Εργαστήριο 5

15/11/2020

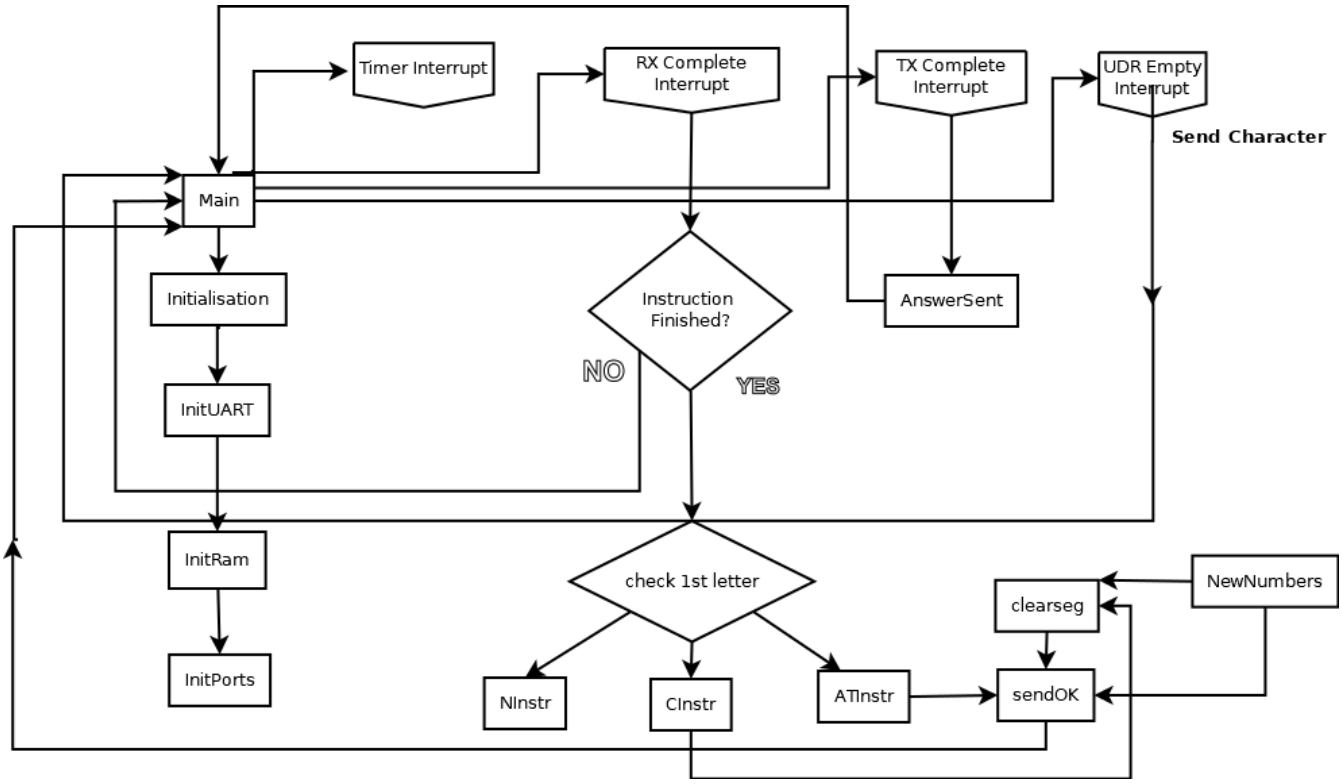
ΚΩΔΙΚΑΣ C ΣΤΟΝ ATMEL AVR

Ντουνέτας Δημήτρης
AM: 2016030141

Εισαγωγή

Σκοπός του εργαστηρίου είναι η εξοικείωση με τη γλώσσα C, τον Compiler AVR GCC καθώς και το AVRLIBC πακέτο για προγραμματισμό Μικροελεγκτών AVR. Επιπλέον, η κατανόηση της χρήσης C για την υλοποιήση εφαρμογών με πιο Optimized κώδικες. Τέλος, παρατήρηση της assembly εξόδου από τον C Compiler και πως κάνει χρήση των πόρων του συστήματος. Δημιουργία προγράμματος για επικοινωνία με χρήση USART του ATMEGA16, για παραλαβή εντολών και αποστολή απαντήσεων σε συνδυασμό με οδήγηση με πολυπλεξία στον χρόνο μίας οθόνης 7-segment LED για (έως) οκτώ ψηφία.

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 τον ατέρμονα βρόγχο πάνω στον οποίο τρέχει το πρόγραμμα. 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 συνάρτήσεις οι οποίες κάνουν τις αρχικοποιήσεις μνήμης , καταχωρητών, θέσεων μνήμης και Ports όπως αναφέρθηκαν στο προηγούμενο εργαστήριο. Συνοπτικά χρησιμοποιούνται 4 σημεία στην μνήμη SRAM στα οποία το ένα με όνομα display_address είναι το σημείο στο οποίο γράφονται τα BCD στοιχεία που θα δείξει η οθόνη. Το δεύτερο είναι το UARTAnsAddress όπου γράφεται η αυτοματοποιημένη απάντηση που δίνει ο Μικροελεγκτής μας όταν λάβει ένα ολόκληρο Instruction σωστά δηλαδή OK<CR><LF>. Το τρίτο, είναι το InstrAddress όπου αποθηκεύεται το instruction που λαμβάνουμε ώστε να μπορούμε να το διαχειριστούμε κατάλληλα όταν θέλουμε να εκτελέσουμε την εντολή. Τέλος το τέταρτο είναι το DecAddress όπου γίνεται το Decode από BCD σε 7-segment αριθμό ώστε να αποδίδεται σωστά στην οθόνη.

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

int main(void)
{
    initialisation();
    while (1)
    {

    }
}
```

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

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

```
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 initRam(){
    // RAM Initialization for display
    uint8_t i;
    for(i=0;i<=7;i++){

        *display_address = i;
        display_address += 1;
        }
    display_address = 0x0070;
    // UART answer Initialization
        *UARTAnsAddress = ASCII_O;
        UARTAnsAddress += 1;
        *UARTAnsAddress = ASCII_K;
        UARTAnsAddress += 1;
        *UARTAnsAddress = ASCII_CR;
        UARTAnsAddress += 1;
        *UARTAnsAddress = ASCII_LF;
        UARTAnsAddress += 1;

        UARTAnsAddress = 0x0080;

        //Instruction Address Initialization
        InstrAddress = 0x0090;

        //Counters Initializations
        RingCounterAddress = 0x0100;
        *RingCounterAddress = 0b00000001;

        DigitRegAddress = 0x0101;
        DigitCounterAdress = 0x0102;
}
```

```
//Decode Address Initialization
//7seg 0
*DecAddress = 0b11000000;
DecAddress += 1;
//7seg 1
*DecAddress = 0b11111001;
DecAddress += 1;
//7seg 2
*DecAddress = 0b10100100;
DecAddress += 1;
//7seg 3
*DecAddress = 0b10110000;
DecAddress += 1;
//7seg 4
*DecAddress = 0b10011001;
DecAddress += 1;
//7seg 5
*DecAddress = 0b10010010;
DecAddress += 1;
//7seg 6
*DecAddress = 0b10000010;
DecAddress += 1;
//7seg 7
*DecAddress = 0b11111000;
DecAddress += 1;
//7seg 8
*DecAddress = 0b10000000;
DecAddress += 1;
//7seg 9
*DecAddress = 0b10011000;
DecAddress += 1;
//7seg clear
*DecAddress = 0b11111111;
DecAddress = 0x00A0;
```

RAM MAP

0x70	0x71	0x72	0x73	0x74	0x75	0x76	0x77
7segbit7(LSB)	7segbit6	7segbit5	7segbit4	7segbit3	7segbit2	7segbit1	7segbit0(MSB)

Η λέξη που βρίσκεται στη θέση 0x70 είναι το δεξιότερο Digit που θα εμφανιστεί στο 7seg-Screen.

0x80	0x81	0x82	0x83
UARTAnsByte7(O)	UARTAnsByte6(K)	UARTAnsByte5(CR)	UARTAns4(LF)

0x90	0x91	0x92	0x93	0x94	0x95	0x96	0x97	0x98
instrByte0	instrByte1	instrByte2	instrByte3	instrByte4	instrByte5	instrByte6	instrByte7	instrByte8

0x99	0x9A	0x9B
instrByte9	instrByte10	instrByte11

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

0xA0	0xA1	0xA2	0xA3	0xA4	0xA5	0xA6	0xA7	0xA8
DecByte0	DecByte1	DecByte2	DecByte3	DecByte4	DecByte5	DecByte6	DecByte7	DecByte8

0xA9	0xAA	0xAB
DecByte9	DecByte10	DecByte11

0x100	0x101	0x102
RingCounterAddress	DigitRegAddress	DigitCounterAdress

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:

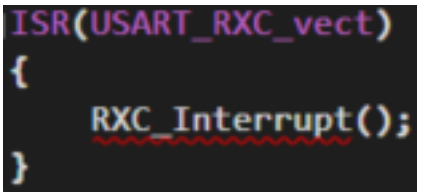
1. Timer/Counter Compare1 Interrupt
2. USART RX Complete Interrupt
3. USART DATA Registry Empty Interrupt
4. USART TX Complete Interrupt

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

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

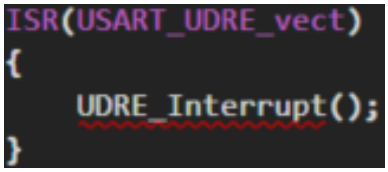
- Καλεί την συνάρτηση CTC_TIMER_Interrupt η οποία μας δίνει το κατάλληλο χρονικό περιθώριο για να ανανεώνουμε την 7segment οθόνη για να δώσουμε τα FPS που θέλουμε.

- 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.



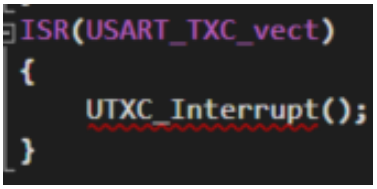
  Καλεί την συνάρτηση RXC _Interrupt η οποία λαμβάνει ένα χαρακτήρα και τον αποθηκεύει. Επίσης ελέγχει αν έχει ολοκληρωθεί η λήψη της εντολής, καθώς και ποια εντολή έχει ληφθεί.

- Το USART DATA Registry Empty Interrupt είναι υπεύθυνο να ενημερώνει όταν το UDR καταχωρητής είναι διαθέσιμος για εγγραφή ώστε να μην απανογράφουμε χωρίς να έχουν αποσταλεί πρώτα τα δεδομένα που θέλουμε.



  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.



Διαχείριση εντολών

Για την διαχείριση των εντολών που λαμβάνουμε μέσω του UART υπάρχουν ξεχωριστές συναρτήσεις που ενεργούν αφού λάβουμε σωστά ολόκληρη την εντολή και ουσιαστικά μας έρθει ο χαρακτήρας CR ακολουθούμενος από τον LF. Ελέγχουμε τον πρώτο χαρακτήρα αφού θεωρούμε σωστές τις εντολές και εκτελούμε κατάλληλα τη συνάρτηση που πρέπει για κάθε εντολή.

```
//  if character is LF it means we have the last char of the instruction
// so we check which instruction we have
    if(*InstrAddress == ASCII_LF){
        InstrAddress = 0x0090;
        switch(*InstrAddress){
            case ASCII_N:
            NumberInstr();
            break;
            case ASCII_C:
            ClearInstr();
            break;
            case ASCII_A:
            AttentionInstr();
            break;
            default:
            //error wrong instruction
            printf("Error wrong instruction");
            break;
        }
    }
```

ATTENTION Εντολή

Όταν λάβουμε την εντολή AT<CR><LF> αρκεί να στείλουμε πίσω μια απάντηση OK. Αυτό εκτελεί η SendOK στην οποία ενεργοποιούμε το Interrupt για να στείλουμε την απάντηση.

```
void AttentionInstr(){
//  just send answer
    sendOK();
```

```
void sendOK(){
//  enable UDRIE Flag
    UCSRB = (1<<UDRIE)|(1<<RXEN)|(1<<TXEN)|(1<<RXCIE)|(1<<TXCIE);
}
```

CLEAR Εντολή

Όταν λάβουμε την εντολή C<CR><LF> αρκεί να καθαρίσουμε την οθόνη ώστε να μη δείχνει τίποτα και να στείλουμε πίσω την απάντηση OK. Έτσι καλούμε την ρουτίνα clearseg και μετά την SendOK.

```
void ClearInstr(){
//  clear segments
    clearseg();
//  send answer
    sendOK();
}
```

```
void clearseg(){
    uint8_t i;
//  put 0x0A in every segment in display
    display_address = 0x0070;
    for (i=0;i<=7;i++){
        *display_address=0x0A;
        display_address +=1;
    }
    display_address = 0x070;
}
```

NUMBER Εντολή

Όταν λάβουμε την εντολή N239…<CR><LF> αρκεί να καθαρίσουμε την οθόνη ώστε να μη δείχνει τίποτα ύστερα να βάλουμε τα σωστά ψηφία στην οθόνη και να στείλουμε πίσω την απάντηση OK. Καθαρίζουμε , περνάμε τα δεδομένα στην display_Address και στέλνουμε την απάντηση όπως προηγούμενα.

```
void NumberInstr(){
    clearseg();
//  go to end of display address
    InstrAddress = 0x009B;
//  go back until you find CR
    while (*InstrAddress != ASCII_CR){
        InstrAddress -=1;
    }
//  go back one more
    InstrAddress -=1;
//  put number from instruction to display in the opposite order
//  left values in display are the smallest numbers in display
    while (*InstrAddress != ASCII_N){
        *display_address = *InstrAddress;
        display_address +=1;
        InstrAddress -=1;
    }
//  return addresseses to the start
    InstrAddress = 0x0090;
    display_address = 0x0070;
//  send the answer
    sendOK();
}
```

Αυτή η ρουτίνα διαβάζει την αποθηκευμένη εντολή από το τέλος και αφού βρει που ξεκινάνε τα νούμερα που έχουμε ως ορίσματα τα αποθηκεύει 1-1 κατάλληλα στις θέσεις μνήμης ώστε να τα δείχνει σωστά η οθόνη 7seg. Δηλαδή το δεξιότερο νούμερο το αποθηκεύει στο δεξιότερο 7seg και τα υπόλοιπα αντίστοιχα.

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.

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.

Στην διαδικασία αποσφαλμάτωσης παρατηρήθηκε επίσης ότι κατά τη διάρκεια αποστολής συμβόλου που χρησιμοποιούμε το Flag UDRE για την ενεργοποίηση του Interrupt αυτό το Flag όπως και το RXC χρειάζεται να το προσπελάσουμε και να το γράψουμε 2 φορές και να ανταποκριθεί το USART καθώς και ότι αφού σταλούν δεδομένα χρειάζεται περίπου 180.000 κύκλους για να σηκωθεί το επόμενο Flag και να καλεστεί το interrupt. Αυτό σύμφωνα με τους υπολογισμούς που λαμβάνουμε υπόψιν το Baud Rate θα έπρεπε να συμβαίνει κάθε 10.000 κύκλους χοντρικά. Πιθανόν να είναι σφάλμα που οφείλεται στο Atmel Studio είτε στην τρόπο που θεωρεί το Stim File ρολόι επειδή αυτή η διαφορά μιας τάξης μεγέθους θα μπορούσε να δικαιολογηθεί αν το ρολόι είχε τιμή κοντά στο 1 MHz.