bit size for everything we transmit and receive
out UCSRC, r16

//The baud rate parameter is assumed to be stored in the r17/r16 registers
ldi r16, 0x41 // 9600 baud rate means we have to set 65 to UBRRL
ldi r17, 0x00
out UBRRH, r17
out UBRRL, r16

sei // Enable Global Interrupt before initialization
```

(c) Usart Initialization

```
MEM_SHIFT:
ldi ZL, LOW(data_addr)
ldi ZH, HIGH(data_addr)
ldi r16, 0x06
add ZL, r16

ldi r16, 0x07

while_shift:
ldi r17, Z+
st Z, r17
subi ZL, 0x02
dec r16
brne while_shift

ret

MEM_CLEAR:
ldi ZL, LOW(data_addr)
ldi ZH, HIGH(data_addr)
ldi r16, 0x08
ldi r17, 0x0A

while_clear:
st Z+, r17
dec r16
brne while_clear

ret
```

(d) Memory clear and data shift function

```
SEND_OK:
ldi ZL, LOW(ok_msg_addr)
ldi ZH, HIGH(ok_msg_addr)
ldi r17, 0x04

USART_T:
sbis UCSRA, UDRE //UDRE is 1 when the UDR is empty.
rjmp USART_T //So we wait untill we send all the 4 bytes

ldi r18, Z+
//out UDR, r18
out TCNT2, r18
dec r17
brne USART_T

ldi read_char, 0x00
ret
```

(e) Function for sending OK

```
USART_RXC:
rcall RETC

cpi read_char, 0x00 // If read_char register has 0x00 and the value we receive is A, we store A
brne done_1
cpi r18, A_HEX
brne done_1
mov read_char, r18
reti

done_1:
cpi read_char, 0x00 // If read_char register has 0x00 and the value we receive is C or N, we store them and clear memory
brne done_2
cpi r18, C_HEX
brne if_1
cpi r18, N_HEX
brne done_2
if_1:
mov read_char, r18
rcall MEM_CLEAR
reti

done_2:
cpi read_char, A_HEX // If read_char register has A and the value we receive is T, we replace A with T
brne done_3
cpi r18, T_HEX
brne done_3
mov read_char, r18
reti
```

(f) Usart Interrupt Handler - A

```
done_3:
cpi r18, CR_HEX //If read_char register has CR and the value we receive is T or C or X (X means number),
brne done_4 //we replace them with 0
cpi read_char, X_HEX
brne if_2
cpi read_char, C_HEX
brne if_2
cpi read_char, X_HEX
brne done_5
if_2:
mov read_char, r18
reti

done_4:
cpi r18, 0x30 //If we receive a number 0-9 and the the read_char is N or an other number, we replace them with X
brll done_5 //and then we shift data in memory, store the new number and return
cpi r18, 0x0A
brne done_5
cpi read_char, N_HEX
brne if_3
cpi read_char, X_HEX
brne done_5
```

(g) Usart Interrupt Handler - B

```
if_3:
rcall MEM_SHIFT
subi r18, 0x20 // The mask we need to apply in order to isolate the bcd form of the number
sts data_addr, r18
ldi read_char, X_HEX
reti

done_5:
cpi read_char, CR_HEX //If read_char register has CR and the value we receive is LF, we send OK(CR+LF)
brne done_6
cpi r18, IF_HEX
brne done_6
rcall SEND_OK
reti

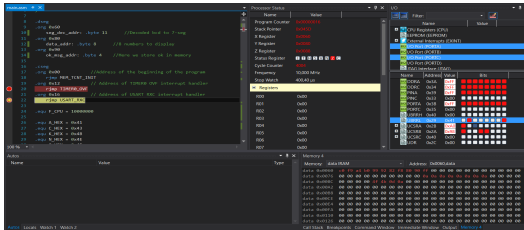
done_6:
ldi read_char, 0x00 //We clear read_char register if we receive an unknown char without the correct order
reti
```

(h) Usart Interrupt Handler - C

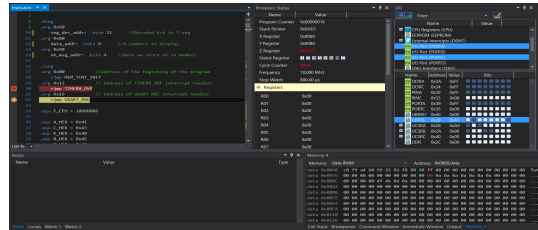
Προσομοίωση Αποτελεσμάτων

Όπως φαίνεται και στην εικόνα (a) όλες οι αρχικοποιήσεις που περιγράφηκαν παραπάνω γίνονται σωστά και για το USART και για τον TIMER/COUNTER0. Επιπλέον, Βλέπουμε ότι οι αποχωδικοποιήσεις των BCD, οι αριθμοί προς εμφάνιση και το OK<CR><LF> αποθηκεύονται στην SRAM. Στην επόμενη εικόνα παρατηρούμε ότι εκτελείται ο USART_RXC handler και αποθηκεύεται ο αριθμός 0 που λαμβάνουμε μέσω του USART. Στην εικόνα (c) βλέπουμε ότι κάποια στιγμή εκτελείται ο TIMER0_OVF handler και εμφανίζεται η τιμή 2 στο πρώτο digit. Έπειτα μετά από αρκετές εκτελέσεις του USART_RXC handler, βλέπουμε ότι και οι οχτώ τιμές που λήφθηκαν μέσω του USART αποθηκεύονται σωστά στη μνήμη (εικόνα (e)).

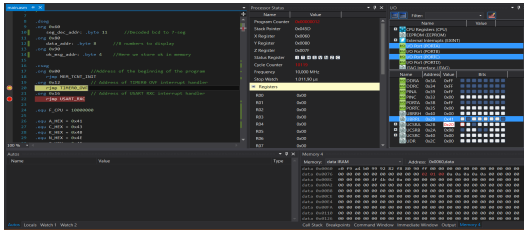
Στην ίδια εικόνα βλέπουμε ότι έρχεται για δεύτερη φορά ο TIMER0_OVF handler και στην επόμενη εικόνα βλέπουμε ότι εμφανίζεται το 6 που αποθηκεύσαμε στη δεύτερη θέση μνήμης στο digit 2. Μόλις τελειώσει η λήψη και του <LF>, εκτελείται μόνο ο TIMER0_OVF και γίνεται σωστά η ανανέωση των ψηφίων, όπως φαίνεται και στις επόμενες εικόνες. Στις εικόνες (g) και (h) φαίνεται ότι μετά την εμφάνιση του 6, εμφανίζεται το 5 στο ψηφίο 3 και στην επόμενη εκτέλεση εμφανίζεται το 4 στο digit 4 που είναι αποθηκευμένα στην τρίτη και τέταρτη θέση μνήμης αντίστοιχα. Συνεπώς, βλέπουμε ότι η λήψη και εμφάνιση των αριθμών γίνεται σωστά. Τέλος, όσον αφορά το OK<CR><LF> που στέλνουμε μετά από κάθε εντολή, φαίνεται ότι αποστέλνονται σωστά τα byte OK<CR><LF> στο αρχείο lab3.log, αφού γίνεται η εγγραφή τους σε HEX μορφή σε αυτό το αρχείο.



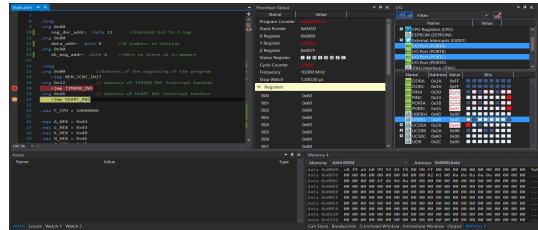
(a) Results from Atmel Studio 7 - Initialization of registers and memory



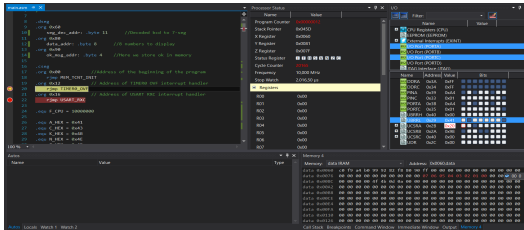
(b) Results from Atmel Studio 7 - Store of the first number



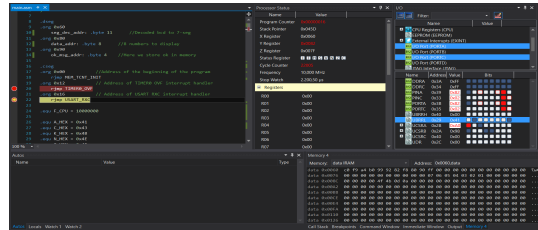
(c) Results from Atmel Studio 7 - Three number written and execution of TIMER0_OVF handler



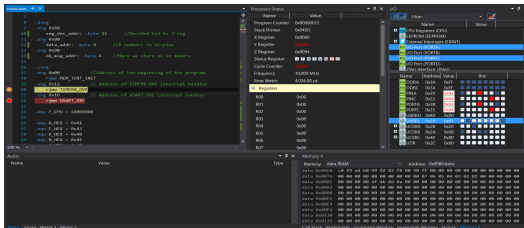
(d) Results from Atmel Studio 7 - After the execution of TIMER0_OVF handler



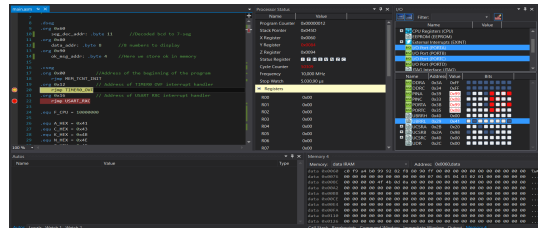
(e) Results from Atmel Studio 7 - All numbers stored and TIMER0_OVF handler is executed once again



(f) Results from Atmel Studio 7 - After the second execution of TIMER0_OVF handler



(g) Results from Atmel Studio 7 - After the third execution of TIMER0_OVF handler



(h) Results from Atmel Studio 7 - After the fourth execution of TIMER0_OVF handler