

## Ενσωματωμένα Συστήματα Μικροεπεξεργαστών – ΗΡΥ411

### Αναφορά Εργαστηρίου 3 – Εξοικείωση με την σειριακή θύρα στον Atmel AVR

**Ομάδα:** LAB41145877

Κιούλος Ευάγγελος 2016030056

#### Εισαγωγή:

Σκοπός της τρίτης εργαστηριακής άσκησης ήταν η εξοικείωση με την σειριακή θύρα USART του Atmel AVR. Μέσο της σειριακής θύρας εισάγουμε έναν αριθμό έως 8 ψηφία τον οποίο και εμφανίζουμε στην οθόνη που υλοποιήθηκε στο προηγούμενο εργαστήριο. Η σειριακή θύρα μπορεί να εκτελέσει τις εξής εντολές:

- AT<CR><LF>: Απαντά με το μήνυμα OK<CR><LF> προς το PC.
- C<CR><LF>: Καθαρισμός της οθόνης και έπειτα απάντηση OK προς το PC.
- N<X><CR><LF>: Καθαρισμός της οθόνης, εμφάνιση των χαρακτήρων <X> στην οθόνη και έπειτα απάντηση OK προς το PC.

Για τη σειριακή θύρα επιλέχθηκε απλή σύνδεση με 8 data bits. Για αυτό το λόγο, τα bits UCSZ01 και UCSZ00 του καταχωρητή UCSRC τέθηκαν στην τιμή '1' (Ref. Table 66. Σελ. 167 ATmega16 manual). Επιπλέον, επιλέχθηκε 1 stop bit και κανένα parity bit, επομένως, το bit USBS του καταχωρητή UCSRC τέθηκε στην τιμή '0' και τα bit UPM1 και UPM0 τέθηκαν στην τιμή '0' αντίστοιχα (Ref. Table 64. Και Table 65. Σελ. 166 ATmega16 manual). Επιπλέον, η ταχύτητα της σειριακής θύρας είναι 9600baud. Η τιμή του καταχωρητή UBRR προκύπτει από τον παρακάτω τύπο με  $f_{osc} = 10\text{MHz}$  και  $BAUD = 9600$ :

$$UBRR = \frac{f_{osc}}{16BAUD} - 1 = 64.$$

(Ref. Table 60. σελ. 147 ATmega16 manual).

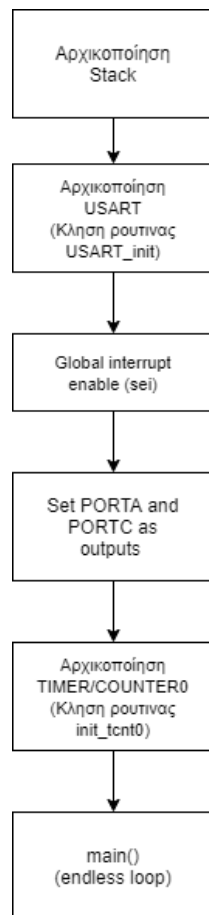
Τέλος, στον καταχωρητή UCSRB τα bit RXEN και TXEN τέθηκαν στην τιμή '1' για την ενεργοποίηση του transmit και του receive και το bit RXCIE τέθηκε στην τιμή '1' για την ενεργοποίηση του RX interrupt.

#### Υλοποίηση Κώδικα:

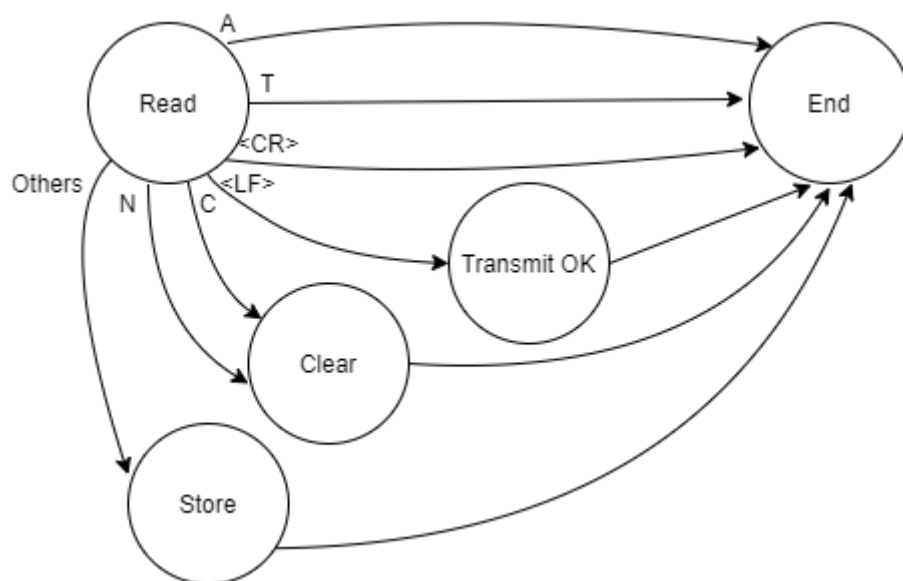
Αρχικά προστέθηκε ο χαρακτήρας "0x0A" στον κώδικα του δεύτερου εργαστηρίου ο οποίος αποκωδικοποιείται σε "0xFF" με αποτέλεσμα να μένει σβηστό κάποιο segment. Στην συνέχεια προστέθηκε η παρακάτω εντολή, με την οποία ορίζουμε την διεύθυνση του interrupt handler για το URX interrupt.

```
; We get here only when USART interrupt occurs.  
.org URXCaddr           ; Specify the address of USART RXD interrupt.  
rjmp ISR_uart           ; Jump to USART Interrupt service routine.
```

Η ροή του βασικού προγράμματος φαίνεται στο παρακάτω διάγραμμα:



Όταν συμβεί το interrupt URX, μεταβαίνουμε στον handler του συγκεκριμένου interrupt και στην συνέχεια στην ρουτίνα εξυπηρέτησης του interrupt (ISR\_uart). Η ρουτίνα εξυπηρέτησης του interrupt λειτουργεί σαν μία μηχανή πεπερασμένων καταστάσεων, η λειτουργία της οποίας φαίνεται στο παρακάτω σχεδιάγραμμα.



Η κατάσταση "Read" είναι η αρχική κατάσταση. Εκεί διαβάζουμε τα περιεχόμενα του καταχωρητή UDR και μεταβαίνουμε στην επόμενη κατάσταση που αντιστοιχεί στον χαρακτήρα που διαβάσαμε. Στην κατάσταση "Clear" σβήνουμε τα περιεχόμενα της οθόνης,

πανωγράφοντας στα περιεχόμενα της δεσμευμένης μνήμης RAM την τιμή “0x0A” και έπειτα μεταβαίνουμε στην κατάσταση “End”. Στην κατάσταση “Store” ολισθαίνουμε τα bytes της δεσμευμένης μνήμης RAM κατά μία θέση προς τα δεξιά, γράφουμε στην πρώτη θέση τα τέσσερα τελευταία bits του χαρακτήρα που διαβάστηκε από τον UDR και μεταβαίνουμε στην κατάσταση “End”. Στην κατάσταση “Transmit OK” στέλνουμε τους χαρακτήρες “OK<CR><LF>”. Μετά την μετάδοση, μεταβαίνουμε στην κατάσταση “End”. Τέλος, στην κατάσταση “End” επιστρέφουμε στο κυρίως πρόγραμμα με “reti”.

Για να αποθηκεύεται η κατάσταση που βρισκόμαστε κάθε φορά, χρησιμοποιείται ο καταχωρητής “r21” ενώ για την επόμενη κατάσταση ο καταχωρητής “r20”. Στην αρχή του προγράμματος ορίζεται σε ποια τιμή αντιστοιχεί η κάθε κατάσταση όπως βλέπουμε παρακάτω.

```
.def next_state = r20    ; r20 holds the next state
.def current_state = r21    ; r21 holds the current state

; States needed
.equ state_read = 1
.equ state_clear = 2
.equ state_store = 3
.equ state_OK = 4
.equ state_end = 5
```

**ΣΗΜΕΙΩΣΗ:** Για την μετάδοση χαρακτήρων, χρησιμοποιήθηκε η ρουτίνα USART\_transmit, όπως υπάρχει στο παράδειγμα στην σελίδα 151 του εγχειριδίου του ATmega16, με την διαφορά ότι χρησιμοποιείται ο καταχωρητής TCNT2 αντί του UDR, για ευκολία στην προσομοίωση της σειριακής θύρας καθώς το πρόγραμμα κρεμάει με την χρήση του UDR επειδή δεν υπάρχει κάποιος να διαβάσει τα δεδομένα μας.

#### Αποτελέσματα προσομοίωσης:

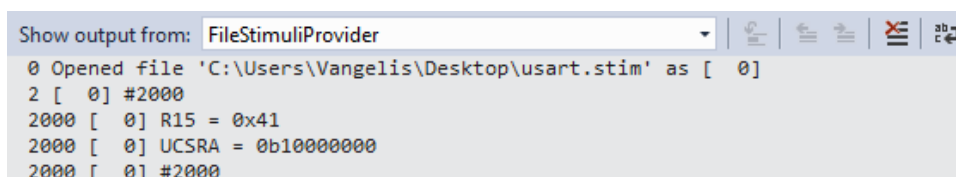
Για την προσομοίωση, χρησιμοποιήθηκε ένα αρχείο “.stim” από το οποίο εισάγουμε τους χαρακτήρες στον μικροελεγκτή, το οποίο έχει την εξής δομή:

```
# “cycles to wait”
UCSRA = 0b10000000 ; to enable RXC
R15 = “ASCII char we want to save”
```

**ΣΗΜΕΙΩΣΗ:** Τα αρχεία “.stim” βρίσκονται στον φάκελο του συνημμένου Atmel project.

#### Προσομοίωση AT<CR><LF>:

Αρχικά μόλις ξεκινήσει η το debugging και το πρόγραμμα είναι ακόμα σταματημένο, φορτώνουμε και εκτελούμε το επιθυμητό αρχείο “.stim”. Στην συνέχεια το πρόγραμμα μεταβαίνει στο break point που υπάρχει στο τέλος της ρουτίνας εξυπηρέτησης του RX interrupt. Παρατηρούμε ότι μέσω του stimfile έχει εισέρθει και διαβαστεί ο πρώτος χαρακτήρας που περιμένουμε, δηλαδή το “A”, όπως φαίνεται και παρακάτω.



```
Show output from: FileStimuliProvider
0 Opened file 'C:\Users\Vangelis\Desktop\usart.stim' as [ 0]
2 [ 0] #2000
2000 [ 0] R15 = 0x41
2000 [ 0] UCSRA = 0b10000000
2000 [ 0] #2000
```

Επιπλέον, παρατηρούμε ότι η επιθυμητή τιμή (A => 0x41) υπάρχει στον καταχωρητή r16.

R15	0x41
R16	0x41
R17	0x00

Όπως φαίνεται και παρακάτω, αντίστοιχα αποτελέσματα έχουμε και για τους χαρακτήρες T(0x54) και <CR> (0x0D).

- Μετά την ανάγνωση του χαρακτήρα "T":

```
Output
Show output from: FileStimuliProvider
0 Opened file 'C:\Users\Vangelis\Desktop\usart.stim' as [ 0]
2 [ 0] #2000
2000 [ 0] R15 = 0x41
2000 [ 0] UCSRA = 0b10000000
2000 [ 0] #2000
4001 [ 0] R15 = 0x54
4001 [ 0] UCSRA = 0b10000000
4001 [ 0] #2000
```

R15	0x54
R16	0x54
R17	0x00

- Μετά την ανάγνωση του χαρακτήρα "<CR>":

```
Output
Show output from: FileStimuliProvider
0 Opened file 'C:\Users\Vangelis\Desktop\usart.stim' as [ 0]
2 [ 0] #2000
2000 [ 0] R15 = 0x41
2000 [ 0] UCSRA = 0b10000000
2000 [ 0] #2000
4001 [ 0] R15 = 0x54
4001 [ 0] UCSRA = 0b10000000
4001 [ 0] #2000
6000 [ 0] R15 = 0x0D
6000 [ 0] UCSRA = 0b10000000
6000 [ 0] #2000
```

R15	0x0D
R16	0x0D
R17	0x00

Στην συνέχεια διαβάζουμε τον χαρακτήρα <LF>, όπως φαίνεται και παρακάτω:

```
Output
Show output from: FileStimuliProvider
0 Opened file 'C:\Users\Vangelis\Desktop\usart.stim' as [ 0]
2 [ 0] #2000
2000 [ 0] R15 = 0x41
2000 [ 0] UCSRA = 0b10000000
2000 [ 0] #2000
4001 [ 0] R15 = 0x54
4001 [ 0] UCSRA = 0b10000000
4001 [ 0] #2000
6000 [ 0] R15 = 0x0D
6000 [ 0] UCSRA = 0b10000000
6000 [ 0] #2000
8000 [ 0] R15 = 0x0A
8000 [ 0] UCSRA = 0b10000000
8000 [ 0] $log TCNT2
8000 [ 0] $startlog lab3.log
8000 [ 0] #10000
```

R15	0x0A
R16	0x0A
R17	0x00

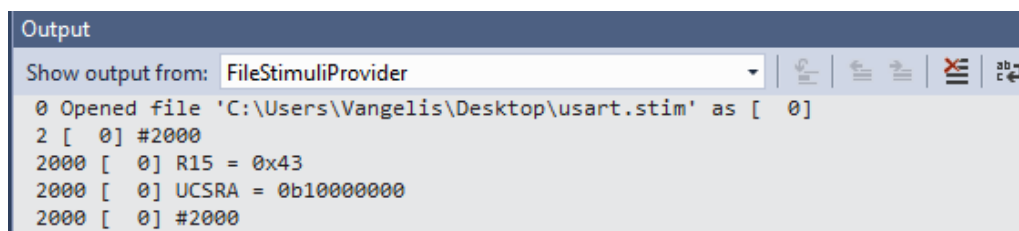
Παρατηρούμε ότι αφού διαβάσουμε τον χαρακτήρα <LF>, γίνεται η μετάδοση των χαρακτήρων “OK<CR><LF>”. Στο αρχείο “lab3.log” παρατηρούμε ότι έχουν γραφτεί οι χαρακτήρες “OK<CR><LF>”, επομένως η μετάδοση ήταν επιτυχημένη.

```
#39
TCNT2 = 0x4f
#11
TCNT2 = 0x4b
#11
TCNT2 = 0x0d
#11
TCNT2 = 0x0a
```

Στην συνέχεια το αρχείο “.stim” κλείνει και το κυρίως πρόγραμμα συνεχίζει να εκτελείται.

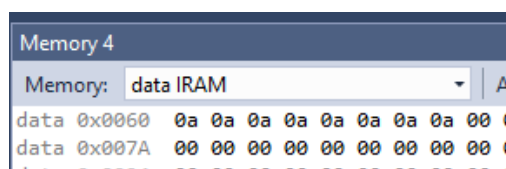
Προσομοίωση C<CR><LF>:

Αφού ξεκίνησε η εκτέλεση του προγράμματος με το αρχείο “.stim”, παρατηρούμε αρχικά ότι διαβάζουμε τον χαρακτήρα “C”(0x43).



R15	0x43
R16	0x43
R17	0x00

Αφού διαβάσουμε τον χαρακτήρα “C” παρατηρούμε ότι στις δεσμευμένες θέσεις μνήμης υπάρχει ο χαρακτήρας “0x0A” ο οποίος αντιστοιχεί στην τιμή “0xFF” στο 7 segment και άρα και σε σβηστό segment.



Στην συνέχεια διαβάζουμε τους χαρακτήρες <CR><LF> και παρατηρούμε πως προκύπτουν αντίστοιχα αποτελέσματα με την προηγούμενη εντολή. Έπειτα, παρατηρούμε ότι στα segments της οθόνης υπάρχει η τιμή “0xFF” επομένως η οθόνη είναι σβηστή.

[illegible]

Αντίστοιχα συνεχίζοντας την προσομοίωση παρατηρούμε την τιμή 0xFF και στα υπόλοιπα ψηφία.

#### Προσομοίωση N<X><CR><LF>:

Στην παρών προσομοίωση λαμβάνουμε τους χαρακτήρες “N123<CR><LF>”. Αφού ξεκίνησε η εκτέλεση του προγράμματος με το αρχείο “.stim”, παρατηρούμε αρχικά ότι διαβάζουμε τον χαρακτήρα “N”(0x4E).

```
Output
Show output from: FileStimuliProvider
0 Opened file 'C:\Users\Vangelis\Desktop\usart.stim' as [ 0]
2 [ 0] #2000
2000 [ 0] R15 = 0x4E
2000 [ 0] UCSRA = 0b10000000
2000 [ 0] #2000
```

Αφού διαβάσουμε τον χαρακτήρα “C” παρατηρούμε ότι στις δεσμευμένες θέσεις μνήμης υπάρχει ο χαρακτήρας “0x0A” ο οποίος αντιστοιχεί σε σβηστή οθόνη.

```
Memory 4
Memory: data IRAM
data 0x0060 0a 0a 0a 0a 0a 0a 0a 0a 00
data 0x007A 00 00 00 00 00 00 00 00 00
```

Στην συνέχεια διαβάζουμε τον χαρακτήρα που αντιστοιχεί στο “1” (δηλ. την τιμή 0x31). Όπως φαίνεται και παρακάτω.

```
Output
Show output from: FileStimuliProvider
2 [ 0] #2000
2000 [ 0] R15 = 0x4E
2000 [ 0] UCSRA = 0b10000000
2000 [ 0] #2000
4001 [ 0] R15 = 0x31
4001 [ 0] UCSRA = 0b10000000
4001 [ 0] #2000
```

Στην συνέχεια από την τιμή “0x31” που διαβάσαμε, παίρνουμε τα πρώτα τέσσερα bit και τα αποθηκεύουμε στην μνήμη. Επομένως, όπως βλέπουμε και στην παρακάτω εικόνα, στην μνήμη θα αποθηκευτεί η τιμή “0x01”.

```
Memory 4
Memory: data IRAM
data 0x0060 01 0a 0a 0a 0a 0a 0a 0a 00
data 0x0077 00 00 00 00 00 00 00 00 00
data 0x0085 00 00 00 00 00 00 00 00 00
```

Στην συνέχεια διαβάζουμε τον χαρακτήρα που αντιστοιχεί στο “2” (δηλ. την τιμή 0x32). Όπως φαίνεται και παρακάτω

```
4001 [ 0] R15 = 0x31
4001 [ 0] UCSRA = 0b10000000
4001 [ 0] #2000
6001 [ 0] R15 = 0x32
6001 [ 0] UCSRA = 0b10000000
6001 [ 0] #2000
```

