Εργαστήριο 8

07/12/2020

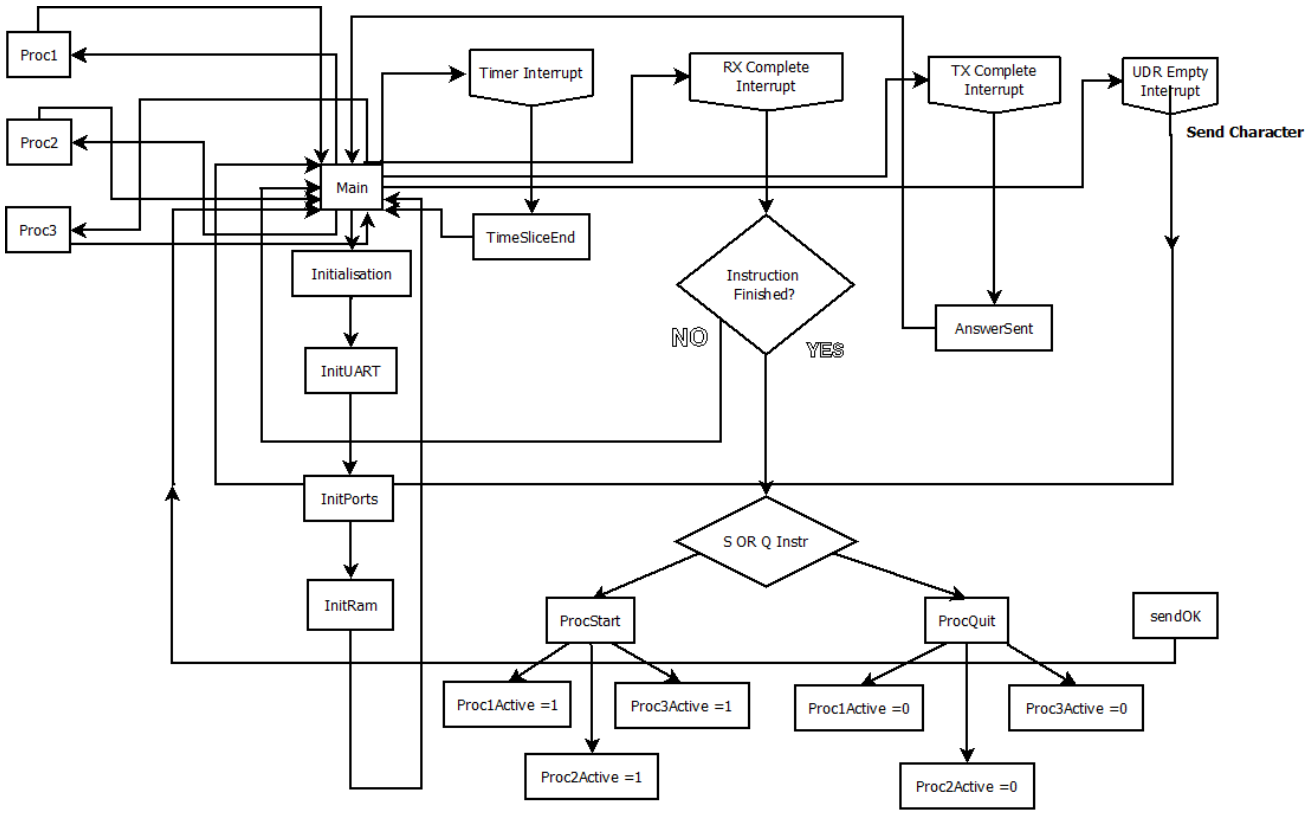
ΥΠΟΡΟΥΤΙΝΕΣ, ΚΑΙ ΕΝΑΣ ΠΟΛΥ ΑΠΛΟΣ ΔΡΟΜΟΛΟΓΗΤΗΣ

Ντουνέτας Δημήτρης
ΑΜ: 2016030141

Εισαγωγή

Σκοπός του εργαστηρίου είναι η δημιουργία ενός απλού δρομολογητή για την δρομολόγηση 3 απλών υπορουτίνων σε Μικροελεγκτή AVR. Επιπλέον, η παρατήρηση της assembly εξόδου από τον C Compiler και πως κάνει χρήση των πόρων του συστήματος. Τέλος η παρατήρηση για τη λειτουργία χωρίς προβλήματα και πανωγράψιμο της μνήμης και χρήση USART του ATMEGA16, για παραλαβή εντολών και αποστολή απαντήσεων.

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 την λειτουργία του βρόγχου που εκτελεί τον Scheduler. Για γίνει η αρχικοποίηση, η Main καλεί τη συνάρτηση initialization. Ο 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 initUART, initRam και initPorts συναρτήσεις οι οποίες κάνουν τις αρχικοποιήσεις μνήμης , καταχωρητών, θέσεων μνήμης και Ports όπως έχουν αναφερθεί και στα προηγούμενα εργαστήρια. Οι μόνες διαφορές στην αρχικοποίηση του εργαστηρίου είναι ότι αλλάζει το prescale του TIMER/COUNTER1 που χρησιμοποιούμε για το timeslice και γίνεται 1/64 βάζοντας άσσους στα bit CS11 και CS10 του καταχωρητή TCCR1B και αλλάζουμε την τιμή του Καταχωρητή OCR1A σε 15625 ώστε με ρολόι 10Mhz να μετράει για χρόνο 100ms. Συνοπτικά χρησιμοποιούνται 3 σημεία στην μνήμη SRAM στα οποία το ένα με όνομα UARTAnsAddress όπου γράφεται η αυτοματοποιημένη απάντηση που δίνει ο Μικροελεγκτής μας όταν λάβει ένα ολόκληρο Instruction σωστά δηλαδή OK<CR><LF>. Το δεύτερο, είναι το InstrAddress όπου αποθηκεύεται το instruction που λαμβάνουμε ώστε να μπορούμε να το διαχειριστούμε κατάλληλα όταν θέλουμε να εκτελέσουμε την εντολή. Τέλος το τρίτο είναι το κομμάτι που αποθηκεύουμε τις μεταβλητές που χρειαζόμαστε για να λειτουργούν σωστά οι υπορουτίνες μας και ο Scheduler. Το πρόγραμμα αρχικοποιείται με καμία διεργασία ενεργή.

```
void initRam(){

    // UART answer Initialization
    *UARTAnsAddress = ASCII_O;
    UARTAnsAddress += 1;
    *UARTAnsAddress = ASCII_K;
    UARTAnsAddress += 1;
    *UARTAnsAddress = ASCII_CR;
    UARTAnsAddress += 1;
    *UARTAnsAddress = ASCII_LF;
    UARTAnsAddress += 1;
    UARTAnsAddress = (uint8_t *)0x0090;

    //  Instruction Address Initialization
    InstrAddress = (uint8_t *)0x00A0;

    //  Left RingCounter init
    *RingCounterL = 0b00000001;

    //  Right RingCounter init
    *RingCounterR = 0b10000000;

    //  Led Toggle init
    *LedToggle = 0b01010101;

    //  TimeSliceEND init
    *TimeSliceEND = 0;
    //  Current Process
    *CurrProcess = 0;
    //  Process1 Active
    *Proc1Active = 0;
    //  Process2 Active
    *Proc2Active = 0;
    //  Process3 Active
    *Proc3Active = 0;
}
```

```
void initUART(){

    UBRRL = LOW(UBRRValue);         // load baud prescale
    UBRRH = HIGH(UBRRValue);        // to UBRR

    UCSRB = (1<<RXEN)|(1<<TXEN)|(1<<RXCIE)|(1<<TXCIE); // enable transmitter, receiver
    UCSRC = (1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0);          //Set frame format: 8data, 1stop bit
}
```

```
void initPorts(){
    //  Ports Initialization
    DDRA = 0xFF;    //Port A Initialization as output

    //  Enable Compare Interrupt
    TIMSK = (1<<OCIE1A);
    OCR1AH = HIGH(LoopValue);
    OCR1AL = LOW(LoopValue);
    TCCR1B = (1<<CS11)|(1<<CS10)|(1<<WGM12);
}
```

```
void initialisation(void){
    //Chip Initializations
        initUART();
        initRam();
        initPorts();
    //  Enable interrupts
        sei();
}
```

RAM MAP

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

0xA0	0xA1	0xA2	0xA3
instrByte0	instrByte1	instrByte2	instrByte3

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

0xB0	0xB1	0xB2	0xB3	0xB4	0xB5	0xB6	0xB7
RingCounterL	RingCounterR	LedToggle	TimeSliceEnd	currProcess	Proc1Active	Proc2Active	Proc3Active

Υπολογισμός του TimeSlice

Για τον υπολογισμό του TimeSlice χρησιμοποιούμε τον 16-bit μετρητή Timer/Counter 1 σε CTC mode με prescale 1/64 και τιμή Compare = 15625. Έτσι όταν ο μετρητής μετρήσει μέχρι αυτή τη τιμή θα έχουν περάσει 100ms σύμφωνα με το ρολόι του Μικροελεγκτή χρονισμένο στα 10Mhz. Έτσι για να υποδηλώσουμε τη λήξη του TimeSlice κάνουμε την τιμή του 1.

```
    *TimeSliceEND = 1;
    return;
}
```

Λειτουργία του κύριου προγράμματος και τα Interrupt 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

- Το Timer/Counter 1 Compare A Interrupt είναι υπεύθυνο για το σωστό χρονισμό του timeslice της οθόνης.

```
ISR(TIMER1_COMPA_vect)
{
    CTC_Timer_Interrupt();
}
```

- Καλεί την συνάρτηση CTC_TIMER_Interrupt η οποία μας δίνει την τιμή 1 στη μεταβλητή TimeSliceEND και προσθέτει +1 στην τιμή του currеProcess. Αν το currProcess πάρει την τιμή 4 το αρχικοποιεί ξανά στη τιμή 1

- 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.

```
ISR(USART_RXC_vect)
{
    RXC_Interrupt();
}
```

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 απανωγράφουμε χωρίς να έχουν αποσταλεί πρώτα τα δεδομένα που θέλουμε.

```
ISR(USART_UDRE_vect)
{
    UDRE_Interrupt();
}
```

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

- Το 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.

```
ISR(USART_TXC_vect)
{
    UTXC_Interrupt();
}
```

Οι Υπορουτίνες

Οι υπορουτίνες που θα διαχειρίζεται ο απλός δρομολογητής είναι 3 απλές διεργασίες.
Η πρώτη είναι ένας αριστερόστροφος μετρητής δακτυλίου, η δεύτερη ένας δεξιόστροφος μετρητής δακτυλίου και η τρίτη μια διαδικασία εναλλαγής Led από 01010101 σε 10101010 και αντίστροφα. Είναι πολύ απλές διεργασίες καθώς αυτή ήταν η απαίτηση της άσκησης με τη σημαντική λεπτομέρεια ότι όλες έχουν ως έξοδο το PortA του Μικροελεγκτή. Η έξοδος ανανεώνεται κάθε φορά που γίνεται κάποια αλλαγή και υπολογίζεται μια καινούργια τιμή για κάθε μετρητή, έτσι η έξοδος αλλάζει πού γρήγορα για να είναι αντιληπτή από το ανθρώπινο μάτι όμως μπορούμε να καταγράψουμε τις αλλαγές που συμβαίνουν στο log αρχείο που παράγεται μέσω ενός Stimuli File.

```
void Proc1(){

        //  Ringcounter Left
        if (*RingCounterL == 0b00000000)
        {
            *RingCounterL = 0b00000001;
        }else{
            *RingCounterL = *RingCounterL << 1;
        }

        PORTA = *RingCounterL;

    // Return is not necessary because Compiler adds- return in disassembly
}
```

```
void Proc2(){

        //  Ringcounter Right
        if (*RingCounterR == 0b00000000)
        {
            *RingCounterR = 0b10000000;
        }else{
            *RingCounterR = *RingCounterR >> 1;
        }
        PORTA = *RingCounterR;

    // Return is not necessary because Compiler adds- return in disassembly
}
```

```
void Proc3(){

    *LedToggle = *LedToggle ^ 0b11111111;

    PORTA = *LedToggle;

    *TimeSliceEND = 0;
    // Return is not necessary because Compiler adds- return in disassembly
}
```

Ενεργοποίηση / Απενεργοποίηση Διεργασιών

Για την ενεργοποίηση και απενεργοποίηση των διεργασιών λαμβάνουμε από τον χρήστη μέσω UART εντολές με μορφή SxCRLF για την ενεργοποίηση μιας διεργασίας και μορφή QxSRLF για την απενεργοποίηση της. Έτσι καθώς έχουμε λάβει την ολοκληρωμένη εντολή από τον UART καλούμε την κατάλληλη συνάρτηση ανάλογα με το πρώτο χαρακτήρα που έχουμε στην θέση InstrAddress όπου αποθηκεύεται ολόκληρη η εντολή. Έτσι όταν έχουμε λάβει μια εντολή που ξεκινάει από S καλούμε την εντολή ProcStart, ενώ αν λάβουμε εντολή που ξεκινάει από Q καλούμε την εντολή ProcQuit.

```
if(*InstrAddress == ASCII_LF){
    InstrAddress = (uint8_t *)0x00A0;
    switch(*InstrAddress){
        case ASCII_S:
        ProcStart();
        break;
        case ASCII_Q:
        ProcQuit();
        break;
        default:
        //error wrong instruction

        break;
    }
```

Οι συναρτήσεις ProcStart και ProcQuit

Σε αυτές τις συναρτήσεις ελέγχουμε τον δεύτερο χαρακτήρα την θέση μνήμης InstrAddress που μας ενημερώνει ποια διεργασία ενεργοποιούμε ή απενεργοποιούμε. Μέσα στην procStart σύμφωνα με τον αριθμό που είναι αποθηκευμένος αλλάζουμε την τιμή της κατάλληλης μεταβλητής από τις 3 που έχουμε δημιουργήσει και της δίνουμε την τιμή 1. Δηλαδή σε μια εντολή S1CRLF δίνουμε στην μεταβλητή Proc1Active την τιμή 1. Αντίθετα, στην συνάρτηση ProcQuit με ακριβώς τον αντίστοιχο τρόπο δίνουμε στην μεταβλητή την τιμή 0. Έτσι με αυτές τις 3 μεταβλητές γνωρίζουμε ποιες διεργασίες είναι ενεργές. Τέλος, αφού γίνει σωστά η ενεργοποίηση/απενεργοποίηση μιας διεργασίας στέλνουμε την απάντηση μέσω της συνάρτησης sendAns, όπως έχει εξηγηθεί στα προηγούμενα εργαστήρια.

```
void ProcStart(){
    InstrAddress +=1;
    switch(*InstrAddress){
        case ASCII_1:
        *Proc1Active = 1;
        break;
        case ASCII_2:
        *Proc2Active = 1;
        break;
        case ASCII_3:
        *Proc3Active = 1;
        break;
    }
    sendAns();
    InstrAddress = (uint8_t *)0x00A0;
```

```
void ProcQuit(){
    InstrAddress +=1;
    switch(*InstrAddress){
        case ASCII_1:
        *Proc1Active = 0;
        break;
        case ASCII_2:
        *Proc2Active = 0;
        break;
        case ASCII_3:
        *Proc3Active = 0;
        break;
    }
    sendAns();
    InstrAddress = (uint8_t *)0x00A0;
}
```

Ορισμός Επόμενης Διεργασίας προς Δρομολόγηση

Κάθε φορά που έρχεται το τέλος ενός TimeSlice πρέπει να αποφασίσουμε ποια διεργασία θα τρέξει μετά, οπότε και να ενημερώσουμε κατάλληλα το currProc ώστε ο δρομολογητής να επιτρέψει σε αυτή τη διεργασία να τρέξει. Αυτό γίνεται μέσα στην συνάρτηση CTC_Timer_Interrupt. Ο τρόπος που ελέγχουμε και ορίζουμε την επόμενη διεργασία είναι απλός και έχει σκοπό την δρομολόγηση με ίση προτεραιότητα σε όλες τις διεργασίες ώστε ο χρόνος που τρέχει η κάθε μια να διανέμεται ομοιόμορφα. Έτσι λαμβάνοντας υπόψιν κάθε 100ms ποια διεργασία έτρεχε στο προηγούμενο TimeSlice και ποιες διεργασίες είναι ενεργές βάζουμε στη μεταβλητή currProccess την διεργασία που θέλουμε να τρέξει κατά τη διάρκεια του επόμενου TimeSlice.

```
void CTC_Timer_Interrupt(){

    switch(*CurrProcess){
        case 1:
        if (*Proc2Active == 1)
        {
            *CurrProcess = 2;
        }
        else if (*Proc3Active)
        {
            *CurrProcess = 3;
        }
        else if(*Proc1Active)
        {
            *CurrProcess = 1;
        }
        else
        {
            *CurrProcess = 0;
        }
        break;
        case 2:
        if (*Proc3Active == 1)
        {
            *CurrProcess = 3;
        }
        else if(*Proc1Active)
        {
            *CurrProcess = 1;
        }
        else if(*Proc2Active)
        {
            *CurrProcess = 2;
        }
        else
        {
            *CurrProcess = 0;
        }
        break;
```

```
case 3:
if (*Proc1Active == 1)
{
    *CurrProcess = 1;
}
else if(*Proc2Active)
{
    *CurrProcess = 2;
}
else if(*Proc3Active)
{
    *CurrProcess = 3;
}
else
{
    *CurrProcess = 0;
}
break;
default:
if (*Proc1Active == 1)
{
    *CurrProcess = 1;
}
else if(*Proc2Active)
{
    *CurrProcess = 2;
}
else if(*Proc3Active)
{
    *CurrProcess = 3;
}
else
{
    *CurrProcess = 0;
}
break;
```

Ο απλός Δρομολογητής

Στο σώμα της συνάρτησης Main υλοποιείται ο απλός δρομολογητής που εναλλάσσει κατάλληλα τις διεργασίες κάθε 100ms. Σε αρχική φάση και για το εργαστήριο 7 δεν ενεργοποιούνται – απενεργοποιούνται διεργασίες και έτσι οι 3 διεργασίες εναλλάσσονται διαδοχικά η μια μετά την άλλη. Για να γίνει αυτό, απλά επιλέγουμε τη διεργασία που θα τρέξει σύμφωνα με τη τιμή που έχει η μεταβλητή currProcess. Η διεργασία τρέχει και αφού τελειώσει επιστρέφει πίσω εκεί που κλήθηκε όπου και ξανά καλείται για όσο διαρκεί το κάθε Timeslice. Το Timeslice τελειώνει όταν ο Timer/ Counter που έχουμε υλοποιήσει μετρήσει 100 ms και τότε αλλάζει τη τιμή του TimeSliceEND από 0 σε 1. Βγαίνουμε από τον βρόγχο κλήσης της υπορουτίνας και τότε κάνουμε ξανά το TimeSliceEND 0 ώστε να συνεχίσει η επόμενη υπορουτίνα.

```
switch(*CurrProcess){
    case 1:

        while(*TimeSliceEND == 0){

        Proc1();

        }
        *TimeSliceEND = 0;

    break;
    case 2:

        while(*TimeSliceEND == 0)
        {
            Proc2();
        }
        *TimeSliceEND = 0;

    break;
    case 3:

        while(*TimeSliceEND == 0)
        {
            Proc3();
        }
        *TimeSliceEND = 0;

    break;
    default:
    CTC_Timer_Interrupt();
    break;
```

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. Μπορούμε να δούμε την έξοδο που παράγει το πρόγραμμα στο PortA και τις απαντήσεις που δίνει μέσω UART στον καταχωρητή TCNT2.

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.