

Εργαστήριο 3

ΟΝΟΜΑ: Μαυρογιώργης Δημήτρης

ΑΜ: 2016030016

ΗΡΥ 411 - Ενσωματωμένα Συστήματα Μικροεπεξεργαστών
ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

April 19, 2021

Σκοπός

Σκοπός του τέταρτου εργαστηρίου είναι να υλοποιήσουμε σε κώδικα C τις αρχικοποιήσεις των TIMER0 και USART, κρατώντας αυτούσιους τους κώδικες των interrupt handler που είχαν υλοποιηθεί στο εργαστήριο 2 και 3 αντίστοιχα. Στόχος μας είναι, δηλαδή, να συνδυάσουμε τον κώδικα C των αρχικοποιήσεων και του κώδικα assembly των handler προκειμένου να υλοποιήσουμε τις προδιαγραφές που είχαν περιγραφθεί στα δύο προηγούμενα εργαστήρια.

Περιγραφή της υλοποίησης

Στο συγκεκριμένο εργαστήριο οι αλλαγές που χρειάστηκε να γίνουν είναι να κάνουμε τις αρχικοποιήσεις των δύο handler σε C και στη main του προγράμματος να χρησιμοποιήσουμε ένα infinite loop. Ειδικότερα, οι αρχικοποιήσεις των τιμών των καταχωρητών είναι ακριβώς ίδιες με του προηγούμενου εργαστηρίου. Οι αρχικοποιήσεις αυτές έγιναν σε δύο ξεχωριστές συναρτήσεις και το μοναδικό που αλλάζει είναι ότι καλούνται στο main πρόγραμμα.

Όσον αφορά τον κώδικα assembly, η υλοποίησης των handler είναι η ίδια με του εργαστηρίου 3, με μοναδική αλλαγή τη μετονομασία τους από TIMER0_OVF και USART_RXC σε TIMER0_OVF_vect και USART_RXC_vect αντίστοιχα. Οι συγκεκριμένοι handler, δηλώθηκαν ως global για να μπορούν να αναγνωριστούν από τη C.

Παράλληλα, δηλώθηκε και η συνάρτηση MEM_INIT ως global, για την αρχικοποίηση των X και Y registers έτσι, ώστε να κρατούν τις διευθύνσεις των array που είναι αποθηκευμένες οι αποκωδικοποιήσεων των αριθμών για τα 7-segment και τα δεδομένα προς εμφάνιση αντίστοιχα.

Κατόπιν, για να είναι σβησμένα τα 7-segment display, έγινε ένα clear της μνήμης με τη συνάρτηση MEM_CLEAR, δηλαδή αποθηκεύτηκαν και στις 8 θέσεις το 0x0A. Για να μπορέσουμε να καλέσουμε και αυτή τη συνάρτηση στη C, δηλώθηκε και αυτή ως global. Τέλος, όλες οι δηλώσεις με .equ των γραμμάτων των εντολών, που δηλώθηκαν στην assembly του 3ου εργαστηρίου, έγιναν με #define.

Παρακάτω φαίνονται οι κώδικες C και οι αλλαγές του κώδικα Assembly, όπως περιγράφηκαν προηγουμένως.

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

#define F_CPU 1000000
// #define FOSC 10000000

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

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

uint8_t data_addr[8]; //8 numbers to display
uint8_t ok_msg_addr[4] = {0x4f, 0x48, 0x4d, 0x4a}; //Here we store ok in memory
uint8_t seg_dec_addr[11] = {0xc0, 0xf9, 0x4d, 0xb8, 0x59, 0xc2, 0x82, 0xf3, 0x40, 0x90, 0xf7}; //Decoded led to 7-seg

void TCNT0_INIT();
void USART_INIT(unsigned int ubrr);

extern void MEM_CLEAR();
extern void MEM_INIT();
```

(a) C code for defines global variables and function signatures

```
int main(void){
    cli(); //Turn off interrupts before initialization
    TCNT0_INIT(); //Init the timer0
    USART_INIT(UBRR_BAUD); //Init the usart
    sei(); //Enable interrupts after initialization
    MEM_INIT(); //Set X point to seg_dec_addr array and Y register point to data_addr array
    MEM_CLEAR(); //Initialize memory with 0x00 in order to turn off the 7-segment display

    while (1){
    }
}
```

(b) C code for main

```
void TCNT0_INIT(){
    TCNT0 = 0x63; // Set TCNT0 the value of 99(0x63) = 255 - (1 ms)*(1 Mhz)/64(prescale)
    TCCR0 = (1<<CS00)|(1<<CS00); //Set prescale 64 - these bits of TCCR0 are set - CS02 = 0 CS01 = 1 CS00 = 1
    TIMSK = (1<<TOIF0); // TOIF0 is set to 1 in order to enable the timer overflow interrupt

    DDRA = 0xff; //Set as output PORTA AND PORT C
    DDRC = 0xff;

    PORTA = 0xff; //Initialize PORTA with 0xff which means that 7-seg display is turned off
    PORTC = 0x00; //Initialize PORTC with 0x00 WHICH mean that none of the digits has enable

    return;
}
```

(c) C code for TIMER0 Initialization

```
void USART_INIT(unsigned int ubrr){
    UCSRB = (1<<RXIE)|(1<<RXEN)|(1<<TXEN); //Enable RXC interrupts, Receive and Transmit
    UBRRH = (unsigned char)(ubrr>>8); // Set baud rate
    UBRRL = (unsigned char)ubrr;

    UCSRC = (1<<URSEL); // We make it "1" in order to write to UCSRC
    UCSRC = (1<<UCSZ0)|(1<<UCSZ1); //8-bit size for everything we transmit and receive

    return;
}
```

(d) C code for USART RXC Initialization

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

#undef __SFR_OFFSET
#define __SFR_OFFSET 0

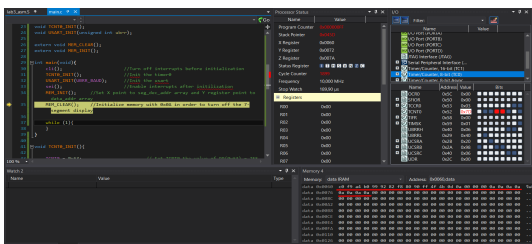
#define A_HEX 0x41
#define C_HEX 0x43
#define F_HEX 0x46
#define M_HEX 0x4E
#define Q_HEX 0x4F
#define T_HEX 0x54
#define X_HEX 0x58 // Used to indicate that we have stored a number - we don't care what number
#define LP_HEX 0x5a
#define CR_HEX 0x0d

global MEM_INIT
global MEM_CLEAR
global TIMER0_OVF_vect
global USART_RXC_vect

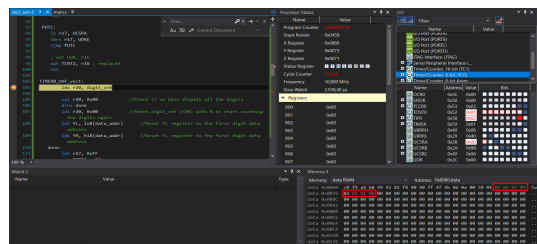
.section .text
```

(e) Assembly code changes

Οι προσομοιώσεις του συγκεκριμένου εργαστηρίου είναι ακριβώς ίδιες με του προηγούμενου, εφόσον δεν έγιναν καθόλου αλλαγές στην υλοποίηση των TIMER0 και USART handler. 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>. Επίσης, βλέπουμε ότι εκτελείται σωστά ο USART handler και οι τιμές οι οποίες λαμβάνονται αποθηκεύονται στη μνήμη, ενώ παράλληλα φαίνεται ότι εκτελείται σωστά και ο TIMER0 handler. Τέλος, όσον αφορά την αποστολή του OK<CR><LF>, μετά τη λήψη των αριθμών, γίνεται η εγγραφή του στο αρχείο lab4.log, γεγονός που επιβεβαιώνει τη σωστή λειτουργία.



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



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

Επιπρόσθετα, κατα την προσομοίωση του κώδικα C, μπορούμε να δούμε τον κώδικα Assembly που δημιουργήθηκε για τις αρχικοποιήσεις. Όπως φαίνεται παρακάτω ο κώδικας αρχικοποίησης σε assembly του 3ου εργαστηρίου και ο κώδικας που δημιούργησε ο compiler είναι παρόμοιοι, με μικρές διαφορές. Οι κυριότερες διαφορές είναι ότι τα ονόματα των καταχωρητών μεταφράστηκαν στις διευθύνσεις τους και μία άλλη αλλαγή είναι ότι χρησιμοποιούνται διαφορετικοί καταχωρητές (πχ στο 3ο Lab είχε χρησιμοποιηθεί για τον TIMER0 ο R16, ενώ στο συγκεκριμένο Lab χρησιμοποιείται ο R24).

```
TCNT0 = 0x63; // Set TCNT0 the value of 99(0x63) = 255 - (1 ms)*(1 Mhz)/64(prescale)
0000000E LDI R24,0x63 Load immediate
0000000F OUT 0x20,R24 Out to I/O location
TCCR0 = (1<<CS01)|(1<<CS00); //Set prescale 64 - these bits of TCCR0 are set - CS02 = 0 CS01 = 1 CS00 = 1
0000000B LDI R24,0x03 Load immediate
0000000C OUT 0x20,R24 Out to I/O location
TIMSK = (1<<TOIE0); // TOIE0 is set to 1 in order to enable the timer overflow interrupt
0000000E LDI R24,0x01 Load immediate
0000000F OUT 0x20,R24 Out to I/O location
DDRA = 0xFF; //Set as output PORTA AND PORT C
00000004 SER R24 Set Register
00000005 OUT 0x10,R24 Out to I/O location
DDRC = 0xFF;
0000000C OUT 0x10,R24 Out to I/O location
PORTA = 0xFF; //Initialize PORTA with 0xFF which means that 7-seg display is turned off
00000007 OUT 0x10,R24 Out to I/O location
PORTC = 0x00; //Initialize PORTC with 0x00 WHICH mean that none of the digits has enable
0000000B OUT 0x10,R24 Out to I/O location
00000009 RET Subroutine return
```

(a) Assembly generated code for TIMER0 initialization

```
UCSRB = (1<<RXCIF)|(1<<RXEN)|(1<<TXEN); //Enable RXC interrupts, Receive and Transmit
0000000A LDI R18,0x98 Load immediate
0000000B OUT 0x0A,R18 Out to I/O location
UBRRH = (unsigned char)(ubrr>>8); // Set baud rate
0000000C OUT 0x20,R25 Out to I/O location
UBRRL = (unsigned char)ubrr;
0000000D OUT 0x09,R24 Out to I/O location
UCSRC = (1<<URSEL); // We make it "1" in order to write to UCSRC
0000000E LDI R24,0x80 Load immediate
0000000F OUT 0x20,R24 Out to I/O location
UCSRC = (1<<UCS20)|(1<<UCS21); //8-bit size for everything we transmit and receive
0000000F LDI R24,0x06 Load immediate
0000000F1 OUT 0x20,R24 Out to I/O location
0000000F2 RET Subroutine return
```

(b) Assembly generated code for USART initialization