**Εργαστήριο 6**

**22/11/2020**

**WARM START, COLD START, ΚΑΙ Ο WATCHDOG TIMER**

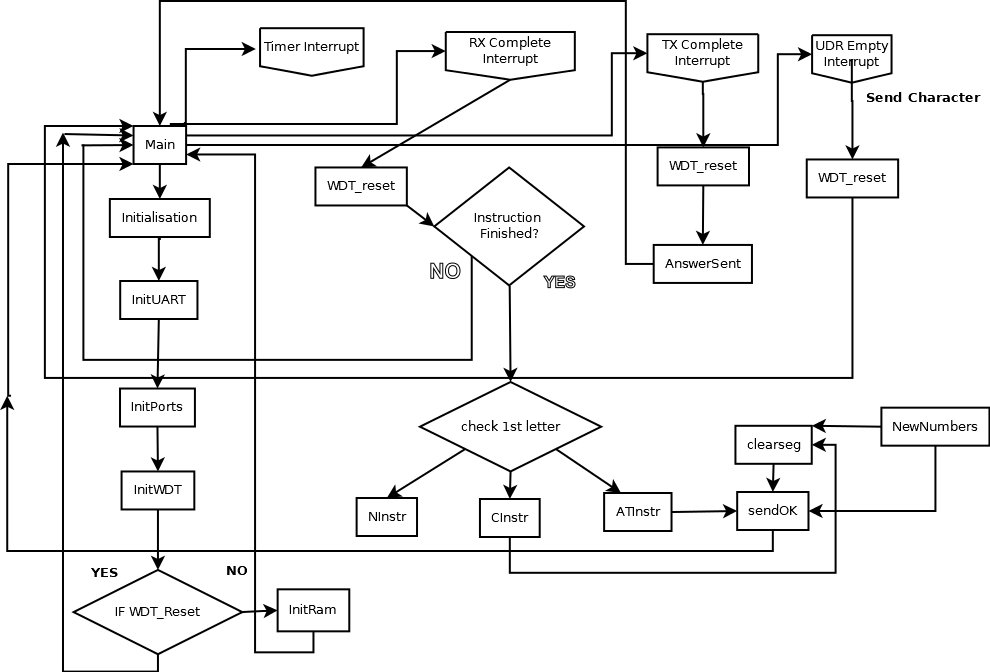
Ντουνέτας Δημήτρης

ΑΜ: 2016030141

# Εισαγωγή

Σκοπός του εργαστηρίου είναι η εξοικείωση με τον Watchdog timer καθώς και τη σωστή χρήση του σε Μικροελεγκτή AVR. Επιπλέον η εξοικείωση με την έννοια του Cold και Warm Start, όπως και του Reset σε περίπτωση Watchdog timer Reset. Τέλος, παρατήρηση της assembly εξόδου από τον C Compiler και πως κάνει χρήση των πόρων του συστήματος. Δημιουργία προγράμματος για επικοινωνία με χρήση USART του ATMEGA16, για παραλαβή εντολών και αποστολή απαντήσεων σε συνδυασμό με οδήγηση με πολυπλεξία στον χρόνο μίας οθόνης 7-segment LED για (έως) οκτώ ψηφία.

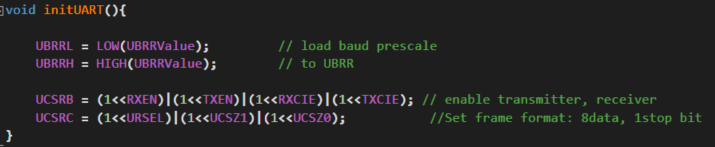
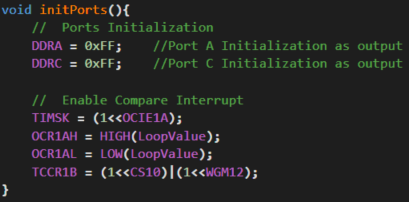
# Block Diagram Προγράμματος



# Αρχικοποίηση του προγράμματος σε Γλώσσα C

Αρχικά το πρόγραμμα ξεκινάει από την συνάρτηση main που βρίσκεται στο αρχείο main.c . Η main συνάρτηση είναι υπεύθυνη για την αρχικοποίηση του προγράμματος και της μνήμης καθώς και τον ατέρμονα βρόγχο πάνω στον οποίο τρέχει το πρόγραμμα. Για γίνει η αρχικοποίηση η Main καλεί τη συνάρτηση initialization. O Stack Pointer δεν χρειάζεται να αρχικοποιηθεί αφού τον αρχικοποιεί ο C Compiler από μόνος του ώστε να μπορούμε να επιστρέφουμε σωστά από τις ρουτίνες και τις συναρτήσεις που καλούμε.

Η Initialization στην συνέχεια καλεί τις initUΑRT, initRam, initPorts και InitWDT συνάρτήσεις οι οποίες κάνουν τις αρχικοποιήσεις μνήμης , καταχωρητών, θέσεων μνήμης και Ports όπως αναφέρθηκαν στο προηγούμενο εργαστήριο. Συνοπτικά χρησιμοποιούνται 4 σημεία στην μνήμη SRAM στα οποία το ένα με όνομα display\_address είναι το σημείο στο οποίο γράφονται τα BCD στοιχεία που θα δείξει η οθόνη. Το δεύτερο είναι το UARTAnsAddress όπου γράφεται η αυτοματοποιημένη απάντηση που δίνει ο Μικροελεγκτής μας όταν λάβει ένα ολόκληρο Instruction σωστά δηλαδή OK<CR><LF>. Tο τρίτο, είναι το InstrAddress όπου αποθηκεύεται το instruction που λαμβάνουμε ώστε να μπορούμε να το διαχειριστούμε κατάλληλα όταν θέλουμε να εκτελέσουμε την εντολή. Τέλος το τέταρτο είναι το DecAddress όπου γίνεται το Decode από BCD σε 7-segment αριθμό ώστε να αποδίδεται σωστά στην οθόνη.



# 

# RAM MAP

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0x80 | 0x81 | 0x82 | 0x83 | 0x84 | 0x85 | 0x86 | 0x87 |
| 7segbit7(LSB) | 7segbit6 | 7segbit5 | 7segbit4 | 7segbit3 | 7segbit2 | 7segbit1 | 7segbit0(MSB) |

Η λέξη που βρίσκεται στη θέση 0x80 είναι το δεξιότερο Digit που θα εμφανιστεί στο

7seg-Screen.

|  |  |  |  |
| --- | --- | --- | --- |
| 0x90 | 0x91 | 0x92 | 0x93 |
| UARTAnsByte7(O) | UARTAnsByte6(K) | UARTAnsByte5(CR) | UARTAns4(LF) |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0xΑ0 | 0xΑ1 | 0xΑ2 | 0xΑ3 | 0xΑ4 | 0xΑ5 | 0xΑ6 | 0xΑ7 | 0xΑ8 |
| instrByte0 | instrByte1 | instrByte2 | instrByte3 | instrByte4 | instrByte5 | instrByte6 | instrByte7 | instrByte8 |

|  |  |  |
| --- | --- | --- |
| 0xΑ9 | 0xΑA | 0xΑB |
| instrByte9 | instrByte10 | instrByte11 |

Στις θέσεις ορισμένες ως instrByte εισάγεται διαδοχικά η λέξη της κάθε εντολής που λαμβάνουμε.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0xΒ0 | 0xΒ1 | 0xΒ2 | 0xΒ3 | 0xΒ4 | 0xΒ5 | 0xΒ6 | 0xΒ7 | 0xΒ8 |
| DecByte0 | DecByte1 | DecByte2 | DecByte3 | DecByte4 | DecByte5 | DecByte6 | DecByte7 | DecByte8 |

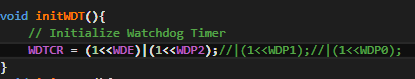
|  |  |  |
| --- | --- | --- |
| 0xΒ9 | 0xΒA | 0xΒB |
| DecByte9 | DecByte10 | DecByte11 |

|  |  |  |
| --- | --- | --- |
| 0xC0 | 0xC1 | 0xC2 |
| ResAddress0(R) | RecAddress1(CR) | RecAddress0(LF) |

|  |  |  |  |
| --- | --- | --- | --- |
| 0xC3 | 0x100 | 0x101 | 0x102 |
| ResVar(is\_WDR\_reset) | RingCounterAddress | DigitRegAddress | DigitCounterAddress |

# WDR Initialisation

Στην συνάρτηση Initialization υπάρχει η συνάρτηση initWDT η οποία αρχικοποιεί τον Watchdog timer ώστε να είναι ενεργός και να μετράει. Για να γίνει αυτό αρκεί να ενεργοποιήσουμε στον WDTCR καταχωρητή τα Bits WDE και WDP2,WDP1,WDP0 ανάλογα τι αριθμό κύκλων θέλουμε να μετράει ο Watchdog Timer. Όπως μπορούμε να δούμε και στο Datasheet, ο Watchdog Timer έχει δικό του ξεχωριστό ρολόι το οποίο τρέχει στο 1MHz.

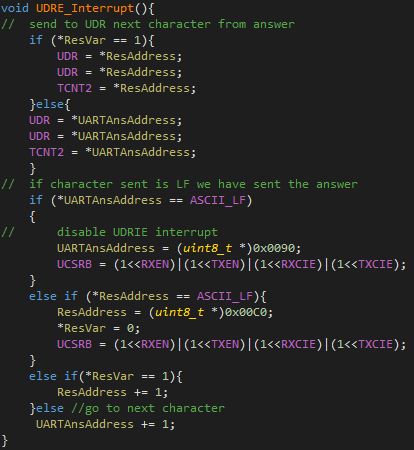
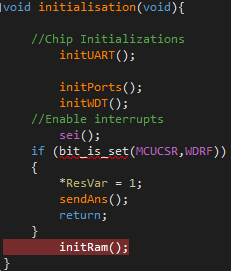


# Λειτουργία του Watchdog Timer

Ο βασικός λόγος που θέλουμε να κάνουμε χρήση του Watchdog είναι για να αποτρέψουμε το κλείδωμα του μικροελεγκτή σε κάποιο λάθος state ή ατέρμονα βρόγχο που δεν επιθυμούμε. Γι αυτό, όταν εκτελούμε διαδικασίες που είναι γνωστό ότι θέλουμε να τις εκτελέσουμε μπορούμε να κάνουμε Reset των Watchdog timer και έτσι δε θα γίνει Reset στον Μικροελεγκτή μας και θα συνεχίσει κανονικά τη ροή του προγράμματος. Όταν ο μετρητής μετρήσει μέχρι την τιμή που του έχουμε ορίσει μέσω του Prescaler του, τότε ενεργοποιεί ένα Watchdog Timer Reset και ο Μικροελεγκτής ξεκινάει εξαρχής την εκτέλεση και τη ροή του από την αρχή του προγράμματος του στην main. Στο υπάρχον πρόγραμμα ο Μετρητής γίνεται Reset κάθε φορά που μπαίνουμε σε ένα από τα 3 Interrupts του USART που έχουμε χρησιμοποιήσει. Δηλαδή κατά το διάβασμα ενός χαρακτήρα, κατά την αποστολή του ή κατά την ολοκλήρωση μια εντολής.

# Υλοποίηση Warm-Start και απάντησης R CR LF

Για να καταφέρουμε να υλοποιήσουμε ένα κάπως σωστό warm start στο Atmel Studio και να στέλνουμε στον USART πίσω απάντηση όταν έχει εκτελεστεί Watchdog Timer Reset τότε χρειάζεται να γνωρίζουμε στην αρχή του Initialisation αν βρισκόμαστε σε Power\_on Reset ή σε Watchdog Timer Reset. Την πληροφορία αυτή μας την παρέχει ο καταχωρητής MCUCSR και συγκεκριμένα το WDRF bit που στη ουσία είναι ένα flag που ενημερώνει αν ο Watchdog Timer ευθύνεται για το Reset που προηγήθηκε. Ελέγχοντας αυτό το bit μπορούμε να αποφασίσουμε αν θα χρειαστεί να κάνουμε Initialize ξανά την Ram ή αν θα την αφήσουμε όπως έχει. Θέλουμε μετά από ένα Reset του Watchdog στην δική μας περίπτωση να μην αρχικοποιείται ξανά η μνήμη καθώς δε θέλουμε να χάσουμε τα δεδομένα που έχουμε αποθηκεύσει πρότινος. Παρόλα, αυτά θέλουμε να αρχικοποιήσουμε ξανά τα υπόλοιπα κομμάτια του Μικροελεγκτή όπως τα Ports το Usart κτλ, επειδή χάνουν τις τιμές που τους είχαμε δώσει μετά το Reset. Τέλος δημιουργούμε μια καινούργια μεταβλητή στην οποία γράφουμε την τιμή 1 όταν το Reset μας οφείλεται σε Watchdog και μέσω αυτής , μπαίνουμε στο UDR interrupt και στέλνουμε το μήνυμα R CR LF. Τέλος αποθηκεύουμε το μήνυμα R CR LF σε ξεχωριστό σημείο της SRAM από το O K CR LF.



# Πιθανές αιτίες λανθασμένης λειτουργία που οφείλονται στον Watchdog Timer και ελέγχθηκαν.

Πολλές αιτίες υπάρχουν που μπορούν να οδηγήσουν σε λανθασμένη ή απρόβλεπτη συμπεριφορά όσο αναφορά την υλοποίηση που έχει δημιουργηθεί μέχρι στιγμής. Παρόλο που ο Watchdog timer γίνεται Reset πριν από κάθε φορά που τρέχει ένα Interrupt του USART σε μια προσπάθεια περιορισμού των σφαλμάτων υπάρχουν ακόμη περιπτώσεις που μπορεί να δημιουργηθεί πρόβλημα. Μια από αυτές είναι να γίνει WDT\_RESET(WatchDog Timer Reset) κατά την λήψη ή αποστολή ενός χαρακτήρα. Κάτι τέτοιο με τη υλοποίηση που υπάρχει είναι πιθανό κυρίως πριν το διάβασμα του πρώτου χαρακτήρα καθώς μπορεί να έρθει οποτεδήποτε και δεν μπορούμε να το προβλέψουμε κάπως. Αντίθετα στην αποστολή γνωρίζουμε περίπου πότε θα γίνει αφού την κάνουμε εμείς και έτσι χοντρικά ξέρουμε πόσο χρόνο χρειαζόμαστε. Έτσι είναι σημαντικό να αναφερθεί ότι μπορεί να γίνει μια μελέτη σχετικά με ποια είναι η κατάλληλη τιμή ώστε να έχουμε όσο το δυνατόν μικρότερη χρονική καθυστέρηση για τον εντοπισμό ενός λάθους στο σύστημα μας αλλά και να μην δημιουργεί πρόβλημα στις διαδικασίες που εκτελούμε. Στην περίπτωση μας δίνουμε 256 χιλιάδες κύκλους ώστε να μετρήσει ο μετρητής μέχρι να τρέξει ένα Reset.

Ένα άλλο λάθος που θα μπορούσαμε να έχουμε είναι να δώσουμε πολύ μικρή τιμή στον Watchdog Timer και να δημιουργείται πρόβλημα επειδή καθώς προσπαθεί ο Μικροελεγκτής να κάνει τη δουλεία του κανονικά, έρχεται WDT\_RESET και ξεκινάει ξανά από την αρχή.

# Παραδείγματα Αποτελεσμάτων

Μια σωστή υλοποίηση μας δίνει ένα αποτέλεσμα όπως παρακλατω:

Αρχικά στέλνει την απάντηση O K CR LF αφού εκτελέσει την εντολή που δέχτηκε.

Δεν στέλνει R CR LF αφού είναι Power On Reset και όχι WDT\_RESET.

#220522

TCNT2 = 0x4f

#10508

TCNT2 = 0x4b

#10400

TCNT2 = 0x0d

#10400

TCNT2 = 0x0a

#282910

Ύστερα γίνεται Reset και το βλέπουμε από το μηδενισμό του TCNT2 καταχωρητή.

TCNT2 = 0x00

Στέλνει το R CR LF αφού υλοποιήθηκε WDT\_RESET

#318

TCNT2 = 0x52

#10486

TCNT2 = 0x0d

#10400

TCNT2 = 0x0a

Μια λάθος υλοποίηση μας δίνει ένα τέτοιο αποτέλεσμα:

Ξεκινά η ανάγνωση χαρακτήρων

#50721

TCNT2 = 0x4f

Διακόπτεται από WDR\_RESET και χάνεται το διάβασμα του ερωτήματος προς τον AVR.

#318

TCNT2 = 0x00

Αποστέλλεται η απάντηση R CR LF

#10530

TCNT2 = 0x52

#10486

TCNT2 = 0x0d

#10400

TCNT2 = 0x0a

Προχωράει σε επόμενη απάντηση άλλου ερωτήματος χωρίς να έχει εξυπηρετήσει το πρώτο.

#188471

TCNT2 = 0x4f

#10494

TCNT2 = 0x4b

#10400

TCNT2 = 0x0d

#10400

TCNT2 = 0x0a

#282910

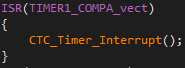
TCNT2 = 0x00

## **Λειτουργία του κύριου προγράμματος και τα Ιnterrupt του USART(UART).**

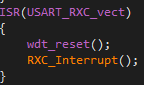
Καθώς το πρόγραμμα έχει ξεκινήσει και βρίσκεται στον ατέρμονα βρόγχο καλούνται τα κατάλληλα interrupts και αυτά χρησιμοποιούμε για να δώσουμε λειτουργικότητα στο πρόγραμμα μας. Αυτή τη στιγμή υπάρχουν 4 Interrupts τα οποία αξιοποιεί το πρόγραμμα για να δείξει στην οθόνη ότι εμείς του δίνουμε ως είσοδο μέσω του UART. Τα Interrupts αυτά είναι:

1. Timer/Counter Compare1 Interrupt
2. USART RX Complete Interrupt
3. USART DATA Registry Empty Interrupt
4. USART TX Complete Interrupt

* To Timer/Counter 1 Compare A Interrupt είναι υπεύθυνο για το σωστό χρονισμό της οθόνης.

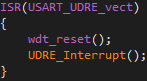


* Καλεί την συνάρτηση CTC\_TIMER\_Interrupt η οποία μας δίνει το κατάλληλο χρονικό περιθώριο για να ανανεώνουμε την 7segment οθόνη για να δώσουμε τα FPS που θέλουμε.
* Το USART RX Complete Interrupt είναι υπεύθυνο για το σωστό διάβασμα όταν σηκωθεί το Flag RXC το οποίο υποδηλώνει ότι έχουμε λάβει κάποιον χαρακτήρα έτοιμο να διαβάσουμε.



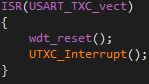
Καλεί την συνάρτηση RXC \_Interrupt η οποία λαμβάνει ένα χαρακτήρα και τον αποθηκεύει. Επίσης ελέγχει αν έχει ολοκληρωθεί η λήψη της εντολής, καθώς και ποια εντολή έχει ληφθεί.

* Το USART DATA Registry Empty Interrupt είναι υπεύθυνο να ενημερώνει όταν το UDR καταχωρητής είναι διαθέσιμος για εγγραφή ώστε να μην απανογράφουμε χωρίς να έχουν αποσταλεί πρώτα τα δεδομένα που θέλουμε.



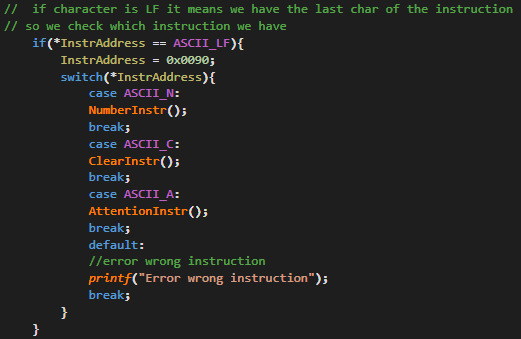
Καλεί την συνάρτηση UDRE\_Interrupt η οποία στέλνει έναν χαρακτήρα στο UDR ώστε αυτός να αποσταλεί ορθά από τον USART.

* To USART TX Complete Interrupt είναι υπεύθυνο για την ενημέρωση της αποστολής ενός χαρακτήρα. Μας ενημερώνει ότι η αποστολή έχει ολοκληρωθεί και ο χαρακτήρας έχει φτάσει στον προορισμό του. Η διαφορά με το DATA Registry Empty Interrupt είναι ότι περιμένει μέχρι το Byte να φύγει εντελώς από τον Shift Register που αποστέλλει σειριακά και έτσι μας είναι χρήσιμο μόνο σε HALF-Duplex πρωτόκολλα ή όταν θέλουμε να κάνουμε ενέργεια μετά το πέρας της αποστολής. Έτσι το χρησιμοποιούμε για να οριστικοποιήσουμε την αποστολή όλης της απάντησης του Μικροελεγκτή μας.



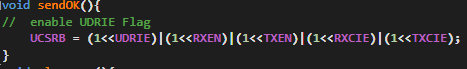
# Διαχείριση εντολών

Για την διαχείριση των εντολών που λαμβάνουμε μέσω του UART υπάρχουν ξεχωριστές συναρτήσεις που ενεργούν αφού λάβουμε σωστά ολόκληρη την εντολή και ουσιαστικά μας έρθει ο χαρακτήρας CR ακολουθούμενος από τον LF. Ελέγχουμε τον πρώτο χαρακτήρα αφού θεωρούμε σωστές τις εντολές και εκτελούμε κατάλληλα τη συνάρτηση που πρέπει για κάθε εντολή.



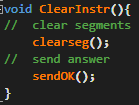
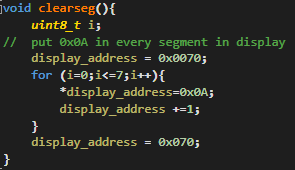
# ΑΤΤΕΝΤΙΟΝ Εντολή

Όταν λάβουμε την εντολή AT<CR><LF> αρκεί να στείλουμε πίσω μια απάντηση ΟΚ. Αυτό εκτελεί η SendOK στην οποία ενεργοποιούμε το Interrupt για να στείλουμε την απάντηση.



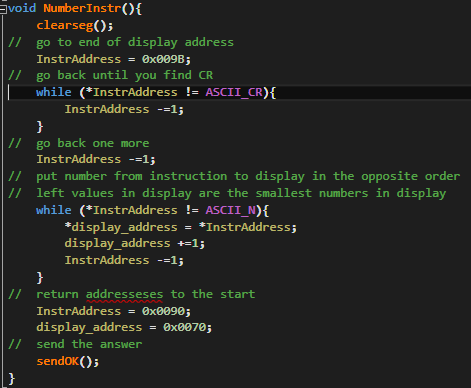
# CLEAR Εντολή

Όταν λάβουμε την εντολή C<CR><LF> αρκεί να καθαρίσουμε την οθόνη ώστε να μη δείχνει τίποτα και να στείλουμε πίσω την απάντηση OK. Έτσι καλούμε την ρουτίνα clearseg και μετά την SendOK.



# NUMBER Εντολή

Όταν λάβουμε την εντολή Ν239…<CR><LF> αρκεί να καθαρίσουμε την οθόνη ώστε να μη δείχνει τίποτα ύστερα να βάλουμε τα σωστά ψηφία στην οθόνη και να στείλουμε πίσω την απάντηση OK. Καθαρίζουμε , περνάμε τα δεδομένα στην display\_Address και στέλνουμε την απάντηση όπως προηγούμενα.



Aυτή η ρουτίνα διαβάζει την αποθηκευμένη εντολή από το τέλος και αφού βρει που ξεκινάνε τα νούμερα που έχουμε ως ορίσματα τα αποθηκεύει 1-1 κατάλληλα στις θέσεις μνήμης ώστε να τα δείχνει σωστά η οθόνη 7seg. Δηλαδή το δεξιότερο νούμερο το αποθηκεύει στο δεξιότερο 7seg και τα υπόλοιπα αντίστοιχα.

# Τελικό Αποτέλεσμα

Το πρόγραμμα τρέχει συνεχόμενα χωρίς να σταματάει αφού ως βασικό μέρος έχει έναν ατέρμονα βρόγχο. Διαβάζει εντολές και ενεργεί κατάλληλα καθώς επίσης απαντάει και OK αφού διαβάσει μια σωστή εντολή.

# Παρατήρηση

Επειδή το Atmel Studio δεν έχει κατάλληλο περιβάλλον για αποσφαλμάτωση USART επικοινωνίας έχουν γίνει κάποιες αλλαγές στο πρόγραμμα ώστε να είναι δυνατή η προσομοίωση του με STIMFILE και να logάρονται κατάλληλα οι απαντήσεις. Η εισαγωγή των bytes που λαμβάνονται γίνεται με τον Καταχωρητή 16 και έτσι τα διαβάζουμε ως είσοδο στο πρόγραμμά μας. Επίσης οι απαντήσεις δίνονται στον Καταχωρητή TCNT2 αυθαίρετα. Θα μπορούσαμε να χρησιμοποιήσουμε οποιουσδήποτε καταχωρητές δεν χρησιμοποιούμε απλά το Stim File δεν μπορεί να διαβάσει τον UDR ούτε τα PINS RX και TX.

Στην διαδικασία αποσφαλμάτωσης παρατηρήθηκε επίσης ότι κατά τη διάρκεια αποστολής συμβόλου που χρησιμοποιούμε το Flag UDRE για την ενεργοποίηση του Interrupt αυτό το Flag όπως και το RXC χρειάζεται να το προσπελάσουμε και να το γράψουμε 2 φορές και να ανταποκριθεί το USART καθώς και ότι αφού σταλούν δεδομένα χρειάζεται περίπου 180.000 κύκλους για να σηκωθεί το επόμενο Flag και να καλεστεί το interrupt. Αυτό σύμφωνα με τους υπολογισμούς που λαμβάνουμε υπόψιν το Baud Rate θα έπρεπε να συμβαίνει κάθε 10.000 κύκλους χοντρικά. Πιθανόν να είναι σφάλμα που οφείλεται στο Atmel Studio είτε στην τρόπο που θεωρεί το Stim File ρολόι επειδή αυτή η διαφορά μιας τάξης μεγέθους θα μπορούσε να δικαιολογηθεί αν το ρολόι είχε τιμή κοντά στο 1 MHz.