



Τμήμα Μηχανικών Η/Υ και Πληροφορικής
Computer Engineering and Informatics Department (CEID)
www.ceid.upatras.gr

Εργαστήριο Προηγμένοι Μικροεπεξεργαστές

Εργαστηριακές Ασκήσεις

Πάτρα, 2024-25

1 Εισαγωγικά

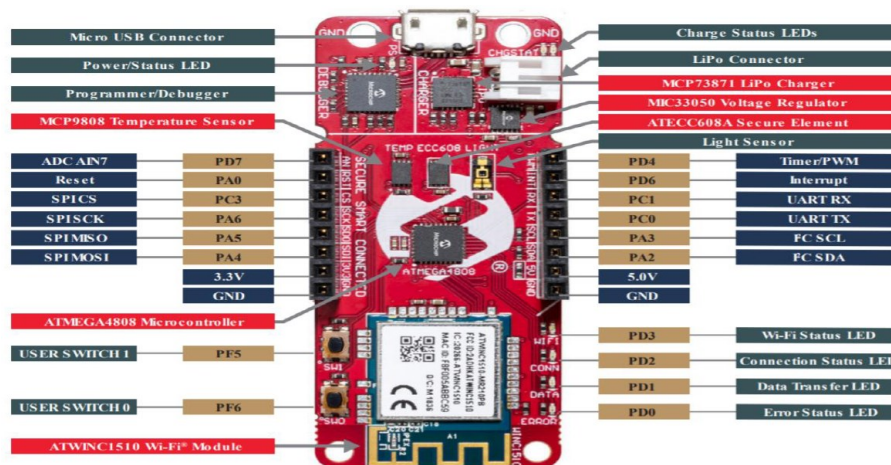
Η AVR είναι μια οικογένεια μικροεπεξεργαστών που αναπτύχθηκε αρχικά από την Atmel και μετέπειτα αγοράστηκε από την Microchip Technology. Βασίζεται σε μια τροποποιημένη Harvard αρχιτεκτονική, 8-bit RISC single-chip μικροεπεξεργαστή. Η AVR είναι μια από τις πρώτες οικογένειες μικροεπεξεργαστών που χρησιμοποίησαν on-chip flash memory, για αποθήκευση προγραμμάτων, σε αντίθεση με μια προγραμματιζόμενη ROM, EPROM ή EEPROM που χρησιμοποιούνταν από άλλους μικροελεγκτές. Οι μικροεπεξεργαστές της οικογένειας αυτής χρησιμοποιούνται σε πολλές εφαρμογές, ως πλήρη ενσωματωμένα συστήματα. Αξιοποιούνται ως εκπαιδευτικές ενσωματωμένες διατάξεις και έγιναν διάσημοι λόγω της ευρείας χρήσης τους σε πολλά open hardware development boards, της σειράς Arduino.

Το Microchip Studio (ή Atmel® Studio 7) είναι μια ολοκληρωμένη πλατφόρμα ανάπτυξης (Integrated Development Platform - IDP), για την δημιουργία εφαρμογών με AVR και SAM microcontrollers (MCU). Η πλατφόρμα προσφέρει εύχρηστο περιβάλλον για τη σύνταξη, την κατασκευή και τον εντοπισμό σφαλμάτων εφαρμογών που είναι γραμμένα σε C/C++ ή/και σε Assembly. Επίσης, παρέχεται η δυνατότητα εισαγωγής σχεδίων σε Arduino ως C/C++ projects και υποστηρίζονται 500+ AVR και SAM διατάξεις. Τέλος, το Microchip Studio περιλαμβάνει μια τεράστια βιβλιοθήκη πηγαίου κώδικα με 1600+ παραδείγματα εφαρμογών. Για περισσότερες πληροφορίες επισκεφτείτε την σελίδα της Microchip, στην οποία καταγράφονται αναλυτικά όλες οι δυνατότητες του Studio.

2 Το AVR-IoT Wx Development Board

Το AVR-IoT Wx Development Board είναι μια ευέλικτη και εύκολα επεκτάσιμη πλατφόρμα δημιουργίας και ανάπτυξης εφαρμογών. Βασίζεται στην αρχιτεκτονική μικροελεκτή AVR που χρησιμοποιεί τεχνολογία Wi-Fi. Συνοπτικά, μια αντίστοιχη εφαρμογή που αναπτύσσεται στο συγκεκριμένο σύστημα αποτελείται από τρία βασικά blocks:

- 1 ATmega4808 Microcontroller.
- 2 ATECC608A Secure Element.
- 3 ATWINC1510 Wi-Fi Controller Module.



Σχήμα 0.1 : AVR Development Board Overview

3 Εγκατάσταση Microchip Studio

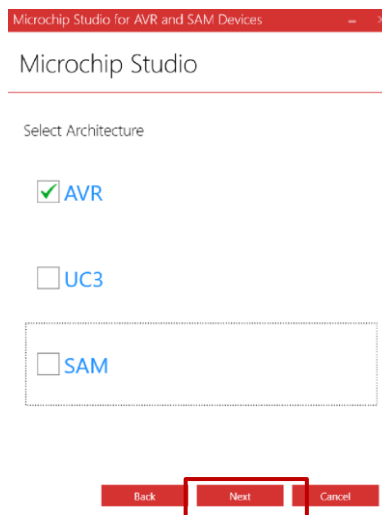
Το Microchip Studio μπορείτε να το βρείτε διαθέσιμο στον παρακάτω σύνδεσμο:

<https://www.microchip.com/mplab/microchip-studio>

Η εγκατάσταση του γίνεται εύκολα, ακολουθώντας τα παρακάτω βήματα:

- 1 Πατήστε Download στην επιλογή Microchip Studio for AVR and SAM Devices 7.0.2594 Web Installer.
- 2 Αφού ολοκληρωθεί η παραπάνω διαδικασία, ανοίξετε το αρχείο και περιμένετε.
 - Στο παράθυρο που θα εμφανιστεί επιλέξτε το path στο οποίο θα αποθηκευτεί το Studio και επιλέξτε το κουτί "I agree to the Terms and Conditions".
 - Στη συνέχεια επιλέξτε την επιλογή "Next".
- 3 Στο επόμενο παράθυρο, έχετε την επιλογή να εγκαταστήσετε τρεις διαφορετικές οικογένειες που υποστηρίζονται από το Studio:
 - Το πρώτο είναι το AVR, το οποίο θα πρέπει να παραμείνει επιλεγμένο καθώς με αυτό θα ασχοληθούμε.
 - Το UC3 και το SAM αποτελούν άλλα είδη αρχιτεκτονικών και δεν είναι απαραίτητο να τα επιλέξετε για εγκατάσταση.

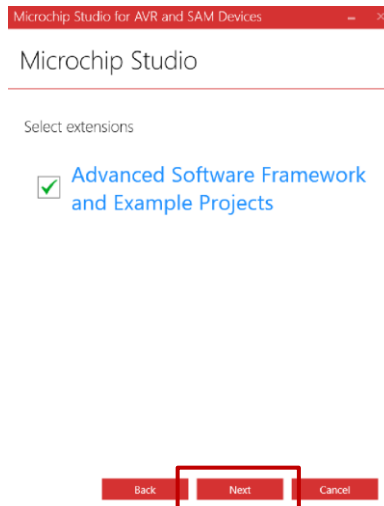
Αφού σιγουρευτείτε ότι επιλέξατε το AVR επιλέξτε "Next".



Σχήμα 0.2 : Παράθυρο Επιλογών AVR και SAM Devices

- 4 Advanced Software Framework (ASF):

Το ASF παρέχει ένα πλούσιο σύνολο λειτουργικών drivers και code modules που αναπτύχθηκαν από ειδικούς για τη μείωση του χρόνου σχεδιασμού. Το ASF είναι μια δωρεάν βιβλιοθήκη ανοιχτού κώδικα, που έχει σχεδιαστεί για να χρησιμοποιείται στις φάσεις αξιολόγησης, πρωτοτυπίας, σχεδιασμού και παραγωγής. Μπορείτε να το κρατήσετε επιλεγμένο και στον ελεύθερο χρόνο σας να δείτε τα παραδείγματα. Στη συνέχεια, μπορείτε να επιλέξετε "Next".



Σχήμα 0.3 : Παράθυρο Επιλογής ASF

- 5 Εφόσον όλες οι επιλογές του Validation είναι επιλεγμένες, είστε έτοιμοι στη συνέχεια να επιλέξετε “Next”. Στο επόμενο παράθυρο μπορείτε να επιλέξετε το “Install” και να περιμένετε να εγκατασταθεί το Microchip Studio και τα υπόλοιπα εργαλεία που χρειάζονται.



Σχήμα 0.4 : Παράθυρο Επιλογής System Validation

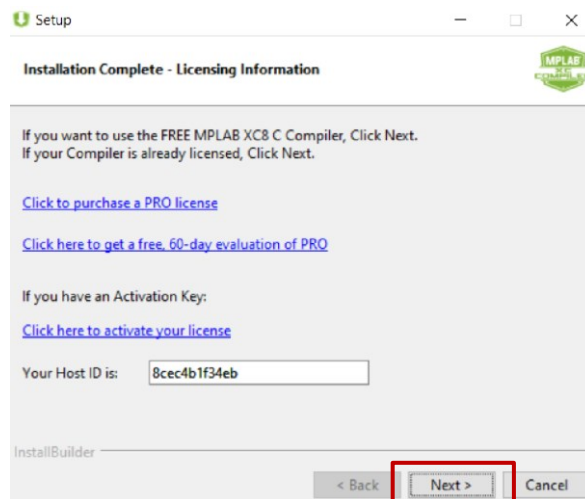
- 6 Κατά τη διάρκεια της εγκατάστασης, θα εμφανιστεί ένα επιπλέον παράθυρο επιλογών, για την εγκατάσταση του MPLAB XC8 C Compiler, με τον οποίο προσφέρεται η δυνατότητα βελτιστοποίησης του κώδικα.
- Επιλέξτε το “Next”, όπως φαίνεται στο παρακάτω σχήμα.
 - Έπειτα, επιλέξτε το «I accept the agreement» και στη συνέχεια επιλέξτε το “Next”.
 - Στα επόμενα παράθυρα επιλέξτε με τη σειρά το “Free” και μετά επιλέξτε το “Next”.
 - Μόλις τελειώσει η εγκατάσταση του, θα εμφανιστεί το αντίστοιχο παράθυρο στο οποίο μπορείτε να επιλέξετε το “Next”.
 - Στο επόμενο και τελευταίο για τον compiler παράθυρο μπορείτε να επιλέξετε το “Finish”, με το οποίο ολοκληρώνεται η εγκατάστασή του.

Εργαστήριο:

Προηγμένοι Μικροεπεξεργαστές



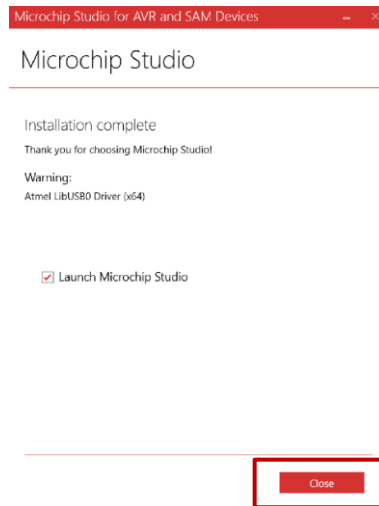
Σχήμα 0.5 : Παράθυρο Compiler Setup Wizard



Σχήμα 0.6 : Παράθυρο Licensing Information

- 7 Ένα άλλο εργαλείο που θα εγκατασταθεί είναι και το Microsoft Visual, το οποίο γίνεται αυτόματα, χωρίς να απαιτείται κάποια περαιτέρω ενέργεια.
- 8 Όταν ολοκληρωθεί η εγκατάσταση, θα εμφανιστεί το παρακάτω παράθυρο, στο οποίο μπορείτε να επιλέξετε "Close" και να ανοίξει αυτόματα το Microchip Studio.

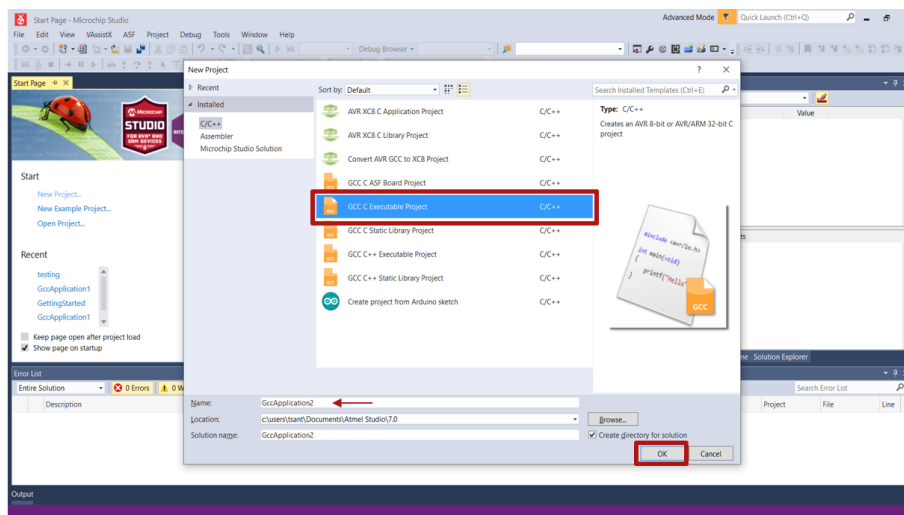
Εργαστήριο:
Προηγμένοι Μικροεπεξεργαστές



Σχήμα 0.7 : Παράθυρο Installation Complete

4 Δημιουργία Νέου Project

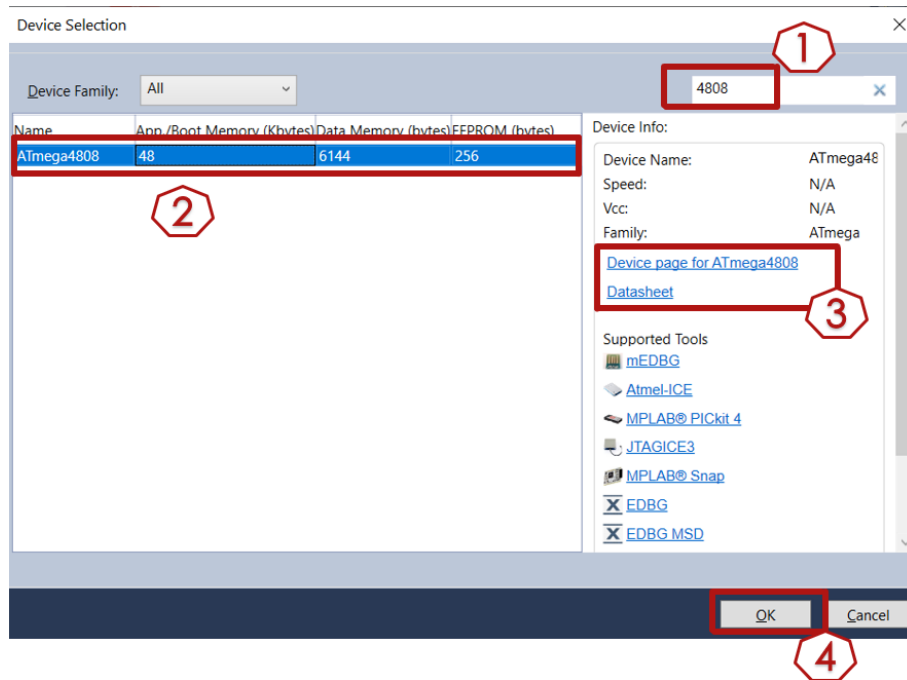
Στο εγκαταστημένο πλέον Microchip Studio, κατευθυνθείτε στην επιλογή “File → New → Project”. Τα παραδείγματα που θα υλοποιηθούν, θα είναι στη γλώσσα C και επομένως θα επιλέξουμε το “GCC C Executable Project”. Επιλέγουμε το όνομα της εφαρμογής μας και στη συνέχεια επιλέγουμε “OK”.



Σχήμα 0.8 : Παράθυρο Δημιουργίας “New Project”

Επιλέγουμε τον μικροελεγκτή “ATmega4808”, ακολουθώντας τα βήματα, που φαίνονται στην εικόνα του παρακάτω σχήματος. Μπορούμε να δούμε κατευθείαν το Datasheet και το Device Page του συγκεκριμένου μικροελεγκτή, για περισσότερες πληροφορίες σχετικά με τις δυνατότητές του (Βήμα 3). Αφού επιλέξουμε την συσκευή μας, στη συνέχεια επιλέγουμε το “OK” (Βήμα 4) και στη συνέχεια μπορούμε να αρχίσουμε να προγραμματίζουμε.

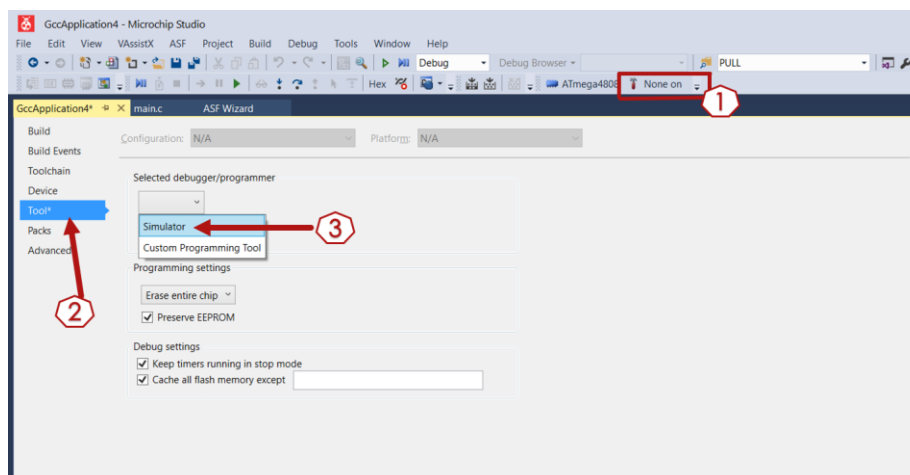
Εργαστήριο:
Προηγμένοι Μικροεπεξεργαστές



Σχήμα 0.9 : Παράθυρο Επιλογής “Device”

5 Simulation και Debugging

Είναι σημαντικό όταν κάνουμε πρώτη φορά Debug να έχουμε επιλέξει το “Simulator” ως “Debugger”, ακολουθώντας τα βήματα που φαίνονται στο παρακάτω σχήμα.



Σχήμα 0.10 : Παράθυρο Simulator – Debugger

6 Παράδειγμα 1: Λειτουργία ενός LED

Για να ενεργοποιηθεί/απενεργοποιηθεί ένα LED με κάποια συγκεκριμένη τιμή στην περίοδο, π.χ. $T=10\text{ms}$, θα πρέπει πρώτα να οριστεί το “Direction” του συγκεκριμένου Pin ως “Output”. Αρχικά, το PORT που θα χρησιμοποιηθεί είναι το PORTD καθώς οι τέσσερες πρώτοι ακροδέκτες (PINs) του συνδέονται με LEDs, πάνω στην πλακέτα (όπως φαίνεται και στο Board Overview). Για να οριστεί ένα PIN ως “Output” πρέπει να τεθεί το αντίστοιχο ψηφίο (bit) του Register DIR (Data Direction) με ‘1’. Τώρα μπορεί να δοθεί η τιμή ‘0’ ή ‘1’ στον Register Out (Output Value) και να “ενεργοποιείται” ή να “απενεργοποιείται”, το LED αντίστοιχα.

Σημείωση: Στη φάση του *Simulation* οι χρόνοι του “delay” όπως και των “timers”, (που θα δούμε σε επόμενη άσκηση) δεν είναι αντιπροσωπευτικοί του χρόνου που αναμένουμε. Μόνο με την εκτέλεση του κώδικα πάνω στην πλακέτα μπορούμε να αντιληφθούμε τους πραγματικούς χρόνους. Για το λόγο αυτό, μπορούμε να επιλέγουμε σχετικά μικρούς χρόνους, για να αποφύγουμε μεγάλες καθυστερήσεις αναμονής κατά την προσομοίωση.

Ο κώδικας της υλοποίησης του παραδείγματος παρουσιάζεται παρακάτω:

```
#include <avr/io.h>
#include <util/delay.h>
#define del 10
int main(void){

    PORTD.DIR |= 0b00000010; //PIN1_bm //PIN is output
    PORTD.OUT |= 0b00000010; //PIN1_bm //LED is off

    while (1) {

        PORTD.OUTCLR= 0b00000010; //PIN1_bm //LED is on
        _delay_ms(del);           //wait for 10ms

        PORTD.OUT |= 0b00000010; //PIN1_bm //LED is off
        _delay_ms(del);           //wait for 10ms

    }
}
```

Σχήμα 0.11 : Κώδικας Παραδείγματος 1

Μπορείτε να χρησιμοποιήσετε τον κώδικα του παραδείγματος, για να εκτελέσετε τη διαδικασία του “Simulation” στο “Microchip Studio”. Ανοίξτε το I/O παράθυρο (Debug ⇒ Windows ⇒ I/O) και ξεκινήστε το Debugging. Εκτελέστε βήμα-βήμα τις εντολές (η συνάρτηση delay δεν εκτελείται βηματικά) και παρατηρήστε τις τιμές που παίρνουν οι καταχωρητές (Registers) του PORTD.

Σημείωση: Για να δείτε τις τιμές των δηλωμένων σταθερών *PIN1_bm* και άλλων που θα δούμε στη συνέχεια, μπορείτε να ανατρέξετε στο header file *iom4808.h*, το οποίο βρίσκεται στον φάκελο που έχετε εγκαταστήσει το *Microchip Studio* :

π.χ. Path: “~\Atmel\Studio\7.0\packs\atmel\ATmega_DFP\1.7.374\include\avr\iom4808.h”

7 Παράδειγμα 2: Διακοπή (Interrupt)

Οι διακόπτες (switches) που μπορούν να χρησιμοποιηθούν βρίσκονται στο PORTF και είναι οι PIN5 και PIN6 (ανατρέξτε στο Board Overview). Για τη χρήση ενός διακόπτη (switch), το pullup enable bit που του αντιστοιχεί πρέπει να είναι ενεργοποιημένο (ανατρέξτε σελ. 158 στο ATmega4808 DataSheet). Επίσης, το σημείο του παλμού στο οποίο θα ενεργοποιηθεί η μονάδα διαχείρισης της διακοπής (interrupt) πρέπει να οριστεί. Εδώ η ενεργοποίησή της και στις δύο άκρες του παλμού επιλέγεται (ανατρέξτε σελ. 158 στο ATmega4808 DataSheet). Με την ενεργοποίηση των συγκεκριμένων ψηφίων (bits) του PIN5, το σύστημα μπορεί να δεχτεί διακοπές (interrupts) όταν πατηθεί ο διακόπτης (switch).

Όταν γίνει η διακοπή (interrupt), συγκεκριμένη συνάρτηση η οποία θα την διαχειριστεί, ενεργοποιείται. Αυτή η συνάρτηση είναι μια ρουτίνα διαχείρισης διακοπών (ISR routine) η οποία παίρνει ως όρισμα μια διεύθυνση (interrupt vector) η οποία αντιστοιχεί σε μία διακοπή. Για παράδειγμα, το interrupt vector για τη διακοπή του PORTF είναι το `PORTF_PORT_vect`. Στο αρχείο `iom4808.h` υπάρχουν όλα τα interrupt vectors που μπορούν να χρησιμοποιηθούν από το συγκεκριμένο board.

Ο κώδικας της υλοποίησης ενός παραδείγματος με διακοπή, παρουσιάζεται παρακάτω:

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

int x=0; //logic flag

int main() {
    PORTD.DIR |= 0b00000010; //PIN is output
    PORTD.OUT |= 0b00000010; //LED is off
    //pullup enable and Interrupt enabled with sense on both edges
    PORTF.PIN5CTRL |= PORT_PULLUPEN_bm | PORT_ISC_BOTHEDGES_gc;
    sei(); //enable interrupts
    while (x==0){
        PORTD.OUTCLR= 0b00000010; // LED is on
    }

    PORTD.OUT |= 0b00000010; //LED is off
    cli(); //disable interrupts
}

ISR(PORTF_PORT_vect){
    int y = PORTF.INTFLAGS; //Procedure to
    PORTF.INTFLAGS=y; //clear the interrupt flag
    x=1; //change logic flag to get out of loop
}
```

Σχήμα 0.12 : Κώδικας Παραδείγματος 2

Η λειτουργία του κώδικα του παραδείγματος 2, είναι η εξής:

Στο σύστημα υπάρχει ένα LED και ένας διακόπτης (switch). Σύμφωνα με την κανονική λειτουργία του προγράμματος, το LED ενεργοποιείται μέσα σε ένα βρόχο (while loop). Όταν πατηθεί ο διακόπτης ενεργοποιείται η διακοπή (interrupt) και αλλάζει η τιμή μιας τυχαίας μεταβλητής (int x) ώστε να σταματήσει η λειτουργία του LED και του προγράμματος.

Για να ενεργοποιηθεί η διακοπή (interrupt) το ψηφίο (bit) 5 του register INTFLAGS στο PORTF πρέπει να πατηθεί, εφόσον το πρόγραμμα είναι σε κάποιο breakpoint. Μόλις το ψηφίο (bit) αλλάξει από '0' σε '1' και το πρόγραμμα συνεχίσει με την επόμενη εντολή (STEP OVER), θα ενεργοποιηθεί η ρουτίνα διαχείρισης διακοπών (interrupt routine).

8 Παράδειγμα 3: Χρονιστής (Timer)

Για να λειτουργήσει ο χρονιστής TCA0 timer σε κανονική λειτουργία (normal mode) και να ενεργοποιηθεί διακοπή (interrupt), όταν φτάσει σε μία προβλεπόμενη τιμή θα πρέπει:

- Να δοθεί η τιμή '0' στον CTRLB register (Normal Mode).
- Να δοθεί η τιμή '0' στον CNT register (θέτουμε τον χρονιστή – timer στο μηδέν).
- Να δοθεί η προβλεπόμενη τιμή στον CMP0 register.
- Να ενεργοποιηθούν οι διακοπές (interrupts) μέσω του INTCTRL register.
- Να τεθεί η συχνότητα ρολογιού (clock frequency).
- Τέλος, να ενεργοποιηθεί ο timer μέσω του CTRLA register.

Σημείωση: Για περισσότερες λεπτομέρειες μπορείτε να ανατρέξετε στη σελ. 243, στο ATmega4808 DataSheet.

Ο TCA0 θα αρχίσει να μετράει και όταν θα φτάσει την τιμή που τέθηκε στον CMP0 θα ενεργοποιηθεί η ISR routine με όρισμα το vector TCA0_CMP0_vect. Αυτό είναι το interrupt vector που έχει οριστεί για τον συγκεκριμένο χρονιστή/μετρητή (timer/counter).

Ο TCA0 καταχωρητής μπορεί να λειτουργήσει ως χρονιστής/μετρητής είτε των 16-bit, επιλέγοντας την υποκατηγορία SINGLE, είτε των 8-bit, επιλέγοντας την υποκατηγορία SPLIT. Ο TCA0.SINGLE καταχωρητής αξιοποιεί ομάδες των δύο 8-bit καταχωρητών ως έναν ενιαίο καταχωρητή των 16-bit για να μετρήσει χρόνο. Σε αντίθεση, ο TCA0.SPLIT χωρίζεται σε δύο διαφορετικούς 8-bit χρονιστές/μετρητές, ο καθένας από τους οποίους αξιοποιεί ομάδες των 8-bit καταχωρητών, είτε τα High Bits είτε τα Low Bits. Επομένως, κάποιοι καταχωρητές του TCA0.SINGLE, όπως ο CMP0, χωρίζεται σε High και Low για τον TCA0.SPLIT, δηλαδή σε HCMP0 και LCMP0 (βλ. ATmega4808 DataSheet σελ 217-232).

Ο κώδικας της υλοποίησης του παραδείγματος, παρουσιάζεται παρακάτω:

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

#define value 20

int x=0; //logic flag

int main() {

    PORTD.DIR |= 0b00000010; //PIN is output
    PORTD.OUTCLR= 0b00000010;//LED is on

    //(σελ 219, 224, 205) 16-bit counter high and low
    TCA0.SINGLE.CNT = 0; //clear counter
    TCA0.SINGLE.CTRLB = 0; //Normal Mode (TCA_SINGLE_WGMODE_NORMAL_gc σελ 207)
    TCA0.SINGLE.CMP0 = value;//When CMP0 reaches this value -> interrupt
    //CLOCK_FREQUENCY/1024
    TCA0.SINGLE.CTRLA = 0x7<<1; //TCA_SINGLE_CLKSEL_DIV1024_gc σελ 224
    TCA0.SINGLE.CTRLA |= 1; //Enable
    TCA0.SINGLE.INTCTRL = TCA_SINGLE_CMP0_bm; //Interrupt Enable (=0x10)
    sei(); //begin accepting interrupt signals
    while (x==0) {
        ; //similar to a nop function
    }

    PORTD.OUT |= PIN1_bm; //LED is off
    cli();

}

ISR(TCA0_CMP0_vect){

    TCA0.SINGLE.CTRLA = 0; //Disable
    int intflags = TCA0.SINGLE.INTFLAGS; //Procedure to
    TCA0.SINGLE.INTFLAGS=intflags; //clear interrupt flag
    x=1; //change flag to get out of the loop

}
```

Σχήμα 0.13 : Κώδικας Παραδείγματος 3

Η λειτουργία του κώδικα του παραδείγματος 3, είναι η εξής:

Ενεργοποιείται το LED και ο χρονιστής (timer). Όταν ο χρονιστής φτάσει την τιμή που έχει τεθεί αρχικά, τότε:

- Ενεργοποιείται η ρουτίνα διαχείρισης της διακοπής.
- Αλλάζει μια τυχαία μεταβλητή (x), για να μπορεί να ελεγχθεί μέσα στην συνάρτηση main αν εκτελέστηκε η ρουτίνα διαχείρισης της διακοπής, ώστε το πρόγραμμα να συνεχίσει με την επόμενη λειτουργία.
- Το πρόγραμμα ολοκληρώνεται απενεργοποιώντας το LED.

Σημείωση: Μπορείτε να χρησιμοποιείτε breakpoints για να δείτε εάν ένα interrupt ενεργοποιείται.

Η τιμή του καταχωρητή CMPO (value) ορίζει το χρονικό διάστημα μέχρι να κάνει διακοπή (interrupt) ο χρονιστής (timer), (γενικά να τελειώσει και να αρχίσει από την αρχή). Ο τύπος για τον υπολογισμό είναι ο παρακάτω:

$$f_{timer} = \frac{f_{system}}{N_{prescaler}}$$

$$value = T * f_{timer}$$

Ο ATmega4808 λειτουργεί σε μέγιστο F=20 MHz και το $N_{prescaler}$ παίρνει την τιμή που έχουμε ορίσει εμείς για συχνότητα ρολογιού (clock frequency). Για παράδειγμα στον κώδικά μας δώσαμε τιμή value=20 άρα ο χρόνος του χρονιστή (timer) είναι:

$$f_{timer} = \frac{20MHz}{1024} = 19,531KHz$$

$$T = \frac{value}{f_{timer}} = 1,024ms$$

9 Παράδειγμα 4: Λειτουργία του Μετατροπέα Αναλογικού σε Ψηφιακό (Analog to Digital Converter – ADC)

Ο Μετατροπέας Αναλογικού σε Ψηφιακό σήμα (Analog to Digital Converter – ADC) είναι ένα σύστημα το οποίο μετατρέπει αναλογικά σήματα που έρχονται ως είσοδοι στα κατάλληλα PINs, όπως ήχος, ρεύμα, φως, κτλ., σε ψηφιακά σήματα. Επομένως, οι τιμές των αναλογικών σημάτων μπορούν να διαβαστούν πλέον από τον μικροελεγκτή και να αξιοποιηθούν κατάλληλα. Ο ADC που εμπεριέχεται στην πλακέτα δέχεται μετρήσεις από το PIN7 του PORTD. Οι βασικοί τρόποι (modes) λειτουργίας του ADC είναι δύο. Το free-running mode στο οποίο ο ADC δέχεται συνεχώς τιμές και τις μετατρέπει σε ψηφιακά σήματα. Η δεύτερη και προκαθορισμένη (default) λειτουργία του (single-conversion mode) εκτελεί μόνο μία μετατροπή όποτε αυτή ζητηθεί από τον κώδικα και μετά σταματά.

Στο παράδειγμα, όταν ο ADC θα διαβάζει μετρήσεις που είναι μικρότερες από μία προκαθορισμένη τιμή (για παράδειγμα RES=10) θα ενεργοποιείται μια διακοπή (interrupt). Με την ενεργοποίηση της διακοπής (interrupt) θα ανάβει ένα LED για T=5ms και μετά θα επιστρέψουμε στην αρχική ροή του προγράμματος και θα περιμένουμε ξανά την επόμενη μέτρηση του ADC.

Για την ενεργοποίηση του ADC:

- 1 Πρώτα το resolution ορίζεται σε 10-bit ή 8-bit → Resolution Selection bit (RESSEL) στον Control A register (ADCn.CTRLA).
- 2 Έπειτα, το Free-Running Mode ενεργοποιείται → '1' στο Free-Running bit (FREERUN) στο ADCn.CTRLA. Στην περίπτωση που ζητείται να εκτελεστεί η προκαθορισμένη λειτουργία του ADC (single-conversion mode), δηλαδή να εκτελεστεί μία μόνο μετατροπή όποτε ζητηθεί, αυτό το ψηφίο (bit) δεν ενεργοποιείται και παραμένει '0'.
- 3 Ορίζεται το ψηφίο (bit) με το οποίο θα συνδεθεί ο ADC → MUXPOS bit field στο MUXPOS register (ADCn.MUXPOS).
- 4 Ο ADC ενεργοποιείται → Γράφουμε '1' στο ENABLE bit στο ADCn.CTRLA.
- 5 Δίνεται η επιλογή της ενεργοποίησης του Debug Mode (ADCn.DBGCTRL) ώστε ο ADC να μην σταματά ποτέ να τρέχει και να λαμβάνει τιμές.

Με τη λειτουργία του Window Comparator Mode ελέγχονται όλες οι τιμές που εισάγονται στον ADC με μια δοθείσα τιμή, δηλαδή ένα κατώφλι (threshold). Για να ενεργοποιηθεί αυτή η λειτουργία πρέπει να ακολουθηθούν τα παρακάτω βήματα:

- 1 Πρώτα εισάγεται το κατώφλι (threshold) στον καταχωρητή ADCn.WINLT και/ή ADCn.WINHT.
- 2 Ενεργοποιείται η λειτουργία δημιουργίας διακοπών (interrupts) → Window Comparator Interrupt Enable bit (WCOMP) στον Interrupt Control register (ADCn.INTCTRL).
- 3 Ορίζεται το επιθυμητό Mode (στην περίπτωσή μας θέλουμε διακοπές – interrupt όταν RESULT<THRESHOLD) → WINCM bit field στον ADCn.CTRLE.

Εφόσον προγραμματιστεί κατάλληλα ο ADC και οι λειτουργίες του, απομένει να γραφτεί το λογικό '1' στο Start Conversion Bit (STCONV) στον Command Register (ADCn.COMMAND), το οποίο εκκινεί το conversion. Οι τιμές καταγράφονται στον καταχωρητή RES (Result). Αυτή η εντολή πρέπει να εκτελεστεί μόνο εφόσον προγραμματιστεί εξ' ολοκλήρου ο ADC με τους παραπάνω τρόπους που εξηγήθηκαν. Στο Free-Running Mode αρκεί να ενεργοποιηθεί μία φορά αυτή η εντολή ώστε συνεχώς να γίνονται μετατροπές νέων τιμών από αναλογικό σε ψηφιακό. Στην προκαθορισμένη λειτουργία (single-conversion mode) κάθε φορά που απαιτείται μία μετατροπή νέων τιμών, πρέπει να εκτελείται η εντολή Start Conversion ώστε ο ADC να μετατρέψει το καινούριο σήμα εισόδου από αναλογικό σε ψηφιακό.

Εργαστήριο:

Προηγμένοι Μικροεπεξεργαστές

Συμβουλή: Ανατρέξτε στο *ATmega4808 DataSheet* και διαβάστε καλά τις σελίδες 394-421. Αναγράφονται αναλυτικά όλες οι πιθανές λειτουργίες που μπορείτε να χρησιμοποιήσετε για να κάνετε *Analog to Digital Conversion* και πως να τις προγραμματίσετε.

Ο κώδικας της υλοποίησης του παραδείγματος παρουσιάζεται παρακάτω:

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

int main(){

    PORTD.DIR |= PIN1_bm;                //PIN is output

    //initialize the ADC for Free-Running mode
    ADC0.CTRLA |= ADC_RESSEL_10BIT_gc;   //10-bit resolution
    ADC0.CTRLA |= ADC_FREERUN_bm;        //Free-Running mode enabled
    ADC0.CTRLA |= ADC_ENABLE_bm;         //Enable ADC
    ADC0.MUXPOS |= ADC_MUXPOS_AIN7_gc;   //The bit
    ADC0.DBGCTRL |= ADC_DBGRUN_bm;       //Enable Debug Mode

    //Window Comparator Mode
    ADC0.WINLT |= 10;                    //Set threshold
    ADC0.INTCTRL |= ADC_WCMP_bm;          //Enable Interrupts for WCM
    ADC0.CTRLE |= ADC_WINCM0_bm;         //Interrupt when RESULT < WINLT
    sei();
    ADC0.COMMAND |= ADC_STCONV_bm;        //Start Conversion
    while(1){

        ;

    }

}

ISR(ADC0_WCOMP_vect){
    int intflags = ADC0.INTFLAGS;
    ADC0.INTFLAGS = intflags;
    PORTD.OUTCLR= PIN1_bm;                //LED is on
    _delay_ms(5);
    PORTD.OUT |= PIN1_bm;                 //LED is off
}
```

Σχήμα 0.14 : Κώδικας Παραδείγματος 4

Σημείωση: Για την προσομοίωση της λειτουργίας του ADC πρέπει να γράφετε εσείς την τιμή εισόδου στον καταχωρητή *RES (Result)* χειροκίνητα.

10. Παράδειγμα 5: Λειτουργία του Παλμοευρικού Διαμορφωτή (Pulse-Width Modulator – PWM)

Ο Παλμοευρικός Διαμορφωτής (Pulse-Width Modulator – PWM) μπορεί να χρησιμοποιηθεί με πολλά περιφερειακά και εφαρμογές, όπως:

- Audio
- Ρύθμιση έντασης LED
- Analog signal generation
- Ρύθμιση servos, DC motors, κτλ.

Μέσω του μικροελεγκτή δημιουργείται μια κυματομορφή (waveform) η οποία συνδέεται με το PIN ενός περιφερειακού. Συγκεκριμένα, η έξοδος της κυματομορφής (waveform output) είναι διαθέσιμη στο PORT που επιλέγεται και μπορεί να χρησιμοποιηθεί ως έξοδος (αρκεί να οριστεί το PORT ως έξοδος).

Στο παράδειγμα αυτό ενεργοποιείται ο PWM ώστε να δημιουργηθεί ένας παλμός που θα λειτουργεί σαν ρολόι. Με αυτό θα αναβοσβήνει ένα LED, το οποίο θα ενεργοποιείται όταν ο παλμός ανεβαίνει στην ψηλή στάθμη (rising edge) και θα απενεργοποιείται όταν πέφτει στη χαμηλή στάθμη (falling edge). Για την εκτέλεση της PWM λειτουργία, χρησιμοποιείται ο TCA χρονιστής ως ένας 16-bit PWM. Ωστόσο υπάρχει η δυνατότητα χρήσης του και ως δύο 8-bit timers/counters, οι οποίοι δημιουργούν δύο διαφορετικά PWMs (δείτε σελ. 196 και σελ. 224 του ATmega4808 Datasheet).

Η χρήση ενός Single-Slope PWM generator θα γίνει με τη βοήθεια του χρονιστή TCA:

- 1 Αρχικά, πρέπει να οριστεί ο prescaler του χρονιστή (timer), όπως και στην περίπτωση του απλού timer → `TCA0.SINGLE.CTRLA=TCA_SINGLE_CLKSEL_DIV1024_gc`.
- 2 Έπειτα, πρέπει να δοθεί η μέγιστη τιμή TOP μέχρι την οποία θα μετρήσει ο χρονιστής (timer) → `TCA0.SINGLE.PER = value`.
- 3 Ορίζεται ο κύκλος λειτουργίας (duty cycle) του παλμού μέσω της χρήσης του καταχωρητή `CMPx` → `TCA0.SINGLE.CMP0 = value`.
- 4 Γίνεται η επιλογή του Waveform Generation Mode, στην περίπτωση μας το Single-Slope → `TCA0.SINGLE.CTRLB |= TCA_SINGLE_WGMODE_SINGLESLOPE_gc`.
- 5 Τέλος, ενεργοποιείται ο TCA → `TCA0.SINGLE.CTRLA |= TCA_SINGLE_ENABLE_bm`.

Με την επιλογή του prescaler και την εκχώρηση τιμής στον καταχωρητή PER, η συχνότητα υπολογίζεται με τον παρακάτω τύπο.

$$f_{PWM_SS} = \frac{f_{CLK_PER}}{N(PER + 1)}$$

Ο καταχωρητής PER είναι 16-bit, επομένως το ελάχιστο resolution είναι `TCA0.SINGLE.PER = 0x0002` και το μέγιστο είναι `TCA0.SINGLE.PER = MAX-1`. Η τιμή του `CMPx` καθορίζει το duty cycle. Για παράδειγμα, αν έχει την μισή τιμή του καταχωρητή PER τότε το duty cycle είναι 50%.

Σημείωση: Υπάρχουν και άλλες λειτουργίες του PWM. Για περισσότερες λεπτομέρειες ανατρέξτε στο ATmega4808 DataSheet σελ.191-199.

Εργαστήριο:

Προηγμένοι Μικροεπεξεργαστές

Όταν ανεβαίνει στην υψηλή στάθμη και όταν κατεβαίνει στην χαμηλή στάθμη, ενεργοποιούνται οι κατάλληλες διακοπές (interrupts). Η λειτουργία αυτή ορίζεται ως εξής:

- Ενεργοποιώντας το Overflow Interrupt: εκτελείται διακοπή όταν ο χρονιστής (timer) είναι ίσος με την BOTTOM τιμή (ανεβαίνουμε στην υψηλή στάθμη) →
TCA0.SINGLE.INTCTRL |= TCA_SINGLE_OVF_bm.
- Ενεργοποιώντας και το CMPx Interrupt: εκτελείται διακοπή όταν ο χρονιστής (timer) είναι ίσος με την τιμή του καταχωρητή CMPx (πηγαίνουμε στην χαμηλή στάθμη) →
TCA0.SINGLE.INTCTRL |= TCA_SINGLE_CMP0_bm.

Συμβουλή: Ανατρέξτε στο ATmega4808 DataSheet και διαβάστε καλά τις σελίδες σχετικά με τον TCA0. Αναγράφονται αναλυτικά όλες οι πιθανές λειτουργίες του PWM και υπάρχουν παραδείγματα για τον σχηματισμό παλμών.

Ο κώδικας της υλοποίησης του παραδείγματος παρουσιάζεται παρακάτω:

```
#include <avr/io.h>
#include <avr/interrupt.h>
int main(){
    PORTD.DIR |= PIN1_bm; //PIN is output
    //prescaler=1024
    TCA0.SINGLE.CTRLA=TCA_SINGLE_CLKSEL_DIV1024_gc;
    TCA0.SINGLE.PER = 254; //select the resolution
    TCA0.SINGLE.CMP0 = 127; //select the duty cycle
    //select Single_Slope_PWM
    TCA0.SINGLE.CTRLB |= TCA_SINGLE_WGMODE_SINGLESLOPE_gc;
    //enable interrupt Overflow
    TCA0.SINGLE.INTCTRL |= TCA_SINGLE_OVF_bm;
    //enable interrupt COMP0
    TCA0.SINGLE.INTCTRL |= TCA_SINGLE_CMP0_bm;
    TCA0.SINGLE.CTRLA |= TCA_SINGLE_ENABLE_bm; //Enable
    sei();
    while (1){
        ;
    }
}

ISR(TCA0_OVF_vect){
    //clear the interrupt flag
    int intflags = TCA0.SINGLE.INTFLAGS;
    TCA0.SINGLE.INTFLAGS = intflags;
    PORTD.OUTCLR |= PIN1_bm; //LED is on
}

ISR(TCA0_CMP0_vect){
    //clear the interrupt flag
    int intflags = TCA0.SINGLE.INTFLAGS;
    TCA0.SINGLE.INTFLAGS = intflags;
    PORTD.OUT |= PIN1_bm; //LED is off
}
```

Σχήμα 0.15 : Κώδικας Παραδείγματος 5

Εργαστηριακή Άσκηση 05:

Λειτουργία Έξυπνου Θερμοκηπίου

1 Περιγραφή

Σε αυτήν την εργαστηριακή άσκηση, υλοποιείται η λειτουργία ενός έξυπνου θερμοκηπίου. Σε αυτό το θερμοκήπιο χρησιμοποιούνται τα παρακάτω στοιχεία:

- 1 Ένας Αναλογικό σε Ψηφιακό Μετατροπέας (Analog to Digital Converter – ADC), ο οποίος δέχεται τιμές από έναν αισθητήρα υγρασίας.
- 2 Ένα στοιχείο χρονιστή (timer/counter).
- 3 Ένας Παλμοευρικός Διαμορφωτής (Pulse-Width Modulator – PWM).
- 4 Τρία LEDs (LED0, LED1, LED2 του PORTD).
- 5 Δύο διακόπτες (switches) (SW5 και SW6 του PORTF).

Αρχικά, ενεργοποιείται ο ADC, ο οποίος συνεχώς δέχεται τιμές και τις ελέγχει σύμφωνα με δύο προκαθορισμένα κατώφλια υγρασίας (thresholds). Αν οι τιμές είναι μικρότερες από το κατώτερο επιτρεπτό κατώφλι (threshold) που έχει οριστεί, τότε ενεργοποιείται το LED0 του PORTD που σηματοδοτεί πως τα φυτά του θερμοκηπίου πρέπει να ποτιστούν. Αν οι τιμές είναι μεγαλύτερες από το μέγιστο επιτρεπτό κατώφλι (threshold) που έχει οριστεί, τότε ενεργοποιείται το LED1 του PORTD που σηματοδοτεί πως η υγρασία έχει υπερβεί τα επιθυμητά επίπεδα και πρέπει να κατέβει μέσω ενός συστήματος αερισμού (αυτόματες πόρτες, παράθυρα ή σύστημα ανεμιστήρων τα οποία ενεργοποιούνται μέσω ενός μοτέρ, για την ανανέωση του αέρα του θερμοκηπίου).

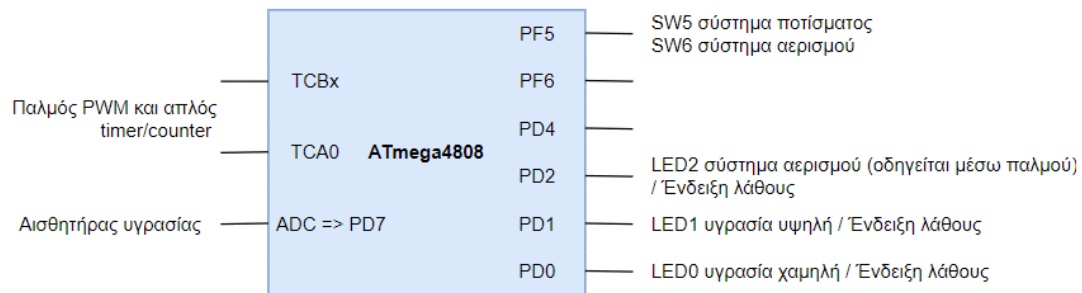
Ανάλογα με το LED που έχει ενεργοποιηθεί (LED0 ή LED1) και μέσω των δύο διακοπών (switches 5 και 6 του PORTF), δίνεται η δυνατότητα στον χρήστη να επιλέξει την επόμενη ενέργεια που θα εκτελέσει το σύστημα του θερμοκηπίου. Ενεργοποιώντας τον διακόπτη SW5 του PORTF, το σύστημα ποτίσματος ενεργοποιείται. Η λειτουργία αυτή θα προσομοιωθεί με τον χρονιστή (timer/counter), ο οποίος θα λειτουργεί για ένα χρονικό διάστημα που θα υπολογίζεται μέσω της αφαίρεσης του κατώτερου επιτρεπτού κατωφλίου της υγρασίας με την τελευταία τιμή που μετέτρεψε ο ADC. Μόλις το υπολογισμένο χρονικό διάστημα περάσει, ο χρονιστής (timer/counter) και το LED0 απενεργοποιούνται, και το σύστημα επιστρέφει στην αρχική του λειτουργία, στην οποία περιμένει την επόμενη μετατροπή του ADC. Ενεργοποιώντας τον διακόπτη SW6 του PORTF, το σύστημα αερισμού ενεργοποιείται. Η λειτουργία αυτή θα προσομοιωθεί με έναν PWM παλμό, ο οποίος θα οδηγεί το LED2 του PORTD με περίοδο $T1 = 1 \text{ ms}$ και κύκλο λειτουργίας (duty cycle) $D1 = 50\%$. Το LED2 βάσει του παλμού αυτού, θα ανάβει ή θα σβήνει αντίστοιχα σε κάθε ανερχόμενη ακμή. Ο PWM παλμός και τα ενεργοποιημένα LEDs (LED1 και LED2) θα απενεργοποιηθούν εντελώς, όταν παρέλθουν τέσσερις ανερχόμενες ακμές (rising edges) του παλμού. Μετέπειτα, το σύστημα θα επιστρέψει στην αρχική του λειτουργία, δηλαδή θα περιμένει την επόμενη μετατροπή του ADC.

Επομένως, σύμφωνα με το LED που έχει ενεργοποιηθεί, βάσει των ενδείξεων του ADC, ο χρήστης μπορεί να καταλάβει ποιον διακόπτη, δηλαδή ποια λειτουργία, πρέπει να ενεργοποιήσει. Ωστόσο, μπορεί ο χρήστης να ενεργοποιήσει το λάθος σύστημα, δηλαδή να πατηθεί άλλος διακόπτης

Εργαστήριο:

Προηγμένοι Μικροεπεξεργαστές

από αυτόν που ενδείκνυται μέσω των δύο LEDs. Σε αυτήν την περίπτωση, ενεργοποιούνται και τα τρία LEDs προειδοποιώντας τον χρήστη πως εκτελείται άλλη λειτουργία από την επιθυμητή. Τα LEDs απενεργοποιούνται με βηματική εκτέλεση.



Σχήμα 5.1 : Απεικόνιση ATmega4808, Εργαστηριακής Άσκησης 05

Παρατηρήσεις:

Πρέπει να δοθεί προσοχή στην αναγνώριση του διακόπτη που ενεργοποιήθηκε στο PORTF, με τον σωστό τρόπο ελέγχου του INTFLAGS, δηλαδή *bit masking*.

2 Ερωτήματα Εργαστηριακής Άσκησης 5

- 1 Ενεργοποιείτε την αρχική λειτουργία του συστήματος, δηλαδή τη λειτουργία του ADC με τα δύο κατώφλια υγρασίας. Ανάλογα με την συνθήκη σύγκρισης που ισχύει κάθε φορά, ενεργοποιείται το αντίστοιχο LED (LED0 ή LED1).
- 2 Προσθέστε τις δυο λειτουργίες του συστήματος, δηλαδή τη λειτουργία ποτίσματος (ενεργοποίηση χρονιστή – timer/counter) και αερισμού (ενεργοποίηση ενός PWM), που εκτελούνται ανάλογα με τον διακόπτη (SW5 ή SW6) που επιλέχθηκε από τον χρήστη.
- 3 Υλοποιείτε τη λειτουργία που ειδοποιεί τον χρήστη για λάθος εισαγωγή κουμπιού (δηλαδή ενεργοποίηση όλων των LEDs).

3 Αναφορά Εργαστηριακής Άσκησης 5

- 1 Αναπτύξτε **τον κώδικά σας** συμπεριλαμβάνοντας τα απαραίτητα σχόλια επεξήγησης του κώδικα και βεβαιωθείτε πως όλες οι λειτουργίες εκτελούνται, όπως προβλέπεται.
- 2 Παραδίδετε **διάγραμμα ροής** καθώς και **αναλυτική περιγραφή** της λειτουργίας του κώδικά σας. Εξηγήστε τον τρόπο με τον οποίο ο κώδικάς σας εκτελεί όλες τις διαδικασίες της εφαρμογής που αναλύθηκε παραπάνω, καθώς και τα πιθανά στοιχεία (interrupts, PWM, ADC κτλ.), που επιλέξατε και χρησιμοποιήσατε.

*** Σημείωση: Παραδίδετε ένα συνολικό αρχείο με τον κώδικα, τα αναλυτικά σχόλια για τις εντολές που χρησιμοποιήσατε, το διάγραμμα και την περιγραφή ΑΝΑ ΟΜΑΔΑ (μόνο ο ένας εκ των δυο της ομάδας κάνει υποβολή).

*** ΑΝΑΓΡΑΦΕΤΕ **ΑΠΑΡΑΙΤΗΤΩΣ τα ονόματα και ΑΜ των μελών της ομάδας** εντός του αρχείου που παραδίδετε.

*** Το όνομα του αρχείου να είναι: AM1_AM2_EA5.docx

Βιβλιογραφία - Αναφορές

- 1 Microchip, Επίσημη ιστοσελίδα:
<https://www.microchip.com/mplab/microchip-studio>
- 2 AVR-IoT WxHardware User Guide:
<https://ww1.microchip.com/downloads/en/DeviceDoc/AVR-IoT-Wx-Hardware-User-Guide-DS50002805C.pdf>
- 3 ATmega4808/4809 Data Sheet:
<https://ww1.microchip.com/downloads/en/DeviceDoc/ATmega4808-09-DataSheet-DS40002173B.pdf>
- 4 Microchip, “Advanced Software Framework (ASF)”:
<https://www.microchip.com/mplab/avr-support/advanced-software-framework>
- 5 Microchip, “MPLAB® XC Compilers”:
<https://www.microchip.com/en-us/development-tools-tools-and-software/mplab-xc-compilers>