1. Περιβάλλον εργασίας – παραδείγματα σύνταξης

Θα χρειαστούμε τα προγράμματα **Atmel Studio 7.0** (http://www.atmel.com/) και **khazama AVR Programmer** (http://khazama.com/project/programmer/). Τα προγράμματα διατίθενται ελεύθερα στο internet για κατέβασμα και εγκατάσταση. Λειτουργούν κανονικά στα Windows 10 και στις παλαιότερες εκδόσεις Windows 8 και 7.

Ανοίγουμε το Atmel Studio και επιλέγουμε **New project**. Στην καρτέλα με τίτλο **New project** που ανοίγει επιλέγουμε **GCC C Executable Project** και στο πεδίο **Name:** πληκτρολογούμε το όνομα που επιθυμούμε για την εργασία. Είναι χρήσιμο να έχουμε επιλέξει **Create directory for solution** ώστε η εργασία μας να αποθηκευτεί σε νέο φάκελο με όνομα ίδιο με την εργασία που θα δημιουργηθεί στην τοποθεσία που έχουμε επιλέξει. Εξ ορισμού οι εργασίες αποθηκεύονται στη θέση:

C:\Users*(user_name)*\Documents\Atmel Studio\7.0

Στη συνέχεια πατάμε **OK**.

Στη νέα καρτέλα με τίτλο **Device Selection** που ανοίγει επιλέγουμε τον μικροελεγκτή και πατάμε **OK** (στο εργαστήριο ψηφιακών κυκλωμάτων χρησιμοποιούμε τον **ATmega16A**). Η επιλογή μικροελεγκτή είναι υποχρεωτική γιατί δεν υποστηρίζονται όλες οι εντολές από όλους τους μικροελεγκτές, μπορεί όμως να γίνει αλλαγή στην πορεία εφόσον οι εντολές που θα χρησιμοποιήσουμε υποστηρίζονται. (Η αλλαγή μπορεί να γίνει από το εικονίδιο που συμβολίζει ένα ολοκληρωμένο και αναγράφει το όνομα του μικροελεγκτή που έχουμε επιλέξει και βρίσκεται στο κέντρο της δεύτερης γραμμής εργαλείων στο πάνω μέρος του παραθύρου.)

Στο περιβάλλον εργασίας στη συνέχεια ανοίγει ένας editor με τίτλο **main.c** όπου γίνεται η σύνταξη του κώδικα σε γλώσσα προγραμματισμού **Embedded C for AVR** και ισχύουν γενικά οι κανόνες της **C**. Σε συντομία μπορούμε να αναφέρουμε τις βασικές αρχές. Στην αρχή υπάρχει μια περιοχή για σχόλια (τα σχόλια εμφανίζονται με πράσινο χρώμα). Ακολουθεί η εντολή `#include <avr/io.h>` η οποία φορτώνει ένα header file που είναι απαραίτητο γιατί περιέχει όλα τα στοιχεία για τους μικροελεγκτές που χρειάζεται ο compiler. Στο επόμενο πεδίο βρίσκεται ο κυρίως κώδικας. Ξεκινάει με την εντολή `int main(void)` και κλείνεται μέσα σε αγκύλες, τις οποίες για ευκολία τις τοποθετούμε στο ίδιο κατακόρυφο επίπεδο. Κατά τη σύνταξη θα χρησιμοποιήσουμε αρκετές φορές εντολές που το περιεχόμενό τους κλείνεται σε αγκύλες. Είναι βολικό η αγκύλη έναρξης να είναι στο ίδιο κατακόρυφο επίπεδο με την αγκύλη λήξης και κάθε εσωτερική σε αυτές εντολή, η οποία περιέχει επίσης αγκύλες, να είναι ένα

περιθώριο προς τα δεξιά σε δικό της κατακόρυφο επίπεδο. Η μορφή αυτή μας διευκολύνει να αναγνωρίζουμε πιο εύκολα τα loops, ειδικά όταν ο κώδικας αρχίζει να γίνεται πολύπλοκος και λαμβάνει μεγάλη έκταση.

Αρχικά γράφονται οι εντολές που εκτελούνται μια φορά καθώς ξεκινάει το πρόγραμμα και στη συνέχεια μέσα στις αγκύλες της εντολής **`while(1)`** οι εντολές που εκτελούνται συνεχώς. Στο τέλος κάθε εντολής, εκτός από τις **`#include`** και **`#define`**, πρέπει να υπάρχει το σύμβολο **`;`** (semicolon, σύμβολο του ελληνικού ερωτηματικού)[1] το οποίο δηλώνει το τέλος της εντολής ή της συνθήκης μέσα στην εντολή. Για τις εντολές **`for`**, **`switch`**, **`if`** και **`while`** υπάρχει μια συνθήκη μέσα σε παρενθέσεις. Όταν υπάρχει **`;`** (semicolon) στο τέλος της εντολής μετά το κλείσιμο της παρένθεσης, εκτελείται το περιεχόμενο της παρένθεσης για όσο ισχύει. Όταν δεν υπάρχει, αλλά κάτω από την εντολή υπάρχει μια άλλη εντολή, εκτελείτε αυτή για όσο ισχύει το περιεχόμενο της παρένθεσης, ενώ αν υπάρχουν αγκύλες εκτελείται το περιεχόμενο αυτών. Υπάρχουν εντολές για μεταβλητές, σταθερές, δηλώσεις, ανάθεση τιμών, αριθμητικές ή λογικές πράξεις (κάνοντας χρήση συμβόλων για αριθμητικούς ή λογικούς τελεστές), συναρτήσεις, κλήσεις ρουτινών. Επίσης υπάρχει ένα σύνολο εντολών που αφορά αποκλειστικά λειτουργίες και καταχωρητές των μικροελεγκτών, το οποίο μπορεί να διαφοροποιηθεί ανάλογα με τον τύπο του μικροελεγκτή. Η εμφάνιση των αριθμών σε όλες τις περιπτώσεις σύνταξης μπορεί να είναι σε δυαδική (**`0b10110011`**), δεκαεξαδική (**`0xB3`**) ή δεκαδική μορφή (**`179`**), ενώ όταν πρόκειται για χαρακτήρα ASCII, ο χαρακτήρας γράφεται μέσα τόνους (**`'n'`**) ο οποίος αντιστοιχεί στην αριθμητική τιμή (**`0x6E`**). Παρακάτω μπορούμε να εξετάσουμε ορισμένα παραδείγματα.

```
#include <avr/io.h>                //AVR IO
#include <util/delay.h>            //Delay routine header file
#define F_CPU 4000000UL            //MCU Frequency, Definition for Delay routine
#define En 0b00001000              //Definition example
unsigned char i3;                  //Variable example
DDRA = 0b00000000;                 //Port A all inputs
```

[1] *Στην κωδικοποίηση χαρακτήρων ASCII, για το σύμβολο **;** επιστρέφει ο ίδιος κωδικός (0x3B) είτε προέρχεται από ελληνικό πληκτρολόγιο ως ερωτηματικό, είτε από αγγλικό ως άνω τελεία (semicolon), κάτι που όμως δεν ισχύει για όλες τις κωδικοποιήσεις χαρακτήρων. Στην κωδικοποίηση χαρακτήρων UTF-8 για παράδειγμα, επιστρέφει ο κωδικός (0x003B) όταν προέρχεται το σύμβολο **;** από αγγλικό πληκτρολόγιο ως άνω τελεία (semicolon), ενώ επιστρέφει ο κωδικός (0xCDBE) όταν προέρχεται από ελληνικό πληκτρολόγιο ως ερωτηματικό. Στα Windows 10 και το Atmel Studio 7.0 επιστρέφει ASCII χαρακτήρας για τον compiler με αποτέλεσμα να μη δημιουργείται πρόβλημα. Υπάρχει περίπτωση όμως σε διαφορετικά λειτουργικά συστήματα ή compilers να γίνεται χρήση άλλης κωδικοποίησης χαρακτήρων και το σύμβολο **;** να επιστρέφει διαφορετικό κωδικό ανάλογα με τη γλώσσα πληκτρολογίου από όπου προέρχεται. Σε αυτήν την περίπτωση ο compiler θα βγάλει σφάλμα αν δεν προέρχεται από αγγλικό πληκτρολόγιο.*

```
PORTA = 0b11111111;              //Enable pull up resistors to all
DDRB = 0b11111111;               //Port B all outputs
PORTB = 0b11011010;              //Send 11011010 to outputs
i3 = 0x2F;                       //Sets i3 value 2F
i4 = 'n';                        //Sets i4 ASCII character n
while (PINA == 0b11101110);      //"while" with equality example
for (i1 = 0; i1 < 4; i1++)       //"for" example
_delay_ms (20);                  //delay example
PORTB = ~PORTB;                  //NOT
PORTB = PORTB << 1;              //SHIFT
PORTB = i3 & 0b11110000;         //AND
PORTB = (i3 | 0b11110000);       //OR
```

Πριν ξεκινήσουμε με τη σύνταξη του κώδικα καλό είναι να ορίσουμε τη συχνότητα λειτουργίας του μικροελεγκτή. Η συχνότητα θα χρειαστεί κατά τη μετάφραση για τη σωστή λειτουργία της εντολής ***_delay_ms*()** και ***_delay_us*()** και για τον **debugger**, εφόσον χρησιμοποιηθούν. Σε παλαιότερες εκδόσεις Atmel Studio αρκούσε μια δήλωση **#define F_CPU 8000000UL** για συχνότητα 8MHz, όμως σε αυτή την έκδοση χρειάζονται και κάποια επιπλέον βήματα. Στο κέντρο της δεύτερης γραμμής εργαλείων στο πάνω μέρος του παραθύρου δεξιά από το εικονίδιο του μικροελεγκτή υπάρχει ένα εικονίδιο με σύμβολο ένα σφυρί που προς στιγμήν γράφει **No Tool**. Πατάμε το εικονίδιο και ανοίγει μια καρτέλα με τίτλο το όνομα της εργασίας μας. Από το μενού αριστερά επιλέγουμε **Toolchain**, στο μενού που ανοίγει δεξιότερα επιλέγουμε **Symbols**, στο πεδίο **Defined Symbols** πατάμε το εικονίδιο **+** (στο οποίο εμφανίζεται η περιγραφή **Add Item** μόλις βάλουμε επάνω τον κέρσορα) και στο παράθυρο που ανοίγει πληκτρολογούμε **F_CPU=8000000UL** (για συχνότητα 8MHz) και **OK.** Στην ίδια καρτέλα, με την επιλογή **Optimization**, μπορούμε να τροποποιήσουμε τη βελτιστοποίηση του κώδικα από τον compiler η οποία αρχικά είναι σε ένα απλό βαθμό.

Ως πρώτη εφαρμογή θα ορίσουμε τις θύρες του **PORTA** εισόδους και του **PORTB** εξόδους και θα συνδέσουμε δύο τουλάχιστο διακόπτες (buttons) στις εισόδους και τα LED στις εξόδους. Τα LED ανάβουν με χαμηλή λογική στάθμη και οι διακόπτες δίνουν επίσης χαμηλή λογική στάθμη και για αυτό το λόγο θα πρέπει να ενεργοποιήσουμε τις αντιστάσεις πρόσδεσης (pull up) στις εισόδους. Στον κώδικα που ακολουθεί τα LED αναβοσβήνουν κάθε μισό δευτερόλεπτο και αν πατηθεί ο ένας διακόπτης παραμένουν σβηστά για όση ώρα είναι πατημένος.

```
/*
 * GccApplication2.c
 *
 * Created: 17/10/2016 9:46:56 pm
 * Author : Dimitrios Porlidas
 */

#include <avr/io.h>
#include <util/delay.h>                    //Delay routine header file

int main(void)
{
       _delay_ms(100);                     //Delay for start up (voltage stabil.)
       DDRA = 0b00000000;                  //Port A all inputs
       PORTA = 0b11111111;                 //Enable pull up resistors to all
       DDRB = 0b11111111;                  //Port B all outputs
       PORTB = 0b11111111;                 //LEDs off, initial state
    while (1)
    {
              PORTB = ~PORTB;              //Toggle outputs
              _delay_ms(500);              //Delay 500ms
              while(PINA == 0b11111110)    //Button pressed
              PORTB = 0b11111111;          //LEDs off
    }
}
```

Αφού ολοκληρώσουμε με τη σύνταξη του κώδικα προχωράμε στη μετάφραση, ώστε να δημιουργηθεί στη θέση:

C:\Users*(user_name)*\Documents\Atmel Studio\7.0*(project_name)*\Debug

το αρχείο που θα κατεβάσουμε στον μικροελεγκτή για τον προγραμματισμό του, με όνομα ίδιο με το όνομα της εργασίας μας και κατάληξη .hex. Από το αρχικό μενού επιλέγουμε **Build** και στη συνέχεια **Build Solution** ή πατάμε το εικονίδιο στο κέντρο της δεύτερης γραμμής εργαλείων που έχει το ίδιο σύμβολο. Αν η μετάφραση ολοκληρωθεί με επιτυχία, στο κάτω μέρος του περιβάλλοντος εργασίας μας όπου υπάρχει ένα παράθυρο με τίτλο **Output**, θα εμφανιστεί το μήνυμα:

```
Build succeeded.
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

Αν μετακινήσουμε προς τα πάνω τη γραμμή κύλισης δεξιά του παραθύρου θα δούμε διάφορες πληροφορίες όπως το μέγεθος του παραγόμενου αρχείο, το ποσοστό της μνήμης Flash που θα καταλάβει, τη μνήμη Ram που απαιτείται κλπ. Αν δεν μπορέσει να ολοκληρωθεί με επιτυχία η μετάφραση, στην ίδια θέση θα εμφανιστεί το παράθυρο με τίτλο **Error List** όπου θα είναι γραμμένη η αιτία της αποτυχίας και τα τυχόν σφάλματα που υπάρχουν στον κώδικα. Διορθώνουμε τα λάθη που έχουν εντοπιστεί και επαναλαμβάνουμε τη μετάφραση.

Ο προγραμματισμός του μικροελεγκτή μπορεί να γίνει με πολλούς τρόπους. Στο παράδειγμα θα χρησιμοποιήσουμε τον προγραμματιστή **USBasp** και το πρόγραμμα **khazama AVR Programmer**. Συνδέουμε τον προγραμματιστή σε μια θύρα USB και ανοίγουμε το πρόγραμμα. Στο κέντρο του προγράμματος υπάρχει ένα πεδίο με τίτλο **AVR** όπου επιλέγουμε τον μικροελεγκτή. Αν δεν ανοίγει η λίστα των μικροελεγκτών από το βελάκι του πεδίου δεν έχει γίνει σωστή εγκατάσταση των drivers του προγραμματιστή. Στη σελίδα: http://porlidas.gr/InstDev/InstDevGr.htm παρουσιάζονται όλες οι απαραίτητες πληροφορίες για διόρθωση του προβλήματος. Αφού επιλέξουμε τον μικροελεγκτή πατάμε το σύμβολο της κλειδαριάς στο δεξιό μέρος. Στο παράθυρο που ανοίγει με τίτλο **Fuses and Lock Bits settings** μπορούμε να δούμε και να προγραμματίσουμε τα Fuses και Lock bits του μικροελεγκτή. Πατάμε **Read All** στο κάτω δεξιό μέρος και αν η επικοινωνία με τον μικροελεγκτή είναι σωστή, το πρόγραμμα διαβάζει τις πληροφορίες του. Αν δεν έχουμε σκοπό να χρησιμοποιήσουμε το **JTAG Interface** καλό είναι να το απενεργοποιήσουμε τσεκάροντας το δεύτερο τετραγωνάκι στη σειρά **H-Fuse.** Μετακινώντας προς τα κάτω τη γραμμή κύλισης στα δεξιά του παραθύρου, υπάρχει στο τέλος η ρύθμιση για το χρονισμό του μικροελεγκτή. Αν γίνεται με εξωτερικό κρύσταλλο μεγαλύτερο του 1MHz επιλέγουμε: **Ext. Crystal/Resonator High Freq.; Start-up time: 16K CK + 64 ms;** Αφού ολοκληρώσουμε τις ρυθμίσεις πατάμε **Write All** και οι ρυθμίσεις φορτώνονται στον μικροελεγκτή. Κλείνουμε το παράθυρο των Fuses και επιστρέφουμε στο αρχικό παράθυρο του προγράμματος. Πατάμε στο εικονίδιο με το σύμβολο του φακέλου και το γράμμα **F** και ανοίγει ένας browser ώστε να αναζητήσουμε και να φορτώσουμε το αρχείο .hex που ετοιμάσαμε και στη συνέχεια πατάμε **Auto Program**, στο δεξιό μέρος του παραθύρου, ώστε να κατεβεί στον μικροελεγκτή. Αν κατεβεί κανονικά θα διαβάσουμε ένα μήνυμα επιτυχίας και ο κώδικάς θα αρχίσει να εκτελείται αμέσως στον μικροελεγκτή.