Laborator 1 P.M.
Introducere in Microcontrollere

I. Ce este un microcontroller ?

Un microcontroller este un tip de circuit care integreaza un microprocesor si alte dispozitive periferice intr-un singur chip punandu-se accent pe un cost redus de productie si consum redus de energie electrica. Principala diferenta dintre un microcontroller (µC) si un microprocesor (µP) o constituie faptul ca un µC integreaza memoria de program, memoria de date si alte interfete de intrare-iesire sau periferice.

Din cauza integrarii unui numar mare de periferice si a costului redus de productie,un µC opereaza la viteze reduse, frecventa sa de ceas fiind de obicei de zeci sau sute de MHz, cu un ordin de marime mai mica decat cea a unui µP actual. Cu toate acestea, microcontrollerele se preteaza la o gama variata de aplicatii fiind folosite atat in mediul industrial cat si in produse de larg consum, de la sisteme din industria aerospatiala pana la telefoane mobile, cuptoare cu microunde si jucarii.

Majoritatea sistemelor de calcul intalnite in ziua de azi au “migrat” din suportul unui calculator obisnuit si au fost integrate in alte dispozitive cum ar fi automobilele sau produsele electronice formand ceea ce se cheama un sistem embedded.

Un sistem embedded poate veni intr-un “pachet” foarte diferit fata de un calculator obisnuit. De exemplu, acesta poate sa nu aiba un display sau o tastatura, intrarile si iesirile fiind simple comutatoare si led-uri. Un astfel de sistem poate avea cerinte minime de performanta sau capacitate de memorie si viteza de executie. Un microcontroller dintr-un sistem embedded poate sa faca o gama variata de operatii, de la controlul unor motoare electrice pana la comunicatia cu alte device-uri asemanatoare intr-o retea wireless.

Majoritatea µC nu au un bus extern de adrese sau date deoarece toate memoriile de date sunt interne. Acest lucru duce la integrarea acestora in capsule cu un numar mic de pini si reduse ca dimensiuni, ceea ce duce in schimb la micsorarea costulrilor de productie.

Cele mai intalnite structure din circuitul integrat al unui µC sunt urmatoarele:

- Unitatea centrala de procesare (µP core) cu o arhitectura care poate fi pe 8, 16, 32 sau 64 de biti.
- Memorie de date volatila (RAM) sau nevolatila pentru date sau program (Flash sau EEPROM)
- Porturi digitale de intrare-iesire
- Interfete seriale (RS232, SPI, I2C, CAN, RS485)
- Timere, generatoare de PWM sau watchdog
- Convertoare analog-digitale
- Suport pentru programare si debugging (ISP)

Unele microcontrollere au o arhitectura Harvard, cu magistrale separate de date si instructiuni, permitand astfel un acces concurent. In cazul folosirii acestei arhitecturi, instructiunile pot avea o lungime diferita fata de registrele si memoria interna. De exemplu, pentru familia AVR de la Atmel instructiunile au lungimea de 16 biti iar registrele interne pe sunt de 8 biti.

Costul unui µC depinde in mare masura de numarul de periferice integrate. Cu cat numarul acestora este mai mare cu atat nivelul de integrare creste ducand la un cost de productie mai mare. Din aceasta cauza, arhitecturile de µC pe piata la ora actuala variaza in limite destul de largi, de la chipuri cu doar 6 pini de I/O pana la procesoare digitale de semnal (DSP) sau procesoare cu arhitecturi ARM.

La momentul actual sunt cateva zeci de firme producatoare in lume ce ofera o gama variata de µC din punct de vedere al arhitecturii, dotarilor sau vitezei de executie. Principalele concurente in piata de 8 biti sunt companiile Atmel si Microchip, cu famiile de µC AVR, respectiv PIC.

II. Atmel AVR

Famila AVR de la Atmel este formata din microcontrollere cu arhitectura Harvard pe 8 biti si set redus de instructiuni (RISC). Arhitectura de baza AVR a fost conceputa de doi studenti de la Norwegian Institute of Technology (NTH) Alf-Egil Bogen and Vegard Wollan.

Ele au fost introduce pe piata in 1996, fiind printer primele controllere care foloseau memoria Flash pentru program in locul memoriilor OTPROM sau EPROM folosite de competitie.

AVR-urile sunt clasificate in patru mari categorii:

1. **tinyAVR**
   - 1-8 kB memorie de program
   - capsula de 8 pana la 32 pini
   - set limitat de periferice
2. **megaAVR**
   - 4-256 kB memorie de program
   - capsula de 28 pana la 100 de pini
   - set extins de instructiuni (instructiuni pentru inmultire si adresare indirecta)
   - set extins de periferice
3. **XMEGA**
   - 16-256 kB memorie de program
   - capsula de 44 pana la 100 de pini
   - interfete performante extinse, ca DMA, "Event System", si support pentru criptografie
   - set extins se periferice
4. **Application specific AVR**
   - megaAVR cu functii speciale, care nu sunt prezente la familia AVR, cum ar fi controller de LCD , controller USB , CAN etc.
   - FPSLIC (Field Programmable System Level Integrated Circuit), un core AVR integrat cu un FPGA.

Memoriile Flash, EEPROM, si SRAM sunt integrate in acelasi chip, inlaturand nevoia de memorie externa. Programul este format din instructiuni de 16 biti lungime care sunt stocate in memoria Flash nevolatila.

Marimea memoriei de program este indicate de numele componentei respective. De exemplu, ATmega128 are 128kB de memorie Flash. Spatiul de adresa consta din registrele generale, registrele de I/O si memoria SRAM. Sunt in total 32 de registre generale de cate 8 biti.

AVR au o unitate de executie in banda de asamblare cu trei niveluri, acest lucru permitand ca urmatoarea instructiune sa fie adusa din memorie (fetch) in timp ce instructiunea curenta este in executie. Majoritatea instructiunilor se executa intr-un singur ciclu de instructiune, acest lucru permitand atingerea unui throughput de 1MIPS pe MHz.

Un avantaj fata de celelalte familii concurente de µC in constituie faptul ca arhitectura AVR este optimizata pentru executia de cod C compilat.

III. Digital I/O

III.1. Porturi digitale:

Cel mai simplu mod de comunicatie dintre µC si exteriorul il constituie porturile digitale de intrare/iesire. Microcontrollerul ATmega16 folosit in lucrarile de laborator are patru astfel de porturi, numite PORTA, PORTB, PORTC si PORTD.

Aceste porturi au corespondenta cu exteriorul prin pinii circuitului integrat. Deoarece arhitectura AVR este pe 8 biti iar porturile corespund unor registre interne, acestea vor avea la randul lor 8 pini: pin1..8 pentru PORTB, pin14..20 pentru PORTD etc.

Figura 1. Porturile digitale de intrare/iesire pentru ATmega16

Toate cele patru porturi sunt bidirectionale, datele pot circula fie dinspre chip catre exterior (output) sau invers, dinspre exterior catre chip (input). Mai mult, fiecare bit dintr-un port poare fi controlat independent, fie ca intrare, fie ca iesire digitala. Fiecare port are asociat trei registre prin care utilizatorul poate schimba fluxul datelort si poate scrie sau citi date din portul respectiv. Aceste trei registre sunt DDRn, PORTn si PINn, unde n poate sa fie A, B, C sau D in functie de portul selectat.

De exemplu, pentru portul A, registrele asociate sunt urmatoarele:

1. **Data Direction Register A (DDRA). Controleaza directia datelor prin portul A. Fiecare pin extern ce corespunde portului A poate fi asignat ca intrare sau iesire digitala prin scrierea unei valorii "0" respective "1" pe pozitia corespunzatoare in DDRA.**

   Exemplu:
   DDRA = 0xFF; Toti pinii din PORTA sunt iesiri.
   DDRA = 0x00; Toti pinii din PORTA sunt intrari.
   DDRA = 0xF1; Pinul 1 este iesire, 2, 3 si 4 intrari, iar restul iesiri.

2. **Output Register A (PORTA). Valoarea scrisa in acest registru va fi pusa pe pinii exteriori ai PORTA sub forma de tensiuni (5V= "1", 0V = "0"). Aceasta operatie este posibila numai dupa ce portul a fost configurat ca iesire.**

   Exemplu:
   PORTA = 0xFF Pe toti pinii PORTA (33..40) va fi prezenta tensiunea de 5V
   PORTA = 0x0F Pinii 37..40 vor avea 5V iar pinii 33..36, 0V

3. **Input Register A (PINA). Daca portul a fost configurat ca intrare, valoarea citita din acest registru corespunde starii logice in care se afla pinii portului A.**

Figura 2. Registrele de control asociate portului A

In aceasta lucrare de laborator veti invata sa folositi porturile de I/O ale microcontrollerului ATmega16 pentru a face operatii simple de citire sau scriere. La sfarsitul acestui laborator veti sti cum sa programati, sa simulati si sa rulati un program care sa citeasca o serie de intrari logice si in functie de valorile citite sa actioneze niste dispozitive de iesire.

III.2. Mediul de dezvoltare Proteus:

Proteus reprezinta o suita de aplicatii produsa de Labcenter Electronics pentru designul de sisteme electronice. Suita ofera aplicatii pentru realizarea de scheme electronice, simularea acestora, scrierea si compilarea de cod destinat circuitelor simulate, PCB design etc.

Utilitarul folosit cel mai des la orele de laborator este ISIS care va permite realizarea si simularea schemelor electronice folosite in cursul lucrarilor de laborator.

Figura 3. Utilitarul ISIS

III.3. Diode LED:

In acest prim laborator vom seta si vom face cateva operatii de scriere pe un port digital. Cel mai simplu mod de a urmari starea logica a bitilor unui port consta in legarea unor diode LED pe pinii corespunzatori portului respectiv. Led-ul se va aprinde daca bitul are valoarea logica "1", corespunzatoare unei tensiuni de 5V si va fi stins pentru starea logica "0".

Diodele LED sunt dispozitive realizatea prin doparea unui element semiconductor pentru a realize o jonctiune p-n. La trecerea unui curent electric prin aceasta jonctiune se elibereaza energie sub forma de fotoni. Ledurile emit de obicei intr-o banda foarte restransa de lungimi de unda, facand ca lumina generata sa aiba o singura culoare. Lungimea de unda a luminii emise poate sa varieze in functie de tipul semiconductorului si de modul in care se realizeaza doparea. Astfel se pot fabrica leduri care sa emita in orice lungime de unda a spectrului, de la infrarosu la ultraviolet.

Datorita fiabilitatii si consumului redus de energie aplicatiile sunt multiple. Ledurile pot fi folosite in telecomenzi sau in sistemele de vedere pe timp de noapte, in afisaje de toate tipurile sau formele, la iluminat, la transmisia de date prin fibra optica sau la sterilizarea apei potabile prin lumina ultravioleta.

Deoarece sunt in esenta diode, ele conduc intr-un singur sens si trebuiesc polarizate corect pentru a emite lumina. Cel mai simplu mod de a face acest lucru este prin inserierea cu o rezistenta, ca in figura de mai jos.

Valoarea rezistentei se calculeaza simplu folosind legea lui Ohm. Cele mai multe diode LED au nevoie de o cadere de tensiune de 2V si un curent de doar 10mA pentru a emite la capacitate maxima. Avand aceste date putem calcula rezistenta cu urmatoarea formula:

$$R = (Vin - U) / I$$

Pentru o tensiune de alimentare de 5V, valoarea lui R este de 300Ω. Acest lucru nu inseamna ca dioda nu va emite lumina daca valoarea rezistentei nu este de exact 300Ω, in practica puteti folosi orice valoare intre 100Ω si 1KΩ. Atentie insa, o valoare foarte mica a rezistentei poate duce la distrugerea diodei din cauza curentului excesiv.

III.4. Primul program:

Incarcati schema de la laboratorul curent in ISIS. Dupa cum se poate observa in figura de mia jos, la controllerul ATmega16 au fost legate 8 leduri pe portul A si patru butoane pe portul D. Aceasta schema corespunde configuratiei placilor de dezvoltare din laborator.

Figura 4. Schema electrica laborator 1

In primul program scris pentru acest laborator vom configure PORTA ca iesire pentru a putea comanda cele 8 leduri si PORTD ca intrare, pentru a putea citi apasarea unui buton, apoi vom scrie o valoare binara in registrul de iesire PORTA.

Incarcati si rulati programul de mai jos:

```
; Laborator 1 PM
.INCLUDE "C:\Program Files\Labcenter Electronics\Proteus 6
Professional\Tools\AVRASM\APPNOTES\m16def.inc"

.cseg                          ; Declaratia segmentului de cod
.org 0                         ; Adresa 0, aici incepe uP executia imediat
dupa reset
        rjmp init
.org 30
init:
        ldi r16, 0b11111111    ; Se incarca valoarea 255 in registrul general
R16
        out DDRA, r16          ; Initializam pinii din PORTA ca pini de iesire

        ldi r16, 0b00110011    ; Incarcam o valoare oarecare in R16
        out PORTA, r16         ; Scriem valoarea din R16 pe portul de iesire
```

Primele linii din cod sunt directive de compilator si nu vor fi incluse in codul obiect generat in urma compilarii:

.INCLUDE "cale_catre_fisier" specifica calea catre fisierul cu definitii pentru resursele specific microcontrollerului ATmega16.

.cseg toate instructiunile de dupa aceasta directive se vor pune in memoria de program

.org x directiva de adresa. Compilatorul va pune codul primei instructiuni de dupa aceasta directiva la adresa x in memoria de program, unde x este este o constanta hexazecimala.

Prima instructiune este rjmp init, salt la eticheta init si se afla la adresa 0 in memoria de program. Aceasta este adresa la care µC incepe executia programului imediat dupa ce a primit un semnal de RESET.

Urmatoarea instructiune se afla la adresa 0x30 in memoria de program si este o instructiune load immediate (ldi), de incarcare a unei valori imediate intr-un registru general. Aceasta valoare este constanta binara 0b11111111.

Pentru a initializa pinii portului A ca iesiri, aceasta valoare trebuie sa fie incarcata in registrul de directie al portului A, DDRA. Acest lucru se face prin urmatoarea instructiune: out DDRA, R16 care scrie valoarea stocata anterior in R16 in registrul DDRA.

Dupa ce am configurat portul A ca iesire, este timpul sa scriem ceva pe acest port, punand o valoare pe opt biti in registrul PORTA.

In urma compilarii si executiei acestui cod, ledurile corespunzatoare bitilor de 1 din registrul PORTA se vor aprinde:

Atentie: Arhitectura interna a µC specifica doua tipuri de registre: registre generale (R0, R1…R30) si registre de I/O (toate celelalte registre in afara de cele generale, DDRA, PORTB, PIND etc.)

Doar registrele generale sunt incluse in core-ul µP si au acces direct la magistrala de instructiuni. Din aceasta cauza nu putem incarca o constanta intr-un registru de I/O printr-o singura instructiune si trebuie mai intai sa incarcam constanta intr-un registru general cu ldi apoi sa mutam continutul registrului general intr-un registru de I/O printr-o instructiune de out .

Instructiuni de tipul out DDRA, 0b11100111 sau ldi DDRA, 255 sunt invalide si vor genera eroare la compilare.

Programul anterior comanda bitii portul A scriind o singura valoare pe iesire. Daca vrem sa interactionam cu µC trebuie sa avem un mechanism prin care sa-i trimitem comenzi. Cel mai simplu mod de interactiune cu utilizatorul il constituie niste simple butoane, care prin apasare pot genera diferite comenzi. Modul in care se conecteaza un push-button este dat in figura de mai jos.

Figura 5. Conectarea unui push-button: a) incorect, cu intrare flotanta, b) corect, cu rezistenta de pull-up

Fig. 5.a arata un buton conectat la un pinul PD0 al µC. La apasarea butonului, intrarea PD0 va fi legata la masa, deci va fi in starea logica "0". Intrebarea este, in ce stare logicaeste intrarea atunci cand butonul nu este apasat? Raspunsul este HiZ sau stare de impedanta marita. Este ca si cum intrarea ar fi lasata in aer, ea nefiind conectata nici la masa, nici la Vcc. Aceasta stare nu poate fi citita de catre circuitele interne ale µC, in definitiv un bit dintr-un registru putand sa ia doar valorile 0 sau 1. Figura 5.b arata modul correct de conectare al butonului, folosind o rezistenta de pull-up intre pinul de intrare si Vcc. Aceasta rezistenta are rolul de a aduce intrarea in starea "1" logic atunci cand butonul este liber prin "ridicarea" potentialului liniei la Vcc.

Pentru a economisi spatiu exterior, aceste rezistente au fost incluse in interiorul circuitului integrat. Initial ele sunt dezactivate iar activarea acestora se poate face prin software scriind o valoare in registrul de iesire (PORTn) al unui port care a fost configurat drept intrare.

Codul urmator aprinde o secventa de leduri atunci cand este apasat primul buton si o secventa complementara atunci cand este apasat al doilea buton.

```
; Laborator 1 PM

.INCLUDE "C:\Program Files\Labcenter Electronics\Proteus 6
Professional\Tools\AVRASM\APPNOTES\m16def.inc"

.cseg                                   ; Declaratia segmentului de cod
.org 0                                  ; Adresa 0, aici incepe uP executia
imediat dupa reset
        rjmp init
.org 30
init:
        ldi r16, 0b11111111             ; Se incarca valoarea 255 in registrul
general R16
        out DDRA, r16                   ; Initializam pinii din PORTA ca pini
de iesire

        ldi r16, 0
        out DDRD, r16                   ; Initializam pinii din PORTD ca pini
de intrare

        ldi r16, 0b11111111
        out PORTD, r16                  ; Initializam rezistentele de pull-up
de pe PORTD

        ldi r16, 0b00110011
        ldi r17, 0b11001100
loop:                           ; Bucla principala
        in r18, PIND            ; Citim valoarea portului de intrare
```

```
        sbrs r18, 2             ; Daca butonul 1 nu a fost apasat nu se executa
instructiunea urmatoare
        out PORTA, r16

        sbrs r18, 3             ; Daca butonul 2 nu a fost apasat nu se executa
instructiunea urmatoare
        out PORTA, r17

        rjmp loop
```

Putem executa mai mult de o singura instructiune la o apasare de buton prin introducerea de rutine in codul nostru. Rutinele sunt segmente separate de cod care pot si apelate din programul principal. La terminarea executiei unei rutine, programul se intoarce la urmatoarea instructiune din codul principal.

```
; Laborator 1 PM
.INCLUDE "C:\Program Files\Labcenter Electronics\Proteus 6
Professional\Tools\AVRASM\APPNOTES\m16def.inc"

.cseg                                   ; Declaratia segmentului de cod
.org 0                                  ; Adresa 0, aici incepe uP executia
imediat dupa reset
        rjmp init
.org 30

init:
        ;initializare stiva
        ldi r16, HIGH(RAMEND)   ; Octetul HIGH al adresei stivei
        out SPH, r16
        ldi r16, LOW(RAMEND)    ; Octetul LOW al adresei stivei
        out SPL, r16

        ldi r16, 0b11111111             ; Se incarca valoarea 255 in registrul
general R16
        out DDRA, r16                   ; Initializam pinii din PORTA ca pini
de iesire

        ldi r16, 0
        out DDRD, r16                   ; Initializam pinii din PORTD ca pini
de intrare

        ldi r16, 0b11111111
        out PORTD, r16                  ; Initializam rezistentele de pull-up
de pe PORTD

loop:
        in r18, PIND            ; Citim valoarea portului de intrare

        sbrs r18, 2             ; Daca butonul 1 nu a fost apasat se face skip
la instructiunea urmatoare
        call pattern1           ; Apel rutina pattern1

        sbrs r18, 3             ; Daca butonul 2 nu a fost apasat se face skip
la instructiunea urmatoare
        call pattern2
```

```
        rjmp loop

pattern1:                       ; Rutina pattern1
        ldi r16, 0b00110011
        out PORTA, r16
ret                             ; Instructiunea return

pattern2:
        ldi r16, 0b11111111
        out PORTA, r16
ret
```

Pentru a putea folosi rutinele in codul nostru trebuie mai intai sa initializam stiva. Aceasta este tinuta in memoria SRAM a µC si este incrementata de la jos in sus, de la ultima adresa catre prima. Initializarea stivei consta in initializarea registrului Stack Pointer (SP) cu o adresa de start, la care se va introduce primul element. In cod, acest lucru este realizat de secventa urmatoare:

```
ldi r16, HIGH(RAMEND)  ; Octetul HIGH al adresei stivei
out SPH, r16
ldi r16, LOW(RAMEND)   ; Octetul LOW al adresei stivei
out SPL, r16
```

Constanta RAMEND are valoarea 0x045F si reprezinta ultima adresa din memoria RAM de 1kB a µC. Registrul SP este impartit in doua registre de cate opt biti, SPL si SPH care vor primi la initializare octetul low, respectiv high din RAMEND. La orice operatie de introducere in stiva (push), registrul SP va fi decrementat, iar la fiecare operatie de extragere din stiva (pop) SP va fi incrementat.

Instructiunea call pattern1 din exemplul de cod anterior va salva in stiva adresa urmatoare din memoria de program apoi va executa instructiunile din corpul rutinei pattern1. In momentul intalnirii instructiunii ret programul extrage din stiva adresa salvata si se intoarce in programul principal la instructiunea urmatoare (sbrs r18, 3).

Lucrarea de laborator 1

Folosind exemplele de cod prezentate si documentatia data, scrieti un program care sa produca mai multe efecte de lumina dinamica cu ledurile de pe placa de la laborator.

Trebuie sa implementati minim trei secvente. Fiecare secventa poate fi pornita prin apasarea unui buton.

Exemplu:

```
      Secventa1      Secventa2      Secventa3

  t0   *-------      *------*       **------
  t1   -*------      -*----*-       -**-----
  t2   --*-----      --*--*--       --**----
  t3   ---*----      ---**---       ---**---
  t4   ----*---      ---**---       ----**--
  t5   -----*--      --*--*--       -----**-
  t6   ------*-      -*----*-       ------**
  t7   -------*      *------*       *------*
```