1. Introducere
   1. Sisteme de calcul embeded

Evoluația calculatoarelor a cunoscut în ultimul deceniu un avînt puternic, facilitat pe deoparte de dezvoltările spectaculoase ce au avut loc în microelectronică iar pe de altă parte de continua scădere a prețurilor.

Dacă în anii 70 primul calculator – Mark I , ocupa un întreg bloc, fiind construit pe cîteva etaje, elementele de comutație fiind realizate cu tuburi electronice (triode și pentode), apariția tranzistorului , în anul 1947 a schimbat radical cursul istoriei.

Sistemele de calcul , au incput sa cunoască o răspîndire din ce în ce mai mare, destrămînd monopulul institutelor de calcul.

Evoluția lor poate fi clasificată prin generații astfel :

* Generatia 0 presupunea implementarea circuitelor electronicu cu ajutorul releelor
* Generatia 1 sa trecut la utilizarea tuburilor electronice , viteza de calcul a crescut la mii de operații pe secundă
* Generația a 2-a , procesorul este realizat utilizînd dispozitivele semiconductoare

Fiabilitatea sistemului creste MTBF fiind îmbunătășit la mii de ore față de tuburile electronice, crește capacitatea de stocare, apar memoriile pe ferită magnetică, de asemenea viteza de calcul atinge zeci de de mii de instrucșiuni pe secundă

* Generașia a 3-a apare odată cu a nașterea primului circuit integrat pe scară largă, odată cu această generașie apare notiunea de system de operare – primitiv , fiabilitatea sistemului crește la 100 de mii de ore, viteza de calcul depășește pragul de 1MIPS, apar mediile de stocare magnetice externe pentru program și date, caseta magnetică si banda magnetică.
* Generatia a-4-a se relevă odată cu apariția circuitelor integrate VLSI, nu se mai pune problema capacității de memorie, Intel promovează o soluție privind memoria virtuală, procesorul dezvoltat de această firmă 80386, putînd lucre ăn modul protejat, accesănd memoria situate pe unitatea externă de disc ca și memorie RAM.

Arhitecturile sistemelor de calcul, două la număr, au evoluat în parallel cu tendințele pieței.

Astfel, desi cunoscundui-se potențialul , arhitectura Hardvare a stagnat o perioadă bună de timp , lăsînd loc arhitecturii VonNeumann.

Primul processor, realizat în tehnologie MSI a fost dezvoltat de către firma Intel din Statele Unite ale Americii, acesta a fost un processor de 4 biți, destinația lui inițială fiind pentru calculatoarele de buzunar, dar la scurt timp, o firmă din S.U.A a găsit ca și aplicație , gestionarea traficului în intersecții.

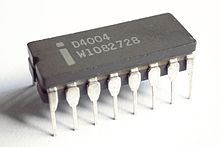


Figura. 1.1 Intel 4004

Microprocesoarele au fost și sunt intens folosite ăn sistemele electronice de calcul de ordin general, cel mai răspîndit fiind PC -Personal Computer.

In paralele cu acesta , automatica, a cunoscut și ea o dezfoltare profundă, sistemele automate, au fost implementate initial utilizănd logica cablată, circuitele realizate cu tranzistoare bipolare, ce implementau porțile logice fundamentale, circuite basculante bistabile, monostabile și astabile, regiștrii de deplasare, circuite trigger-schmit , etc …

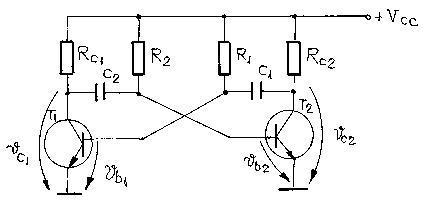


Figura. 1.2 Circuit astabil cu tranzistoare

Logica cablată , sa bucurat la început de viteza de prelucrare a semnalului prin circuitele logice, viteză care era net superioară, la aceea vreme de cea conferită de microprocesoare.

Circuitele de tip Emmiter Coupled Logic, avînd cel mai mic timp de propagare prin poarta elementară.

Dezvoltarea unei aplicații utilizănd logica cablată, în special a automatelor, presupunea un proces elaborat, cun un număr de pași bine definiți, rezultatul final fiind puternic dependent de soluția utilizată, orice ăncercare ulterioară de modificare a soluției, ducea la reproiectarea de la zero a întregului sistem.

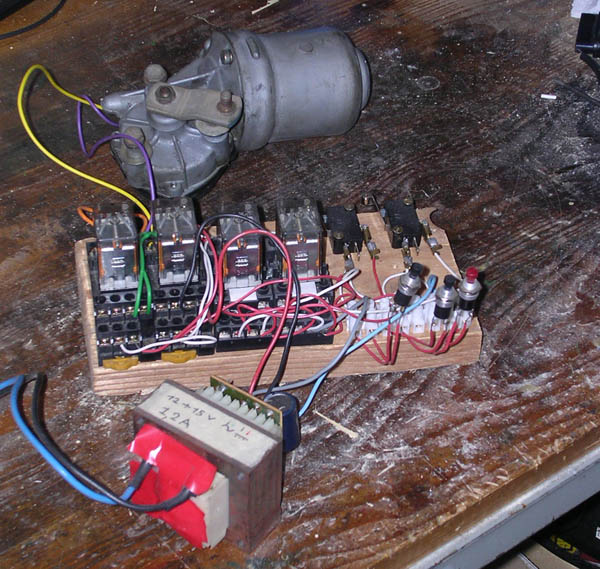


Figura.1.3 Comanda unui motor folosind logica cablată – implementare cu relee

Sistemele embeded folosesc microprocesoare, avînd puteri variate de calcul în funcșie de cerințele aplicației ce urmează a fi dezvoltate.

De regulă ele acoperă un spectru larg de aplicații începănd de la procesarea de semnal, unde se utilizează așa zisele DSP = Digital Signal Processor, sisteme de coordonare a proceselor în timp real, sateliți, navete spațiale.

Utilizarea sistemelor embeded se face la

* Aplicații medicale, ecografe, cardiografe, glucometre, tensiometre
* Aplicații casnice, mașini de spalat rufe, aspiratoare, frigider, aer condiționat
* Comunicații – telefoane mobile, ethernen
* Calculatoare – plăci video, plăci de rețea, placi de sunet ..
* Iot – Internet of Thing
* Multimedia, console de mixaj audio, mp3 playere, recordere audio-video, DVD, BlueRAY

Desigur, sistemele embeded au o structură diferită față de procesoarele clasice, acestea de regulă fiind orientate către aplicație , cunoscutele ASIC, FPGA și microcontrolerele.

In lucrarea de față, atenția a fost orientata către microcontrolere.

Piața oferă astșzi un spectru larg de microcontrolere, diversitatea producătorilor, gradul de integrare, disponibilitatea, orientarea puternică către aplicație, facilitățile oferite sunt puncte forte care fac atractive utilizarea acestora în variate proiecte.

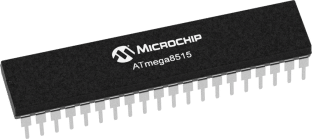


Figura.1.4 Microcontrolerul

1. Microcontrolerele în sistemele automate
   1. Generalitați

Microcontrolerele sunt circuite logice complexe ce conțin o unitate centală de prelucrare a informației cu ajutorul căreia se fac operații aritmetice și logice asupra datelor, operații ce sunt executate secvențial fiind controlate de un program.

Datele și instrucțiunile sunt memorate în memorii interne de tip RAM sau EEPROM.

In aceeiaș capsulă sunt înglobate circuite de suport, cum ar fii temporizatoare, numărătoare, porturi paralele, porturi seriale de comunicație, convertoare analog numerice, circuite de ceas etc.

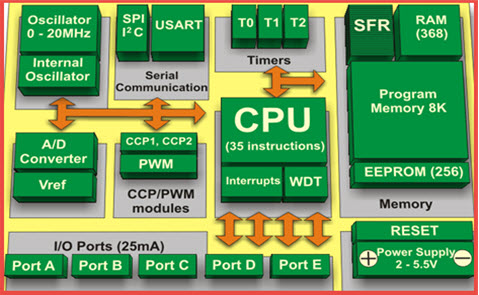


Figura. 2.1 Arhitectura microcontrolerului

Există, în principal două arhitecturi de bază, anume Harvard și Von Neumann, fiecare arhitectură are avantajele și dezavantajele sale.

Arhitectura Von Neumann își trage numele de la cel care a pus bazele teoretice ale calculatoarelor moderne , anume John Vonn Neumann, principala caracteristică a a cestei arhitecturi o reprezintă memoria unică pentru date și instrucțiuni. Această particularitate face ca lu un moment dat să potem considera datele ca și instrucțiuni, existănd posibilitatea apariției unor bucle în program.

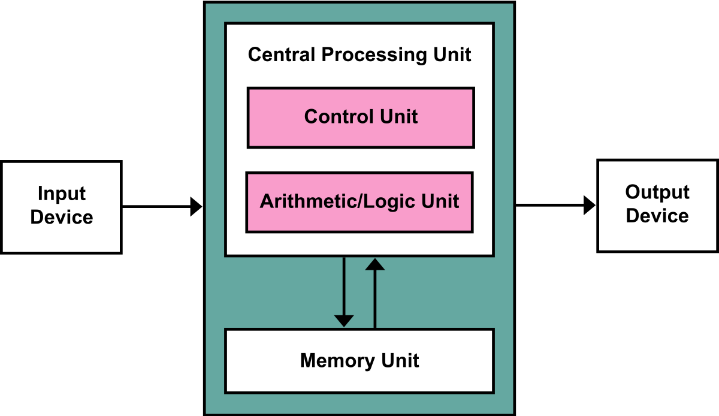


Figura. 2.2 Arhitectura Von Neumann

Evoluția calculatoarelor, respectiv a unităților centrale de prelucrare sa făcut mult timp pe baza acestei arhitecturi, factorul determinist fiind prețul prohibitiv al harwareului.

O altă particularitate a arhitecturii Von Neumann este aceea că procesorul poate executa la un moment dat doar o singură instrucțiune

Cea de a doua arhitectură, cea care stă la baza tuturor microcontrolerelor este arhitectura Harvard, dezvoltată la universitatea din Harvard, separă memoria program de cea de date, în acest fel se poate realiza un paralelism în executarea instrucțiunilor.

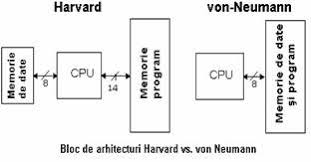


Figura. 2.3 Arhitectura Harvard, stînga și von Neumann , dreapta

**Arhitectura Harvard este o arhitectură a calculatoarelor caracterizată prin stocarea separată a instrucțiunilor și datelor. Numele acestei arhitecturi provine de la sistemul de calcul**[**Harvard Mark I**](https://ro.wikipedia.org/w/index.php?title=Harvard_Mark_I&action=edit&redlink=1)**, ce stoca instrucțiunile pe 24**[**biți**](https://ro.wikipedia.org/wiki/Bit)**pe o**[**bandă perforată**](https://ro.wikipedia.org/w/index.php?title=Band%C4%83_perforat%C4%83&action=edit&redlink=1)**, iar datele în contoare electromecanice ce permiteau 23 de cifre.**

**Din cauza lungimii cuvintelor, a tehnologiei de implementare și a structurii memoriei de adresare diferite, în cadrul acestei arhitecturi nu este necesar ca cele două tipuri de memorie *(program si date)* să dispună de aceleași caracteristici. De regulă memoria pentru instrucțiuni are o capacitate mai mare decăt cea de date. De exemplu, microcontrolerele PIC au un cuvânt de date de 8 biți și o lungime a instrucțiunii de 12, 14, 16 sau 32 biți.**

**În funcție de necesarul de memorie, instrucțiunile pot fi stocate de exemplu într-o**[**memorie de tip ROM**](https://ro.wikipedia.org/wiki/Memorie_ROM)**(„*read-only*”), în timp ce datele se află într-o memorie de tip „citire-scriere”.**

**În**[**arhitectura von Neumann**](https://ro.wikipedia.org/wiki/Arhitectura_von_Neumann)**pură**[**CPU**](https://ro.wikipedia.org/w/index.php?title=Unitate_central%C4%83_de_procesare&action=edit&redlink=1)**-ul poate fie citi o instrucțiune, fie citi/scrie date din/în memorie. Nu pot fi efectuate ambele operațiuni simultan, deoarece se folosește aceeași magistrală. Prin contrast, într-un calculator bazat pe arhitectura Harvard, CPU-ul poate citi o instrucțiune și în același timp poate accesa și date, chiar și fără cache. De aceea o arhitectură Harvard poate fi mai rapidă: *fetch*-urile instrucțiunilor și accesul la date nu concurează pentru aceeași cale de transfer de date. Arhitectura Harvard are spații de adresare separate pentru date și instrucțiuni: adresa 0 pentru date nu este aceeași cu adresa 0 pentru instrucțiuni.**

**În perioada modernă, arhitectura Harvard este folosită în principal în două tipuri de dispozitive:**

* ***Procesoarele de semnale* (DSP) specializate din cadrul dispozitivelor pentru prelucrarea semnalelor audio și/sau video.**
* [***Microcontrollerele***](https://ro.wikipedia.org/wiki/Microcontroller)**din cadrul aplicațiilor electronice (PIC de la Microchip, AVR de la Atmel); acestea dispun de dimensiuni mici ale memoriilor de program și de date și se folosesc de avantajul arhitecturii**[**RISC**](https://ro.wikipedia.org/w/index.php?title=Reduced_Instruction_Set_Computing&action=edit&redlink=1)**– executarea instrucțiunilor într-un ciclu mașină (nu neapărat un ciclu de ceas).[Wikipedia -** [**https://ro.wikipedia.org/wiki/Arhitectura\_Harvard**](https://ro.wikipedia.org/wiki/Arhitectura_Harvard)**]**
  1. Evoluția microcontrolerelor

Primul microcontroller a avut la baza procesorul 80C51, care a fost proiecta de Intel.

Producători conacrați ce au dezvoltat platforme bazate pe sisteme RISC – cu set redus de instrucțiuni sunt Atmel, Texas Instrument, Analog Devices, Microcip, Intel.

Evoluția microcontrolerelor este strîns legată de noile tehnologii descoperite în domeniul electronicii.

2.2.1 Familii de microcontrolere reprezentative

**8048 (Intel MCS-48 - www.intel.com )**

**8051 (Intel MCS-51 și mulți alții: Atmel, Philips)**

A doua generație de microcontrolere de 8 biți a firmei Intel care, deși apărută acum 20 de ani, încă ocupă un segment semnificativ de piață. Cu o arhitectură destul de ciudată, este suficient de puternic și ușor de programat (odată învățat!). Arhitectura sa are spații de memorie separate pentru program și date. Poate adresa 64KBytes memorie de program, din care primii 4(8..32)KBytes locali (ROM). Poate adresa 64KBytes memorie de date externă, adresabilă doar indirect. Are 128 (256) octeți de RAM local, plus un număr de registre speciale pentru lucrul cu periferia locală. Are facilități de prelucrare la nivel de bit (un procesor boolean, adresare pe bit). Intel a dezvoltat si un “super 8051” numit generic 80151. Actualmente există zeci de variante produse de diverși fabricanți (Philips, Infineon, Atmel, Dallas, Temic, etc.) precum și cantități impresionante de soft comercial sau din categoria freeware/shareware. Au apărut și dezvoltări ale acestei familii în sensul trecerii la o arhitectură similară (în mare), dar pe organizată pe 16 biți, cu performanțe îmbunătățite ca viteză de prelucrare: familia XA51 eXtended Arhitecture de la Philips și familia 80C251 (Intel). Din păcate aceste noi variante nu s-au bucurat nici pe departe de succesul „bătrânului” 8051.

**80C196 (Intel MCS-96)**

Este un microcontroler pe 16 biți făcând parte din generația treia de microcontrolere a firmei Intel. Destinat inițial unor aplicații din industria de automobile, are o arhitectură von Neumann, cu un spațiu de adresare de 64KBytes, o unitate de intrări/ieșiri numerice de mare viteză (destinată inițial controlului injecției la un motor cu ardere internă), ieșiri PWM, convertor analog numeric, timer watchdog. Există multe variante, ultimele cronologic apărute, fiind mult superioare variantei inițiale. Există și o dezvoltare recentă sub forma familiei MCS-296 (80C296).

**80C186, 80C188 (Intel, AMD, ș.a.)**

Derivate din clasicele 8086/88 prin includerea pe același microcircuit a 2 canale DMA, 2 numărătoare/timere, un sistem de întreruperi și un controler pentru DRAM. Marele avantaj al acestor cvasi(aproape) microcontrolere (ele nu au memorie integrată!) este legat de utilizarea ca mediu de dezvoltare a unor platforme de calcul tip IBM-PC, compatibile 80x86, cu tot softul aferent.

**68HC05 (Freescale)**

Un microcontroler de 8 biți derivat din microprocesorul M6800 și care prezintă multe asemănări cu un alt microprocesor răspândit, la timpul său, 6502. Are un spațiu de memorie unic (64Kbytes) în care sunt plasate și registrele perifericelor (I/O, timere) cu un indicator de stivă (SP) hard pe 5biți (stivă de maxim 32 octeți !). Există variante cu memorie EEPROM, CAN, port serial, etc. Este unul din cele mai [răspândite](https://ro.wikipedia.org/w/index.php?title=R%C4%83sp%C3%A2ndite&action=edit&redlink=1) microcontrolere (comparabil cu 8051). Varianta evoluată a acestei familii este seria 68HC08 bazată pe o nouă unitate centrală de 8 biți numită CPU08, cu cea mai recentă dezvoltare sub forma seriei 68HCS08 destinată în mod special unor aplicații din industria automobilului.

2.2.2 Limbaje de programare

**a. Limbajul mașină și de cel de asamblare**

Limbajul mașină (instrucțiunile mașină) este singura formă de reprezentare a informației pe care un microcontroler o "înțelege" (ca de altfel orice alt sistem de calcul !). Din păcate această formă de reprezentare a informației este total nepractică pentru un programator, care va utiliza cel puțin un limbaj de asamblare, în care o instrucțiune (o mnemonică cu operanzii aferenți) are drept corespondent o instrucțiune în limbaj mașină (excepție fac macroinstrucțiunile disponibile la unele asambloare). Un program în limbaj de asamblare este rapid și compact. Aceasta nu înseamnă că un astfel de program, prost scris, nu poate fi lent și de mari dimensiuni, programatorul având controlul total (și responsabilitatea !) pentru execuția programului și gestiunea resurselor. Limbajul de asamblare este primul care trebuie învățat, chiar sumar, atunci când dorim să proiectăm o aplicație hard/soft cu un anume microcontroler (familie), el permițând înțelegerea arhitecturii acestuia și utilizarea ei eficientă.

**b. Interpretoare**

Un interpretor este o implementare a unui limbaj de nivel înalt, mai apropiat de limbajul natural. Este de fapt un program rezident care, în acest caz, rulează pe o platformă de calcul de tip microcontroler. Caracteristic pentru execuția unui program interpretat, este citirea și executarea secvențială a instrucțiunilor (instrucțiune cu instrucțiune). De fapt fiecare instrucțiune de nivel înalt este interpretată într-o secvență de instrucțiuni mașină care se execută imediat.

**c. Compilatoare**

Un compilator combină ușurința în programare oferită de un interpretor (de fapt de limbajul de nivel înalt) cu o viteză mai mare de execuție a codului. Pentru aceasta programul, în limbaj de nivel înalt, este tradus direct în limbaj mașină sau în limbaj de asamblare (urmând a fi apoi asamblat). Codul mașină rezultat are dimensiuni relativ mari (dar mai mici decât cel interpretat) și este executat direct, ca un tot, de microcontroler. De regulă codul generat poate fi optimizat fie ca dimensiune, fie ca timp de execuție. Se pot enumera compilatoare pentru limbajele: C, BASIC, Pascal, PL/M (Intel), Forth.

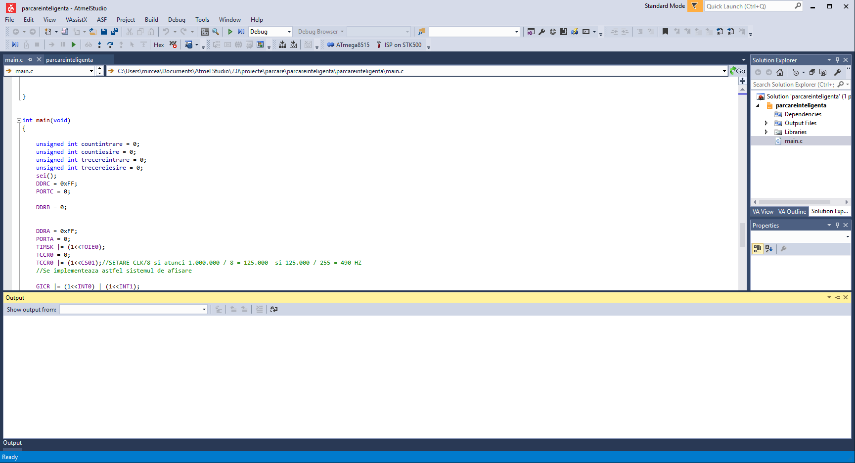


Figura. 2.4 Interfața IDE Atmel Studio

Există o serie foarte largă de medii de dezvoltare , o parte din acestea sunt puse la dispozitie de către firmele ce dezvoltă microcontrolerele, cum putem observa în figura 2.4, aici este interfața mediului IDE – Atmel Studio, ajuns la versiunea 7, special create pentru a funcționa pe sistemele gazdă de 64 de biți și windows 10, fiind dezvoltată în limbajul DOT NET 5.

Texas Instrument, dezvoltă un IDE , CodeComposer, ce reunește toate resursele necesare scrierii de firmware, editor graphic cu facilitate de autosuggest, autocomplete, detectarea erorilor de sintaxă, depanator integrat, atăt în mod simulare căt și în timp real cu ajutorul diferitelor plăci de dezvoltare și a starter- kitt-urilor.

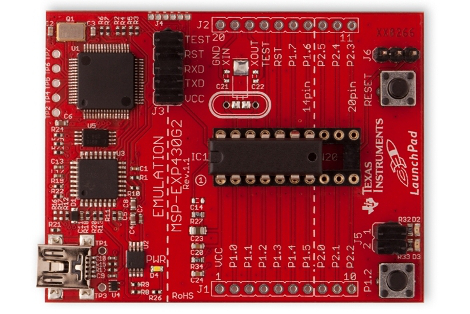


Figura.2.4 – LaunchPad – exemplu de starter-kitt de la T.I

Cu ajutorul acestor resurse , hardware și software operația de dezvoltare a unui system bazat pe microcontroller este mult ușurată.

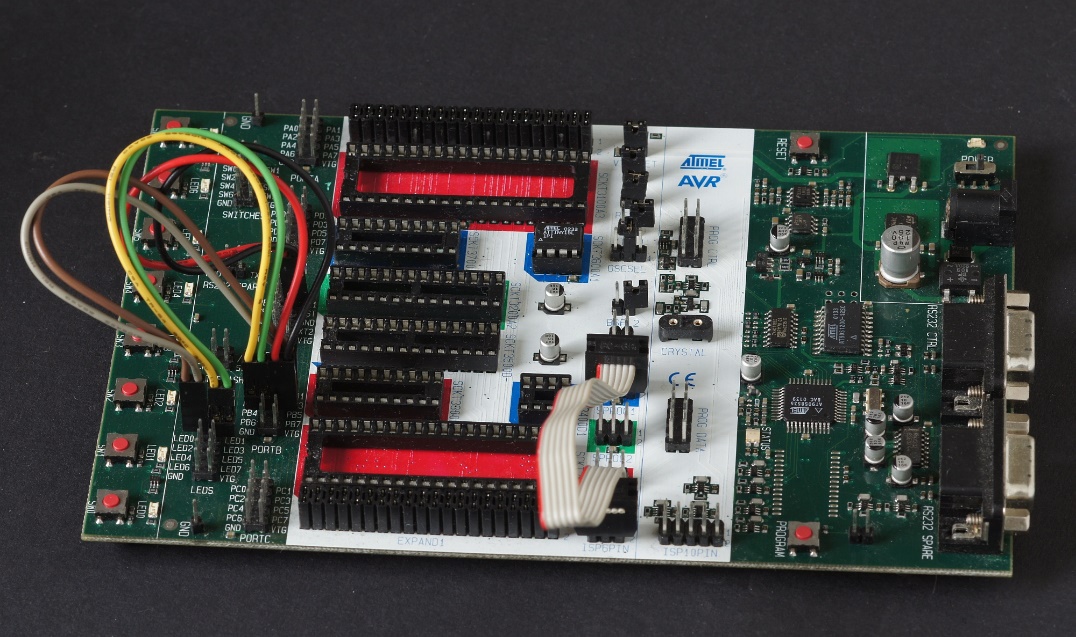


Figura.2.5 Kitt dezvoltare produs de Atmel – STK500

Kit-urile de dezvoltare conțin resursele necesare pentru a testa și dezvolta o aplicație.

În figura 2.5 se ilustrează placa de dezvoltare , ce are codul STK-500, produsă de către firma Atmel, aceasta poate acomoda o serie foare mare de microcontrolere de 8 biti, în zona ei mediană putîndu-se observa o serie de socluri cu număr variat de pini, simbolizate în culori diferite, roșu, verde, albastru, socluri ce acomodeaza diverse microcontrollere , de la cele mai mici, Atiny, pîna la cele mai evaluate microcontrolere de 8 biți – Atmegaxxxx.

Placa dispune de o serie de 8 leduri ce pot fii utilizate ca dispositive de ieșire și un set de 8 microswitch-uri de tip push-button ce sunt utilizate ca dispositive de intrare.

Toate acestea putînd fii conectate la fiecare din cele 4 porturi, PORTA, PORTB, PORTC, PORTD, cu ajutorul unor pinheaderi cu 10 pini.

Operația de pogramare a sistemului țintă , este și ea configurabilă, putînd allege dintre metoda ISP (In System Programming) și cea Paralelă.

Cu ajutorul jumperilor putem configura diverse resurse cum ar fii , tipul oscilatorului , intern sau extern (caz în care dispunem de un oscillator de tip Collpits cu cristal de cuarț), referința analogică , etc ..

Programarea se face prin portul serial USART, tot aici mai avem la dispoziție un port USART pentru testare.

Alimentarea electrică este de tip multipolaritate, în cc.

1. **Descrierea aplicației**

Proiectarea aplicației sa făcut ținînd cont , pe deoparte de considerentele și specificașiile tehnice , iar pe cealaltă parte de latura practică și economică.

Sa pornit de la o idee de gestionare a locurilor disponibile într-o parcare auto.

Ca orice problem, și abordarea noastră sa făcut într-un mod etapizat, căutîndu-se la fiecare etapă din procesul de proiectare o soluție eficientă, pornind de la datele problemei.

Astfel, într-o primă etapă sa indentificat și testat soluții pentru determinarea numărului de vehicole ce intră sau ies din parcare, metode de detecție pot include,

Metode optice, metode mecanice, metode radio … etc.

Analizănd diverse documentatii tehnice , a rezultat că cea mai eficientă metodă de detecție este cea cu barieră de infraroșu , folosind o pereche emițător – recemtor cu infraroșu.



Figura. 3.1 Receptorul infraroșu.

Radiația infraroșie este o radiație electromagnetic ace are o frecvență foarte mare , spectral situînduse pe o poziție ca cea exemplificată în figura de mai jos.

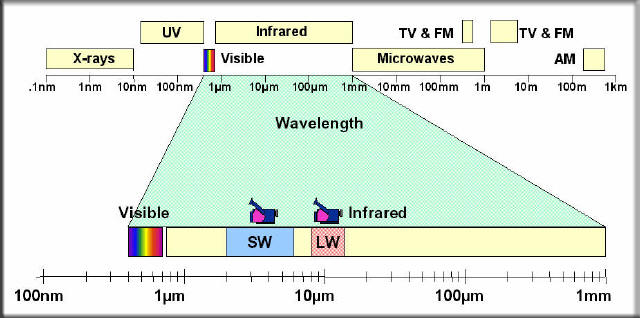
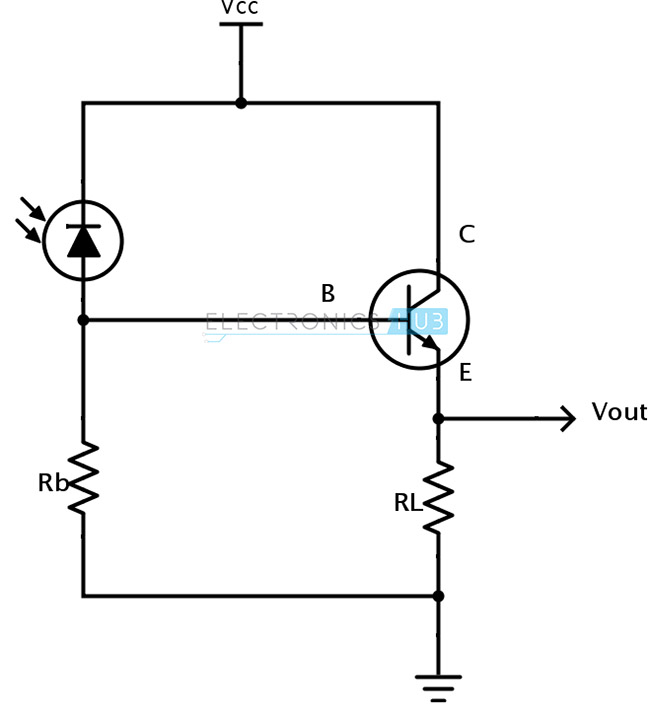


Figura. 3.2 Spectrul radiației electromagnetice.

Receptorul de infraroșu este practic o fotodiodă care este sensibilă la această lungime de undă.

O metodă clasică , simplă de conectare a fotodiodei, pentru a putea detecta semnalul luminous este cea în care se utilizează un transistor NPN în configurație de amplificator, repetor pe emitor.

****

**Figura. 3.3 Circuit simplu de amplificare pentru senzorul IR**

În această schema avem două rezistențe electrice și anume RL, rezistența de sarcină prin care va trece curentul de collector al tranzistorului, practic rezistența electrică a consumatorului, Rb este rezistența de polarizare a bazei , ea se dimensionează în funcție de caracteristica de transfer a fotodiodei.

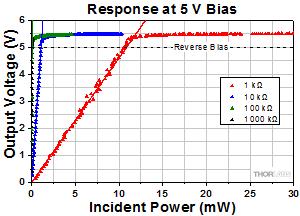


Figura.3.4 Exemplu de caracteristică de transfer

În graficul de mai sus observăm trasat o caracteristică de transfer, pentru mai multe valori ale rezistenței de polarizare.

Analizînd graficul, observăm că odată cu creșterea valorii rezistenței de polarizare, panta devine din ce în ce mai abruptă, ajungîndu-se la tensiunea de saturație la valori din ce în ce mai mici ale puterii radiației incidente.

Un alt aspect de care trebuie ținut con teste reprezentat de sensibilitatea spectrală a fotodiodei, deoarece, în funcție de acesta vom decide asupra tipului de emițător optic (LED-IR).

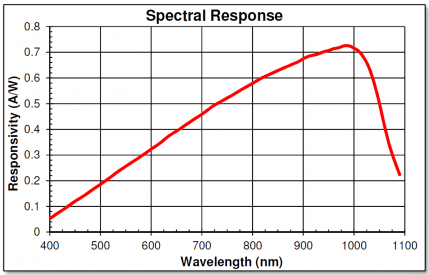


Figura.3.5 Răspunsul spectral

Pentru ca întreg sistemul să fie functional mai avem nevoie de o sursă de energie luminoasă cu spectrul situate în domeniul infraroșu.

Aceasta poate fi un LED infraroșu, o diodă electroluminiscentă, care spre deosebire de diodele clasice au o fantă transparentă pentru radiația emisă.

La polarizare direct, diode va începe să emită radiație electromacnetică, existănd în funcție de producător, diferite nivele ale tensiunii de polarizare direct precum și a curentului electric consumat, totul depinzînd de puterea emisa (în mWatt).



Figura.3.6 Emitător infraroșu – LED

Emitătorul ca și receptorul sunt perechi în funcție de banda spectrală în principiu conform graficului de răspuns spectral.

Simbolic, în electronica cele două elemente sunt reprezentate grafic astfel

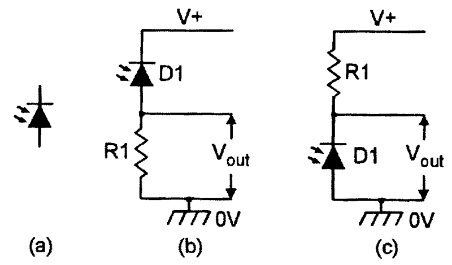
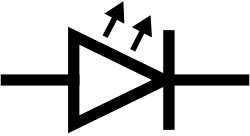


Figura. 3.7 Simbolul diodei led, (a) – receptorul , (c), (d) mod de conectare

Interfațarea cu utilizatorul se face cu ajutorul sistemului de afișare.

Acesta poate fi realizat în mai multe moduri, utilizănd ecrane, sau display de tip CRT, cee ace impune un consum mare de energie, neeconomic, complex din punct de vedere al interfațării , sau utilizînd sistemele de afișaj electronic reprezentat prin digiți cu 7 segmente, deasemenea se poate utiliza o versiune omonimă, pe bază de cristale lichide.

Sistemele de afișare cu 7 – segmente, reprezintă avantajul lzibilității pe timp de zi, existînd panouri de dimensiuni mari , panouri ce pot fi montate la intrarea parcării.

În oricare caz, principiul de funcționare este asemănător, elemental de bază fiind diode electroluminiscentă.

Aceste diode sunt plasate pe un support isolator din punct de vedere electic, într-o configurație specifică, suportul asigurînd atăt conexiunile electrice, ăntre terminalele diodelor căt și stabilitatea mecanica necesară bunei funcționări.

Dispunerea fizică a diodelor electroluminiscente se face astfel încăt sa se realizeze simbolul cifrei 8.

În fața fiecărei diode din segment, se află un difuzor optic, difuzor care transferă lumina de la sursă către suprafață, eliminarea interferențelor luminoase dintre segmente se face brin inglobarea difuzoarelor optice în rășini sintetice opace.

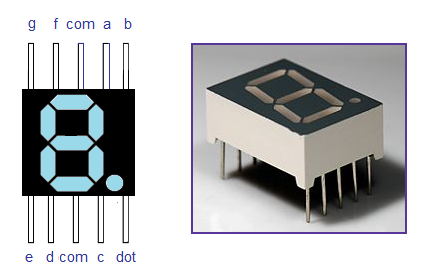


Figura.3.8 Afișaj cu 7 segmente.

În figura de mai sus putem vedea un digit , acesta poate reprezenta un singur symbol numeric la un moment dat.

În practică, datorită necesității reprezentării numerice a unor valori mai mari decăt cele date de ordinal unităților, se folosesc grupări de 3, 4, 6 9 12 digiți.

Modul de conectare al elementului comun , anod comun sau catod comun, va determina logica de activare a segmentelor pentru realizarea afișării, astfel, pentru catod comun avem logica pozitivă, unde pentru a active un segment, pe pinul respective corespunzător segmentului trebuie să aplicăm o tensiune pozitivă, iar pentru sistemele cu anod comun, vom folosii logica negative, unde , pentru a active un segment va trebui sa aplicăm o tensiune de 0v.

Tabelul următor evidențiază modul de folosire ale celor două logici, pozitivă și negative.

* 1. Logica pozitivă – system de afișare cu catod comun

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| a | b | c | d | e | f | g | simbol |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 2 |
| 1 | 1 | 1 | 1 | 0 | 0 | 1 | 3 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 4 |
| 1 | 0 | 1 | 1 | 0 | 1 | 1 | 5 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 6 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 7 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 8 |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 | 9 |

* 1. Logina negativă – sistem de afișare cu anod comun

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| a | b | c | d | e | f | g | simbol |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 2 |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 3 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 4 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 5 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 6 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 7 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 9 |

Indiferent de metoda aleasă, trebuie asigurată pentru o iluminare uniformă a simbolului, rezistențe de limitare a curentului pentru fiecare segment.

Topologiile de conectare a digiților cu microcontrolerul sunt în număr de două, conexiunea putînd fii directă sau prin multiplexare în timp.

Conexiunea directă necesită utilizarea unui număr mare de porturi pentru microcontroller, a unui număr mare de trasee sau conductori electrici, de exemplu, pentru a putea afișa numere de ordinul miilor, avem nevoie depatru digiți, ceea ce înseamnă un număr de 28 de conexiuni.

Conexiunea prin multiplexare cu diviziune în timp, simplifică mult problema, pentru situația prezentată mai sus neavănd nevoie decît de 7 conexiuni pentru cele 7 segmente și 4 conexiuni pentru elemenții comuni.

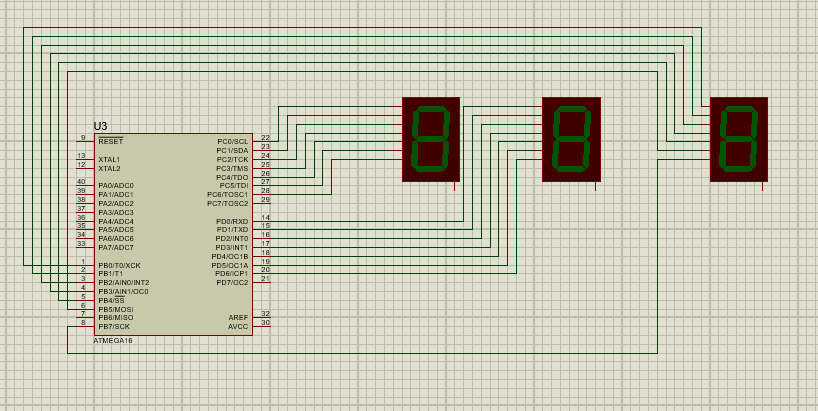


Figura. 3.8 Conectarea clasică

În figura de mai sus putem vedea o schemă electrică unde este exemplificată conectarea clasică , nemultiplexată a digitilor de afișare.

Pentru cei trei digiți , practic au fost necesare toate porturile microcontrollerului.

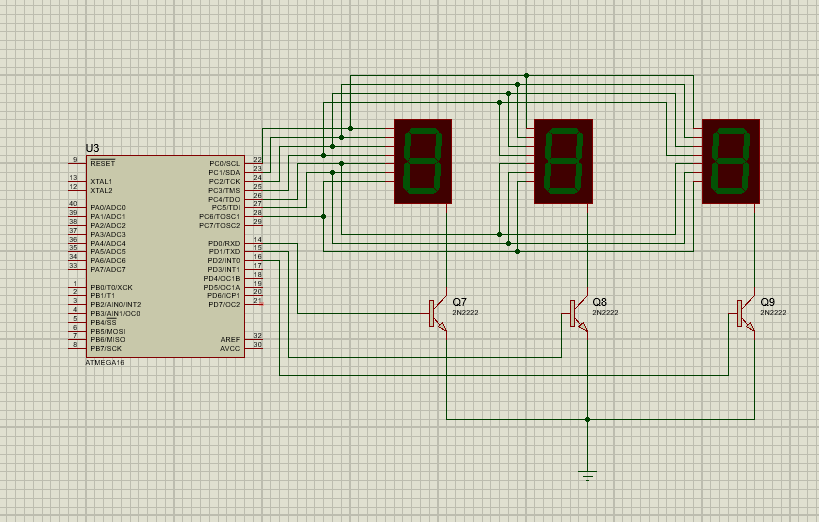


Figura.3.9 Conexiunea prin multiplexare

Avînd în vedere cele discutate, putem contura schema bloc a sistemului ca fiind următoarea.

UC

SENZORI INTRARE

SENZORI IEȘIRE

AFIȘAJ

Figura.3.10 Schema de principiu

Folosim tehnica de proiectare, bottom-up, utilizată cu precădere cănd putem discerne între elementele constituiente ale sistemului, acele cărămizi care stau la baza sistemului, de regulă , porți logice, circuite logice combinaționale (CLC), circuite logice secvențiale (CLS), alte subsisteme integrate, putem, realiza schema electrică de principiu.

Elementul de bază este sistemul integrat de procesare, microcontrollerul, alegerea acestuia se face punînd în balanță costurile de producție și necesarul de resurse ce-l poate pune la dispoziție.

Resursele fizice necesare sunt date, simplu prin analiza problemei.

Asfel avem nevoie pentru senzorii de intrare și ieșire de cîte o intrare disponibilă, pentru afișare avem nevoie de 10 ieșiri.

Atmega 8515 pune la dispoziția utilizatorului, la un preț rezonabil de cost, tot necesarul specificat mai inainte.

Analizănd foaia de catalog, observăm că putem folosii un număr de 32 de porturi digitale, configurabile independent la nivel de bit, ca porturi de intrare respectiv ca porturi de ieșire.

Deasemenea, microcontrollerul dispune de un număr de 3 intrari externe de întrerupere, notate, INT0, INT1, INT2.

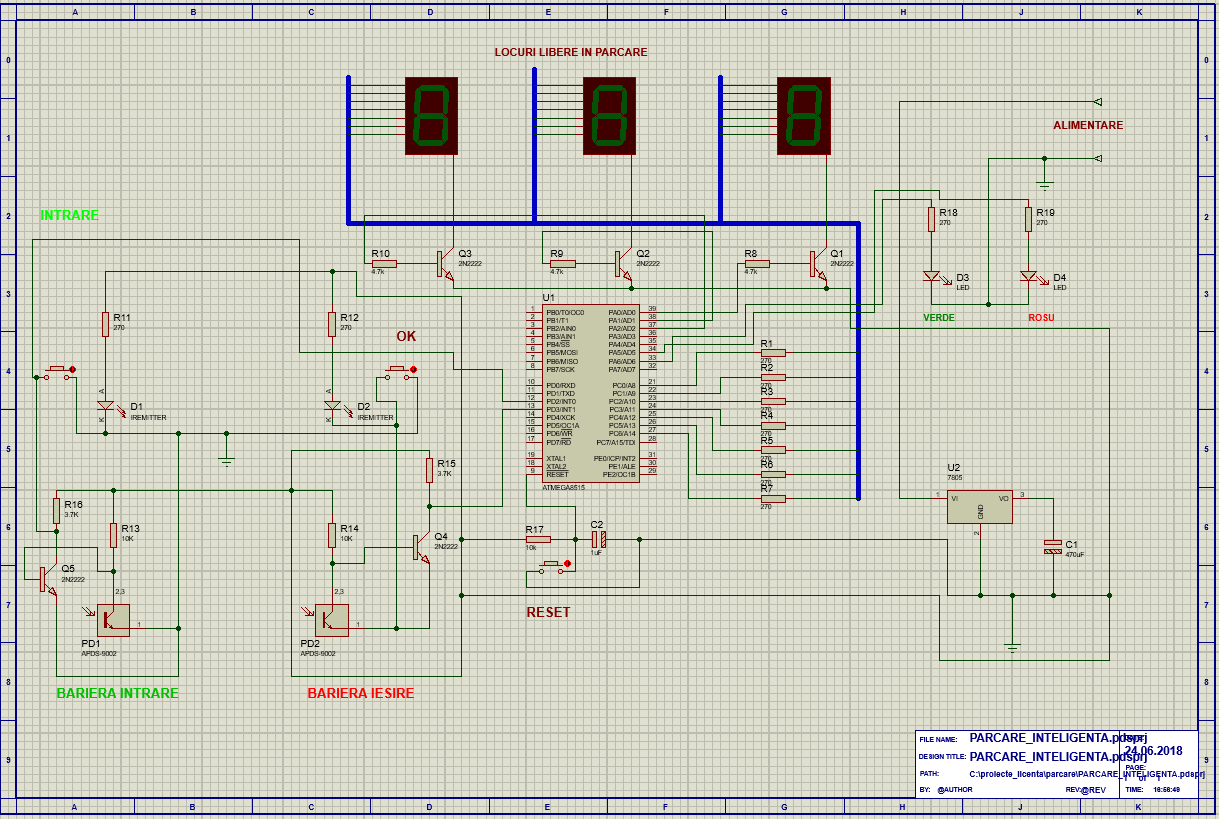


Figura.3.11 Schema electrică de principiu.

Ieșirile fiecărui traductor optic se vor cupla la intrările de întreruperi INT0 respectiv INT1 astfel.

Pentru bariera optică poziționată la intrarea în parcare, se alege intrarea INT0, iar pentru ieșirea din parcare, bariera a fost cuplată la intrarea INT1.

La intrare, s-a prevăzut, ca metodă alternativă, în cazul de defectare a senzorului optic, un buton fără reținere, a cărei acționare manuală va produce în sistemul nostru acelaș efect ca și întreruperea radiației luminoase la trecerea prin barieră.

La ieșire, traductorul optic, este cuplat la microcontroler pe intreruperea simbolizată INT1, aici a fost prevăzut și un buton de confirmare a ieșirii, a cărei apăsare va duce la incrementarea numărului de locuri disponibile în parcare.

Afișarea locurilor disponibile se realizează în mod automat, utilizănd trei digiți cu 7 segmente, pentru fiecare rang din simbolul corespunzător numarului de locuri.

Comanda catozilor comuni se realizează prin comutare electronică succesivă – permanentă cu o anumita cadență în ordinea – unități – zeci – sute, cu ajutorul a trei tranzistori bipolari cuplați în modul emitor comun.

Activarea se face utilizănd biții portului A, pinii find alocați conform tabelului următor

|  |  |
| --- | --- |
| PIN PORTA | RANG |
| PA0 | UNITAȚI |
| PA1 | ZECI |
| PA2 | SUTE |

Modul de conectare poate fi întămplător dar, trebuie menținută o ordine a lucrurilor pentru a ne putea ușura munca în elaborarea programului firmware.

Conectarea segmentelor, se poate face deasemenea utilizînd oricare dintre porturile rămase libere, conform foii de catalog al producătorului microcontrollerului Atmega 8515, fiecare port poate furniza un curent de maximum 40mA, arhisuficient pentru comanda afișajului cu 7 segmente.

Am ales conectarea celor sapte segmente utilizănd PORTC, conexiunea se realizează conform tabelului.

|  |  |
| --- | --- |
| PORTC | SEGMENT |
| PA6 | a |
| PA5 | b |
| PA4 | c |
| PA3 | d |
| PA2 | e |
| PA1 | f |
| PA0 | g |

În afară de acestea, mai rămîne de asigurat alimentarea cu tensiune electrică stabilizată de 5volți preoblemă simplu de rezolvat, utilizămd un stabilizator integrat linear L7805.

De asemenea a fost prevăzut un buton de reset.

Avînd în vedere că , pentru a putea funționa, sistemele cu logică programată au nevoie de un program, o aplicație software, dezvoltată de către persoane ce stăpînesc un limbaj de programare adecvat, capabile să abstractizeze problema la nivel software.

Aplicația poate fii dezvoltată utilizînd diverse limbaje, dar dintre toate cel mai utilizat este limbajul C acesta dispunînd de

1. Portabilitate
2. Standardizare
3. Flexibilitate
4. Diversitate
5. Suplețe
6. Compatibilitate
7. Maturitate

Aceste caracteristici, fac ca limbajul C sa fie utilizat în programarea diferitelor aplicații pe variate platforme, platforme care înclud și microcontrolerele.

Dezvoltarea aplicației poate fii făcută, în funcție de cunoțtințele, iscusința , priceperea programatorului , începănd cu simple editoare de text, pînă la medii integrate de dezvoltare.

Oricare ar fii metoda aleasă, etapele în scrierea unui program sunt următoarele

1. Editarea codului sursă
2. Verificarea sintactică și morfologică
3. Compilarea
4. Editarea de legături – Link-Editarea
5. Execuția programului
6. Detectarea și corectarea erorilor
7. Întreținerea și dezvoltarea programului

Alegerea modului de programare se face urmărind un efort minim.

Programele ce necesită relaționări logice se dezvoltă utilizînd limbaje logice

Cele matematico – logic – utilizănd limbaje funcționale.

Comun, pentru aproape toate tipurile de probleme este utilizarea programării structurate.

Astfel un algoritm, poate fii implementat utilizînd minim trei structuri de program și anume

1. Secvența
2. Decizia
3. Iterația

Programarea structurată, utilizează gruparea instrucțiunior în secvențe de instrucțiunii creînd astfel funcții, fapt ce usurează pe deoparte scrierea programului , iar pe de altă parte citirea lui de către alte persoane cu scopul de al dezvolta și întreține pe mai departe.

Orice program C, are o structură bine definită, astfel avem

#directive preprocesor

Ex #include <avr/io.h>

Variabile de tipul global – vizibile din orice funcție a programului

Ex

static unsigned int locurilibere = 10;

Declarații și definiții de funcții, zona de fișier text în care sunt declarate funcțiile utilizator

Funcția main, este o funcție specială, existentă în orice program C, în corpul ei avem declaratii de variabile locale și apelul funcțiilor declarate anterior în program.

Aplicați a fost împărțită în cîteva funcții ce vor avea utilitate în afișare, gestionarea locurilor disponibile în parcare și bineînțeles funcția main

void afisare\_LCD(unsigned int numar1, unsigned int digit)

Funcția afisare\_LCD implementează afișarea în mod multiplexat al numărului de locuri libere din parcare.

Parametrii funcție, ce sunt transmiși prin valoare, la momentul apelului fiind înlocuiti cu valoarea numerică reală, sunt

Unsigned int numar1, reprezintă valoarea numerică ce trebuie afișată, el putănd lua orice valoare în intervalul numeric 0 – 999.

Unsigned int digit, reprezintă , prin codificare rangul numărului astfel

|  |  |
| --- | --- |
| **Valoare unsigned int digit** | **RANG** |
| 0 | unitați |
| 1 | zeci |
| 2 | sute |

Valoarea numerică memorată în variabila rang1, este separata în cifre utilizănd un algoritm pe baza memorării restului împărțirii la 10, pîna cănd rezultatul împărțirii este zero.

Numărul este astfel separat în cifre constituente, memorate separat în variabile denumite sugestiv, unități zeci, sute.

Cît timp numar diferit de zero

Execută

Rest = numar modulo 10

Numar = numar / 10

Codul sursa este următorul

unsigned int separat[3];

unsigned int unitati, zeci, sute;

unsigned int valoare, count, aux;

valoare = numar1;

count = 0;

aux = 0;

separat[0] = separat[1] = separat[2] = 0;

while(valoare!=0)

{

aux = valoare % 10;

valoare = valoare / 10;

separat[count] = aux;

count ++;

}//END WHILE

sute = separat[2];

zeci = separat[1];

unitati = separat[0];//Separa numarul in componente numar1

Astfel , elementele numărului pot fii afisate succesiv, prin multiplexare în timp, prin trimiterea conținutului variabilelor sute, zeci unități catre PORTC, urmată de selectarea corespunzatoare a pinului de comanda prin PORTA.

Un fragment de cod, corespuzător afișarii rangului corespunzător unităților este următorul

if(digit == 0){

PORTA |= (1<<PA0);

PORTA &= ~((1<<PA2)|(1<<PA1));

switch(unitati){

case 0 : PORTC = 0b01111110; break;

case 1 : PORTC = 0b00110000; break;

case 2 : PORTC = 0b01101101; break;

case 3 : PORTC = 0b01111001; break;

case 4 : PORTC = 0b00110011; break;

case 5 : PORTC = 0b01011011; break;

case 6 : PORTC = 0b01011111; break;

case 7 : PORTC = 0b01110000; break;

case 8 : PORTC = 0b01111111; break;

case 9 : PORTC = 0b01111011; break;

}

}//END IF

Identic se procedează pentru rangul zecilor respectiv sutelor, deosebirea constă în valoarea numărului ce este comparat în clauza instrucțiunii if , 1 – pentru zeci respectiv 2 pentru sute.

Funcșia main a aplicației conține instrucțiuni de configurare a porturilor ca porturi de iesire sau de intrare , de exemplu

DDRC = 0xFF;

PORTC = 0;

DDRB = 0;

DDRA = 0xFF;

PORTA = 0;

Unde portul PORTC este configurat ca port de ieșire

Instrucțiuni pentru activarea globală a sistemului de întreruperi

Sei()

Configurarea întreruperilor INT0 respectiv INT0 și activarea acestora pe port descrescător, dată fiind efectul de inversare a logicii conferit de tranzistorul de comandă din bariera optică

GICR |= (1<<INT0) | (1<<INT1);

MCUCR |= (1<<ISC11)|(1<<ISC10)|(1<<ISC01)|(1<<ISC00);

Deoarece, microcontrolerul, pune la dispoziție programatorului o modalitate de implementare a firelor de execuție separate în program, realizănd asfel un sistem multithread, execuția funcției de afișare, care este destul de critică, orice înărziere în program ducînd la o afișare nesincronă, defectoasă, putînd fii chiar ilizibilă utilizatorului, se va face prin apelarea într-un fir separat de execuție implementat printr-un serviciu de tratare a întreruperilor, alocat intreruperilor în cascadă generate de depășirea capacității de numărare a registului asignat temporizatorului de opt biți TIMER0.

Implementarea se face astfel

ISR(TIMER0\_OVF\_vect){

afisare\_LCD(locurilibere, baleiaj);

baleiaj = baleiaj + 1;

if(baleiaj == 3){baleiaj = 0;}

}

Funcția ISR fiind apelată foarte des, va realiza pe deoparte afișarea valorii memorate de către variabila locuri libere, variabilă ce va fii decrementată cu o unitate la întrarea unei mașini în parcare, respectiv incrementată cu o unitate la ieșirea unei mașini din parcare.

Afișarea se va face pentru fiecare rang din număr cu o viteză , experimental aleasă, astfel încăt să se evite efectul de pîlpăire.

Incrementarea respectiv decrementarea locurilor din parcare, se face la apariția unei întreruperi corespunzătoare, întrerupei tratate de către funcțiile INT0 și INT1

ISR(INT0\_vect){

if(locurilibere > 0){

locurilibere = locurilibere - 1;

}

}

ISR(INT1\_vect){

if(locurilibere < 100){

locurilibere = locurilibere + 1;

}

}

La apariția unei tranziții negative pe front pentru INT0 asignată senzorului de intrare , se va apela ISR(INT0\_vect) serviciu de tratare a intreruperii corespunzător intrării de întrerupere INT0, situație în care se vor decrementa locurile libere.

La apariția unei tranziții pe front scăzător, pentru ieșire, respectiv pentru intrarea de întrerupere INT1, se va apela serviciu de tratare a ăntreruperii alocat INT1, caz în care variabila locurilibere va fii incrementată, singura constrîngere este ca ea să nu depășească numărul de locuri disponibile, respectiv 100 de locuri.

Activarea temporizatorului se face în funcția main prin secvența

TIMSK |= (1<<TOIE0);

TCCR0 = 0;

TCCR0 |= (1<<CS01);//SETARE CLK/8 si atunci 1.000.000 / 8 = 125.000 si 125.000 / 255 = 490 HZ

Tot aici se realizează semnalizarea cu ajutorul a două led-uri distinct colorate, respectiv – ROȘU cînd numărul de locuri din parcare este 0 și VERDE cînd există cel puțin un loc liber în parcare, prin următoarea secvență de cod

if(locurilibere>0){

PORTA |= (1<<PA6);

PORTA &= ~(1<<PA5);

}

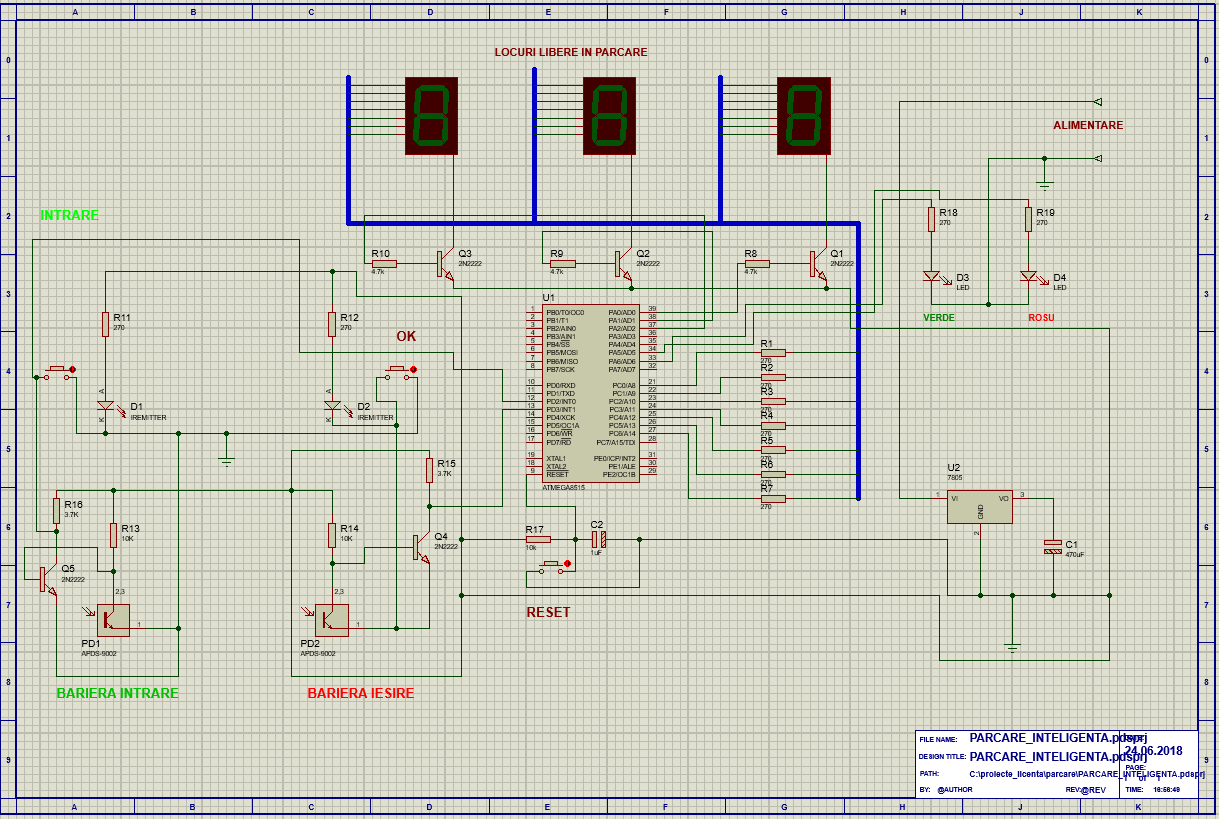
else {

PORTA |=(1<<PA5);

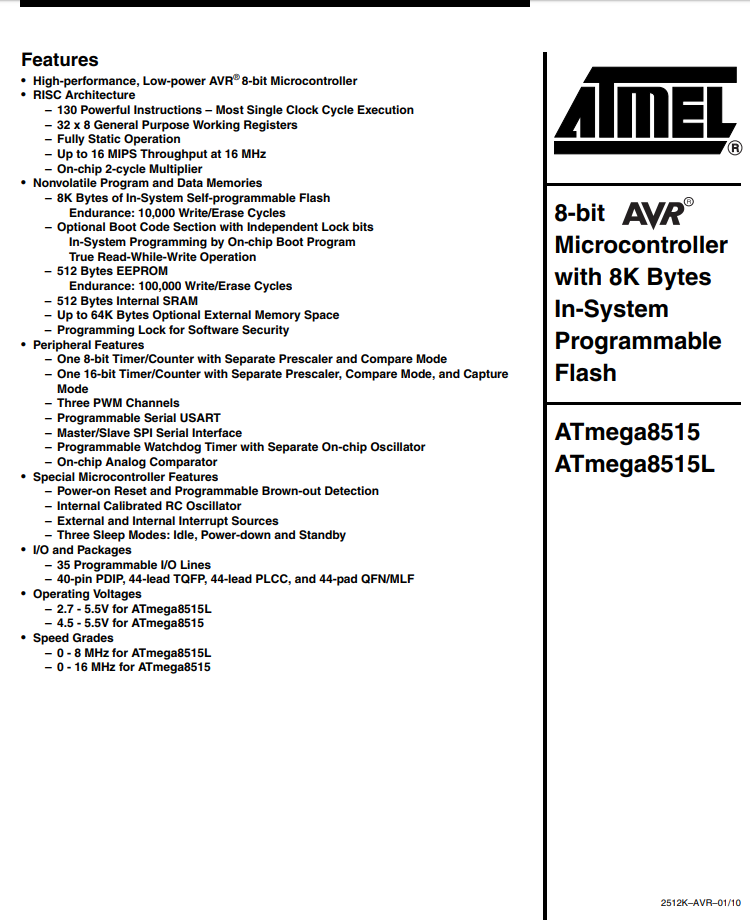
PORTA &= ~(1<<PA6);

}

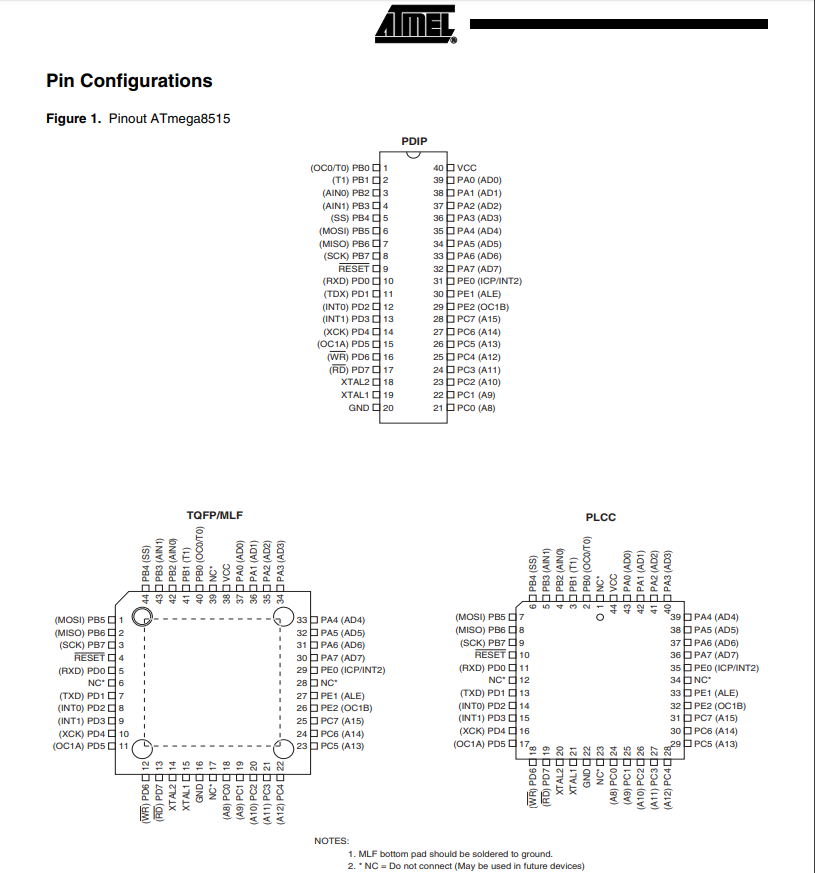
Anexa1. Schema electrică de principiu



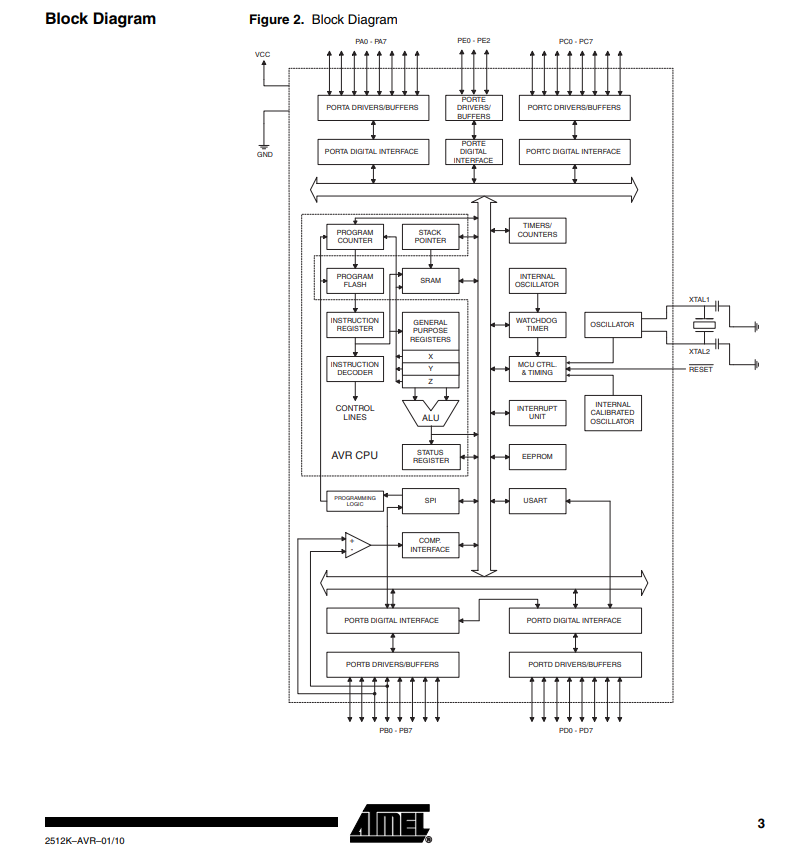
Anexa2. Facilități și caracteristici Atmega 8515



Anexa 3. 8515 asignarea pinilor la capsulă



Anexa.4 8515 – Structură



Anexa. 5 Cod sursă

#include <avr/io.h>

#include <avr/interrupt.h>

#include <avr/delay.h>

#define *F\_CPU* 1000000

#define zero PORTD = 0xb00000010

#define unu PORTD = 0xb10011110

#define doi PORTD = 0xb00100100

#define trei PORTD = 0xb00001100

#define patru PORTD = 0xb10011000

#define cinci PORTD = 0xb01001000

#define sase PORTD = 0xb01000000

#define sapte PORTD = 0xb00011110

#define opt PORTD = 0xb00000000

#define noua PORTD = 0xb00001000

static unsigned int locurilibere = 10;

static unsigned int baleiaj = 0;

static unsigned int intrare = 0;

static unsigned int iesire = 0;

void afisare\_LCD(unsigned int numar1, unsigned int digit)

{

unsigned int separat[3];

unsigned int unitati, zeci, sute;

unsigned int valoare, count, aux;

valoare = numar1;

count = 0;

aux = 0;

separat[0] = separat[1] = separat[2] = 0;

while(valoare!=0)

{

aux = valoare % 10;

valoare = valoare / 10;

separat[count] = aux;

count ++;

}//END WHILE

sute = separat[2];

zeci = separat[1];

unitati = separat[0];//Separa numarul in componente numar1

if(digit == 0){

PORTA |= (1<<PA0);

PORTA &= ~((1<<PA2)|(1<<PA1));

switch(unitati){

case 0 : PORTC = 0b01111110; break;

case 1 : PORTC = 0b00110000; break;

case 2 : PORTC = 0b01101101; break;

case 3 : PORTC = 0b01111001; break;

case 4 : PORTC = 0b00110011; break;

case 5 : PORTC = 0b01011011; break;

case 6 : PORTC = 0b01011111; break;

case 7 : PORTC = 0b01110000; break;

case 8 : PORTC = 0b01111111; break;

case 9 : PORTC = 0b01111011; break;

}

}//END IF

if(digit == 1){

PORTA |=(1<<PA1);

PORTA &= ~((1<<PA2)|(1<<PA0));

switch(zeci){

case 0 : PORTC = 0b01111110; break;

case 1 : PORTC = 0b00110000; break;

case 2 : PORTC = 0b01101101; break;

case 3 : PORTC = 0b01111001; break;

case 4 : PORTC = 0b00110011; break;

case 5 : PORTC = 0b01011011; break;

case 6 : PORTC = 0b01011111; break;

case 7 : PORTC = 0b01110000; break;

case 8 : PORTC = 0b01111111; break;

case 9 : PORTC = 0b01111011; break;

}

}

if(digit == 2){

PORTA |=(1<<PA2);

PORTA &= ~((1<<PA1)|(1<<PA0));

switch(sute){

case 0 : PORTC = 0b01111110; break;

case 1 : PORTC = 0b00110000; break;

case 2 : PORTC = 0b01101101; break;

case 3 : PORTC = 0b01111001; break;

case 4 : PORTC = 0b00110011; break;

case 5 : PORTC = 0b01011011; break;

case 6 : PORTC = 0b01011111; break;

case 7 : PORTC = 0b01110000; break;

case 8 : PORTC = 0b01111111; break;

case 9 : PORTC = 0b01111011; break;

}

}

}

ISR(TIMER0\_OVF\_vect){

afisare\_LCD(locurilibere, baleiaj);

baleiaj = baleiaj + 1;

if(baleiaj == 3){baleiaj = 0;}

}

ISR(INT0\_vect){

if(locurilibere < 100){

locurilibere = locurilibere + 1;

}

}

ISR(INT1\_vect){

if(locurilibere > 0){

locurilibere = locurilibere - 1;

}

}

int main(void)

{

unsigned int countintrare = 0;

unsigned int countiesire = 0;

unsigned int trecereintrare = 0;

unsigned int trecereiesire = 0;

sei();

DDRC = 0xFF;

PORTC = 0;

DDRB = 0;

DDRA = 0xFF;

PORTA = 0;

TIMSK |= (1<<TOIE0);

TCCR0 = 0;

TCCR0 |= (1<<CS01);//SETARE CLK/8 si atunci 1.000.000 / 8 = 125.000 si 125.000 / 255 = 490 HZ

//Se implementeaza astfel sistemul de afisare

GICR |= (1<<INT0) | (1<<INT1);

MCUCR |= (1<<ISC11)|(1<<ISC10)|(1<<ISC01)|(1<<ISC00);//pagina 77 din datasheet

//Se implementeza sistemul de citire al potentiometrelor

//Am implementat citirea potentiometrilor pe ADC, sistemul de afisare

/\* Replace with your application code \*/

while (1)

{

//

if(locurilibere>0){

PORTA |= (1<<PA6);

PORTA &= ~(1<<PA5);

}

else {

PORTA |=(1<<PA5);

PORTA &= ~(1<<PA6);

}

}

}