Ενσωματωμένα Συστήματα Μικροεπεξεργαστών – ΗΡΥ411

Αναφορά Εργαστηρίου 4 – Κώδικας C και Assembly στον Atmel AVR

Ομάδα: LAB41145877
Κιούλος Ευάγγελος 2016030056

Εισαγωγή:

Σκοπός της τέταρτης εργαστηριακής άσκησης, ήταν η εξοικείωση με την ταυτόχρονη χρήση της γλώσσας C και της Assembly στον μικροελεγκτή Atmel AVR. Η λειτουργεία του νέου προγράμματος είναι η ίδια με το πρόγραμμα του εργαστηρίου 3. Παρόλα αυτά, στο παρών εργαστήριο ο κώδικας των αρχικοποιήσεων και του κύριου προγράμματος θα είναι σε γλώσσα C ενώ οι ρουτίνες εξυπηρέτησης των interrupt σε Assembly. Επιπλέων, η ρουτίνα "USART_Transmit" που χρησιμοποιήθηκε για μετάδοση χαρακτήρων και υλοποιήθηκε στο προηγούμενο εργαστήριο με polling, στο παρών εργαστήριο υλοποιήθηκε σε C.

Υλοποίηση Κώδικα:

Αρχικά, δημιουργήθηκε ένα αρχείο ".c" το οποίο περιέχει τον κώδικα σε γλώσσα C και ένα αρχείο ".S" που περιέχει τον κώδικα σε Assembly.

Στο αρχείο ".c", στην αρχή του προγράμματος, με "#include" ορίστηκαν οι βιβλιοθήκες "avr/io.h" για την επεξεργασία i/o του μικροελεγκτή και "avr/interrupt.h" για τη χρήση των interrupt vectors και με "#define" οι σταθερές FOSC που ορίζει την ταχύτητα του ρολογιού, BAUD που ορίζει το baud rate και MYUBRR που ορίζει την τιμή του καταχωρητή UBRR, όπως φαίνεται και στην παρακάτω εικόνα.

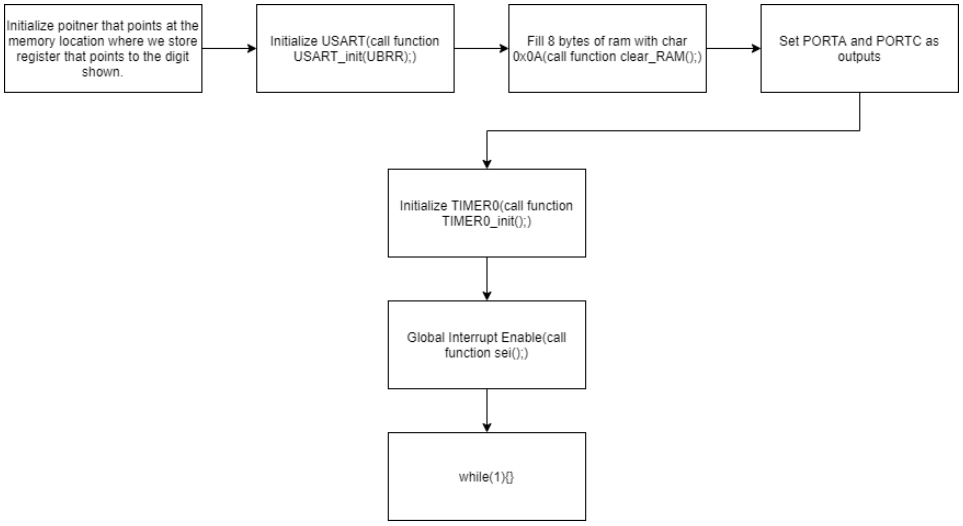
```
#include <avr/io.h>
#include <avr/interrupt.h>

#define FOSC 10000000   // clock speed at 10MHz
#define BAUD 9600       // baud rate
#define MYUBRR FOSC/16/BAUD-1   // register ubrr value
```

Στην συνέχεια δηλώθηκαν οι συναρτήσεις που χρειάστηκαν για την υλοποίηση του προγράμματος. Για να μπορέσουμε να καλέσουμε τις ρουτίνες από την Assembly, χρειάστηκε να δηλωθούν οι αντίστοιχες συναρτήσεις με όνομα ίδιο με το label που χρησιμοποιήθηκε στην Assembly και τύπο "extern void", όπως βλέπουμε και παρακάτω.

```
/* Assembly routines */
extern void ISR_timer(void);
extern void ISR_uart(void);
```

Στην συνάρτηση main(), αρχικά ορίζεται ένας pointer ο οποίος δείχνει στην θέση "0x68" της RAM όπου αποθηκεύουμε το ποιο ψηφίο δείχνουμε στην οθόνη και στην συνέχεια καλείται η συνάρτηση "USART_init(MYUBRR)" η οποία δέχεται ως όρισμα την τιμή που θα τεθεί ο καταχωρητής UBRR και αρχικοποιεί τους απαραίτητους καταχωρητές για την χρήση της σειριακής θύρας σε Asynchronous mode, χωρίς parity bit, με ένα stop bit και 8 data bits, με τον transmitter, τον receiver και τα RXC interrupts ενεργοποιημένα. Στην συνέχεια, καλείται η συνάρτηση "clear_RAM()" η οποία εισάγει τον χαρακτήρα "0x0A" στις οχτώ δεσμευμένες θέσεις μνήμης που έχουμε για την αποθήκευση των χαρακτήρων που προβάλλονται και

ορίζονται τα PORTA και PORTC ως έξοδοι. Έπειτα, καλείται η συνάρτηση “TIMER0_init()” με την οποία αρχικοποιείται ο TIMER/COUNTER0 και ο prescaler και ενεργοποιούνται τα TIMER0 Overflow interrupt και μετά καλείται η συνάρτηση “sei()” για την ενεργοποίηση των global interrupt. Τέλος, το πρόγραμμα μεταβαίνει σε έναν ατέρμονο βρόγχο. Στο παρακάτω διάγραμμα περιγράφεται η ροή του κύριου προγράμματος.



Για τις ρουτίνες εξυπηρέτησης των interrupt, χρησιμοποιήθηκε η συνάρτηση “ISR()” η οποία ως όρισμα παίρνει την διεύθυνση του αντίστοιχου interrupt vector και καλεί την αντίστοιχη ρουτίνα εξυπηρέτησης του interrupt από την Assembly. Παρακάτω φαίνεται η χρήση της συνάρτησης ISR().

```
/*
 * Interrupt Service Routine for TIMER0 Overflow interrupt.
 * Calls TIMER0 Overflow assembly routine
 * TIMER0_OVF_vect is the interrupt vector for TIMER0 Overflow interrupt.
 */
ISR(TIMER0_OVF_vect){
    ISR_timer();
}

/*
 * Interrupt Service Routine for RXC interrupt.
 * Calls USART RXC interrupt assembly routine
 * USART_RXC_vect is the interrupt vector for RXC interrupt.
 */
ISR(USART_RXC_vect){
    ISR_uart();
}
```

Το όρισμα “TIMER0_OVF_vect” αντιστοιχεί στο Timer0 Overflow interrupt vector ενώ το όρισμα “USART_RXC_vect” αντιστοιχεί στο RXC interrupt vector.

Στο αρχείο “.S”, υλοποιήθηκαν οι ρουτίνες εξυπηρέτησης των interrupt, με την ίδια λογική των προηγούμενων εργαστήριων, αλλά με μικρές αλλαγές. Στην συνάρτηση ISR_uart, για την μετάδοση των χαρακτήρων “OK<CR><LF>” καλείται η συνάρτηση “send_OK(void)” η οποία υλοποιείται στη C. Επιπλέων, χρησιμοποιήθηκε το macro “_SFR_IO_ADDR()” όπου χρησιμοποιήθηκαν οι εντολές “in Rd, P” και “out P, Rd”, για να δώσουμε στον compiler την σωστή διεύθυνση του καταχωρητή “P” (π.χ. out _SFR_IO_ADDR(PORTA), r16). Επίσης, στην

θέση των εντολών high()/low() χρησιμοποιήθηκαν οι εντολές hi8()/lo8() αντίστοιχα. Τέλος, στην αρχή του κώδικα, με “#include” δηλώθηκε η βιβλιοθήκη “avr/io.h” της C και με “.global label”, τα labels των ρουτίνων σε Assembly για να μπορούν να κληθούν από τον κώδικα της C .

Χάρτης Μνήμης:

Στο πρόγραμμα του τέταρτου εργαστηρίου, χρησιμοποιούνται οι θέσεις “0x0060” έως “0x0067” της μνήμης RAM για να αποθηκεύσουμε τους χαρακτήρες που προβάλλονται στην οθόνη. Στην θέση “0x0068” αποθηκεύεται η τιμή που μας δείχνει το ψηφίο που εμφανίζουμε κάθε στιγμή στην οθόνη. Τέλος, το stack υπάρχει στο τέλος της μνήμης RAM στην διεύθυνση “0x045F”. Παρακάτω βλέπουμε τον χάρτη της μνήμης RAM.

0x0060
0x0061
0x0062
0x0063
0x0064
0x0065
0x0066
0x0067
0x0068
. . .
0x045F

Στις θέσεις με “μπλε” χρώμα αντιστοιχούν οι θέσεις που χρησιμοποιούνται για την αποθήκευση των χαρακτήρων που προβάλλονται στην οθόνη. Στην θέση με “πορτοκαλί” χρώμα, αντιστοιχεί η θέση που αποθηκεύεται το ψηφίο που προβάλουμε. Η θέση με “γκρι” χρώμα αντιστοιχεί στην θέση του stack.

Επιπλέων, στο πρόγραμμα χρησιμοποιείται ο καταχωρητής “r17” μόνο για την υλοποίηση του ring counter. Τέλος, η αποκωδικοποίηση από BCD ψηφία σε 7-segment γίνεται μέσω κώδικα.

Αποτελέσματα προσομοίωσης:

Για την προσομοίωση του προγράμματος, χρησιμοποιήθηκαν τα αρχεία “.stim” από το τρίτο εργαστήριο. Παρατηρήθηκε ότι το αποτέλεσμα της προσομοίωσης ήταν το ίδιο με το αποτέλεσμα του τρίτου εργαστηρίου. Επιπλέων, με την επιλογή “Disassembly”, παρατηρούμε ότι ο κώδικας Assembly που παράγει ο compiler έχει πολλές ομοιότητες με τον κώδικα assembly του τρίτου εργαστηρίου. Τα αρχεία “.stim” που χρησιμοποιήθηκαν περιλαμβάνονται στον φάκελο με το παραδοτέο project του Atmel Studio 7.