Εργαστήριο 5

Ονομα: Μαυρογιώργης Δημήτρης
ΑΜ: 2016030016
ΗΡΥ 411 - Ενσωματωμένα Συστήματα Μικροεπεξεργατών
Πολυτεχνειο Κρητης

April 19, 2021

Σκοπός

Σκοπός του τέταρτου εργαστηρίου είναι να μετατρεψουμε το προηγούμενο εργαστήριο εξ' ολοκλήρου σε C, αλλά έχοντας υπόψη το τι ποροι χρησιμοποιούνται στον μικροελεγκτή AVR. Στόχος μας είναι δηλαδή να υλοποιήσουμε σε κώδικα C και τις συναρτήσεις για τα Interrupts των TIMER0 και USART, καθώς και όλων των βοηθητικών συναρτήσεων για clear της μνήμης και shift των αριθμών που αποθηκεύουμε σε ένα array κατά το receive.

Περιγραφή της υλοποίησης

Αρχικά, όσον αφορά το κωμάτι της μνήμης, οπως και στα προηγούμενα εργαστήρια με τη συνάρτηση MEM_CLEAR αρχικοποιούμε τις θέσεις μνήμης που είναι αποθηκευμένοι οι αριθμοί με την τιμή "0x0A". Για το MEM_SHIFT αυτό που κάνουμε είναι ένα shift προς τα δεξιά όλων των στοιχείων του array.

Παράλληλα, για τη ληψη και αποστολή μηνυμάτων δημιουργήθηκαν δύο συναρτήσεις, η USART_RECEIVE και η USART_TRANSMIT. Στην πρώτη, αυτό που γίνεται είναι να διαβάζουμε από τον καταχωρητή R15 και να επιστρέφουμε το αποτέλεσμα που διαβάστηκε. Οσον αφορά τη δεύτερη, αυτό που κάνουμε είναι να προσπαθούμε με polling να στείλουμε το μηνυμα OK<CR><LF>.

Οσον αφορά τον TIMER0 OVF Handler αυτό που κάνουμε είναι να ελέγχουμε το digit counter για το ποιο ψηφιο προβάλλουμε στα 7-segments και αν η τιμη του έχει φτάσει το 0x08 σημαίνει ότι πρεπει να ξεκινήσουμε από το πρώτο στοιχείο. Στη συνέχεια, παίρνουμε από το array RECEIVED_NUMBERS το στοιχείο στη θέση digit_cnt. Παράλληλα, βγαζουμε στο PORTC έναν "1"στη θέση digit_cnt. Στο PORTA βγάζουμε ως έξοδο το στοιχείο στη θέση display_num, στην οποία βρίσκεται η αποκωδικοποίηση του αριθμού BCD που διαβάσαμε στη συγκεκριμένη μεταβλητή display_num. Τέλος, αρχικοποιούμε τον καταχωρητή TCNT0 ξανά με την τιμή "0x63" και αυξάνουμε τον digit_cnt κατα ένα.

Για τον Interrupt Handler του USART RXC αυτό που γίνεται είναι να διαβάζουμε τον καταχωρητή R15 με τη συνάρτηση USART _RECEIVE και ελέγχουμε κάθε φορά τον τρέχοντα χαρακτήρα που λήφθηκε με τον προηγούμενο που αποθηκεύσαμε στη μνήμη. Αυτό γίνεται με σκοπό να ελέγχουμε αν έρχεται σωστά η εντολή και αναλόγως να κάνουμε την κατάλληλη ενέργεια και να στείλουμε το OK<CR><LF>, αν όλα έγιναν με επιτυχία.

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

#define F_CPU 10000000
//#define FOSC 10000000
#define A_HEX 0x41
#define C_HEX 0x43
#define K_HEX 0x4B
#define N_HEX 0x4E
#define O_HEX 0x4F
#define T_HEX 0x54
#define X_HEX 0x58      // Used to indicate that we have stored a number - we don't care what number
#define LF_HEX 0x0A
#define CR_HEX 0x0D

#define BAUD 9600               //Definition of the desired baud rate
#define UBRR_BAUD F_CPU/16/BAUD-1       //Calculation of UBRR's initial value

register unsigned char input_char asm ("15");

uint8_t digit_cnt = 0x00;       //Keeps the digit we display
uint8_t prev_char= 0x00;        //Keeps the char we receive
uint8_t num_of_digits = 0x00;

uint8_t RECEIVED_NUMBERS[8];                                                    //8 numbers to display
uint8_t OK_MSG[4] = {0x6F, 0x4B, 0x0D, 0x0A};                                   //Here we store ok in memory
uint8_t DECODED_BCD_NUMBERS[11] = {0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8, 0x80, 0x90, 0xFF};      //Decoded bcd to 7-seg
```

(a) C code for define global variables

```c
void MEM_CLEAR(){
    for(uint8_t i=0; i<8; i++){
        RECEIVED_NUMBERS[i] = 0x0A;
    }
    return;
}

void MEM_SHIFT(){
    uint8_t temp;
    for(uint8_t i=7; i>0; i--){
        temp = RECEIVED_NUMBERS[i-1];
        RECEIVED_NUMBERS[i] = temp;
    }
    return;
}
```

(b) C code for memory clear and shift

```c
ISR(TIMER0_OVF_vect){
    uint8_t display_num;

    if(digit_cnt == 0x08){
        digit_cnt = 0x00;
    }
    display_num = RECEIVED_NUMBERS[digit_cnt];

    PORTA = 0xFF;
    PORTC = (1<<digit_cnt);

    PORTA = DECODED_BCD_NUMBERS[display_num];
    TCNT0 = 0x63;
    digit_cnt++;
}
```

(c) C code for TIMER0 OVF Interrupt Handler

```c
ISR(USART_RXC_vect){
    uint8_t current_char = USART_RECEIVE();

    if(prev_char == 0x00 && current_char == A_HEX){
        prev_char = A_HEX;
    }else if(prev_char == 0x00 && (current_char == C_HEX || current_char == N_HEX)){
        MEM_CLEAR();
        num_of_digits = 0x00;
        prev_char = current_char;
    }else if(prev_char == A_HEX && current_char == T_HEX){
        prev_char = T_HEX;
    }else if(current_char == CR_HEX && (prev_char == T_HEX || prev_char == C_HEX || prev_char == X_HEX)){
        prev_char = CR_HEX;
    }else if((current_char >= 0x30 && current_char <= 0x39) && (prev_char == N_HEX || prev_char == X_HEX)){
        MEM_SHIFT();
        num_of_digits++;
        if(num_of_digits<9){
            RECEIVED_NUMBERS[0] = current_char - 0x30;
            prev_char = X_HEX;
        }
    }else if(prev_char == CR_HEX && current_char == LF_HEX){
        USART_TRANSMIT();
    } else{
        prev_char = 0x00;
    }
}
```

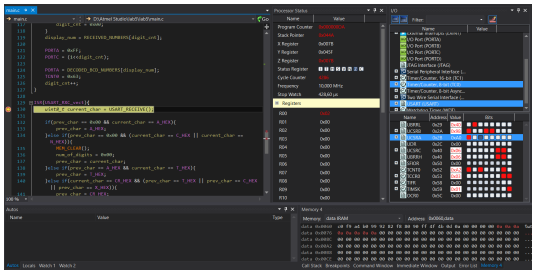
(d) C code for USART RXC Interrupt Handler

```c
void USART_TRANSMIT(){

    for(uint8_t i=0; i<4; i++){
        while (!( UCSRA & (1<<UDRE)));

        //UDR = ok_msg_addr[i];
        TCNT2 = OK_MSG[i];
    }
    prev_char = 0x00;
    return;
}

uint8_t USART_RECEIVE(){
    uint8_t received_data;

    while (!(UCSRA & (1<<RXC)));
    received_data = UDR;
    received_data = UDR;
    received_data = input_char;

    return received_data;
}
```

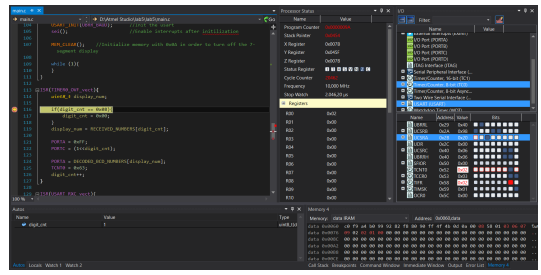
(e) C code for USART transmit and receive

Προσομοίωση Αποτελεσμάτων

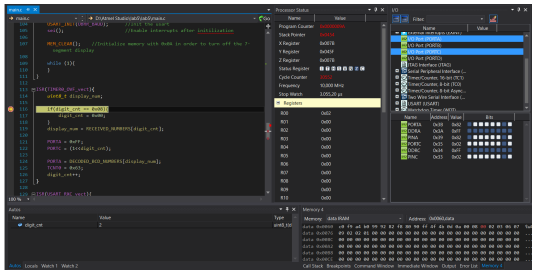
Οι προσομοιώσεις του συγκεκριμένου εργαστηρίου είναι ακριβώς ίδιες με των προηγούμενων, εφόσον δεν προστέθηκαν επιπλέον λειτουργίες. Ειδικότερα, στις παρακάτω εικόνες φαίνεται ότι γίνεται σωστά το initialization των καταχωρητών του TIMER0 και USART, καθώς και η αποθήκευση των αποκωδικοποιήσεων για τα 7-segment και του μηνυματος OK<CR><LF>. Παρατηρούμε ότι όλες οι τιμές αποθηκεύονται σειριακά στη μνήμη ξεκινώντας από τη διεύθυνση "0x60". Επίσης, βλέπουμε ότι εκτελείται σωστά ο USART handler και οι τιμές οι οποίες λαμβάνονται αποθηκεύονται στη μνήμη, ενώ τέλος φαίνεται ότι εκτελείται σωστά και ο TIMER0 handler και γίνεται σωστά η ανανέωση των ψηφίων των 7-segments κάθε περίπου 1 ms.



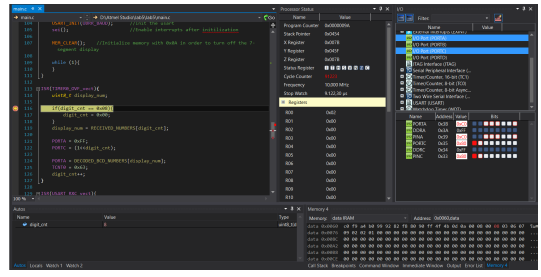
(a) Atmel Studio 7 - Initialization of TIMER0 and USART



(b) Atmel Studio 7 - Execution of N01229763<CR><LF>



(c) Atmel Studio 7 - Display Digit 1



(d) Atmel Studio 7 - Display Digit 7