Sistem de afișare a mesajelor cu afișoare LED cu 16 segmente

Florin Nica

Cod sursă Github

Un microcontroler este un "calculator pe un chip". Spre diferență de microprocesor, microcontrolerul este un calculator pe un chip deorece el conține și memorie și interfețe de intrare-ieșire pe lângă CPU. Deoarece memoria și interfețele care încap pe un chip sunt limitate, microcontrolerele tind să fie utilizate în sisteme mai mici care necesită doar un microcontroler și câteva elemente adiționale.

Deși tradițional microcontrolerele se programează în limbajul nativ, asembler, evoluția compilatoarelor pentru limbaje de nivel înalt (C) a ușurat foarte mult sarcina programatorilor. Totuși, pentru anumite aplicații acesta este indispensabil, deoarece este mai rapid decât limbajele de nivel înalt.

Toate microcontrolerele PIC conțin "biți de configurare". Aceștia pot modifica diferiți parametri de funcționare: tipul oscilatorului, protecția memoriei de program, specificații pentru reset și alimentare.

PIC-urile au fost împărțite în trei mari familii: familia Base-Line(linia de bază), care folosește cuvinte de instrucțiuni pe 12 biți pentru unele PIC-uri (12C508), familia Mid-Range(clasa medie), care folosește cuvinte pe 14 biți (și din care fac parte 16F84 și 16F876), și familia High-End (clasa de vârf) care folosește cuvinte de 16 biți. Au apărut și alte familii ulterior, ca Enhanced family, și vor continua să apară.

Memoria microcontrolerelor este folosită pentru a stoca atât date cât și instrucțiuni. Instrucțiunile sunt decodificate de către CPU și sunt executate iar datele pot fi scrise/citite în/din memorie. De aceea, de modul în care este organizată memoria și de modul în care comunică cu CPU depinde performanța dispozitivului. Microcontrolerele PIC sunt construite după modelul arhitecturii hardware Harvard, care presupune memorii separate pentru date și instrucțiuni. Deoarece se folosesc magistrale separate, se pot manipula datele și instrucțiunile simultan ceea ce duce la avantajul unei viteze sporite de execuție a programelor. Memoria SRAM (volatilă) este folosită pentru date iar cea flash pentru program.

Microcontrolerele PIC pot funcționa în diverse configurații pe partea de semnal de clock: generator extern, cristal de cuarț, rezonator ceramic, circuit RC extern, oscilator intern. În funcție de

aplicația concretă se alege și tipul de clock (de exemplu, un ceas realizat cu oscilator RC ar avea o eroare foarte mare, inacceptabilă, deci se va folosi un cuarț).

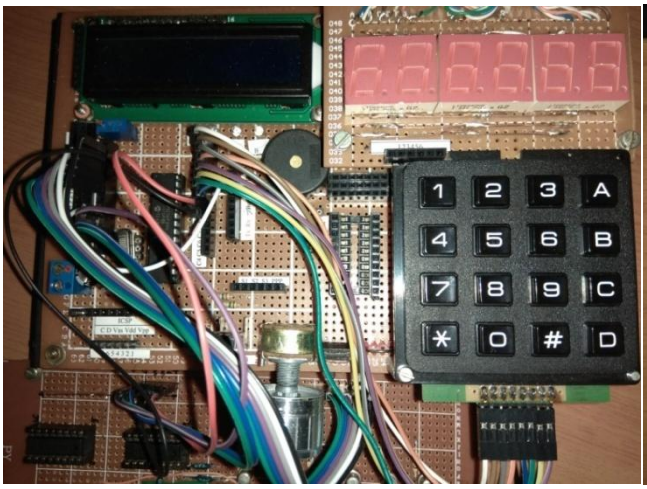
Clock-ul aplicat PIC-urilor este predivizat cu 4 în interiorul acestuia. Aceasta este baza de timp care stabilește durata unui ciclu. Dacă folosim de exemplu, un cuarț de 4MHz, obținem deci 1.000.000 cicluri/secundă, sau, cum PIC-ul execută practic o instrucțiune pe ciclu, (exceptând salturile) rezultă o viteză de procesare de 1 MIPS (1 milion de instrucțiuni/secundă).

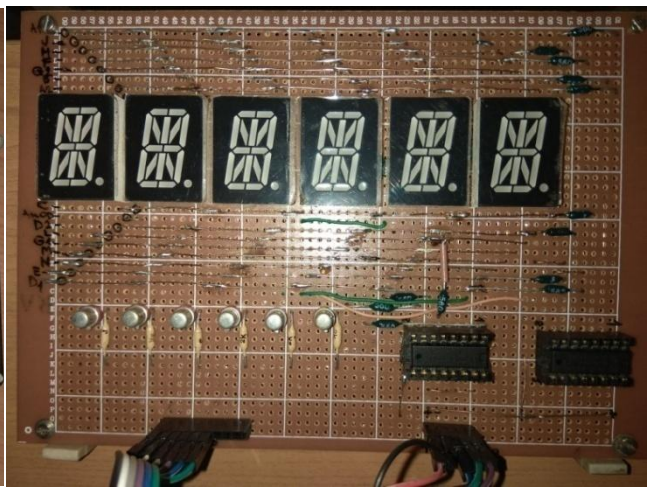
PIC-urile sunt componente statice, ceea ce înseamnă că frecvența de ceas poate fi redusă până la oprirea completă, fară pierdere de date sau disfuncționalități. Spre deosebire de acestea, componentele dinamice (cum sunt procesoarele din PC), au nevoie ca frecvența de ceas să rămână în limite foarte precise.

Un aspect foarte important este sistemul de întreruperi. Acestea pot veni din diferite surse: schimbarea nivelului logic al unui pin, o măsurătoare a convertorului A/D, depășirea (overflow) a unui timer, scriere în EEPROM completă, etc. Întreruperile pot fi activate/dezactivate dintr-un bit general (GIE) și individual pentru fiecare în parte. Registrele responsabile pentru controlul întreruperilor sunt RCON, INTCON, PIR, PIE,IPR ele diferind în funcție de PIC. Există și PIC-uri care au două tipuri de întreruperi, cu prioritate mare și mică. Întreruperile cu prioritate mare le pot întrerupe pe celelalte. Întreruperile au indicator individual (flag) care este setat atunci când se generează o întrerupere iar verificarea acestuia determină care este sursa responsabilă.

Sistemul este realizat cu microcontrolerul PIC 16F886, conţine şase afişoare de tip LED cu 16 segmente, două registre de deplasare 74HC595 şi o tastatură 4x4. Afişoarele sunt comandate multiplexat cu ajutorul a şase tranzistoare PNP BC177. Sistemul permite introducerea de la tastatură a caracterelor alfanumerice, dând astfel posibilitatea de afişare a unor cuvinte sau propoziţii scurte care vor fi afişate prin efectul de "scroll" de la dreapta la stânga. Viteza de "scroll" este reglabilă. Sistemul de dezvoltare este de tip "home-made", realizat din cablaj perforat. Mesajele pot fi rulate din memoria RAM sau pot fi salvate în EEPROM pentru afişare ulterioară.

Acest proiect poate fi folosit pentru a afişa în locuri publice (săli de aşteptare) mesaje de informare sau atenţionare.





Proiectul este realizat cu două plăci: prima conține microcontrolerul, tastatura și LED-ul care indică apăsarea unei taste, iar cea de-a doua conține afișoarele, cele șase tranzistoare corespunzătoare și cele două registre de deplasare.

Microcontrolerul 16F886 are memoria de program de 8192 kB, memoria RAM de 368 B și EEPROM de 256 B. Am folosit două tipuri de întreruperi, IOC pentru tastatură și TMR0 pentru multiplexarea afișajului. Afișoarele sunt activate prin portul A iar tastatura este conectată la portul B. Am folosit un LED conectat la pinul RA6 care indică faptul că o tastă a fost apăsată.

Pe tastele numerice sunt alocate numere și litere, după modelul tastaturii telefoanelor mobile mai vechi (cu "butoane"):

1 – 1
2 – 2,A,B,C
3 – 3,D,E,F
4 – 4,G,H,I
5 – 5,J,K,L
6 – 6,M,N,O
7 – 7,P,Q,R,S
8 – 8,T,U,V
9 – 9,W,X,Y,Z
0 – 0,space(blank)

La o apăsare scurtă pe o tastă numerică se afișează numerele. Pentru a afișa și literele se ține apăsat pe tastă până apar și celelalte caractere.

Funcțiile tastelor nonnumerice sunt:

A – (apăsare scurtă) – arată cursorul (_) sau îl mută la caracterul următor

A – (apăsare lungă) – începe ”defilarea” mesajului stocat în memoria RAM

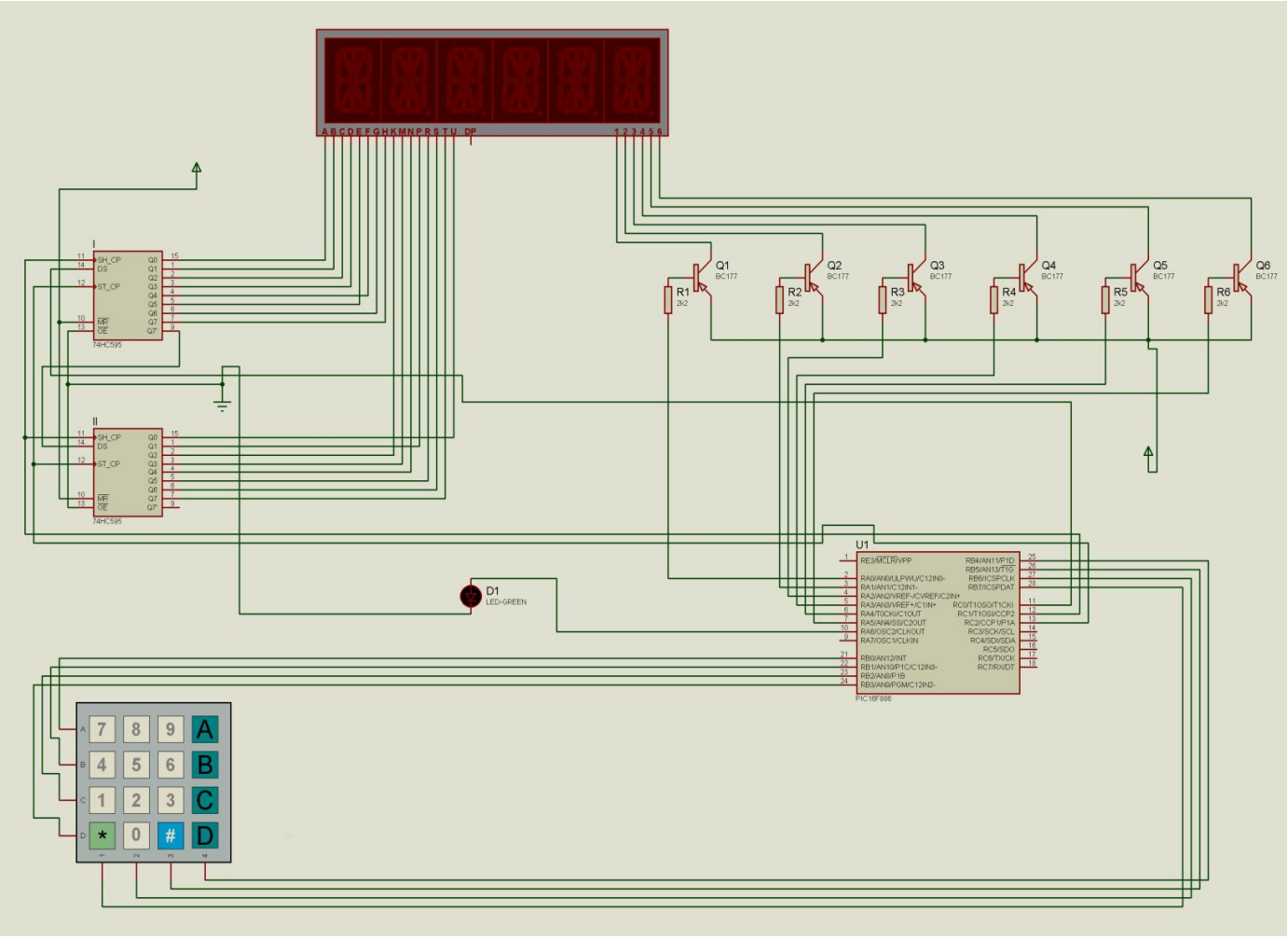
B – începe ”defilarea” mesajului stocat anterior în memoria EEPROM

C – salvează mesajul introdus în memoria EEPROM (după salvare se afișează STORED)

D – ieșire din modul de afișare a mesajului (apare mesajul standard de întâmpinare)

* – micșorează viteza de defilare a mesajului (apăsări succesive).

– mărește viteza de defilare a mesajului (apăsări succesive).



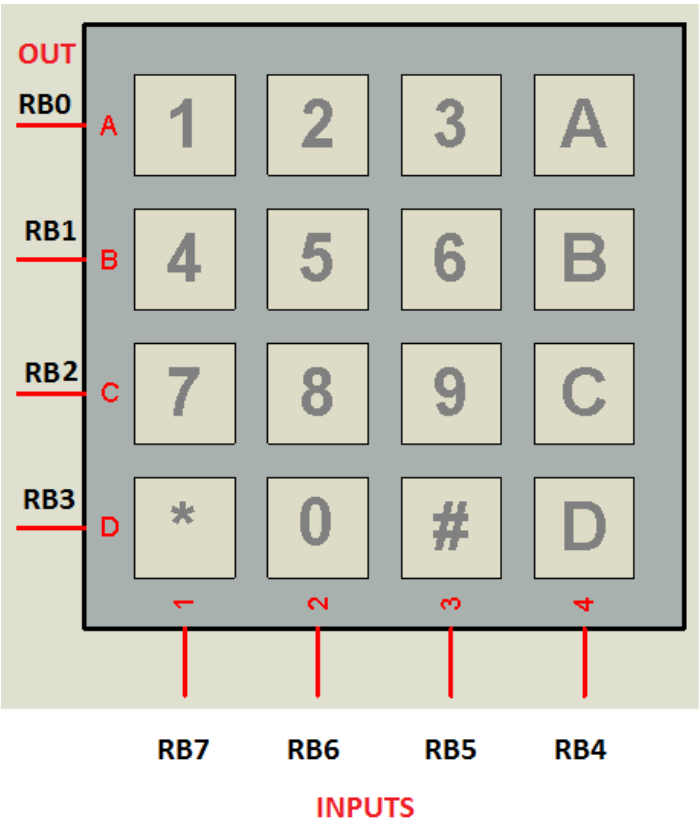
Proiectul este realizat în mediul de programare MPLAB X IDE v4.05 cu compilatorul XC8 v1.44, având ca debugger/programator PICkit 3. MPLAB X IDE este un mediu de programare cu facilități avansate, inspirat din proiectul "open source" NetBeans IDE. Deși am folosit versiunea

gratuită a compilatorului, care nu optimizează foarte bine codul generat, rezultatul a fost satisfăcător.

Descrierea aplicației demonstrative

În funcția *main* este apelată funcția *init* în care se stabilesc intrările și ieșirile porturilor, se activează întreruperile și rezistențele PULL-UP necesare tastaturii. Apoi, într-o buclă infinită, se testează apăsarea tastelor nonnumerice (A,B,C, D), corespunzătoare funcțiilor principale, detaliate mai sus. În aceste testări se apelează și funcția *control_speed*, care verifică apăsarea tastelor "* " și "#", responsabile de controlul vitezei de deplasare a mesajului.

Modul de funcționare al tastaturii



Tastatura este conectată la portul B, care are pinii RB0, RB1, RB2, RB3 configurați ca ieșiri iar pinii RB4, RB5, RB6, RB7 configurați ca intrări. Totodată, portul B are activată întreruperea de tip IOC. Detectarea tastei apăsate se face prin explorarea secvențială a pinilor configurați ca intrări. Intrările au activate rezistența de PULL-UP din soft (registrul WPUB) și se află în 1 Logic. Liniile (RB0 ... RB3) se pun pe rând în 0 Logic (celelalte fiind în 1 Logic) și se testează pe rând coloanele (RB7...RB4). De exemplu, să presupunem că la un anumit moment de timp, linia conectată la RB2

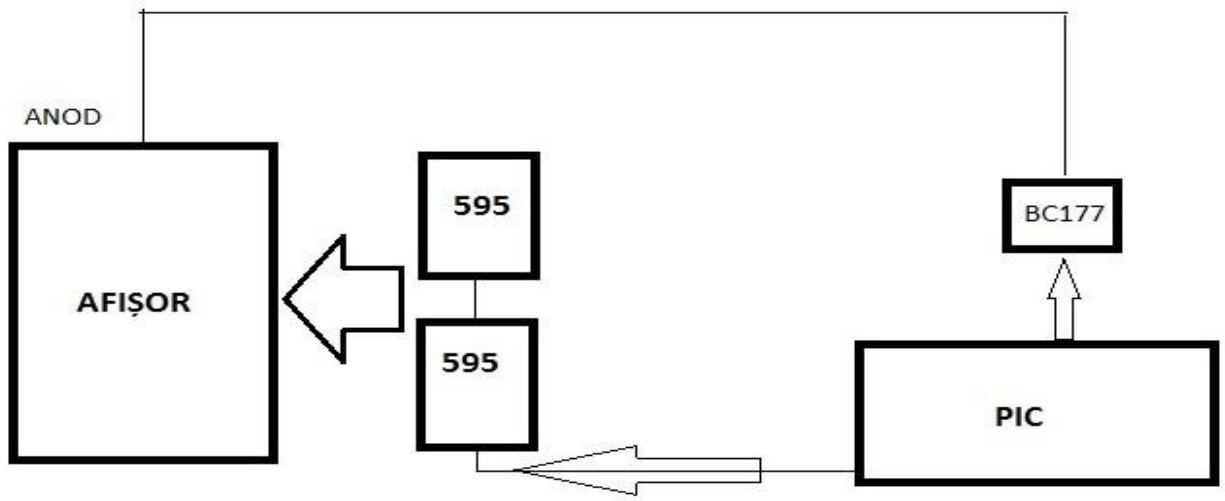
este în 0 Logic. Dacă programul sesizează că RB6 este în 0 Logic (*if(!COL2)*) rezultă că a fost apăsată tasta 8 (deoarece toate intrările stau în 1 Logic atunci când nu este nimic apăsat).

La o apăsare scurtă pe o tastă se afișează numerele. Pentru a afișa și literele am folosit variabila *pressed_time*. Se ține apăsat pe tastă până apar și celelalte caractere.

Atunci când se ține apăsat un buton se incrementează variabila *pressed_time*. Am stabilit niște praguri pentru această variabilă care setează variabila *character* cu valori diferite:

if (pressed_time>2500) *character = "A";*
if(presed_time>5000) character= "B";
if (presed_time>7500) character= "C";
etc.
Așadar la apăsarea prelungă a unui buton defilează caracterele asociate acestuia.

Pentru tastele nonnumerice (A, B, C, D, *, #) am folosit o variabilă separată de tip enumerare (*button_pressed*). Pentru ca la apăsarea oricărui buton să înceapă "defilarea" caracterelor de la început (4, 6, H, I) sau (5, J, K, L) variabila *pressed_time* trebuie resetată atunci când nu este apăsată nicio tastă. Am folosit pentru aceasta TMR2. TMR2 este resetat de fiecare dată când se intră în întrerupere, deci acesta nu poate să ajungă la valoarea PR2 (adică TMR2 IF=1) decât atunci când nu este nici un buton apăsat.
Afișarea unui caracter



Se folosesc două registre de deplasare HC595 "înseriate", adică ieșirea ultimului bit din primul registru este cuplată la intrarea serială a celui de-al doilea. Cei 16 biți, corespunzători celor 16 segmente din afișor, sunt introduși bit cu bit pe intrarea serială a primului registru (funcția *send_data*), apoi se transmite 1 logic pe pinii ST ai celor două registre. Cei 16 biți apar la ieșirile Q0-Q7 ale celor două registre și, implicit la catozii afișorului. PIC-ul trimite 0 Logic spre baza

tranzistorului corespunzător afișorului (în rutina de întrerupere), acesta se deschide și astfel se aplică 1 Logic pe anodul comun. În funcție de configurația (masca) celor 16 biți se iluminează corespunzător afișorul (apare caracterul). Pentru aceasta am folosit tabloul de constante *CHAR_DATA* care este stocat în memoria FLASH pentru a economisi memoria RAM.

Tabloul *buffer* este folosit pentru stocarea caracterelor care apar efectiv pe cei șase digiți ai afișorului. Cu ajutorul funcției *choose_digit_display* se selectează digitul corespunzător și se stochează caracterele în *buffer* (pentru a fi afișate) și în *message.*

În rutina de întrerupere se testează doi indicatori, T0IF (întrerupere TMR0) și RBIF (întrerupere pe pinii portului B). Multiplexarea afișajului are loc cu ajutorul unei instrucțiuni *switch* și a unei variabile *display_flag.* Această variabilă este modificată în cadrul fiecărei ramuri a *switch*-ului, pregătind astfel activarea următorului digit la momentul generării întreruperii TMR0 următoare.

Funcția *save_in_eeprom* execută salvarea mesajului în memoria EEPROM. În locația 0 este salvată variabila locală *message_length* care reprezintă numărul de caractere al mesajului, calculat cu funcția *strlen*, deoarece la citirea ulterioară a mesajului din EEPROM (*read_eeprom*) trebuie cunoscută lungimea acestuia. Funcția *left_rotate* asigură deplasarea spre stânga a elementelor din tablouri (arrays).

Bibliografie

https://embedjournal.com/interrupt-on-change-ioc-in-pic-microcontrollers/

http://embedded-lab.com/blog/lab-18-matrix-keypad-interfacing/

http://embedded-lab.com/blog/lab-15-scrolling-text-message-on-an-led-dot-matrix-display/

http://www.geeksforgeeks.org/array-rotation/

Cod sursa

// fisierul config.h

```
// CONFIG1
#pragma config FOSC = INTRC_NOCLKOUT// Oscillator Selection bits (INTOSCIO oscillator: I/O function on
RA6/OSC2/CLKOUT pin, I/O function on RA7/OSC1/CLKIN)
#pragma config WDTE = OFF       // Watchdog Timer Enable bit (WDT disabled and can be enabled by SWDTEN bit
of the WDTCON register)
#pragma config PWRTE = OFF      // Power-up Timer Enable bit (PWRT disabled)
#pragma config MCLRE = OFF      // RE3/MCLR pin function select bit (RE3/MCLR pin function is digital input,
MCLR internally tied to VDD)
#pragma config CP = OFF         // Code Protection bit (Program memory code protection is disabled)
#pragma config CPD = OFF        // Data Code Protection bit (Data memory code protection is disabled)
#pragma config BOREN = ON       // Brown Out Reset Selection bits (BOR enabled)
#pragma config IESO = ON        // Internal External Switchover bit (Internal/External Switchover mode is enabled)
#pragma config FCMEN = ON       // Fail-Safe Clock Monitor Enabled bit (Fail-Safe Clock Monitor is enabled)
#pragma config LVP = OFF        // Low Voltage Programming Enable bit (RB3 pin has digital I/O, HV on MCLR must
be used for programming)

// CONFIG2
#pragma config BOR4V = BOR40V   // Brown-out Reset Selection bit (Brown-out Reset set to 4.0V)
#pragma config WRT = OFF        // Flash Program Memory Self Write Enable bits (Write protection off)
```

// fisierul main

```
/*
  inspired from:
    https://embedjournal.com/interrupt-on-change-ioc-in-pic-microcontrollers/
    http://embedded-lab.com/blog/lab-18-matrix-keypad-interfacing/
    http://embedded-lab.com/blog/lab-15-scrolling-text-message-on-an-led-dot-matrix-display/
    http://www.geeksforgeeks.org/array-rotation/

 */

/*
     Keypad connection

         RB7 RB6 RB5 RB4
          |   |   |   |
          |   |   |   |
    RB0 -- 1   2   3   A

    RB1 -- 4   5   6   B

    RB2 -- 7   8   9   C

    RB3 -- *   0   6   #

  1 -- 1                  A -- short press->show/move cursor, long press->start display message without save in EEPROM
  2 -- 2,A,B,C            B -- start display message saved previous in EEPROM
  3 -- 3,D,E,F            C -- save message in EEPROM
  4 -- 4,G,H,I            D -- Exit
  5 -- 5,J,K,L            * -- decrease speed
  6 -- 6,M,N,O            # -- increase speed
```

```
  7 -- 7,P,Q,R,S
  8 -- 8,T,U,V
  9 -- 9,W,X,Y,Z
  0 -- 0,space(blank)
 */

#include <xc.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include "config.h"

#define _XTAL_FREQ 8000000

#define DIGIT0 PORTAbits.RA0   // left
#define DIGIT1 PORTAbits.RA1
#define DIGIT2 PORTAbits.RA2
#define DIGIT3 PORTAbits.RA3
#define DIGIT4 PORTAbits.RA4
#define DIGIT5 PORTAbits.RA5   // right

#define KEYPAD_LED PORTAbits.RA6

#define LOW 0
#define HIGH 1

// Define 74HC595 connections.
#define SERIAL_DATA PORTCbits.RC0  // serial data input
#define SH_CLK PORTCbits.RC1       // shift register clock input
#define ST_CLK PORTCbits.RC2       // storage register clock input


// Keypad column connections
#define COL1 PORTBbits.RB7
#define COL2 PORTBbits.RB6
#define COL3 PORTBbits.RB5
#define COL4 PORTBbits.RB4

//  Global  ////////////////////////////////////////////////////////////
char character = '_'; //
enum not_number_buttons {DEFAULT, NO_PRESS, A, A_LONG, B, C, D, HASH, STAR};  // tastele A,B,C,D,#,*
enum not_number_buttons button_pressed = NO_PRESS;

unsigned char display_flag = 0;
unsigned int pressed_time;
char cursor = 0;
int speed = 600;
unsigned int index = 0;


// ASCII 33-47, 58-64, 91-94 nu conteaza, nu sunt afisate

// ASCII-->32   33   34   35   36   37   38
//      space
const unsigned int CHAR_DATA[] = {0xFFFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   // 39   40   41   42   43   44   45-   46   47
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF3F, 0x00, 0x00,
   //  0      1       2      3      4
   0xF6, 0xCFF7, 0x113F, 0x3BF, 0xCE3F,
   //  5     6      7       8      9
   0x223F, 0x203F, 0x3F36, 0x3F, 0x23F,
```

```
    // ASCII  58                >         64     A
    0x00, 0x00, 0x00, 0x00, 0xFFDE, 0x00, 0x00, 0xC3F,
    //  B      C       D       E       F       G
    0x3AD, 0x30FF, 0x3ED, 0x307F, 0x3C7F, 0x20BF,
    //  H       I       J       K       L       M
    0xCC3F, 0x33ED, 0xC1FF, 0xFC73, 0xF0FF, 0xCCD7,
    //  N       O       P       Q     R       S
    0xCCDB, 0xFF, 0x1C3F, 0xFB, 0x1C3B, 0x223F,
    // T        U        V       W       X       Y
    0x3FED, 0xC0FF, 0xFCF6, 0xCCFA, 0xFFD2, 0xFFD5,
    //  Z                       _
    0x33F6, 0x00, 0x00, 0x00, 0x00, 0xF3FF};

char greeting_message[35] = "A--WRITE B--SHOW C--STORE D--EXIT ";
char message[80] = " ";


// stocheaza caracterele care trebuie afisate pe fiecare Display
//                Digit1 .....    Digit6
char buffer[] = {' ', ' ', ' ', ' ', ' ', ' ', '\0'};  // blank



 //Remove trailing whitespace characters from string -- taie spatiile mari de la sfarsit
void rTrim(char * str)
{
    int index, i;

    // Find last index of non-white space character
    i = 0;
    while(str[i] != '\0')
    {
        if(str[i] != ' ' && str[i] != '\t' && str[i] != '\n')
        {
            index = i;
        }

        i++;
    }

   // str[index + 1] = ' ';
   // str[index + 2] = ' ';
   // str[index + 3] = ' ';
    /* Mark next character to last non-white space character as NULL */
    str[index + 1] = '\0';

}

void clear()
{
    int m;
    int n;
    short message_length;
    button_pressed = DEFAULT;
    message_length = strlen(message);
    cursor = 0;

    for (m = 0; m <= 5; m++) // clear display
    {
        buffer[m] = ' ';
    }
    for (n = 0; n <= message_length; n++) // clear message
```

```
    {
        message[n] = ' ';
    }
    //message[3]= '\0';
    character = '_'; // cursor
}

void init()
{
    ANSEL = 0x00; // all pin are Digital
    ANSELH = 0x00;
    TRISA = 0;
    TRISC = 0; // PORTC output
    PORTA = 0;
    TRISB = 0b11110000; // C1-C4 input  R1-R4 output

    // tranzistoare PNP, se blocheaza cu tensiuni pozitive,
    // se deschid cu tensiuni negative
    DIGIT0 = HIGH; // Disable digit 1      stanga
    DIGIT1 = HIGH; // Disable digit 2
    DIGIT2 = HIGH; // Disable digit 3
    DIGIT3 = HIGH; // Disable digit 4
    DIGIT4 = HIGH; // Disable digit 5
    DIGIT5 = HIGH; // Disable digit 6      dreapta
    KEYPAD_LED = LOW;

    T2CON = 0b00000000; // timer2 off, prescaler = 1, postscaler = 1
    TMR2 = 0;
    PR2 = 250;
    PIR1bits.TMR2IF = 0;

    //
    // Configure TIMER0 interrupts
    //
    TMR0 = 100; // Load TMR0
    //OPTION_REG = 0b00000100; // pull-up enabled, prescaler = 1:32
    OPTION_REG = 0b00000011;   // pull-up enabled, prescaler = 1:16
    //OPTION_REG = 0b00000010; // pull-up enabled, prescaler = 1:8
    PORTB = 0xF0; // 11110000
    //
    // Configure interrupts and enable interrupts
    //
    //RBIF_bit = LOW;
    INTCONbits.RBIF = LOW;
    INTCONbits.T0IE = HIGH;
    INTCONbits.RBIE = HIGH;
    INTCONbits.GIE = HIGH;
    IOCB = 0b11111111; // Interrupt-on-change enabled
    WPUB = 0b11110000;
    SH_CLK = LOW;
    ST_CLK = LOW;
    SERIAL_DATA = LOW;
}

void delay()
{
    for (int i = 0; i < speed; i++)
    {
        __delay_us(100);
    }
}
```

```
void save_in_eeprom()
{
   int j;
   char temp;
   char message_length;
   message_length = strlen(message);
   eeprom_write(0, message_length);
   for (j = 1; j <= message_length; j++)
   {
      temp = message[j - 1];
      eeprom_write(j, temp);
      __delay_ms(1);
   }
   __delay_ms(20);
   cursor = 7;
   buffer[0] = 'S';
   buffer[1] = 'T';
   buffer[2] = 'O';
   buffer[3] = 'R';
   buffer[4] = 'E';
   buffer[5] = 'D';
   delay();
   delay();
   delay();
   delay();
   delay();
   clear();
}

void read_eeprom() // load message from EEPROM to RAM
{
   int j;
   char temp;
   char message_length;
   message_length = eeprom_read(0); // citeste cate caractere are mesajul
   for (j = 1; j <= message_length; j++)
   {
      temp = eeprom_read(j);
      message[j - 1] = temp;
   }

}

void left_rotate_by_one(char arr[], int n)
{
   int j, temp;
   temp = arr[0];
   for (j = 0; j < n - 1; j++)
      arr[j] = arr[j + 1];
   arr[j] = temp;
}

// Function to left rotate arr[] of size n by d
void left_rotate(char arr[], int d, int n)
{
   int k;
   for (k = 0; k < d; k++)
      left_rotate_by_one(arr, n);
}

void send_data(unsigned char rw)
{
```

```
    unsigned int mask, t, flag;
    mask = 0x01;
    for (t = 0; t < 16; t++)
    {
        flag = CHAR_DATA[rw] & mask;
        if (flag == 0) SERIAL_DATA = LOW;
        else SERIAL_DATA = HIGH;
        SH_CLK = HIGH;
        SH_CLK = LOW;
        mask = mask << 1;
    }
    // Apply clock on ST_Clk
    ST_CLK = HIGH;
    ST_CLK = LOW;
}

void display(unsigned char ch)
{
    send_data(ch - 32);
}


void control_speed()
{
    if (button_pressed == HASH) //#     speed up
    {
        speed -= 100;
        if (speed < 200) speed = 200;
        button_pressed = DEFAULT;
    }

    if (button_pressed == STAR) //*      speed down
    {
        speed += 100;
        if (speed > 1600) speed = 1600;
        button_pressed = DEFAULT;
    }
    delay();
}



void choose_digit_display(char n)
{
    button_pressed = DEFAULT;
    //if (i > 80) i = 0;
    if (character == '_')
    {
        buffer[n] = ' '; // clear previous underline
        message[index] = ' ';
    }
    else
    {
        buffer[n] = character;
        message[index] = character;
    }
    character = '_'; // cursor
    cursor++;
    index++;
}

//
```

```
// Interrupt Service Routine
//

void interrupt isr(void)
{
   if (INTCONbits.T0IF == HIGH) // If Timer interrupt occured
   {
      TMR0 = 100; //
      INTCONbits.T0IF = LOW; // Clear timer interrupt flag

      switch (display_flag)
      {
         case 0: // primul digit (stanga)
         {
            DIGIT5 = HIGH; // Disable digit 6
            if (cursor == 0) display(character);
            else display(buffer[0]);
            DIGIT0 = LOW; // Enable digit 1
            display_flag = 1;
            break;
         }
         case 1:
         {
            DIGIT0 = HIGH; // Disable digit 1
            if (cursor == 1) display(character);
            else display(buffer[1]);
            DIGIT1 = LOW; // Enable digit 2
            display_flag = 2;
            break;
         }
         case 2:
         {
            DIGIT1 = HIGH; // Disable digit 2
            if (cursor == 2) display(character);
            else display(buffer[2]);
            DIGIT2 = LOW; // Enable digit 3
            display_flag = 3;
            break;
         }
         case 3:
         {
            DIGIT2 = HIGH; // Disable digit 3
            if (cursor == 3) display(character);
            else display(buffer[3]);
            DIGIT3 = LOW; // Enable digit 4
            display_flag = 4;
            break;
         }
         case 4:
         {
            DIGIT3 = HIGH; // Disable digit 4
            if (cursor == 4) display(character);
            else display(buffer[4]);
            DIGIT4 = LOW; // Enable digit 5
            display_flag = 5;
            break;
         }
         case 5: // ultimul digit (dreapta)
         {
            DIGIT4 = HIGH; // Disable digit 5
            if (cursor == 5) display(character);
            else display(buffer[5]);
```

```
            DIGIT5 = LOW; // Enable digit 6
            display_flag = 0;
            break;
        }

    }

}


if (INTCONbits.RBIF == HIGH) // If PORTB Change Interrupt occurred
{
    TMR2 = 0; // TMR2 overflow only when no button is pressed
    T2CONbits.TMR2ON = HIGH; // start Timer2

    KEYPAD_LED = HIGH;
    pressed_time++;
    if (pressed_time > 12500) pressed_time = 0;
    // Scan row 1
    PORTB = 0b00001110;
    if (!COL1)
    { // tasta 1
        character = '1'; // ascii 49
    }

    if (!COL2)
    { // tasta 2
        character = '2';
        if (pressed_time > 2500) character = 'A'; // A
        if (pressed_time > 5000) character = 'B'; // B
        if (pressed_time > 7500) character = 'C'; // C
    }
    if (!COL3)
    { // tasta 3
        character = '3'; // 3
        if (pressed_time > 2500) character = 'D'; // D
        if (pressed_time > 5000) character = 'E'; // E
        if (pressed_time > 7500) character = 'F'; // F
    }
    if (!COL4)
    { /// tasta A
        button_pressed = A;
        if (pressed_time > 4000) button_pressed = A_LONG; // scroll
    }

    // Scan row 2
    PORTB = 0b00001101;
    if (!COL1)
    { // tasta 4
        character = '4'; // 4
        if (pressed_time > 2500) character = 'G'; // G
        if (pressed_time > 5000) character = 'H'; // H
        if (pressed_time > 7500) character = 'I'; // I
    }
    if (!COL2)
    { // tasta 5
        character = '5'; //
        if (pressed_time > 2500) character = 'J'; // J
        if (pressed_time > 5000) character = 'K'; // K
        if (pressed_time > 7500) character = 'L'; // L
    }
    if (!COL3)
```

```
{ // tasta 6
   character = '6'; //6
   if (pressed_time > 2500) character = 'M'; // M
   if (pressed_time > 5000) character = 'N'; // N
   if (pressed_time > 7500) character = 'O'; // O
}
if (!COL4)
{ /// tasta B
   button_pressed = B;
}

// Scan row 3
PORTB = 0b00001011;
if (!COL1)
{ // tasta 7
   character = '7'; // 7
   if (pressed_time > 2500) character = 'P'; // P
   if (pressed_time > 5000) character = 'Q'; // Q
   if (pressed_time > 7500) character = 'R'; // R
   if (pressed_time > 10000) character = 'S'; // S
}
if (!COL2)
{ // tasta 8
   character = '8'; //8
   if (pressed_time > 2500) character = 'T'; // T
   if (pressed_time > 5000) character = 'U'; // U
   if (pressed_time > 7500) character = 'V'; // V
}
if (!COL3)
{ // tasta 9
   character = '9'; //9
   if (pressed_time > 2500) character = 'W'; // W
   if (pressed_time > 5000) character = 'X'; // X
   if (pressed_time > 7500) character = 'Y'; // Y
   if (pressed_time > 10000) character = 'Z'; // Z
}
if (!COL4)
{ // tasta C
   button_pressed = C;
}

// Scan row 4
PORTB = 0b00000111;
if (!COL1)
{ // tasta *
   button_pressed = STAR; //*
}
if (!COL2)
{ // tasta 0
   character = '0'; // 0
   if (pressed_time > 2500) character = ' '; // space (blank)
   if (pressed_time > 5000)
   {
      character = '0'; // 0
      pressed_time = 0;
   }
}
if (!COL3)
{ // tasta #
   button_pressed = HASH; //#
}
if (!COL4)
```

```
        { /// tasta D
            button_pressed = D;
        }
        PORTB = 0b00001111;
        PORTB = 0xF0; // 11110000
        KEYPAD_LED = LOW;

        INTCONbits.RBIF = LOW; // Clear interrupt flag
    }

}


//
// Start of MAIN Program
//

void main()
{
    char blank[] = "    ";
    init();

    for (;;)
    {   // atunci cand Timer2 == PR2(cand nu este apasat niciun buton),
        // TMR2IF_bit = 1 si se reseteaza time_pressed
        if (TMR2IF == HIGH) // reset time_pressed to start again with first character when press a button
        {
            pressed_time = 0;
        }

        if (cursor >= 6) // cursor == 6
        {
            cursor = 5;
            character = '_'; // cursor
        }

        if (button_pressed == NO_PRESS) // mesaj de pornire, niciun buton apasat
        {
            short message_length;
            message_length = strlen(greeting_message);
            button_pressed = DEFAULT;
            cursor = 6;
            index = 0;
            while (1)
            {
                control_speed();
                strncpy(buffer, greeting_message, 6); // copies first 6 characters form the string 'greeting_message' to 'Buffer'
                left_rotate(greeting_message, 1, message_length);
                if (button_pressed == A)
                {
                    button_pressed = DEFAULT;
                    clear();
                    break;
                }
                if (button_pressed == B) break;
            }
            index = 0;
        }

        if (button_pressed == D)
        {
            clear();
```

```
    button_pressed = NO_PRESS;
    cursor = 6;
    index = 0;
}


if ((button_pressed == A) && (display_flag == 0) && (cursor == 0))
{
    choose_digit_display(0);
}

if ((button_pressed == A) && (display_flag == 1) && (cursor == 1))
{
    choose_digit_display(1);
}

if ((button_pressed == A) && (display_flag == 2) && (cursor == 2))
{
    choose_digit_display(2);
}

if ((button_pressed == A) && (display_flag == 3) && (cursor == 3))
{
    choose_digit_display(3);
}

if ((button_pressed == A) && (display_flag == 4) && (cursor == 4))
{
    choose_digit_display(4);
}

if ((button_pressed == A) && (display_flag == 5) && (cursor == 5))
{
    if ((buffer[0] == ' ')&&(buffer[1] == ' ')&&
          (buffer[2] == ' ')&&(buffer[3] == ' ')&&
          (buffer[4] == ' ')&&(character == '_'))
    {
       button_pressed = NO_PRESS;
    }
    else choose_digit_display(5);
    short buffer_length = strlen(buffer);
    left_rotate(buffer, 1, buffer_length);
}



if ((button_pressed == A_LONG)&&(display_flag > 1)&&(cursor > 1)) // scroll - apasare lunga pe A
{
    if ((cursor < 6) && (strlen(message) < 6))
    {

       strncat(message, blank, 6 - cursor); // completeaza cu blank daca nu se introduc cel putin 6 caractere

    }
    if (strlen(message) > 6) rTrim(message);
    strncat(message, blank, 4); // add four space after message
    short message_length;
    button_pressed = DEFAULT;
    cursor = 6;
    index = 0;
    //strncat(message, blank, 4); // add four space after message
    message_length = strlen(message);
```

```
            while (button_pressed != D)
            {
               control_speed();
               strncpy(buffer, message, 6);
               left_rotate(message, 1, message_length);
            }
            clear();
            button_pressed = NO_PRESS;
            index = 0;
         }

         if (button_pressed == B) // scroll mesaj din EEPROM - apasare B
         {
            short message_length;
            button_pressed = DEFAULT;
            cursor = 6;
            index = 0;
            read_eeprom();
            //Rtrim(message);   //taie toate "spatiile" adunate la sfarsit
            //strncat(message, blank, 4); // add four space after message
            message_length = strlen(message);

            while (button_pressed != D)
            {
               control_speed();
               strncpy(buffer, message, 6);
               left_rotate(message, 1, message_length);
            }
            clear();
            button_pressed = NO_PRESS;
            index = 0;
         }

         if ((button_pressed == C)&&(display_flag > 1)&&(cursor > 1)) // store message in eeprom
         {
            rTrim(message);
            strncat(message, blank, 4); // add four space after message
            button_pressed = DEFAULT;
            save_in_eeprom();
            button_pressed = B;
            cursor = 6;
            index = 0;
         }

      }
   }
```