Universitatea POLITEHNICA Bucuresti

Proiect 2

Echipa nr. 04

Studenti:

Mitrea Bogdan

Eftimie Albert

Constantinescu Adelina

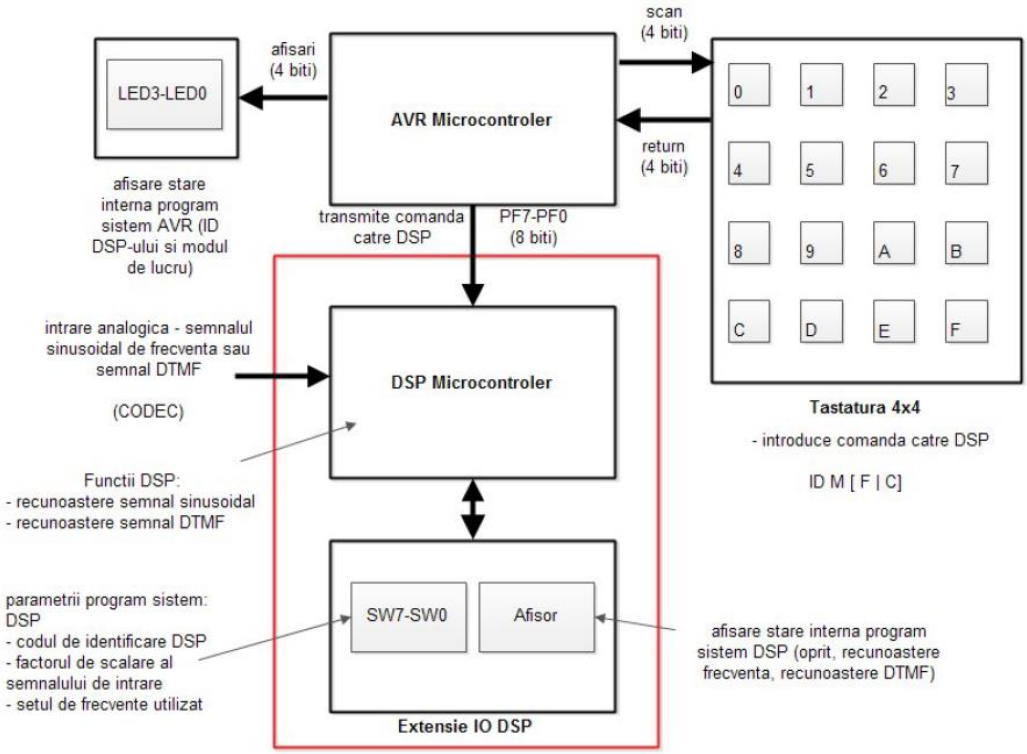
Grupa 432D

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 si DSP) și are ca funcție testarea apariției unui semnal sinusoidal sau DTMF pe intrarea analogică.

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

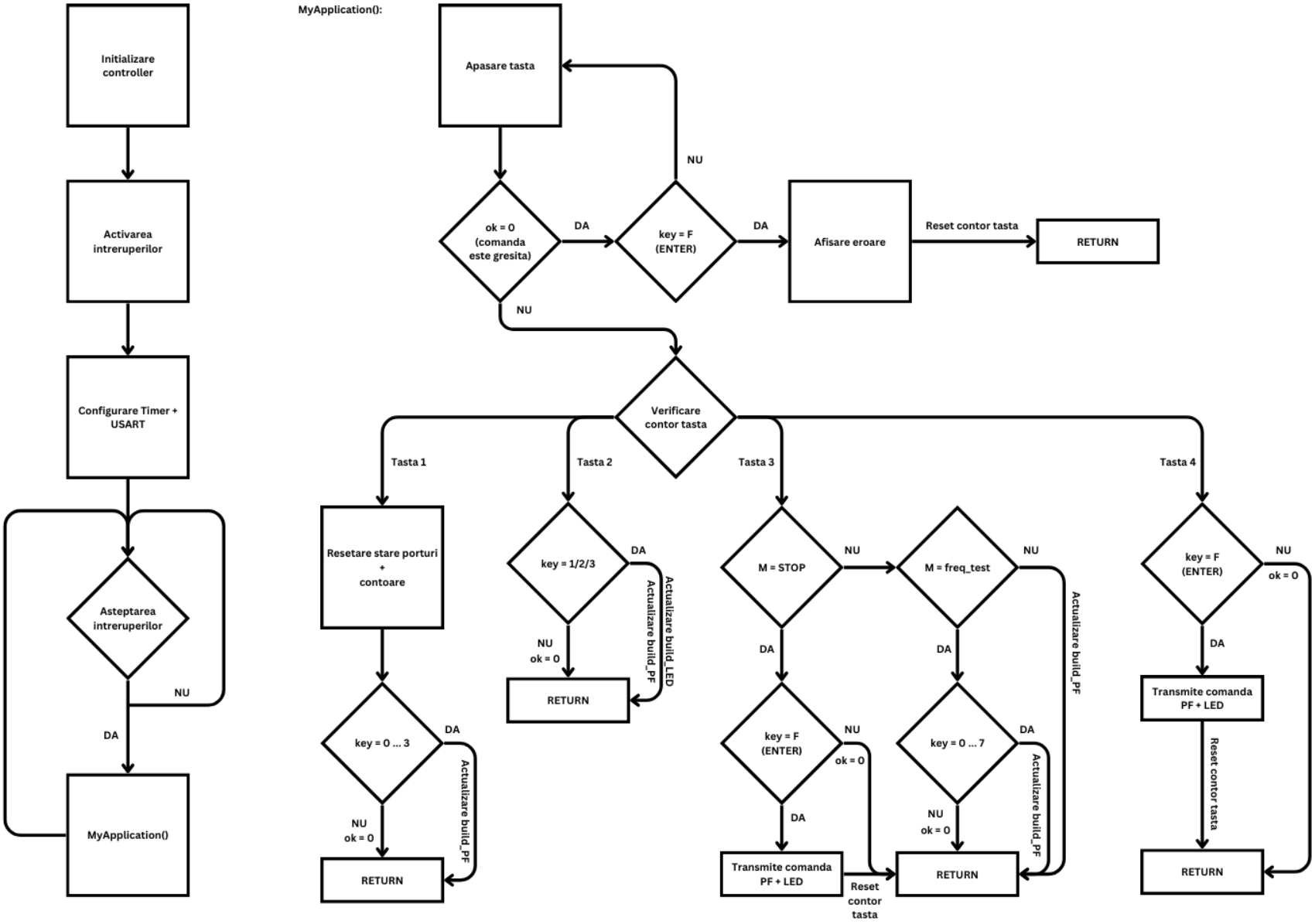
Subsistemul AVR preia o comandă de la tastatura și o retransmite către DSP prin portul PF.

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 frecvența, 2 – testează cod DTMF și 3 – stop DSP) F – frecvența testată (valori 0, 1, ..., 7) și C codul DTMF testat (valori 0, 1, ..., 7). Semnificația simbolurilor este: [] – optional, | - alegere.

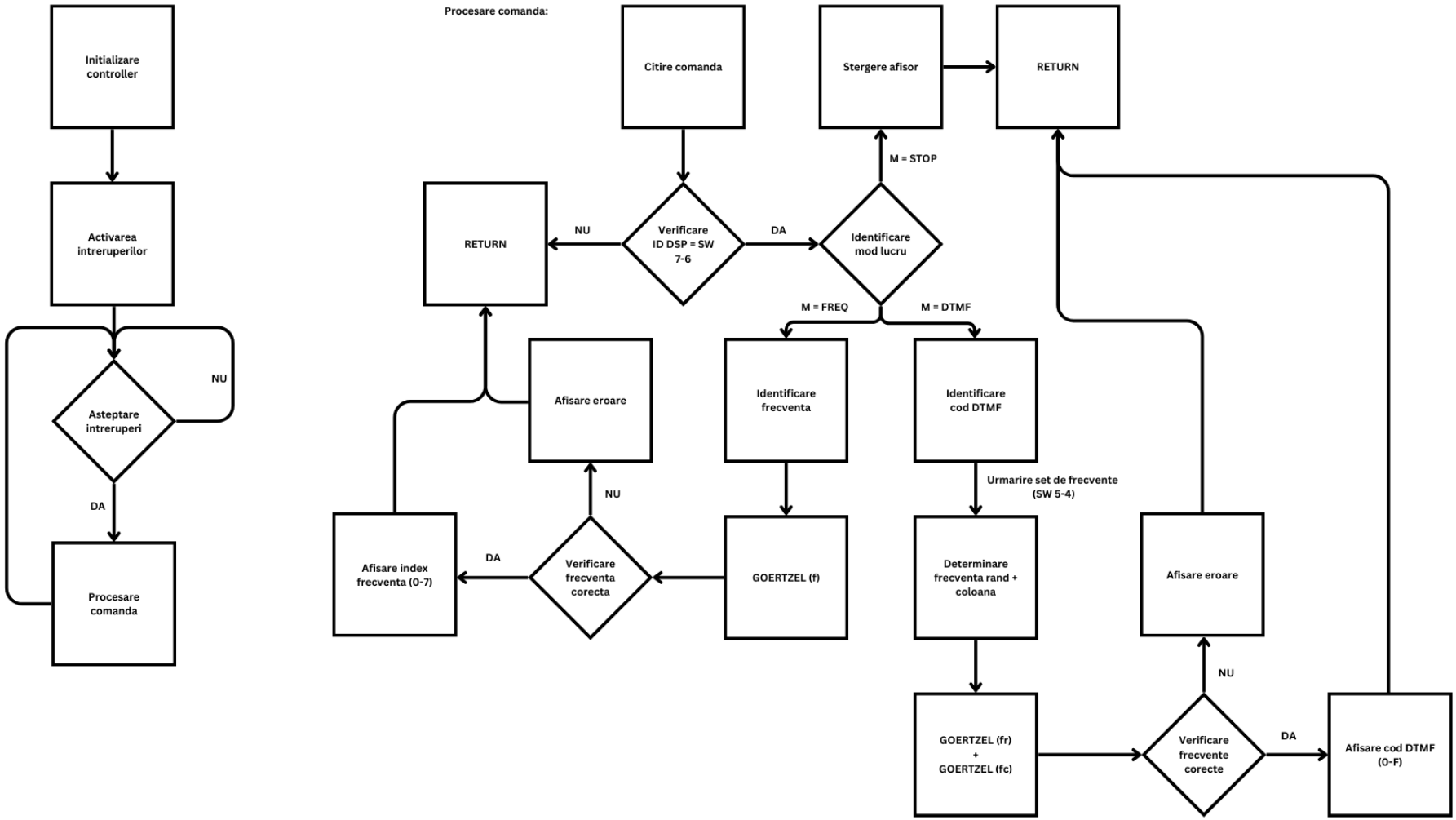
Ambele subsisteme își afișează starea proprie (pe LED3-LED0, respectiv pe un afișor cu 7 segmente – Afișor). Subsistemul AVR utilizează un microcontroler ATMega164. Subsistemul DSP are în componență placa de evaluare EZ-Kit LITE ADSP2181 și o interfața de intrare ieșire (IO DSP).

Descrierea prelucrarilor

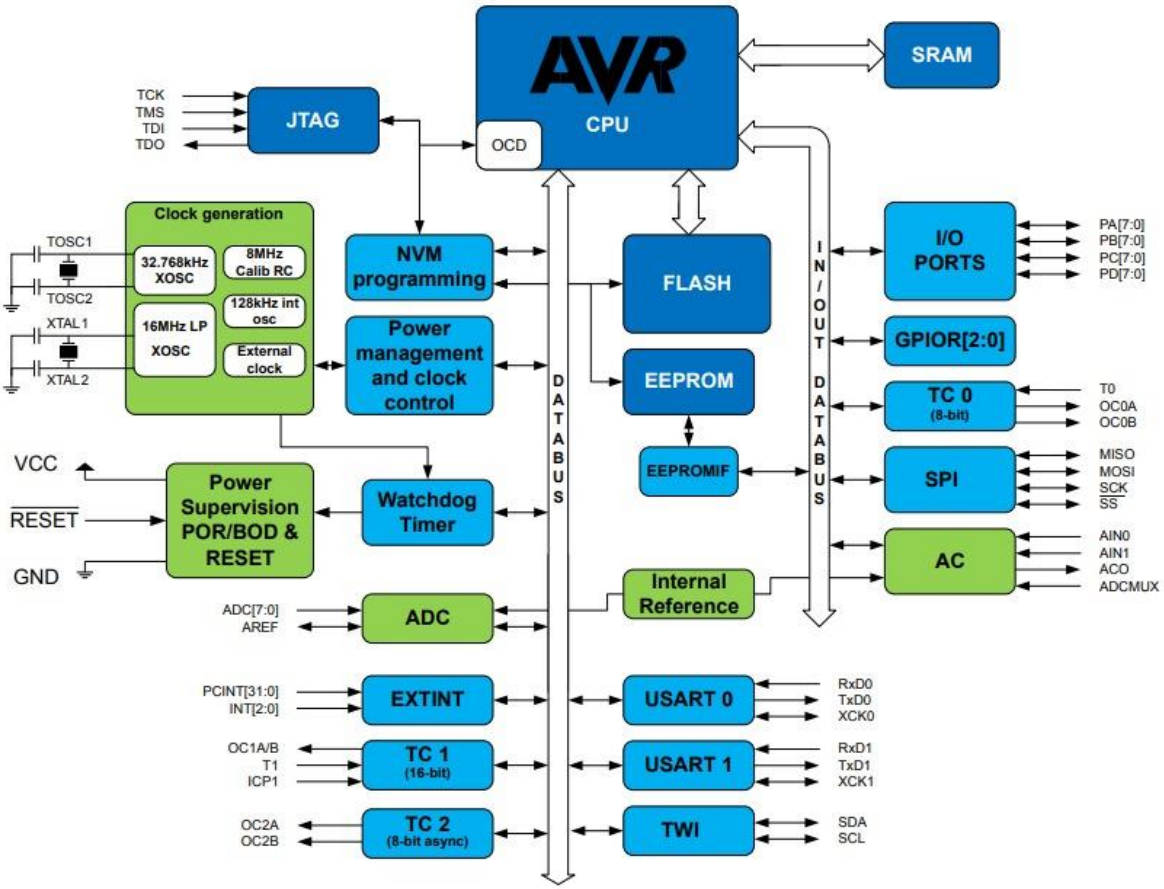
2.1. AVR:



2.2. DSP:


Initializare controller
Activarea intreruperilor
Asteptare intreruperi
NU
DA
Procesare comanda
Procesare comanda:
Citire comanda
Stergere afisor
RETURN
M = STOP
RETURN
NU
Verificare ID DSP = SW 7-6
DA
Identificare mod lucru
M = FREQ
M = DTMF
Afisare eroare
Identificare frecventa
Identificare cod DTMF
Urmarire set de frecvente (SW 5-4)
NU
Afisare index frecventa (0-7)
DA
Verificare frecventa corecta
GOERTZEL (f)
Determinare frecventa rand + coloana
Afisare eroare
NU
GOERTZEL (fr) + GOERTZEL (fc)
Verificare frecvente corecte
DA
Afisare cod DTMF (0-F)

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).

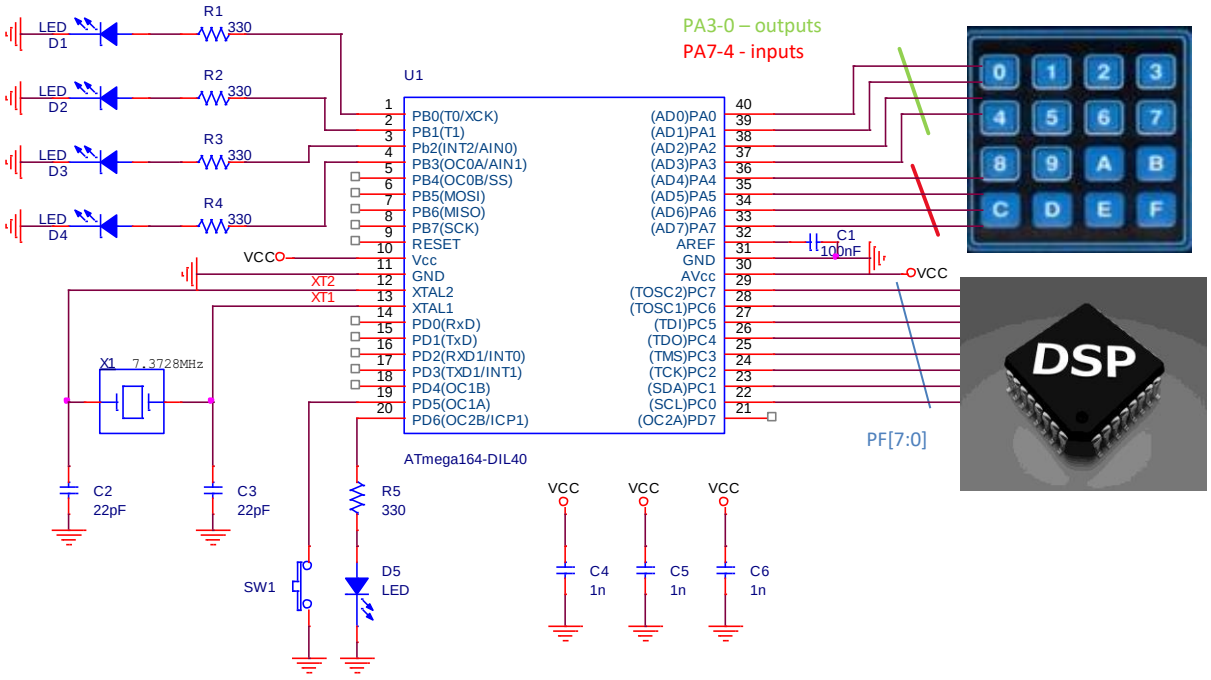
Acest microcontroller poate citi atât intrări analogice, cât și digitale, poate executa instrucțiuni bazate pe aceste intrări și poate transmite semnale către dispozitive externe sau alte circuite prin intermediul porturilor sale de intrare/ieșire (I/O). De asemenea, poate comunica cu alte dispozitive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

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.

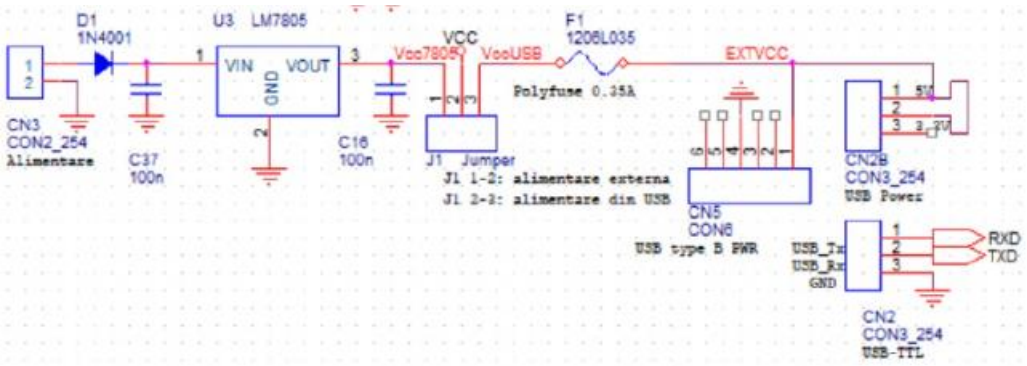
Butonul SW1 conectează pinul PD5 la masă (nivel logic '0') atunci când este apăsat. Pentru a asigura o stare logică definită pentru pinul PD5 atunci când butonul nu este apăsat, este necesar să se activeze o rezistență de pull-up internă prin intermediul unor instrucțiuni software. Astfel, în absența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

Chiar dacă nu este esențial, C6 ajută la minimizarea riscului de resetări neintenționate ale microcontrolerului cauzate de fluctuațiile bruște 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.

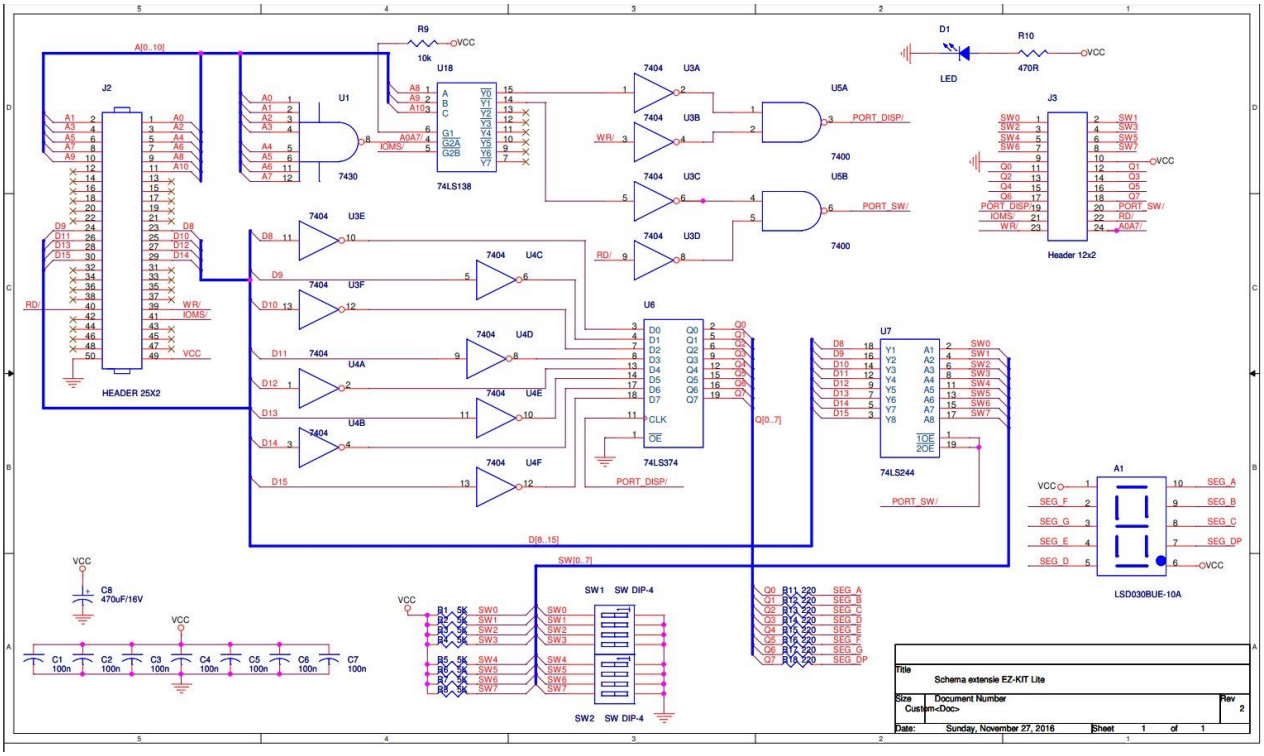
Pinii 22-29 sunt destinați conectării la DSP. La pinii 33-40, a fost adăugată o tastatură, care reprezintă o interfață de intrare ce permite utilizatorului să introducă comenzi sau date în microcontroler.



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 așa cum este prezentat în figura 2. Fiecare pin liber al microcontrolerului este adus la un pad adiacent, 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 Afișaje

LED: Afișează 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;
               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;
}
```

Algoritmul Goertzel:

```
/*{
  Acest exemplu ne prezinta detectia DTMF utilizand algoritmul Goertzel.

}
*/

#define   f_sample  8000
#define   N        200

#define   scale -8

.section/dm data1;

.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

// "tone-present" mnsqr level
.var      min_tone_level=0x0100;

// amplitudinea (in  1.15)  mnsqr Goertzel
.var mnsqr;

// frecventa existenta (1) sau nu (0)
.var freq_OK;

.section/pm pm_da;
// 2.14 coeficientii Goertzel : 2*cos(2*PI*k/N) pentru toate cele 8 frecvente
// 200, 360, 520, 680, 1240, 1400, 1560, 1720

.var/circ coefs[8]=0x7E6D,0x7AEB,0x7579,0x6E2D, 0x47F2, 0x3A1C, 0x2B5C, 0x1BEC;


//------------------------------------------------------------------------
//-------------------- PROGRAMUL PRINCIPAL ----------------------------------
//------------------------------------------------------------------------
.section/pm interrupts;

      jump begin;  rti; rti; rti;     // 00: reset
           jump processing;
                                rti; rti; rti;     // 04: IRQ2
      rti;        rti; rti; rti;     // 08: IRQL1
      rti;        rti; rti; rti;     // 0c: IRQL0
           rti;        rti; rti; rti;                     // 10: SPORT) tx
```

```
        rti;            rti; rti; rti;    // 14: SPORT0 rx
        rti;        rti; rti; rti;    // 18: IRQE
        rti;        rti; rti; rti;    // 1c: BDMA
        rti;        rti; rti; rti;    // 20: SPORT1 tx or IRQ1
        rti;        rti; rti; rti;    // 24: SPORT1 rx or IRQ0
        rti;        rti; rti; rti;    // 28: timer
        rti;        rti; rti; rti;    // 2c: power down



.section/pm seg_code;

begin:   call setup;
        call restart;

                        i0=Q1Q2_buff;
        i5=coefs;
                        m5=2;
        modify(i5,m5);        // i5 pointer catre coeficientul asociat frecventei testate

        IMASK = 0x200;        // valideaza intreruperea IRQ2

stop : jump stop;                        // asteapta intreruperi

//--------- PROCESAREA UNUI ESANTION ------------------------------------------
//
processing:

si=dm(port_in);        // 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:


                    sr=ashift si by scale (hi);
      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 = noul Q1
            dm(i0,m0)=mx0;                //vechiul Q1 = noul 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;
                                    rts;
```

Simulari

AVR:

PORTA: Tastatura

PORTB3-0: Starea celor 4 LED-uri.

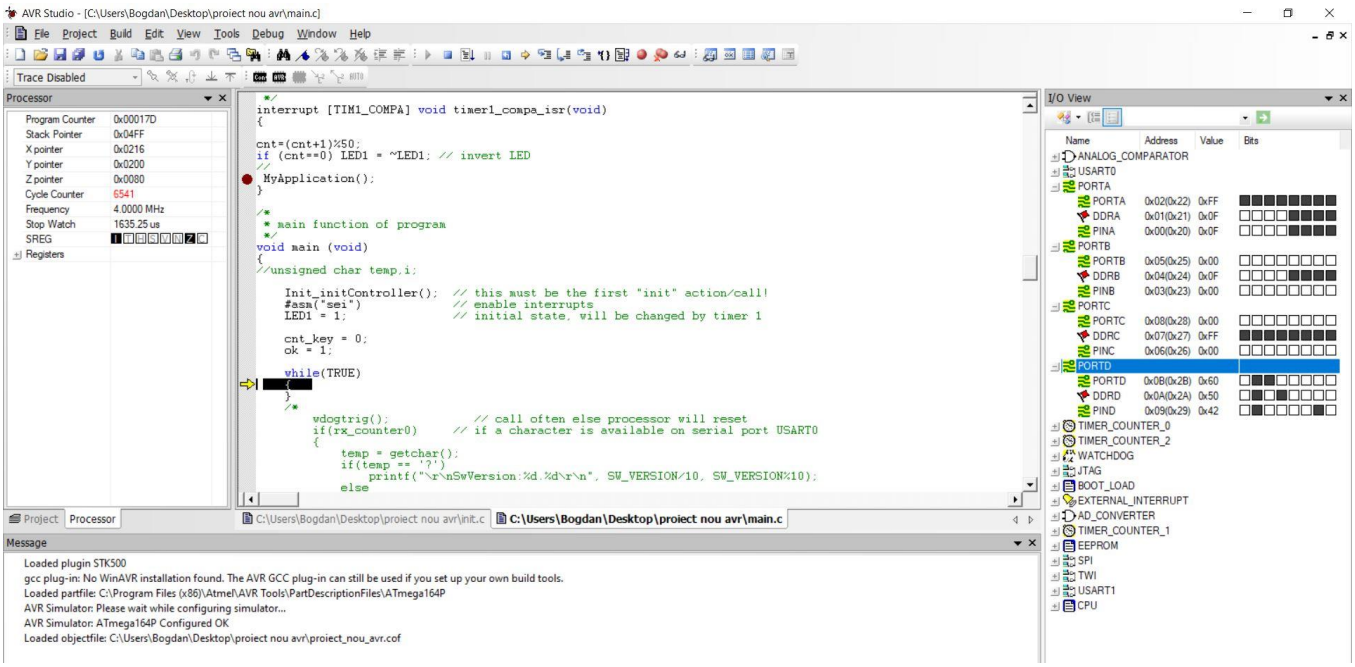
PORTC: Starea PF

PORTC7-6: ID

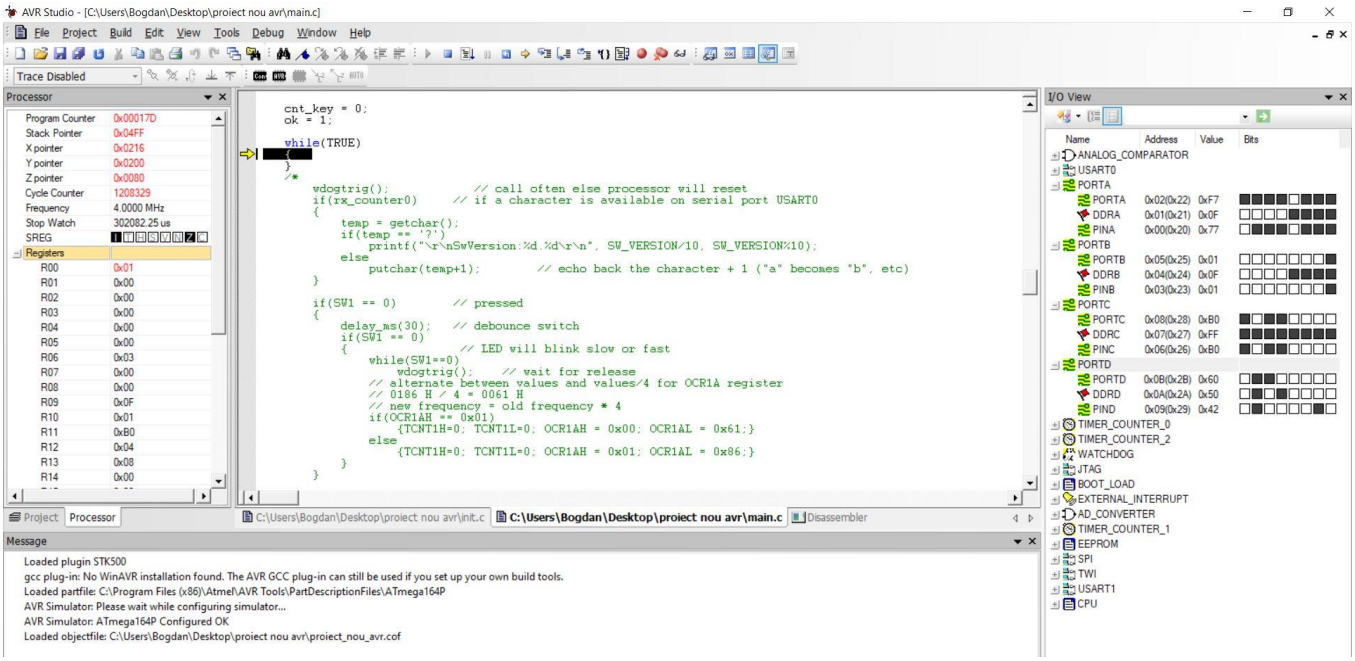
PORTC5-4: M

PORTC3-0: frecventa testata | cod DTMF

- Starea porturilor dupa initializare:

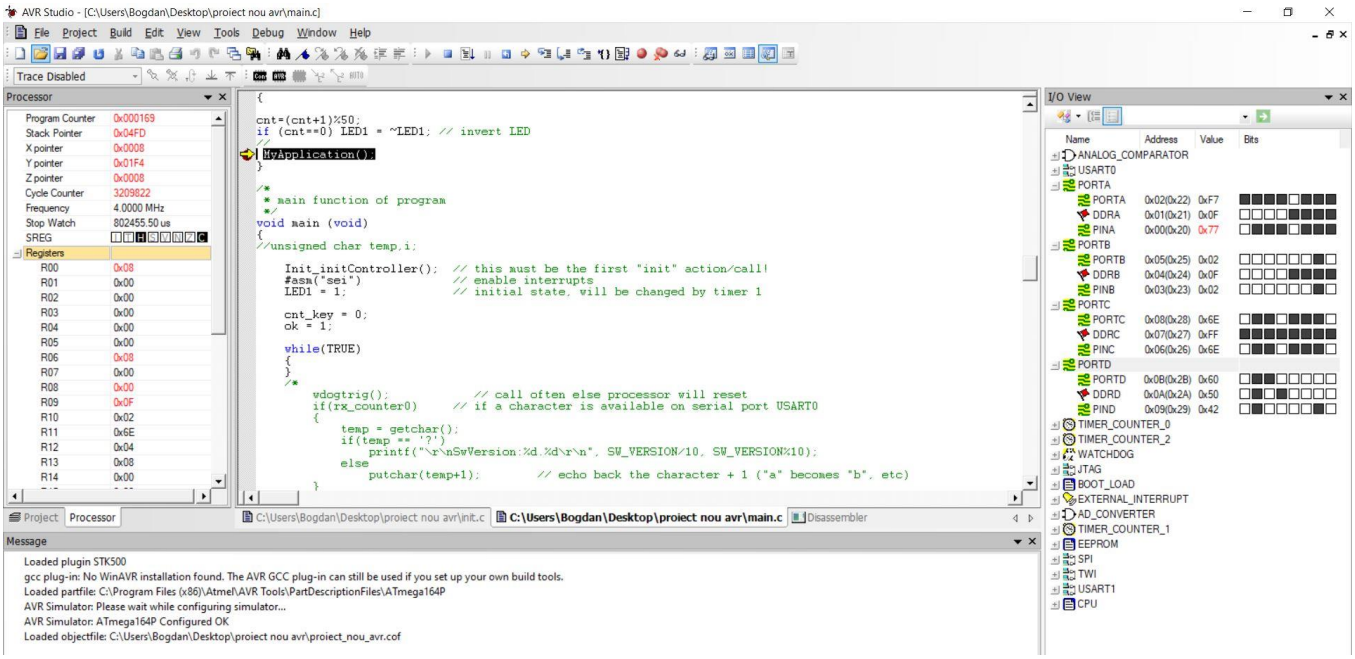


- Starea porturilor dupa introducerea comenzii “2 3 F” (modul STOP):



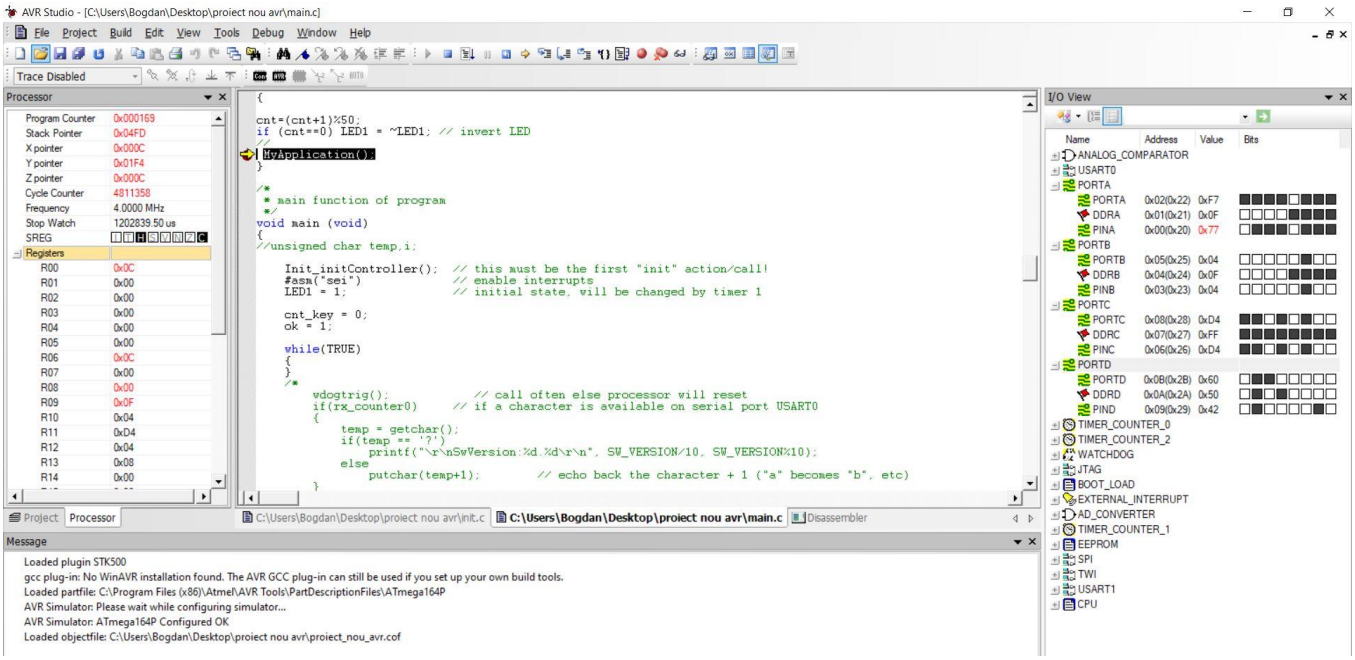
PORTB a luat valoarea 0001, semnificand ca s-a aprins ultimul LED, iar pe PORTC s-a scris comanda asociata.

- Starea porturilor dupa introducerea comenzii “1 2 E F” (modul DTMF):



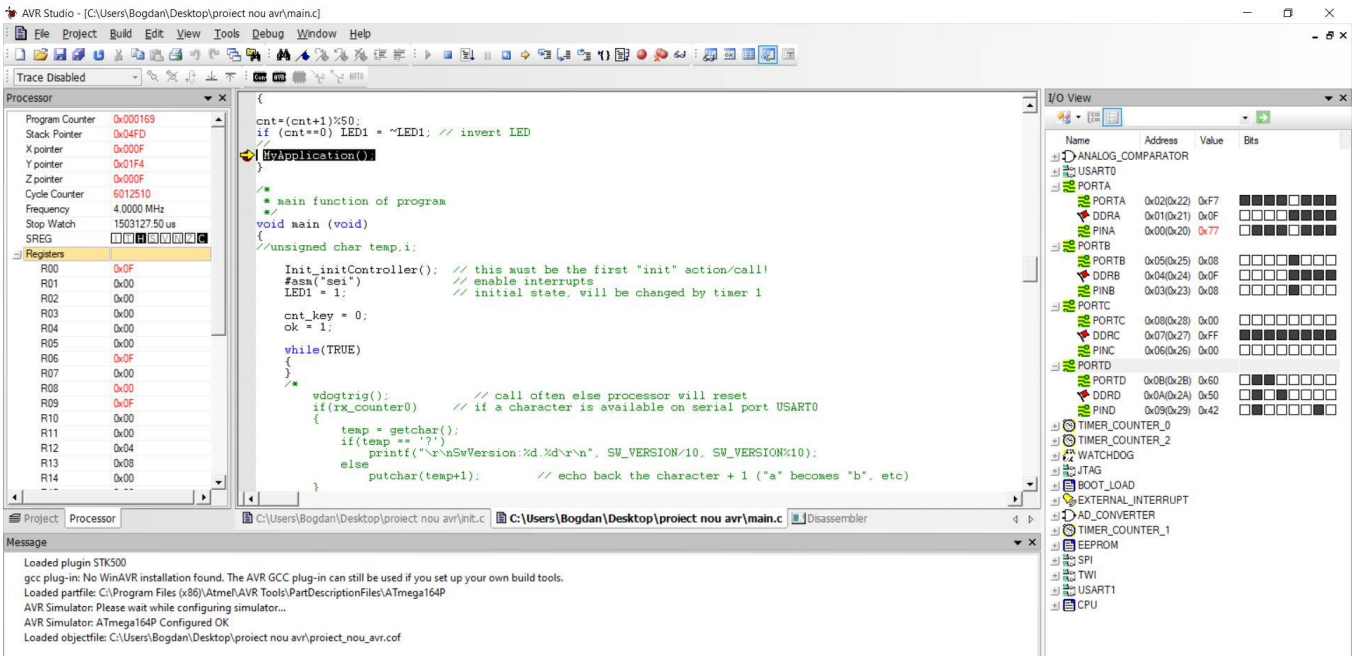
PORTB a luat valoarea 0010, semnificand ca s-a aprins al treilea LED, iar pe PORTC s-a scris comanda asociata.

- Starea porturilor dupa introducerea comenzii “3 1 4 F” (modul frecventa):



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

- Starea porturilor dupa introducerea comenzii “0 4 F” (comanda incorecta):



PORTB 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.

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
```

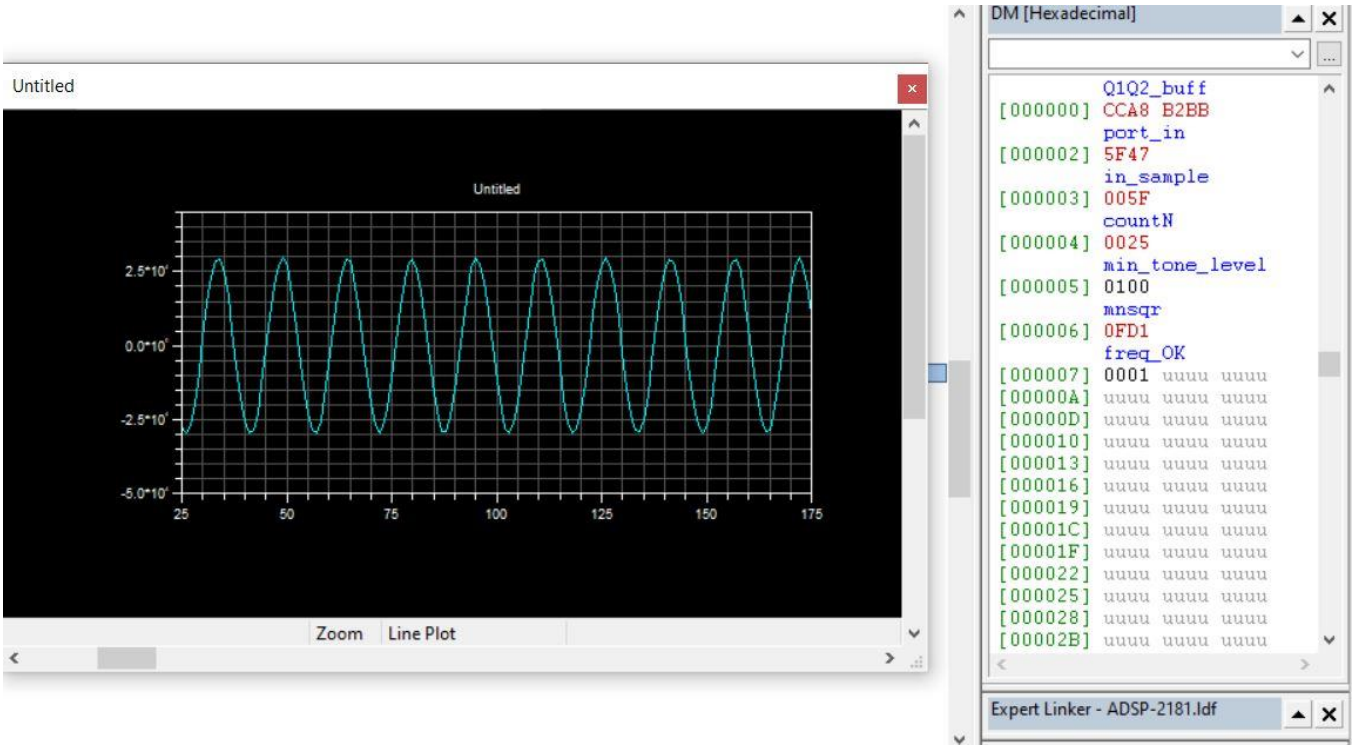
In simulator, am introdus pe streamul de intrare un fisier de semnal DTMF generat mai sus. Pentru verificarea aparitiei unei anumite frecvente, trebuie introdus coeficientul corespunzator pentru una dintre cele doua frecvente.

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

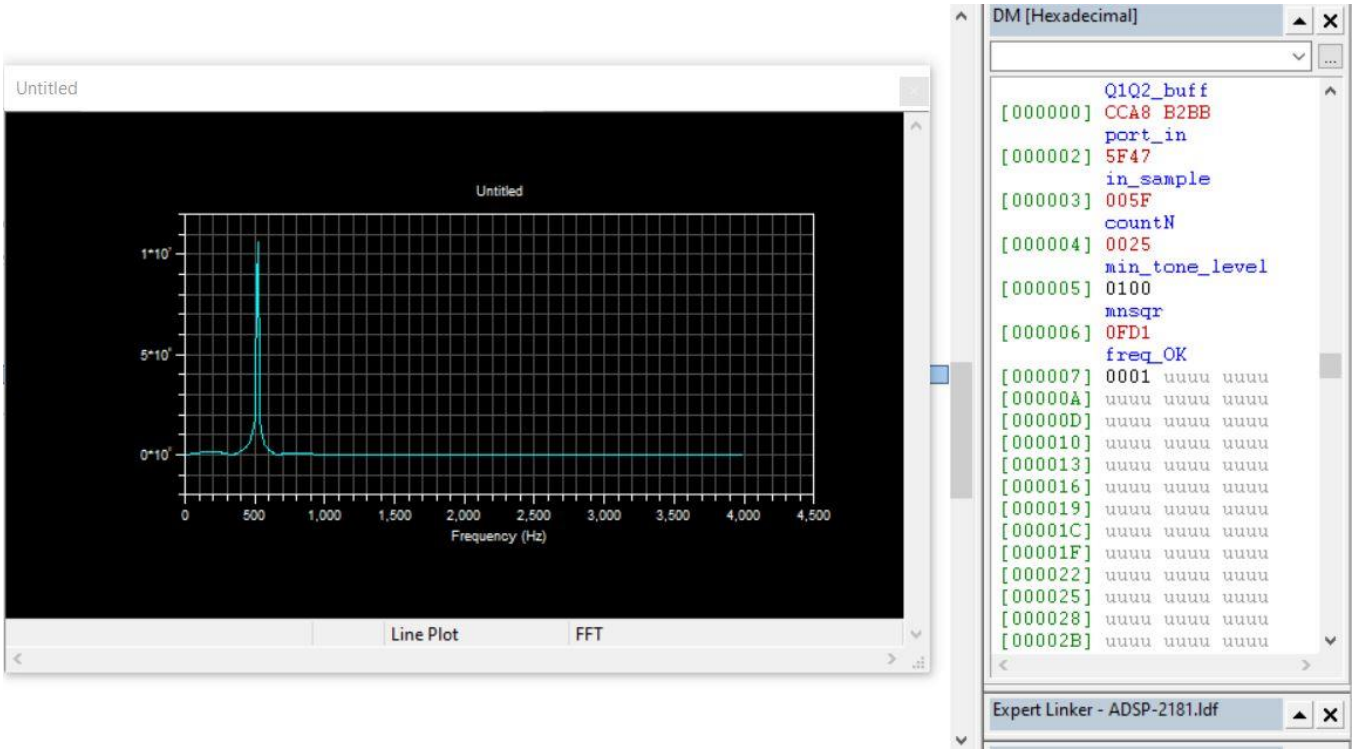
In cazul simularii unui semnal sinusoidal la intrare, testarea se realizeaza o singura data pentru frecvanta acestuia, si se decide daca se potriveste cu frecventa cautata.

Exemplu pentru mod frecventa(semnal sinusoidal):

- Semnal sinusoidal de frecventa 520 Hz:

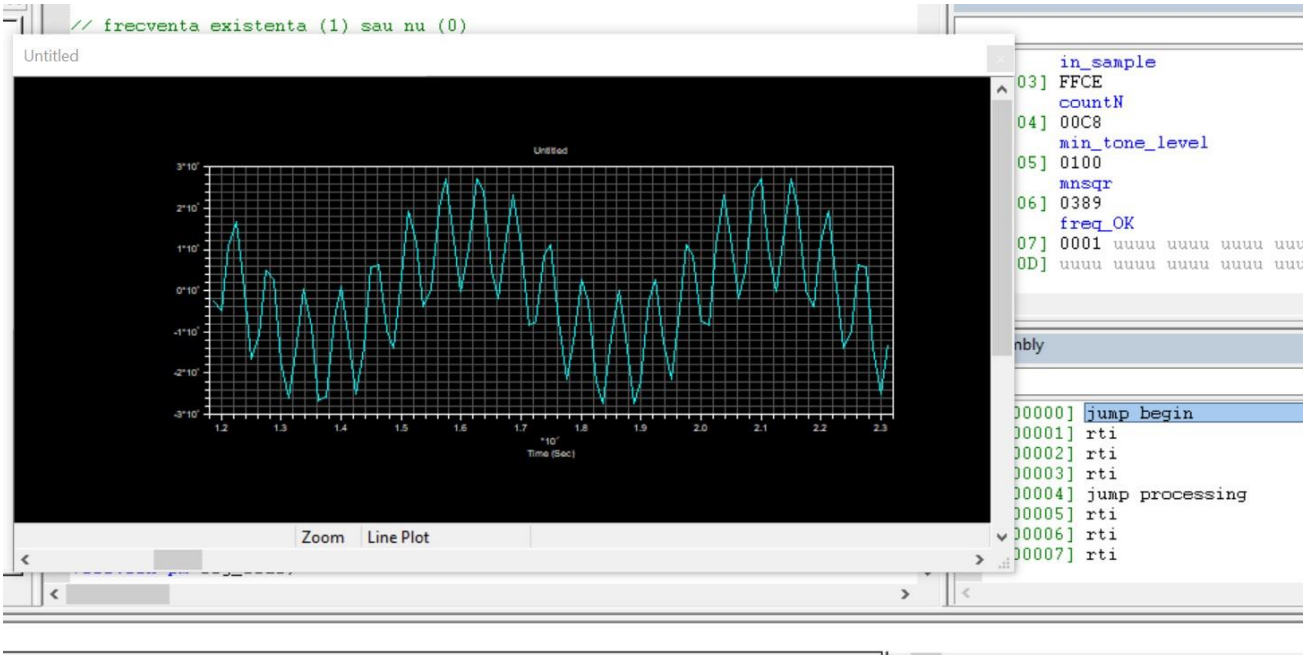


FFT:

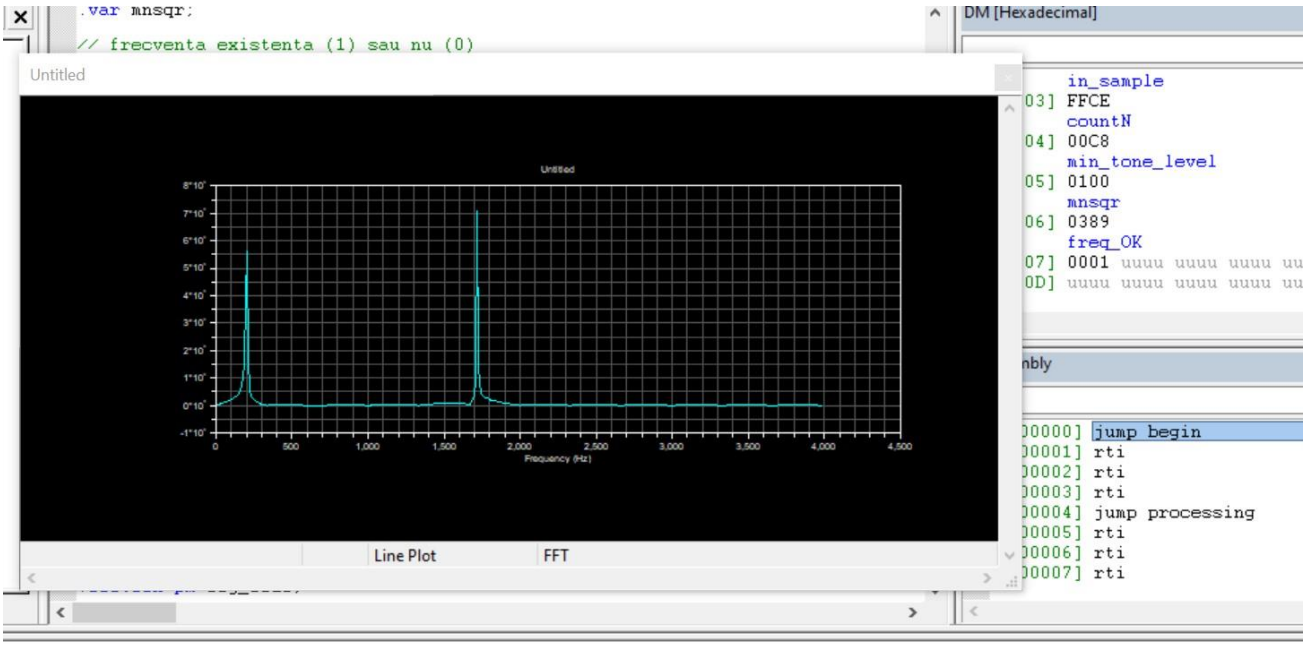


Exemple de semnale DTMF:

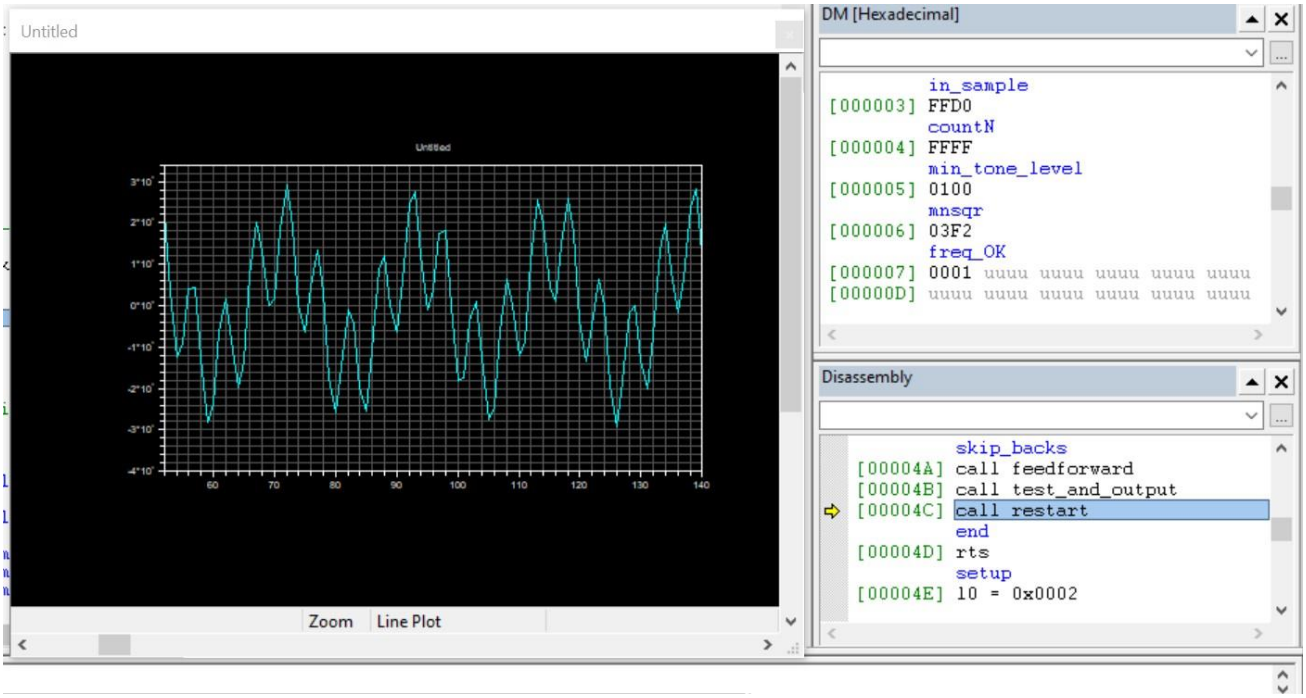
- Semnalul DTMF 200_1720:



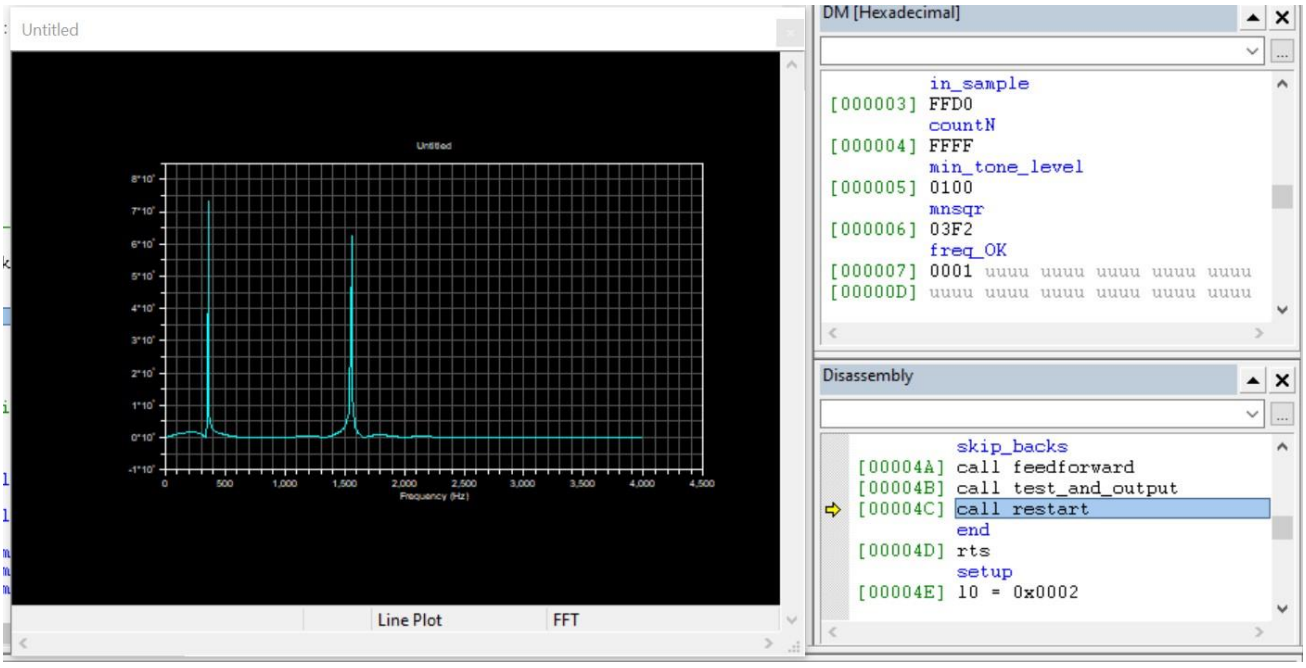
FFT:



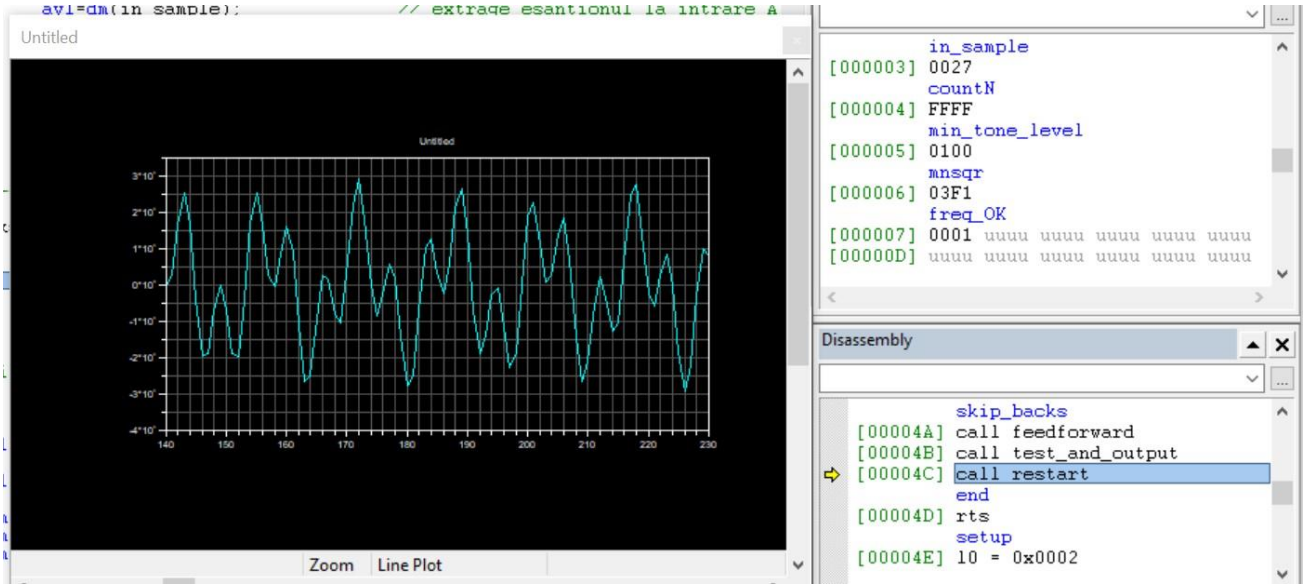
- Semnalul DTMF 360_1560:



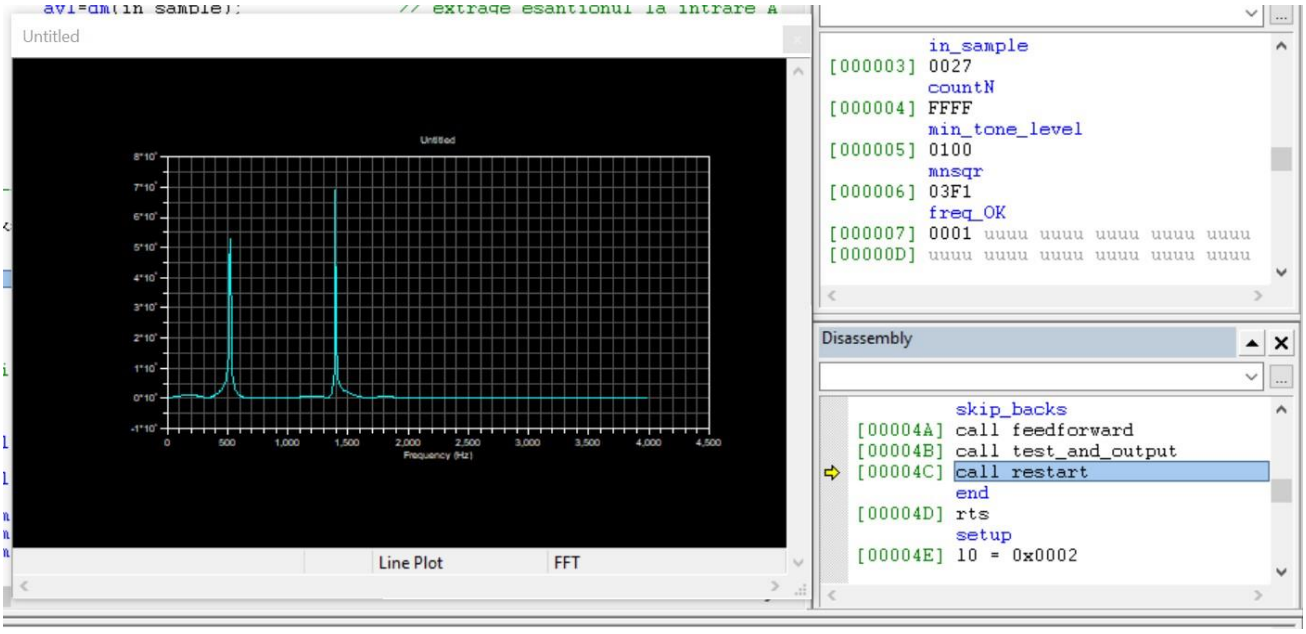
FFT:



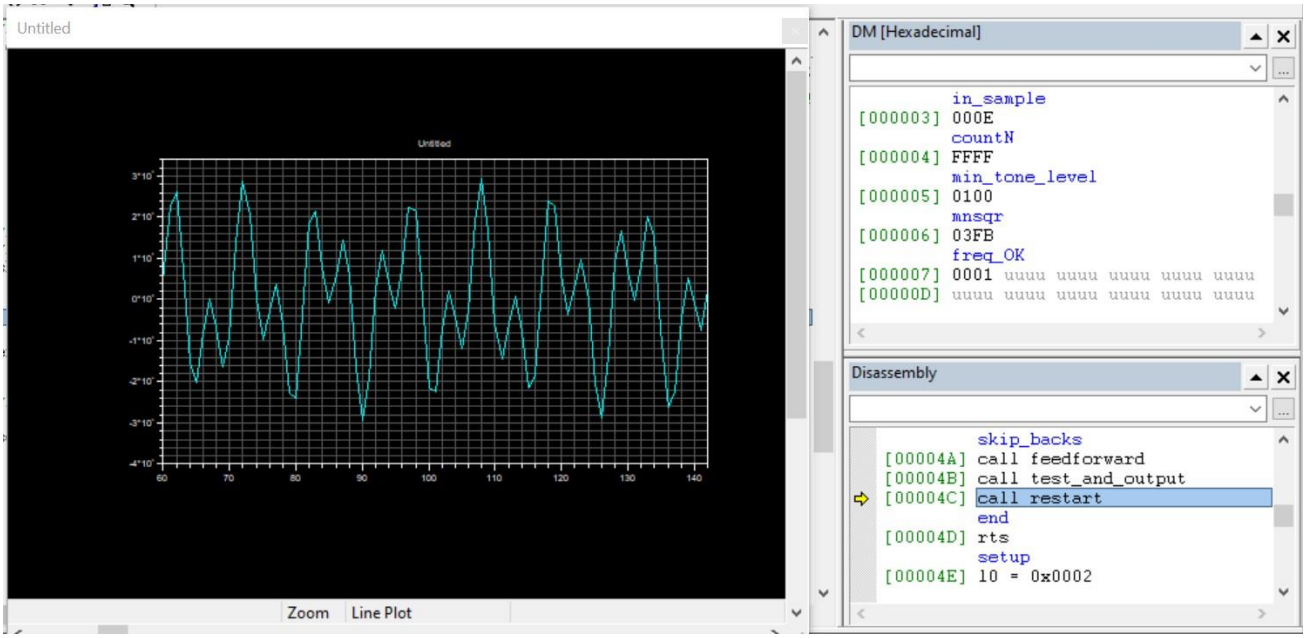
- Semnalul DTMF 520_1400:



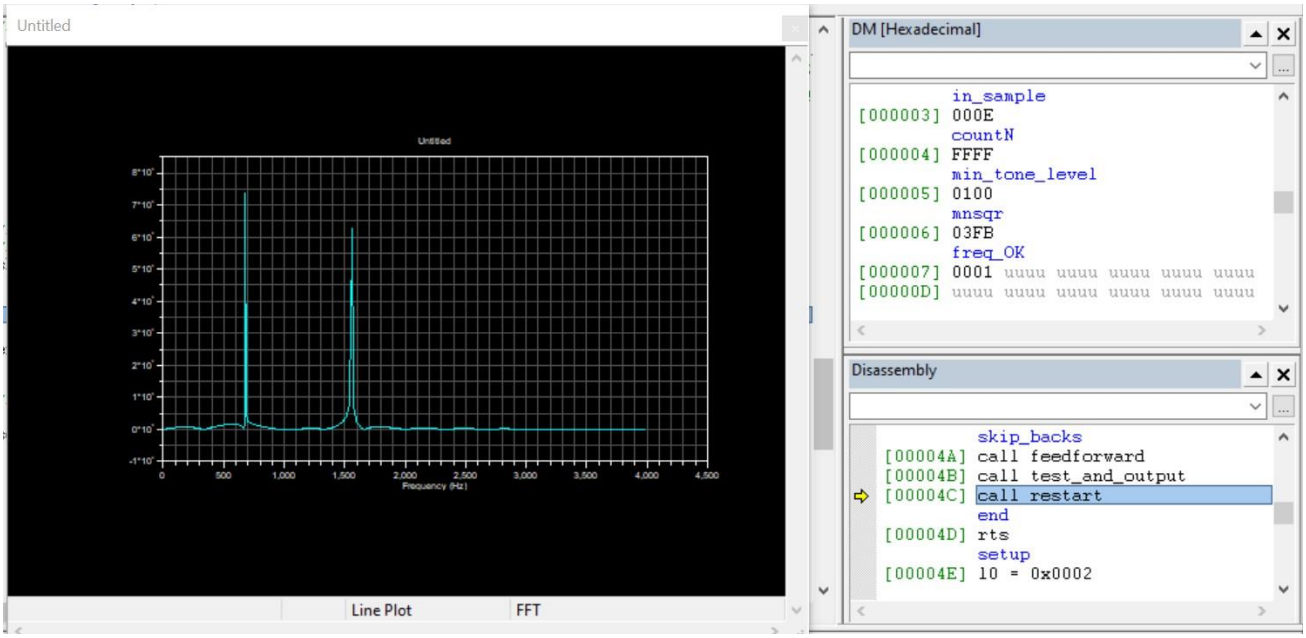
FFT:



- 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/