Universitatea POLITEHNICA din Bucureşti
Facultatea de electronică, Telecomunicații și Tehnologia Informației

Proiect 2 – Contor de energie electrica

Studenti : Belu Florentina-Alexandra
Funda Gheorghe
Grigoras Marin-Jan
Hrisotimos Andrei

Grupa : 433Da

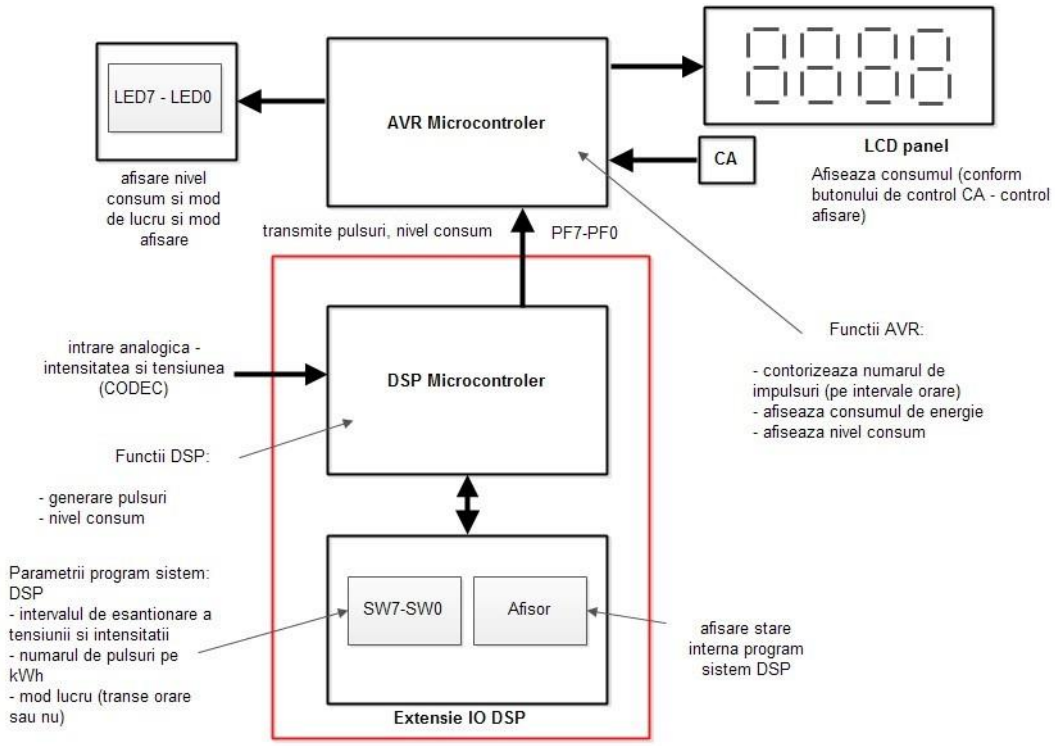
Profesor coordonator
Dr. Ing. Sorin Zoican

An universitar 2022 – 2023

CUPRINS

Tema P2-2023

Sa se realizeze un sistem cu arhitectura din figura 1 cu următoarele specificații:



Arhitectura sistemului

Sistemul contorizează si afișează consumul de energie electrica si este compus din doua subsisteme (AVR si DSP).

Subsistemul DSP măsoară tensiunea si intensitatea, calculează energia consumata si generează un număr **P** de pulsuri pentru 1kWh. Pulsurile generate se vor transmite pe un pin al portului PF. Pe același port se transmite si valoarea intensității curentului consumat. Se va stabili un mod de lucru al contorizării energiei (cu sau fără intervale orare de consum). Intervalul de eșantionare a tensiunii si intensității $\Delta\mathbf{T}$ si numărul de pulsuri **P** per kWh, se vor stabili din SW7-SW0. Se considera puterea maxima de 10kW .

Subsistemul AVR contorizeaza pulsurile, calculeaza si afiseaza energia consumata conform modului indicat de subsistemul DSP.

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 in componenta placa de evaluare EZ-Kit LITE ADSP2181 si o interfața de intrare ieșire (IO DSP).

Se vor implementa:

La nivel hardware:

Subsistemul AVR (cu microcontroler ATMega164) si extensia IO DSP

La nivel software:

1. Descrierea formala a programelor pentru subsistemele AVR si DSP
2. Scrierea codului pentru cele 2 subsisteme (in limbaj C pentru AVR si in limbaj de asamblare ADSP2181 pentru subsistemul DSP)
3. Testarea programelor in CVAVR si Astudio, respectiv in Visual DSP++ 3.5

In final se va verifica functionalitatea sistemlui fizic realizat.
Se va lucra in echipe de 4 studenti (2 pentru AVR si 2 pentru DSP) cu impartirea sarcinilor de proiectare specifice AVR si DSP. Sustinerea este individuala, in toate fazele proiectului.

ETAPA 1

1. Descrierea prelucrarilor (graf, organigrama)
2. Shema bloc, schemele electrice pentru placile de evaluare
3. Proiectarea de principiu a schemei ce completeaza placa de evaluare
4. Rezultate ale simularii variantei de baza a codului pe Visual DSP++, pe CodeVision / AVRStudio

2.Descrierea prelucrarilor

2.1.Organigrama functionare cod ADSP

SW0 = 1 → mod cu transe orare

SW1 = 1 → mod fara transe orare

Δ t, prag → valori fixe

P1 → prag consum mare

P2 → prag consum mediu

P_generare_puls → proces ce va genera un puls de o durata stabilita

PF: 0 – va stoca pulsul, corespunzator consumului unui prag, ce va fi transmis la AVR

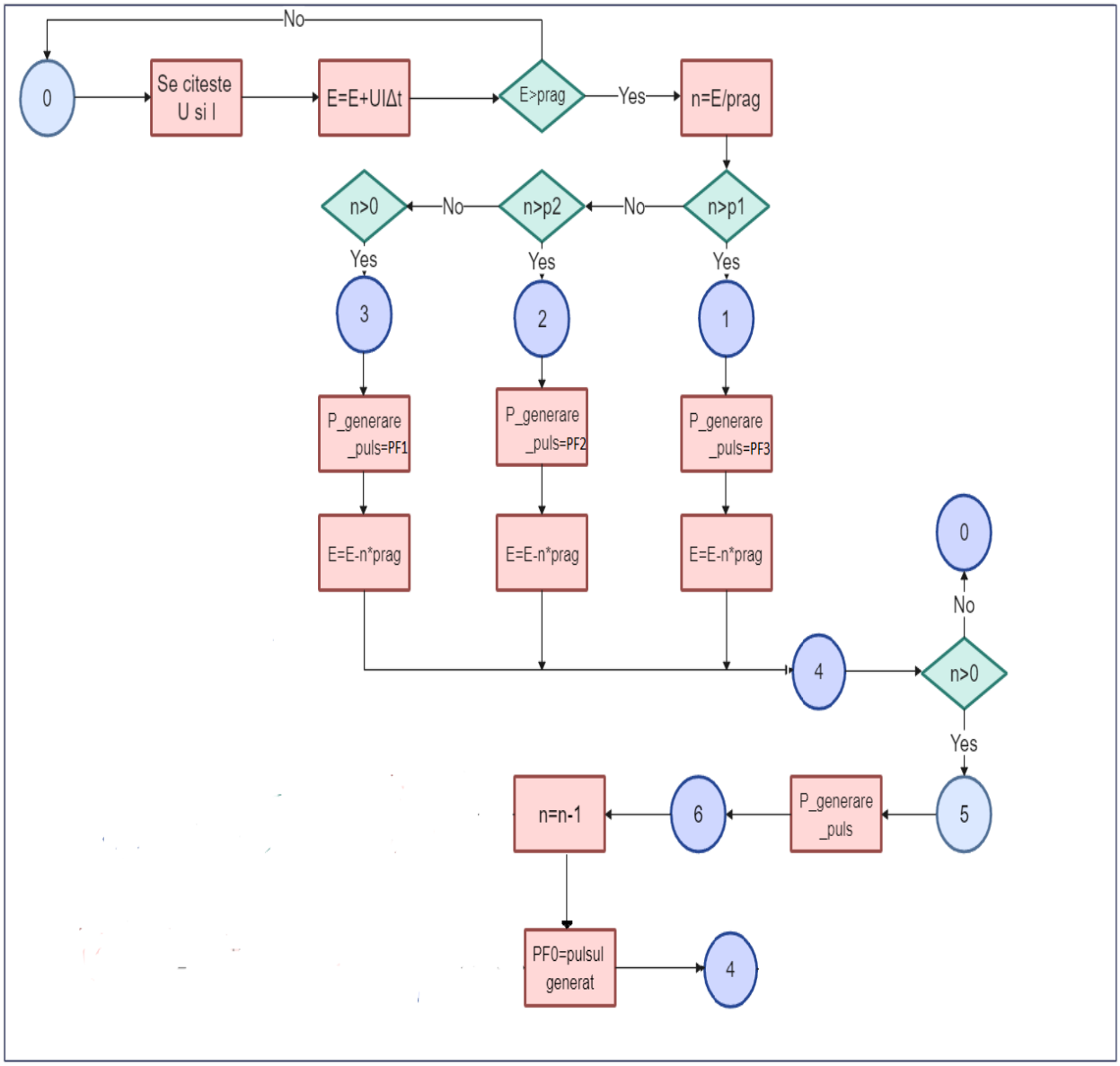
1 – va stoca un bit ce indica un nivel de consum mic

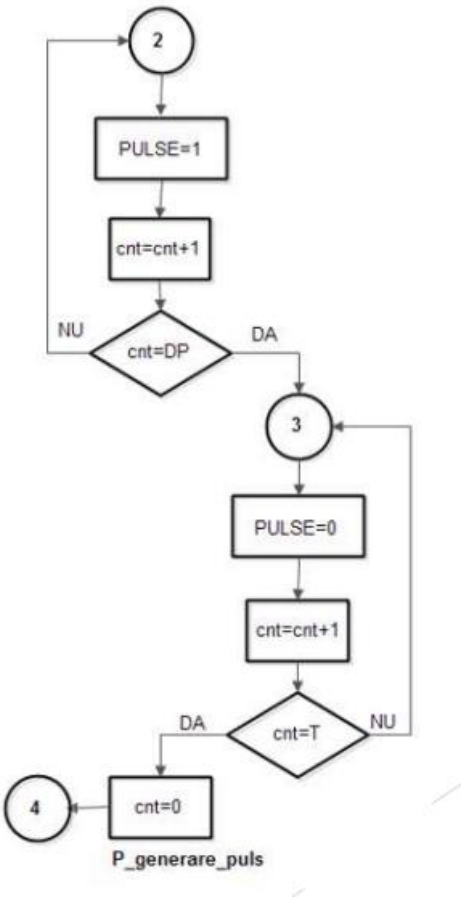
2 – va stoca un bit ce indica un nivel de consum mediu

3 – va stoca un bit ce indica un nivel de consum mare

4 – va stoca un bit ce indica modul de lucru cu transe orare

5 – va stoca un bit ce indica modul de lucru fara transe orare

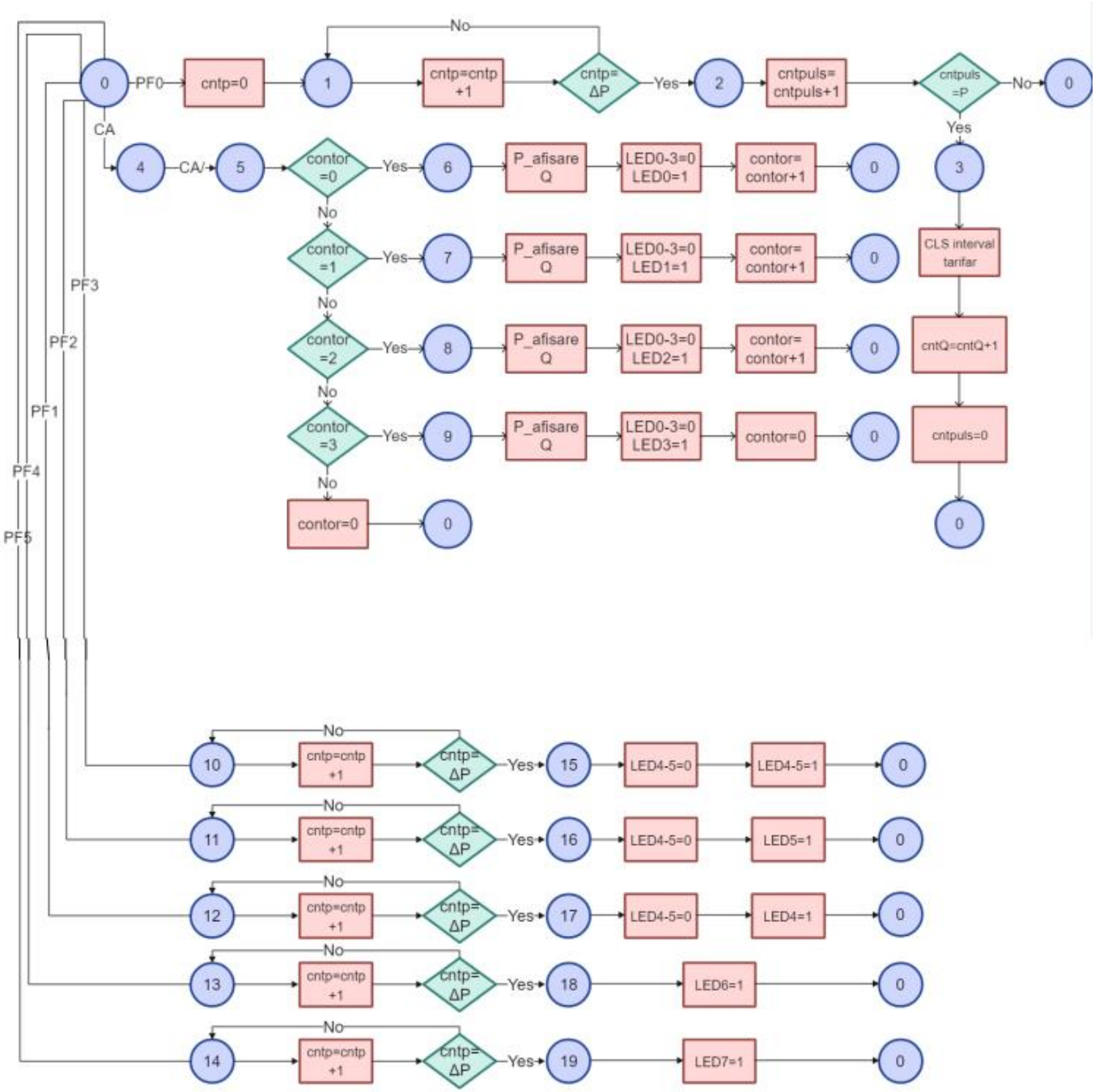

No
0
Se citeste U si I
E=E+UIΔt
E>prag
Yes
n=E/prag
n>0
No
n>p2
No
n>p1
Yes
Yes
Yes
3
2
1
P_generare _puls=PF1
P_generare _puls=PF2
P_generare _puls=PF3
E=E-n*prag
E=E-n*prag
E=E-n*prag
0
No
4
n>0
Yes
n=n-1
6
P_generare _puls
5
PF0=pulsul generat
4

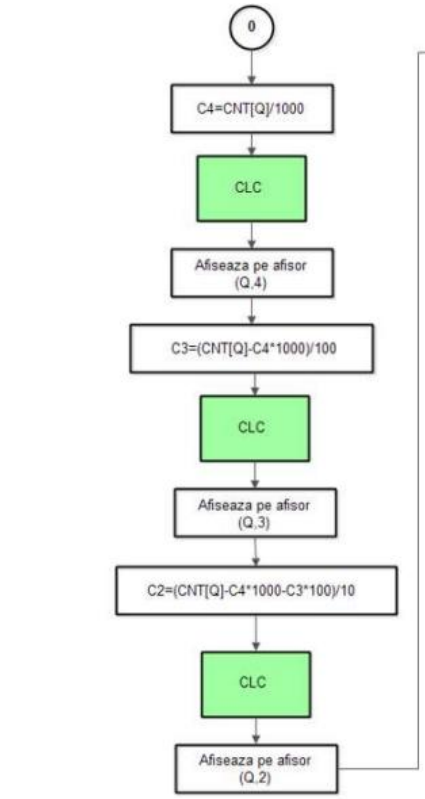


2.2.Organigrama functionare cod AVR

P_afisareQ – procesul secvential de afisare pentru nitervalul terifar corespunzator starii Q.
cntp – contor ce va masura timpul pana la trecerea duratei pulsului venit de la ADSP pentru a nu incrementa CNT(Q) de mai multe ori pentru un singur puls.
contor – va stoca valoarea numarului starii intervalului tarifar pentru care dorim sa afisam consumul pe afisor.
LED :

- 0 → indica afisaj regim tarifar weekend
- 1 → indica afisaj regim tarifar intre orele 0 si H1 (in timpul saptamanii)
- 2 → indica afisaj regim tarifar intre orele H1 si H2 (in timpul saptamanii)
- 3 → indica afisaj regim tarifar intre orele H2 si 0 (in timpul saptamanii)
- 4-5 → indica nivelul de consum :
  - 01 → mic
  - 10 → mediu
  - 11 → mare
- 6 → indica modul de lucru cu transe orare
- 7 → indica modul de lucru fara transe orare


0
PF0
cntp=0
1
No
cntp=cntp +1
cntp= ΔP
Yes
2
cntpuls= cntpuls+1
cntpuls =P
No
0
Yes
3
CLS interval tarifar
cntQ=cntQ+1
cntpuls=0
0
CA
4
CA/
5
contor =0
Yes
6
P_afisare Q
LED0-3=0 LED0=1
contor= contor+1
0
No
contor =1
Yes
7
P_afisare Q
LED0-3=0 LED1=1
contor= contor+1
0
No
contor =2
Yes
8
P_afisare Q
LED0-3=0 LED2=1
contor= contor+1
0
No
contor =3
Yes
9
P_afisare Q
LED0-3=0 LED3=1
contor=0
0
No
contor=0
0
PF1
PF2
PF3
PF4
PF5
10
No
cntp=cntp +1
cntp= ΔP
Yes
15
LED4-5=0
LED4-5=1
0
11
No
cntp=cntp +1
cntp= ΔP
Yes
16
LED4-5=0
LED5=1
0
12
No
cntp=cntp +1
cntp= ΔP
Yes
17
LED4-5=0
LED4=1
0
13
No
cntp=cntp +1
cntp= ΔP
Yes
18
LED6=1
0
14
No
cntp=cntp +1
cntp= ΔP
Yes
19
LED7=1
0

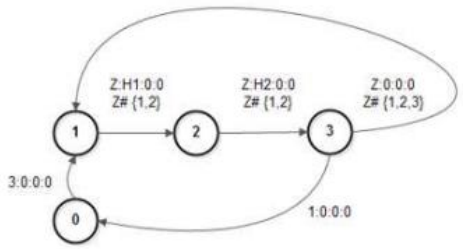


Afiseaza pe afisor (Q,i) : afiseaza cifra i pe afisorul Q
i=4,3,2,1 si Q=0,1,2,3 (regimul de inregistrare)
Intrarea in CLC este cifra curenta de afisat
Se face afisarea pe 4 cifre : C4 C3 C2 C1

P_afisare(Q)

Tabela CLC

0	3F
1	06
2	5B
3	4F
4	66
5	6D
6	7D
7	07
8	7F
9	6F



Eveniment: Z:H:M:S

Z = 1 simbata, Z = 2 duminica, Z = 3, 4,..., 7 = luni, marti, .., vineri
Zilele se numara astfel: 1,2,3,4,5,6,7,1,2,3,...
Initial Z=1; H=0; M=0; S=0
Stare initiala Q=0

CLS - determinare interval de inregistrare

A0: (3:0:0:0, 0); (8:24:24:24, 3);

A1: (3:H1:0:0, 1); (4:H1:0:0, 1); (5:H1:0:0, 1); (6:H1:0:0, 1); (7:H1:0:0, 1); (8:24:24:24, 0);

A2: (3:H2:0:0, 2); (4:H2:0:0, 2); (5:H2:0:0, 2); (6:H2:0:0, 2); (7:H2:0:0, 2); (8:24:24:24, 1);

A3: (4:0:0:0, 0); (5:0:0:0, 0); (6:0:0:0, 0); (7:0:0:0, 0); (1:0:0:0, 3); (8:24:24:24, 2);

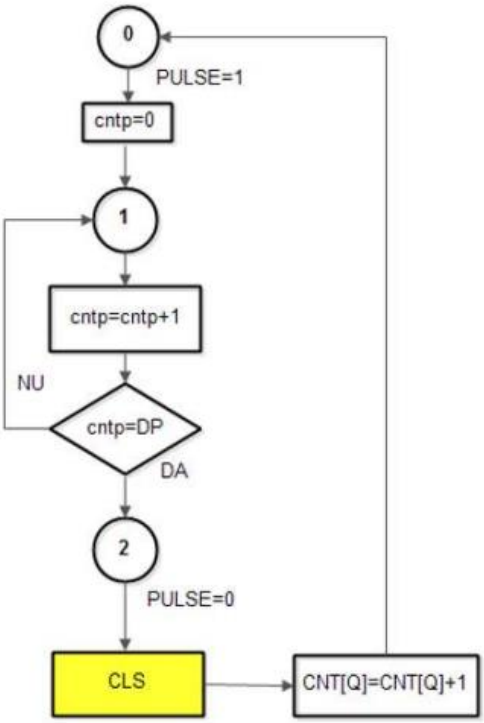
OUT: 0, 1, 2, 3

Q (intervale de inregistrare):

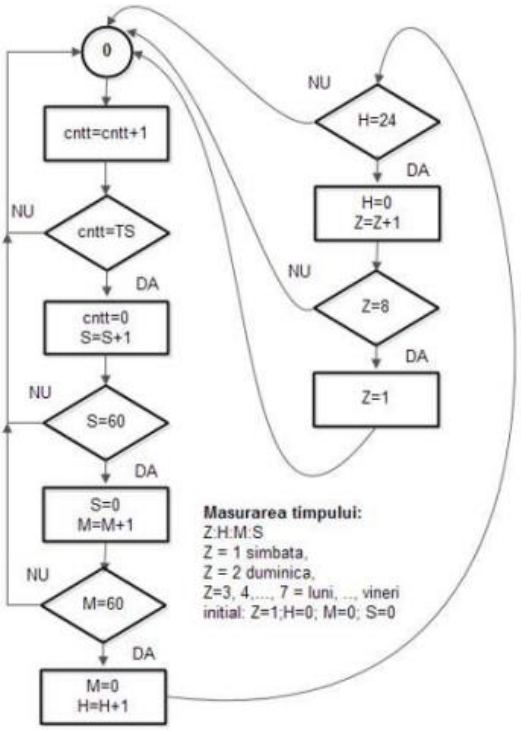
0: in zilele de simbata si duminica
1: intre orele 0 si H1, cu exceptia zilelor de simbata si duminica
2: intre orele H1 si H2, cu exceptia zilelor de simbata si duminica
3: intre orele H2 si 0 (ziua urmatoare), cu exceptia zilelor de simbata si duminica

CLC pentru afişarea pe 7 segmente

```
While (1)
{
Citeste intrarea;
Selecteaza bitii de intrare (variabila X);
Y=TAB(X);
scrie la iesire Y;
}
```



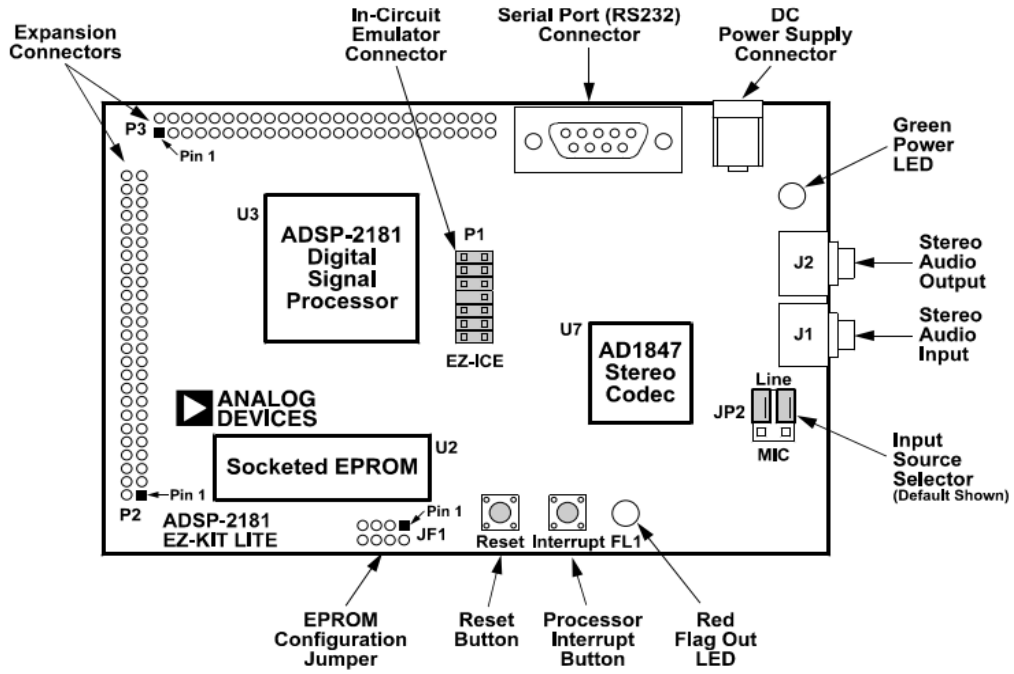
P_Inregistrare_Dif

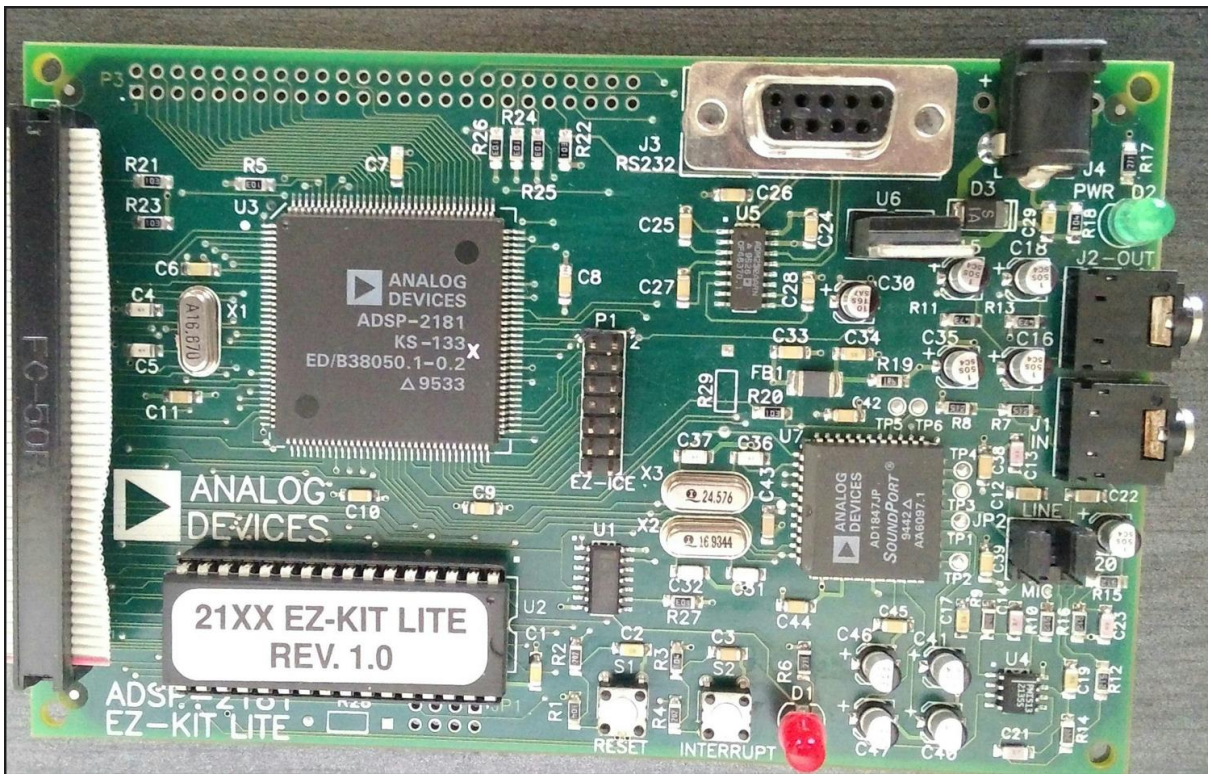


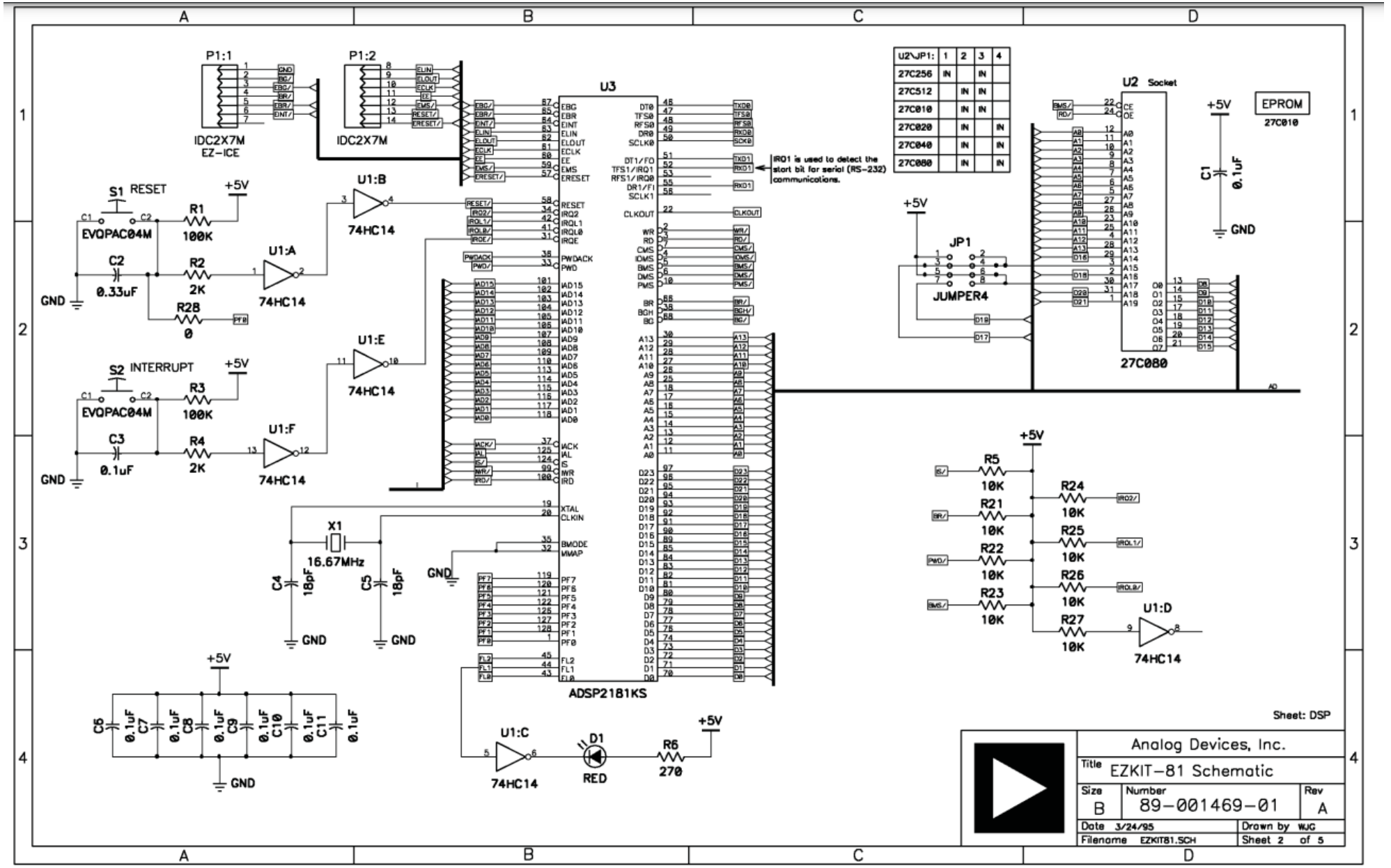
P_Masura_Timp

2. Shema bloc, schemele electrice pentru placile de evaluare

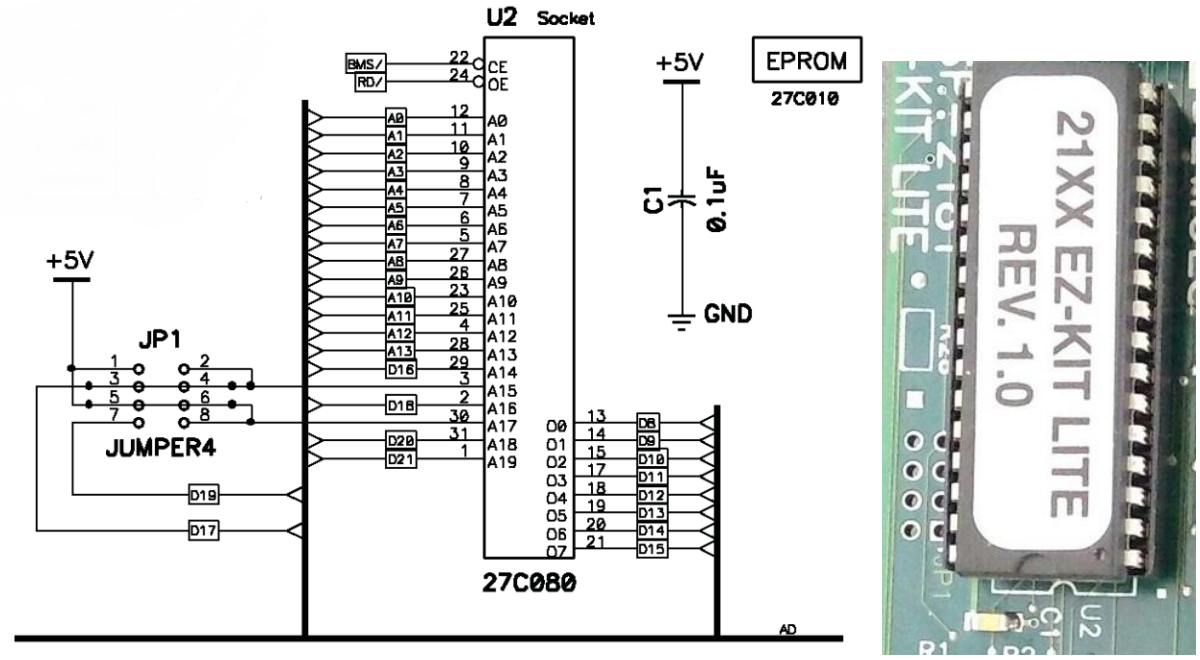
2.1. EZ-Kit LITE ADSP2181






P1:1
IDC2X7M
EZ-ICE
P1:2
IDC2X7M
U3
ADSP2181KS
IRQ1 is used to detect the start bit for serial (RS-232) communications.
U2\JP1:
27C256
27C512
27C010
27C020
27C040
27C080
U2 Socket
EPROM
27C010
27C080
JP1
JUMPER4
S1 RESET
S2 INTERRUPT
EVQPAC04M
R1 100K
R2 2K
R28 0
R3 100K
R4 2K
C1 0.1uF
C2 0.33uF
C3 0.1uF
C4 18pF
C5 18pF
X1
16.67MHz
C5 C7 C8 C9 C10 C11 0.1uF
U1:A
U1:B
U1:C
U1:D
U1:E
U1:F
74HC14
R5 10K
R21 10K
R22 10K
R23 10K
R24 10K
R25 10K
R26 10K
R27 10K
D1
RED
R6 270
+5V
GND
Sheet: DSP
Analog Devices, Inc.
Title EZKIT-81 Schematic
Size B
Number 89-001469-01
Rev A
Date 3/24/95
Drawn by WJG
Filename EZKIT81.SCH
Sheet 2 of 5

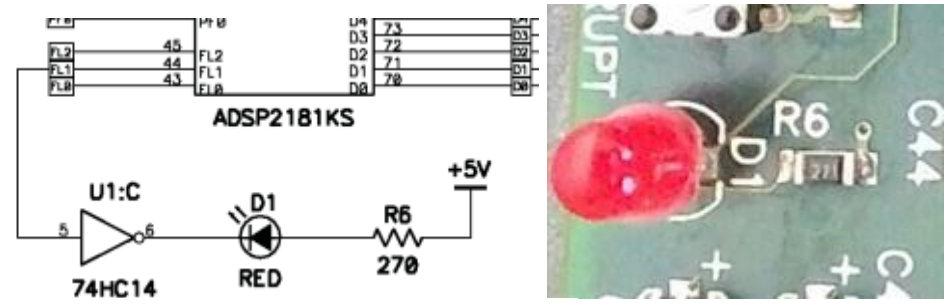
JP1 si Socketed Memory

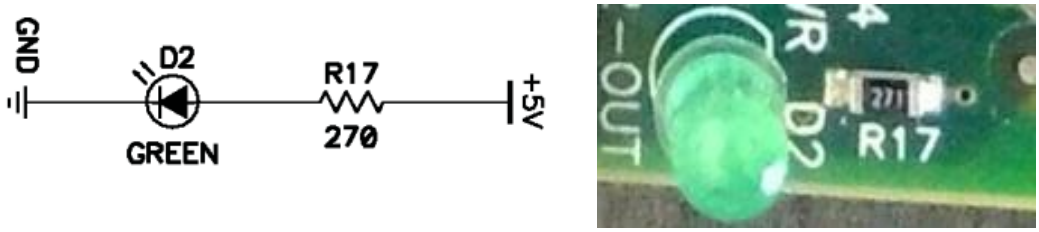


Slotul EPROM asigura 128k x 8 bits de stocare care pot fi incarcati ADSP-2181 cand este programat sa booteze din slotul de EPROM.

Jumperul JP1 este folosit pentru a configura placa pentru o stocare diferita a EPROM de 1Mbit (noi nu utilizam).

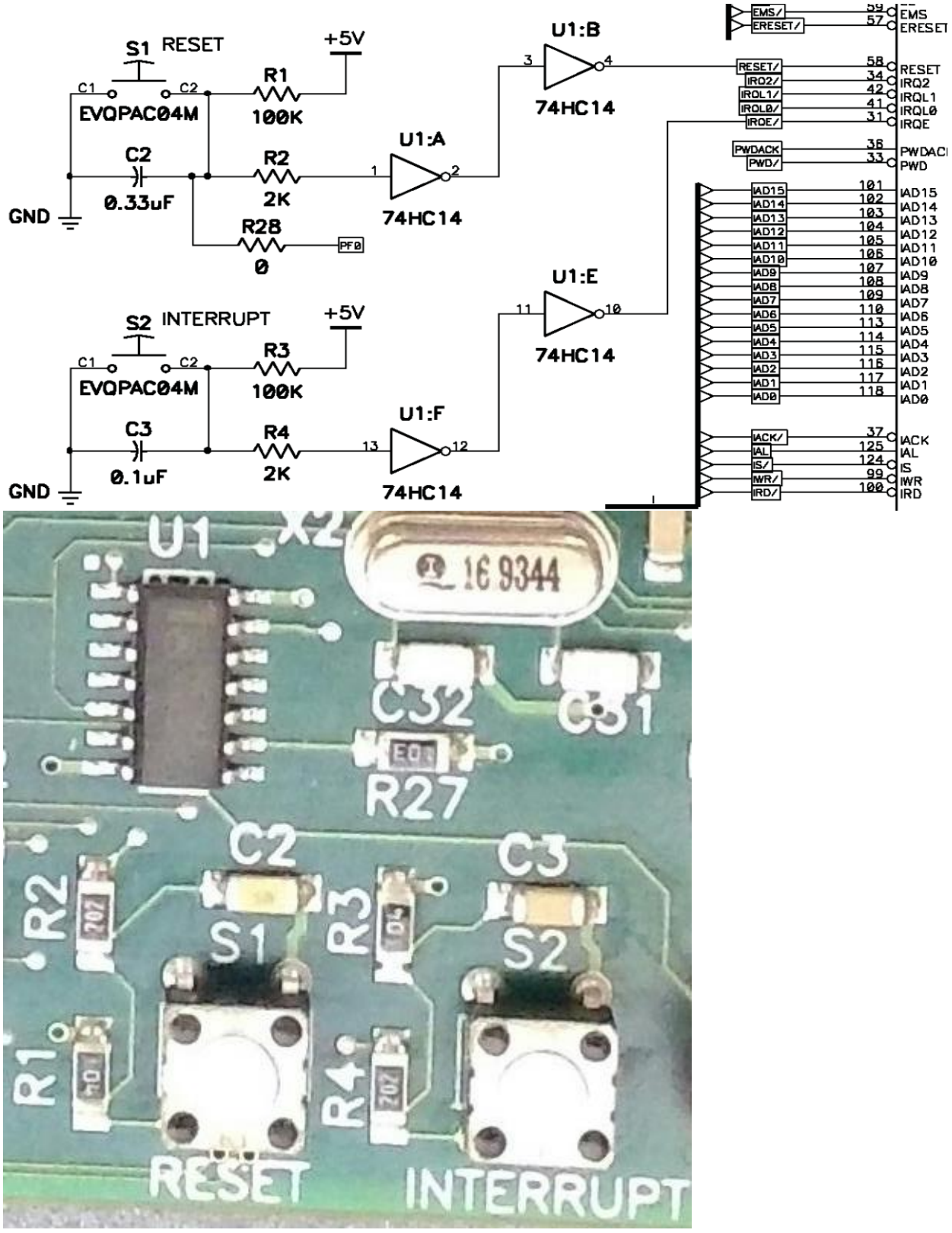
LEDurile utilizatorilor





D1 LED este o dioada rosie, care este controlata de iesirea FL1 (output flag) a ADSP-2181. Putem controla starea acestui indicator scriind intr-un registru intern. D2 LED este o diode verde care se aprinde de fiecare data cand alimentam placa.

Switch-uri

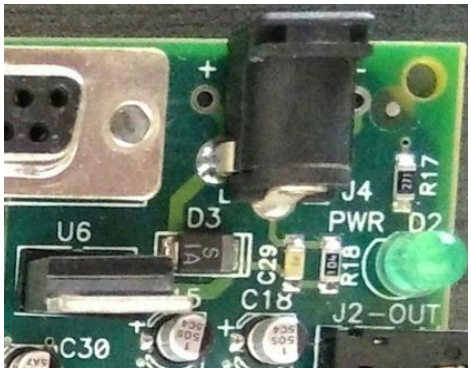


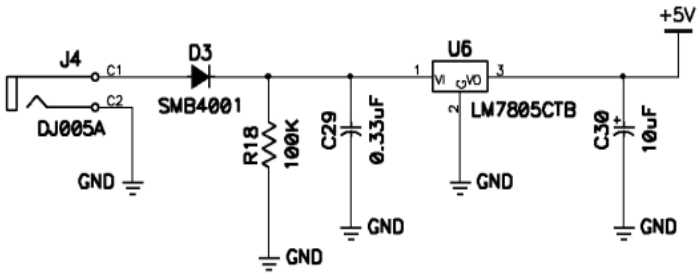
Switchul S1 este butonul de reset. Apasand acest buton face ca procesorul ADSP-2181 si CODECul AD1847 sa intre intr-o stare de reset hardware si sa ramana asa pana cand este eliberat (buton cu revenire).

Switchul S2 este butonul de intrerupere. Apasand butonul determina ADSP-2181 sa primeasca un input de intrerupere de la IRQE/ (edge-sensitive interrupt request). Executa intreruperea curenta IRQE daca este enable.

U1 A, B, F, E fac parte din integratul U1 care este un integrat cu 6 inversoare interne (inversor hex).

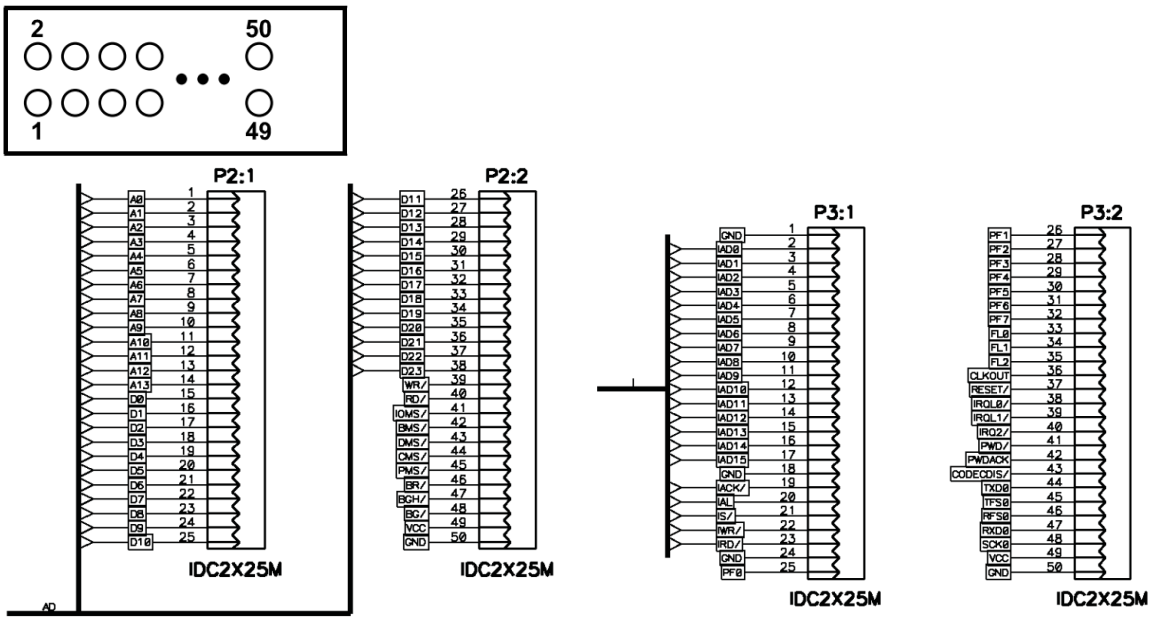
JP4

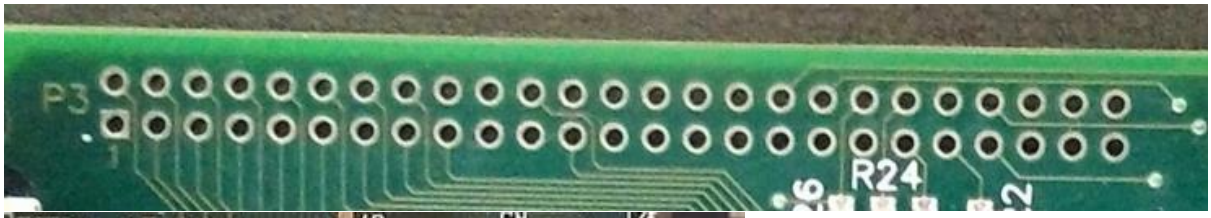


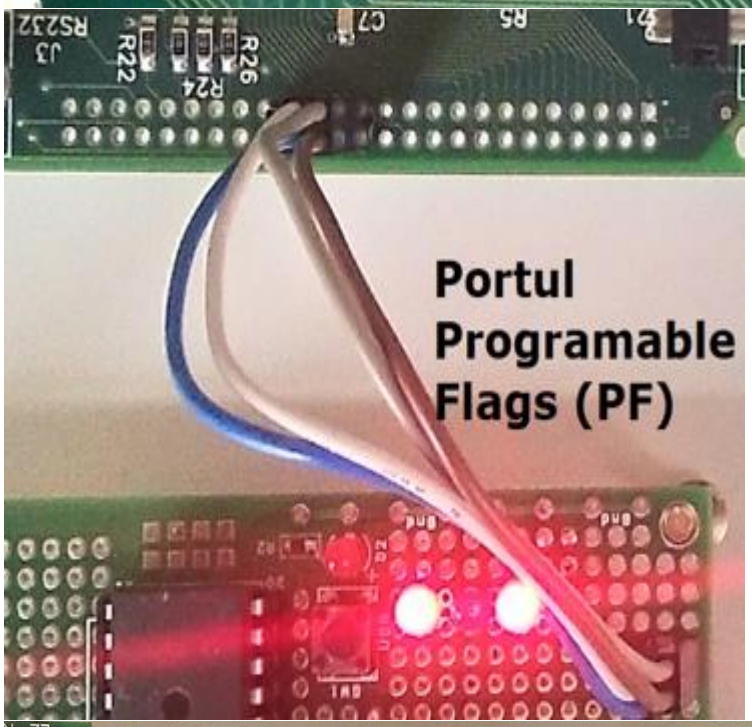


Conectorul J4 este un jack pentru o mufa. Este utilizat pentr a alimenta placa de circuit.

Conectori pentru portul de extindere

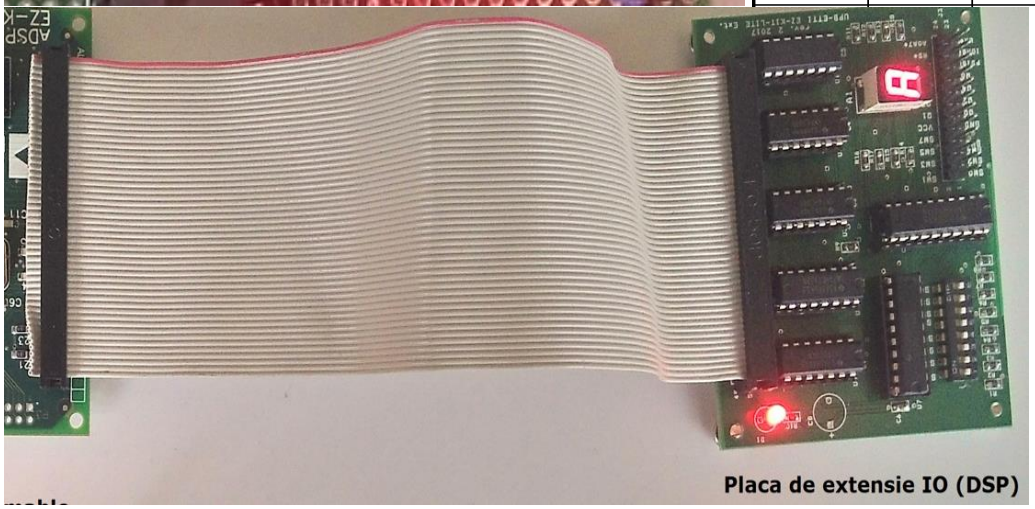






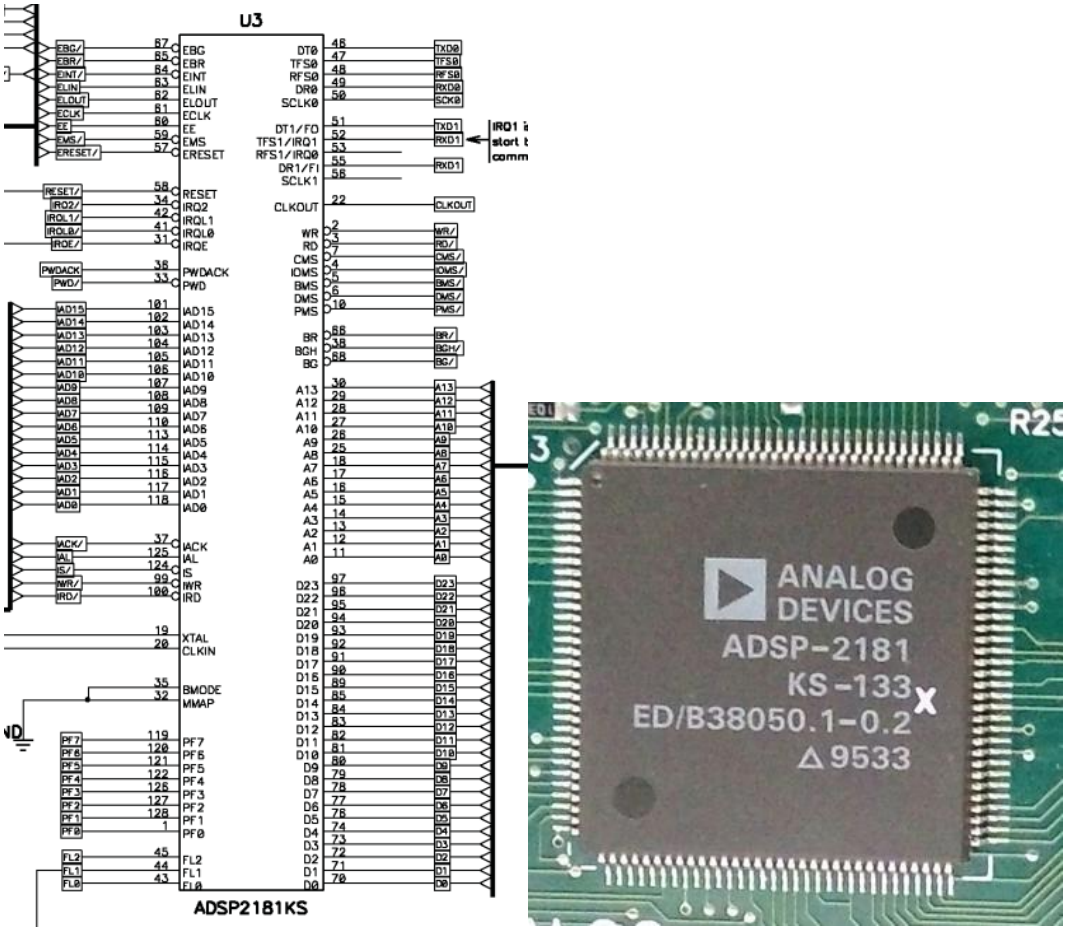
Pulsurile sunt transmise de la subsistemul ADSP catre subsistemul ATmega prin pinii 25-32 ai portului de expansiune P3, care corespund cu semnalele PF0-7.

P3			
Pin Number	**Signal Name**	**Pin Number**	**Signal Name**
17	IAD15	18	GND
19	IACK	20	IAL
21	IS	22	IWR
23	IRD	24	GND
25	PF0	26	PF1
27	PF2	28	PF3
29	PR4	30	PF5
31	PF6	32	PF7



P2 realizeaza conexiunea dintre placa EZ Kit Lite si placa de extensie IO(DSP)

3.2. ADSP-2181



ADSP-218xN(LQFP package cu 128 de pini) (microcomputar) este folosit pentru procesarea semnalului digital.

Un timer cu interval programabil genereaza intreruperi periodic. Un registru de numarare de 16 biti(TCOUNT) decrementeaza la fiecare n cicluri ale procesorului, numarul n este stocat intr-un registru de 8 biti(TSCALE). Cand valoarea registrului de numarare ajunge la 0 o intrerupere este generata si registru de numarare este reincarcat din registrul de perioada(TPERIOD) 16 bit.

Table 6-2. Example Of Timer Operation (Cont'd)

Cycle	TCOUNT	Action
n–1	5	ENA TIMER executed
n	5	Since TSCALE = 1, no decrement
n+1	5	Decrement TCOUNT
n+2	4	No decrement
n+3	4	Decrement TCOUNT
n+4	3	No decrement
n+5	3	Decrement TCOUNT
n+6	2	No decrement
n+7	2	Decrement TCOUNT
n+8	1	No decrement
n+9	1	Decrement TCOUNT
n+10	0	No decrement
n+11	0	Zero reached, interrupt occurs load TCOUNT from TPERIOD
n+12	5	No decrement
n+13	5	Decrement TCOUNT
n+14	4	No decrement
n+15	4	Decrement TCOUNT, etc.

ADSP are 2 porturi seriale (trimite sau primeste biti de informatie cate unul pe rand) sincrone (bitii transmisi sunt sincronizati printr-un semnal de clock) SPORT0, SPORT1..

Pentru initializarea pinilor I/O PF0-7 se folosec registrele PFTYPE (determina directia 1=iesire, 0=intrare) si PFDATA (folosit pentru a scrie sau a citi valori pe pini).

Tactul clockului pote fi generat de un cristal sau de un semnal de clock compatibil. Inputul CLKIN nu poate fi oprit sau schimbat in timpul unei operatii.
Oscilator cu cristal conectat la CLKIN si XTAL.

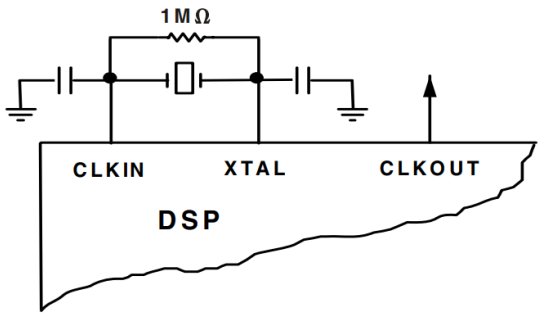
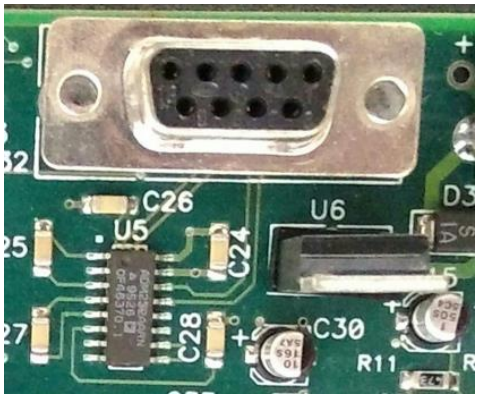
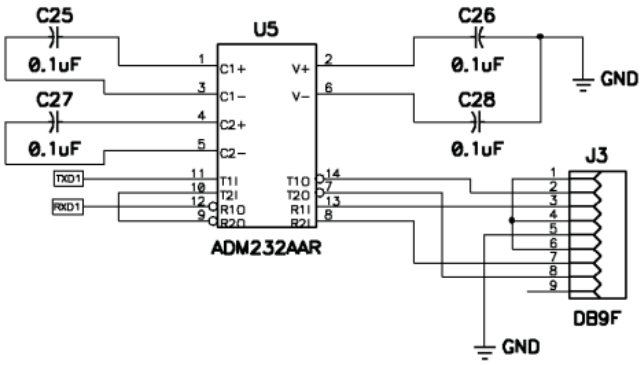


Figure 3. External Crystal Connections

Semnalul de RESET/ initiaza un master reset (factory reset) care trebuie facut doar cand este pornit si ajunge la tenisunea necesara ADSP ului pentru o initialiare buna(trebuie tinut suficient ca clockul intern sa se stabilizeze).

SERIAL PORTS



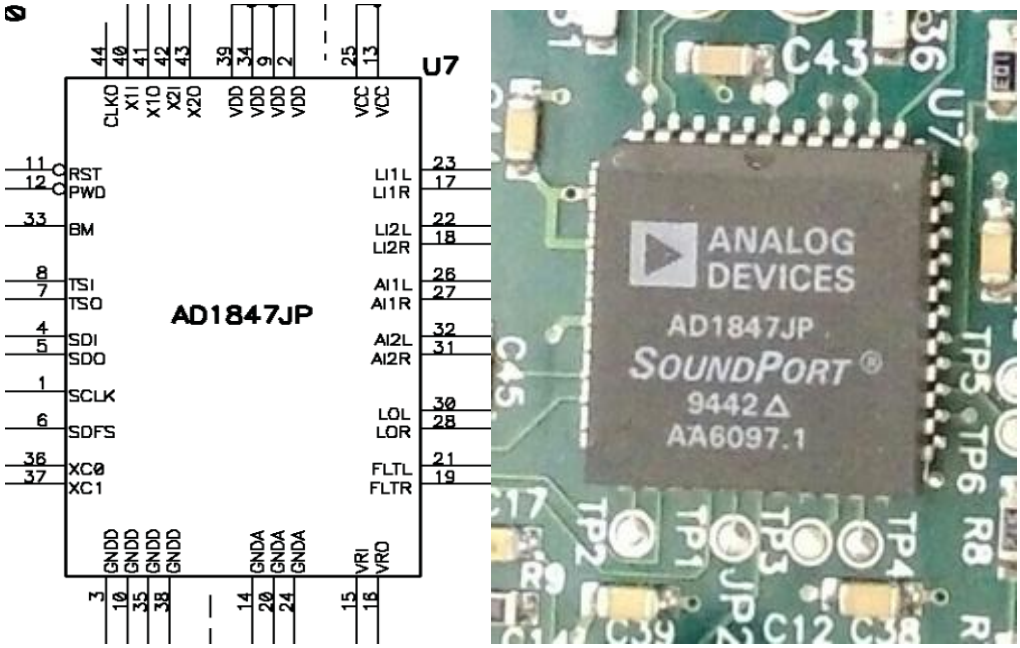


ADSP dispunde de doua porturi seriale bidirectionale sincrone (SPORT-uri), SPORT0 si SPORT1. SPORT-urile pot opera la o frecventa maxima de 1 x frecventa ceasului, oferind fiecaruia o rata maxima de transfer de date de 30 Mbit/sec.
SPORT0 este conectat la codec-ul AD1847 aflat pe placa. Semnalul CODECD este disponibil pe conectorul P3. SPORT1 este conectat la interfata RS-232 (interfata serial) si este utilizat ca receptor/transmitator asincron (UART). Comunicarea intre monitor si gazda se face prin intermediul SPORT1.

RESET

Pentru a efectua o resetare la pornire, se deconecteaza alimentarea placii timp de cel putin trei secunde si apoi se reconecteaza. De asemenea, se poate reseta placa in timpul functionarii prin intermediul comenzii DEBUG si RESET in depanator.

3.3. CODEC

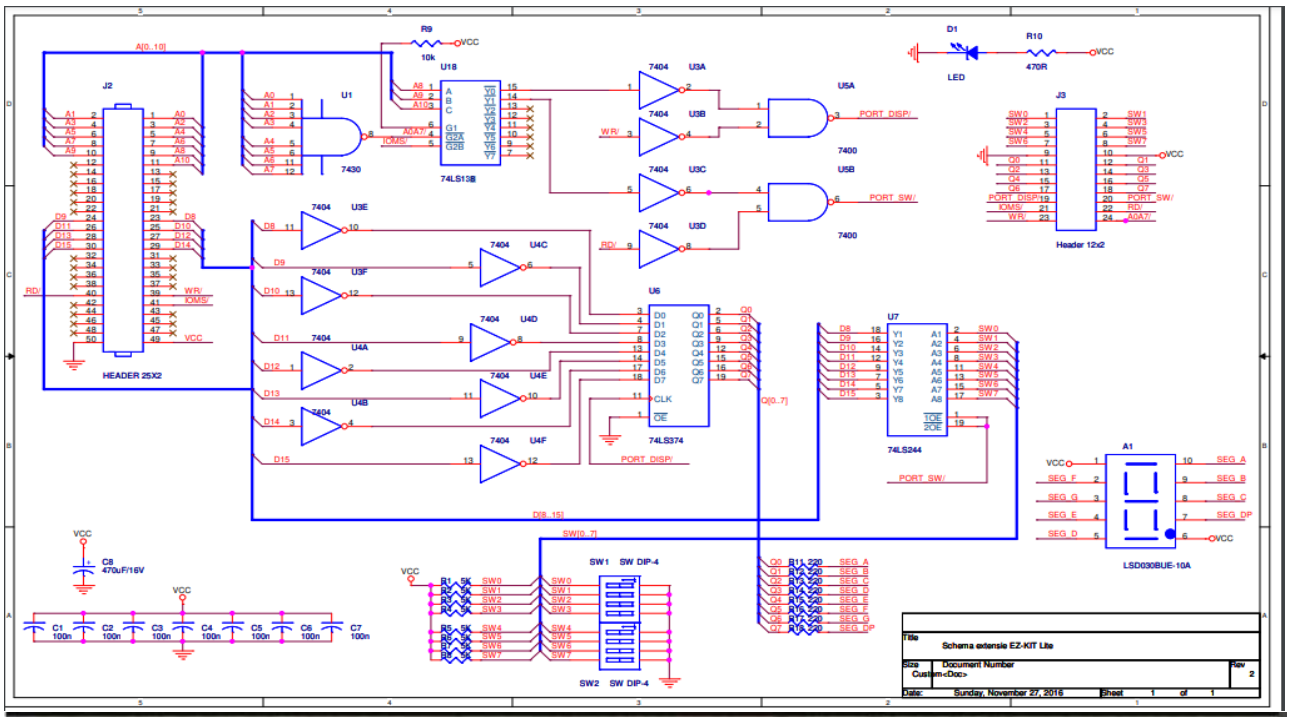


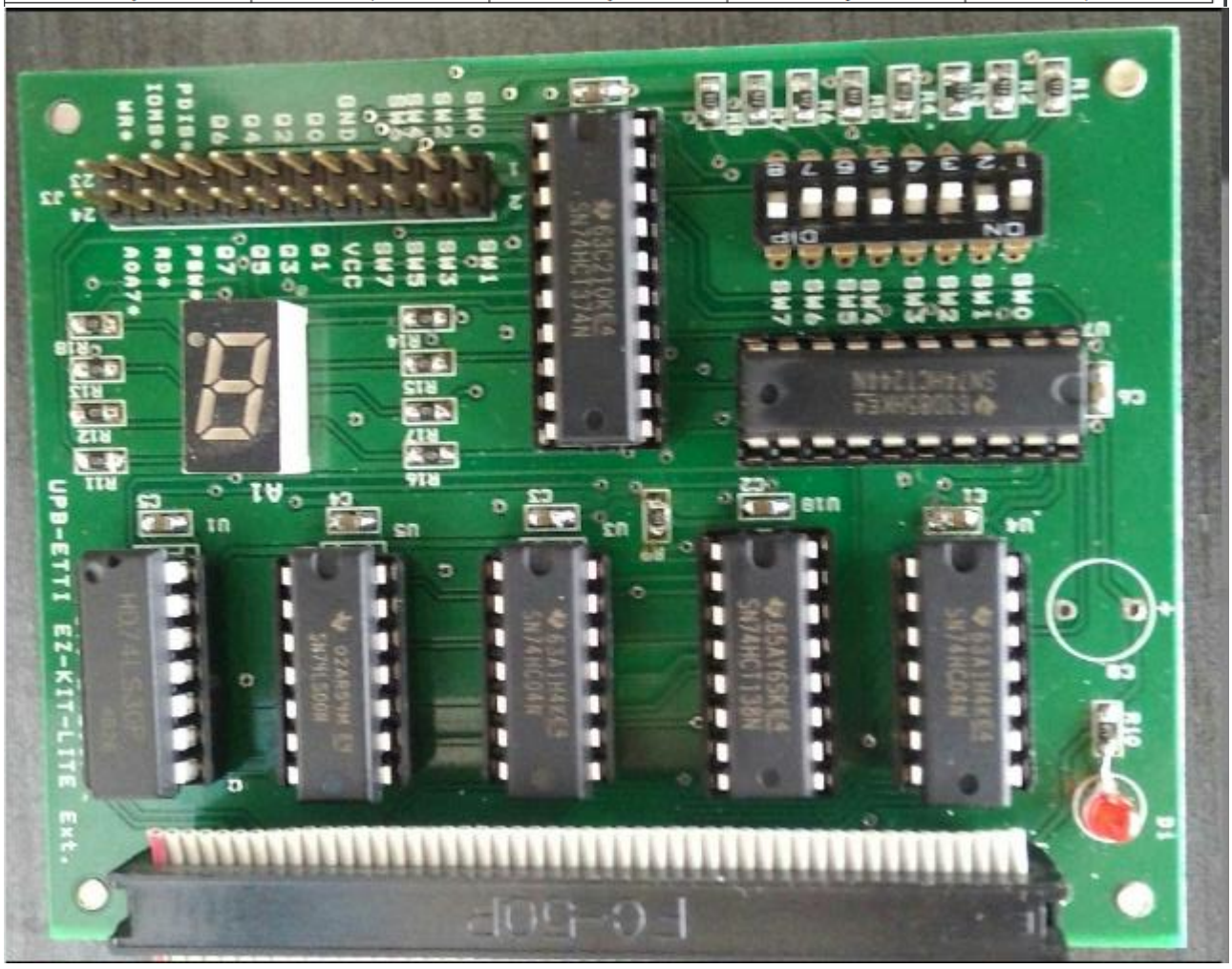
ADSP proceseaza doar semnale digitale. Aceste semnale provin din semnale analogice din lumea reala. ADSP deobicei proceseaza intrari de la convertoare analogice-digital (ADC) si trimite iesirea catre un convertor digital-analogic (DAC).

CODEC incorporeaza doua functii: COdor/DECodor, echivalent cu conversiile ADC/DAC. Astffel, CODECul este un convertor ADC/DAC pe un singur cip care poate fi programat, spre deosebire de un ADC sau DAC. ADSP poate programa diversi parametrii ai CODECului in functie de aplicatia dorita si poate selecta sursa sau destinatia pentru unde informatia poate fi luata sau trimisa dupa prelucrare. Programarea CODECului se face printr-un registru intern.

Dupa programare, AD1847 (CODECul) genereaza un semnal de clock folosit pt a transfera datele catre SPORT0. ADSP-2181 initiaza toate transmiterile de date cu AD1847 prin transmiterea unui puls de sincronizare. Chiar daca AD1847 transmite semnalul de clock, poate sa nu fie gata pentru prelucrarea datelor. Initializarea AD1847 se realizeaza prin transmiterea a 13 cuvinte de control (Numarul de cuvinte de programare) care se afla intr-un buffer circular al CODECului. Aceasta actiune se realizeaza de obicei prin rutina de intreruperi a SPORT0 TX.

3.4. Placa de extensie I/O

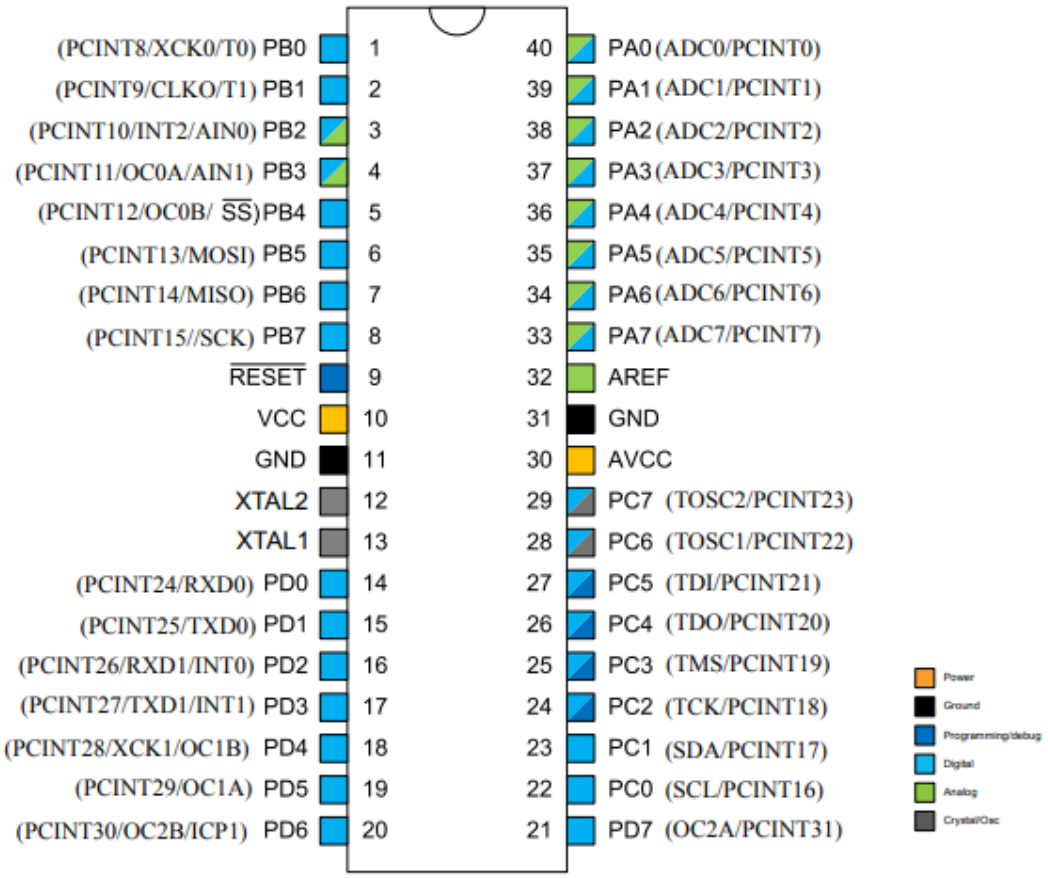




3.5. ATmega 164P

ATmega 164P (DIP package cu 40 de pini) este folosit la numararea pulsurilor, calcularea si afisarea energiei consumate.

5.1.1. PDIP



Prezentarea pinilor:

VCC- alimentare
GND- masa

Port A (PA[7:0])- Este un port I/O bidirectional de 8 biti cu o rezistenta pull-up (rezistenta folosita in circuite pentru a asigura un nivel logic bine definit de 1 sau 0 si nu de Hi-Z, impedanta mare, in cazul butoanelor care sunt un circuit deschis atunci cand nu sunt apasate; butoanele in cazul nostu fiind conectare la masa, atunci cand le apasam transmit valoarea de 0). De asemenea, prezinta si functii alternative.

Port B (PB[7:0]) Este un port I/O bidirectional de 8 biti cu o rezistenta pull-up interna selectabila individual pentru fiecare bit. Acest port poate servi functii pentru caracteristici special.

Port C (PC[7:0]) Este un port I/O bidirectional de 8 biti cu o rezistenta pull-up interna selectabila individual pentru fiecare bit. Acest port poate servi functii pentru interfata JTAG(pentru verificarea si testarea circuitului dupa fabricare) si caracteristici speciale.

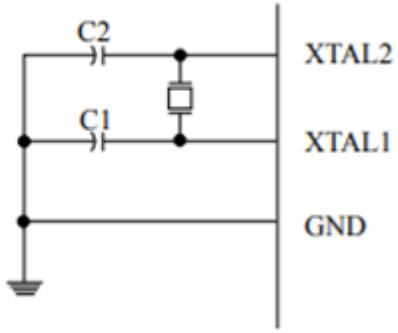
Port D (PD[7:0]) Este un port I/O bidirectional de 8 biti cu o rezistenta pull-up interna selectabila individual pentru fiecare bit. Acest port poate servi functii pentru caracteristici special.

RESET/(active in 0) Resetarea intrarii.

XTAL1 Intrare pentru amplificatorul inversors conectat la oscilatorul de cristal si input pentru circuitul ceasului intern.

XTAL2 Iesire din amplificatorul inversors conectat la oscilatorul de cristal.

Figure 10-2. Crystal Oscillator Connections

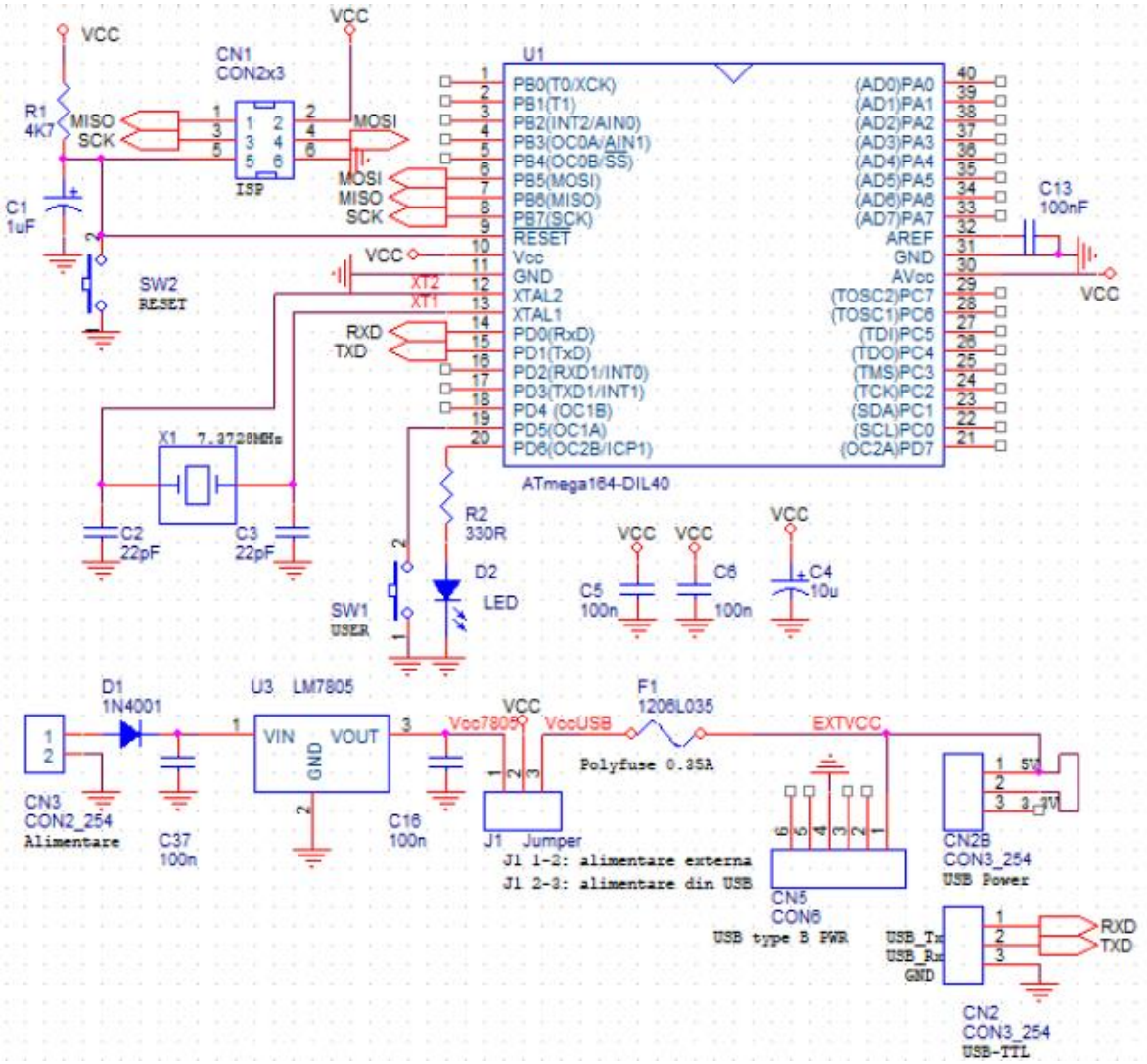


AVCC Este pinul pentru sursa de current pentru Port A si convertorul analog-digital.

AREF Este pinul de referinta pentru convertorul analog-digital.

Pinii I/O:
Cu exceptia pinilor de alimentare si de cuarț, toti pinii procesorului pot fi folosiți in mod independent unul de altul ca pini digitali de intrare (la care se pot lega butoane, senzori etc) sau de iesire (la care se pot conecta LED-uri sau alte elemente de actionare).

Procesorul si sursa de alimentare



R1, C1 formeaza circuitul de “power-on reset”. In momentul aplicarii Vcc, C1 este descarcat si “trage” linia RESET/ in “0”, resetind procesorul. Ulterior, condensatorul se va incarca prin R1 si linia va sta in “1”(procesul este asimptotic deci de durata infinita, dar conventional sfirsitul incarcarii se considera dupa un timp aproximativ egal cu 5R1C1). Acest reset este necesar pentru a asigura pornirea in bune conditii a procesorului; in lipsa lui, tensiunile tranzitorii care apar in momentul alimentarii pot duce la ajungerea procesorului intr-o stare incerta. Practic, linia RESET/ e tinuta in “0” un timp semnificativ mai lung decit are nevoie sursa de alimentare sa intre in regim stationar.

Condensatoarele C5 si C6 se afla cat mai aproape de procesor, intre pinii 10-11, respectiv 30-31. Ele sint condensatoare de decuplare si sint specifice alimentarii oricarui circuit digital - asigura o “rezerva” de energie in momentul comutarii, si astfel previne aparitia zgomotului de comutare pe liniile de alimentare. Condensatorul electrolitic C4 nu este obligatoriu, dar este recomandat pentru reducerea riscului ca procesorul sa se reseteze din cauza unor glitch-uri pe alimentare.

Cristalul de cuart, impreuna cu condensatoarele C2, C3 si cu amplificatorul intern de la bornele XTAL1,2 formeaza un oscilator cu cuart. Aceste componente trebuie sa se afle, de asemenea, cit mai aproape de pinii respectivi ai procesorului.

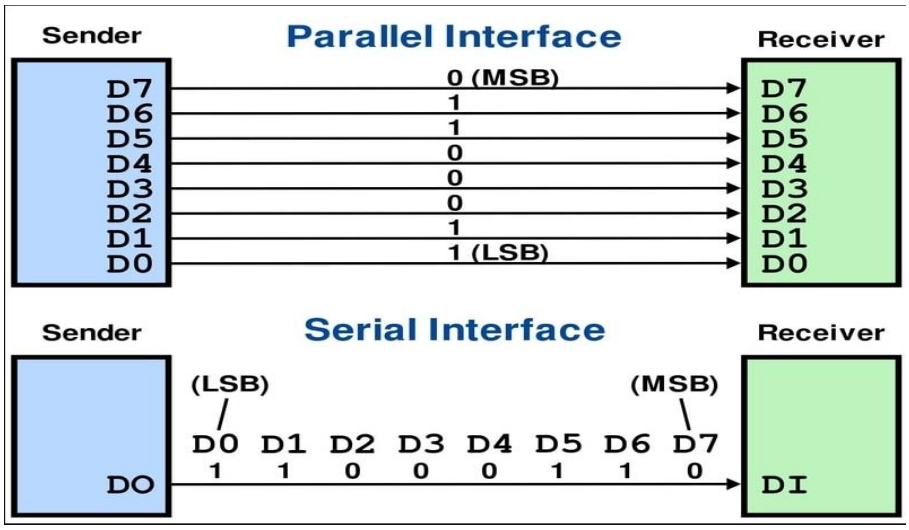
Pentru alimentare se foloseste stabilizatorul U3 cu 3 terminale , 2 condensatoare de 100nF necesare pentru stabilitatea functionarii acestuia, si dioda D1 care protejeaza la alimentarea inversa.

Jumperul J1 va fi pe pozitia 1-2 pentru folosirea lui 7805, si pe pozitia 2-3 pentru scoaterea din circuit a lui 7805 (de exemplu daca aplicam Vcc extern de la USB, sau daca dorim alimentare de la 3.3V).

Dioda LED este pozitionata astfel incat se aprinde cand PD6 este pe “1” logic; rezistenta R2 asigura limitarea curentului prin LED la cca. (5-1.6)/330= 10mA. De notat ca in multe cazuri cind se conecteaza un LED la iesirea unui circuit digital, fara tranzistor de comanda, el se monteaza invers (intre pinul circuitului si Vcc, evident cu anodul la Vcc), intrucat majoritatea circuitelor suporta un curent mai mare cand pinul este in “0” logic decit atunci cand e in “1”. Dezavantajul e ca LED-ul se aprinde pe “0”. In cazul nostru insa, procesorul AVR suporta curenti egali pe “0” si pe “1” logic.

Butonul SW1 leaga PD5 la masa (0 logic) in momentul apasarii. Intrucit, atunci cind nu este apasat, starea pinului PD5 nu este definita, va trebui activata o rezistenta de pull-up intern prin software.

CN2 este conectorul care permite folosirea unui convertor TTL-USB pentru programare (cu bootloader) si comunicatia seriala.

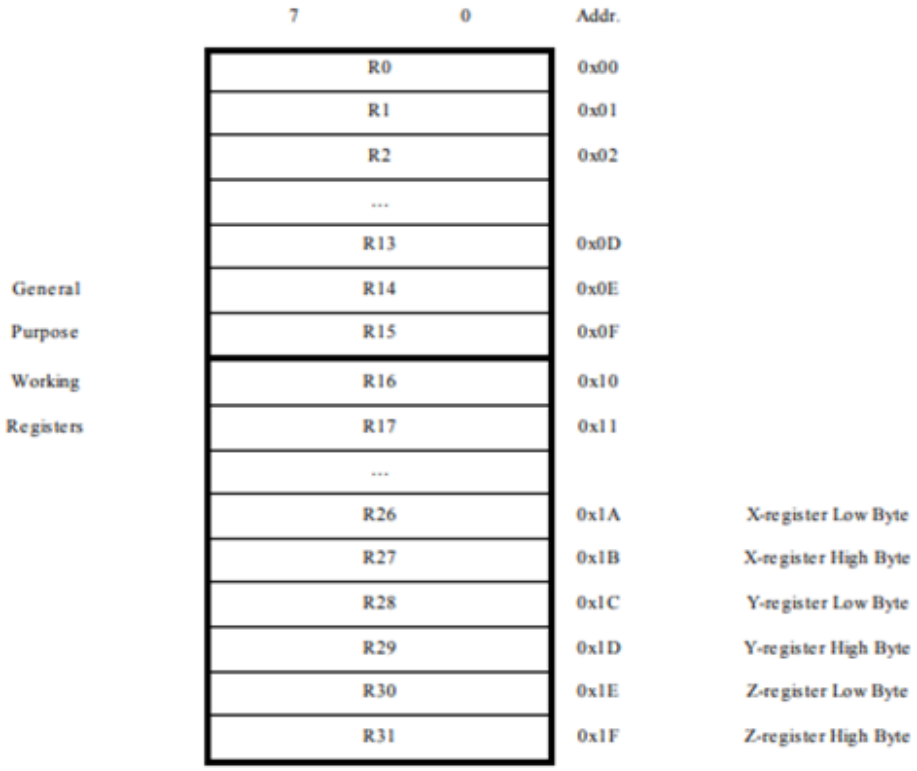


Transmiterea seriala

ATmega164 include 2 seriale UART de nivel TTL.

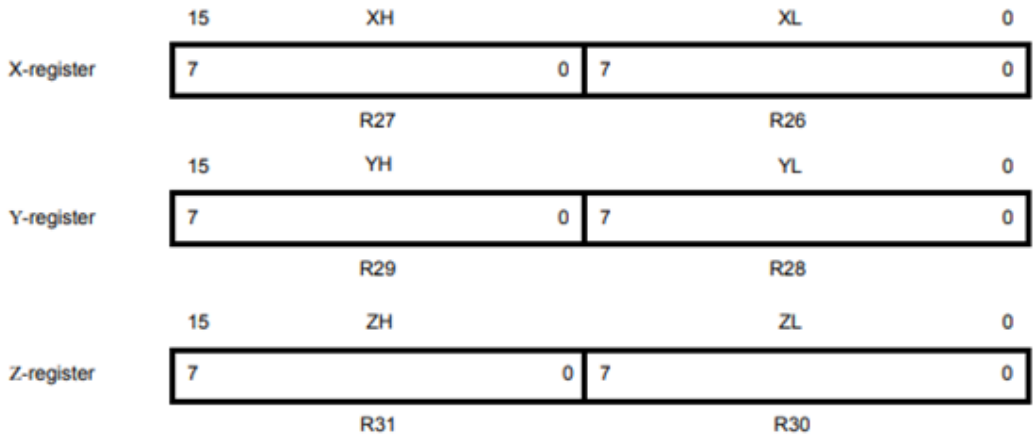
Registre

Figure 8-2. AVR CPU General Purpose Working Registers

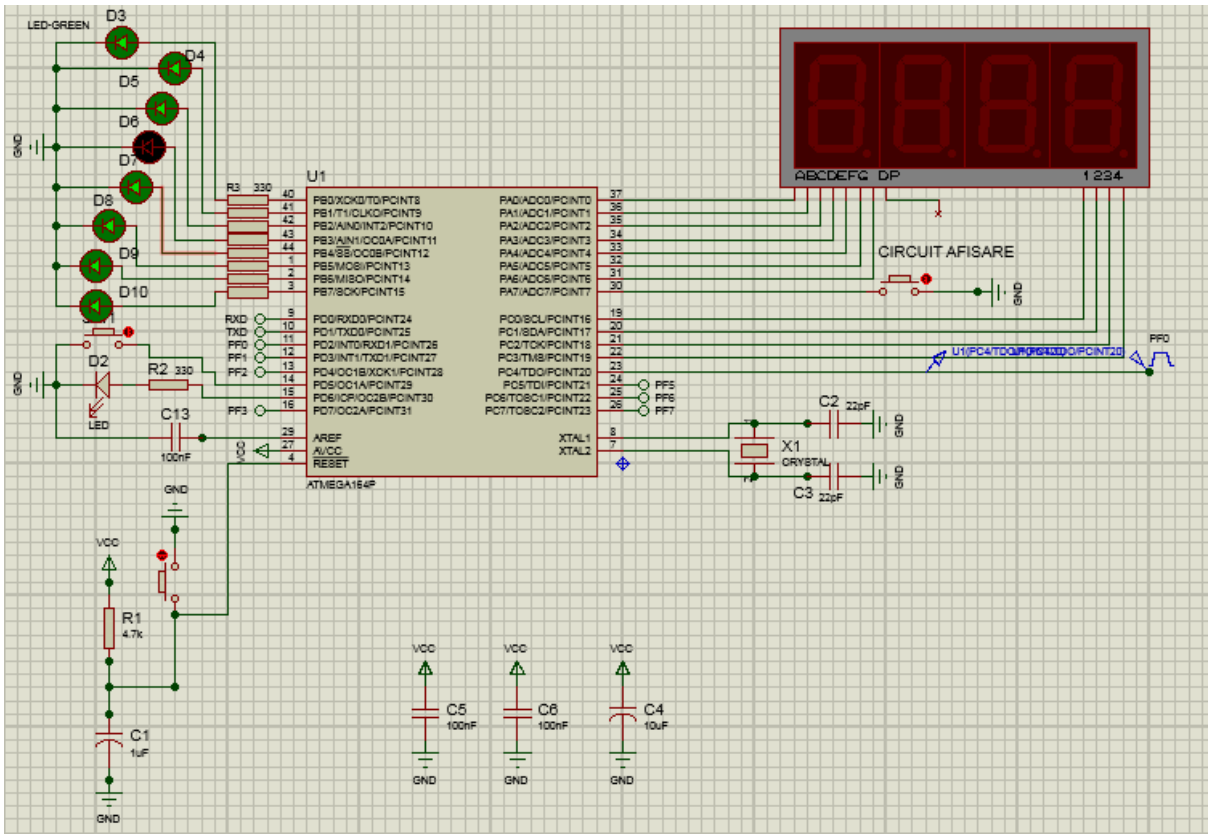


.

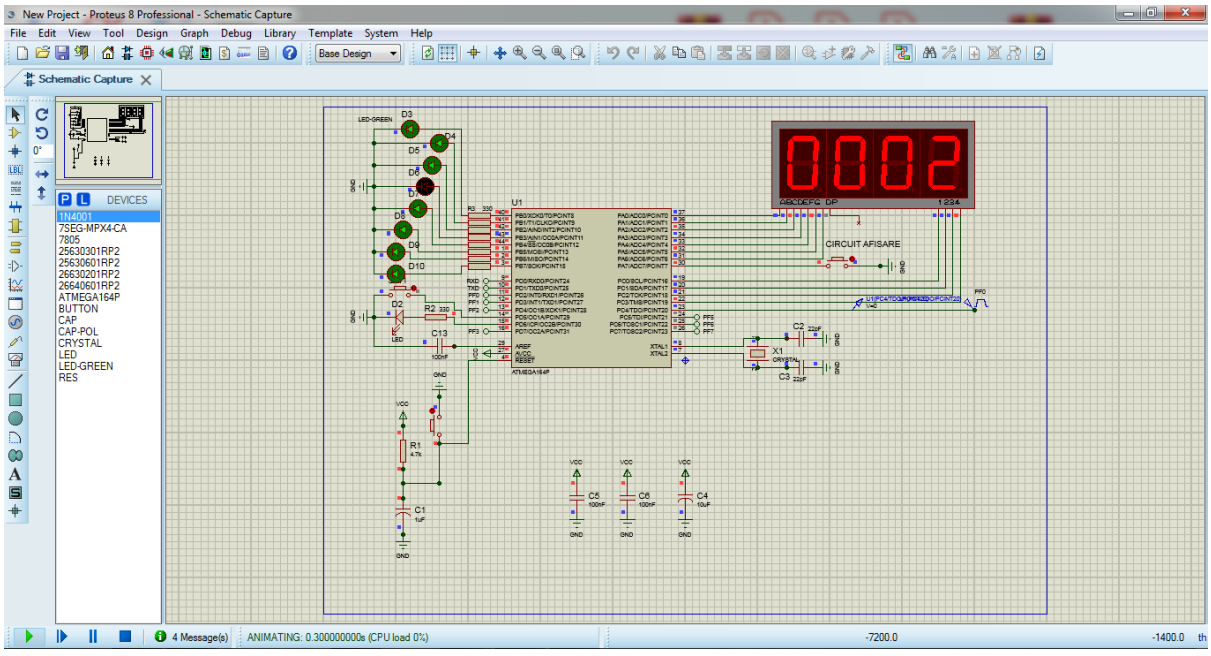
Figure 8-3. The X-, Y-, and Z-registers

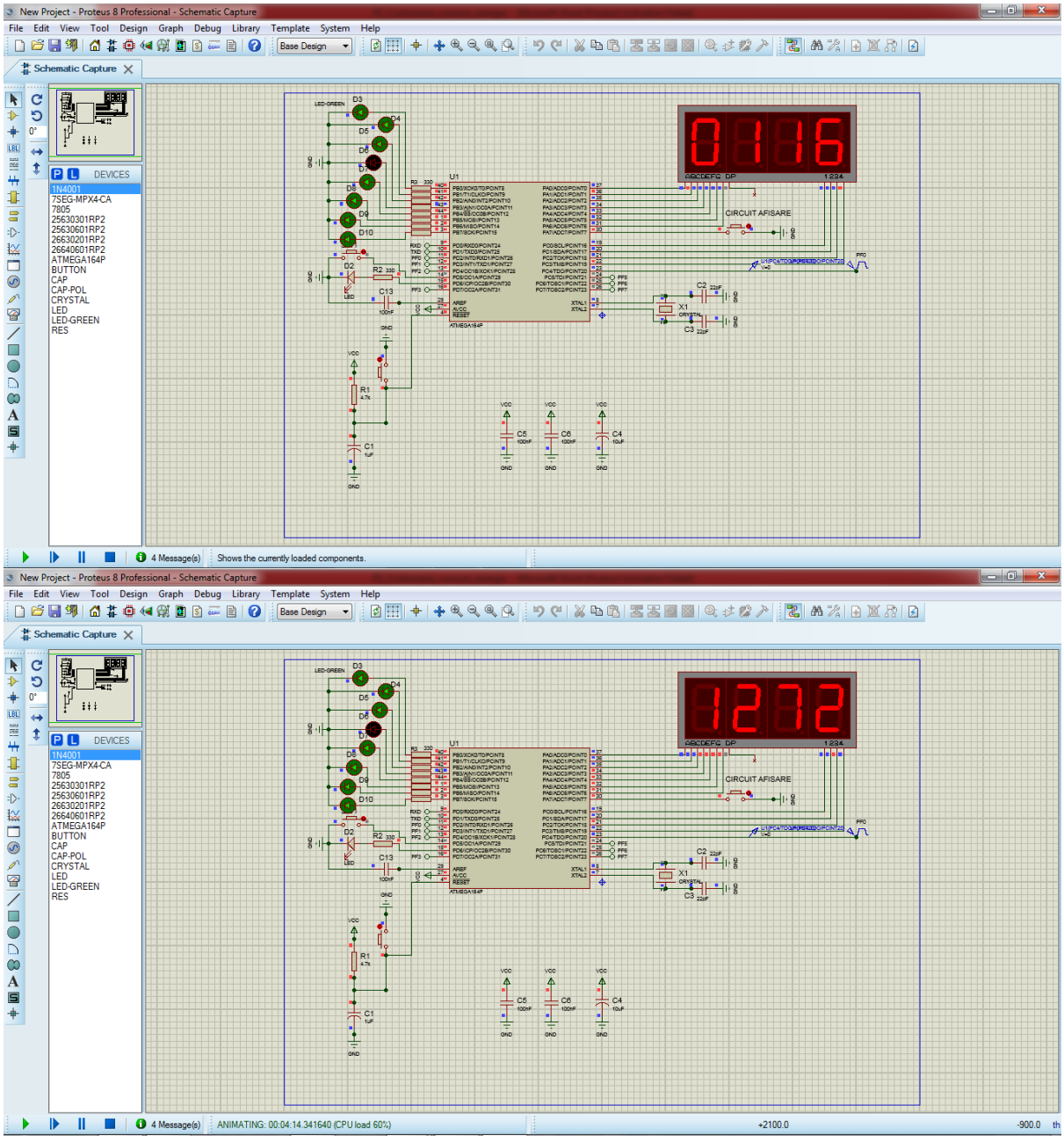


3. Schema de principiu in Proteus pentru ATmega164p



4. Rezultate ale simularii si coduri




New Project - Proteus 8 Professional - Schematic Capture
File Edit View Tool Design Graph Debug Library Template System Help
Base Design
Schematic Capture
DEVICES
1N4001
7SEG-MPX4-CA
7805
25630301RP2
25630601RP2
26630201RP2
26640601RP2
ATMEGA164P
BUTTON
CAP
CAP-POL
CRYSTAL
LED
LED-GREEN
RES
0116
CIRCUIT AFISARE
4 Message(s)
Shows the currently loaded components.
New Project - Proteus 8 Professional - Schematic Capture
File Edit View Tool Design Graph Debug Library Template System Help
Base Design
Schematic Capture
DEVICES
1N4001
7SEG-MPX4-CA
7805
25630301RP2
25630601RP2
26630201RP2
26640601RP2
ATMEGA164P
BUTTON
CAP
CAP-POL
CRYSTAL
LED
LED-GREEN
RES
1272
CIRCUIT AFISARE
4 Message(s)
ANIMATING: 00:04:14.341640 (CPU load 60%)
+2100.0
-900.0

Cod contorizare pulsuri aplicate artificial pe PF0 printr-un generator de pulsuri

```
#include <mega164.h>
#include <delay.h>

int intrare;
int iesire;
int S=0;
long contorp,contor=0,CNTQ=0;
long cnt1,cnt2,cnt3,cnt4;
int verificare = 0;

char out_clc;
char TAB[10]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};

void clc(int a){
  out_clc=TAB[a];
}

#define T 0x11;

// Timer 0 output compare A interrupt service routine
interrupt [TIM0_COMPA] void timer0_compa_isr(void)
{
    // Place your code here
    TCNT0 = 0X4E;
    intrare = PINC.4;
    switch (S)
    {
    case 0:
    if(intrare == 1 && verificare == 0)
    {
    S=1;
    CNTQ = CNTQ+1;
    }
    else S=2;

    case 1:
    cnt1 = CNTQ/1000;
    PORTC = 0X00;
    PORTA = 0XFF;
    clc(cnt1);
    PORTA = out_clc;
    PORTC = 0x01;

    delay_ms(2);

    cnt2 = (CNTQ - cnt1*1000)/100;
    PORTC = 0X00;
```

```
   PORTA = 0XFF;
   clc(cnt2);
   PORTA = out_clc;
   PORTC = 0x02;

   delay_ms(2);

   cnt3 = (CNTQ - cnt1*1000-cnt2*100)/10 ;
   PORTC = 0X00;
   PORTA = 0XFF;
   clc(cnt3);
   PORTA = out_clc;
   PORTC = 0x04;

   delay_ms(2);

   cnt4 = (CNTQ - cnt1*1000-cnt2*100-cnt3*10);
   PORTC = 0X00;
   PORTA = 0XFF;
   clc(cnt4);
   PORTA = out_clc;
   PORTC = 0x08;

   delay_ms(2);
   S=0;
   verificare = verificare + 1;


   case 2:
   if (intrare==0 && verificare ==1)
   {
   verificare=0;
   S=0;
   }
   else S=2;

}
}
```

COD IMPLEMENTARE ORGANIGRAMA:

```
// I/O Registers definitions
#include <mega164.h>

int S=0;  // Stare proces secvential
long contorp,contor=0; // contor va determina pentru care din intervalele tarifare realizam
afisajul
```

```
int CNTQ[4]={0,0,0,0}; // contorp stocheaza numarul de intreruperi ce corespunde duratei unui puls de la ADSP
long cntpuls = 0, P=1000 ; // cntpuls numara impulsurile pana la P, pentru care va incrementa CNTQ
int cnt1,cnt2,cnt3,cnt4; // contoare pentru afisare pe fiecare dintre cele 4 afisoare
int verificare=0;
char zi_ora = 0x20; // ziua de inceput este ziua 1 (sambata) pt care ziua_ora va fi 00100000


char out_clc;
char TAB[10]={0x40,0x79,0x24,0x30,0x19,0x12,0x02,0x78,0x00,0x10}; // Iesirile CLC ului corespunde numarului pe care dorim sa-l afisam cu ledurile afisorului active in 0
                                                    // Spre exemplu pentru 0 vom avea 1 doar pe led-ul G

void clc(int a){
  out_clc=TAB[a];
}

#define T 0x11;
char Q; //stare CLS

char outcls;
char *TABA[4];

char A0[]={0x60,1,T,0},  // stare weeknd si luni
    A1[]={0x66,2,0x86,2,0xA6,2,0xC6,2,0xE6,2,T,1},  //stare saptamana intre orele 00 si 06
    A2[]={0x74,3,0x94,3,0xB4,3,0xD4,3,0xF4,3,T,2},  //stare saptamana intre orele 06 si 20
    A3[]={0x80,1,0xA0,1,0xC0,1,0xE0,1,0x10,0,T,3},  //stare saptamana intre orele 20 si 00 (dispare ziua de luni)

    Tout={0x00,0x01,0x02,0x03}; // corespund numarului intervalului tarifar


TABA[0]=A0;
TABA[1]=A1;
TABA[2]=A2;
TABA[3]=A3;
S=0;
Q=0;

void cls(void)
{
   char in;
   char *adr;
   char ready;
   adr=TABA[Q];
   char i=0;
   char ready=0;
   in= zi_ora;
```

```
   while (!ready)
   {
      if ((in) ==(adr+i)) {Q=(adr+i+1); ready=1;} //daca intrarea este egala cu prima valooare
din tabela de valori relevante
      else if (*(adr+i)==T) ready=1;            //Q - urmatoare starea, va lua valoarea urmatoare
din tabela
      else i=i+2;                               // in caz contrar se va verifica urm vlaoare din tabela si se
repeta pana cand ajungem la T
}
   outcls=Tout[Q];
}

interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{

TCNT0=0x3C;

switch(S){
   case 0:
   if(PIND.2==1){//Impuls venit pe PF0
   contorp=0;
   S=1;     // se determina intervalul tarifar;
   }
   if(PINA.7==1){ //Se apasa butonul CA
   S=4;    // afisare pe afisor
   }
   if(PIND.7==1){ //Impuls venit pe PF3
   S=9;   // de aici in jos afisare led-uri 4 5 6 7
   }
   if(PIND.4==1){ //Impuls venit pe PF2
   S=10;
   }
   if(PIND.3==1){ //Impuls venit pe PF1
   S=11;
   }
   if(PINC.4==1){ //Impuls venit pe PF4
   S=12;
   }
   if(PINC.5==1){ //Impuls venit pe PF5
   S=13;
   }
   break;

   case 1:       // masoara impulsurile
   contorp=contorp+1;
   if((contorp == 4){ // A trecut un impuls de 80 ms (se considera interrupt ul de 20ms)
      contorp = 0 ;
      S=2;
   }
   else S=1;
```

```
   break;

   case 2:
   cntpuls = cntpuls + 1; // contorizam cate pulsuri au fost primite pe PF0
   if (cntpuls == P)    // cand ajungem la P pulsuri inseamna ca a fost consumat un kwh si
vom contoriza in CNTQ
   {
   cntpuls = 0;
   S=3;
   }
   else S=0;
   break;

   case 3:         //ne calc intervalul tarifar si ne contorizeaza pretul in CNTQ
   cls();
   CNTQ[outcls]=CNTQ[outcls]+1; // VA CONTORIZA IN VARIABILA
CORESPUNZATOARE intervalului tarifar determinat de CLS
   S=0;
   break;

   case 4:
   if(contor==0) S=5;       // pentru afisare pe 7 digiti
   if(contor==1) S=6;      // fiecare contor reprezinta un interval
   if (contor==2) S=7;    // contor=0 S-D contor=1 L-V primul interval si asa mai departe
   if (contor==3) S=8;

   break;

   case 5:            // afisare S-D
   cnt1=CNTQ[0]/1000;    //AFISOR MII
   PORTC.3 = 0;
   clc(cnt1);
   PORTA = out_clc;
   PORTC.0 = 1;

   delay_ms(2);

   cnt2=(CNTQ[0]-cnt1*1000)/100; //AFISOR SUTE
   PORTC.0 = 0;
   clc(cnt2);
   PORTA = out_clc;
   PORTC.1 = 1;

   delay_ms(2);

   cnt3=(CNTQ[0]-cnt2*100-cnt1*1000)/10; //AFISOR ZECI
   PORTC.1 = 0;
   clc(cnt3);
   PORTA = out_clc;
   PORTC.2 = 1;
```

```
delay_ms(2);

cnt4=CNTQ[0]-cnt3*10-cnt2*100-cnt1*1000; //AFISORI UNITATI
PORTC.2 = 0;
clc(cnt4);
PORTA = out_clc;
PORTC.3 = 1;
contor=contor+1;
S=0;
break;

case 6:              // afisare L-V 00:H1
cnt1=CNTQ[1]/1000;    //AFISOR MII
PORTC.3 = 0;
clc(cnt1);
PORTA = out_clc;
PORTC.0 = 1;

delay_ms(2);

cnt2=(CNTQ[1]-cnt1*1000)/100; //AFISOR SUTE
PORTC.0 = 0;
clc(cnt2);
PORTA = out_clc;
PORTC.1 = 1;

delay_ms(2);

cnt3=(CNTQ[1]-cnt2*100-cnt1*1000)/10; //AFISOR ZECI
PORTC.1 = 0;
clc(cnt3);
PORTA = out_clc;
PORTC.2 = 1;

delay_ms(2);

cnt4=CNTQ[1]-cnt3*10-cnt2*100-cnt1*1000; //AFISORI UNITATI
PORTC.2 = 0;
clc(cnt4);
PORTA = out_clc;
PORTC.3 = 1;
contor=contor+1;
S=0;
break;

case 7:              // afisare L-V H1:H2
cnt1=CNTQ[2]/1000;    //AFISOR MII
PORTC.3 = 0;
clc(cnt1);
```

```
PORTA = out_clc;
PORTC.0 = 1;

delay_ms(2);

cnt2=(CNTQ[2]-cnt1*1000)/100; //AFISOR SUTE
PORTC.0 = 0;
clc(cnt2);
PORTA = out_clc;
PORTC.1 = 1;

delay_ms(2);

cnt3=(CNTQ[2]-cnt2*100-cnt1*1000)/10; //AFISOR ZECI
PORTC.1 = 0;
clc(cnt3);
PORTA = out_clc;
PORTC.2 = 1;

delay_ms(2);

cnt4=CNTQ[2]-cnt3*10-cnt2*100-cnt1*1000; //AFISORI UNITATI
PORTC.2 = 0;
clc(cnt4);
PORTA = out_clc;
PORTC.3 = 1;
contor=contor+1;
S=0;
break;

case 8:             // afisare L-V H2:00
cnt1=CNTQ[3]/1000;    //AFISOR MII
PORTC.3 = 0;
clc(cnt1);
PORTA = out_clc;
PORTC.0 = 1;

delay_ms(2);

cnt2=(CNTQ[3]-cnt1*1000)/100; //AFISOR SUTE
PORTC.0 = 0;
clc(cnt2);
PORTA = out_clc;
PORTC.1 = 1;

delay_ms(2);

cnt3=(CNTQ[3]-cnt2*100-cnt1*1000)/10; //AFISOR ZECI
PORTC.1 = 0;
clc(cnt3);
```

```
 PORTA = out_clc;
 PORTC.2 = 1;

 delay_ms(2);

 cnt4=CNTQ[3]-cnt3*10-cnt2*100-cnt1*1000; //AFISORI UNITATI
 PORTC.2 = 0;
 clc(cnt4);
 PORTA = out_clc;
 PORTC.3 = 1;
 contor=0;
 S=0;
 break;

 // PENTRU timer afisaje LED-uri
     case 9:
contorp = contorp  + 1;
 if (contorp = 4)    // Toate pulsurile venite de la adsp au aceeasi durata
 {
 contorp = 0;
 S=14;
 }
 else S=9;
 break;

    case 10:
 contorp = contorp + 1;
 if (contorp == 4)
 {
 contorp = 0;
 S=15;
 }
 else S=10;
 break;

    case 11:
 contorp = contorp + 1;
 if (contorp == 4)
 {
 contorp = 0;
 S=16;
 }
 else S=11;
 break;

    case 12:
 contorp = contorp + 1;
 if (contorp == 4)
 {
 contorp = 0;
```

```
S=17;
}
else S=12;
break;

  case 13:
contorp = contorp + 1;
if (cntpuls == P)
{
contorp = 0;
S=18;
}
else S=13;
break;

case 14:            //  consum mare - SEMNALAT DE IMPULS PF3

PORTB.4=1; //vom stinge led-urile 4 si 5
PORTB.5=1;
delay_ms(2);
PORTB.4=0; // aprindem cele 2 led-uri
PORTB.5=0;
S=0;
break;

case 15:            // consum mediu - SEMNALAT DE IMPULS PF2

PORTB.4=1; //vom stinge cele 2 led-uri
PORTB.5=1;
delay_ms(2);
PORTB.5=0; // aprindem doar led-ul 5 ceea ce semnaleaza consum mediu
S=0;
break;

case 16:            // consum mic - SEMNALAT DE IMPULS PF1
PORTB.4=1; // Vom stinge cele 2 led-uri
PORTB.5=1;
delay_ms(2);
PORTB.4=0; // vom aprinde doar led-ul 4 ceea ce semnaleaza consum mic
S=0;
break;

case 17:            // mod de lucru cu transe orare
PORTB.6=1
PORTB.7=1;
delay_ms(2);
PORTB.6=0;
break;

case 18:            // mod de lucru fara transe orare
```

```
   PORTB.6=1
   PORTB.7=1;
   delay_ms(2);
   PORTB.7=0;
   break;
}


}

int cntt=0;
char S=0x00,M=0x00,H=0x00,Z=0x01;
int S1=0;
void masurare(void){     // PS masurarea timpului

while(1){
   switch{
   case 0:
   cntt=cntt+1;
   if(cntt%50==0) cntt=0; S=S+0x01; S1=1;  // Dupa 50 de intreruperi stim ca a trecut o
secunda deci vom incrementa variabila S
   else S1=0;  // daca a trecut o secunda vom trece in starea 1, daca nu, ramanem in starea 0
   break;

   case 1:
   if(S==0x3C) S=0; M=M+0x01; S1=2; // S=60;  Daca trec 60 de secunde trecem in starea 2,
daca nu trecem in starea 0;
   else S1=0;
   break;

   case 2:
   if(M==0x3C) // M = 60 // Daca au trecut 60 de minute vom face M=0 si vom incrementa
ora, daca nu, trecem in starea 0
   {
   M=0; H=H+0x01;
   zi_ora = zi_ora + H ; // vom stoca in variabila zi_ora ORA H a zilei Z
   }
   else S1=0;
   break;

   case 3:
   if(H==0x18) //H=24  Similar daca au trecut 24 de ore vom incrementa ziua si vom face
H=0, trecand apoi in starea 4, daca nu, ne intoarcem in starea 0
   {
   H=0;
   Z = Z+1;   // Incrementeaza ziua
   Z = Z << 5; // Shifteaza valoarea zilei pentru ca bitii corespunzatori zilei sa fie bitii 5 6 7
   zi_ora = Z; // Vom stoca in zi_ora 0x ziua(3 biti) / ora(5 biti , care va fi 0)
   S1 = 4;
   }
```

```
   else S1=0;
   break;

   case 4:
   if(Z==0x08)    //Dupa ce a trecut o saptamana vom stoca in zi_ora = ziua 1 /ora 0
   { Z=1;         // Se verifica daca Z este 8, daca a devenit 8 atunci vom treze Z in 1 si ne
intoarcem in starea 0
     S1=0;
     zi_ora = 0x20; // vom trece variabila zi_ora in 0x20
   }
   else S1=0;
   break;
   }
}
}
```

Blue-print pentru codul in VisualDSP++:

```
.var Prag=3600; // pentru un 1kWh se genereaza 1000 de pulsuri
.var Pulse;
.var cnt;
.var DP=4; // genereaza impulsuri cu o durata a frontului pozitiv de 4 intreruperi
.var T=20;
.var Energie=0;
.var ver=0;
.var n;

pulsuri:

Pulse=1; // impuls pe care il vom pune pe portul de iesire
cnt=cnt+1;
if cnt!=DP jump pulsuri;

pulsuri1:
Pulse=0; // impuls pe care il vom pune pe portul de iesire
cnt=cnt+1;
if cnt!=T jump pulsuri1;
cnt=0;
jump continuare;


input_samples:
      ena sec_reg;              /* use shadow register bank */

      sr1 = dm (rx_buf + 2); /* intrare tensiune */
                  sr0 = dm (rx_buf + 1); /* intrare curent */

generare:
                  Energie=sr1*sr0*IRQ2; // calculare energie=U*I*deltaT
```

```
                ver=ver+Energie;

                if ver<Prag jump generare; // verificam daca input-ul depaseste sau nu pragul
                n=DIVS ver,Prag; // in caz ca energia depaseste pragul vrem sa aflam n (n- nr
de pulsuri generate pentru energia stocata in ver)
                ver=ver-n*Prag; // ver=0;

verificare:
                if n==0 jump generare; // daca numarul de pulsuri este 0 ne intoarcem la citirea
lui U si I
                jump pulsuri; // daca n>0 atunci vom genera pulsurile
continuare:
                n=n-1; // decrementam n
                jump verificare; // ne intoarcem in "verificare" unde verificam daca mai
trebuie transmise pulsuri


nofilt: /sr=ashift sr1 by -1 (hi);/   /* save the audience's ears from damage */
        mr1=sr1;
        mr0=sr0;

        si=IO(PORT_IN);
        IO(PORT_OUT)=si;
output:
        dm (tx_buf + 1) = mr1;      /* filtered output to SPORT (to spkr) */
        dm (tx_buf + 2) = mr1;      /* filtered output to SPORT (to spkr) */
        rti;
```