

Universitatea POLITEHNICA Bucuresti

Proiect 2

Echipa nr. 04

Studenti:

Mitrea Bogdan

Eftimie Albert

Constantinescu Adelina

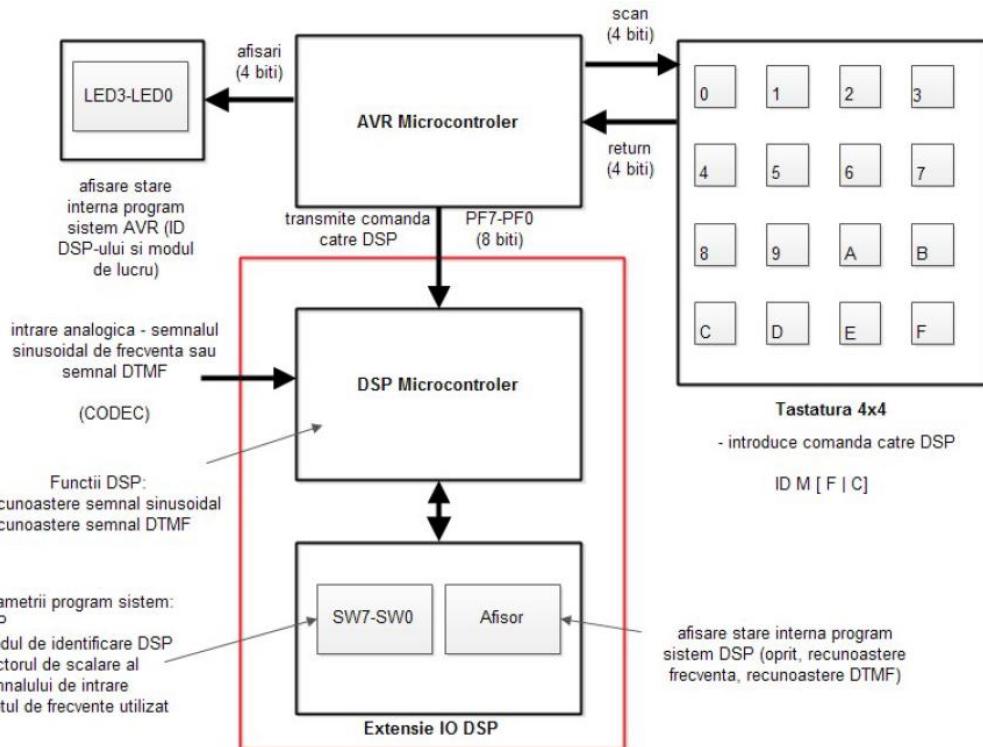
Grupa 432D

Profesor coordonator: Zoican Sorin

Bucuresti 2024

Cerinta de proiectare

Să se realizeze un sistem cu arhitectura din figura următoare cu următoarele specificații:



Sistemul este compus din două subsisteme (AVR și DSP) și are ca funcție testarea apariției unui semnal sinusoidal sau DTMF pe intrarea analogică.

Subsistemuul DSP citește semnalul analogic (semnal continuu) și determină dacă acest semnal este un semnal sinusoidal cu frecvența f_i ($i=0, 1, \dots, 7$) sau un semnal ce conține suma a două semnale sinusoidale pe frecvențele f_r și f_c cu $r, c = 0, 1, 2, 3$ (semnal Dual Tone MultiFrequency – DTMF). Se va afișa pe afișor codul frecvenței ($0, 1, 2, \dots, 7$) sau al semnalului DTMF ($0, 1, 2, \dots, E, F$)

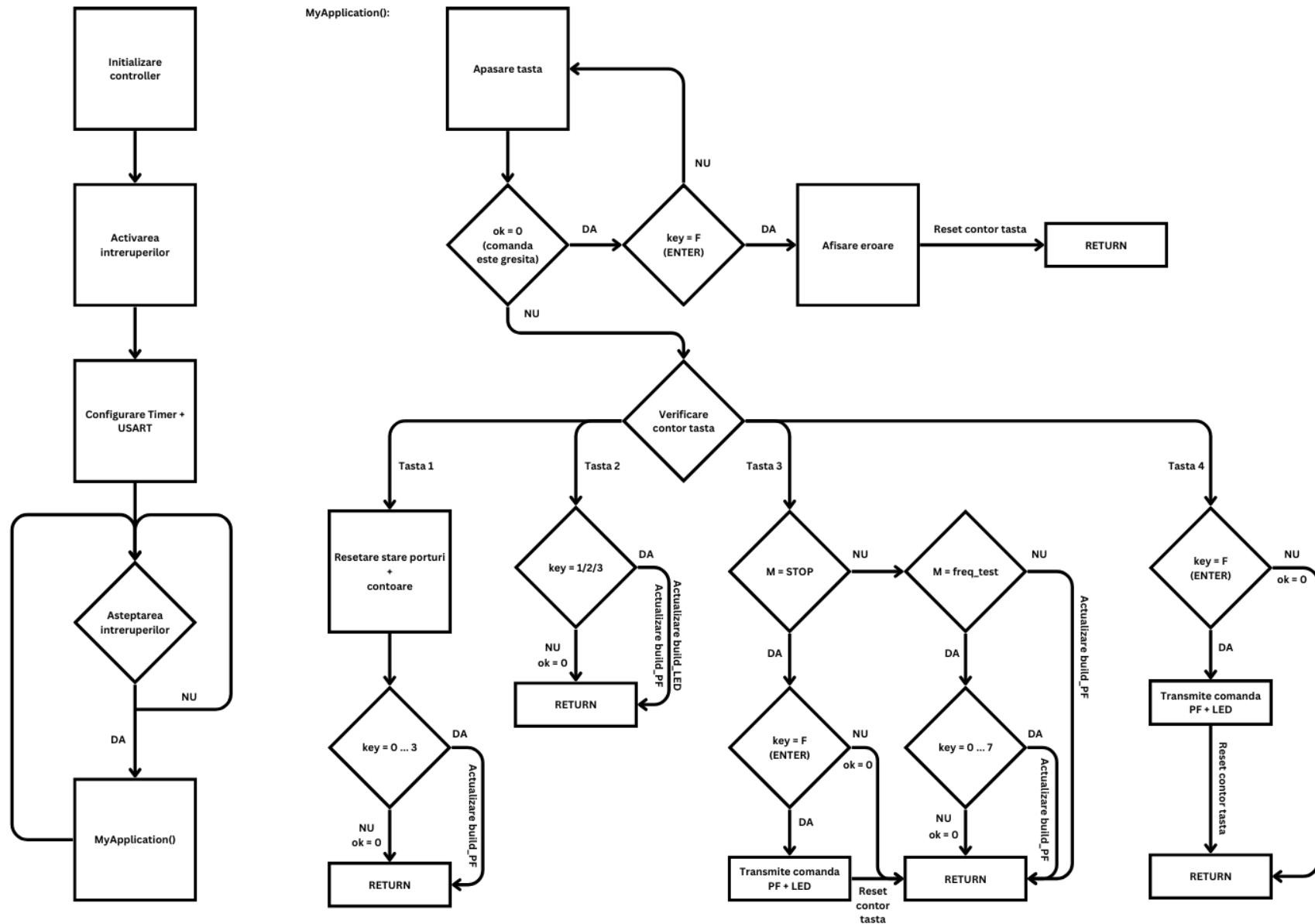
Subsistemuul AVR preia o comandă de la tastatura și o retransmite către DSP prin portul PF.

Comanda este de forma $ID, M, [F | C]$, unde ID este identificatorul sistemului DSP (valori 0, 1, 2 sau 3), M – este modul de lucru (valori 1 – testează frecvență, 2 – testează cod DTMF și 3 – stop DSP) F – frecvență testată (valori $0, 1, \dots, 7$) și C codul DTMF testat (valori $0, 1, \dots, 7$). Semnificația simbolurilor este: $[]$ – optional, $|$ – alegere.

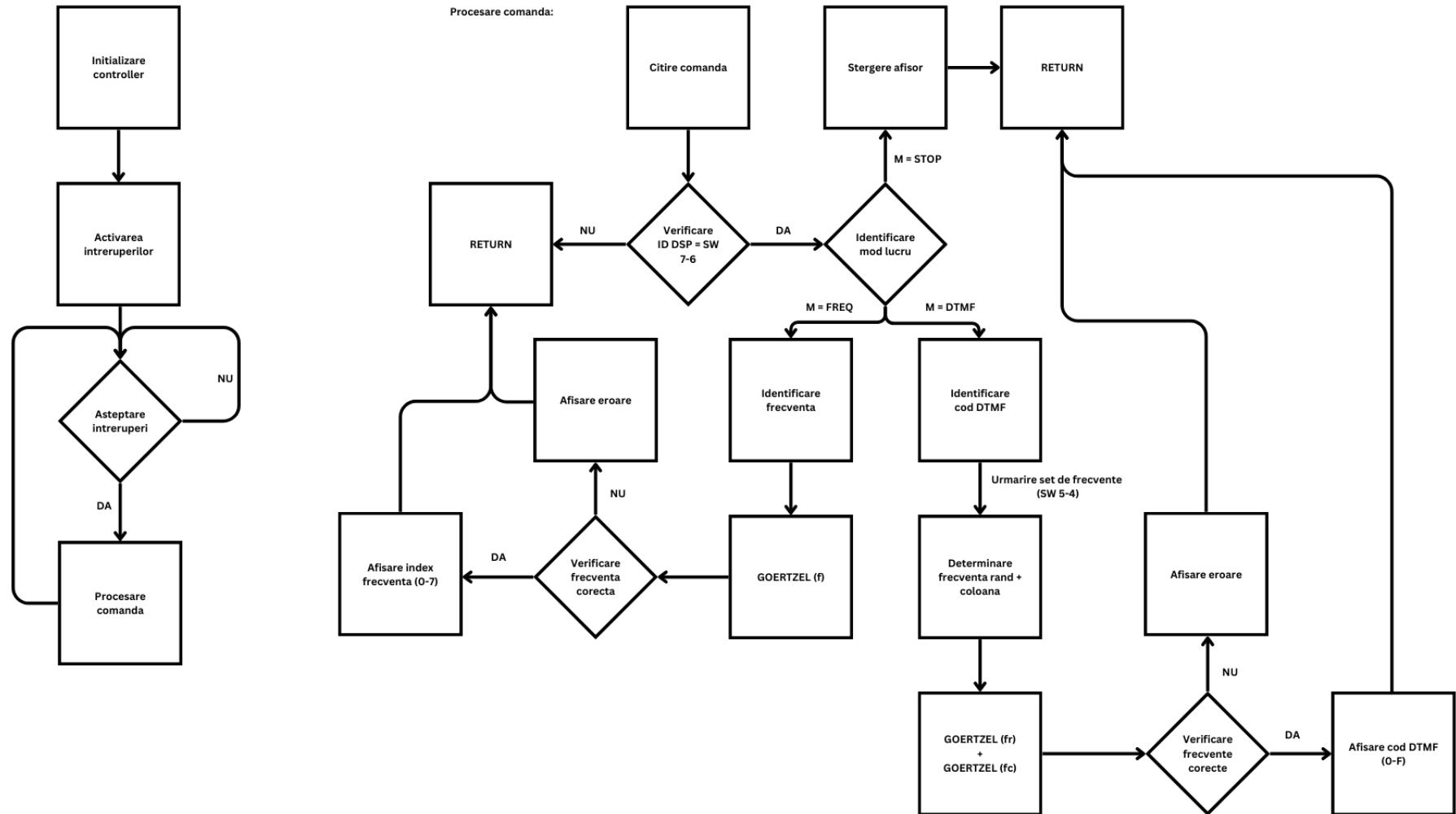
Ambele subsisteme își afișează starea proprie (pe LED3-LED0, respectiv pe un afișor cu 7 segmente – Afișor). Subsistemuul AVR utilizează un microcontroler ATMega164. Subsistemuul DSP are în componentă placa de evaluare EZ-Kit LITE ADSP2181 și o interfață de intrare ieșire (IO DSP).

Descrierea prelucrărilor

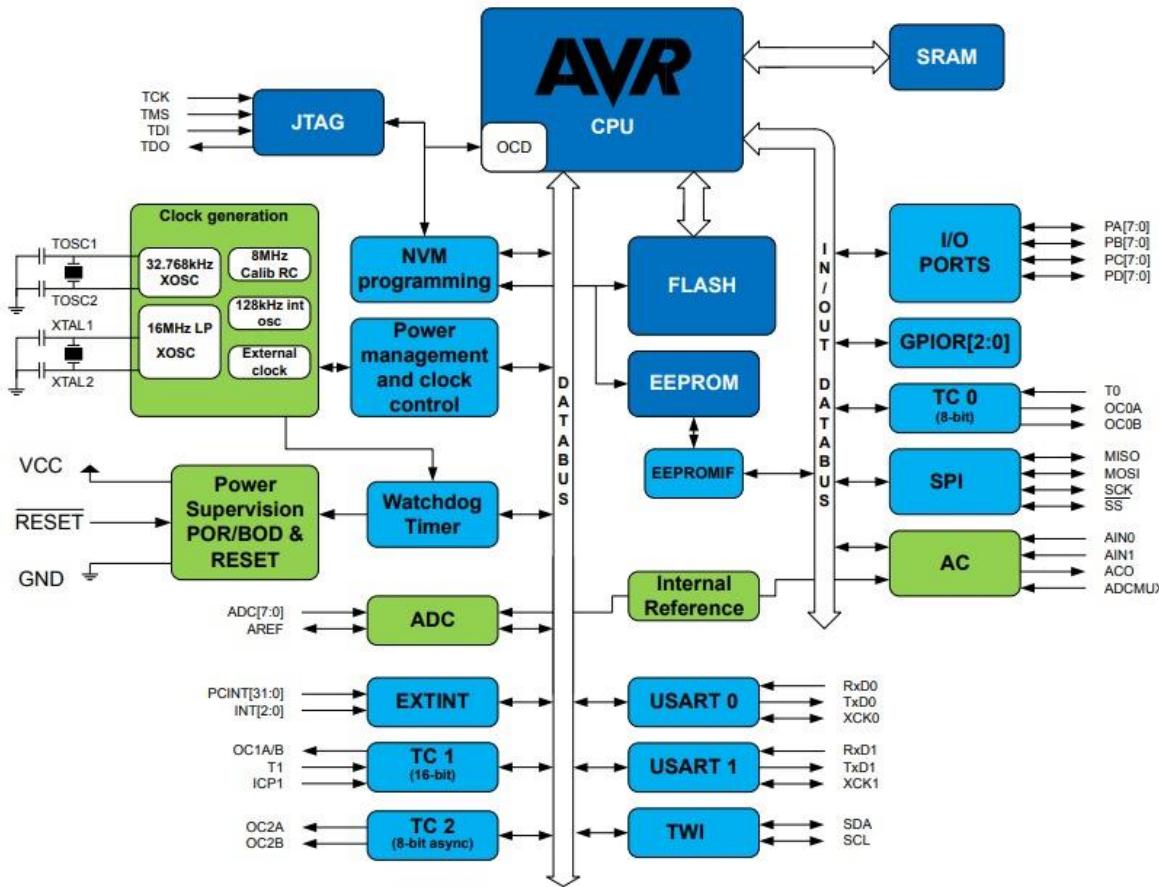
2.1. AVR:



2.2. DSP:



Schema bloc



În diagramă este ilustrată structura internă a unui microcontroller AVR, care reprezintă un tip de procesor compact, utilizat în domeniul electronic pentru controlul operațiilor, atât simple cât și complexe. Microcontroller-ul AVR integrează mai multe funcționalități esențiale într-un singur cip, inclusiv o unitate centrală de procesare (CPU), memorie pentru stocarea codului (FLASH) și setări (EEPROM), precum și memorie pentru execuția programelor (SRAM).

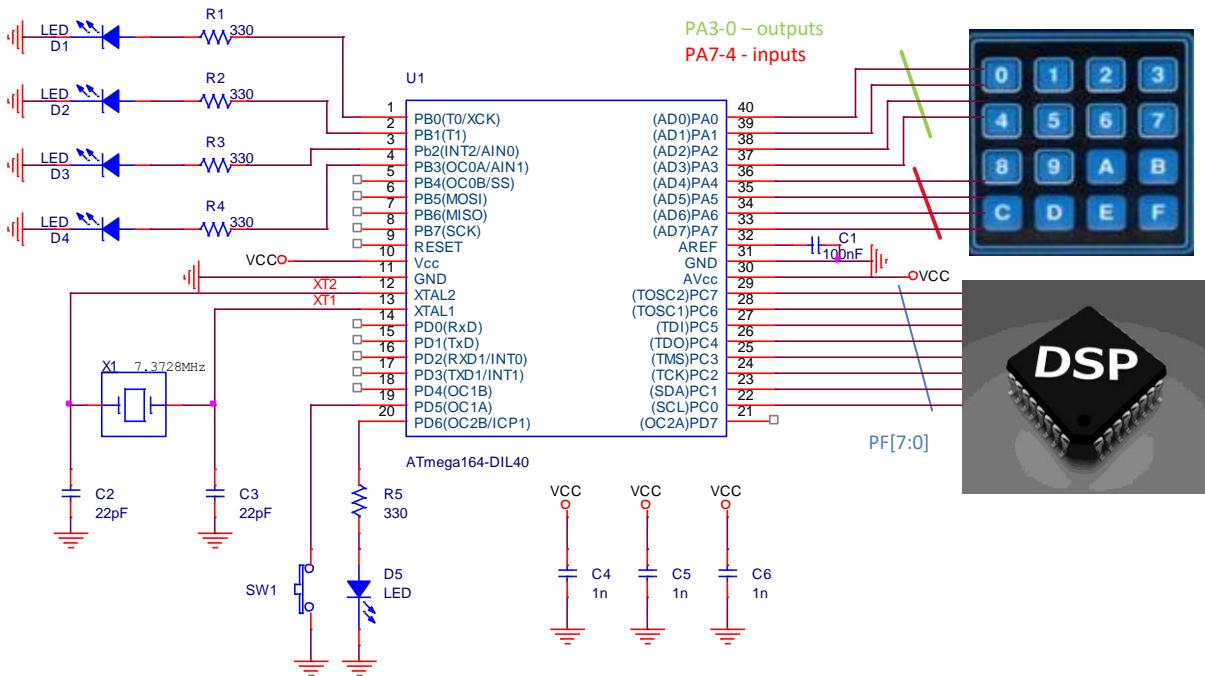
Acest microcontroller poate citi atât intrări analogice, cât și digitale, poate executa instrucțiuni bazate pe aceste intrări și poate transmite semnale către dispozitive externe sau alte circuite prin intermediul porturilor sale de intrare/ieșire (I/O). De asemenea, poate comunica cu alte dispozitive sau microcontrolere prin interfețe de comunicație precum USART, SPI și TWI.

Un sistem de management al energiei contribuie la optimizarea consumului de energie, iar diferitele cronometre și contoare permit măsurarea timpului sau a evenimentelor. Capabilitățile JTAG și OCD facilitează programarea și depanarea microcontroller-ului.

În esență, microcontroller-ul AVR reprezintă un sistem digital versatil care rulează un program încărcat în memoria sa și interacționează cu mediul extern printr-o varietate de metode de intrare și ieșire.

Schema electrică

AVR:



Centrul sistemului este reprezentat de microcontrolerul ATmega164, care constituie nucleul întregului dispozitiv. Acesta administrează toate intrările și ieșirile (I/O), procesează datele și controlează dispozitivele externe.

Patru LED-uri (D1-D4) sunt legate la microcontroler (pinii 1-4) prin intermediul rezistoarelor de 330 ohmi. Aceste LED-uri au rolul de a indica starea semnalelor emise de microcontroller, iar rezistoarele R1-R4 sunt folosite pentru a limita curentul care trece prin LED-uri, astfel încât să se evite deteriorarea acestora.

Pinii XTAL1 și XTAL2 (pinii 12 și 13) sunt conectați la un cristal oscilator extern și două condensatoare de 22pF. Împreună cu cristalul oscilator (X1), condensatoarele (C2 și C3) formează un circuit rezonant care stabilizează oscilația la o frecvență specifică. Ele contribuie la modelarea și stabilizarea semnalului de ceas, asigurând ca microcontrolerul să funcționeze la o frecvență constantă și corectă. Cristalul este fabricat dintr-un material piezoelectric, de obicei cuarț, care vibrează la o anumită frecvență în prezența unei anumite tensiuni.

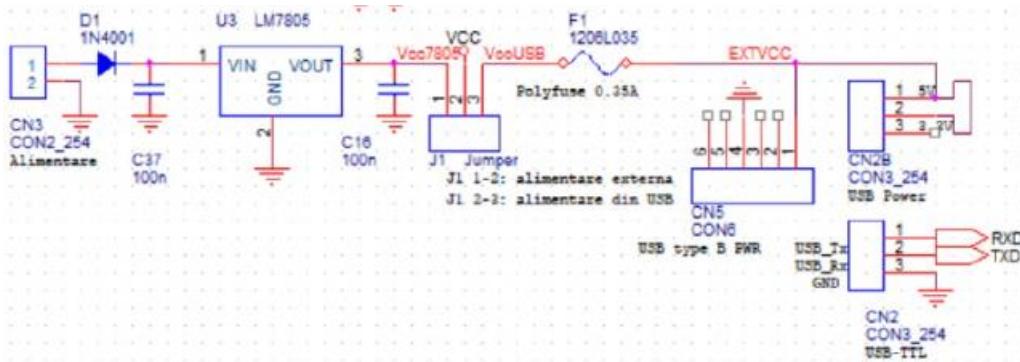
Butonul SW1 conectează pinul PD5 la masă (nivel logic '0') atunci când este apăsat. Pentru a asigura o stare logică definită pentru pinul PD5 atunci când butonul nu este apăsat, este necesar să se activeze o rezistență de pull-up internă prin intermediul unor instrucțiuni software. Astfel, în absența apăsării butonului, pinul PD5 va fi menținut la un nivel logic '1' datorită rezistenței de pull-up.

LED-ul este configurat pentru a se aprinde atunci când pinul PD6 al microcontrolerului este la nivel logic '1'. Rezistența R2 este folosită pentru a limita curentul care trece prin LED la aproximativ 10 mA. Este important de menționat că, în timp ce în multe configurații LED-urile sunt conectate cu anodul la sursa de tensiune și catodul la pinul circuitului digital, în cazul microcontrolerelor AVR, pinii de ieșire pot furniza curent indiferent de setarea lor la '0' sau '1'.

Condensatoarele C4 și C5 sunt condensatoare de decuplare și sunt plasate cât mai aproape posibil de circuitele care necesită o sursă de alimentare "curată". Ele filtrează zgomotul de pe liniile de alimentare, absorbând variațiile rapide de tensiune și eliberându-le lent, pentru a menține o tensiune de alimentare stabilă pentru componente sensibile.

Chiar dacă nu este esențial, C6 ajută la minimizarea riscului de resetări neintenționate ale microcontrolerului cauzate de fluctuațiile brusă și scurte ale tensiunii de alimentare, cunoscute sub numele de "glitch-uri". Prin filtrarea acestor perturbații, condensatorul contribuie la menținerea unui nivel stabil de tensiune pentru procesor, asigurând o funcționare continuă și fără erori.

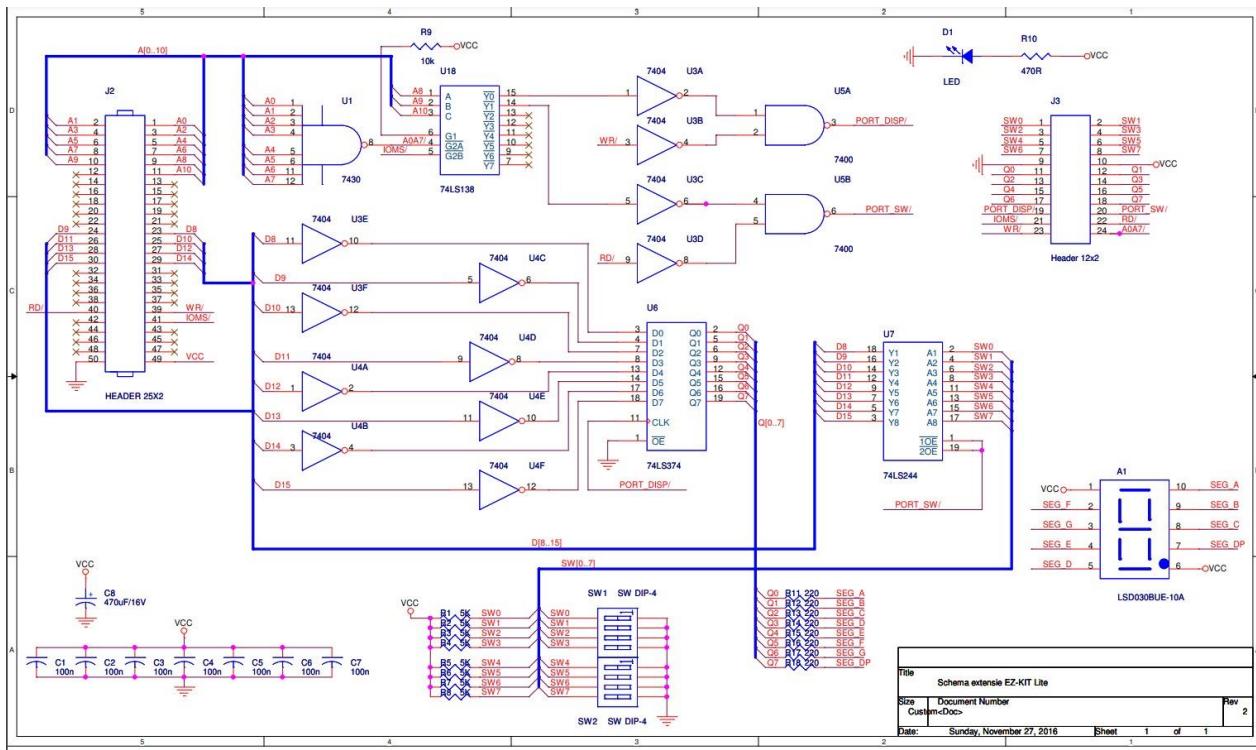
Pinii 22-29 sunt destinați conectării la DSP. La pinii 33-40, a fost adăugată o tastatură, care reprezintă o interfață de intrare ce permite utilizatorului să introducă comenzi sau date în microcontroler.



Această secțiune a schemei electronice descrie modul în care circuitul este alimentat și protejat împotriva condițiilor de alimentare neregulate sau incorecte. Jumperul J1 oferă flexibilitate în alegerea sursei de tensiune, iar componentele de protecție asigură integritatea și securitatea circuitului.

Schema din figura este deja rutată pe un PCB aşa cum este prezentat în figura 2. Fiecare pin liber al microcontrolerului este adus la un pad adjacente, de unde poate fi conectat, prin intermediul unui fir, la componente adiționale ce pot fi integrate în zona de paduri libere (de tip placă de test).

Extensia IO DSP:



Circuite Integrate (IC-uri)

7404: NOT gate.

7430: NAND gate cu 8 intrări. Poarta produce o ieșire 'low' doar atunci când toate intrările sunt 'high'.

74LS138: Este un decodor 3-la-8 linii, care transformă trei intrări de selecție în opt ieșiri, cu doar una activă ('low') în orice moment, bazat pe codul de intrare.

74LS374: Registru cu octet flip-flop cu latch-uri. El este folosit pentru a stoca date și poate fi folosit ca memorie temporară sau pentru sincronizarea transferurilor de date.

74LS244: Buffer.

LED-uri și Afisaje

LED: Afisează starea atunci când este alimentat.

Afișor LED: Utilizat pentru a arăta informații utilizatorului, cum ar fi cifre sau caractere.

Comutatoare

SW0 – SW7: Permit utilizatorului să interacționeze cu circuitul, schimbând stările logice manual.

Functia MyApplication():

```
void MyApplication (void)
{
    key = read_keyboard();
    if(key != -1) {

        if(ok == 0)
        {
            if(key == 0x0F) {
                cnt_key = 0;
                ok = 1;
                write_LED(error_LED);
                return;
            }
            if(key != 0x0F) return;
        }

        switch (cnt_key)
        {
            case 0:
            {
                build_PF = 0;
                build_LED = 0;
                write_PF(0); // sets PF to initial command
                write_LED(0); // sets LEDs to LOW
                if(key == 0x0F) {
                    cnt_key = 0;
                    ok = 1;
                    write_LED(error_LED);
                    return;
                }
                if(key > 0x3) ok = 0;
                else {
                    cnt_key++;
                    build_PF_1(key);
                }
            }
            break;

            case 1:
            {
                if(key < 0x1 || key > 0x3) ok = 0;
                else switch(key)
                {
                    case 1:
                    {
                        cnt_key++;
                        build_LED = freq_test_LED;
                        build_PF_2(key);
                    }
                    break;

                    case 2:
                    {
                        cnt_key++;
                        build_LED = DTMF_test_LED;
                    }
                    break;
                }
            }
            break;
        }
    }
}
```

```
        build_PF_2(key);
    }
}
break;

case 3:
{
    cnt_key++;
    build_LED = stop_LED;
    build_PF_2(key);
}
break;
}

break;

case 2:
{
    if(build_LED == stop_LED)
    {
        if(key != 0x0F) ok = 0;
        else {
            cnt_key = 0;
            write_PF(build_PF);
            write_LED(build_LED);
        }
    }
    else if(build_LED == freq_test_LED)
    {
        if(key > 0x7) ok = 0;
        else {
            cnt_key++;
            build_PF_3(key);
        }
    }
    else if(build_LED == DTMF_test_LED)
    {
        cnt_key++;
        build_PF_3(key);
    }
}
break;

case 3:
{
    if(key != 0x0F) ok = 0;
    else {
        cnt_key = 0;
        write_PF(build_PF);
        write_LED(build_LED);
    }
}
break;
}
```

Initializari variabile:

```
char build_PF = 0x00; // PF to be built and then transmitted
char build_LED = 0x00; // LED to be built and then transmitted
char error_LED = 0x8;
char freq_test_LED = 0x4;
char DTMF_test_LED = 0x2;
char stop_LED = 0x1;
```

Functii asociate codului MyApplication():

```
void write_LED(char a)
{
// write PORTB bits 3-0 with a 4 bits value a3-a0
char val;
val=a & 0x0F;
PORTB=(PORTB & 0xF0) | val;
}

void write_PF(char a)
{
// write PORTC bits 7-0 with a 8 bits value a7-a0
PORTC = a;
}

void build_PF_1(char a)
{
// write x bits 7-6 with a 2 bits value a1-a0
char val;
val = a & 0x03;
build_PF = (build_PF & 0x3F) | (val << 6);
}

void build_PF_2(char a)
{
// write x bits 5-4 with a 2 bits value a1-a0
char val;
val = a & 0x03;
build_PF = (build_PF & 0xCF) | (val << 4);
}

void build_PF_3(char a)
{
// write x bits 3-0 with a 4 bits value a3-a0
char val;
val=a & 0x0F;
build_PF = (build_PF & 0xF0) | val;
}
```

Fisierul test_ext:

```
#include "def2181.h"

#define PORT_OUT 0xFF
#define PORT_IN 0x1FF

#define f_sample 8000
#define N      200

/////////
.....
.....
.....
/////////

.SECTION/DM          data1;
.var    stat_flag;
.var    PF_input;
.var    PF_output;
.var/circ Q1Q2_buff[2];           // ultimele 2 valori ale lui Q
.var    port_in;                 // portul de intrare
.var in_sample;                // esantionul de intrare curent
.var    countN;                  // numara esantioanele 1, 2, 3, ..., N
.var    min_tone_level=0x0100; // "tone-present" mnsqr level
.var mnsqr;                    // amplitudinea (in 1.15) mnsqr Goertzel
.var freq_OK; // frecventa existenta (1) sau nu (0)

.var index_set;
.var SW7_6;
.var SW5_4;
.var SW3_0;
.var PF7_6;
.var PF5_4;
.var PF3_0;
.var OK;
.var index_freq1;
.var ok_afisor;
.var freq2_OK;
.var dtmf_freq_cnt;

.section/pm pm_da;

.var/circ coefs0[8]=0x7E6D, 0x7AEB, 0x7579, 0x6E2D, 0x47F2, 0x3A1C, 0x2B5C, 0x1BEC;
.var/circ coefs1[8]=0x7CEB, 0x786F, 0x720D, 0x69DE, 0x4E74, 0x4128, 0x32D6, 0x23B6;
.var/circ coefs2[8]=0x7E6D, 0x7AEB, 0x7579, 0x6E2D, 0x4E74, 0x4128, 0x32D6, 0x23B6;
.var/circ coefs3[8]=0x7CEB, 0x786F, 0x720D, 0x69DE, 0x47F2, 0x3A1C, 0x2B5C, 0x1BEC;
```

Gestionarea intreruperilor pe IRQ2:

input_samples:

```
ena sec_reg;           /* use shadow register bank */

sr1 = dm (rx_buf + 2); /* get new sample from SPORT0 (from codec) */
// aici dau pe streams rx_buf+2 manual

nofilt: /*sr=ashift sr1 by -1 (hi);*/ /* save the audience's ears from damage */
mr1=sr1;

si=IO(PORT_IN);
sr=lshift si by -6 (hi);
ax0=sr1; // ax0=SW7-6
dm(SW7_6)=ax0;

ax1=si;
ay1=0x0030;
ar=ax1 and ay1;
sr=lshift ar by -4 (hi);
ay0=sr1; // ay0=SW5-4
dm(SW5_4)=ay0;

ay1=0x000F;
ar=ax1 and ay1;
ax1=ar; // ax1=SW3-0
dm(SW3_0)=ax1;

// iau fiecare set de biti si ii stochez undeva ***FACUT***
//IO(PORT_OUT)=si; aici '=' si' e de test
```

output:

```
dm (tx_buf + 1) = mr1; /* filtered output to SPORT (to spkr) */
dm (tx_buf + 2) = mr1; /* filtered output to SPORT (to spkr) */

// test PF
// PF inputs 0-7
    si=dm(Prog_Flag_Data);
    dm(PF_input)=si;

    sr=lshift si by -6 (hi);
    mx0=sr1; // PF7-6
    dm(PF7_6)=mx0;

    ay1=0x0030;
    sr0=si;
    ar=sr0 and ay1;
    sr=lshift ar by -4 (hi);
```

```

my0=sr1; // PF5-4
dm(PF5_4)=my0;

ay1=0x000F;
sr0=si;
ar=sr0 and ay1;
mx1=ar; // PF3-0
dm(PF3_0)=mx1;

l5=8; //alegere set de frecvente
ar=dm(SW5_4);
ar=ar-0;
if eq jump set_0;
ar=ar-1;
if eq jump set_1;
ar=ar-1;
if eq jump set_2;
ar=ar-1;
if eq jump set_3;
// final alegere set frecvente

set_0: i5=coefs0;
jump verif_id;
set_1: i5=coefs1;
jump verif_id;
set_2: i5=coefs2;
jump verif_id;
set_3: i5=coefs3;
jump verif_id;

verif_id:
ax0=dm(SW7_6);
ay0=dm(PF7_6);
ar=ax0-ay0;
if eq jump id_ok;
if ne rti;

id_ok:
ax0=dm(PF5_4);
ay0=1;
ar=ax0-ay0;
if eq jump caz_freq;
ay0=2;
ar=ax0-ay0;
if eq jump caz_dtmf;
ay0=3;
ar=ax0-ay0;
if eq jump caz_stop;
rti;

caz_freq:
ax0=dm(PF3_0);
dm(index_freq1)=ax0;
ar=dm(OK);
ar=ar-2;
if eq jump final_freq;
jump goertzel;

caz_dtmf:
ax0=dm(dtmf_freq_cnt);
ar=ax0-2;
if eq jump caz_dtmf_2;
si=dm(PF3_0);
sr=ashift si by -2 (hi);
dm(index_freq1)=sr1;
ar=dm(OK);
ar=ar-2;
If eq jump final_dtmf;
jump goertzel;

caz_dtmf_2:
sr0=dm(PF3_0);
ay1=0x0003;
ar=sr0 and ay1;
ar=ar+4;
dm(index_freq1)=ar;
ar=dm(OK);
ar=ar-2;
if eq jump final_dtmf_2;
jump goertzel;

caz_stop:
ax0=0x00;
IO(PORT_OUT)=ax0;
rti;

final_freq:
ax0=dm(freq_OK);
ar=ax0-1;
if ne call drawErr;
if eq call drawLetter;
ax0=0;
dm(OK)=ax0;
rti;

drawLetter:
ax0=dm(PF3_0);
ar=ax0-0;
if eq call draw0;
ar=ax0-1;
if eq call draw1;
ax0=ar;

```

```

ar=ax0-1;
if eq call draw2;
ax0=ar;
ar=ax0-1;
if eq call draw3;
ax0=ar;
ar=ax0-1;
if eq call draw4;
ax0=ar;
ar=ax0-1;
if eq call draw5;
ax0=ar;
ar=ax0-1;
if eq call draw6;
ax0=ar;
ar=ax0-1;
if eq call draw7;
rts;
afisor_modif:
dm(ok_afisor)=ax0;
rts;

final_dtmf:
ar=dm(freq_OK);
dm(freq2_OK)=ar;
ar=2;
dm(dtmf_freq_cnt)=ar;
ax0=0;
dm(OK)=ax0;
rti;

final_dtmf_2:
ax0=dm(freq_OK);
ay0=dm(freq2_OK);
ar=ax0 and ay0;
ar=ar-0;
if eq call drawErr_dtmf;
if ne call drawLetter_dtmf;
ax0=0;
dm(OK)=ax0;
ax0=1;
dm(dtmf_freq_cnt)=ax0;
rti;

drawLetter_dtmf:
ax0=dm(PF3_0);
ar=ax0-0;
if eq call draw0;
ar=ax0-1;
if eq call draw1;
ax0=ar;

ar=ax0-1;
if eq call draw2;
ax0=ar;
ar=ax0-1;
if eq call draw3;
ax0=ar;
ar=ax0-1;
if eq call draw4;
ax0=ar;
ar=ax0-1;
if eq call draw5;
ax0=ar;
ar=ax0-1;
if eq call draw6;
ax0=ar;
ar=ax0-1;
if eq call draw7;
ax0=ar;
ar=ax0-1;
if eq call draw8;
ax0=ar;
ar=ax0-1;
if eq call draw9;
ax0=ar;
ar=ax0-1;
if eq call drawA;
ax0=ar;
ar=ax0-1;
if eq call drawB;
ax0=ar;
ar=ax0-1;
if eq call drawC;
ax0=ar;
ar=ax0-1;
if eq call drawD;
ax0=ar;
ar=ax0-1;
if eq call drawE;
ax0=ar;
ar=ax0-1;
if eq call drawF;
call drawPoint;
rts;

goertzel:
ar=dm(OK);
ar=ar-0;
if eq jump begin;
if gt jump processing;
rti;

```

Codul Goertzel:

```
//-----
//----- PROGRAMUL PRINCIPAL -----
//-----

begin: call setup;
call restart;
    i0=Q1Q2_buff;
    m5=dm(index_freq1);
    modify(i5,m5); // i5 pointer catre coeficientul asociat frecventei testate
    dm(index_set)=i5;
    ax0=1;
    dm(OK)=ax0;
    //IMASK = 0x200; // valideaza intreruperea IRQ2, deja validata

//stop : jump stop;           // asteapta intreruperi, deja e in intrerupere

//----- PROCESAREA UNUI ESANTION -----
//
processing:
i5=dm(index_set);
si=mr1; // citeste esantionul curent

// det_freq function

// input:
// si - esantionul de intrare
// scale - factorul de scalare
// countN - numarul de esantioane
// min_tone_level - apmlitudinea minima a frecventei detectate
// i0 - pointer la bufferul Q (ultimele 2 valori ale lui Q)
// l0 = 2
// m0 = 1, m1 = -1
// i5 - pointer la coeficientul asociuat frecventei testate
// l5 = 8
// m4 = 0

// output
// freq_OK = 1 - frecventa a fost detectata
//           0 - frecventa nu a fost detectata
call det_freq;

rti;

//-----

det_freq:

    ax1=dm(SW3_0);
```

```

loopScale:
    ar=ax1-1;
    ax1=ar;
    sr=ashift si by -1 (hi);
    si=sr1;
    if gt jump loopScale;
    dm(in_sample)=sr1; // stocarea esantionului de intrare , scalat

//----- DECREMENTAREA CONTORULUI DE ESANTIOANE -----
//  

decN:
    ay0=dm(countN);
    ar=ay0-1;
    dm(countN)=ar;
    if lt jump skip_backs;

//----- F A Z A   F E E D B A C K -----
//  

feedback:
    ay1=dm(in_sample); // extrage esantionul la intrare AY1=1.15
    mx0=dm(i0,m0), my0=pm(i5,m4); //extrage Q1 si COEF Q1=1.15, COEF=2.14
    mr=mx0*my0(rnd), ay0=dm(i0,m1); //inmulteste, get Q2 MR=2.30, Q2=1.15
    sr=ashift mr1 by 1 (hi); //schimba 2.30 in 1.15
    ar=sr1-ay0; //Q1*COEF - Q2 AR=1.15
    ar=ar+ay1; //Q1*COEF - Q2 + intrarea AR=1.15
    dm(i0,m0)=ar; //rezultatul = nou Q1
    dm(i0,m0)=mx0; //vechiul Q1 = nou Q2
    jump end;;

//----- F A Z A   F E E D B A C K   E S T E G A T A -----
//  

skip_backs:
    call feedforward;
    call test_and_output;
    call restart;

end:
    rts;

// functii de initializare si reinitializare

setup:
    l0 = 2;
    l5 = 8;
    m0 = 1;
    m1 = -1;
    m4 = 0;

```

```

rts;

restart: i0=Q1Q2_buff;
          cntr=2;
          do zloop until ce;
zloop:      dm(i0,m0)=0;
          ax0=N;
          dm(countN)=ax0;
          //ax0=0;
          //dm(freq_OK)=ax0;
          rts;

//%%%%%%%%%%%%% F A Z A  F E E D F O R W A R D %%%%%%%%%%%%%%
//feedforward:
          mx0=dm(i0,m0);      // extrage doua copii Q1      1.15
          my0=mx0;
          mx1=dm(i0,m0);      // extrage doua copii Q2      1.15
          my1=mx1;
          ar=pm(i5,m4);      // extrage COEF           2.14
          mr=0;
          mf=mx0*my1(rnd);    // Q1*Q2             1.15
          mr=mr-ar*mf(rnd);   // -Q1*Q2*COEF       2.14
          sr=ashift mr1 by 1 (hi); // 2.14 -> 1.15 format conv.  1.15
          mr=0;
          mr1=sr1;
          mr=mr+mx0*my0(ss);  // Q1*Q1 + -Q1*Q2*COEF  1.15
          mr=mr+mx1*my1(rnd); // Q1*Q1 + Q2*Q2 + -Q1*Q2*COEF 1.15
          dm(mnsqr)=mr1;      // socheaza in mnsqr 1.15
          rts;

//%%%%% Testarea nivelului %%%%%%
//test_and_output:

          ar=dm(mnsqr);
          ay0=dm(min_tone_level);
          ar=ar-ay0;
          if lt jump no_freq;
          si=1;
          jump rez;
no_freq:
          si=0;
rez:
          dm(freq_OK)=si;
          ax0=2;
          dm(OK)=ax0;
          rts;

```

Label-uri pentru afisarea simbolului pe afisor:

```
draw0: ax0=0x3F;  
        IO(PORT_OUT)=ax0;  
        rts;  
draw1: ax0=0x06;  
        IO(PORT_OUT)=ax0;  
        rts;  
draw2: ax0=0x5B;  
        IO(PORT_OUT)=ax0;  
        rts;  
draw3: ax0=0x4F;  
        IO(PORT_OUT)=ax0;  
        rts;  
draw4: ax0=0x66;  
        IO(PORT_OUT)=ax0;  
        rts;  
draw5: ax0=0x6D;  
        IO(PORT_OUT)=ax0;  
        rts;  
draw6: ax0=0x7D;  
        IO(PORT_OUT)=ax0;  
        rts;  
draw7: ax0=0x07;  
        IO(PORT_OUT)=ax0;  
        rts;  
draw8: ax0=0x7F;  
        IO(PORT_OUT)=ax0;  
        rts;  
draw9: ax0=0x6F;  
        IO(PORT_OUT)=ax0;  
        rts;  
drawA: ax0=0x77;  
        IO(PORT_OUT)=ax0;  
        rts;  
  
drawB: ax0=0x7C;  
        IO(PORT_OUT)=ax0;  
        rts;  
drawC: ax0=0x39;  
        IO(PORT_OUT)=ax0;  
        rts;  
drawD: ax0=0x5E;  
        IO(PORT_OUT)=ax0;  
        rts;  
drawE: ax0=0x79;  
        IO(PORT_OUT)=ax0;  
        rts;  
drawF: ax0=0x71;  
        IO(PORT_OUT)=ax0;  
        rts;  
drawPoint:  
        ax0=IO(PORT_OUT);  
        ay0=0x80;  
        ar=ax0 or ay0;  
        IO(PORT_OUT)=ar;  
        rts;  
drawErr:  
        ax0=0x40;  
        IO(PORT_OUT)=ax0;  
        rts;  
drawErr_dtmf:  
        ax0=0xC0;  
        IO(PORT_OUT)=ax0;  
        rts;
```

Explicatii detaliate cod DSP (in gestionarea intreruperilor):

Initial, simulam pe streamul de intrare un semnal sinusoidal / semnal DTMF, iar la adresa rx_buf+2 vom primi esantioane noi ale acelui semnal la fiecare interrupt.

In prima faza, am creat locatii de memorie in Data Memory(DM), corespunzatoare fiecarui set de biti necesar pentru prelucrarea datelor in continuare: SW7_6, SW5_4, SW3_0, respectiv PF7_6, PF5_4, PF3_0.

Apoi, cu cele 4 seturi a cate 8 coeficienti definite in Program Memory(PM), am aflat indexul primului element din setul corect, folosind variabila SW5_4.

Am verificat ca ID_PF = ID_DSP. In caz contrar, seiese din interrupt, si nu se intampla nimic in continuare, afisorul ramanand in starea precedenta.

Am folosit variabila PF5_4 pentru a afla modul de lucru: 1-testare frecventa, 2-testare DTMF, 3-stop. Daca PF5_4=0, nu se intampla nimic (ar fi cazul doar daca nu se trimit corect codul de la tastatura, caz in care PF=0x00).

MODUL 1 – Testare frecventa:

Se stocheaza indexul care se va aduna la adresa primei frecvenete din set, apoi se verifica mereu daca variabila OK=2 (aceasta devine 2 la finalul intregului algoritm Goertzel, dupa stabilirea freq_OK). Apoi, se intra in algoritmul Goertzel pentru a prelucra esantionul curent.

NOTA: In label-ul “begin” se intra o singura data, la finalul caruia OK devine 1, pentru a evita o urmatoare intrare in begin si reinitializarea eronata a countN sau alte variabile.

Algoritmul Goertzel este cel obisnuit. Dupa 200 de esantioane, se trece la etapa de feedforward, si la stabilirea freq_OK.

La urmatoarea intrerupere, se intra in label-ul final_freq, unde se verifica daca frecventa este cea corecta, si se afiseaza fie “-” pe afisor (freq_OK=0), fie un numar 0-7 (freq_OK=1).

MODUL 2 – Testare DTMF:

Se procedeaza similar cu modul 1, singura diferenta fiind ca se asteapta finalizarea a 2 cicluri Goertzel, spre care se trimit indecsi diferiti, in functie de stadiul in care ne aflam: mai intai se trimit frecvenete de linie (PF3_0 >> 2), apoi se trimit frecvenete de coloana (PF1_0).

La final, se verifica daca ambele freq_OK au fost 1, caz in care se afiseaza pe display simbolul 0-F corespunzator, insotit de “.”, pentru a diferentia modul DTMF de cel de frecventa.

MODUL 3 – STOP:

Acesta este cel mai simplu de manipulat caz, fiind necesara doar stingerea tuturor LED-urilor de pe afisor.

Simulari

AVR:

PORTEA: Tastatura

PORTEB3-0: Starea celor 4 LED-uri.

PORTEC: Starea PF

PORTEC7-6: ID

PORTEC5-4: M

PORTEC3-0: frecventa testata | cod DTMF

- Starea porturilor dupa initializare:

The screenshot shows the AVR Studio interface with two main windows. The left window is the Processor view, displaying assembly code for a timer interrupt and the main loop. The right window is the I/O View, showing the state of various ports and pins. The I/O View table is as follows:

Name	Address	Value	Bits
ANALOG_COMPARATOR			
USART0			
PORTA	0x02(0x22)	0xFF	████████████████
PINA	0x01(0x21)	0x0F	□□□□□□□□
PORTB	0x05(0x25)	0x00	□□□□□□□□
DDRB	0x04(0x24)	0x0F	████████████
PINB	0x03(0x23)	0x00	□□□□□□□□
PORTC	0x08(0x28)	0x00	□□□□□□□□
DDRC	0x07(0x27)	0xF	███████████
PINC	0x06(0x26)	0x00	□□□□□□□□
PORTD	0x0B(0x2B)	0x60	□████████□
DDRD	0x0A(0x2A)	0x50	□□□□□□□□
PIND	0x09(0x29)	0x42	□□□□□□□□
TIMER_COUNTER_0			
TIMER_COUNTER_2			
WATCHDOG			
JTAG			
BOOT_LOAD			
EXTERNAL_INTERRUPT			
AD_CONVERTER			
TIMER_COUNTER_1			
EEPROM			
SPI			
TWI			
USART1			
CPU			

- Starea porturilor după introducerea comenzi “2 3 F” (modul STOP):

AVR Studio - [C:\Users\Bogdan\Desktop\proiect nou avr\main.c]

Processor

Program Counter	0x00017D
Stack Pointer	0x04FF
X pointer	0x0216
Y pointer	0x2000
Z pointer	0x0000
Cycle Counter	1208329
Frequency	4.0000 MHz
Stop Watch	302082.25 us
SREG	0x80

Registers

R00	0x01
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x03
R07	0x00
R08	0x00
R09	0x0F
R10	0x01
R11	0xB0
R12	0x04
R13	0x08
R14	0x00

Code:

```

cnt_key = 0;
ok = 1;

while(TRUE)
{
    /*

        vdogtrig();           // call often else processor will reset
        if(rx_counter0)      // if a character is available on serial port USART0
        {
            temp = getchar();
            if(temp == '?')
                printf("\r\nSwVersion:%d.%d\r\n", SW_VERSION/10, SW_VERSION%10);
            else
                putchar(temp+1); // echo back the character + 1 ("a" becomes "b", etc)
        }

        if(SW1 == 0)          // pressed
        {
            delay_ms(30);   // debounce switch
            if(SW1 == 0)
            {
                while(SW1==0) // LED will blink slow or fast
                    vdogtrig(); // wait for release
                // alternate between values and values/4 for OCRA1 register
                // 018E H / 4 = 0061 H
                // new frequency = old frequency * 4
                if(OCRAH == 0x01)
                    {TCNT1H=0; TCNT1L=0; OCRAH = 0x00; OCRA1L = 0x61;}
                else
                    {TCNT1H=0; TCNT1L=0; OCRAH = 0x01; OCRA1L = 0x86;}
            }
        }
    }
}

```

I/O View

Name	Address	Value	Bits
ANALOG_COMPARATOR			
USART0			
PORTA	0x02(0x22)	0xF7	██████████
DDRA	0x01(0x21)	0x0F	███
PIN	0x00(0x20)	0x77	██████████
PORTB	0x05(0x25)	0x01	██
DDRB	0x04(0x24)	0x0F	███
PINB	0x03(0x23)	0x01	██
PORTC	0x08(0x28)	0x80	██████████
DDRC	0x07(0x27)	0xFF	██████████
PINC	0x06(0x26)	0x80	██████████
PORTD	0x0B(0x2B)	0x60	██████████
DDRD	0x0A(0x2A)	0x50	██████████
PIND	0x09(0x29)	0x42	██████████
TIMER_COUNTER_0			
TIMER_COUNTER_2			
WATCHDOG			
JTAG			
BOOT_LOAD			
EXTERNAL_INTERRUPT			
AD_CONVERTER			
TIMER_COUNTER_1			
EPPROM			
SPI			
TWI			
USART1			
CPU			

PORTE a luat valoarea 0001, semnificand ca s-a aprins ultimul LED, iar pe PORTC s-a scris comanda asociata.

- Starea porturilor după introducerea comenzi “1 2 E F” (modul DTMF):

AVR Studio - [C:\Users\Bogdan\Desktop\proiect nou avr\main.c]

Processor

Program Counter	0x000169
Stack Pointer	0x04FD
X pointer	0x0008
Y pointer	0x01F4
Z pointer	0x0008
Cycle Counter	3209822
Frequency	4.0000 MHz
Stop Watch	802455.50 us
SREG	0x80

Registers

R00	0x08
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x08
R07	0x00
R08	0x00
R09	0x0F
R10	0x02
R11	0x6E
R12	0x04
R13	0x08
R14	0x00

Code:

```

cnt=(cnt+1)%5;
if (cnt==0) LED1 = ~LED1; // invert LED
MyApplication();

/* main function of program */
void main (void)
{
unsigned char temp,i;

Init_initController(); // this must be the first "init" action/call!
#asm("sei") // enable interrupts
LED1 = 1; // initial state, will be changed by timer 1

cnt_key = 0;
ok = 1;

while(TRUE)
{
    /*

        vdogtrig();           // call often else processor will reset
        if(rx_counter0)      // if a character is available on serial port USART0
        {
            temp = getchar();
            if(temp == '?')
                printf("\r\nSwVersion:%d.%d\r\n", SW_VERSION/10, SW_VERSION%10);
            else
                putchar(temp+1); // echo back the character + 1 ("a" becomes "b", etc)
        }

        if(SW1 == 0)          // pressed
        {
            delay_ms(30);   // debounce switch
            if(SW1 == 0)
            {
                while(SW1==0) // LED will blink slow or fast
                    vdogtrig(); // wait for release
                // alternate between values and values/4 for OCRA1 register
                // 018E H / 4 = 0061 H
                // new frequency = old frequency * 4
                if(OCRAH == 0x01)
                    {TCNT1H=0; TCNT1L=0; OCRAH = 0x00; OCRA1L = 0x61;}
                else
                    {TCNT1H=0; TCNT1L=0; OCRAH = 0x01; OCRA1L = 0x86;}
            }
        }
    }
}

```

I/O View

Name	Address	Value	Bits
ANALOG_COMPARATOR			
USART0			
PORTA	0x02(0x22)	0xF7	██████████
DDRA	0x01(0x21)	0x0F	███
PIN	0x00(0x20)	0x77	██████████
PORTB	0x05(0x25)	0x02	██
DDRB	0x04(0x24)	0x0F	███
PINB	0x03(0x23)	0x02	██
PORTC	0x08(0x28)	0x6E	██████████
DDRC	0x07(0x27)	0xFF	██████████
PINC	0x06(0x26)	0x6E	██████████
PORTD	0x0B(0x2B)	0x60	██████████
DDRD	0x0A(0x2A)	0x50	██████████
PIND	0x09(0x29)	0x42	██████████
TIMER_COUNTER_0			
TIMER_COUNTER_2			
WATCHDOG			
JTAG			
BOOT_LOAD			
EXTERNAL_INTERRUPT			
AD_CONVERTER			
TIMER_COUNTER_1			
EPPROM			
SPI			
TWI			
USART1			
CPU			

PORTE a luat valoarea 0010, semnificand ca s-a aprins al treilea LED, iar pe PORTC s-a scris comanda asociata.

- Starea porturilor dupa introducerea comenzi "3 1 4 F" (modul frecventa):

AVR Studio - [C:\Users\Bogdan\Desktop\proiect nou avr\main.c]

Processor

Program Counter	0x000169
Stack Pointer	0x04FD
X pointer	0x000C
Y pointer	0x01F4
Z pointer	0x000C
Cycle Counter	4811358
Frequency	4.0000 MHz
Stop Watch	120233.50 us
SREG	0x80

Registers

R00	0x0C
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x0C
R07	0x00
R08	0x00
R09	0x0F
R10	0x04
R11	0xD4
R12	0x04
R13	0x08
R14	0x00

Code:

```

cnt=(cnt+1)%50;
if (cnt==0) LED1 = ~LED1; // invert LED
// MyApplication();

/* main function of program
 */
void main (void)
{
unsigned char temp,i;

Init_initController(); // this must be the first "init" action/call!
#asm("sei") // enable interrupts
LED1 = 1; // initial state, will be changed by timer 1

cnt_key = 0;
ok = 1;

while(TRUE)
{
}
/*
 vdogtrig(); // call often else processor will reset
 if(rx_counter0) // if a character is available on serial port USART0
 {
 temp = getchar();
 if(temp == '?')
 printf("\r\nSwVersion: %d.%d\r\n", SW_VERSION/10, SW_VERSION%10);
 else
 putchar(temp+1); // echo back the character + 1 ("a" becomes "b", etc)
}

```

I/O View

Name	Address	Value	Bits
PORTA	0x02(0x22)	0xF7	███████
DDRA	0x01(0x21)	0x0F	███
PINA	0x00(0x20)	0x77	██████████
PORTB	0x05(0x25)	0x04	██
DDRB	0x04(0x24)	0x0F	███
PINB	0x03(0x23)	0x04	███
PORTC	0x08(0x28)	0x04	██
DDRC	0x07(0x27)	0xFF	████████████████████
PINC	0x06(0x26)	0x04	███
PORTD	0x0B(0x2B)	0x60	███
DDRD	0x0A(0x2A)	0x50	███
PIND	0x09(0x29)	0x42	███

Message

Loaded plugin STK500
gcc plug-in: No WinAVR installation found. The AVR GCC plug-in can still be used if you set up your own build tools.
Loaded partfile: C:\Program Files (x86)\Atmel\AVR Tools\PartDescriptionFiles\ATmega16P
AVR Simulator: Please wait while configuring simulator...
AVR Simulator: ATmega16P Configured OK
Loaded objectfile: C:\Users\Bogdan\Desktop\proiect nou avr\proiect_nou_avr.cof

PORPB a luat valoarea 0100, semnificand ca s-a aprins al doilea LED, iar pe PORTC s-a scris comanda asociata.

- Starea porturilor dupa introducerea comenzi "0 4 F" (comanda incorecta):

AVR Studio - [C:\Users\Bogdan\Desktop\proiect nou avr\main.c]

Processor

Program Counter	0x000169
Stack Pointer	0x04FD
X pointer	0x000F
Y pointer	0x01F4
Z pointer	0x000F
Cycle Counter	6012510
Frequency	4.0000 MHz
Stop Watch	1503127.50 us
SREG	0x80

Registers

R00	0x0F
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x0F
R07	0x00
R08	0x00
R09	0x0F
R10	0x00
R11	0x00
R12	0x04
R13	0x08
R14	0x00

Code:

```

cnt=(cnt+1)%50;
if (cnt==0) LED1 = ~LED1; // invert LED
// MyApplication();

/* main function of program
 */
void main (void)
{
unsigned char temp,i;

Init_initController(); // this must be the first "init" action/call!
#asm("sei") // enable interrupts
LED1 = 1; // initial state, will be changed by timer 1

cnt_key = 0;
ok = 1;

while(TRUE)
{
}
/*
 vdogtrig(); // call often else processor will reset
 if(rx_counter0) // if a character is available on serial port USART0
 {
 temp = getchar();
 if(temp == '?')
 printf("\r\nSwVersion: %d.%d\r\n", SW_VERSION/10, SW_VERSION%10);
 else
 putchar(temp+1); // echo back the character + 1 ("a" becomes "b", etc)
}

```

I/O View

Name	Address	Value	Bits
PORTA	0x02(0x22)	0xF7	███████
DDRA	0x01(0x21)	0x0F	███
PINA	0x00(0x20)	0x77	██████████
PORTB	0x05(0x25)	0x00	██
DDRB	0x04(0x24)	0x0F	███
PINB	0x03(0x23)	0x08	███
PORTC	0x08(0x28)	0x00	██
DDRC	0x07(0x27)	0xFF	████████████████████
PINC	0x06(0x26)	0x00	██
PORTD	0x0B(0x2B)	0x60	███
DDRD	0x0A(0x2A)	0x50	███
PIND	0x09(0x29)	0x42	███

Message

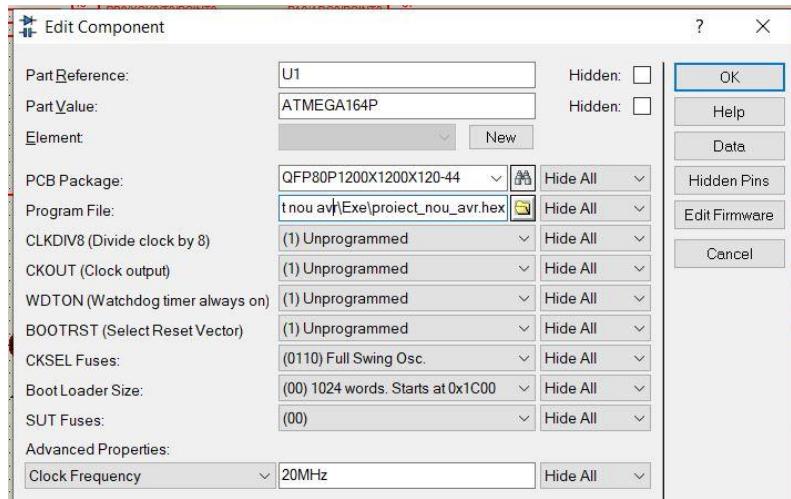
Loaded plugin STK500
gcc plug-in: No WinAVR installation found. The AVR GCC plug-in can still be used if you set up your own build tools.
Loaded partfile: C:\Program Files (x86)\Atmel\AVR Tools\PartDescriptionFiles\ATmega16P
AVR Simulator: Please wait while configuring simulator...
AVR Simulator: ATmega16P Configured OK
Loaded objectfile: C:\Users\Bogdan\Desktop\proiect nou avr\proiect_nou_avr.cof

PORPB a luat valoarea 1000, semnificand ca s-a aprins al primul LED(comanda gresita), iar pe PORTC nu s-a mai scris nimic, deoarece nu trimitem comanda incorecta catre DSP.

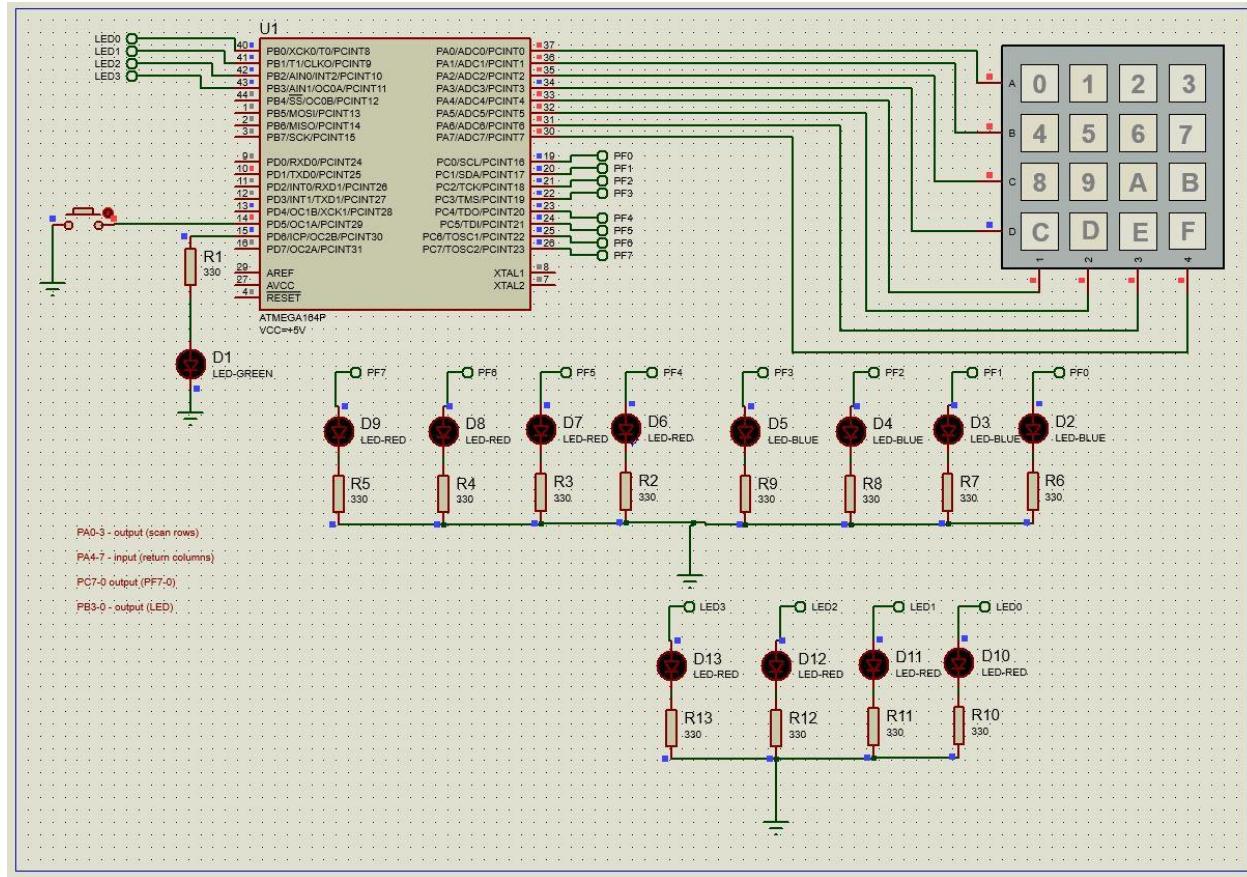
Proteus:

Pentru a simula practic functionarea microcontrolerului AVR, am folosit aplicatia Proteus.

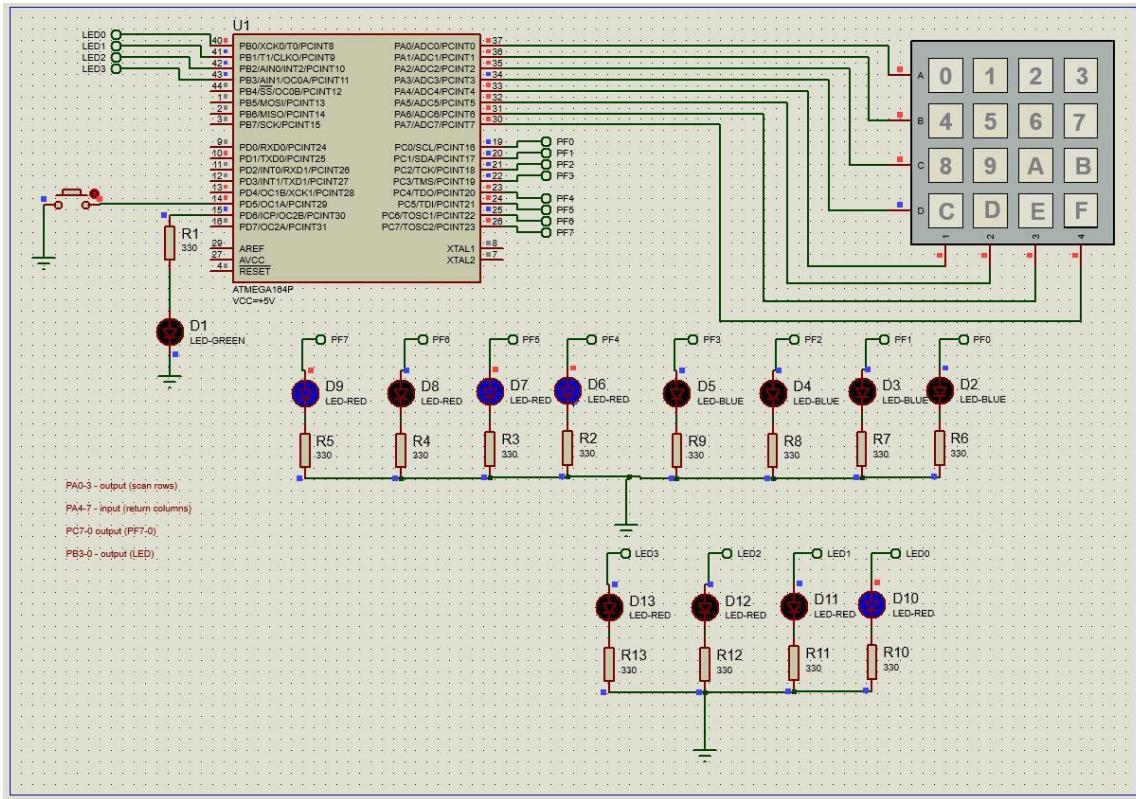
- Pentru configurarea ATMEGA164A, am introdus fisierul .hex rezultat din programul dezvoltat in CodeVisionAVR;



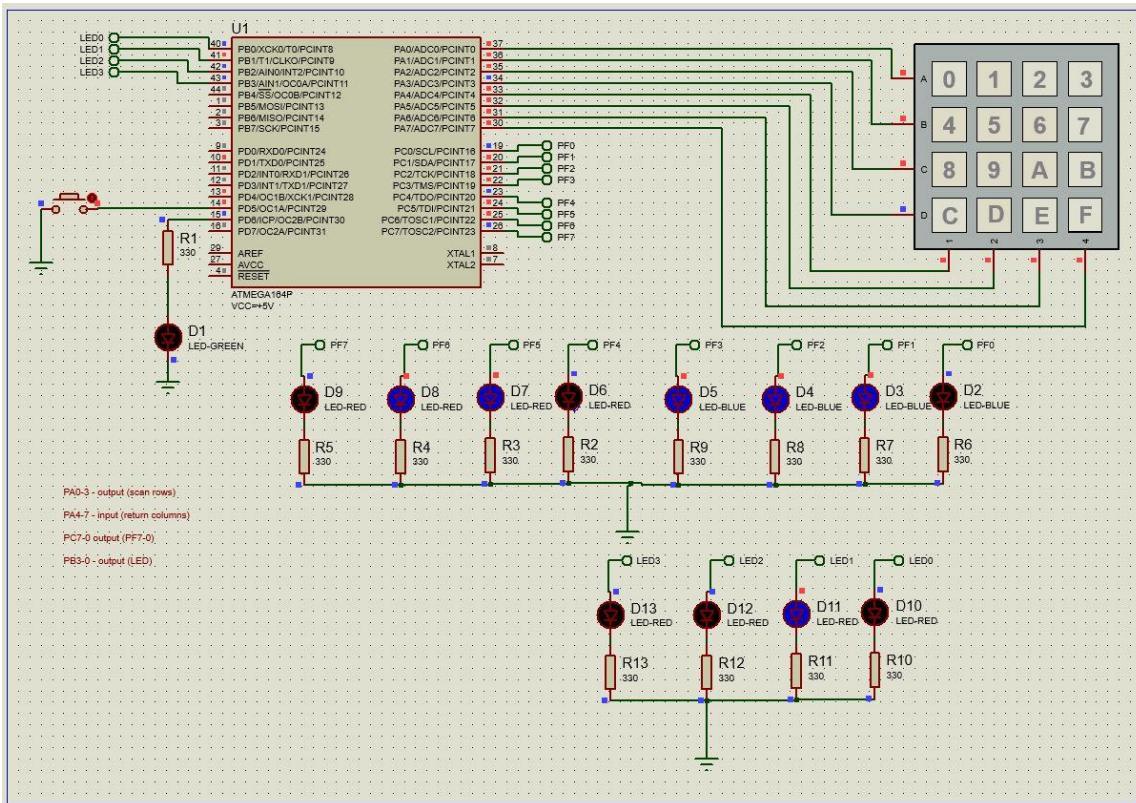
- Starea initiala a PF si a celor 4 LED-uri:



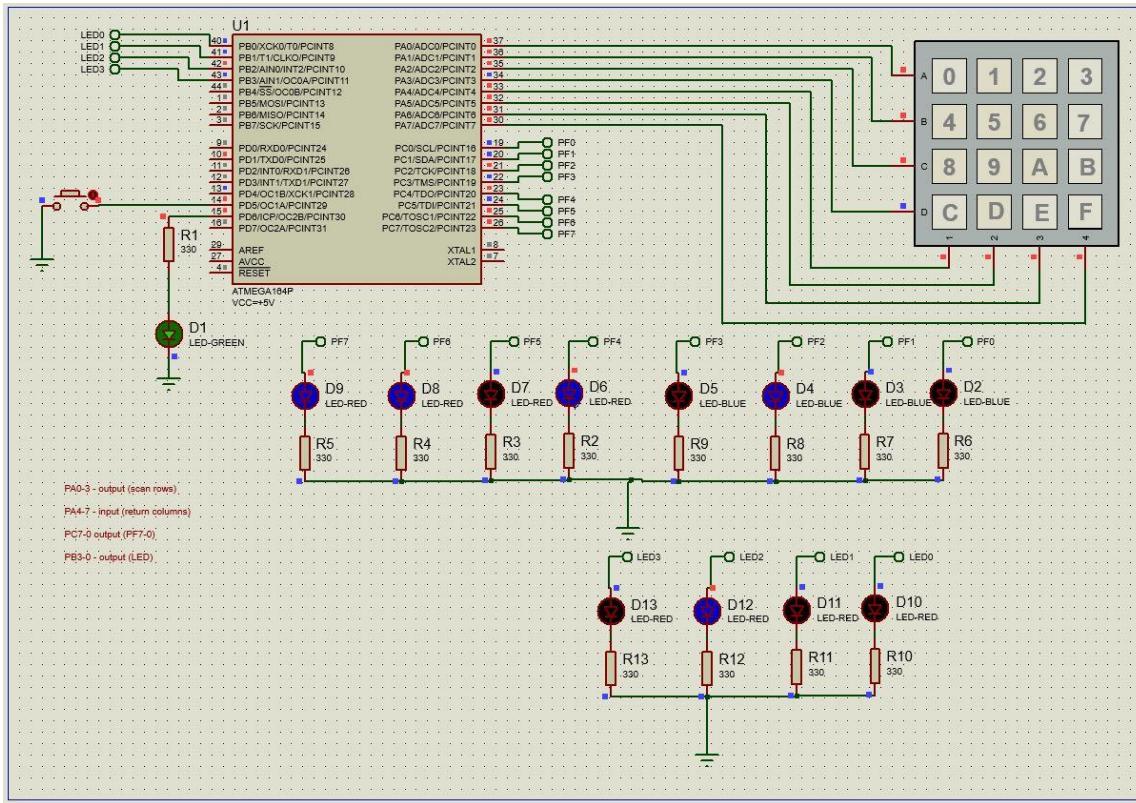
- Starea porturilor după introducerea comenzi “2 3 F” (modul STOP):



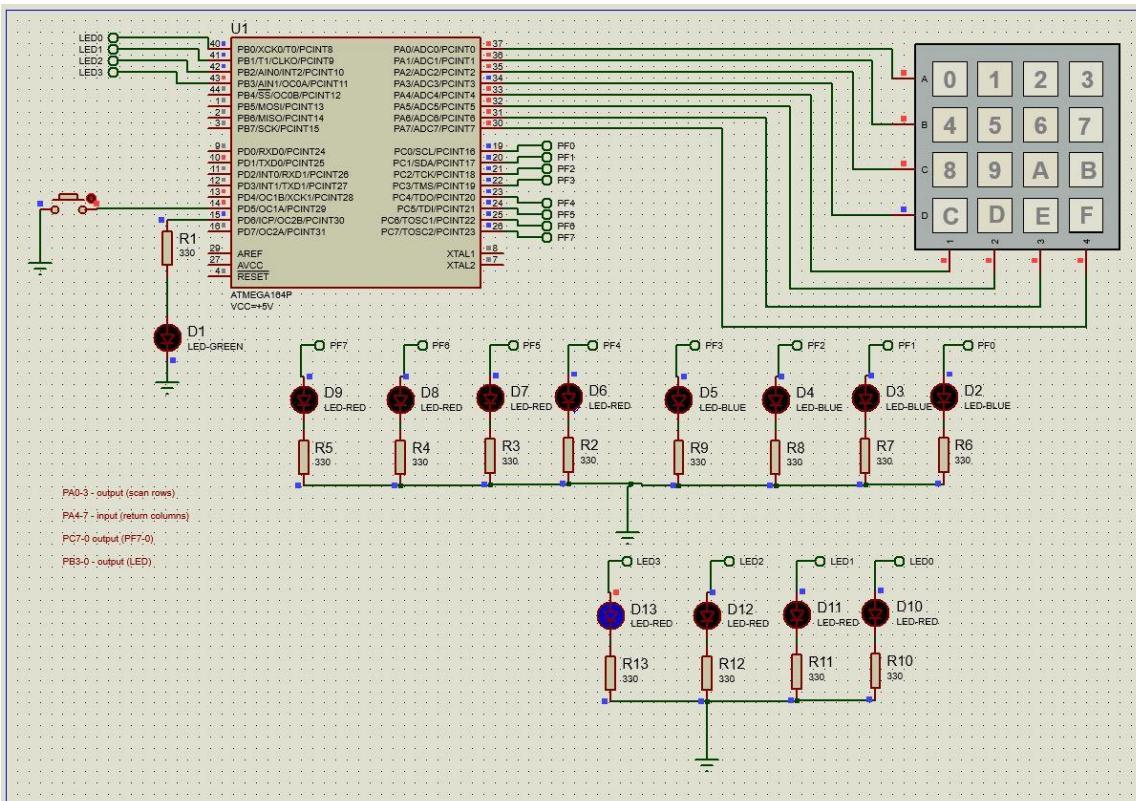
- Starea porturilor după introducerea comenzi “1 2 E F” (modul DTMF):



- Starea porturilor după introducerea comenzi “3 1 4 F” (modul frecvența):



- Starea porturilor după introducerea comenzi “0 4 F” (comanda incorecta):



DSP:

Pentru generarea tuturor celor 64 de semnale DTMF pentru simularea in VisualDSP++, am folosit urmatorul cod MATLAB:

```
80
81     fx = zeros(4,8);
82     fx(1,:) = [ 200, 360, 520, 680, 1240, 1400, 1560, 1720];
83     fx(2,:) = [ 280, 440, 600, 760, 1160, 1320, 1480, 1640];
84     fx(3,:) = [ 200, 360, 520, 680, 1160, 1320, 1480, 1640];
85     fx(4,:) = [ 280, 440, 600, 760, 1240, 1400, 1560, 1720];
86
87     for i=1:4
88         for j=1:4
89             for l=5:8
90                 for k=1:M
91                     x(k)=A/2*sin(2*pi*k*fx(i,j)/fs)+A/2*sin(2*pi*k*fx(i,l)/fs);
92                 end
93                 filename = sprintf('%d_%d.dat', fx(i, j), fx(i, l));
94                 fis=fopen(filename,'wt');
95                 fprintf(fis,'%1.14f\n' ,x);
96                 fclose(fis);
97             end
98         end
99     end
100
```

In simulator, am introdus pe streamul de intrare un fisier de semnal DTMF generat mai sus. Pentru verificarea aparitiei unei anumite frecvente, trebuie introdus coeficientul corespunzator pentru una dintre cele doua frecvente.

Algoritmul are succes atunci cand s-au verificat ambele frecvente asociate codului DTMF de la intrare.

In cazul simularii unui semnal sinusoidal la intrare, testarea se realizeaza o singura data pentru frecventa acestuia, si se decide daca se potriveste cu frecventa cautata.

Simulari test_ext – DSP:

1. Starea portului de iesire PORT_OUT (0xFF) avand ca input:

PF = 0xE6 => ID_PF = 3, M = 2, F = 6

PORT_IN = 0xF8 => ID_DSP = 3, SET = 3, Scale = 8

Semnal de intrare DTMF(440 Hz, 1560 Hz).

The screenshot shows the DS5 debugger interface with several windows open. On the left is the project tree for 'test_ext.dpj'. The assembly code window shows lines 634 to 672 of the code. The PF Registers window shows the state of the Processor Feature Registers. The DM [Hexadecimal] windows show memory locations 0x1010, freq_OK, index_freq1, and tx_buf. The IOM [Hexadecimal] windows show memory locations 0x3000 (coefs) and 0xff (PORT_OUT). The computational register window shows AX0=008, AX1=008, AR=0002, AF=0018.

```

634     dm(ok_afisor)=ax0;
635     rts;
636
637 final_dtmf:
638     ar=dm(freq_OK);
639     dm(freq2_OK)=ar;
640     ar=ax0;
641     ar=(dtmf_freq_cnt)=ar;
642     ax0=0;
643     dm(OK)=ax0;
644     rti;
645
646 final_dtmf_2:
647     ax0=dm(freq_OK);
648     ay0=dm(freq2_OK);
649     ar=ax0 and ay0;
650     ar=ar0;
651     if eq call drawErr_dtmf;
652     if eq call drawLetter_dtmf;
653     ax0=0;
654     dm(OK)=ax0;
655     rti;
656
657 drawLetter_dtmf:
658     ax0=dm(PF3_0);
659     ar=ax0-0;
660     if eq call draw0;
661     ar=ax0-1;
662     if eq call draw1;
663     ar=ax0;
664     ar=ax0-1;
665     if eq call draw2;
666     ar=ax0;
667     ar=ax0-1;
668     if eq call draw3;
669     ar=ax0;
670     ar=ax0-1;
671     if eq call draw4;
672     ar=ax0;

```

ID-urile corespund, deci se transmite pe afisor. Se face test DTMF a tastei 6, ce are frecvențele din setul 3, adică exact (440, 1560). Asadar, verificarea este finalizată cu succes, iar pe afisor se transmite codul 0x02, corespunzător “6.”.

2. Starea portului de iesire PORT_OUT (0xFF) avand ca input:

PF = 0xF6 => ID_PF = 3, M = 3, C = 6

PORT_IN = 0xF8 => ID_DSP = 3, SET = 3, Scale = 8

Semnal de intrare DTMF(440 Hz, 1560 Hz).

The screenshot shows the DS5 debugger interface with several windows open. On the left is the project tree for 'test_ext.dpj'. The assembly code window shows lines 576 to 672 of the code. The PF Registers window shows the state of the Processor Feature Registers. The DM [Hexadecimal] windows show memory locations 0x1010, freq_OK, index_freq1, and tx_buf. The IOM [Hexadecimal] windows show memory locations 0x3000 (coefs) and 0xff (PORT_OUT). The computational register window shows AX0=0FF, AX1=008, AR=0000, AF=0018.

```

576     sr=shift si by -2(hi);
577     if(index_freq1)=sr1;
578     ar=dm(OK);
579     ar=ar-2;
580     if eq jump final_dtmf;
581     jump goertzel;
582 caz_dtmf_2:
583     ax0=dm(PF3_0);
584     ax0=0x0003;
585     ar=sr1 and ay1;
586     ar=ar4;
587     dm(index_freq1)=ar;
588     ar=dm(OK);
589     if eq jump final_dtmf_2;
590     jump goertzel;
591
592 caz_stop:
593     ax0=0FF;
594     IO(PORT_OUT)=ax0;
595     rti;
596
598 final_freq:
599     ax0=dm(freq_OK);
600     ar=ax0-1;
601     if eq call drawErr;
602     if eq call drawLetter;
603     ax0=0;
604     dm(OK)=ax0;
605     rti;
606
607 drawLetter:
608     ax0=dm(PF3_0);
609     ar=ax0-0;
610     if eq call draw0;
611     ar=ax0-1;
612     if eq call draw1;
613     ar=ax0;
614     ar=ax0-1;

```

In modul “STOP”, se observă că la final se transmite pe PORT_OUT valoarea 0xFF, adică toate LED-urile stinse (logica inversă).

3. Starea portului de iesire PORT_OUT (0xFF) avand ca input:

PF = 0xD3 => ID_PF = 3, M = 1, F = 3

PORT_IN = 0xC8 => ID_DSP = 3, SET = 0, Scale = 8

Semnal de intrare sinusoidal (680 Hz).

The screenshot shows the Keil MDK-ARM IDE interface with several windows open:

- Project: test_ext.dpj**: Shows the project structure with files like `test_ext.asm`, `tone_detection.dsp`, and `tone_detection.c`.
- PM [Hexadecimal]**: Registers window showing values for AX0, AT0, MX0, MY0, SR1, and SI.
- IOM [Hexadecimal]**: Memory dump window showing data at address 0x3000, which contains coefficients for tone generation.
- DM [Hexadecimal]**: Data memory window showing variables `rx_buf` and `freq_OK`.
- PF Registers**: Registers window showing PFTYPE, PFDATA, IS, and MS.
- Computational**: Registers window showing AX0, AT1, MX1, MY1, MR0, and SR0.
- IOM [Hexadecimal]**: Memory dump window showing data at address 0x1010, which contains the transmitted value 0x64F3.
- DM [Hexadecimal]**: Data memory window showing variables `PF3_0`, `OK`, `index_freq1`, `ok_afisor`, `freq2_OK`, and `dtaf_freq_cnt`.
- IOM [Hexadecimal]**: Memory dump window showing data at address 0x1100, which contains the transmitted value 0x64F3.

The assembly code in the editor shows a loop for tone generation, setting up frequencies and calling subroutines for drawing letters.

Suntem in modul testare frecventa. S-a ales setul 0 de frecvente, iar indexul 3 al acestui set este chiar frecventa de 680 Hz. Asadar, testarea trebuie sa fie cu succes si de aceasta data, lucru care se si intampla, iar pe PORT_OUT se transmite valoarea 0xB0, corespunzatoare simbolului "3".

4. Starea portului de iesire PORT_OUT (0xFF) avand ca input:

PF = 0xD4 => ID_PF = 3, M = 1, F = 4

PORT_IN = 0xC8 => ID_DSP = 3, SET = 0, Scale = 8

Semnal de intrare sinusoidal (680 Hz).

The screenshot shows the Keil MDK-ARM IDE interface with several windows open:

- Project: test_ext.dpj**: Shows the project structure with files like `test_ext.asm`, `tone_detection.dsp`, and `tone_detection.c`.
- PM [Hexadecimal]**: Registers window showing values for AX0, AT0, MX0, MY0, SR1, and SI.
- IOM [Hexadecimal]**: Memory dump window showing data at address 0x3000, which contains coefficients for tone generation.
- DM [Hexadecimal]**: Data memory window showing variables `rx_buf` and `freq_OK`.
- PF Registers**: Registers window showing PFTYPE, PFDATA, IS, and MS.
- Computational**: Registers window showing AX0, AT1, MX1, MY1, MR0, and SR0.
- IOM [Hexadecimal]**: Memory dump window showing data at address 0x1010, which contains the transmitted value 0x64F3.
- DM [Hexadecimal]**: Data memory window showing variables `PF3_0`, `OK`, `index_freq1`, `ok_afisor`, `freq2_OK`, and `dtaf_freq_cnt`.
- IOM [Hexadecimal]**: Memory dump window showing data at address 0x1100, which contains the transmitted value 0x64F3.

The assembly code in the editor shows a loop for tone generation, setting up frequencies and calling subroutines for drawing letters.

Fata de exemplul precedent, s-a schimbat frecventa testata. Aici, pentru ca testarea sa aiba succes, ar fi trebuit ca pe streamul de intrare sa avem semnal de 1240 Hz. Pe PORT_OUT se transmite 0xBF, corespunzator "-".

5. Starea portului de iesire PORT_OUT (0xFF) avand ca input:

PF = 0xEB => ID_PF = 3, M = 2, C = B

PORT_IN = 0xD8 => ID_DSP = 3, SET = 1, Scale = 8

Semnal de intrare DTMF (760 Hz, 1640 Hz).

The screenshot shows the Keil MDK-ARM IDE interface with several windows open:

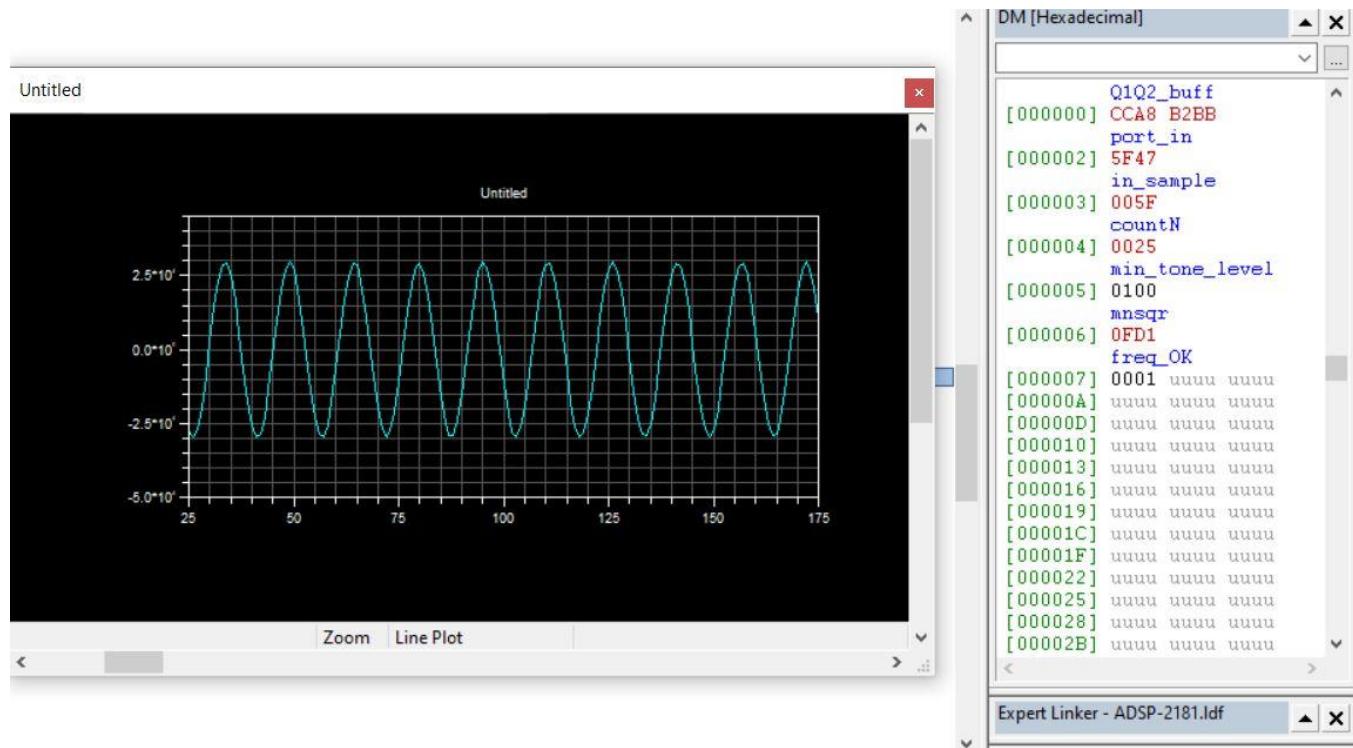
- Project: test_ext.dpj**: Shows the project structure with files like rts.s, final_dtmf.s, and drawLetter_dtmf.s.
- PM [Hexadecimal]**: Registers window showing PFTYPE 00, PFDATA EB, IS 3008, MS 0008.
- PF Registers**: Registers window showing AX0 0000, AX1 0008, AR 0000, AT0 0000, AT1 0003, AF 0008, MX0 0003, MX1 0000, MF 02B7, MY0 0002, MY1 0000, MR0 0000, MR2 FF, MR1 F350, SR0 000B, SI 00EB, SE uu, SB uu.
- IOM [Hexadecimal]**: I/O memory window showing 0x1fff and 0xff.
- DM [Hexadecimal]**: Data memory window showing rx_buf at 0x1010 with value F350, freq_OK at 0x1011 with value PF3_0, index_freq1 at 0x1012 with value 000B, ok_afisor at 0x1013 with value OK, freq2_OK at 0x1014 with value 0001, dta_freq_cnt at 0x1015 with value 0002, and others.
- DM [Hexadecimal]**: Data memory window showing tx_buf at 0x1100 with value C000 F350.

The assembly code in the main editor window shows the implementation of the DTMF detection logic, including calls to draw0, draw1, draw2, draw3, and draw4 routines.

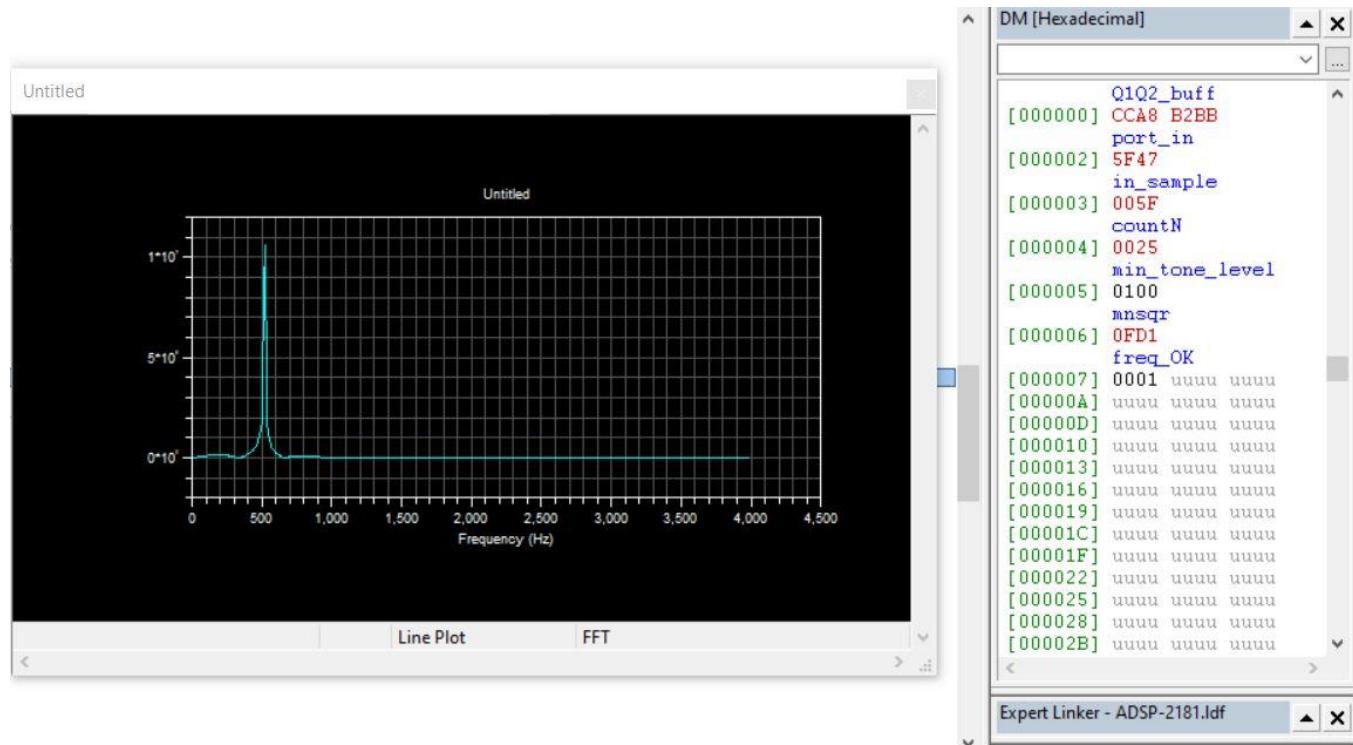
Modul de testare este DTMF. Aici, pentru ca testarea sa aiba succes, ar fi trebuit ca pe streamul de intrare sa avem semnal DTMF de (600Hz, 1640 Hz), sau sa se testeze codul "F". Pe PORT_OUT se transmite 0x3F, corespunzator "-".

Exemplu pentru mod frecventa(semnal sinusoidal):

- Semnal sinusoidal de frecventa 520 Hz:

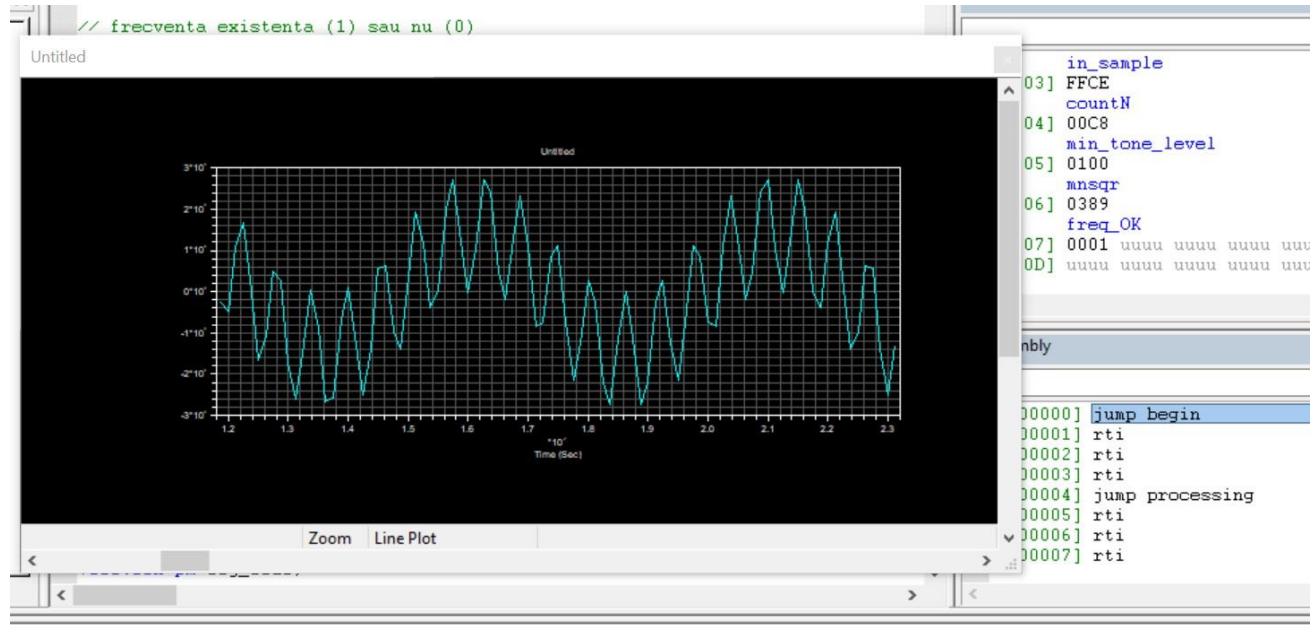


FFT:

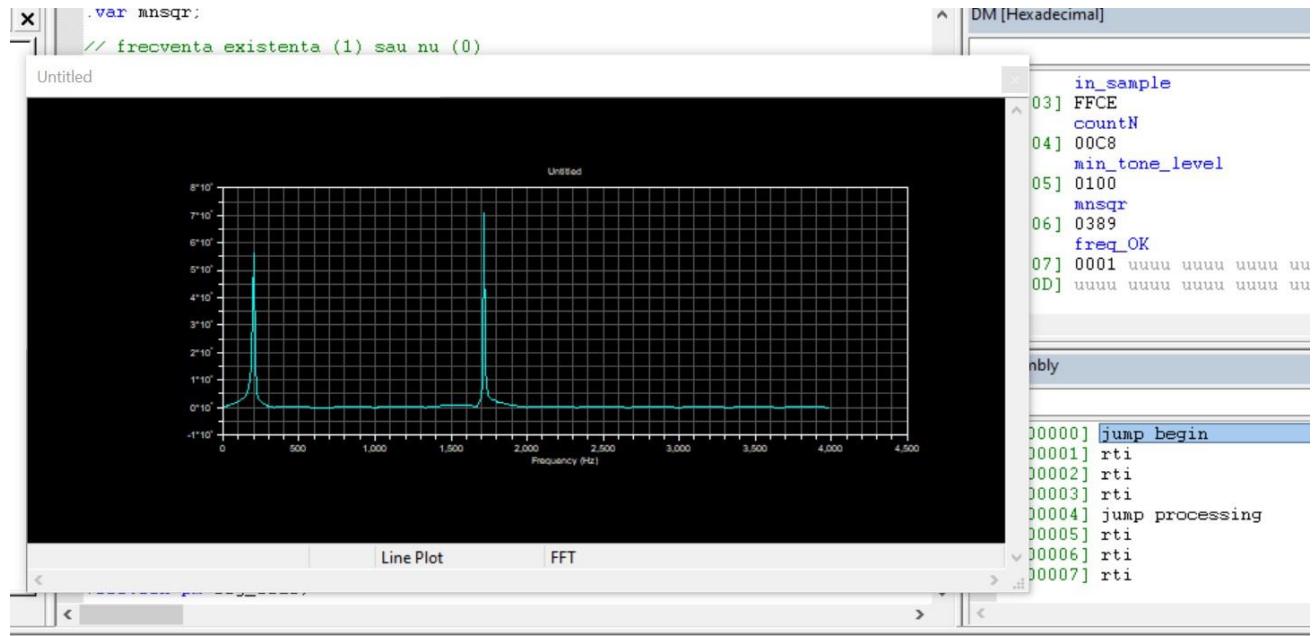


Exemplu de semnal DTMF:

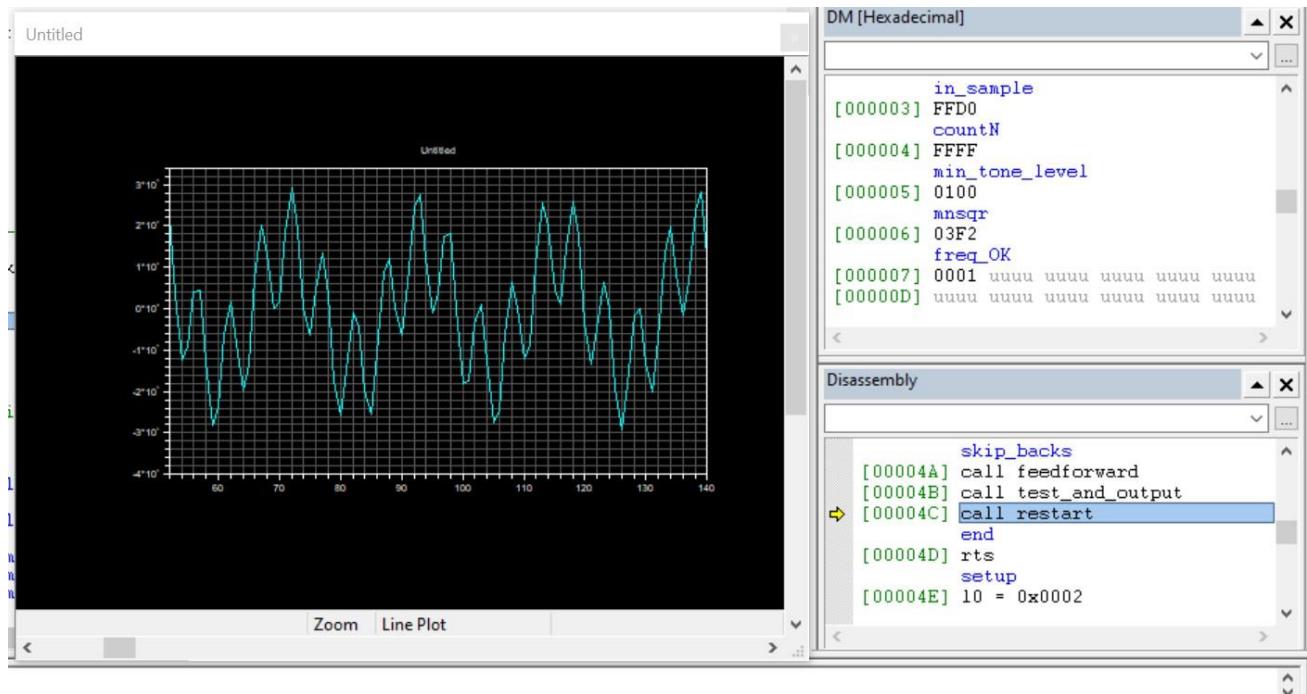
- Semnalul DTMF 200_1720:



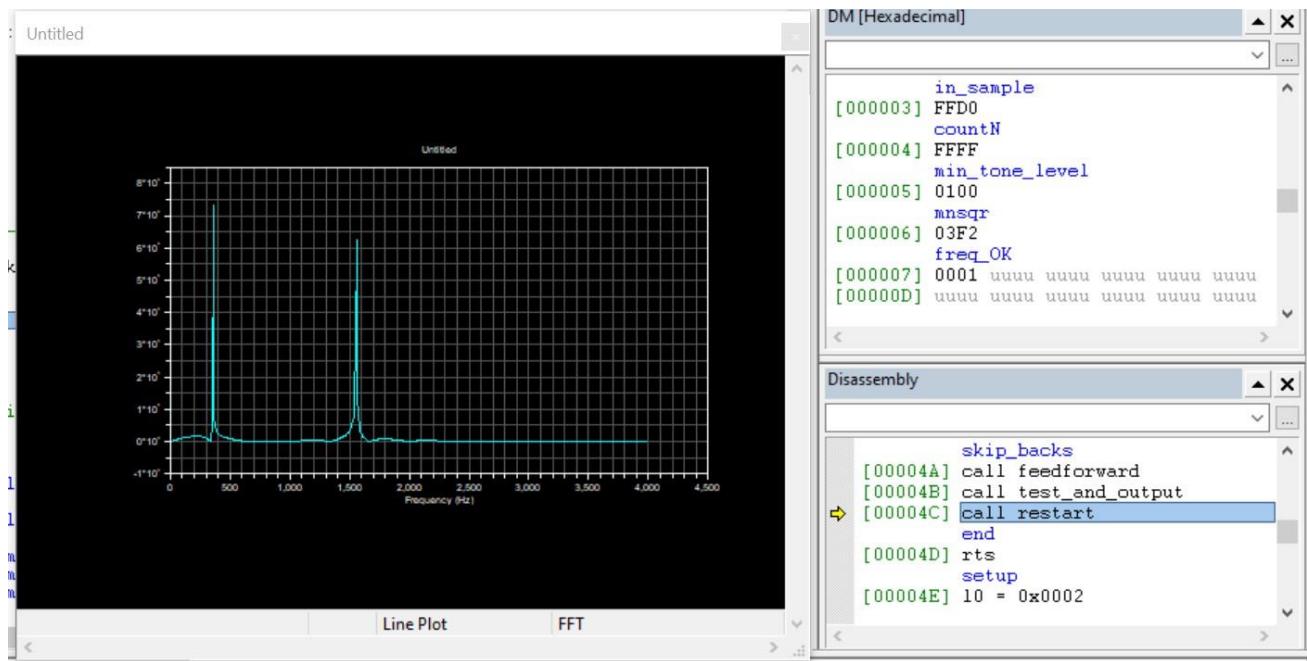
FFT:



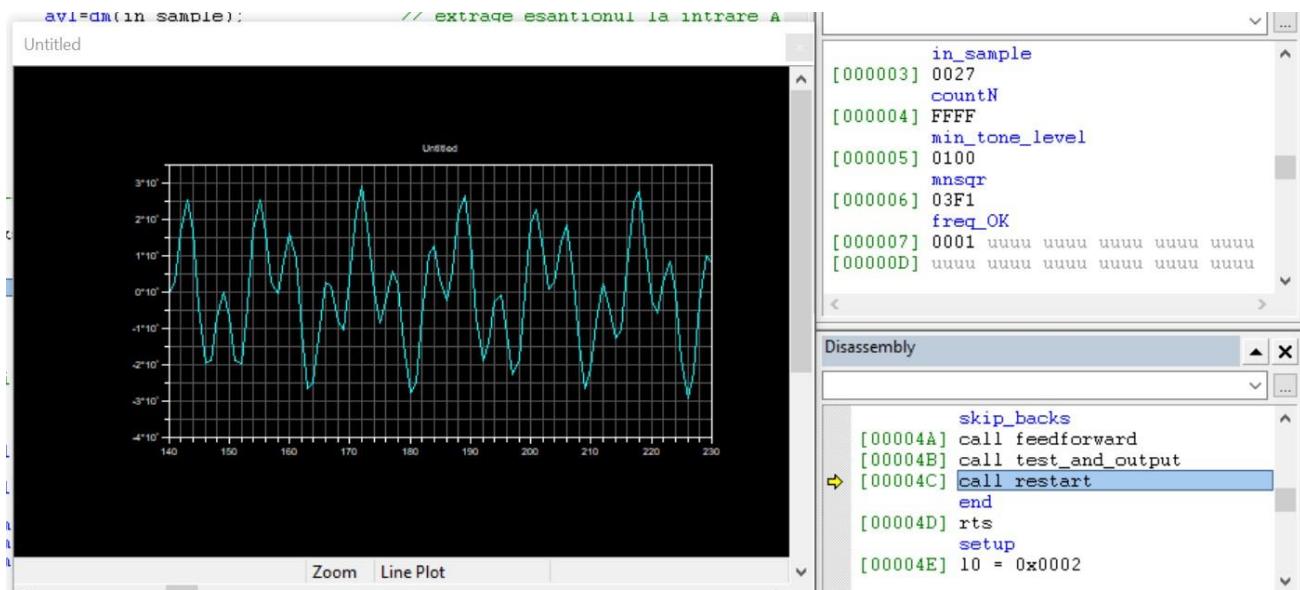
- Semnalul DTMF 360_1560:



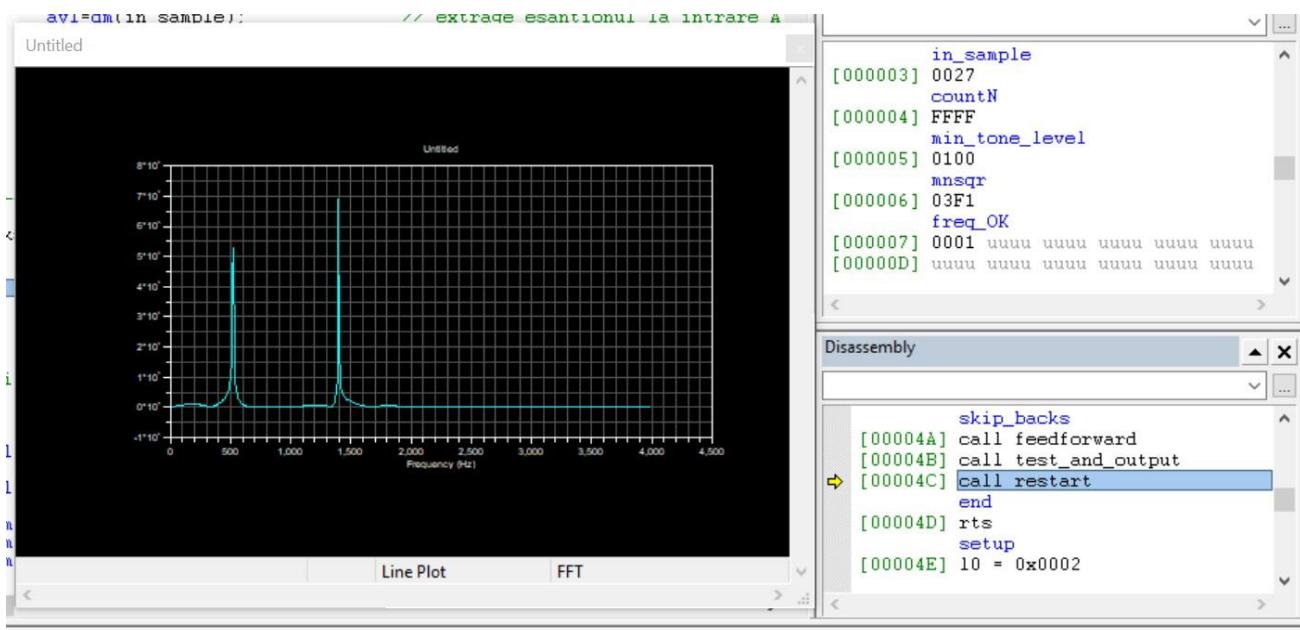
FFT:



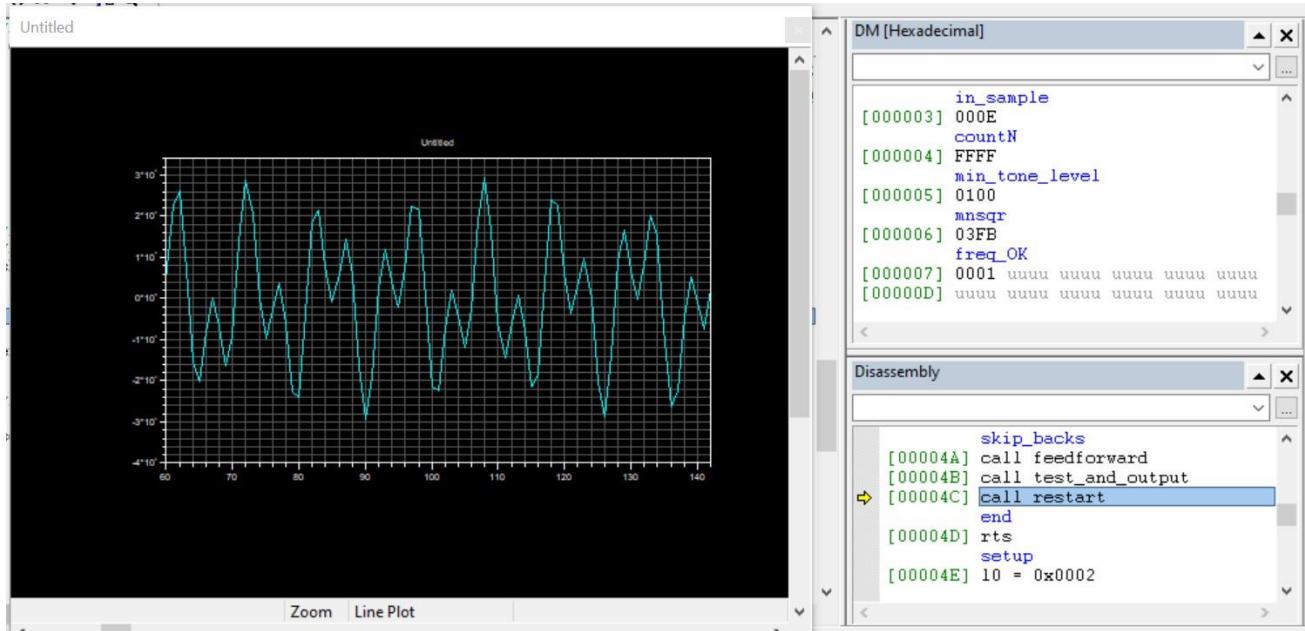
- Semnalul DTMF 520_1400:



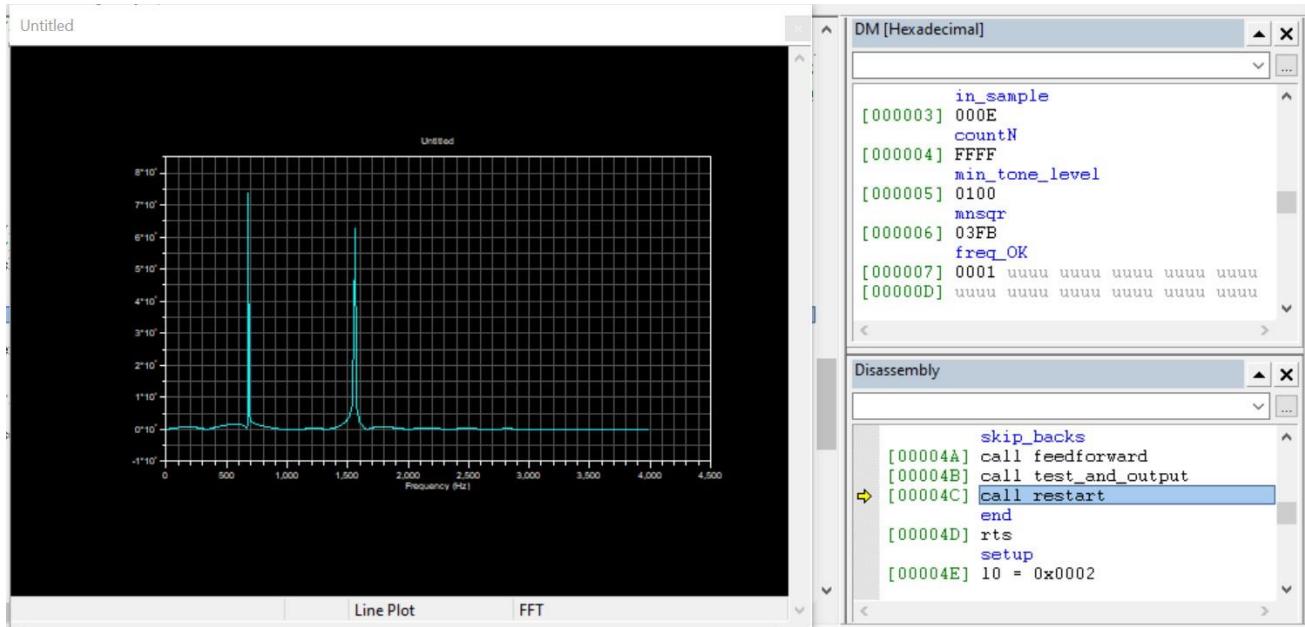
FFT:



- Semnalul DTMF 680_1560:



FFT:



BIBLIOGRAFIE

- Curs Microcontrolere 2023-2024
- <http://discipline.elcom.pub.ro/Proiect2/>
- Datasheet ATMeg164A - http://discipline.elcom.pub.ro/Proiect2/Atmel-42712-ATmega164A_Datasheet.pdf
- Set de instructiune ADSP21xx - http://discipline.elcom.pub.ro/Proiect2/adsp21xx_instruction_set.pdf
- www.avrfreaks.net
- <https://stackoverflow.com/>