Ενσωματωμένα Συστήματα Μικροεπεξεργαστών

Project 5

Ημερομηνία Παράδοσης: 14 Νοεμβρίου 2021

Αμπλιανίτης Κωνσταντίνος

2017030014

Σκοπός της Άσκησης:

Σκοπός της άσκησης είναι η μετατροπή των ιντερρυπτς του εργαστηρίου 4 (ήταν γραμμένα σε γλώσσα Ασσεμβλψ) σε γλώσσα C. Ο λόγος για τον οποίο γίνεται αυτή η διαδικασία είναι για να μπορεί να γίνει διακριτή η διαφορά χρήσης των πόρων μεταξύ των δύο γλώσσεων.

Περιγραφή της τεχνολογίας που χρησιμοποιήθηκε:

Για την υλοποίηση της συγκεκριμμένης εργαστηριακής άσκησης χρησιμοποιήθηκε το Microchip Sudio (version 7.0.2542) της Microchip Technology.

Περιγραφή επίλυσης της άσκησης:

Δεδομένων των ζητουμένων της εκφώνησης, η υλοποίηση ήταν σχετικά εύκολη. Αρχικά, κρατώντας τις ISR εντολές γράφτηκαν σε C τόσο ο κώδικας υπεύθυνος για τα timer interrupts όσο και ο κώδικας υπεύθυνος για τα interrupt λόγω UART. Όπως φαίνεται παρακάτω ο κώδικας που γράφηκε για τις συναρτήσεις που προαναφέρθηκαν είναι εξαιρετικά απλός:

```
// practically tell the compiler that this is for the
// interrupt
ISR(TIMER0_OVF_vect,ISR_NAKED)
{
    uint8_t x = offset_reg; // load the offset register
    offset_reg++;

    x = port_C_segment[x];
    x = seven_seg_status[x];

    PORTA = x;
    PORTC = rotate_reg;

    rotate_reg  = rotate_reg << 1;
    if(rotate_reg == 0) // case we need reset
    {
        rotate_reg = 0b00000001;
        offset_reg = 0b00000000;
    }
     // timer reset
     TCNT0 = 96; // update the value in decimal
    reti();
}
```

(α′) Timer interrupt

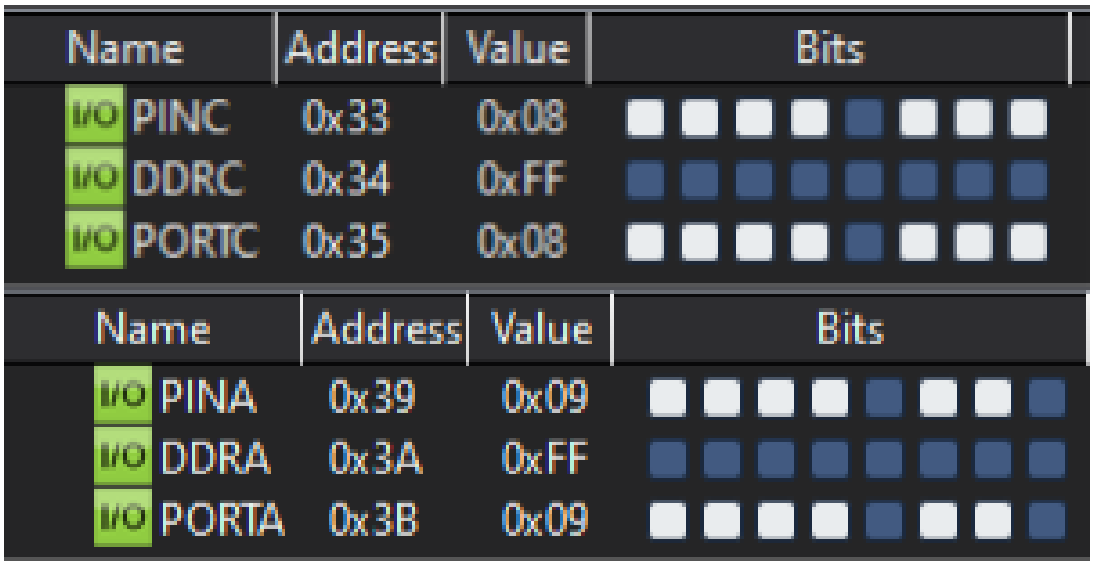
```
ISR(USART_RXC_vect,ISR_NAKED)
{
    uint8_t inputascii = UDR;
    inputascii = UDR; // put the UDR in the inputs to reset its flags so it does not hang up.
    inputascii = PORTB; // use the portB cause i did the exact change in the stimuli file.

    // create the switch function thing based on the input. I cannot use the switch function because of the fact that for the numbers I need a range
    if(inputascii == 0x41) // A
        asm("nop");
    else if(inputascii == 0x54) // T
        asm("nop");
    else if (inputascii == 0x43 || inputascii == 0x4E) // N or C
    {
        for(int i=0; i<8; i++)
            port_C_segment[i] = 0x0A; // set the offset to a condition that will keep the seven segment off.
    }
    else if (inputascii == 0x0A)
    {

        for(int i=0; i<4; i++)
        {
            while ( !( UCSRA & (1<<UDRE)) ) // wait for the buffer to become empty


            TCNT2 = okcrlf_table[i];
        }
    }
    else if(inputascii < 0x30) // non-intresting ascii char
        asm("nop");
    else if(inputascii < 0x3A) // number
    {
        inputascii = inputascii - 0x30; // take the number as bcd
        uint8_t temp ; // will be needed in order to push the values
        for(int i=0; i<8; i++)
        {
            temp = port_C_segment[i]; // save the  value that will be needed for  the  push
            port_C_segment[i] = inputascii; // place the the  pushed value in the  specific location
            inputascii = temp; // update the input with the next number that should be pushed
        }
    }
    reti();
}
```

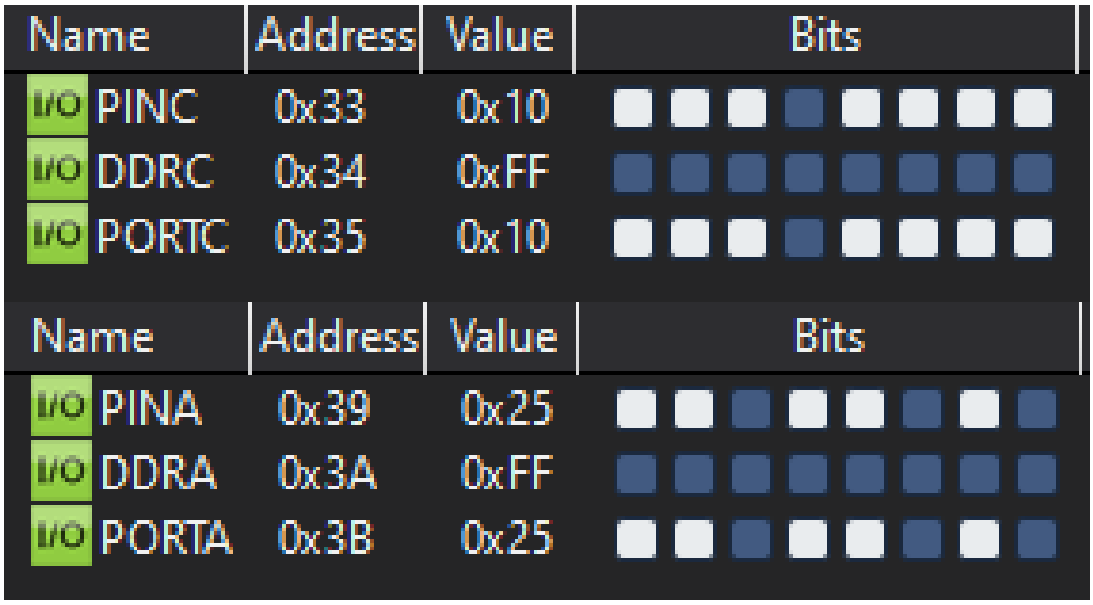
(β′) UART interrupt

Επιπλέον για να γίνει σωστά η λειτουργία χρειάστηκε να γίνουν αλλαγές στο stimuli file. Δεδομένου ότι μέσω της Assembly υπήρχε πρόσβαση κατευθείαν στους καταχωρητές και συνεπώς υπήρχε δυνατότητα να γνωρίζει κανείς ποιός καταχωρητής είναι για είσοδο (r15) το stimuli file των προηγούμενων εργαστηρίων δεν θα δουλέψει. Για να μπορεί να κάνει σωστά την δουλειά του αρκεί ο καταχωρητής R15 να αντικατασταθεί με κάποιο PORT. Δεδομένου ότι τα PORTA PORTC χρησιμοποιούνται ήδη για άλλους σκοπούς θα χρησιμοποιηθεί το PORTB.

Επιβεβαίωση σωστής λειτουργίας

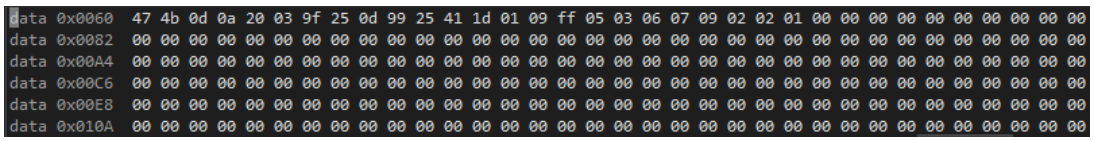
Όπως φαίνεται στις παρακάτω ενδεικτικές εικόνες, η λειτουργία του microcontroller παραμένει ίδια με του προηγούμενου εργαστηρίου όπως και του εργαστηρίου 3.





Κατάσταση Μνήμης:

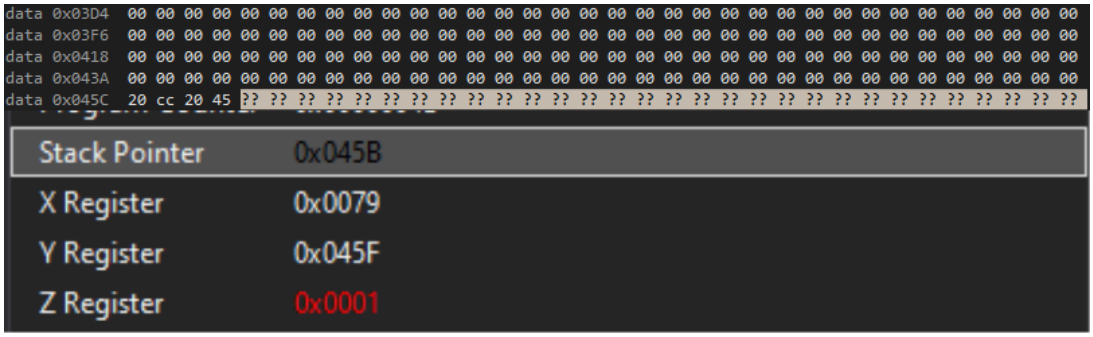
Όσον αφορά την SRAM έχουμε την εξής κατάσταση.



Από την θέση 0x0060 ως την θέση 0x0063 (4 πρώτες θέσεις της SRAM) βρίσκονται οι απεικονίσεις των OK<CR><LF> (προστέθηκαν στο συγκεκριμένο εργαστήριο). Στην

συνέχεια, στην θέση 0x0064 τοποθετείται ο rotate_reg. Ακολουθούν στις επόμενες 11 θέσεις (θέσεις 0x0065-0x006F) αρχικοποιούνται οι πιθανές καταστάσεις απεικόνισης του PORTA. Έπειτα υπάρχει ο offset_reg (θέση 0x0070) και τέλος οι επόμενες 8 θέσεις είναι δευσμευμένες για την κράτηση του offset για την απεικόνιση αριθμού για κάθε μία από τα 8 seven segment displays.

Στο τέλος της SRAM υπάρχει η στοίβα η οποία μας δείχνει ότι έχει τιμές στους καταχωρητές 0x045F(τέλος της στοίβας και τέλος της SRAM) καθώς και στις θέσεις 0x045C-0x045E. Επιπλέον στην δεύτερη φωτογραφία φαίνεται ότι ο stack pointer βρισκεται στην 0x045B όπου δεν φαίνεται να υπάρχει τιμή οπότε κατά πάσα πιθανότητα έγινε popped η τιμή του.



Αντιστοιχίες δομών Assembly.

Για τον timerovf που αντικαταστάθηκε από κώδικα C παρουσιάζεται η Assembly που προκύπτει από το πρόγραμμα

```
    uint8_t x = offset_reg; // load the offset register
00000049  LDS R30,0x0070        Load direct from data space
    offset_reg++;
0000004B  LDI R24,0x01      Load immediate
0000004C  ADD R24,R30       Add without carry
0000004D  STS 0x0070,R24        Store direct to data space
    x = port_C_segment[x];
0000004F  LDI R31,0x00      Load immediate
00000050  SUBI R30,0x8F     Subtract immediate
00000051  SBCI R31,0xFF     Subtract immediate with carry
00000052  LDD R30,Z+0       Load indirect with displacement
    x = seven_seg_status[x];
00000053  LDI R31,0x00      Load immediate
00000054  SUBI R30,0x9B     Subtract immediate
00000055  SBCI R31,0xFF     Subtract immediate with carry
00000056  LDD R24,Z+0       Load indirect with displacement
    PORTA = x;
00000057  OUT 0x1B,R24      Out to I/O location
    PORTC = rotate_reg;
00000058  LDS R24,0x0064        Load direct from data space
0000005A  OUT 0x15,R24      Out to I/O location
    rotate_reg  = rotate_reg << 1;
0000005B  LDS R24,0x0064        Load direct from data space
0000005D  LSL R24       Logical Shift Left
0000005E  STS 0x0064,R24        Store direct to data space
    if(rotate_reg == 0) // case we need reset
00000060  CPSE R24,R1       Compare, skip if equal
00000061  RJMP PC+0x0006        Relative jump
        rotate_reg = 0b00000001;
00000062  LDI R24,0x01      Load immediate
00000063  STS 0x0064,R24        Store direct to data space
       offset_reg = 0b00000000;
00000065  STS 0x0070,R1     Store direct to data space
     TCNT0 = 96; // update the value in decimal
00000067  LDI R24,0x60      Load immediate
00000068  OUT 0x32,R24      Out to I/O location
    reti();
00000069  RETI      Interrupt return
```

Παρατηρείται ότι ο κώδικας διαφέρει από τον αντίστοιχό που υλοποιήθηκε σε Assembly τα προηγούμενα εργαστήρια. Γίνεται χρήση πολύ περισσότερων πόρων καθώς και σπατάλη πολλών κύκλων ρολογιού. Επιπλέον παρατηρείται ότι τα αντίστοιχα rcalls που είχαν υλοποιηθεί στα προηγούμενα εργαστήρια στην μετάβαση στην C αντικαθίστανται από rjmps. Παρόμοια κατάσταση φαίνεται και στο UART interrupt παρακάτω.

```
    uint8_t inputascii = UDR;
0000006A  IN R24,0x0C       In from I/O location
    inputascii = UDR; // put the UDR in the inputs to reset its flags so it does not hang up.
0000006B  IN R24,0x0C       In from I/O location
    inputascii = PORTB; // use the portB cause i did the exact change in the stimuli file.
0000006C  IN R24,0x18       In from I/O location
    if(inputascii == 0x41) // A
0000006D  CPI R24,0x41      Compare with immediate
0000006E  BRNE PC+0x03      Branch if not equal
        asm("nop");
0000006F  NOP       No operation
00000070  RJMP PC+0x0042        Relative jump
    else if(inputascii == 0x54) // T
00000071  CPI R24,0x54      Compare with immediate
00000072  BRNE PC+0x03      Branch if not equal
        asm("nop");
00000073  NOP       No operation
00000074  RJMP PC+0x003E        Relative jump
    else if (inputascii == 0x43 || inputascii == 0x4E) // N or C
00000075  CPI R24,0x43      Compare with immediate
00000076  BREQ PC+0x0B      Branch if equal
```

Συμπεράσματα

Γενικά μέσω του εργαστηρίου μπορεί κανείς να βγάλει το εξής συμπέρασμα. Υπάρχει ένα trade-off ευκολίας-κατανάλωσης πόρων. Η χρήση της γλώσσας C ναι μεν κάνει το πρόγραμμα πιο εύκολο στην υλοποίηση του αλλά δεσμεύει πόρους που θα μπορούσαν να γίνουν διαθέσιμοι για άλλες λειτουργίες. Επιπλέον ο κώδικας που προκύπτει σε Assembly σε περίπτωση χρήσης του για μικρο-αλλαγές είναι αρκετά δυσανάγνωστος. Από την άλλη μερία η υλοποίηση σε Assembly από την αρχή είναι αρκετά δυσκολότερη αλλά υπάρχει πολύ καλύτερη διαχείρηση των πόρων και η οργάνωση της μνήμης γίνεται όπως θέλει ο προγραμματιστής.