**UNIVERSITATEA POLITEHNICĂ DIN BUCUREȘTI**

FACULTATEA DE ELECTRONICĂ, TELECOMUNICAȚII ȘI TEHNOLOGIA INFORMAȚIEI

**PROIECT 2**

**SISTEM DE CONTORIZARE ȘI AFIȘARE A CONSUMULUI DE ENERGIE ELECTRICĂ**

**STUDENȚI:**

LEFTERACHE ALEXANDRU-GABRIEL 431A

CRISTESCU ANA-IOANA 433D

MARINESCU AURORA-CRISTINA 433D

OPREA LIVIA-DANIELA-MIHAELA 433D

**PROFESOR COORDONATOR:**

ZOICAN SORIN

**2022-2023**

**CUPRINS**

**PREZENTAREA TEMEI**

Tema constă în realizarea unui sistem care contorizează și afișează consumul de energie electrică și este compus din două subsisteme (AVR și DSP). Arhitectura acestora este reprezentată în figura 1:

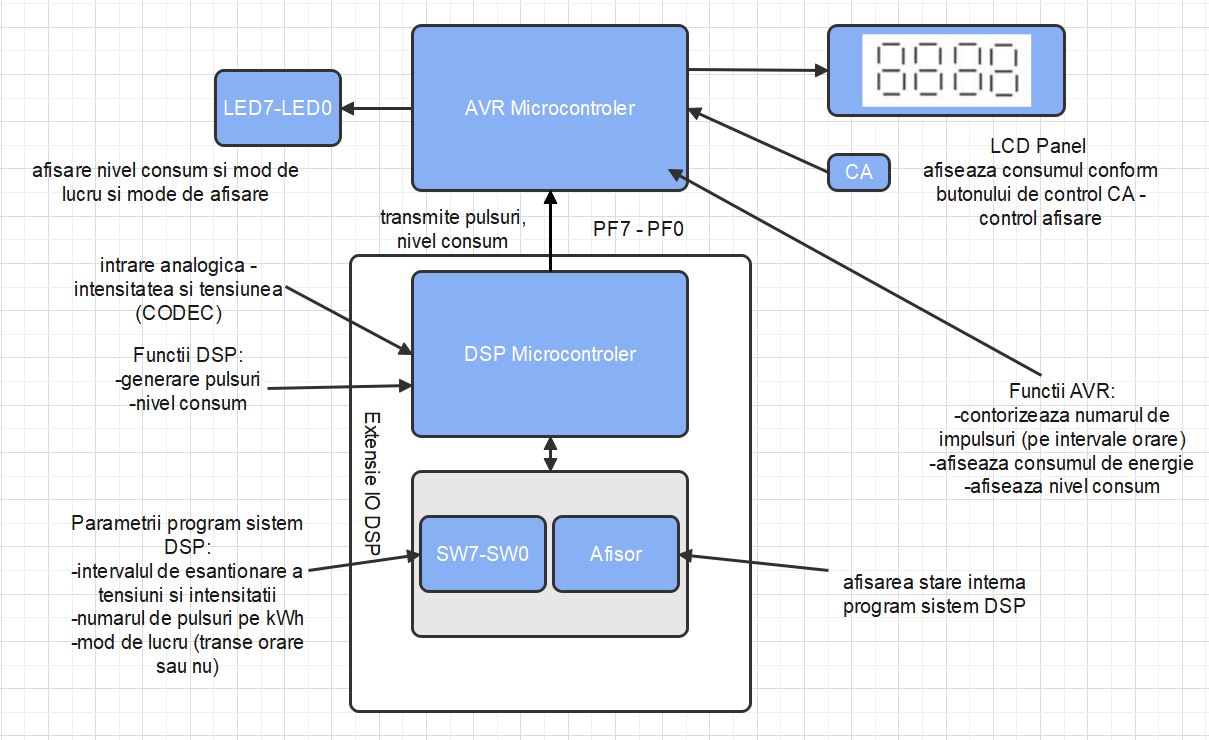


Figura 1

Subsistemul DSP are în componența sa placa de evaluare EZ-Kit LITE ADSP2181 și o interfață de intrare-ieșire, IO DSP. Acesta realizează măsurarea tensiunii și intensității, calcularea energiei consumate și generarea unui număr P de pulsuri pentru 1kWh. Pulsurile generate și valoarea intensității curentului consumat se vor transmite pe un pin al portului PF. Se consideră puterea maximă de 10 kW. Intervalul de eșantionare a tensiunii si intensității ΔT si numărul de pulsuri P per kWh, se vor stabili din SW7-SW0. În cadrul temei, s-a stabilit un mod de lucru al contorizării energiei cu/fără intervale orare.

Subsistemul AVR utilizează un microcontroler ATMega164. Acesta realizează contorizarea pulsurilor, afișarea și calcularea energiei consumate conform modului indicat de subsistemul DSP.

Implementarea la nivel hardware: subsistemul AVR (cu microcontroler ATMega164) și extensia IO DSP

Implementarea la nivel software: descrierea formală a programelor pentru subsistemele AVR și DSP, scrierea codului pentru cele 2 subsisteme ( în limbaj de programare C pentru AVR și în limbaj de asamblare ADSP2181 pentru subsitemul DSP) și testarea programelor în CVAVR și ASTUDIO, respectiv în Visual DSP++ 3.5.

În final se va verifica funcționalitatea sistemului fizic realizat.

**DESCRIEREA PRELUCRĂRILOR - GRAFURI ȘI ORGANIGRAME**

Programul principal AVR, reprezentat în organigrama din figura 2 de mai jos, conține următoarele blocuri:

**Inițializări**, unde sunt inițializate variabilele globale, tebelele pentru circuitele logice combinaționale (CLC), tabelele de semnale relevante de la circuitele logice secvențiale (CLS);

**Afișează consum**, având corespondentul DisplayConsumption în codul ce va urma, reprezintă funcția care va face afișarea consumului, ca mai apoi, să se aștepte **Întreruperi** (acest procedeu se realizează în bucla while a programului principal – main – astfel: de fiecare dată când procesorul trece prin bucla while, se afișează câte un digit, așa încât, ținând cont de frecvența mare a procesorului, ochiul uman va fi ”păcălit”, afișajul părând a fi continuu. Practic, el afișează pe rând, dar foarte repede. Dacă apare întrerupere, se trece în rutina de servire a întreruperii, la **SCI**, iar dacă nu apar intrerupere, procesul se reia.

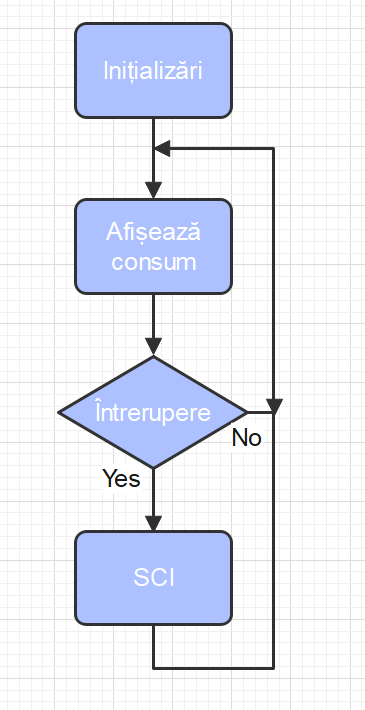


Figura 2

**CONTORIZAREA TIMPULUI**

Pentru sistemul de contorizare și afișare a consumului de energie, se vor implementa două procese secvențiale (PS) astfel: unul pentru măsurarea energiei, după verificarea duratei cu un contor de pulsuri și altul pentru măsurarea timpului cu afișare pe 4 cifre pe intervalele de "+".

În cadrul procesului secvențial asociat contorizării timpului, cu organigrama în figura 3, vom defini câte o variabilă specifică fiecărei unități de timp:

H - hour; D - day; M - minutes; S - seconds;

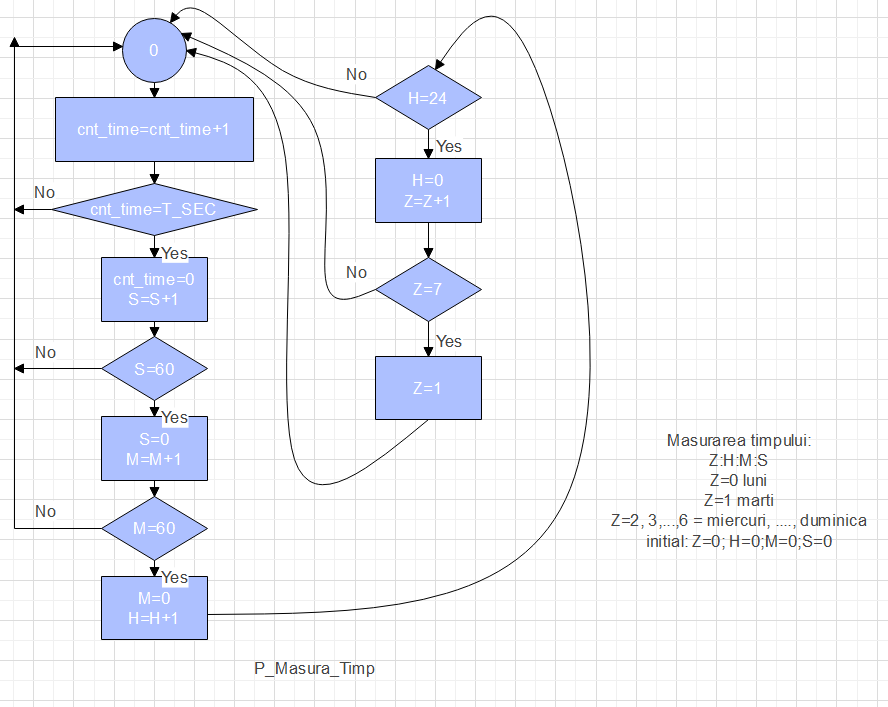


Figura 3

De asemenea, celelalte variabile din cadrul organigramei au semnificații, după cum urmează:

* cnt\_time - contorul de timp
* T\_SEC - numărul de perioade necesare pentru a acoperi o secundă

Pseudocodul asociat programului pentru contorizarea timpului va începe din starea 0, având contorul cnt\_time=0.

Valorile inițiale ale variabilelor H, D, M, S pot fi inițializate cu valoarea 0 sau pot fi inițializate cu valorile reale la începutul funcționării.

void contorizare\_timp (){

cnt\_time + = 1; // incrementare contor de timp

dacă cnt\_time ! = T\_SEC

return ;

altfel // cnt\_time = T\_SEC

{

cnt\_time = 0; //resetează contorul

S + = 1; //incrementeză contor secunde

}

dacă S=60 //au trecut 60 de secunde

{

S = 0; //resetează contor secunde

M + = 1; // incrementează contor minute

}

altfel

return; //nu au trecut 60 de secunde, deci nu este necesară //incrementarea minutelor și prin urmare nu este necesar să //continuăm

dacă M = 60 //nu au trecut 60 de minute

{

M = 0; //resetează contor minute

H + = 1; // incrementează contor ore

}

altfel

return; //nu au trecut 60 de minute, deci nu este necesară //incrementarea orelor și prin urmare nu este necesar să //continuăm

dacă H = 24 //nu au trecut 24 de ore

{

H = 0; //resetează contor ore

Z + = 1;

}

altfel

return;

dacă Z =7 // am încheiat săptămâna

Z = 0; // resetare zi

return; //sfârșit funcție de contorizare timp

}

Codul în limbaj AVR este:

COD

**DisplayConsumption()**

DisplayDigit(display, digit) afișează cifra ”digit” pe display-ul ”display”.

Intrarea în CLC este cifra de afișat, iar ieșirea reprezintă codul binar corespunzător pentru afișarea ieșirii pe display.

Q reprezintă regimul curent de tarifare.

TOTAL\_CONS reprezintă consumul total, fără interval de tarifare.

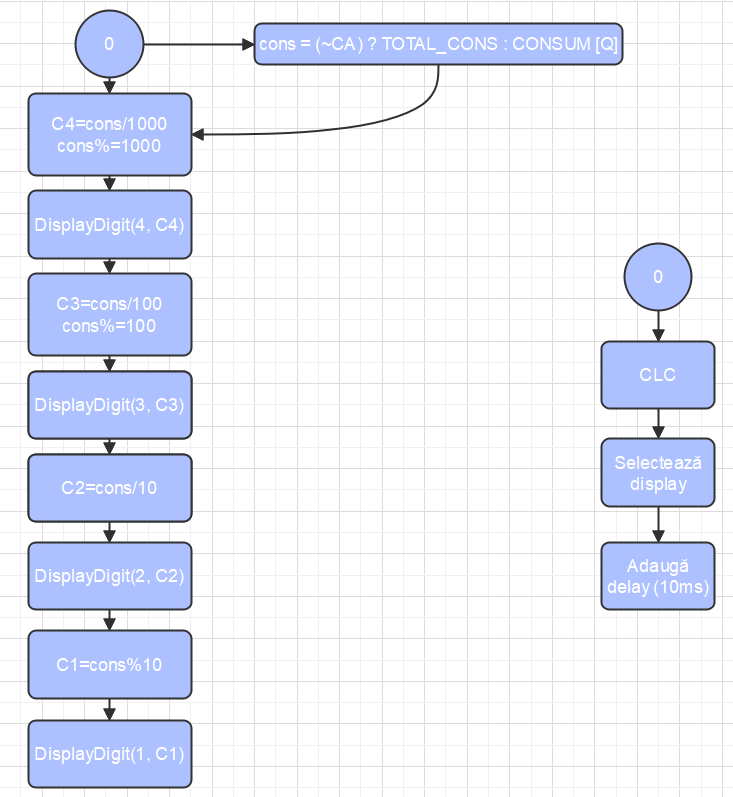


Figura 4

Pseudocod:

Cod:

Explicații:

Pentru metoda DisplayConsumption():

Presupunem port C cu pinii de la 0 la 6, conectați la cele 7 segmente (A-G) ale fiecărui afișor.

Toate segmentele de tipul A vor fi conectate la pinul 0, toate segmentele de tipul B la pinul 1 analog restul.

Pt a afișa fiecare cifră pe rând va trebui să o multiplexăm, adică mai pe scurt, la un moment de timp să afișăm o singură cifră și tot așa.

Port D cu pinii de la 0 la 3 vom face selecția pt fiecare digit, adică se face conexiunea la masa fiecăruia, ca să validăm respectivul afișor.

Q-ul este consumul.

Abordarea la fiecare iterație prin bucla while, afișăm cifrele rând pe rând.

Varieabilei char cons => cons de la consum fie îi atribuim valoarea totală a consumului, fără să ținem cont de intervalele orare, fie consumul corespunzător intervalului pe care suntem noi.

Calculăm și afișăm C4, cifra minimă. Cum facem: Împărțim la 100 consumul, păstrăm câtul.

Metoda DisplayDigit(), primul param pe care ni-l dă ca argument este display-ul pe care am dori noi să îl afișăm.

C este portul pe care îl conectăm la afișoarele noastre și o să îi dăm valoarea DIGITS(digit) unde DIGITS este tabela noastră clc (afișează pe afișor luând reprezentarea combinației binare care ne reprezintă cifra pe acesta).

După ce am selectat combinația binară care ne dă cifra pe care am dorit să o afișăm, vom selecta prin multiplexare afișorul pe care dorim noi să facem afișarea.

Vom lua o variabilă output pe care o vom inițializa la 0xff, output=ieșirea pe care o dăm pe port D, ieșirea pe care facem multiplexarea. Vom trece în 0 pinul corespunzător afișorului pe care vrem să facem afișarea. Case 4=>afișare cifră mii, case 3=> afișare cifră sute analog.

Adăugam întârziere de 10 us.

**UpdateConsumption()**

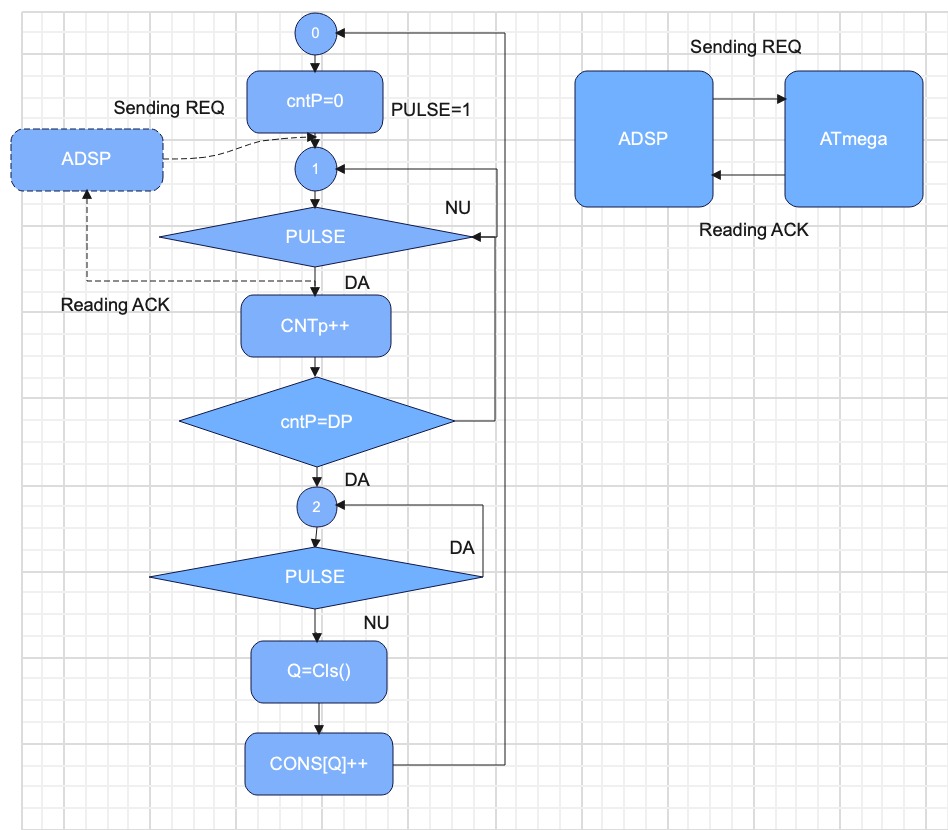


Figura 5

Pseudocod:

Cod:

Explicații:

Pentru metoda UpdateConsumption()

Inițializează o variabilă de tip char cu 0, apoi identificăm primul bit al lui pin A, care este port de intrare). Stare inițială cls este 0, are ca scop de a verificare dacă apare un request din partea adsp-ului, pentru a ne trimite ceva. Primim pe linia pentru sending request aflată pe bitul 6 a portului D request de la adsp, iar pe portul 7 citim aknowledgement-ul de la ATmega164A.

PORTD && 0x40-verif dacă vrea să ne trimită ceva adsp-ul, pe port 6.

PORTD |= 0x80 -trimitem acknowledgement (ack) pe port 7

Trecem mai departe în starea 1: dacă pulse este on, adică dacă avem palier de 1, incrementăm numărătorul. La noi conter-ul P trebuie să fie egal cu DP (cntP==DP), DP fiind durata unui puls, DP=20ms, DP ca variabilă fiind 1. (puls întreg 100ms. intrerupere la 20 ms.)

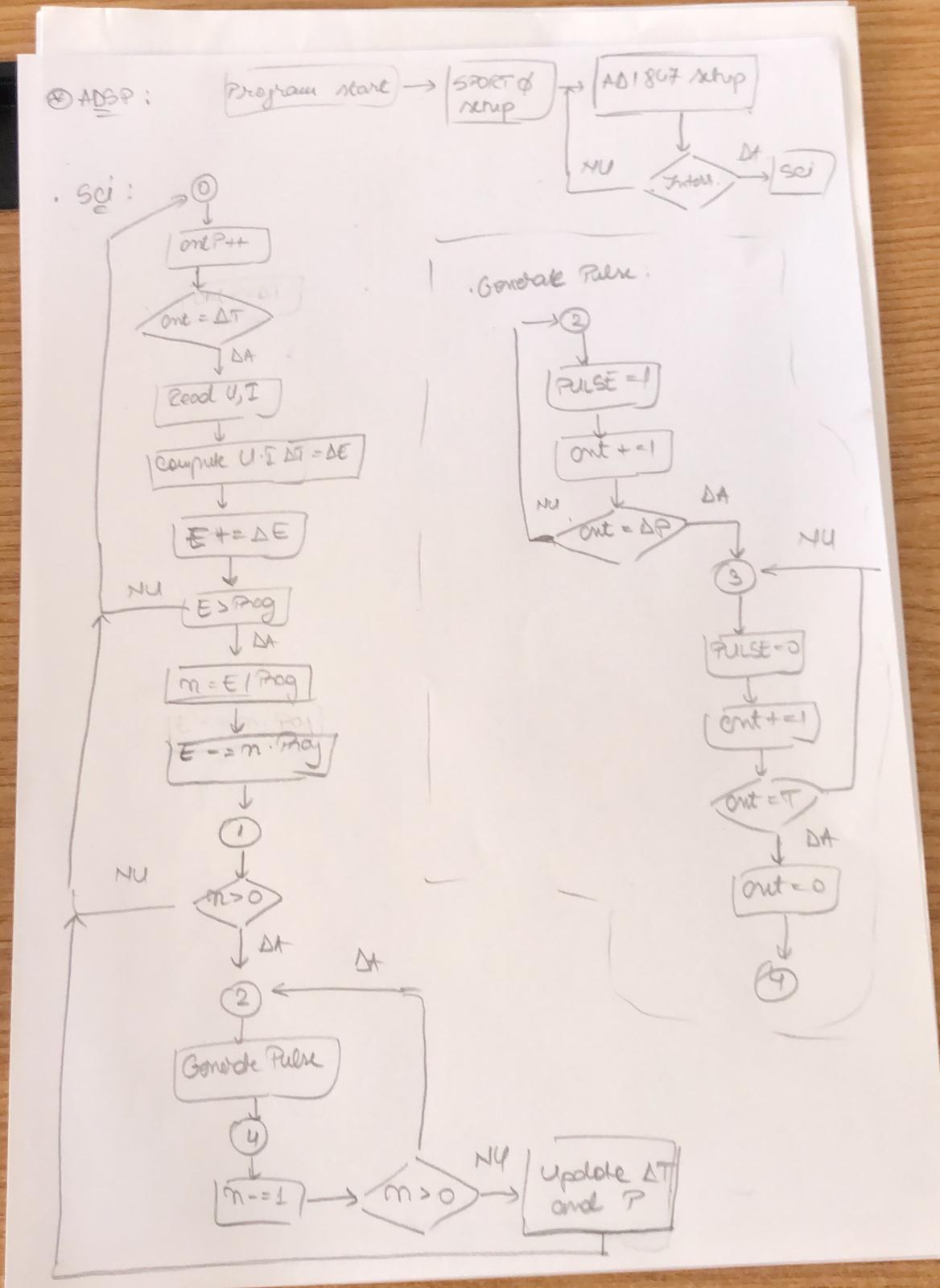
Mai apoi dăm reset la reading flag, deci îl considerăm 0, după ce am făcut citirea, pentru a nu exista conflicte, cum ar fi ca ADSP-ul să nu trimită de două ori mesaj sau să citim noi de mai multe ori mesajul.

Următoarea stare va fi fie 2, dacă cntP==DP, fie rămâne în starea 1 până ajunge acolo. Suntem pe palierul de 0, s-a terminat palierul pozitiv, actualizăm starea, pentru a vedea în ce interval orar suntem, ca să putem mai apoi să incrementăm consumul.

Trem în final la 0, ne întoarcem în 0 pentru a recepționa un nou impuls, nu ne interesează să contorizăm întreaga perioadă.

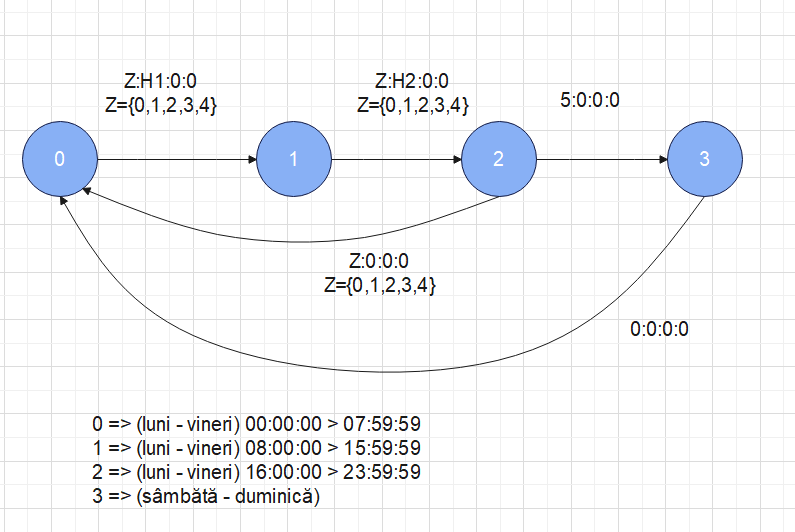
**ADSP**

**De facut in program**

****

**Figura 6**

**GRAF CLS – DETERMINARE INTERVAL DE ÎNREGISTRARE**

Figura 7

PROIECT DE TEST

s-a lucrat in intrerupere