Universitatea Politehnica din București
Facultatea de Electronică, Telecomunicații și Tehnologia Informației

Sistem inteligent de monitorizare a greutății stupilor de albine

Lucrare de licență

Prezentată ca cerință parțială pentru obținerea titlului de
Inginer în domeniul *Electronică și Telecomunicații*
programul de studii de licență *Microelectronică,Optoelectronică și Nanotehnologii*

Conducător științific,
Ș.l.Dr.Ing. Horia CUCU

Absolvent,
Oana – Teodora GRIGORE

București
2015

Universitatea "Politehnica" din Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
Departamentul : DCAE

Aprobat Director de Departament :
Prof. dr. ing. GHEORGHE ŞTEFAN

TEMA PROIECTULUI DE DIPLOMĂ
a studentului Grigore F. Oana – Teodora ,grupa 442E

1. Titlul temei: Sistem inteligent de monitorizare a greutatii stupilor de albine

2. Contribuţia practică, originală a studentului va consta în (*în afara* părţii de documentare):

- realizarea unui cantar electronic inteligent, capabil sa transmita periodic (cu ajutorul unui modul GSM) mesaje SMS programabile precizand greutatea masurata, diferentele de greutate in timp, praguri de greutate, etc.
- componenta hardware va include un procesor (cu acces la un senzor audio, accelerometru, USB, etc), un modem GSM si un modul de conversie, pentru interfatarea cu procesorul, capete cantare, convertoare AD pentru cantare, etc.
- componenta software este reprezentata de programul ce va fi realizat si inscris in microcontroller.

3. Proiectul se bazează pe cunoştinţe dobândite în principal la următoarele 3-4 discipline:
Arhitectura microprocesoarelor, Microcontrolere, Dispozitive electronice, Circuite electronice fundamentale

4. Proprietatea intelectuală asupra proiectului aparţine: UPB

5. Locul de desfăşurare a activităţii: UPB

6. Realizarea practică rămâne în proprietatea: STUDENT

7. Data eliberării temei: 15.08.2014

CONDUCĂTOR LUCRARE:
S.l.dr.ing. Horia Cucu

STUDENT:
Oana-Teodora Grigore
Grigore Oana

Declaraţie de onestitate academică

Prin prezenta declar că lucrarea cu titlul „*Sistem inteligent de monitorizare a greutăţii stupilor de albine”*, prezentată în cadrul Facultăţii de Electronică, Telecomunicaţii şi Tehnologia Informaţiei a Universităţii „Politehnica” din Bucureşti ca cerinţă parţială pentru obţinerea titlului de Inginer în domeniul *Inginerie Electronică şi Telecomunicaţii*, programul de studii *Microelectronică, Optoelectronică şi Nanotehnologii*, este scrisă de mine şi nu a mai fost prezentată niciodată la o facultate sau instituţie de învăţământ superior din ţară sau străinătate.

Declar că toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare, ca referinţe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar şi în traducere proprie din altă limbă, sunt scrise între ghilimele şi fac referinţă la sursă. Reformularea în cuvinte proprii a textelor scrise de către alţi autori face referinţă la sursă. Înţeleg că plagiatul constituie infracţiune şi se sancţionează conform legilor în vigoare.

Declar că toate rezultatele simulărilor, experimentelor şi măsurătorilor pe care le prezint ca fiind făcute de mine, precum şi metodele prin care au fost obţinute, sunt reale şi provin din respectivele simulări, experimente şi măsurători. Înţeleg că falsificarea datelor şi rezultatelor constituie fraudă şi se sancţionează conform regulamentelor în vigoare.

București, *Iulie.2015* Absolventă Oana - Teodora GRIGORE

Cuprins

Listă figuri

Listă acronime

ADC	Analog To Digital Convertor – Convertor Analog Digital
ALU	Arithmetical Logical Unit – Unitatea Operațiilor Aritmetico-Logice
AM	Amplitude Modulation – Modulație În Amplitudine
CISC	Complex Instruction Set Computers – Calculatoare Cu Set Complex De Instrucțiuni
CPU	Central Processing Unit – Unitate Centrală De Procesare
CRT	Catode Ray Tube – Monitor Cu Tub Catodic
DAC	Convertoare Digital – Analogice – Convertor Digital Analogic
DIP	Dual In-Line Package - Circuit Integrat Dreptunghiular cu două rânduri de pini
EDGE	Enhanced Data Rates For GSM Evolution - Rate De Date Crescute Pentru Evoluția GSM
FM	Frequency Modulation – Modulație În Frecvență
FSK	Frequency Shift Keying - Manipulare cu Deplasare în Frecvență
FSR	Force Sensitive Resistor – Senzor Rezistiv De Presiune
GPIO	General Purpose Input/Output – Conectare Generală Intrare/Ieșire
GPRS	General Packet Radio Service - Serviciu De Pachete Comutate Pentru Comunicații Mobile De Date
GPS	Global Positioning System – Sistem Global De Localizare
GSM	Global System For Mobile Communications – Sistem Global Pentru Comunicațiile Mobile
HSDPA	High-Speed Downlink Packet Access - Legatură de descărcare de date de mare viteză
HSUPA	High Speed Uplink Packet Access - Legatură de încărcare de date de mare viteză
I2C	Inter-Integrated Circuit – Interfață serială de date între circuite integrate digitale
IMEI	International Mobile Equipment Identity –Identificarea Internațională A Echipamentelor Mobile
LCD	Liquid Crystal Display – Monitor Cu Cristale Lichide
MMS	Multimedia Messaging Service – Serviciul De Mesaje Multimedia (Video, Imagine, Audio)
NTC	Negative Temperature Coefficient – Coeficient Negativ De Temperatură
PIC	Programmable Intelligent Computer – Calculator programabil inteligent
PTC	Possitive Temperature Coefficient – Coeficient Pozitiv De Temperatură

RAM	Random Acces Memory – Memoria De Acces Aleator
RISC	Reduced Instruction Set Computers – Calculatoare Cu Set Redus De Instrucțiuni
ROM	Read Only Memory – Memoria De Citire
RTD	Resistance Temperature Detector – Detector Rezitiv De Temperatură
SMD	Surface-Mount Technology - Tehnologie de plantare a componentelor electronice direct pe cablajul imprimat
SMS	Short Message Service – Serviciul De Mesaje Scurt (Text)
SPI	Serial Peripheral Interface – Interfață Periferică Serială
SSB	Single Sideband – O Singură Bandă
TWI	Two-Wire Serial Interface – Interfață Serială Prin Două Fire
UART	Universal Asynchronous Receiver-Transmitter - Transmițător/Receptor Asincron Universal
UMTS	Universal Mobile Telecommunications System – Sistem Universal De Telecomunicații Mobile
USB	Universal Serial Bus – Magistrală Serială Universlă
WCDMA	Wideband Code Division Multiple Access - Acces Multiplu Prin Divizare De Cod De Bandă Largă

Introducere

În această lucrare am descris modul de transmitere al unor date preluate de la niște dispozitive periferice de intrare conectate la microcontroler, cum ar fi senzori de presiune și senzori de temperatură, și prelucrate de către un microcontroler. Datele preluate de la aceste dispozitive periferice de intrare vor fi transmise cu ajutorul unui modem GSM, care are încorporată o cartelă SIM, către niște dispozitive periferice de ieșire. Rezultatele vor fi afișate pe ecranul unui telefon mobil și pe display-ul sistemului construit.

În acest moment pe piață nu există un asemenea produs. Este o idee proprie ce folosește cunoștințe dobândite pe parcursul acestor patru de ani de facultate. Materiile care m-au ajutat în realizarea acestui proiect au fost următoarele: Arhitectura microprocesoarelor, Microcontrolere, Dispozitive electronice, Circuite electronice fundamentale și Semnale și Sisteme.

Sistemul proiectat reprezintă rezultatul unei munci depuse având fonduri reduse și pe o perioadă limitată de timp. Bineînteles că procesul de realizare a proiectului nu a fost lipsit de probleme și de obstacole care și-au găsit rezolvarea în multe ore de muncă pentru a găsi soluția optimă.

Am ales această temă pentru proiectul de licență doarece are un impact important pe plan personal. În lucrarea de față este prezentată doar o machetă a produsului final. Acest prototip îndeplinește toate funcțiile stabilite inițial, doar că nu are o formă finală ce ar putea fi utilizată în producție.

Obiectivele acestui proiect de licență sunt:

- realizarea unui cântar electronic inteligent, capabil să transmită periodic (cu ajutorul unui modem GSM) mesaje SMS programabile precizând greutatea măsurată, diferențele de greutate in timp, praguri de greutate;
- interfațarea unui microcontroler cu anumite periferice, senzor de temperatură, senzor de presiune
- afișarea datelor colectate de la aceste periferice pe un display
- sintetizarea informației în ceea ce privește materialele folosite pentru construirea unui astfel de sistem;
- explicații privind rezultatele finale obținute;
- conectarea unui modem GPS pentru monitorizarea locației sistemului;

- dezvoltarea unei aplicații, cu rolul unei baze de date, unde se vor stoca datele culese de la perifericele conectate la microcontroler;

- adăugarea unui senzor de umiditate;

- montarea unui panou solar pentru încărcarea acumulatorului.

Structura lucrării :

Această lucrare de licență își propune realizarea unui sistem inteligent de monitorizare a greutății unor stupi de albine folosind cât mai mult din cunoștințele obținute pe parcursul perioadei universitare.

Prezenta lucrare este structurată în patru capitole, iar la final sunt prezentate concluziile scoase din realizarea proiectului de licență.

Primul capitol va cuprinde o prezentarea amplă a teoriei microcontrolerelor, domeniul în care pot fi folosite, utilitatea lor în mediul de dezvoltare.

Capitolul doi cuprinde o descriere a dispozitivelor periferice de intrare ale unui microcontroler. În special vor fi aduși in discuție senzorii de presiune/greutate, senzorii de temperatură.

Capitolul trei prezintă o scurtă introducere în ceea ce privește modul în care sunt transmise informațiile prin intermediul unui modem GSM. Voi prezenta tot în acest capitol și o descriere a dispozitivelor periferice ce pot fi conectate la un microcontroler, ca exemplu display-ul.

Capitolul patru prezintă atât partea software cât și partea hardware a acestui proiect. Vor fi detaliate etapele realizării proiectului, va fi prezentată funcționalitatea sistemului.

Ultima parte a proiectului va fi un sumar al rezultatelor obținute și oferă o concluzie finală asupra întregului proiect de licență.

CAPITOLUL 1
SISTEME AUTOMATE
CU MICROCONTROLERE

1.1 NOȚIUNI GENERALE

În prezent, atât în domeniul industrial cât și pentru uzul general, se utilizează din ce în ce mai frecvent diverse dispozitive electrice si electronice, cum ar fi pentru regulatoare automate de putere, telecomenzi, faxuri, cuptoare cu microunde, mașini automate de spălat, motoare, automobile.

Automatizarea este necesară pentru a facilita procesele sau mecanismele de operare și control. Stocarea datelor și procesarea reprezintă un segment important al oricărui sistem de control automat. Astfel apare nevoia de un dispozitiv special, așa-numitul „**microcontroler**”, care permite controlul timpului și secvențierea mașinăriilor și proceselor.

Mai mult, cu ajutorul microcontrolerului, este posibilă realizarea operațiilor aritmetice și logice simple. Orice sistem care este controlat de la distanță deține implicit și un microcontroler. Microcontrolerele sunt microcalculatoare integrate pe un singur chip, utilizate în controlul și automatizarea mașinilor și a proceselor.

Microcontrolere au o structură internă formată din :

- Unitatea centrală de procesare (CPU - Central Processing Unit);
- Memorie volatilă (RAM - Random – acces memory) – folosită pentru stocarea datelor;
- Memorie flash (ROM – Read - only memory) – folosită pentru programe și pentru operațiile cu parametrii;
- Porturi de intrare și ieșire;
- Convertoare analog – digitale, sau convertoare digital –analogice;
- Periferice precum numărătoare;

Această structură internă poate varia în funcție de dispozitivul folosit. Toate aceste blocuri funcționale se află pe un singur circuit integrat, ceea ce duce la o reducere a dimesiunii plăcii de control, a puterii de consum și la o mărire a fiabilității și la o ușurință a integrării în aplicații.

Utilizarea microcontrolerelor nu reduce doar costul automatizării, dar oferă și o flexibilitate mai mare. Dipozitivul putând fi programat astfel încât să funcționeze ca un sistem inteligent, acest lucru fiind realizabil datorită datelor ce pot fi procesate și datorită existenței memoriei.

1.1.1 CPU (Central Processing Unit- Unitatea Centrală de procesare)

Așa cum este evidențiat și prin nume – CPU – reprezintă un bloc ce monitorizează și controlează procesele microcontrolerului fără ca utilizatorul să aflecteze acest mecanism.

Este alcătuit din :

- Decodor de instrucțiuni – partea electronică ce se ocupă de recunoașterea instrucțiunilor de program și de rularea circuitelor de bază;

- ALU (Arithmetical Logical Unit – Unitatea aritmetico- logică) – se ocupă de toate operațiile matematice și logice cu date;

- Acumulator – se ocupă cu stocarea datelor asupra cărora se vor executa anumite operații, și mai stochează și rezultate ce urmează a fi folosite în procesele următoare.

1.1.2 RAM (Random Acces Memory - Memoria volatilă)

Este un tip de memorie folosit pentru stocarea temporală a datelor și a rezultatelor intermediare create și utilizate în timpul operațiilor ce au loc în microcontroler. Conținutul acestei memorii este șters imediat ce se oprește alimentarea.

1.1.3 ROM(Read Only Memory – Memoria flash)

Este un tip de memorie folosit pentru a salva permanent programul ce urmează a fi executat. Dimensiunea programului depinde de dimensiunea memoriei. Această memorie poate fi o parte a microcontrolerului sau poate fi ca un chip extern, depinde de modelul microcontrolerului folosit.

Un microcontroler ce are memoria ROM internă, are o memorie de o dimensiune relativ mică și este destul de scump, dar lasă mai mulți pini liberi pentru a conecta cât mai multe periferice. În cazul în care microcontrolerul are o memorie ROM externă, acesta este mai ieftin având un program mai lung, dar lasă mai puțini pini liberi pentru a conecta periferice.

1.1.4 Porturi de intrare/ieșire

Pentru ca un microcontroler să fie cât mai performant este necesar să se poată conecta la acesta cât mai multe componete periferice. Fiecare microcontroler are unul sau mai multe porturi conectate la pinii săi. Se numesc porturi intrare/ieșire deoarece acești pini își pot modifica funcția în concordanță cu ceea ce își dorește utilizatorul.

1.1.5 Numărător

Este ca un motor ce rulează un program și reține adresa din memorie ce conține următoarele instrucțiune ce urmează a fi executate. După executarea fiecărei instrucțiuni, valoarea numărătorului crește cu o unitate. De aceea programul execută câte o instrucțiune pe tact așa cum este și scrisă.

Valoarea numărătorului poate fi modificată în orice moment provocând un slat în memorie la o nouă locație.

1.1.6 Convertoare analog – digitale (ADC)/Convertoare digital – analogice (DAC)

Multe sisteme integrate citesc informații provenite de la senzori ce produc semnale analogice. Acesta este scopul convertorului analog digital. Procesoarele sunt construite să interpreteze și să proceseze date digitale, adică un cod binar compus din zerouri si unuri, ele nu sunt capabile să interpreteze un semnal analogic provenit de la un dispozitiv, de exemplu un senzor.

Așadar convertoarele analogic - digitale sunt folosite pentru a converti datele de intrare provenite de la un sistem în semnale ce pot fi recunoscute de microcontroler. O proprietate mai puțin comună pentru un microcontroler constă în prezența unui convertor digital- analogic ce permite ca la ieșirea microcontrolerului să fie un semnal analogic.

1.2 DIPOZITIVE INTEGRATE ȘI DISPOZITIVE CU MEMORIE EXTRENĂ

Microcontrolerul poate fi considerat un sistem autonom cu procesor, memorie și periferice ce pot fi utilizate ca un sistem integrat. Majoritatea microcontrolerelor folosite în ziua de azi sunt integrate în alte dispozitive, cum ar fi, automobile, telefoane mobile, aparate și periferice pentru calculator.

În timp ce unele sisteme integrate sunt foarte complexe, multe dintre ele au cerințe minime pentru memorie și pentru lungimea programului, fără necesitatea un sistem de operare, iar din punct de vedere software având o complexitate foarte redusă.

In mod uzual porturile de intrare și de ieșire sunt compuse din comutatoare, LED-uri, display-uri, relee, senzori de temperatură, umiditate, greutate, aparate radio etc. Aceste sisteme integrate nu au o tastatură, un ecran, o imprimantă sau orice alte periferice de intrare și ieșire ca la un calculator, rezultând faptul că interacțiunea unamă lipsește.

Exită însă dispozitive ce au atât memorie internă cât și memorie externă. O parte din program poate fi executată de memoria internă, iar restul de memoria externă. Există și o opțiune ca programul sa fie executat în intregime de memoria externă. Această opțiune este folositoare atunci când cerințele memoriei de program sunt mai mari decât cele disponibile pe chip.

1.3 ARHITECTURA CISC ȘI RISC

CISC (Complex Instruction Set Computers - Calculatoate cu set complex de instrucțiuni) și RISC (Reduced Instruction Set Computers - Calculatoare cu set redus de instrucțiuni) sunt două arhitecturi cunoscute atunci când vorbim despre microcontrolere și microprocesoare.

Procesoarele CISC au un set complex de instrucțiuni. Un set mare de instrucțiuni ajută programatorii ce utilizează limajul de asamblare oferindu-le o felxibiliate mare pentru a putea scrie programe scurte cât mai eficiente.

Scopul arhitecturii CISC este acela de a scrie programe care sa aibe cât mai puține linii folosind limbajul de asamblare. Acest lucru este posibil având la dispoziție compomente hardware ce pot interpreta și procesa un anumit număr de operații.

Construirea unor instrucțiuni complexe, al câror rezultat este procesat direct de partea hardware este benefică din două puncte de vedere. Nu doar că implementarea hardware este mai rapidă, dar de asemenea salvează și spațiu în memoria de program în sensul că avem un cod al instrucțiunii mult mai scurt.

Programtorii își doreau să folosească instrucțiuni cât mai simple, mai scrute și mai puține în schimbul instrucțiunilor lungi, complexe și în număr mare oferite de arhitectura CISC. Pentru a îndeplini această cerință a fost nevoie de scriea mai multor linii de cod de o complexitate mult mai redusă.

Astfel, un avantaj al folosiri arhitecturii RISC este acela că deși necesită scriera mai multor insctrucțiuni de o complexitate redusă, este nevoie de mai puțină muncă din punct de vedere hardware ceea ce îl face să aibă un cost de producție mult mai mic și să fie mai ușor de implementat.

Întotdeauna va fi mai ușor sa compilezi un program ce conține un număr ridicat de instrucțiuni, dar de o compleximate mai mică decât un program ce conține instrucțiuni de o complexitate mare.

Bineînteles că atunci când ai două tipuri de arhitecturi, întrebarea este care dintre ele este mai bună. În cazul în care se face o comparație între CISC și RISC, răspunsul depinde de ceea ce ai nevoie. Dacă soluția aleasă implică instrucțiuni complexe și modul de adresare este ca cel folosit de arhitectura CISC, atunci alegrea va fi arhitectura CISC. Dacă programul necesită instrucțiuni simple și un mod de adresare simplu, atunci este mult mai bine să ne axăm pe arhitectura RISC. Alegerea uneia dintre arhitecturi depinde de mai mulți factori, de aceea este nevoie să se știe în primul rând complexitatea proiectului pentru a putea utiliza arhitectura cea mai eficientă.

1.4 ARHITECTURA HARVARD ȘI VON NEUMANN

Exită două clase majoritate în ceea ce privește arhitectura calculatoarelor, având numele, „Arhitectura Harvard” și „Arhitectura Von Neumann”

1.4.1 Arhitectura Harvard

Arhitectura Harvard are atât memorie de program cât și memorie de date, având magistrala de date independentă de magistrala de adrese. Datorită faptului că există două fluxuri separate, unul de date și celălalt de adrese, nu mai este nevoie de un multiplexor ce divizează timpul între cele două magistrale. Nu numai ca arhitectura suportă magistrale paralele pentru date și adrese, dar permite de asemenea o organizare internă diferită, cum ar fi faptul că instrucțiunile sunt aduse și decodate în timp ce se lucrează cu mai multe date simultan. În plus, magistrala de date poate avea dimensiune diferită față de magistrala de adrese. Acest lucru permite diferite lățimi optime pentru cele două magistrale astfel încât execuția instrucțiunilor să fie mai rapidă.

1.4.2 Arhitectura Von Neumann

La arhitectura Von Neumann, programele și datele împart aceeași memorie. Această arhitectură permite stocarea și modificarea programelor mai ușor. În orice caz, implementarea codului cu această

arhitectură nu este una optimă deoarece necesită multe apelări din memorie pentru a forma o instrucțiune. Aducerea din memorie a datelor și a programelor este facută cu ajutorul unui multiplexor al perioadei ceea ce afecteză performanțele.

Figura 1.4.1.1 reprezintă schema bloc a arhitecturii Harvard, având evidențiate cele două magistrale, cea de date și cea de adrese. În figura 1.4.2.1 este prezentată schema bloc a arhitecturii Von Neumann, ce are evidențiată o singură magistrală ce conține atât adrese cât și date.

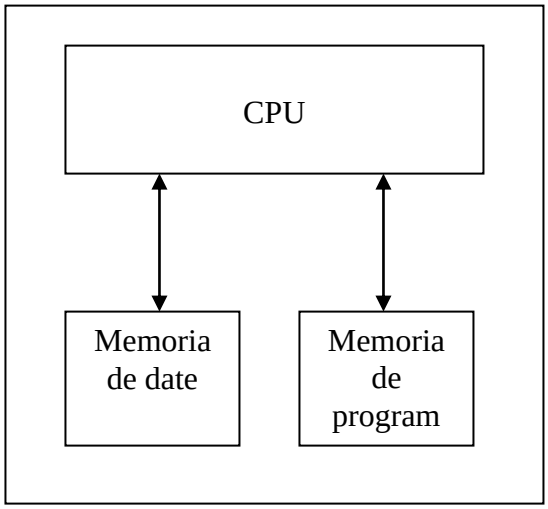

Figura 1.4.1.1 Arhitectura Harvard

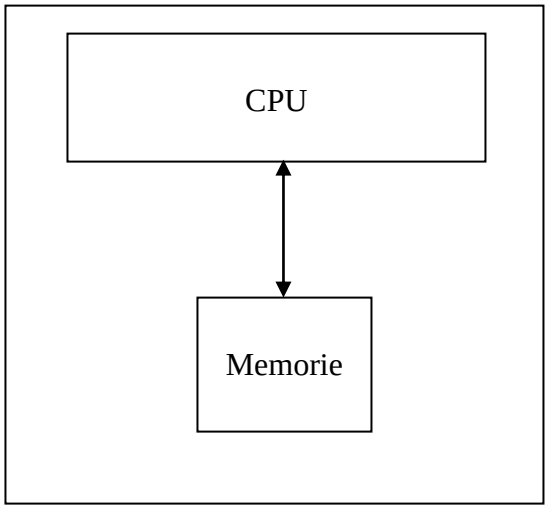

Figura 1.4.2.1 Arhitectura Von Neumann

1.5 MEDII DE DEZVOLTARE

Microcontrolerele la început se programau folosind un limbaj de asamblare. Acum sunt folosite și limbaje de programare de nivel înalt pentru programarea microcontrolerelor. Unele limbaje sunt create special pentru acest scop , de exemplu limbajul de programare C.

Compilatoarele pentru aceste limbaje sunt concepute pentru a susține cât mai bine unicitatea fiecărui microcontroler. Unele microcontrolere au medii de dezvoltare ce ajută la implementarea anumitor tipuri de aplicații.

Producătorii de microcontrolere dezvoltă programe specifice tipului de microcontroler vândut pentru a fi mult mai ușor de accesat și pentru o adaptare mult mai rapidă în ceea ce privește partea hardware.

Multe dintre microcontrolere sunt atât de sofisticate încât necesită propriile derivări de limbaj de la limbajul standard de programare C. Acest lucru împiedică utilizarea aplicațiilor standard (cum ar fi librării de cod sau instrumente de analiză statică) chiar și pentru partea de cod care are legătură cu proprietățile specifice acelei părți hardware.

Interpretoarele sunt în general folosite pentru a ascunde aceste inconveniente. Există și simulatoare care ajută la programarea microcontrolerelor. Acestea îi ofera programatorul un mediu de

simulare al comportamentului microcontrolerului în mod virtual în condiții ce imită mediul real. Un simulator va arăta starea internă a procesorului și a ieșirilor, și în același timp va permite și simularea generării intrărilor.

Pe de altă parte simulatoarele au o limitare în a simula comportamnetul tuturor componentelor hardware, ele putând reproduce anumite condiții ce în realitate ar fi greu de implementat din punct de vedere fizic. Folosirea simulatoarelor este o metodă mult mai rapidă pentru a depana și analiza problemele ce pot aparea în programarea microcontrolerului.

1.6 Familii reprezentative de microcontrolere

O să amintesc câteva dintre cele mai reprezentative companii ce produc microcontrolere. Printre ele se numără : ARM, Intel, STMicroelectronics, Silicon Laboratories, Microchip, Freescale, Infineon, Atmel.

1.6.1 Microcontrolere PIC

PIC 16CXX și PIC17 CXX sunt microcontrolere pe 8 biți proiectate de Microchip și utilizează thenologia CMOS. Microcontrolerele PIC sunt recunsocute pentru perfromanța ridicată, costul redus și dimensiune mică. Utilizează arhitectura RISC ce are o viteză mare de procesare. PIC 16CXX are doar 33 de instrucțiuni de un cuvânt. Frecvența tipică operațională pentru 16CXX variază până la 20MHz. Se poate adăuga și o memorie de program externă, până la 64K cuvinte.

PIC 17C42 are un număr de timere și de intrări/ieșiri ce pot fi modelate după necesități. 16C71 are un ADC pe 8 biți încorporat cu 4 canale. Un ADC pe 10 biți cu 12 canale se regăsește in structură microcontrolerului 17C752.

Există multe modele de microcontrolere proiectate de Microchip sub multe variante. În general ele conțin următoarele component: timer, timer watchdog, ADC, memorie extinsă de date/ instrucțiuni, o comunicație serială, PWM (Pulse Width Modulation – modularea impulsurilor în lăţime) și memorie ROM.

Dispozitiv	Pini (DIP)	Intrări/Ieșiri	Canale ADC
16C56	18	12	-
16C57	28	20	-
17C44	40	33	-
16C71	18	13	4(pe 8 biți)
17C752	40	33	12(pe 10 biți

Tabel 1.6.1.1 Modele de microcontrolere PIC

PIC – Programmable Intelligent Computer – Calculatoare Inteligente Programabile – aparțin familiei de arhitecturi Harvard. În anul 2013 compania a produs peste 12 milarde de componente utilizate în diferite sisteme.

Din punct de vedere hardware, dispozitivele PIC au un interval intre 8 pini DIP (dual in-line package) și 100 de pini SMD (Surface-mount technology), cu intrări / ieșiri discrete, module ADC/DAC și porturi de comunicație UART, I2C, chiar și USB. Ca mediu de dezvoltare se folosește MPLAB, limbaj de asamblare si limbaj de programare C/ C++.

PIC-urile sunt populare prin prisma a două industrii , una cea a dezvoltatorilor și cealaltă cea a pasionaților de thenologie, datorită costului redus, a disponibilității, a bazei mari de utilizatori, a comunicației seriale și a abilității memoriei flash de a fi reprogramabilă.

Arhitectura PIC este caracterizată prin mai multe atribute:

- magistrală separată de date și adrese (Arhitectura Harvard)
- nu număr mic de instrucțiuni de dimensiune fixă
- multe instrucțiuni au un singur tact
- un accumulator , utilitatea fiindu-i implementată
- toate locațiile de memorie RAM, funcționează ca un registru
- o stivă unde sunt stocate adresele de întoarcere
- o cantitate mică de spațiu de date adresabile
- spațiul de date mapat în CPU, porturi și register periferice
- fanioanele de stare ALU sunt mapate în spațiul de date
- numărătorul este implementat și el în spațiul de date și este disponibil pentru scriere (se folosește pentru salturi indirecte)

1.6.2 Microcontrolere ARM

ARM este un microcontroler pe 32 de biți al cărui procesor este proiectat de „ARM Limited” și este oferit altor organizații de prelucrare pentru a adăuga perifericele necesare, pentru a fi fabricat și vândut. Multe modele de microcontrolere ARM sunt disponibile, dar au suferit schimbări substanțiale din momentul când au fost construite, începutul fiind prin 1980. Cum ele sunt foarte eficiente în ceea ce privește consumul de putere lucrând la o putere scăzută, sunt folosite pentru noile tehnologii în domeniul comunicațiilor mobile. Procesoarele ARM sunt folosite în multe sisteme integrate, cum ar fi iPOD, componente pentru jocuri. Având o arhitectură pe 32 de biți, ARM-urile oferă numeroase avantaje pentru sistemele integrate.

O caracteristică importantă a arhitecturii ARM o reprezintă un circuit digital ce cu un număr specificat de biți poate deplasa o dată/informație într-un singur tact de ceas. Acest circuit poate procesa anumite operații înainte de a intra în ALU. Acest lucru permite calcule ușoare pe intervale mai lungi cu expresii și adrese.

Blocul de registre conține un vector cu registre de 32 de biți ce stochează valori cu semn și valori fără semn. Numerele cu semn pe 8 și pe 16 biți sunt convertite în echivalentul lor pe 32 de biți înainte de a fi încărcate în registru prin conversia hardware. ARM adoptă o arhitectură registru la registru și de aceea nu există acumulator. Incrementatorul mărește adresa registrului înainte ca procesorul să scrie și să citească următorul set de instrucțiuni.

Așa cum am spus și la începutul acestui subcapitol ARM Limited proiecta doar procesorul ARM, la care se adăugau diferite periferice în funcție de compania care îl cumpăra, cum ar fi Atmel, Samsung și Philips. Aceste companii au creat diferite versiuni ale procesorului ARM în funcție de ce aveau nevoie.

Complexitatea procesorului ARM poate fi înțeleasă dacă ne axam pe caracteristicile sale principale. De exemplu Samsung S3C4510B, cu procesorul SRM7TDMI, are următoatele proprietăți:

- o fecvență operațională de până la 50 MHz

- o tensiune de 3,3 V +/- 0,05

- o arhitectură RISC pe 16/32 biți

- sisteme integrate pentru aplicații

- magistrală externă de 8/16/32 biți ce suportă de la ROM, memorie flash, RAM, și periferice de intrare/ieșire

- pini configurați individual intrare, ieșire sau intrare/ieșire pentru semnale dedicate

1.6.3 Microcontroler Atmel - AVR

Famila AVR de la Atmel este formată din microcontrolere cu arhitectură Harvard pe 8 biţi şi set redus de instrucţiuni (RISC). Arhitectura de bază AVR a fost concepută de doi studenţi de la Norwegian Institute of Technology (NTH) Alf-Egil Bogen şi Vegard Wollan. Ele au fost introduse pe piaţă în 1996.

AVR-urile sunt clasificate în patru mari categorii:

- **tinyAVR** - 1-8 kB memorie de program, capsulă de 8 până la 32 pini, set limitat de periferice;
- **megaAVR** - 4-256 kB memorie de program, capsulă de 28 până la 100 de pini, set extins de instrucţiuni (instrucţiuni pentru înmulţire şi adresare indirectă), set extins de periferice;
- **XMEGA** - 16-256 kB memorie de program, capsulă de 44 până la 100 de pini, interfeţe performante extinse şi suport pentru criptografie, set extins de periferice;
- **Application Specific AVR** - megaAVR cu funcţii speciale, care nu sunt prezente la familia AVR, cum ar fi controler de LCD, controler USB etc, FPSLIC (Field Programmable System Level Integrated Circuit), un core AVR integrat cu un FPGA.

AVR-urile au o unitate de execuţie în bandă de asamblare cu două niveluri, acest lucru permiţând ca următoarea instrucţiune să fie adusă din memorie (fetch) în timp ce instrucţiunea curentă este în execuţie. Majoritatea instrucţiunilor se execută într-un singur ciclu de instrucţiune. Un avantaj faţă de celelalte familii concurente de microcontrolere îl constituie faptul că arhitectura AVR este optimizată pentru execuţia de cod C compilat.

Un exemplu de microncontroler din aceasta famile ar fi ATmega324. ATmega324 reprezintă un microcontroler cu o arhitectură pe 8 biţi. Ca urmare, registrele şi magistrala internă de date sunt pe 8 biţi. Totuşi, în codul C se pot folosi variabile întregi pe 32 de biţi şi chiar în virgulă mobilă. Compilatorul este cel care se ocupă de translatarea instrucţiunilor cu variabile pe 32 de biţi în cod asamblare care lucrează pe 8 biţi.

Caracteristicile ale procesorului Atmega324A:

- 32Kb Flash, sau dimensiunea maximă a programului care poate fi scris in microcontroler
- 1Kb EEPROM
- 2Kb RAM
- frecvența maximă de lucru de 20 MHz
- tensiune de alimentare între 1.8 și 5.5 V
- 6 canale PWM
- 8 canale de ADC, precizie de 10 biți, situate pe portul A
- 3 porturi de I/O digitale, porturile B, C, D
- 4 porturi de I/O, fiecare cu 8 pini, pentru un total de 32 de pini de I/O
- 3 timere - două pe 8 biți si unul pe 16 biți
- interfețe seriale: USART, SPI(Serial Peripheral Interface), TWI (two-wire serial interface)
- interfaţă JTAG.

CAPITOLUL 2
DIPOZITIVE PERIFERICE DE INTRARE CONECTATE LA MICROCONTROLER

Un periferic reprezintă orice dispozitiv, intern sau extern, care se conectează la un sistem de calcul și îi extinde funcționalitatea de bază. În cazul unui microcontroler, există o serie de astfel de periferice incluse direct pe chip. Deși nu par să aibe același impact ca perifericele unui calculator, cum ar fi monitor, placă grafică, imprimantă, tastatură, mouse, fără ele microcontroler-ul ar fi doar o cutie care poate număra și face operații ALU. Mai mult, perifericele interne ne ajută să conectăm alte dispozitive mai performante la controler și să îi putem oferi funcționalități asemănătoare unui sistem, de exemplu conexiune la internet, linie de date USB, display grafic.

Dispozitivele I/O implementează funcţii speciale degrevând unitatea centrală de toate aspectele specifice de comandă şi control în funcţia respectivă. Există o varietate mare de dispozitive I/O; dispozitivele I/O conduc operaţii generale de comunicaţie (transfer serial sau paralel de date), funcţii generale de timp (numărare de evenimente, generare de impulsuri), operaţii de conversie analog/numerică, funcţii de protecţie, funcţii speciale de comandă, şi enumerarea poate continua. Din această mare varietate, parte din dispozitive se găsesc în configuraţia tuturor microcontrolerelor, iar o altă parte de dispozitive o regăsim doar în microcontrolerele construite pentru a optimiza aplicaţii cu un grad mare de particularitate.

2.1 SENZORI

Senzorul este un dispozitiv tehnic care reacționează calitativ sau cantitativ prin propriile mărimi măsurabile, la anumite proprietăți fizice sau chimice ale mediului din preajma lui. Ca parte componentă a unui aparat sau sistem tehnic detector poate măsura/înregistra de exemplu presiunea, umiditatea, temperatura, câmpul magnetic, accelerația, forța, intensitatea sonoră, radiații.

Senzorul este un dispozitiv care măsoară o mărime fizică (masă, presiune, temperatură, umiditate) și o transformă într-un semnal electic sau optic care poate fi citit de către un observator printr-un instrument sau poate fi prelucrat.

Senzorii (traductoarele) au rolul de a transforma anumiţi parametri ai sistemului în mărimi de altă natură. Parametrul de transformat formează semnalul de intrare al traductorului, iar cel transformat semnal de ieşire.

Alegerea şi aprecierea unui anumit tip de senzor are la bază o serie întreagă de parametri dintre cei mai diferiţi, cum ar fi: dimensiunile, greutatea, costul, gradul de protecţie electrică, domeniul de măsurare, consumul de energie, natura semnalelor de ieşire şi complexitatea lanţului de prelucrare a acestora, sensibilitatea, rezoluţia, precizia.

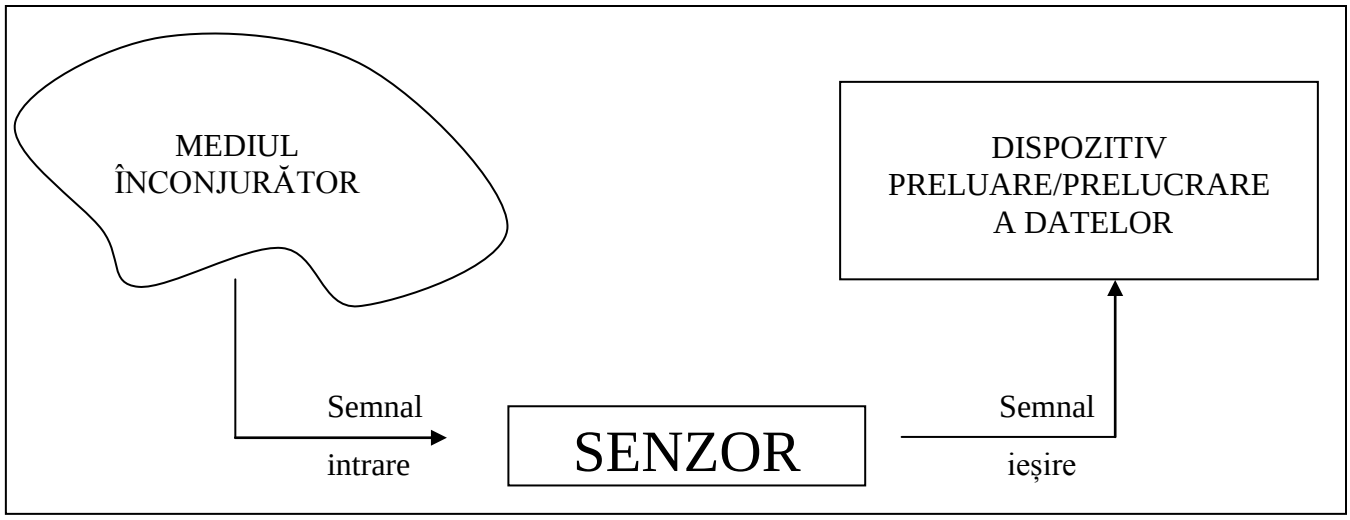

Figura 2.1.1. Recepția unui semnal cu ajutorul unui senzor

2.2. SENZOR DE TEMPERATURĂ

Temperatura este o variabilă a mediului ce este cuantificată cel mai des. Sistemele electronice, fizice, chimice, biologice şi mecanice sunt afectate de temperatură. Reacțiile chimice, procesele biologice și circuitele electrice funcționează mai bine având un domeniu limitat de temperatură.

Temperatura poate fii masurată cu ajutorul unor senzori. Acești senzori pot fii de mai multe tipuri, dar ca principiu, toți determină temperatura prin observarea unor modificări la nivelul caracteristicilor fizice.

Cele mai cunoscute tipuri de senzori de temperatură sunt:

- Termocuplul
- Detector cu rezistență termică
- Termistor
- Senzor digital de temperatură

2.2.1 Termocuplul

Termocuplutrile sunt cele mai cunoscute, convenabile şi versatile dispozitive utilizate pentru măsurarea temperaturii. Ele convertesc unități de căldură în unități electrice utilizabile ce servesc ca semnale de intrare într-un proces de măsurare.

Termocuplurile sunt alcătuite din două metale diferite unite la unul din capete, de obicei se folosesc fire. Locul unde se unesc cele două metale se numește punct cald, iar celălalt capăt se numește punct rece.

Principiul de funcționare al termocuplului este relativ simplu. Când cele două metale sunt conectate împreună, apare un mic voltaj numit tensiunea joncțiunii termice.

Dacă temperatura joncțiunii se modifică, atunci apare o modificare a tensiunii, ce poate fi măsurată la intrarea unui circuit al unui controler electronic. Ieșirea este o tensiune proporțională cu diferența de temperatură dintre punctul rece și punctul cald. Combinând aceste două efecte se poate măsura temperatura.

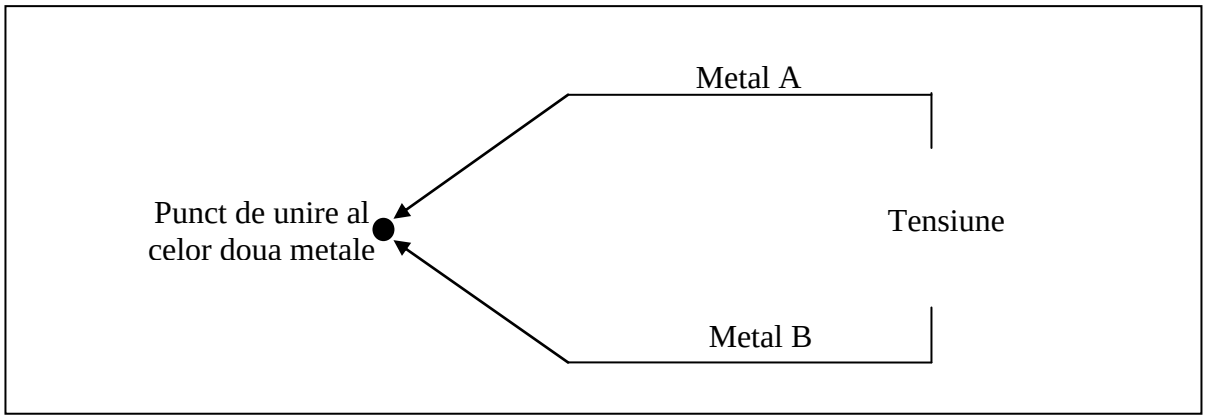

Figura 2.2.1.1 Schema circuitului unui termocuplu

Principalele tipuri de termocupluri și caracteristicile acestora se gasesc în tabelul de mai jos:

Electrodul pozitiv	Electrodul negativ	Domeniu de temperatură[°C]	Sensibilitatea medie [μV/°C]	Notație standard
70% Pt – Rh	94% Pt – Rh	0 … 1700	7	B
90% Pt – Rh	Pt	0 … 1500	10	S
87% Pt – Rh	Pt	0 … 1500	11	R
Cromel	Alumel	150 … 1200	39	K
Cu	Constantan	-150 … 350	44	T
Fe	Constantan	-150 … 700	53	J
Cromel	Constantan	0 … 950	76	E

Tabel 2.2.1.1 Ptincipalele tripuri de termocupluri

Avantajele termocuplurilor sunt următoarele:

- gamă mare de temperaturi (-190 … +1820°C)
- rezistență la șocuri și vibrații
- dimensiuni reduse
- -timp mic de răspuns

2.2.2 Termorezistor

Termorezistorul sau RTD (Resistance Temperature Detectors) este un dispozitiv de detecție a temperaturii a cărui rezistivitate crește cu temperatura. Metalele tipice folosite la realizare termorezistoarelor sunt platina (-200…850 grade Celsius), nichelul (-60…+150°C) și cuprul (-50…+150°C). Platina este cel folosit material, datorită gamei mari de temperaturi, stabilității și rezistenței la agenți chimici de coroziune

Principiul de funcționare al acestui dispozitiv se bazează pe faptul că rezistența electrică a unui metal se schimbă în mod liniar și repetat o dată cu temperatura. RDT-urile au un coeficient de temperatură pozitiv(rezistența crește cu temperatura). Rezistența materialului la o temperatură de bază este proporțională cu lungimea elementului și invers proporțională de-a lungul arie secționate.

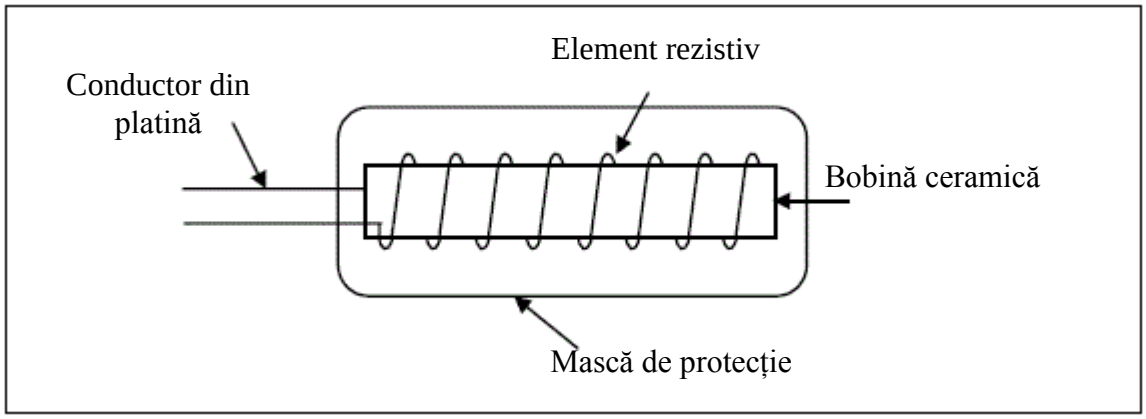

Figura 2.2.2.1 RDT (Detector cu rezistență termică)

Avantajele termorezistoarelor sunt următoarele:

- repetabilitatea și stabilitatea: termomentrul cu termorezistență de platină este folosit ca instrument standard
- sensibilitatea mai mare ca la termocuplu – termorezistoarele dau un răspuns mai liniar decât termocuplurile
- neliniaritățile pot fi corectate prin proiectarea corespunzătoare a schemei de măsurare
- flexibilitate
- folosesc fire de legătură de cupru și nu necesită compensări suplimentare.

2.2.3 Termistor

Termistorul este un element semiconductor de circuit care utilizează dependența rezistenţei electrice a unui semiconductor intrinsec de temperatura. Dispozitivul este realizat din materiale semiconductoare la care rezistivitatea scade repede cu temperatura, precum amestecuri de oxizi metalici (oxid de mangan, oxid de cupru, oxid de zinc, etc) care sunt măcinaţi şi apoi presaţi împreună cu un liant organic, iar apoi sinterizaţi.

Există doua tipuri principale de termistoare, unul având coeficient pozitiv de variație a rezistenței cu temperatura (PTC- Possitive Temperature Coefficient) și altul având coeficient negativ de variație a rezistenței cu temperatura (NTC- Negative Temperature Coefficient). Spre deosebire de metalele la care rezistența electrică crește cu temperatura, la termistori rezistența scade cu creșterea temperaturii lor.

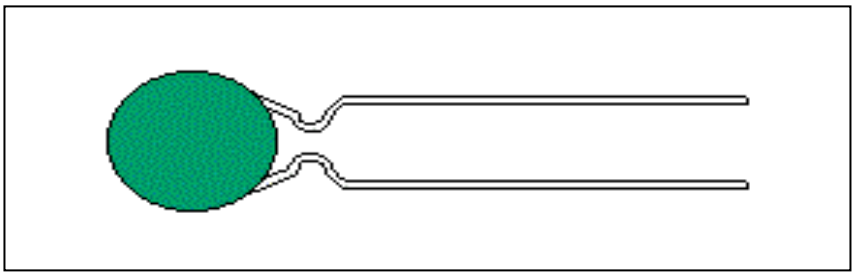

Figura 2.2.3.1 Termistor

Senbiliateta termistoarelor de tip PTC este foarte mare, dar domeniul de temperaturi este limitat(-100 … +400°C pentru cele din oxizi metalici și -150 … +150°C pentru cele din materiale semiconductoare).

Sensibilitatea termistoarelor de tip NTC este mică, iar gama temperaturilor de funcționare este cuprinsă în limitele -250 … +650 °C.

2.2.4. Senzor digital de temperatură

Senzori digitali de temperatură elimină necesitatea unor componente suplimentare, cum ar fi un convertor A / D, ne mai fiind nevoie de o calibrare a componentelor la o temperatură de referință specifică așa cum este necesar atunci când se utilizează termistoare. Senzori digitali de temperatură interfațează cu mediul înconjurător, încercând sa se ajungă la o îmbunătățire a funcționării sistemului de bază de monitorizare a temperaturii, făcându-l mai simplu.

Avantajul ultilizării unui senzor digital de temperatură se referă la precizia foarte bună a acestuia în ceea ce privește măsurarea temperaturii. Ieșirea oferită de acest senzor este una digitală. Nu mai este nevoie de nici o componetntă adițională, exemplul convertorul analog – digital, utilizarea lui fiind mai simplă decât cel mai simplu model de termistor, care asigură la ieșire o caracteristică neliniară a rezistenței cu variația temperaturii.

2.3 SENZOR DE PRESIUNE

Senzorul de presiune este o component a cărui rezitență se modifică atunci când o forță sau o presiune este aplicată asupra sa. Sunt cunoscuți sub numele de resistori sensibil la forță, având ca acronim în limba engleză FSR (Force Sensitive Rezistor).

Acest tip de sensor este alcătuit dintr-un polimer conductiv, care își schimbă rezistența intr-o manieră previzibilă în urma aplicării unei forțe pe suprafața sa. Ei sunt în mod usual furnizați ca o folie de polimer sau cerneală ce este aplicată prin serigrafie. Partea sensibilă, cea care detectează modificarea rezistenței, are in componență atât particule conductive cât și particule non-conductive, aranjate ca intr-o matrice. Aceste particule sunt de ordinul micrometrilor, având rolul de a reduce dependența cu temperatura, îmbunătățind proprietățile mecanice și crescând durabilitatea suprafeței.

Structura unui FRS constă din două membrane separate printr-un strat subțire de aer. Fanta de aer este menținută printr-un distanțier aflat jurul marginilor și prin rigiditatea celor două membrane.

Acești senzori sunt practic rezistori ce își schimbă valoarea rezistenței în funcție de cât de tare sunt apăsați. Costul lor este relativ mic și sunt foarte ușor de folosit, dar în ceea ce privește acuratețea aceștia sunt destul de inexacți. Valoarea acestora variază de la senzor la senzor, deci în cazul utilizării mai multor senzori FSR se va ajunge la o arie de valori.Deoarece acesti senzori FSR sunt rezistoare, ei sunt nepolarizați. Acest lucru înseamnă că pot fi conectați în circuit indiferent de polaritatea acestuia, ei funcționând corect în orice situație.

2.4 SENZOR DE UMIDITATE

Senzori de umiditate detectează umiditatea relativă a mediilor imediate în care sunt plasați. Ei măsoară atât umiditatea cât și temperatura în aer și exprimă umiditatea relativă ca un procent din raportul dintre umezeala din aer și cantitatea maximă care poate fi ținută în aer, la temperatura curentă. Pe măsură ce aerul devine mai fierbinte, deține mai multă umiditate, astfel încât umiditatea relativă se schimbă cu temperatura.

Majoritatea senzorilor de umiditate folosesc măsurarea capacitivă pentru a determina cantitatea de umiditate din aer. Acest tip de măsurătoare se bazează pe doi conductori electrici cu un film polimeric neconductor stabilit între ei, cu scopul de a crea un câmp electric. Umezeala din aer se colectează pe film și provoacă schimbări în nivelurile de tensiune între cele două plăci. Această schimbare este apoi transformată într-o măsurare digitală a umidității relative a aerului, după luarea în considerare a temperaturii aerului. Consumatorii privați de obicei folosesc senzori de umiditate atunci când suferă de alergii sau o boală respiratorie în care umiditate scăzută exacerbează. În schimb, umiditatea ridicată poate încuraja apariția ciupercilor sau dezvoltarea bacteriilor.

Senzorii de umiditate sunt de asemenea folosiți în crame pentru a menține aerul la un nivel de umiditate constant, cu scopul de a prelungi data de consum limită a vinurilor și a trabucurilor depozitate o perioadă îndelungată de timp. Muzee, spații de depozitare, precum și sisteme comerciale folosesc senzori pentru a asigura un nivel consistent de umiditate și calitatea aerului în clădiri. Automobilele folosesc acum senzori de umiditate, ca parte a sistemelor lor de dezaburire sau dezghețare pentru a regla automat cantitatea și tipul de aer utilizat pentru sistemul de aer condiționat al vehiculului. În cele din urmă, senzorii de umiditate sunt de asemenea utilizați în colectarea informațiilor despre vreme sau oceanografice.

CAPITOLUL 3
DISPOZITIVE PERIFERICE DE IEȘIRE CONECTATE LA MICROCONTROLER

3.1 Modul afișaj digital

O dată cu apariţia informaţiilor digitale, a apărut şi nevoie de afişare a acestora. Acest lucru a fost posibil folosind un monitor sau un afişaj digital ca cel folosit în sistemul prezentat, obţinut din configuraţia unui telefon mobil.

Monitoarele sau afişajele digitale sunt periferice de ieşire ce ajută la vizualizarea datelor digitale. În stadiul incipient aceste afişaje au fost formate dintr-o sucesiune de LED-uri a căror rată şi ordine de aprindere semnificau un anumit mesaj pentru operator. Cu timpul, datele ce trebuiau vizualizate au evoluat, cauzând o ncesitate de evoluţie şi în rândul dispozitivelor de afişaj. În această manieră au apărut monitoarele CRT (cu tub), monitoarele bazate pe plasmă, LCD-urile, ecranele de rezoluţie înalta 4k şi 5k, culminând cu o tehnologie în continuă dezvoltare, mai exact tehnologia monitoarelor 3D.

Afişajele CRT(Catode ray tube) funcţionează pe baza unui tub catodic format din unul sau mai mlte tunuri de electroni şi o suprafaţă fluorescentă utilizată în vizualizarea imaginilor. Aceste tunuri aruncă electroni către suprafaţa fluorescentă, care atunci când sunt în contact cu ea se aprind şi proiectează imaginea dorită. Fluxul de electroni este ghidat cu ajutorul câmpurilor magnetice din interiorul tubului.

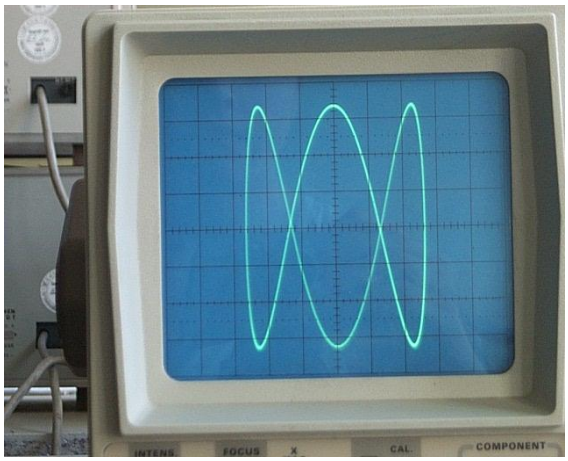

Figura 3.1.1 Figura Lissajous obtinuta cu ajutorul osciloscopului CRT

În cazul osciloscoapelor CRT este folosită deflecţia electrostatică şi nu cea magnetică. Raza este deflectată orizontal aplicând un câmp electric între cele două capete ale tubului, şi vertical aplicând câmp electric între suprafaţa superioară şi cea inferioară. Diferite fosforescente sunt disponibile în

funcţie de măsurătoarea dorită. Luminozitatea, culoarea şi persistenţa iluminării depind de tipul fosforescenţei folosite.

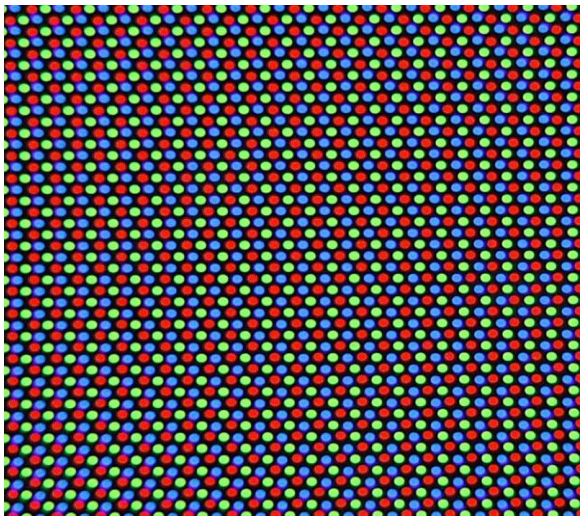

Figura 3.1.2 Configuraţia triunghiulară echilaterală a tunurilor de culoare

Tuburile color folosesc trei tipuri diferite de fosforescenţe ce emit variaţii de albastru, roşu şi verde ale luminii. Ele sunt grupate în pachete de dungi sau grupuri numite triade. CRT-urile color folosesc, câte un tun de electroni pentru fiecare culoare primară, aranjate fie în linie dreapta, fie în configuraţie triunghiulară echilaterală. O grilă de măşti ce conţine o masca de umbrire absoarbe electronii, ce sunt direcţionaţi prin orificii de dimensiune mică cu scopul de a ilumina corect suprafaţă fosforescentă. Orificiile au formă conică, lucru ce face ca electronii care intră şi lovesc suprafaţa dar nu sunt absorbiţi să fie reflectaţi prin acelaşi canal fără a influenţa alt punct din zona ecranului.

În cazul în care masca de umbrire devine magnetizată, câmpul ei magnetic deflectă razele de electroni ce trec prin ea, cauzând o distorsiune în puritatea culorii. Curbarea razelor prin mască rezultă la o deviere a acestora, făcând astfel ca eletronii să atingă zone de culoare greşite. Aşadar este important ca masca de umbrire să fie demagnetizată. Majoritatea monitoarelor CRT prezintă un curcuit de demagnetizare încorporat a cărui componentă principală este o bobină de demagnetizare montată în jurul ecranului. La alimentare circuitul produce un curent alternativ scurt prin interiorul bobine, care scade treptat în intensitate. Acest procedeu duce la formarea unui câmp magnetic alternant care înlătură magentizarea măştii de umbrire.

Chiar dacă monitoarele CRT au fost folosite pentru o lungă perioadă de timp acestea erau ineficiente din punct de vedere al consumului de putere, al calităţii imaginii şi chiar al spaţiului ocupat care din cauza dimensiunii tubului era foarte mare. O dată cu evoluţia tehnologiei şi a materialelor folosite în electronică s-au putut dezvoltă noi tipuri de afişaje.

Un alt model de monitor este afişajul LCD (Liquid Crystal Display) care se poate traduce prin afişaj cu cristale lichide. Cristalele lichide sunt substanţe transparente ce prezintă proprietăţi atât ale materialelor solide cât şi ale celor lichde. Acesta funcţionează datorită faptului că lumina trece prin cristalul lichid urmărind configuraţia moleculară iniţială. În momentul în care se aplică un curent electric asupra lui acesta îşi modifică configuraţia moleculară ducând şi la modificarea traiectoriei luminii.

Există două procedee principale în fabricarea afişajelor LCD. Primul principiu constă în prinderea cristalelor lichide între două suprafeţe finale ce prezintă canale, unde canalele unei suprafeţe sunt perpendicularele pe canalele celeilalte. Datorită acestei configuraţii moleculele unei suprafeţe sunt aliniate de la nord la sud , iar moleculele celeilalte suprafeţe sunt aliniate de la vest la est. Apoi acestea sunt forţate într-o stare de rotaţie la un unghi de 90 de grade. Aplicând un voltaj cristalelor lichide , moleculele se rearanjează vertical lăsând lumina să treacă.

Al doilea principiu al afişajelor LCD se bazează pe proprietăţile filtrelor polarizante şi ale luminii. Valurile de lumină naturală sunt orientate la unghiuri aleatoare iar un filtru polarizat este un set de linii paralele foarte fine. Aceste linii acţionează ca o plasa, blocând toate valurile de lumină care nu sunt orientate paralel cu aceste linii. Un al doilea filtru polarizat cu liniile aranjate perpendicular la primul blochează total lumina deja polarizată. Lumina ar trece prin al doilea numai în cazul în care liniile ar fi paralele cu ale primului sau în cazul în care lumina ar fi fost sucită să corespundă cu filtrul polarizat.

Primele monitoare LCD foloseau operaţii cu un număr limitat de pixeli. Cu trecerea timpului ecranele LCD au început să folosească tehnologia afişajului monocrom, tehnologie ce permite utilizarea unui număr foarte mare de pixeli, controlarea şi conectarea lor. Acest lucru a fost posibil prin utilizarea unei matrice în care intersecţia unei linii cu o coloană genera un pixel. Pentru a activa un anumit pixel, activarea liniei şi a coloanei afernete este necesară. Când un anumit pixel este activat, mecanismul din spatele acestui fenomen este acelaşi cu cel explicat anterior bazat pe răsucirea cristalului lichid.

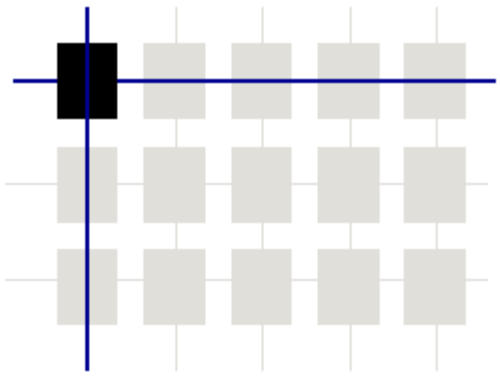

Figura 3.1.3 Matrice pasivă folosită în afişajul monocrom

Când încercăm să dezactivăm un pixel, cristalele lichide nu pot trece dintr-o stare în altă instant. Are nevoie de un anumit timp pentru a trece în altă stare. De asemenea pixelii electrici reprezintă echivalentul unui capacitor în care cei doi electrozi metalici sunt separaţi de un material electric realizat din cristale lichide. Deci atunci când sarcina este eliminată pixel-ul se comportă ca un capacitor când se descarcă. Acest lucru determină timpul de oprire al unui pixel. Acest tip de afişaj este denumit matrice pasivă în care toţi pixelii sunt conectaţi prin fire, formând o reţea.

Atunci când există mii de pixeli, calcularea unor parametrii echivalenţi devine extrem de complicată. În această configuraţie, este destul de probabil ca atunci când activăm un anumit pixel, datorită inductanţei, să se activeze şi unul învecinat cu acesta, ducând la o calitate mai puţin optimă a imaginii.

Această tehnologie era intens folosită în realizarea afişajelor pentru telefoanele mobile. Principalul avantaj era costul scăzut al acestei tehnologii. Tranziţia de la afişajul segmentar la cel monocrom este facil iar costurile de producţie sunt reduse. Se ating scale şi nuanţe de gri prin modulaţia pixelilor în timp. Acest lucru este echivalent cu modulaţia frecvenței semnalului de curent continuu.

Cel mai mare dezavantaj al aceste tehnologii îl reprezintă timpul mare de răspuns datorată efectului de descărcare al capacitorilor. În concluzie aceste afişaje nu sunt potrivite pentru obictele mobile deoarece duce la apariţia fenomenului de fantomare.

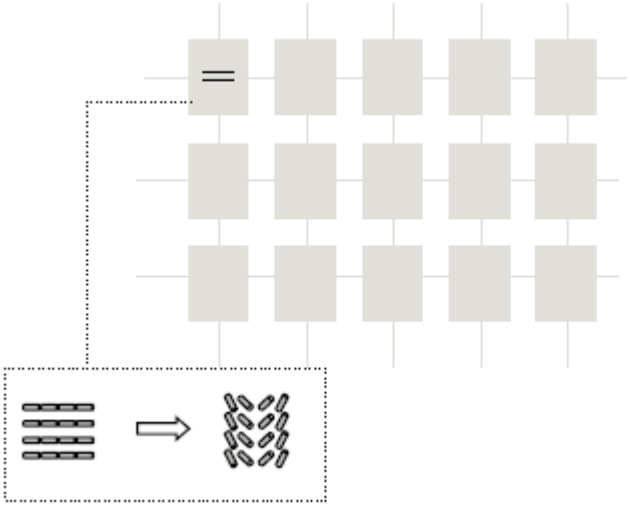

Figura 3.1.4 Orientarea cistalelor lichide în matricea pasivă

O altă abordare asupra acestor tipuri de afişaj a fost din punctul de vedere al culorii ecranului. Iniţial s-a pus problema generării unei imaginii color. Cristalele lichide nu prezintă proprietăţi legate de culoare pentru că ele nu afectează decât polarizarea luminii.

Pentru afişajele color, surse de lumină albă sunt folosite şi apoi filtre color sunt folosite pentru a crea trei culori de bază. Pixelii roşii, verzi şi albaştri sunt plasaţi foarte aproape unul de celălalt pentru a crea un singur pixel colorat.

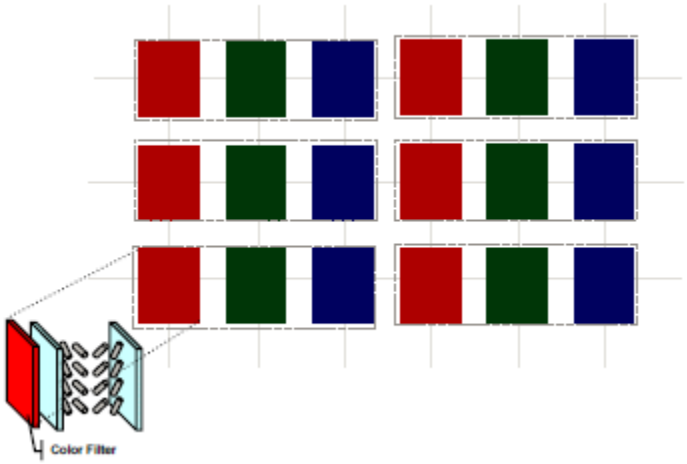

Figura 3.1.5. Afisaj color pasiv de tip CSTN

Pentru a afişa o culoare diferită de cea de bază, mai mulţi pixeli sunt aprinşi simultan. Procesul pentru a aprinde un pixel este acelaşi că în cazul matricei pasive monocrome. Aşadar, în cazul în care roşu şi albastru sunt aprinşi, va rezultă culoarea roz. Tot acest sistem descris se numeşte Afişaj Color Super Twisted Nematic.

3.2 Modem GSM

O comunicație este o transmitere de date și informații, deci de natură tehnică, implicând o conexiune între două sau mai multe puncte distincte. În mass-media de exemplu există numeroase tipuri de comunicații: prin ziare, postere, TV, radio sau și prin Internet.

Termenul de telecomunicații desemnează comunicațiile efectuate la distanță. Astfel radioul, telegrafia, telefonia (fixă sau mobilă), televiziunea, comunicațiile digitale sau rețelele de calculatoare se pot subscrie acestui domeniu.

Elementele componente ale unui sistem de telecomunicații sunt în principiu: emițătorul, canalul de comunicație și receptorul.

În continuare este prezentată o schemă generală a sistemului de comunicație :

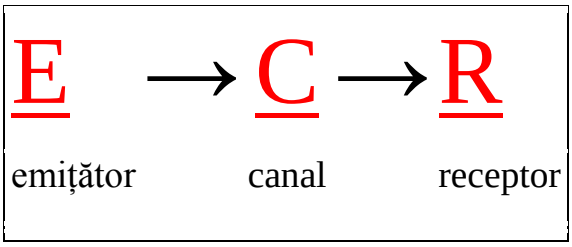

Figura 3.2.1 Schemă generală a sistemului de comunicație

Înainte ca domeniul telecomunicațiilor să aibă o evoluție extraordinară, oamenii comunicau prin diferite mijloacele care nu implicau o tehnologie foarte avansată (exemplu: scrisori, telegraf).

S-a constatat că este nevoie de un mijloc de transmitere a informațiilor într-un mod mult mai rapid și atunci s-a inventat telefonia fixă.

Telefonul reprezintă un dispozitiv de **telecomunicaţie** care serveşte la transmiterea şi receptarea sunetelor la distanţă: constituit de obicei dintr-un **microfon** care are funcţia de transformare a vibraţiilor sonore emise de vocea umană în oscilaţii electrice şi un **receptor** care captează aproape instantaneu aceste **oscilaţii** şi le transformă în **vibraţii sonore**, reproducând identic mesajul transmis de la celălalt capăt al firului.

Dezavantajul telefonului fix era și este în continuare acela că întotdeauna ești legat de un anumit spațiul, unde este instalat acel telefon, pentru a putea transmite informațiile pe care le dorești. Atunci oamnenii de știință s-au gândit sa elimine acest inconvenient și s-a ajuns la inventarea telefonie mobile.

Un telefon mobil este un dispozitiv electronic portabil care funcționează fără fir (prin radio) pe baza rețelei GSM și este folosit în general pentru comunicații personale la distanță mare. Cu inventarea acestui dispozitiv s-a eliminat dezavatajul limitării spațiului. Acum se pune problema să fim în aria de acoperire pentru a avea semnal la telefon și a putea efectua un apel.

În ziua de azi telefonia celulară sau mobilă se bazează pe standardul de comunicații și rețeaua GSM (Global System for Mobile Communications - Sistem Global pentru Comunicații Mobile).

Transmisia informației cu ajutorul telefonului mobil, mai ales cel care are acces la internet se poate face pe mai multe căi: vocal, video, e-mail, SMS/MMS.

O schemă simplă a transmiterii semnalului de la un dispozitiv de comunicație la altul ar putea fii cea de mai jos:

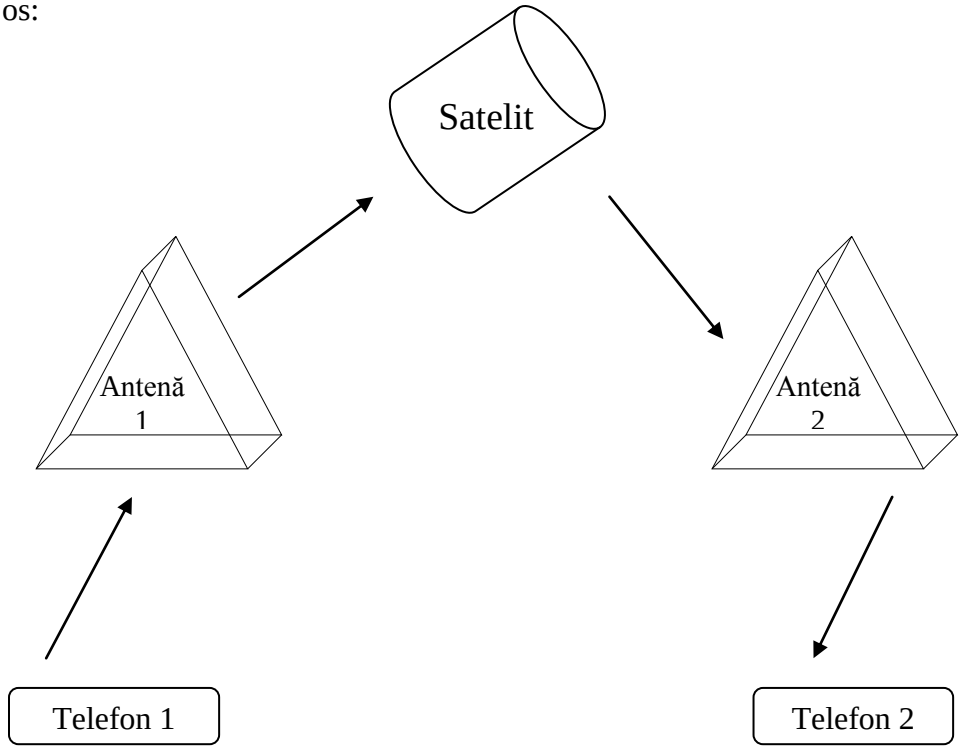

Figura 3.2.2 Transmiterea informațiilor de la un dispozitiv la altul

Global System for Mobile Communications - Sistem Global pentru Comunicații Mobile, prescurtat GSM, este standardul de telefonie mobilă (celulară) cel mai răspândit din lume, precum și numele rețelei de telefonie respective. Atributul „mobil" al multor aparate și dispozitive actuale se referă în primul rând la conectivitatea lor (fără fir, prin semnale radio) la sistemul GSM, practic din orice punct de pe glob unde există oameni. Din aceasta rezultă și mobilitatea utilizatorului.

Sistemul GSM este un sistem numit „celular". Deoarece telefoanele portabile atașabile la GSM (așa-numitele telefoane mobile sau celulare) trebuie să fie ușoare și să aibă acumulatori cât mai ușori, ele au și o putere de emisie radio limitată la circa 4 – 6 km. Drept consecință, releele GSM, numite și „stații de bază", care au antenele în poziții fixe pe stâlpi la sol sau pe clădiri mai înalte, trebuie să fie numeroase, împânzind astfel mari suprafețe, de ordinul unor întregi zone metropolitane și chiar și mai mari, tinzând cu timpul spre acoperirea completă a țărilor.

Fiecare releu GSM acoperă doar o mică suprafață, mai mult sau mai puțin rotundă și cu diametrul de aproximativ 8 – 10 km, numită „celulă". Dacă posesorul telefonului mobil se deplasează

(de ex. călătorește cu mașina), sistemul îl „pasează” de la un releu la altul, urmărindu-l peste tot unde se află. Dacă la trecerea în altă celulă (teritorială) posesorul tocmai vorbește la telefon, convorbirea sa nu este întreruptă și nici măcar deranjată.

Sistemul GSM, bazându-se pe transmisii radio, prezintă în principal riscul captării ilegale a convorbirilor telefonice. El însă prevede ca semnalul sonor, înainte de a fi transmis, să fie digitalizat și criptat, dispunând astfel de o securitate de transmisie ridicată.

Inițial GSM a fost conceput doar pentru telefonie și transmitere de telefaxuri și alte date la viteză constantă. Succesul Internetului a condus însă și la evoluția standardelor GSM, care azi permit, printre altele, accesul mobil la Internet cu viteze mari.

Există o clasificare a sistemelor de comunicații mobile. Această clasificare pe generații se aplică doar rețelelor mobile destinate realizării de legături vocale (sistemelor celulare). Pe de altă parte, rețelele mobile de date au cunoscut și ele o evoluție atât în privința soluțiilor tehnice cât și a performanțelor obținute.

3.3. Clasificare a sistemelor de comunicații mobile

3.2.1 Generația 1 G

Prima generație de sisteme , numită 1G, a folosit transmisia analogică a informațiilor. Inițial, astfel de sisteme au utilizat exclusiv transmisia analogică atât pentru mesaje cât și pentru semnalele de control și semnalizare din sistem. În acest scop se utilizau modulații de tip AM (Amplitude Modulation), SSB (Single SideBand) sau FM (Frequency Modulation).

Ulterior s-a trecut la transmisia numerică a semnalelor de control și apel selectiv, cu modulații de tip FSK (Frequency Shift Keying), menținându-se pentru semnalul vocal transmisia analogică prin modulație de frecvență.

3.2.2 Generația 2 G

Următoarea generație de sisteme , în întregime digitalizată și notată 2G, a fost caracterizată de transmisia numerică a informațiilor de utilizator,integrând mesajele digitalizate cu informațiile de control (comandă) și semnalizare, ceea ce a permis atingerea unor performanțe superioare în raport cu soluțiile folosite anterior și a asigurat compatibilitatea cu rețelele numerice terestre . Rețelele 2G se bazau inițial pe comutația de circuite.

Ulterior, în aceste rețele s-au implementat soluții care să permită creșterea vitezei de transmisie a datelor și utilizarea comutației de pachete de date în paralel cu cea de circuite, născându-se astfel o generație de tranziție, denumită 2,5G.

3.2.3. Generația 3 G

A treia generație, 3G este marcată de o creștere a debitului de transmisie de la valori de ordinul a 10 Kbps la valori cuprinse între 200 Kbps și 2 Mbps. Acest salt presupune o creștere a benzii alocate unui canal precum și trecerea la utilizarea exclusivă a comutației de pachete.

3.2.4 Generația 4 G

A 4-a generaţie de tehnologie mobilă, cea mai stabilă şi rapidă, cu viteză de descărcare de până la 150 Mbps şi încărcare de până la 50 Mbps.

Rețelele mobile celulare GSM oferă o serie de avantaje față de alte soluții tehnice:

- capacitate de transmisie sporită
- consum redus de energie
- acoperire geografică extensivă
- interferențe reduse cu alte semnale
- toleranță la greșeli de transmisie sau defecțiuni
- latență redusă și stabilitate.

Bazate pe aceeaşi tehnologie GSM/GPRS sunt şi modulele GSM ce pot interfaţa cu diverse unităţii de procesare şi control cum ar fi microcontrolere.

Un astfel de modem GSM este defapt un tip specializat de modem ce acceptă o cartelă SIM şi operează sub o anumită reţea de telefonie mobilă, exact ca un telefon mobil. Din punctul de vedere al opreatorului de telefonie mobilă, modemul GSM este un simplu telefon mobil. Când un modem GSM este conectat la un calculator sau la un microcontroler, acesta permite unităţilor să comunice cu reţeaua mobilă. Deşi în principal aceste modemuri sunt folosite pentru asigurarea conectivităţii la intenet, o mare parte dintre ele pot fi folosite pentru a trimite şi primii SMS-uri şi MMS-uri.

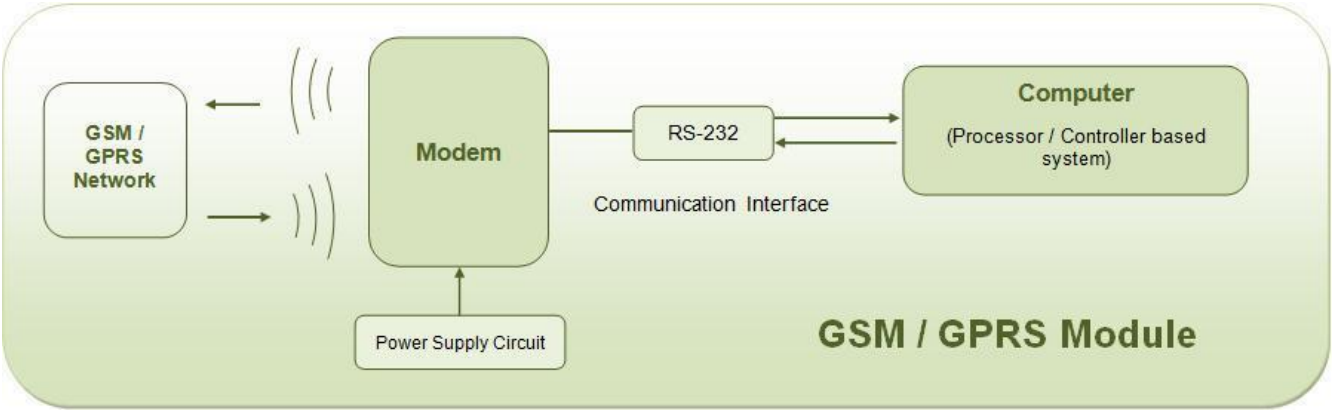

Figura 3.2.3 Schema bloc a unui modem GSM

Un modem GSM poate fi un modul dedicat cu o conexiune serială, USB sau Bluetooth, sau poate fi un telefon mobil ce oferă capabilitățiile unui modem GSM. Acest tip de dispozitiv poate suportă toate tipurile de protocoale de la tehnologii GPRS şi EDGE de 2.5G până la tehnologii 3G şi 4G cum ar fi WCDMA, UMTS, HSDPA şi HSUPA.

Un modem GSM prezintă o interfaţă care permite diferitelor aplicaţii să trimită şi să primească mesaje printr-o platformă modem. Mesajele trimise şi primite prin intermediul acestui modem sunt taxate de către operatorul de telefonie mobilă în regim normal ca şi când ar fi fost trimise de pe un telefon mobil normal. Pentru ca acest lucru să fie posibil, modemul GSM trebuie să prezinte un set extins de comenzi AT pentru primirea şi trimitearea de mesaje SMS, urmărind specificaţiile definite în ETSI GSM 07.05 şi în 3GPP TS 27.005.

Adiţional pe lângă comenzile standard AT, modulele GSM suportă şi un set extins de comenzi AT. Aceste comenzi AT extinse sunt definite în standardele GSM. Prin intermediul acestor comenzi se pot realiza următoarele operaţii:

- cititul, scrisul și ștersul mesajelor SMS
- trimiterea de mesaje SMS
- monitorizarea puterii semnalului
- monitorizarea nivelului de baterie și al stării de încărcare
- cititul, scrisul şi căutatul în întrările agendei telefonice

Numărul de mesaje SMS care pot fi procesate de un modem GSM pe minut este foarte mic – aproximativ între 6 și 10 mesaje SMS pe minut.

Comenzile AT sunt instrucţiuni folosite în controlarea modemului. AT este abrevierea de la Attention. Fiecare linie de comandă începe cu AT sau at, de unde şi denumirea de comenzi AT. Multe dintre comenzile folosite pentru controlul modemelor cu fir , ca de exemplu ATD(Sună), ATA(Răspunde), ATH (Prinde controlul) şi ATO (Întoarcere în starea de date online) sunt suportate şi de modemele GSM/GPRS. Pe lângă aceste comenzi standard, modemele GSM/GPRS mai conţin şi anumite comenzi specifice operaţiilor cu mesaje SMS, cum ar fi AT+CGMS (Trimite mesaj SMS) sau AT+CMGR (Citeşte mesaje SMS). Prefixul AT informează modemul de începutul unei linii noi de comandă şi nu este parte din numele instrucţiunii.

Alte operații ce pot fi îndeplinite cu ajutorul setului de comenzi AT specific modulelor GSM sunt:

- preluare de informații de bază referitoare la modem (numele producătorului, numărul modelului, numărul IMEI și versiunea de software
- trimitere și primire fax
- stabilirea unei conexiuni de date sau de voce cu un alt modem
- îndeplinirea de obiective referitoare la securitatea dispozitivului cum ar fi blocarea sau deblocarea acestuia, schimbarea parolei, etc.

De subliniat este faptul că există și comenzi AT care necesită suportul operatorului de rețea mobilă. De exemplu SMS prin intermediul GPRS poate fi porint folosind comanda +CGSMS , dar dacă operatorul de rețea mobilă nu suportă o asemenea funcție de transmitere a mesajelor SMS prin intermediul GPRS, modemul nu foate folosi aceasta funcție.

CAPITOLUL 4
PROIECTARE ȘI IMPLEMETARE

4.1 Diagrama bloc a sistemului

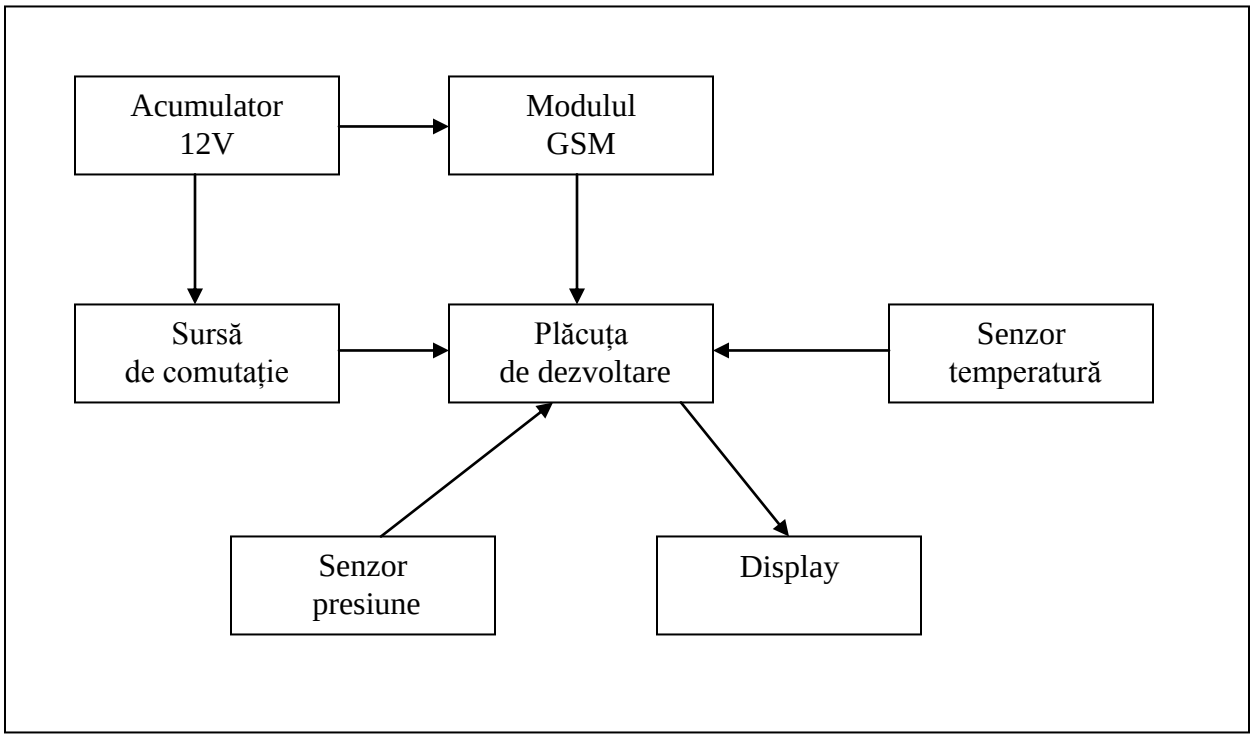

Figura 4.1.1 Schema bloc a sistemului

4.2 Design-ul general

Întregul sistem prezentat mai sus se bazează pe recepționarea unor date din exterior, prelucrarea acestora de către placa de dezvoltare și afișarea lor pe diplay.

Fluxul circuitului este următorul: avem un acumulator de 12V la care modemul GSM este conectat în mod direct , iar placa de dezvoltare este conectată în mod indirect având nevoie de o sursă de comutație ce transformă tensiunea dată de acumulator (12V) într-o tensiune de 5V necesară alimentării plăcuței. La plăcuță sunt conectate dispozitivele periferice, adică senzorul de presiune, senzorul de temperatură, display-ul și modemul GSM. Senzorul de temperatură măsoară temperatura mediul exterior și o transmite către microcontroler. Senzorul de presiune citește o valoare obținută prin

apăsarea suprafeței sale sensibile, pe care o transmite microcontrolerului. Semnalul transmis de senzorul de presiune este un semnal analogic, ce este transformat de către convertorul analog- digital al microcontrolerului în semnal digital. Modemul GSM este conectat și el la placa de dezvoltare. Infomațiile obținute de la aceste periferice sunt afișate pe display –ul sistemului.

4.3 Acumulator

Acumulatorul YUASA imi alimentează ciruitul la o tensiune de 12V având un curent de 2A. Se poate folosi pentru diferite echipamente de testare și măsurare, surse de curent electric , iluminat de urgență, aplicații fotovoltaice.

Am ales această componetă deoarece este îndeajuns de puternică să îmi alimenteze tot circuitul, are un preț accesibil si o dimensiune potivită pentru machetă mea. Consumul în IDLE este în jur de 15mA, ceea ce înseamnă că, ținând cont că acumulatorul este de 2A, autonomia sistemului este de aproximativ 133h.

Figura 4.3.1 Acumulator YUASA

Caracteristici specifice conform foii de catalog:

- reîncarcare rapidă în urma unei descărcări complete
- sistem de suspensie electrolit
- recombinare de gaz
- utilizabilă în orice situație
- densitate de energie mare
- grile de calciu și plumb pentru o durată de viață mai mare

Construcția unică de la Yuasa asigură faptul că nu există pierderi de electrolit din carcasa sau pe la terminale. Sistemul de suspensie electrolit are încroporat o microfibră de sticlă mată ce reține suma maximă de electrolit în celule. Electrolitul este ținut în materialul separator și de acolo nu poate să mai iasă. Nu sunt adăugate alte geluri sau substanțe contaminate.

Acumulatoarele Yuasa au încorporată ultima tehnologie în ceea ce privește tehnica de recombinare a oxigenului ceea ce permite controlul generării de gaz în timpul uzului în condiții normale. Datorită acestor caracteristici se poate spune că bateria se întreține singură.

Aceste baterii sunt echipate cu un sistem de ventilare de joasă presiune ce eliberează gazul în exces și automat este eliberat gazul și atunci când bateriile se supraîncarcă.

Pot fi folosite într-o anumită gama de temperaturi ce dau o flexibilitate considerabilă în construcția sistemului și a locației.

Încărcare – 15°C to 50°C
Descărcare – 20°C to 60°C
Păstrare – 20°C to 50°C

4.4 Modemul GSM Wavecom M1306B

Pentru prezenta lucrare de licență am folosit modemul GSM Wavecom M1306B. Am ales acest model datorită specificațiilor tehnice și a costului relativ redus. Acesta se alimentează la o tensiune cuprinsă între 5V – 32V, oferind o putere de ieșire la o frecvență de 900MHz de 2W sau la o frecvență de 1800MHz o putere de ieșire de 1W.

Un al doilea motiv pentru care am ales acest model este acela că are dimensiuni reduse: 73 x 54 x 25mm cu o greutate totală de 82g. Pentru realizarea machetei aveam nevoie de ceva cât mai compact, dar cu funcționalitatea dorită. Este conceput pentru aplicații de date, fax, SMS si voce.

Figura 4.4.1 Modemul GSM Wavecom M1306B

Modemul GSM este alimentat direct la acumulator, și este legat și de placa de alimentare prin pinii A.2 si A.3 aferenți UART2, 9600, 8N1 - 9600bps- rata detransfer, 8 biti de date, NO PARITY, 1 bit de stop; in carcasa mufei DB9.

Mecanismul prin care sunt trimise și recepționate date este următorul: utilizatorul trimite un mesaj SMS către modemul GSM, modemul GSM comunică cu microcontrolerul, îi semnalează faptul că tocmai a fost interogat, microcontrolerul preia date de la componentele periferice conectate lui, adică senzorul de temperatură și senzorul de presiune, prelucrează aceste informații,apoi le transmite modemului GSM și în final modemul GSM trimite SMS utilizatorului cu informațiile cerute.

4.5 Sursa de comutație

Lângă acumulator există o sursă în comutație care transformă tensiunea inițială dată de acumulator de 12V într-o tensiune de 5V. Această sursă este conectată în mod direct la acumulator. Rolul acestei surse de comutații este acela de a alimenta placa de dezvoltare la o tensiune corespunzătoate specificațiilor plăcii.

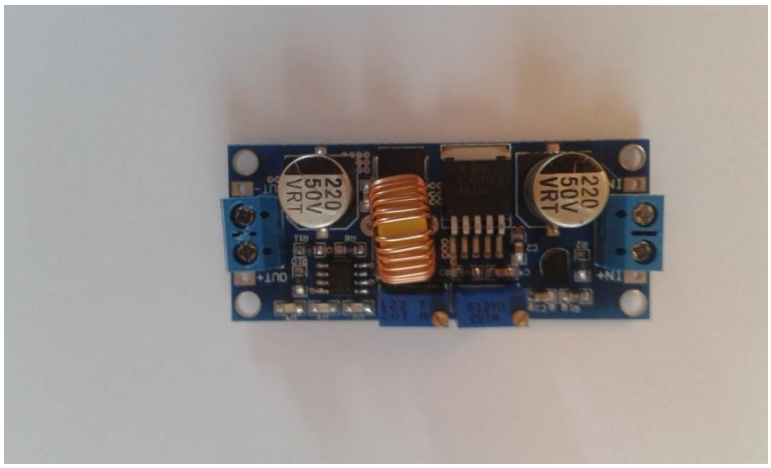

Figura 4.5.1 Sursă în comutație

4.6 Microcontroler STM32F407

STM32F407 face parte din familia procesoarelor ARM, cu o arhitectură RISC. Această linie de procesoare a fost proiectată pentru uzul în medicină, industrie și aplicații ale consumatorilor care au nevoie de un nivel mare de integrare si performanță, memorii încapsulate și un set bogat de periferice. Acesta ofera performanțele unui nucleu Cortex – M4 ce funcționează la frecvența de 168 MHz.

A fost ales datorită eficienței consumului și a scalării dinamice de putere poate duce la un consum de curent foarte mic, de ordinul 238 μA/MHz la frecvența de 168 MHz.

Alte criterii care au dus la utilizarea acestui microprocesor sunt capabilitatea lui bogată de a interfața cu o multitudine de periferice și datorită prețului scăzut cu care a fost achiziționat de pe piața locală.

Acesta are rolul de a recepționa informații de la perifericele de intrare (senzorul de temperatură și senzorul de presiune), să le convertească în semnal digital cu ajutorul unui ADC pe 12 biți, iar apoi să prelucreze informația trimițând un rezultat către perifericele de ieșire (display LCD și modem GSM).

Informația recepționată de la senzorul de presiune și de la senzorul de temperatură prin intermediul pinilor GPIO este convertită cu ajutorul unui ADC pe 12 biți din semnal analogic în semnal digital. Acesta face verificările impuse de algoritmul implementat special pentru acest sistem de măsurare iar în momentul în care condiția este respectată el trimite informațiile către perifericele de ieșire tot prin intermediul unor pini GPIO. Către afișajul LCD acesta trimite informații cu privire la starea modulului GSM, temperatura și greutatea înregistrată și ultimul număr ce a realizat interogarea sistemului.

Informațiile către modemul GSM sunt trimise în două situații: prima este aceea în care un utilizator care cunoaște numărul de telefon al modului GSM interoghează sistemul prin simpla trimitere a unui SMS, caz în care informația legată de temperatură și cea de greutate recepționate în acel moment de timp sunt trimise către modemul GSM iar acesta este comandat să trimită un mesaj SMS conținând aceste informații către numărul ce a realizat interogarea. A doua situație este aceea în care numărul salvat de sistem ca fiind deținătorul platformei primește informații, proces cauzat de modificarea valorii anterior trimise cu un anumit număr de unități, mai exact cu 500 de unități, atât în creștere cât și în scădere.

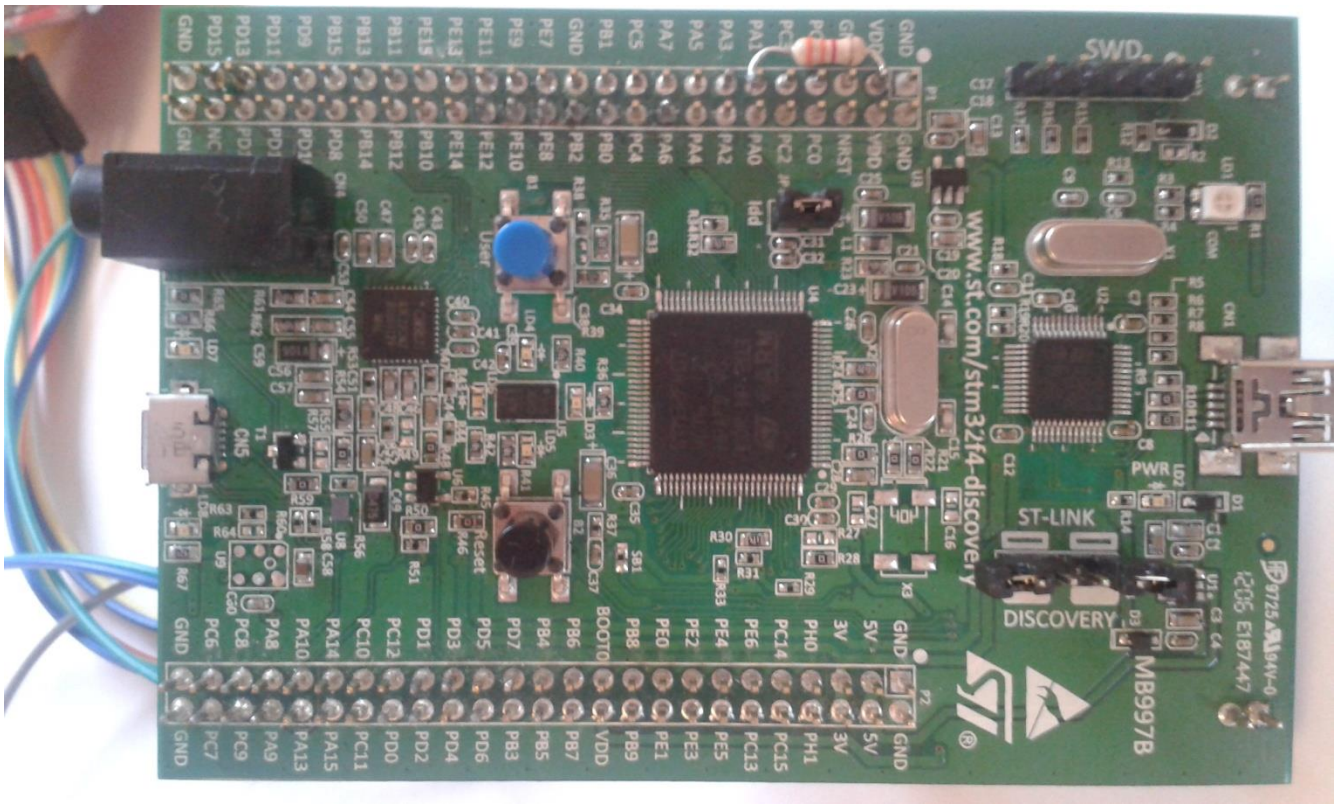

Figura 4.6.1 STM32F407

4.7 Display Nokia 3310

Display - ul reprezintă un dispozitiv periferic conectat la placuța de dezvoltare. Este un LCD des folosit în proiecte. Zona activă acoperă 30 * 22 mm din suprafața inițială de 38 * 35 mm, și are o rezoluție de 84 * 48 pixeli. Am ales această componentă în realizarea proiectului de diplomă în primul rând pentru că se poate interfața foarte ușor și în al doilea rând pentru că aveam unul acasă.

Folosește o comunicație serială standard SPI pentru conectarea cu placa de dezvoltare, funcționează la o tensiune de 2,7 – 3,3 V, având o putere de consum mică, și rezistă într-un interval de temperatură de la -25 la +70 grade Celsius.

Figura 4.7.1 Display Nokia 3310

Diplay-ul este divizat în cinci linii de afișaj:

- pe primele trei linii din partea de sus a ecranului sunt afișate următoatele informații:

a) **GSM : ONLINE/SEARCHING** – când sistemul este alimentat inițial pe prima linie a display - ului apare mesajul „GSM : SEARCHING”; acest mesaj rămâne pe ecran timp de aproximativ 1 minut până când se realizează conexiunea la nivel de rețea telefonică; în momentul în care a fost realizată conexiunea atunci pe ecran va fi afișat mesajul „GSM : ONLINE”; acest mesaj ne anunță că sistemul este conectat și că funcționează și este disponibil pentru recepționarea și transmiterea datelor.

b) **TEMP** : acest mesaj este afișat pe a doua linie a ecranului și reprezintă informația primită de la senzorul de temperatură; temperatura este cea recepționată din mediul ambient.

c) **SCALE** : acest mesaj apare pe a treia linie a display – ului și reprezintă scala senzorului de presiune; inițial este afișată valoarea 21 pe ecran; această valoare 21 reprezintă valoarea cea mai mică înregistrată de senzorul de presiune, mai mic de 21 nu a afișat; afișajul acestei lini se modifică atunci când avem o diferență între măsurători de 500 de unități, atât în minus cât și în plus, valoarea maximă fiind de 4096 de unități.

- pe ultimele două linii se găsesc informații despre ultimele evenimente logate

a) pe linia a patra se afișează pen-ultimul eveniment logat

b) pe linia a cincea se afișează ultimul eveniment logat

4.8 Senzorul de temperatură

Senzorul de temperatură reprezintă o componentă periferică ce este conectată la placa de dezvoltare.

Senzorul de temperatură este conectat la ARM printr-un singur fir - BUS 1W - ONE WIRE, aferent pinului A.1.

Acesta preia temperatura mediului înconjurător , trimite această informație către microcontroler, microcontrolerul transformă datele cu ajutotul unui convertor în informație înțeleasă de el și apoi o trimite către afișajul display -ului .

Figura 4.8.1 Senzor de temperatură

4.9 Senzor de presiune

Această componetă este una dintre cele mai importante in acest sistem. Informația transmisă de el face ca sistemul să își îndeplinească funcționalitatea. Senzorul de presiune este cel care monitorizează greutatea stupului de albine. Existența lui în acest sistem este primordial.

Așa cum am specificat și la începtul acestei lucrări de licență, sistemul reprezintă o machetă a unui produs ce poate fi folosit în producție. Așadar am folosit un singur sensor de presiune. Un senzor de presiune rezsitiv. Acest tip de senzor nu este unul cu o acuratețe foarte mare în ceea ce privește măsurătorile, având erori de câteva procente.

Pentru a construi sistemul la dimensiunile reale , aveam nevoie de patru sensor de presiune, de o calitate superioară, ce avea un preț destul de riciat, undeva la 45euro bucata. De aceea am recurs la această metodă de a realiza o machetă care să aibe funcționalitatea dorită.

Ca specificații acest senzor este capabil să sesizeze apăsarea, dispunând de o suprafață sensibilă de 4.47 cm . Modul de funcționare este foarte simplu, făcând ca interfațarea cu microcontrolerul să fie foarte rapidă. Senzorul dispune de doi conectori, iar rezistența măsurată între acești doi conectori

variază cu apăsarea. Atunci când nimic nu apasă pe senzor, rezistența acestuia este mai mare de 1MΩ, iar pe măsură ce este aplicată presiune rezistența scade.

Greutatea unui stup va fi simulată prin apăsarea suprafeței sensibile. Gama de variație este între 0 – 4095 unități, adică 4096 de valori distinct.Sensibilitatea este setată din 500 în 500 de unități. Informația culeasă de la senzorul de presiune este transmisă către microcontroler, transfomată cu ajutorul unui ADC din semnal analogic în semnal digital, și astfel transformată transmisă către display și modemnul GSM.

Datele de la senzorul de presiune sunt transmise în trei situații către modemul GSM:

- atunci când modemul GSM este interogat, trimite informația către orice număr ce îl interoghează
- trimite informația către un număr prestabilit – cel al deținătorului platformei- atunci când depășește un anumit prag stabilit în programul microcontrolerului
- sau atunci când scade sub un anumit prag

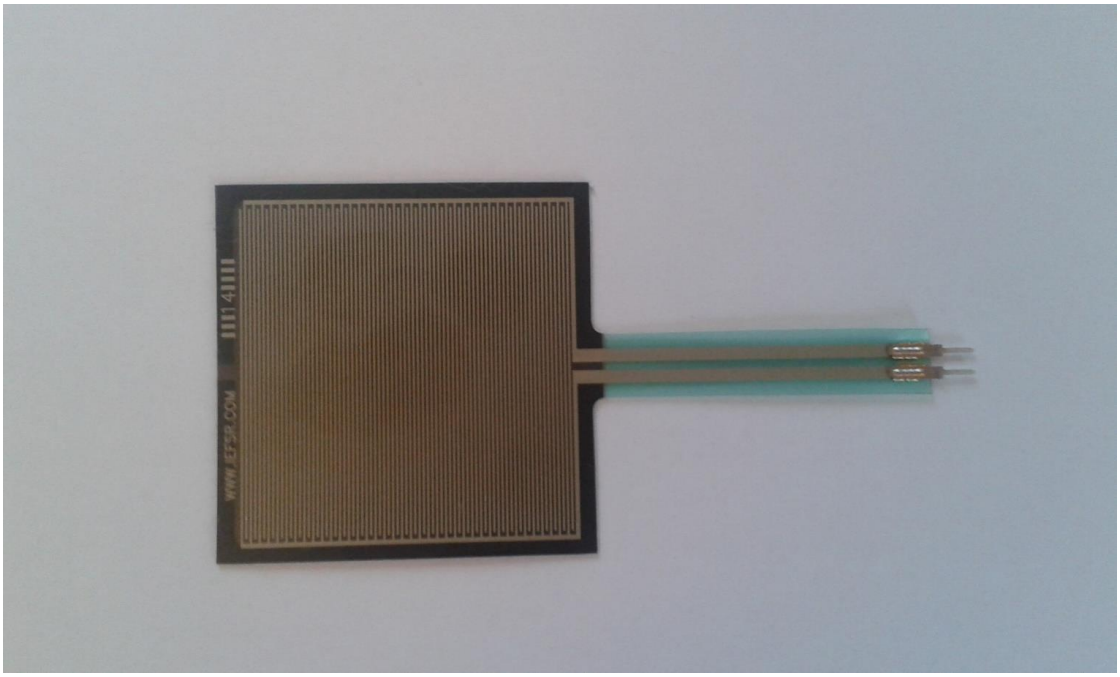

Figura 4.9.1 Senzor de presiune rezistiv

4.10 Realizarea hardware și software a sistemului

Prima etapă în construirea sistemul a fost comandarea pieselor. După o cercetare amănunțită am ajuns la concluzia că piesele deschise mai sus au performanțele necesare pentru a putea realiza întreaga machetă.

După ce am reușit să strâng toate piesele, am trecut la partea efectivă de construcție a sistemului prin conectarea componentelor între ele. Primele piese care au ajuns au fost placa de dezvoltare și modemul GSM. Am asamblat aceste două componente și apoi am trecut la interfațarea acestora.

Pentru a funcționa un sistem are nevoie atât de o parte fizică, hardware, cât și de o parte logică, software. Componenta software asigură interconectarea tuturor celorlalte componente ale sistemului, transformându – le într-o entitate.

Apoi rând pe rând au început să vină piesele, fiecare a fost conectată la sistem, și pentru fiecare componentă există o parte de cod ce îi asigură funcționalitatea în sistem.

Am dorit ca acest sistem să aibă o formă cât mai compactă și cât mai ușor de manevrat, așa că următoarea etapă a fost să construiesc un suport unde să țin aceste componente împreună. Ideea a fost să construiesc un sport din plăci de plexi glass.

Pentru aceasta a fost nevoie de o proiectare 3D a fiecărei componente pentru a ști exact locul unde or să stea, ca atunci când urma să taiem efectiv placa de plexi glass să nu plesnească. Am folosit programul de poiectare CATIA.

Am introdus dimensiunea exactă a fiecărei componente și am stabilit locul unde o sa fie amplasată. Acestea sunt imaginile în urma proiectării.

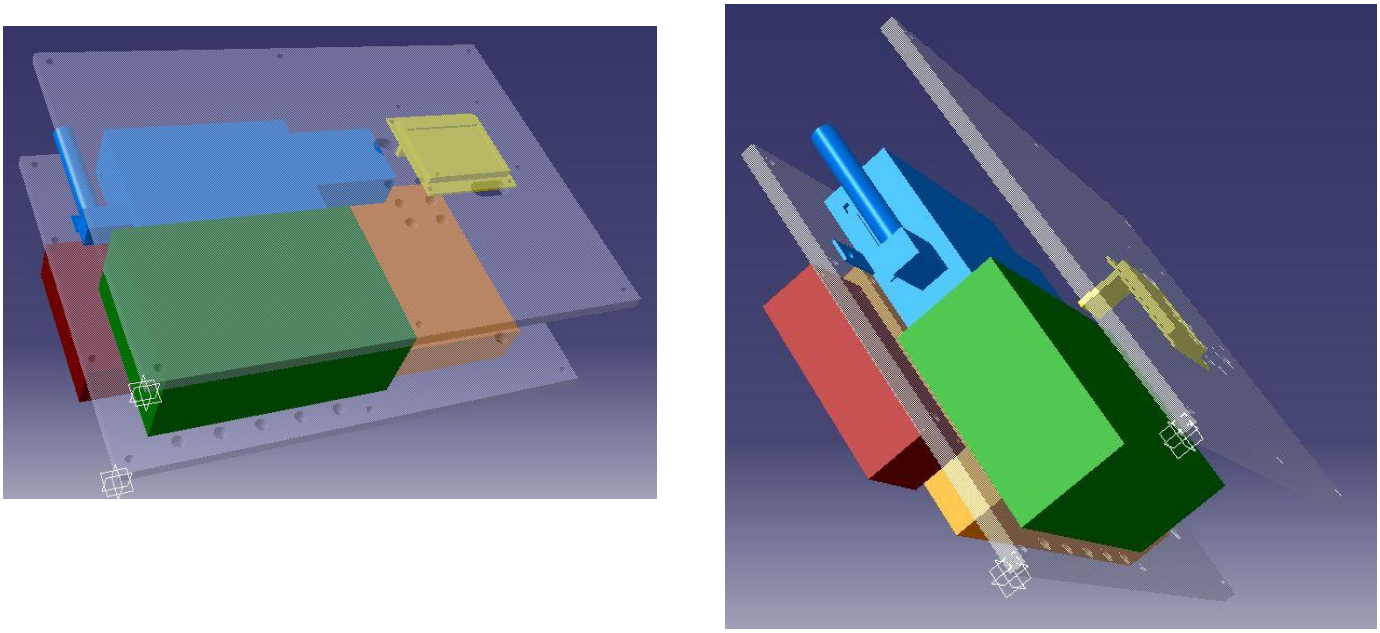

Figura 4.10.1 Proiectarea 3D a sistemului – imaginea 1

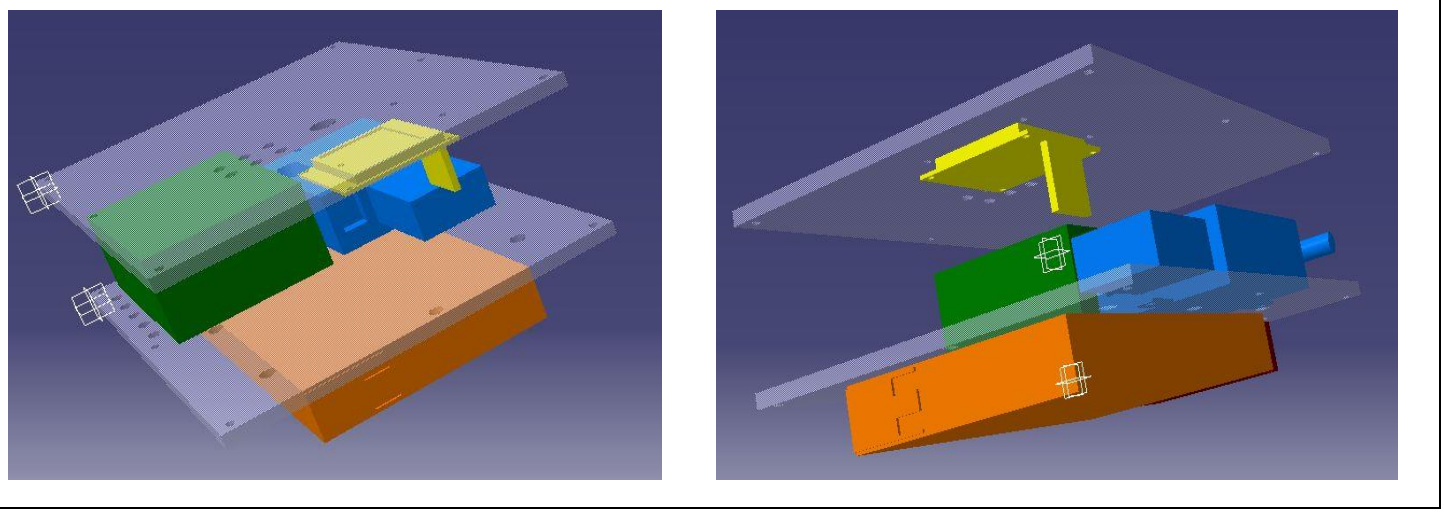

Figura 4.10.2 Proiectarea 3D a sistemului – imaginea 2

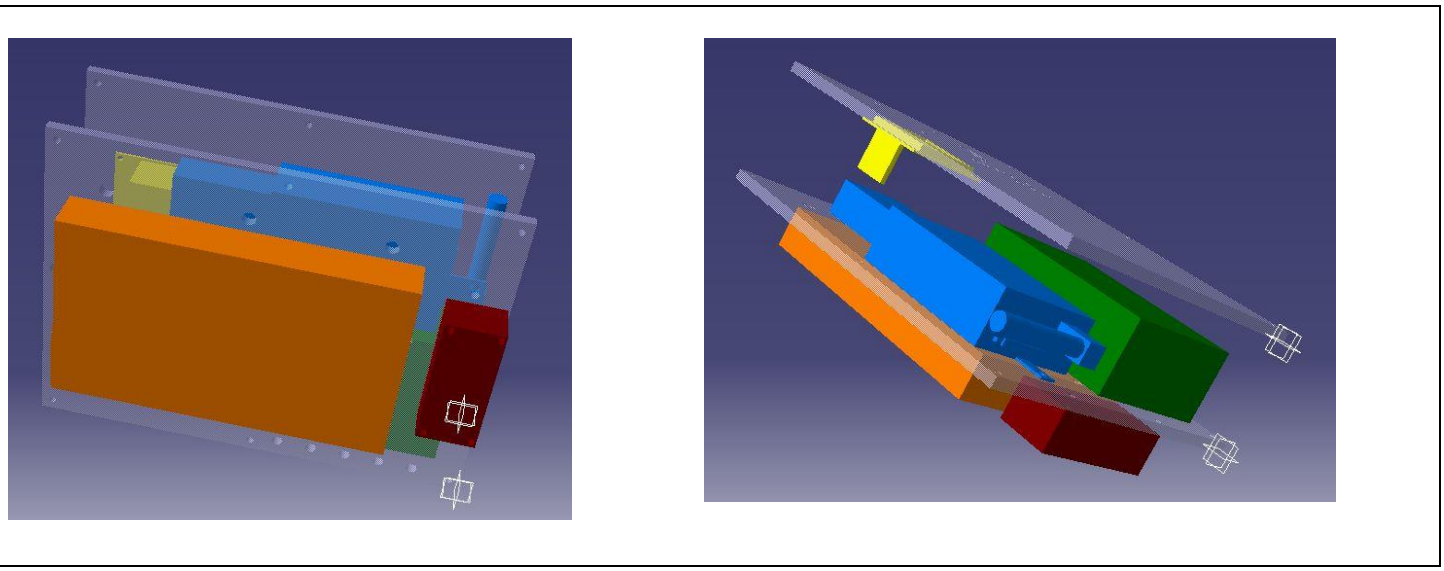

Figura 4.10.3 Proiectarea 3D a sistemului – imaginea 3

Imagini cu proiectul final :

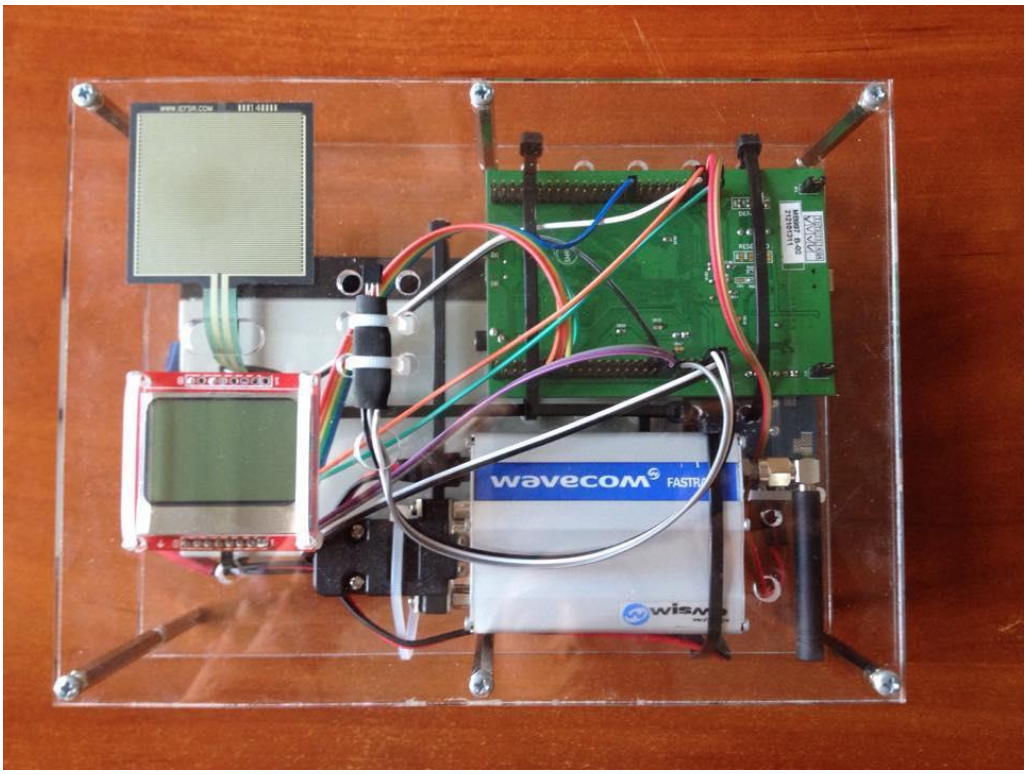

Figura 4.10.4 Poza machetă privită de sus

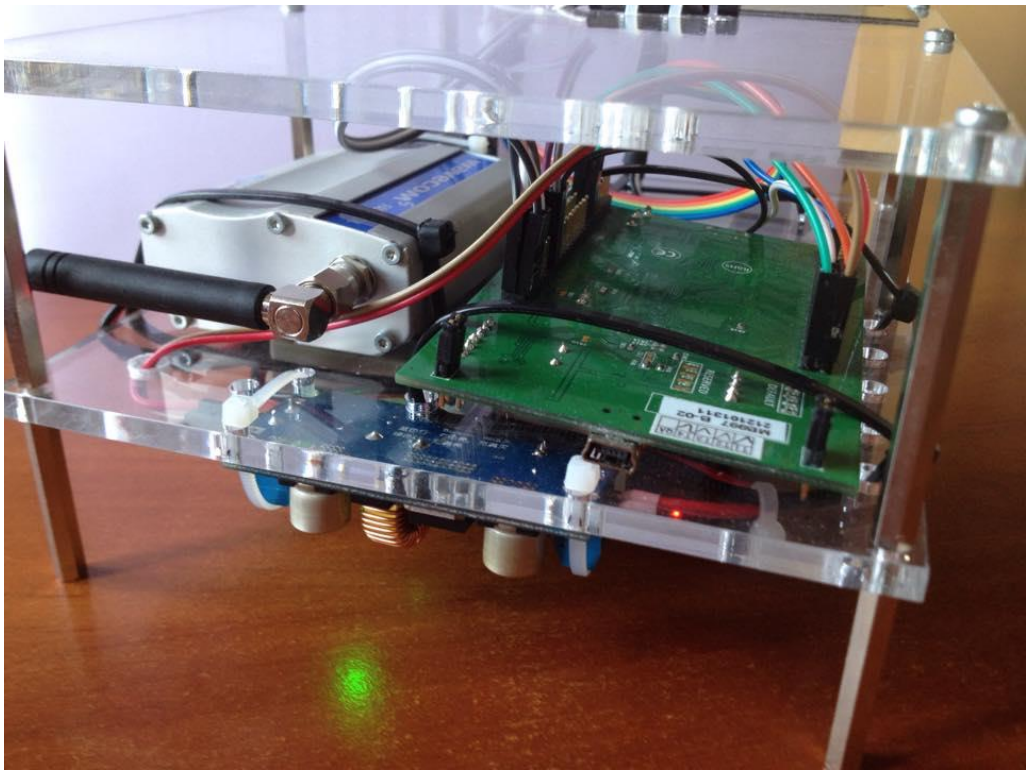

Figura 4.10.5 Poza machetă privită din lateral – imagine 1

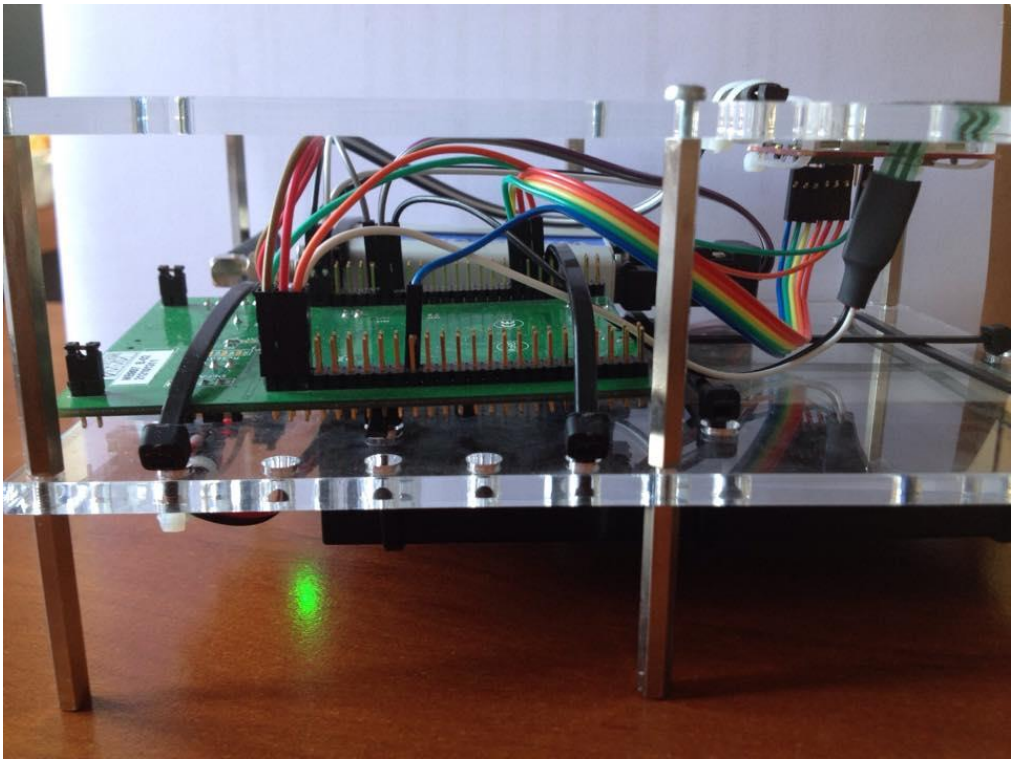

Figura 4.10.6 Poza machetă privită din lateral – imagine 2

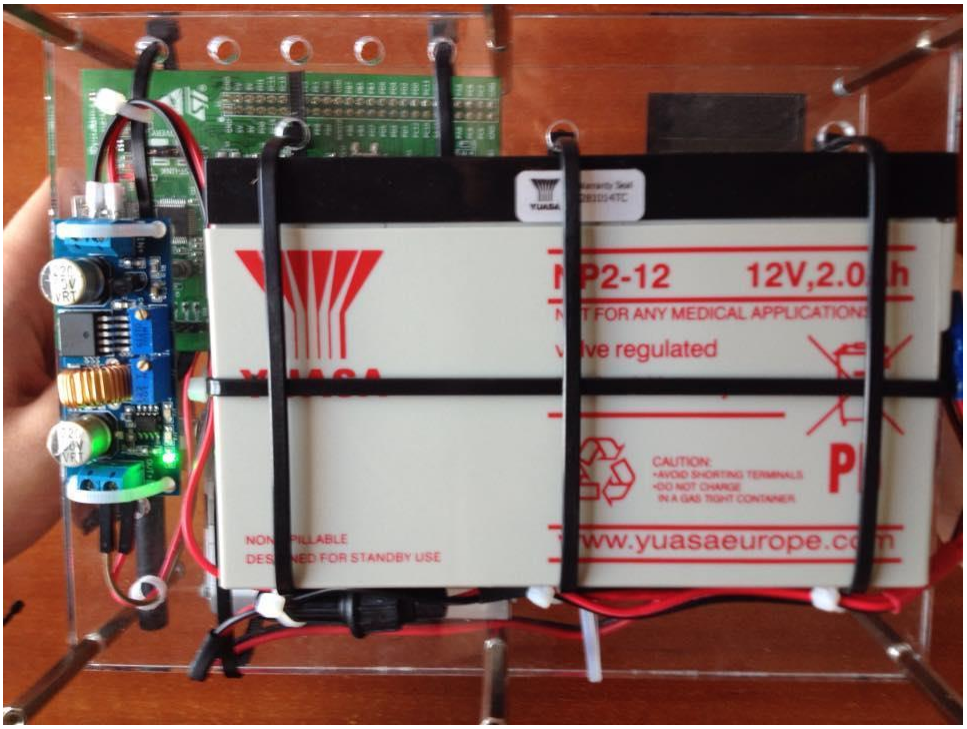

Figura 4.10.7 Poza machetă privită de jos

Concluzii

În urma conectării tuturor componentelor descrise în capitolele anterioare am reușit să realizez un sistem stabil de monitorizare a greutății stupilor. Proiectul de față reprezintă o machetă a unui produs ce poate fi utilizat în producție.

Având în vedere că programele folosite la dezvoltarea logică a sistemului sunt gratuite, costul de producție este minim, având un atu față de programele ce rulează pe piață sub diverse licențe.

Acest sistem construit îndeplinește toate facilitățile gândite inițial. Cu ajutorul unui acumulator am reușit sa alimentez înterg sistemul, având o autonomie de 133 de ore. Cu ajutorul microcontrolerului am reușit să dezvolt partea logică a proiectului, să conetez toate perifericele adiacente și să primesc într-un final infomațiile dorite, atât pe afișajul unui diplay cât și sub forma unor mesaje SMS pe telefonul mobil.

Am reușit în final să realizez un cântar electronic inteligent, capabil să transmită periodic (cu ajutorul unui modem GSM) mesaje SMS programabile precizând greutatea măsurată, diferențele de greutate in timp, praguri de greutate.

Contribuția personală constă în realizarea acestei machete, deoarece nu există un produs pe piață care să aibă funcționalitatea implementă de mine.

M-am gândit și la unele îmbunătățiri ale proiectului ce pot fi implementate intr-un viitor proiect de dizertație. Una din îmbunătățiri ar putea fi conectarea unui modul GPS, care ne-ar oferi informații despre locația sistemul, având o utilitate foarte importantă în cazul în care se comite un furt. O altă îmbunătățire ar fi crearea unei aplicații de stocare a informațiilor primite de la periferiele conetate la microntroler, astfel obținându-se o bază de unde se pot scoate niște rapoarte în funcție de anumiți parametric.

Bibliografie

1. http://electronics.howstuffworks.com
2. http://www.d-d-s.nl/fotos-wavecom/specs-m1306B.pdf
3. http://ro.wikipedia.org/wiki/GSM
4. https://www.elprocus.com/temperature-sensors-types-working-operation/
5. http://www.convectronics.com/images/catalog/Sen4.pdf
6. http://www.pdhonline.org/courses/e166/e166content.pdf
7. https://en.wikipedia.org/wiki/Force-sensing_resistor
8. http://www.sensitronics.com/products/force_sensing_resistor.htm
9. http://akizukidenshi.com/download/ds/interlinkelec/94-00004_Rev_B%20FSR%20Integration%20Guide.pdf
10. https://books.google.ro/books?id=5PDx2Q9Ea_YC&pg=PA133&lpg=PA133&dq=microcontrollers+theory+and+applications+pdf&source=bl&ots=uCpsQ9Zspw&sig=5NF22_C0pFfGOxkA9s6sV9y_NKU&hl=ro&sa=X&ei=z7WCVafCJ4aOsAGi6YDwCA&sqi=2&ved=0CGIQ6AEwCA#v=onepage&q&f=false
11. http://www.mikroe.com/chapters/view/64/chapter-1-introduction-to-microcontrollers/
12. https://en.wikipedia.org/wiki/Microcontroller
13. https://ti.tuwien.ac.at/ecs/teaching/courses/mclu/theory-material/Microcontroller.pdf
14. https://en.wikipedia.org/wiki/PIC_microcontroller
15. https://books.google.ro/books?id=J2vHqNAImwsC&pg=PA373&dq=arm+microcontroller&hl=ro&sa=X&ei=DheUVc-QG4KdsAGAioywCA&ved=0CDIQ6AEwAA#v=onepage&q=arm%20microcontroller&f=false
16. http://andrei.clubcisco.ro/cursuri/3pm/lab1.pdf
17. http://vega.unitbv.ro/~ogrutan/ti/cap10.pdf
18. http://www.developershome.com/sms/atCommandsIntro.asp
19. http://www.tech-faq.com/humidity-sensors.html
20. http://www.st.com/web/en/catalog/mmc/FM141/SC1169/SS1577/LN11

ANEXA 1

MAIN

```
#include <stdio.h>

#include "diag/Trace.h"

#include "timer.h"

#include "led.h"

#include "gsm.h"

#include "uart.h"

#include "lcd.h"

#include "adc.h"

#include "onewire.h"

#include "ds18b20.h"

#pragma GCC diagnostic push

#pragma GCC diagnostic ignored "-Wunused-
parameter"

#pragma GCC diagnostic ignored "-Wmissing-
declarations"

#pragma GCC diagnostic ignored "-Wreturn-
type"

#define SMS_QUE_SIZE 32

#define SMS_PERIOD 2000

struct

{SmsStructType sms[SMS_QUE_SIZE];

uint8_t writeIndex;uint8_t readIndex;

uint8_t count;}

gSmsQue;

void queSmsPush(char* pPhone, char*
pText);

SmsStructType* queSmsPop(void);

OWire gOWHandler;

uint8_t scaleVariationExceeded(uint16_t
scale);

uint8_t gFirst = 1;

int main(int argc, char* argv[])

{uint8_t str[200];SmsStructType sms;

char alertPhoneNumber[20] =
"+40761631739";

char registerPass[] = "REGISTER";

GsmStatusEnumType gsmStatus =
GSM_ERROR_CREG;

uint32_t temp = 0;uint32_t scale = 0;

SmsStructType* pSms = NULL;

uint32_t lastSms = 0;uint8_t gsmOnline =0;

uint8_t gsmCounter = 60;

trace_printf("System clock: %uHz\n",
SystemCoreClock);

timerStart();

ledInit();

adcInit();

uartInit();

lcdInit(0x38);

OWInit(&gOWHandler, GPIOA, GPIO_PIN_1);

ds18b20Init(&gOWHandler);

memset(&gSmsQue, 0x00, sizeof(gSmsQue));

lcdTextLine(GSM_TEXT_LINE,
"GSM:SEARCHING");

lcdTextLine(TEMP_TEXT_LINE,
"TEMP: ERROR");

lcdTextLine(SCALE_TEXT_LINE,
"SCALE:ERROR");

lcdTextLog("INIT DONE");

lcdTextLog("START");

lcdTextLog("SEARCHING...");

while (gsmCounter)

{if (GSM_OK == gsmInit())

{gsmOnline = 1;break;}

timerSleep(1000);gsmCounter--;}

if (gsmOnline)

{lcdTextLine(GSM_TEXT_LINE,"GSM:ONLINE");}

else

{lcdTextLine(GSM_TEXT_LINE,"GSM:OFFLINE");
}

sprintf(str, "#%s", alertPhoneNumber);
```

```
lcdTextLog(str);

queSmsPush(alertPhoneNumber, "SYSTEM UP");

while (1)

{ledToggle(LED_GREEN);

scale = adcReadScale();

sprintf(str, "SCALE: %lu", scale);

lcdTextLine(SCALE_TEXT_LINE, str);

if (scaleVariationExceeded(scale) &&
!gFirst)

{sprintf(str, "TRIGGER %d", scale);

lcdTextLog(str);

sprintf(str, "SCALE=%d TEMP=%d", scale,
temp >> 4);

//gsmSendSms(alertPhoneNumber, str);

if (gsmOnline)

{queSmsPush(alertPhoneNumber, str);

sprintf(str, ">%s", alertPhoneNumber);

lcdTextLog(str);}}

temp = ds18b20GetTemperature();

ds18b20StartConversion();

if (temp != 0x550)

{sprintf(str, "TEMP: %d", temp >> 4);}

else

{sprintf(str, "TEMP: ERROR");}

lcdTextLine(TEMP_TEXT_LINE, str);

if (gsmOnline && (GSM_OK ==
gsmGetSms(&sms)))

{sprintf(str, "<%s", sms.phone);

lcdTextLog(str);

if (0 == strncmp(sms.text, registerPass,
strlen(registerPass)))

{strcpy(alertPhoneNumber,
&sms.text[strlen(registerPass)]);

queSmsPush(alertPhoneNumber,
"REGISTERED");

sprintf(str, "#%s", alertPhoneNumber);

lcdTextLog(str);}

else

{if (temp != 0x550)

{sprintf(str, "SCALE=%d TEMP=%d", scale,
temp >> 4);}

else

{sprintf(str, "SCALE=%d TEMP=ERROR",
scale);}

queSmsPush(sms.phone, str);}}

if (gsmOnline && (timerGetSysTime() >
(lastSms + SMS_PERIOD)))

{if (NULL == pSms)

{pSms = queSmsPop();}

if (pSms != NULL)

{if (gsmSendSms(pSms->phone, pSms->text))

{pSms = NULL;}}}gFirst = 0;}}

#define SCALE_INTERVAL_UNITS 500

uint8_t scaleVariationExceeded(uint16_t
scale)

{static uint16_t lastInterval = 0;

uint16_t interval = 1;uint8_t status = 0;

while (interval < (4096 /
SCALE_INTERVAL_UNITS))

{if (scale > (interval *
SCALE_INTERVAL_UNITS))

{interval++;}

else

{if (interval != lastInterval)

{lastInterval = interval;

status = 1;}break;}}

return status;}

void queSmsPush(char* pPhone, char* pText)

{strcpy(&gSmsQue.sms[gSmsQue.writeIndex].p
hone, pPhone);

strcpy(&gSmsQue.sms[gSmsQue.writeIndex].te
xt, pText);

if (SMS_QUE_SIZE == ++gSmsQue.writeIndex)

{gSmsQue.writeIndex = 0;}

gSmsQue.count++;

if (gSmsQue.count > SMS_QUE_SIZE)

{gSmsQue.count = SMS_QUE_SIZE;

if (SMS_QUE_SIZE == ++gSmsQue.readIndex)

{gSmsQue.readIndex = 0;}}}
```

```
SmsStructType* queSmsPop(void)
{SmsStructType* pSms = NULL;
if (gSmsQue.count)
{pSms = &gSmsQue.sms[gSmsQue.readIndex];
if (SMS_QUE_SIZE == ++gSmsQue.readIndex)
{gSmsQue.readIndex = 0;}
gSmsQue.count--;}
return pSms;}
#pragma GCC diagnostic pop
```

GSM

```
#include "stdlib.h"
#include "stdio.h"
#include "string.h"
#include "stddef.h"
#include "gsm.h"
#include "timer.h"
#include "uart.h"
#include "diag/Trace.h"
#include "led.h"
#define SMS_INDEX 0
#define SMS_STAT 1
#define SMS_PHONE 2
#define SMS_DATE 3
#define CHAR_CTRL_Z 0x1A
static SmsStructType gSms;
typedef struct
{char* pOutStr;
char* pParameter;
char* pExpectedStr;
GsmStatusEnumType error;
uint16_t timeout;
GsmStatusEnumType(*pCallBack)(char*pStr);}
AtCmdStructType;
static AtCmdStructType AT_CMD_ATE0 =
{"ATE0\r",NULL,"OK",GSM_ERROR_ATE0,500,
NULL};
static AtCmdStructType AT_CMD_CREG =
{"AT+CREG?\r",NULL,"+CREG: 0,1",
GSM_ERROR_CREG,500,NULL};
static AtCmdStructType AT_CMD_CMGF =
{"AT+CMGF=1\r",NULL,"OK",
GSM_ERROR_CMGF,500,NULL};
static GsmStatusEnumType
atCmdCmglCallback(char* pStr);
static AtCmdStructType AT_CMD_CMGL =
{"AT+CMGL=\"ALL\"\r",NULL,"+CMGL: ",
GSM_ERROR_CMGL,1000,&atCmdCmglCallback};
static AtCmdStructType AT_CMD_CMGD =
{"AT+CMGD=",NULL,"OK",
GSM_ERROR_CMGD,1000,NULL};
static AtCmdStructType AT_CMD_CMGS =
{"AT+CMGS=",NULL,"OK",
GSM_ERROR_CMGS,1000,NULL};
#define GSM_BUF_MAX_LEN 254
static char gGsmBuf[GSM_BUF_MAX_LEN];
#define AT_CMD_RESPONSE_MAX_PARAM 8
static char* gpAtCmdParamArray
[AT_CMD_RESPONSE_MAX_PARAM];
static uint32_t gGsmTimeoutVar = 0;
static GsmStatusEnumType
gsmAtCmd(AtCmdStructType* pAtCmd);
static void gsmTimeoutSet(uint32_t
timeout)
{gGsmTimeoutVar = timerGetSysTime() +
timeout;}
static GsmStatusEnumType gsmTimeout(void)
{return (timerGetSysTime() >
gGsmTimeoutVar);}
static GsmStatusEnumType
gsmAtCmd(AtCmdStructType* pAtCmd)
{GsmStatusEnumType status = pAtCmd->error;
gsmTimeoutSet(pAtCmd->timeout);
uartReset();
//memset
(gGsmBuf, 0, GSM_BUF_MAX_LEN);
      do
```

```
{if (strlen(pAtCmd->pOutStr) !=
uartWrite(pAtCmd->pOutStr)){break;}

       if (pAtCmd->pParameter != NULL)

{if (strlen(pAtCmd->pParameter) !=
uartWrite(pAtCmd->pParameter)){break;}}

       while (!gsmTimeout())

{if (uartRead(gGsmBuf, GSM_BUF_MAX_LEN))

{      if (0 == strncmp(gGsmBuf, pAtCmd-
>pExpectedStr,

       strlen(pAtCmd->pExpectedStr)))

{if (pAtCmd->pCallBack != NULL)

{status = pAtCmd->pCallBack(gGsmBuf);}
       else

{status = GSM_OK;}break;}}}}

       while (0);

       return status;}

GsmStatusEnumType gsmInit(void)

{GsmStatusEnumType status = GSM_OK;

       do

{//if ((status = gsmAtCmd(&AT_CMD_ATE0))
!= GSM_OK)//{break;}

       if ((status =
gsmAtCmd(&AT_CMD_CREG)) != GSM_OK)

{break;}

       if ((status =
gsmAtCmd(&AT_CMD_CMGF)) != GSM_OK)

{break;}}

       while (0);

       return status;}

static void atCmdExtractParameters(char*
pStr)

{char* pChar = NULL;

char* pStrOld = pStr;

uint8_t paramIndex = 0;

       for (paramIndex = 0; paramIndex <
AT_CMD_RESPONSE_MAX_PARAM; paramIndex++)

{gpAtCmdParamArray[paramIndex] = NULL;}

paramIndex = AT_CMD_RESPONSE_MAX_PARAM;

       for (paramIndex = 0; paramIndex <
AT_CMD_RESPONSE_MAX_PARAM; paramIndex++)

{pChar = strstr(pStr, ",\"");

       if (NULL == pChar)

{gpAtCmdParamArray[paramIndex] = pStrOld;

break;}

pStr = pChar + 2;

*pChar = '\0';

gpAtCmdParamArray[paramIndex] = pStrOld;

pStrOld = pStr;}}

static GsmStatusEnumType
atCmdCmglCallback(char* pStr)

{GsmStatusEnumType status = GSM_EROR_CMGL;

atCmdExtractParameters(&pStr[strlen(AT_CMD
_CMGL.pExpectedStr)]);

       do

{if (NULL == gpAtCmdParamArray[SMS_INDEX])

{break;}

strcpy(gSms.index,
gpAtCmdParamArray[SMS_INDEX]);

if (NULL == gpAtCmdParamArray[SMS_PHONE])

{break;}

strncpy(gSms.phone,
gpAtCmdParamArray[SMS_PHONE], 12);

if (NULL == gpAtCmdParamArray[SMS_DATE])

{break;}

       while (!gsmTimeout())

{if (uartRead(gSms.text, GSM_BUF_MAX_LEN))

{status = GSM_OK;break;}}}

       while (0);

       return status;}

GsmStatusEnumType gsmGetSms(SmsStructType*
pSms)

{GsmStatusEnumType status = GSM_OK;

while (GSM_OK == status)

{status = gsmAtCmd(&AT_CMD_CMGL);

       if (GSM_OK == status)

{//convert the new text

memcpy(pSms, &gSms,
sizeof(SmsStructType));

//delete the last processed sms

       if (strlen(gSms.index))
```

```
{strcat(gSms.index, "\r");

AT_CMD_CMGD.pParameter = gSms.index;

timerSleep(100);

gsmAtCmd(&AT_CMD_CMGD);}break;}}

return status;}

GsmStatusEnumType gsmSendSms(char*
pSmsNumber, char* pSmsStr)

{GsmStatusEnumType status =
AT_CMD_CMGS.error;

       do

{if ((NULL == pSmsNumber) || (NULL ==
pSmsStr) || (0 == strlen(pSmsNumber)))

{break;}

sprintf(gGsmBuf, "\"%s\"\r%s%c",
pSmsNumber, pSmsStr, CHAR_CTRL_Z);

AT_CMD_CMGS.pParameter = gGsmBuf;

status = gsmAtCmd(&AT_CMD_CMGS);}

       while (0);

       return status;}
```

LED

```
#include "stm32f407xx.h"

#include "stm32f4xx_hal.h"

#include "stm32f4xx.h"

#include "led.h"

//#define LED_GREEN_PIN GPIO_PIN_12

//#define LED_ORANGE_PIN GPIO_PIN_13

//#define LED_RED_PIN GPIO_PIN_14

//#define LED_BLUE_PIN GPIO_PIN_15

void ledInit(void)

{GPIO_InitTypeDef gpioInitStructure;

__GPIOD_CLK_ENABLE();

gpioInitStructure.Pull=GPIO_NOPULL;

gpioInitStructure.Pin = LED_GREEN |
LED_ORANGE | LED_RED | LED_BLUE;

gpioInitStructure.Mode =
GPIO_MODE_OUTPUT_PP;

gpioInitStructure.Speed = GPIO_SPEED_HIGH;

HAL_GPIO_Init(GPIOD, &gpioInitStructure);

ledOff(LED_GREEN | LED_ORANGE | LED_RED |
LED_BLUE);}

void ledOn(LedEnumType led)

{HAL_GPIO_WritePin(GPIOD, led,
GPIO_PIN_SET);}

void ledOff(LedEnumType led)

{HAL_GPIO_WritePin(GPIOD, led,
GPIO_PIN_RESET);}

void ledToggle(LedEnumType led)

{HAL_GPIO_TogglePin(GPIOD, led);}
```

LCD

```
#include "stm32f4xx_hal.h"

#include "stm32f4xx_hal_conf.h"

#include "lcd.h"

#include "led.h"

#include "timer.h"

const PixelStructType GSM_TEXT_LINE={0,0};

const PixelStructType TEMP_TEXT_LINE
={0,9};

const PixelStructType SCALE_TEXT_LINE
={0,18};

const PixelStructType LOG1_TEXT_LINE
={0,31};

const PixelStructType LOG2_TEXT_LINE
={0,40};

const PixelStructType STATUS_LINE ={0,27};

uint8_t
PCD8544_Buffer[PCD8544_BUFFER_SIZE];

uint8_t PCD8544_UpdateXmin=0,
PCD8544_UpdateXmax=0,

PCD8544_UpdateYmin=0,PCD8544_UpdateYmax=0;

uint8_t PCD8544_x;uint8_t PCD8544_y;

SPI_HandleTypeDef SPI_InitStructure;

//Fonts 5x7

const uint8_t PCD8544_Font5x7
[97][PCD8544_CHAR5x7_WIDTH] = {

{ 0x00, 0x00, 0x00, 0x00, 0x00 },// sp

{ 0x00, 0x00, 0x2f, 0x00, 0x00 },// !

{ 0x00, 0x07, 0x00, 0x07, 0x00 },// "

{ 0x14, 0x7f, 0x14, 0x7f, 0x14 },// #

{ 0x24, 0x2a, 0x7f, 0x2a, 0x12 },// $

{ 0x32, 0x34, 0x08, 0x16, 0x26 },// %

{ 0x36, 0x49, 0x55, 0x22, 0x50 },// &
```

```
{ 0x00, 0x05, 0x03, 0x00, 0x00 },// '
{ 0x00, 0x1c, 0x22, 0x41, 0x00 },// (
{ 0x00, 0x41, 0x22, 0x1c, 0x00 },// )
{ 0x14, 0x08, 0x3E, 0x08, 0x14 },// *
{ 0x08, 0x08, 0x3E, 0x08, 0x08 },// +
{ 0x00, 0x00, 0x50, 0x30, 0x00 },// ,
{ 0x10, 0x10, 0x10, 0x10, 0x10 },// -
{ 0x00, 0x60, 0x60, 0x00, 0x00 },// .
{ 0x20, 0x10, 0x08, 0x04, 0x02 },// /
{ 0x3E, 0x51, 0x49, 0x45, 0x3E },// 0
{ 0x00, 0x42, 0x7F, 0x40, 0x00 },// 1
{ 0x42, 0x61, 0x51, 0x49, 0x46 },// 2
{ 0x21, 0x41, 0x45, 0x4B, 0x31 },// 3
{ 0x18, 0x14, 0x12, 0x7F, 0x10 },// 4
{ 0x27, 0x45, 0x45, 0x45, 0x39 },// 5
{ 0x3C, 0x4A, 0x49, 0x49, 0x30 },// 6
{ 0x01, 0x71, 0x09, 0x05, 0x03 },// 7
{ 0x36, 0x49, 0x49, 0x49, 0x36 },// 8
{ 0x06, 0x49, 0x49, 0x29, 0x1E },// 9
{ 0x00, 0x36, 0x36, 0x00, 0x00 },// :
{ 0x00, 0x56, 0x36, 0x00, 0x00 },// ;
{ 0x08, 0x14, 0x22, 0x41, 0x00 },// <
{ 0x14, 0x14, 0x14, 0x14, 0x14 },// =
{ 0x00, 0x41, 0x22, 0x14, 0x08 },// >
{ 0x02, 0x01, 0x51, 0x09, 0x06 },// ?
{ 0x32, 0x49, 0x59, 0x51, 0x3E },// @
{ 0x7E, 0x11, 0x11, 0x11, 0x7E },// A
{ 0x7F, 0x49, 0x49, 0x49, 0x36 },// B
{ 0x3E, 0x41, 0x41, 0x41, 0x22 },// C
{ 0x7F, 0x41, 0x41, 0x22, 0x1C },// D
{ 0x7F, 0x49, 0x49, 0x49, 0x41 },// E
{ 0x7F, 0x09, 0x09, 0x09, 0x01 },// F
{ 0x3E, 0x41, 0x49, 0x49, 0x7A },// G
{ 0x7F, 0x08, 0x08, 0x08, 0x7F },// H
{ 0x00, 0x41, 0x7F, 0x41, 0x00 },// I
{ 0x20, 0x40, 0x41, 0x3F, 0x01 },// J
{ 0x7F, 0x08, 0x14, 0x22, 0x41 },// K
{ 0x7F, 0x40, 0x40, 0x40, 0x40 },// L
{ 0x7F, 0x02, 0x0C, 0x02, 0x7F },// M
{ 0x7F, 0x04, 0x08, 0x10, 0x7F },// N
{ 0x3E, 0x41, 0x41, 0x41, 0x3E },// O
{ 0x7F, 0x09, 0x09, 0x09, 0x06 },// P
{ 0x3E, 0x41, 0x51, 0x21, 0x5E },// Q
{ 0x7F, 0x09, 0x19, 0x29, 0x46 },// R
{ 0x46, 0x49, 0x49, 0x49, 0x31 },// S
{ 0x01, 0x01, 0x7F, 0x01, 0x01 },// T
{ 0x3F, 0x40, 0x40, 0x40, 0x3F },// U
{ 0x1F, 0x20, 0x40, 0x20, 0x1F },// V
{ 0x3F, 0x40, 0x38, 0x40, 0x3F },// W
{ 0x63, 0x14, 0x08, 0x14, 0x63 },// X
{ 0x07, 0x08, 0x70, 0x08, 0x07 },// Y
{ 0x61, 0x51, 0x49, 0x45, 0x43 },// Z
{ 0x00, 0x7F, 0x41, 0x41, 0x00 },// [
{ 0x55, 0x2A, 0x55, 0x2A, 0x55 },// 55
{ 0x00, 0x41, 0x41, 0x7F, 0x00 },// ]
{ 0x04, 0x02, 0x01, 0x02, 0x04 },// ^
{ 0x40, 0x40, 0x40, 0x40, 0x40 },// _
{ 0x00, 0x01, 0x02, 0x04, 0x00 },// '
{ 0x20, 0x54, 0x54, 0x54, 0x78 },// a
{ 0x7F, 0x48, 0x44, 0x44, 0x38 },// b
{ 0x38, 0x44, 0x44, 0x44, 0x20 },// c
{ 0x38, 0x44, 0x44, 0x48, 0x7F },// d
{ 0x38, 0x54, 0x54, 0x54, 0x18 },// e
{ 0x08, 0x7E, 0x09, 0x01, 0x02 },// f
{ 0x0C, 0x52, 0x52, 0x52, 0x3E },// g
{ 0x7F, 0x08, 0x04, 0x04, 0x78 },// h
{ 0x00, 0x44, 0x7D, 0x40, 0x00 },// i
{ 0x20, 0x40, 0x44, 0x3D, 0x00 },// j
{ 0x7F, 0x10, 0x28, 0x44, 0x00 },// k
{ 0x00, 0x41, 0x7F, 0x40, 0x00 },// l
{ 0x7C, 0x04, 0x18, 0x04, 0x78 },// m
{ 0x7C, 0x08, 0x04, 0x04, 0x78 },// n
{ 0x38, 0x44, 0x44, 0x44, 0x38 },// o
{ 0x7C, 0x14, 0x14, 0x14, 0x08 },// p
```

```
{ 0x08, 0x14, 0x14, 0x18, 0x7C },// q
{ 0x7C, 0x08, 0x04, 0x04, 0x08 },// r
{ 0x48, 0x54, 0x54, 0x54, 0x20 },// s
{ 0x04, 0x3F, 0x44, 0x40, 0x20 },// t
{ 0x3C, 0x40, 0x40, 0x20, 0x7C },// u
{ 0x1C, 0x20, 0x40, 0x20, 0x1C },// v
{ 0x3C, 0x40, 0x30, 0x40, 0x3C },// w
{ 0x44, 0x28, 0x10, 0x28, 0x44 },// x
{ 0x0C, 0x50, 0x50, 0x50, 0x3C },// y
{ 0x44, 0x64, 0x54, 0x4C, 0x44 },// z
{ 0x00, 0x7F, 0x3E, 0x1C, 0x08 },// >
Filled123
{ 0x08, 0x1C, 0x3E, 0x7F, 0x00 },// <
Filled124
{ 0x08, 0x7C, 0x7E, 0x7C, 0x08 },// Arrow
up125
{ 0x10, 0x3E, 0x7E, 0x3E, 0x10 },// Arrow
down126
{ 0x3E, 0x3E, 0x3E, 0x3E, 0x3E },//Stop127
{ 0x00, 0x7F, 0x3E, 0x1C, 0x08 },//Play128
};
const uint8_t PCD8544_Font3x5[106][3] = {
{ 0x00, 0x00, 0x00 },// sp - 32
{ 0x00, 0x17, 0x00 },// ! - 33
{ 0x03, 0x00, 0x03 },// " - 34
{ 0x1F, 0x0A, 0x1F },// # - 35
{ 0x0A, 0x1F, 0x05 },// $
{ 0x09, 0x04, 0x12 },// %
{ 0x0F, 0x17, 0x1C },// &
{ 0x00, 0x03, 0x00 },// '
{ 0x00, 0x0E, 0x11 },// ( - 40
{ 0x11, 0x0E, 0x00 },// )
{ 0x05, 0x02, 0x05 },// *
{ 0x04, 0x0E, 0x04 },// +
{ 0x10, 0x08, 0x00 },// ,
{ 0x04, 0x04, 0x04 },// - - 45
{ 0x00, 0x10, 0x00 },// .
{ 0x08, 0x04, 0x02 },// /
{ 0x1F, 0x11, 0x1F },// 0
{ 0x12, 0x1F, 0x10 },// 1
{ 0x1D, 0x15, 0x17 },// 2 - 50
{ 0x11, 0x15, 0x1F },// 3
{ 0x07, 0x04, 0x1F },// 4
{ 0x17, 0x15, 0x1D },// 5
{ 0x1F, 0x15, 0x1D },// 6
{ 0x01, 0x01, 0x1F },// 7 - 55
{ 0x1F, 0x15, 0x1F },// 8
{ 0x17, 0x15, 0x1F },// 9 - 57
{ 0x00, 0x0A, 0x00 },// :
{ 0x10, 0x0A, 0x00 },// ;
{ 0x04, 0x0A, 0x11 },// < - 60
{ 0x0A, 0x0A, 0x0A },// =
{ 0x11, 0x0A, 0x04 },// >
{ 0x01, 0x15, 0x03 },// ?
{ 0x0E, 0x15, 0x16 },// @
{ 0x1E, 0x05, 0x1E },// A - 65
{ 0x1F, 0x15, 0x0A },// B
{ 0x0E, 0x11, 0x11 },// C
{ 0x1F, 0x11, 0x0E },// D
{ 0x1F, 0x15, 0x15 },// E
{ 0x1F, 0x05, 0x05 },// F - 70
{ 0x0E, 0x15, 0x1D },// G
{ 0x1F, 0x04, 0x1F },// H
{ 0x11, 0x1F, 0x11 },// I
{ 0x08, 0x10, 0x0F },// J
{ 0x1F, 0x04, 0x1B },// K - 75
{ 0x1F, 0x10, 0x10 },// L
{ 0x1F, 0x06, 0x1F },// M
{ 0x1F, 0x0E, 0x1F },// N
{ 0x0E, 0x11, 0x0E },// O
{ 0x1F, 0x05, 0x02 },// P - 80
{ 0x0E, 0x11, 0x1E },// Q
{ 0x1F, 0x0D, 0x16 },// R
{ 0x12, 0x15, 0x09 },// S
{ 0x01, 0x1F, 0x01 },// T
{ 0x0F, 0x10, 0x0F },// U - 85
```

```
{ 0x07, 0x18, 0x07 },// V
{ 0x1F, 0x0C, 0x1F },// W
{ 0x1B, 0x04, 0x1B },// X
{ 0x03, 0x1C, 0x03 },// Y
{ 0x19, 0x15, 0x13 },// Z - 90
{ 0x1F, 0x11, 0x00 },// [
{ 0x02, 0x04, 0x08 },// 55- backspace- 92
{ 0x00, 0x11, 0x1F },// ]
{ 0x02, 0x01, 0x02 },// ^
{ 0x10, 0x10, 0x10 },// _ - 95
{ 0x01, 0x02, 0x00 },// `
{ 0x1A, 0x16, 0x1C },// a
{ 0x1F, 0x12, 0x0C },// b
{ 0x0C, 0x12, 0x12 },// c
{ 0x0C, 0x12, 0x1F },// d - 100
{ 0x0C, 0x1A, 0x16 },// e
{ 0x04, 0x1E, 0x05 },// f
{ 0x06, 0x15, 0x0F },// g
{ 0x1F, 0x02, 0x1C },// h
{ 0x00, 0x1D, 0x00 },// i - 105
{ 0x10, 0x10, 0x0D },// j
{ 0x1F, 0x0C, 0x12 },// k
{ 0x11, 0x1F, 0x10 },// l
{ 0x1E, 0x0E, 0x1E },// m
{ 0x1E, 0x02, 0x1C },// n - 110
{ 0x0C, 0x12, 0x0C },// o
{ 0x1E, 0x0A, 0x04 },// p
{ 0x04, 0x0A, 0x1E },// q
{ 0x1C, 0x02, 0x02 },// r
{ 0x14, 0x1E, 0x0A },// s - 115
{ 0x02, 0x1F, 0x12 },// t
{ 0x0E, 0x10, 0x1E },// u
{ 0x0E, 0x10, 0x0E },// v
{ 0x1E, 0x1C, 0x1E },// w
{ 0x12, 0x0C, 0x12 },// x - 120
{ 0x02, 0x14, 0x1E },// y
{ 0x1A, 0x1E, 0x16 },// z
{ 0x04, 0x1B, 0x11 },// {
{ 0x00, 0x1F, 0x00 },// |
{ 0x11, 0x1B, 0x04 },// }
{ 0x04, 0x06, 0x02 },// ~
{ 0x1F, 0x1F, 0x1F },// delete};

void HAL_SPI_MspInit(SPI_HandleTypeDef
*hspi)

{GPIO_InitTypeDef GPIO_InitStructure;

__SPI2_CLK_ENABLE();

__GPIOB_CLK_ENABLE();

__SYSCFG_CLK_ENABLE();

if (hspi->Instance == SPI2)

{GPIO_InitStructure.Speed =
GPIO_SPEED_HIGH; //GPIO_Speed_50MHz;

//GPIO_InitStructure.GPIO_OType =
GPIO_OType_PP;

//init spi sclk

GPIO_InitStructure.Pin = GPIO_PIN_13;

GPIO_InitStructure.Mode =
GPIO_MODE_AF_PP;//GPIO_Mode_AF;

GPIO_InitStructure.Alternate =
GPIO_AF5_SPI2; // GPIO_PinAFConfig(GPIOB,
GPIO_PinSource13,  GPIO_AF_SPI2);

GPIO_InitStructure.Pull = GPIO_NOPULL;

HAL_GPIO_Init(GPIOB, &GPIO_InitStructure);

//init spi mosi

      GPIO_InitStructure.Pin =
GPIO_PIN_15;

GPIO_InitStructure.Mode =
GPIO_MODE_AF_PP;//GPIO_Mode_AF;

GPIO_InitStructure.Alternate =
GPIO_AF5_SPI2; //GPIO_PinAFConfig(GPIOB,
GPIO_PinSource15, GPIO_AF_SPI2);

HAL_GPIO_Init(GPIOB, &GPIO_InitStructure);

//init spi miso

GPIO_InitStructure.Pin = GPIO_PIN_14;

GPIO_InitStructure.Mode =
GPIO_MODE_AF_PP;//GPIO_Mode_AF;

GPIO_InitStructure.Alternate =
GPIO_AF5_SPI2;   //GPIO_PinAFConfig(GPIOB,
GPIO_PinSource14, GPIO_AF_SPI2);

HAL_GPIO_Init(GPIOB,&GPIO_InitStructure;}}

void lcdInitIO(void)
```

```
{GPIO_InitTypeDef GPIO_InitStructure;

__GPIOB_CLK_ENABLE();

//init CS

GPIO_InitStructure.Pin = PCD8544_CE_PIN;

GPIO_InitStructure.Speed= GPIO_SPEED_HIGH;

GPIO_InitStructure.Mode=GPIO_MODE_OUTPUT_P
P;

GPIO_InitStructure.Pull = GPIO_PULLUP;

HAL_GPIO_Init(PCD8544_CE_PORT,

&GPIO_InitStructure);

//init DC

GPIO_InitStructure.Pin = PCD8544_DC_PIN;

GPIO_InitStructure.Speed =GPIO_SPEED_HIGH;

GPIO_InitStructure.Mode
=GPIO_MODE_OUTPUT_PP;

GPIO_InitStructure.Pull = GPIO_PULLUP;

HAL_GPIO_Init(PCD8544_DC_PORT,

&GPIO_InitStructure);

//init RST

GPIO_InitStructure.Pin = PCD8544_RST_PIN;

GPIO_InitStructure.Speed= GPIO_SPEED_HIGH;

GPIO_InitStructure.Mode

=GPIO_MODE_OUTPUT_PP;

GPIO_InitStructure.Pull = GPIO_PULLUP;

HAL_GPIO_Init(PCD8544_RST_PORT,
&GPIO_InitStructure);

//RST HIGH

HAL_GPIO_WritePin(PCD8544_RST_PORT,
PCD8544_RST_PIN, GPIO_PIN_SET);

//CE HIGH

HAL_GPIO_WritePin(PCD8544_CE_PORT,
PCD8544_CE_PIN, GPIO_PIN_SET);

//Initialize SPI2

SPI_InitStructure.Instance = SPI2;

SPI_InitStructure.Init.Direction =
SPI_DIRECTION_2LINES;//SPI_Direction_2Line
s_FullDuplex;

SPI_InitStructure.Init.Mode =
SPI_MODE_MASTER; //  SPI_Mode_Master;

SPI_InitStructure.Init.DataSize =
SPI_DATASIZE_8BIT; // SPI_DataSize_8b;

SPI_InitStructure.Init.CLKPolarity =
SPI_POLARITY_LOW;//High;

SPI_InitStructure.Init.CLKPhase =
SPI_PHASE_1EDGE;

SPI_InitStructure.Init.NSS = SPI_NSS_SOFT;


SPI_InitStructure.Init.BaudRatePrescaler =
SPI_BAUDRATEPRESCALER_256;

SPI_InitStructure.Init.FirstBit =
SPI_FIRSTBIT_MSB;

SPI_InitStructure.Init.CRCPolynomial = 1;

HAL_SPI_Init( &SPI_InitStructure);

__HAL_SPI_ENABLE(&SPI_InitStructure);}

void lcdSend(uint8_t data)

{uint8_t tmp = 0;

HAL_GPIO_WritePin(PCD8544_CE_PORT,
PCD8544_CE_PIN, GPIO_PIN_RESET);

HAL_SPI_TransmitReceive(&SPI_InitStructur,
&data, &tmp, 1, 100);

HAL_GPIO_WritePin(PCD8544_CE_PORT,
PCD8544_CE_PIN, GPIO_PIN_SET);}

void lcdPin(PCD8544_Pin_t pin,
PCD8544_State_t state)

{switch (pin)

{case PCD8544_Pin_DC:

if (state != PCD8544_State_Low)

{HAL_GPIO_WritePin(PCD8544_DC_PORT,
PCD8544_DC_PIN, GPIO_PIN_SET);

}      else

{HAL_GPIO_WritePin(PCD8544_DC_PORT,
PCD8544_DC_PIN, GPIO_PIN_RESET);}break;

       case PCD8544_Pin_RST:

       if (state != PCD8544_State_Low)

{HAL_GPIO_WritePin(PCD8544_RST_PORT,
PCD8544_RST_PIN, GPIO_PIN_SET);

}      else

{HAL_GPIO_WritePin(PCD8544_RST_PORT,
PCD8544_RST_PIN,
GPIO_PIN_RESET);}break;default: break;}}

void lcdDelay(uint32_t micros)

{volatile uint32_t i;

       for (i = 0; i < micros; i++){}}

void lcdInit(uint8_t contrast)
```

```
{//Initialize IO's

lcdInitIO();

//Reset

timerSleep(1);

lcdPin(PCD8544_Pin_RST,
PCD8544_State_Low);

timerSleep(1);

lcdPin(PCD8544_Pin_RST,
PCD8544_State_High);

timerSleep(1);

// Go in extended mode

lcdWrite(PCD8544_COMMAND,
PCD8544_FUNCTIONSET |
PCD8544_EXTENDEDINSTRUCTION);

// LCD bias select

lcdWrite(PCD8544_COMMAND, PCD8544_SETBIAS
| 0x4);

// set VOP

if (contrast > 0x7F)

{contrast = 0x3F;}

lcdWrite(PCD8544_COMMAND, PCD8544_SETVOP |
contrast);

       // normal mode

lcdWrite(PCD8544_COMMAND,
PCD8544_FUNCTIONSET);

       // Set display to Normal

lcdWrite(PCD8544_COMMAND,
PCD8544_DISPLAYCONTROL |
PCD8544_DISPLAYNORMAL);

       //Set cursor to home position

lcdHome();

       //Normal display

lcdWrite(PCD8544_COMMAND,
PCD8544_DISPLAYCONTROL |
PCD8544_DISPLAYNORMAL);

       //Clear display

lcdClear();

lcdDrawRectangle(STATUS_LINE.x,
STATUS_LINE.y, PCD8544_WIDTH - 1,

STATUS_LINE.y + 1, PCD8544_Pixel_Set);}

void lcdWrite(PCD8544_WriteType_t cd,
uint8_t data)

{switch (cd)

{//Send data to lcd's ram

       case PCD8544_DATA:

//Set DC pin HIGH

lcdPin(PCD8544_Pin_DC,
PCD8544_State_High);break;

       //Send command to lcd

case PCD8544_COMMAND:

       //Set DC pin LOW

lcdPin(PCD8544_Pin_DC, PCD8544_State_Low);

break;default: break;}

       //Send data

lcdSend(data);}

void lcdSetContrast(uint8_t contrast)

{// Go in extended mode

lcdWrite(PCD8544_COMMAND,
PCD8544_FUNCTIONSET |
PCD8544_EXTENDEDINSTRUCTION);

// set VOP

       if (contrast > 0x7F)

{contrast = 0x7F;}

lcdWrite(PCD8544_COMMAND, PCD8544_SETVOP |
contrast);

       // normal mode

lcdWrite(PCD8544_COMMAND,
PCD8544_FUNCTIONSET);}

void lcdDrawPixel(uint8_t x, uint8_t y,
PCD8544_Pixel_t pixel)

{if (x >= PCD8544_WIDTH){return;}

if (y >= PCD8544_HEIGHT){return;}

if (pixel != PCD8544_Pixel_Clear)

{PCD8544_Buffer[x + (y / 8) *
PCD8544_WIDTH] |= 1 << (y % 8);}

 else

{PCD8544_Buffer[x + (y / 8) *
PCD8544_WIDTH] &= ~(1 << (y % 8));}

lcdUpdateArea(x, y, x, y);}

void lcdInvert(PCD8544_Invert_t invert)

{if (invert != PCD8544_Invert_No)

{lcdWrite(PCD8544_COMMAND,
PCD8544_DISPLAYCONTROL |
PCD8544_DISPLAYINVERTED);}
```

```
 else

{lcdWrite(PCD8544_COMMAND,
PCD8544_DISPLAYCONTROL |
PCD8544_DISPLAYNORMAL);}}

void lcdClear(void)

{unsigned int i;

lcdHome();

       for (i = 0; i < PCD8544_BUFFER_SIZE;
i++)

{PCD8544_Buffer[i] = 0x00;

//PCD8544_Write(PCD8544_DATA, 0x00);}

lcdGotoXY(0, 0);

lcdUpdateArea(0, 0, PCD8544_WIDTH - 1,
PCD8544_HEIGHT - 1);

lcdRefresh();}

void lcdHome(void)

{lcdWrite(PCD8544_COMMAND,
PCD8544_SETXADDR | 0);

lcdWrite(PCD8544_COMMAND, PCD8544_SETYADDR
| 0);}

void lcdRefresh(void)

{uint8_t i, j;

       for (i = 0; i < 6; i++)

{//Not in range yet

if (PCD8544_UpdateYmin > ((i + 1) * 8))

{continue;}

       //Over range, stop

if ((i * 8) > PCD8544_UpdateYmax)

{break;}

lcdWrite(PCD8544_COMMAND, PCD8544_SETYADDR
| i);

lcdWrite(PCD8544_COMMAND, PCD8544_SETXADDR
| PCD8544_UpdateXmin);

for (j = PCD8544_UpdateXmin; j <=
PCD8544_UpdateXmax; j++)

{lcdWrite(PCD8544_DATA, PCD8544_Buffer[(i
* PCD8544_WIDTH) + j]);}}

PCD8544_UpdateXmin = PCD8544_WIDTH - 1;

PCD8544_UpdateXmax = 0;

PCD8544_UpdateYmin = PCD8544_HEIGHT - 1;

PCD8544_UpdateYmax = 0;}

void lcdUpdateArea(uint8_t xMin, uint8_t
yMin, uint8_t xMax, uint8_t yMax)

{if (xMin < PCD8544_UpdateXmin)

{PCD8544_UpdateXmin = xMin;}

if (xMax > PCD8544_UpdateXmax)

{PCD8544_UpdateXmax = xMax;}

if (yMin < PCD8544_UpdateYmin)

{PCD8544_UpdateYmin = yMin;}

if (yMax > PCD8544_UpdateYmax)

{PCD8544_UpdateYmax = yMax;}}

void lcdGotoXY(uint8_t x, uint8_t y)

{PCD8544_x = x;

PCD8544_y = y;}

void lcdPutc(char c, PCD8544_Pixel_t
color, PCD8544_FontSize_t size)

{uint8_t c_height = 0;

uint8_t c_width = 0;

uint8_t i = 0;

uint8_t b = 0;

uint8_t j = 0;

       if (size == PCD8544_FontSize_3x5)

{c_width = PCD8544_CHAR3x5_WIDTH;

c_height = PCD8544_CHAR3x5_HEIGHT;

}else

{c_width = PCD8544_CHAR5x7_WIDTH;

c_height = PCD8544_CHAR5x7_HEIGHT;}

if ((PCD8544_x + c_width) > PCD8544_WIDTH)

{//If at the end of a line of display, go
to new line and set x to 0 position

PCD8544_y += c_height;

PCD8544_x = 0;}

       for (i = 0; i < c_width - 1; i++)

{if (c < 32)

{//b=_custom_chars[_font_size][(uint8_t)ch
r][i];}

       else if (size ==
PCD8544_FontSize_3x5)

{b = PCD8544_Font3x5[c - 32][i];}

       else
```

```
{b = PCD8544_Font5x7[c - 32][i];}

if (b == 0x00 && (c != 0 && c != 32))

//if (c != 0 && c != 32){continue;}

       for (j = 0; j < c_height; j++)

{if (color == PCD8544_Pixel_Set)

{lcdDrawPixel(PCD8544_x, (PCD8544_y + j),
((b >> j) & 1) ? PCD8544_Pixel_Set :
PCD8544_Pixel_Clear);}

else

{lcdDrawPixel(PCD8544_x, (PCD8544_y + j),
((b >> j) & 1) ? PCD8544_Pixel_Clear :
PCD8544_Pixel_Set);}}

PCD8544_x++;}

PCD8544_x++;}

void lcdPuts(char *c, PCD8544_Pixel_t
color, PCD8544_FontSize_t size)

{while (*c)

{lcdPutc(*c++, color, size);}}

void lcdDrawLine(uint8_t x0, uint8_t y0,
uint8_t x1, uint8_t y1, PCD8544_Pixel_t
color)

{uint16_t dx, dy;

uint16_t temp;

if (x0 > x1)

{temp = x1;x1 = x0;x0 = temp;}

if (y0 > y1)

{temp = y1;y1 = y0;y0 = temp;}

dx = x1 - x0;dy = y1 - y0;

       if (dx == 0)

{do

{lcdDrawPixel(x0, y0, color);y0++;}

 while (y1 >= y0);return;}

       if (dy == 0)

{do

{lcdDrawPixel(x0, y0, color);x0++;}

 while (x1 >= x0);return;}

/* Based on Bresenham's line algorithm  */

       if (dx > dy)

{temp = 2 * dy - dx;

while (x0 != x1)

{lcdDrawPixel(x0, y0, color);x0++;

       if (temp > 0)

{y0++;temp += 2 * dy - 2 * dx;}

else

{temp += 2 * dy;}}

lcdDrawPixel(x0, y0, color);}

       else

{temp = 2 * dx - dy; while (y0 != y1)

{lcdDrawPixel(x0, y0, color); y0++;

       if (temp > 0)

{x0++; temp += 2 * dy - 2 * dx;}

       else

{temp += 2 * dy;}}

lcdDrawPixel(x0, y0, color);}}

void lcdDrawRectangle(uint8_t x0, uint8_t
y0, uint8_t x1, uint8_t y1,
PCD8544_Pixel_t color)

{lcdDrawLine(x0, y0, x1, y0, color);
       //Top

lcdDrawLine(x0, y0, x0, y1, color);
       //Left

lcdDrawLine(x1, y0, x1, y1, color);
       //Right

lcdDrawLine(x0, y1, x1, y1, color);
       //Bottom

}

void lcdDrawFilledRectangle(uint8_t x0,
uint8_t y0, uint8_t x1, uint8_t y1,
PCD8544_Pixel_t color)

{for (; y0 < y1; y0++)

{lcdDrawLine(x0, y0, x1, y0, color);}}

void lcdDrawCircle(uint8_t x0, uint8_t y0,
uint8_t r, PCD8544_Pixel_t color)

{int16_t f = 1 - r;

int16_t ddF_x = 1;

int16_t ddF_y = -2 * r;

int16_t x = 0;

int16_t y = r;

lcdDrawPixel(x0, y0 + r, color);

lcdDrawPixel(x0, y0 - r, color);

lcdDrawPixel(x0 + r, y0, color);
```

```
lcdDrawPixel(x0 - r, y0, color);

    while (x < y)

{if (f >= 0)

{  y--; ddF_y += 2; f += ddF_y;}

x++;ddF_x += 2;f += ddF_x;

lcdDrawPixel(x0 + x, y0 + y, color);

lcdDrawPixel(x0 - x, y0 + y, color);

lcdDrawPixel(x0 + x, y0 - y, color);

lcdDrawPixel(x0 - x, y0 - y, color);

lcdDrawPixel(x0 + y, y0 + x, color);

lcdDrawPixel(x0 - y, y0 + x, color);

lcdDrawPixel(x0 + y, y0 - x, color);

lcdDrawPixel(x0 - y, y0 - x, color);}}

void lcdDrawFilledCircle(uint8_t x0,
uint8_t y0, uint8_t r, PCD8544_Pixel_t
color)

{int16_t f = 1 - r;

int16_t ddF_x = 1;

int16_t ddF_y = -2 * r;

int16_t x = 0;

int16_t y = r;

lcdDrawPixel(x0, y0 + r, color);

lcdDrawPixel(x0, y0 - r, color);

lcdDrawPixel(x0 + r, y0, color);

lcdDrawPixel(x0 - r, y0, color);

lcdDrawLine(x0 - r, y0, x0 + r, y0,
color);

    while (x < y)

{if (f >= 0)

{y--;ddF_y += 2;f += ddF_y;}

x++;ddF_x += 2;f += ddF_x;

lcdDrawLine(x0 - x, y0 + y, x0 + x, y0 +
y, color);

lcdDrawLine(x0 + x, y0 - y, x0 - x, y0 -
y, color);

lcdDrawLine(x0 + y, y0 + x, x0 - y, y0 +
x, color);

lcdDrawLine(x0 + y, y0 - x, x0 - y, y0 -
x, color);}}
```

```
void lcdTextLineClear(PixelStructType
textLine)

{uint8_t i = 0;

for (i = 0; i < PCD8544_CHAR5x7_HEIGHT;
i++)

{lcdDrawLine(textLine.x, textLine.y + i,
PCD8544_WIDTH - 1,

textLine.y + i, PCD8544_Pixel_Clear);}}

void lcdTextLine(PixelStructType textLine,
char* pStr)

{lcdTextLineClear(textLine);

lcdGotoXY(textLine.x, textLine.y);

lcdPuts(pStr, PCD8544_Pixel_Set,
PCD8544_FontSize_5x7);

lcdRefresh();}

#define TEXT_LOG_SIZE 14

char gTextLog1[TEXT_LOG_SIZE];

char gTextLog2[TEXT_LOG_SIZE];

void lcdTextLog(char* pStr)

{strncpy(gTextLog1, gTextLog2,
TEXT_LOG_SIZE);

strncpy(gTextLog2, pStr, TEXT_LOG_SIZE);

lcdTextLine(LOG2_TEXT_LINE, gTextLog1);

lcdTextLine(LOG1_TEXT_LINE, gTextLog2);}
```

ADC

```
#include "adc.h"

#include "stm32f4xx.h"

#include "led.h"

#define SCALE_PORT GPIOA

#define SCALE_PIN GPIO_PIN_4

#define SCALE_ADC_CH 4

ADC_HandleTypeDef hADC;

#define AVG_SAMPLES 100

static uint16_t gScale = 0;

static uint32_t gAvgSum = 0;

static uint16_t gAvgIndex = 0;

static uint16_t gAvgSamples[AVG_SAMPLES];

void adcInit()

{GPIO_InitTypeDef GPIO_InitStructure;
```

```
__ADC1_CLK_ENABLE();

__GPIOA_CLK_ENABLE();

GPIO_InitStructure.Pin = SCALE_PIN;

GPIO_InitStructure.Pull = GPIO_NOPULL;

GPIO_InitStructure.Mode =
GPIO_MODE_ANALOG;

GPIO_InitStructure.Speed =
GPIO_SPEED_HIGH;

HAL_GPIO_Init(SCALE_PORT,
&GPIO_InitStructure);

hADC.Instance = ADC1;

hADC.Init.ClockPrescaler =
ADC_CLOCKPRESCALER_PCLK_DIV8;

hADC.Init.Resolution = ADC_RESOLUTION12b;

hADC.Init.DataAlign = ADC_DATAALIGN_RIGHT;

hADC.Init.ScanConvMode = DISABLE;

hADC.Init.EOCSelection = EOC_SINGLE_CONV;

hADC.Init.ContinuousConvMode = ENABLE;

hADC.Init.NbrOfConversion = 1;

hADC.Init.ExternalTrigConv =
ADC_EXTERNALTRIGCONVEDGE_NONE;

HAL_ADC_Init(&hADC);

ADC_ChannelConfTypeDef channel;

channel.Channel = SCALE_ADC_CH;

channel.Rank = 1;

channel.SamplingTime =
ADC_SAMPLETIME_480CYCLES;

HAL_ADC_ConfigChannel(&hADC, &channel);

      //Interrupt vector

HAL_NVIC_SetPriority(ADC_IRQn,0, 3);

HAL_NVIC_EnableIRQ(ADC_IRQn);

      //Start ADC Conversion

HAL_ADC_Start_IT(&hADC);}

void adcNewSample(void)

{HAL_ADC_Start_IT(&hADC);}

void
HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef
* hadc)

{gScale = HAL_ADC_GetValue(hadc);

HAL_ADC_Stop_IT(hadc);

HAL_ADC_Start_IT(hadc);

gAvgSum -= gAvgSamples[gAvgIndex];

gAvgSamples[gAvgIndex] = gScale;

gAvgSum += gScale;

if (++gAvgIndex == AVG_SAMPLES)

{gAvgIndex = 0;}

gScale = gAvgSum / AVG_SAMPLES;}

void
HAL_ADC_ErrorCallback(ADC_HandleTypeDef*
hadc)

{//HAL_ADC_Stop_IT(&hADC);

//gScale = ADC_ERROR;}

uint32_t adcReadScale(void)

{return gScale;}

void ADC_IRQHandler(void)

{HAL_ADC_IRQHandler(&hADC);}
```

UART

```
#include "stdlib.h"

#include "stddef.h"

#include "string.h"

#include "stddef.h"

#include "uart.h"

#include "timer.h"

#define UART_TX_FIFO_SIZE 64

#define UART_RX_FIFO_SIZE 254

static uint16_t gUartInitDone = 0;

static UART_HandleTypeDef gUSART2Handle;

typedef enum

{FIFO_OK = 0,FIFO_EMPTY = 1,FIFO_FULL=2,
FIFO_OVERRUN = 4}

FIFOStatusEnumType;

typedef struct

{uint16_t indexRead;uint16_t indexWrite;

uint16_t dataCount;uint16_t linesCount;

uint16_t status;uint16_t bufSize;

uint8_t* pData;}

FIFOStructType;

static volatile FIFOStructType gUartTxFIFO
={0, 0, 0, 0, 0, UART_TX_FIFO_SIZE, NULL};
```

```
static volatile FIFOStructType gUartRxFIFO
={0, 0, 0, 0, 0, UART_RX_FIFO_SIZE, NULL};

static inline void
INTERRUPTS_DISABLE(void)

{asm volatile ("cpsid i");}

static inline void INTERRUPTS_ENABLE(void)

{asm volatile ("cpsie i");}

static void fifoReset(volatile
FIFOStructType* pFifo)

{pFifo->indexRead = 0;pFifo->indexWrite=0;

pFifo->linesCount = 0;pFifo->dataCount= 0;

pFifo->status = 0;}

static uint16_t fifoIn

(volatile FIFOStructType* pFifo,
char*pDataIn,

uint16_t dataLen)

{INTERRUPTS_DISABLE();

uint16_t dataLenCopy = dataLen;

       do

{if (!gUartInitDone){break;}

if ((NULL == pDataIn) || (0 == dataLen))

{break;}

if (pFifo->status & FIFO_FULL){break;}

pFifo->status &= ~FIFO_EMPTY;

if (dataLen > (pFifo->bufSize - pFifo-
>dataCount))

{dataLen = pFifo->bufSize - pFifo-
>dataCount;

pFifo->status |= FIFO_OVERRUN;}

while (dataLen)

{dataLen--;

if ('\n' == *pDataIn){pDataIn++;continue;}

if ('\r' == *pDataIn){pFifo>linesCount++;}

pFifo->pData[pFifo->indexWrite]
=*pDataIn++;

if (pFifo->bufSize == ++pFifo->indexWrite)

{pFifo->indexWrite = 0;}

if(pFifo->bufSize == ++pFifo->dataCount)

{pFifo->status |= FIFO_FULL;break;}}}

while (0);

       INTERRUPTS_ENABLE();

return (dataLenCopy - dataLen);}

static uint16_t fifoOut(volatile
FIFOStructType* pFifo,

char* pDataOut,uint16_t dataLen)

{INTERRUPTS_DISABLE();

uint16_t dataLenCopy = dataLen;

       do

{if (!gUartInitDone){break;}

if ((NULL == pDataOut) || (0 == dataLen))
{break;}

if (pFifo->status & FIFO_EMPTY){break;}

pFifo->status &= ~FIFO_FULL;

       if (dataLen > pFifo->dataCount)

{dataLen = pFifo->dataCount;}

       while (dataLen)

{dataLen--;

*pDataOut++ = pFifo->pData[pFifo-
>indexRead];

if (('\r' == pFifo->pData[pFifo-
>indexRead]) && pFifo->linesCount)

{pFifo->linesCount--;}

if (pFifo->bufSize == ++pFifo->indexRead)

{pFifo->indexRead = 0;}

if (0 == --pFifo->dataCount)

{pFifo->status |= FIFO_EMPTY;break;}}}

while (0);

       INTERRUPTS_ENABLE();

return (dataLenCopy - dataLen);}

static void uartInitHW(void)

{GPIO_InitTypeDef gpioInitStruct;

if (!gUartInitDone)

{__GPIOA_CLK_ENABLE();

gpioInitStruct.Pin = GPIO_PIN_2;

gpioInitStruct.Mode = GPIO_MODE_AF_PP;

gpioInitStruct.Alternate=GPIO_AF7_USART2;

gpioInitStruct.Speed = GPIO_SPEED_HIGH;

gpioInitStruct.Pull = GPIO_NOPULL;

HAL_GPIO_Init(GPIOA, &gpioInitStruct);
```

```
gpioInitStruct.Pin = GPIO_PIN_3;

gpioInitStruct.Mode = GPIO_MODE_AF_PP;

gpioInitStruct.Alternate=GPIO_AF7_USART2;

gpioInitStruct.Speed = GPIO_SPEED_HIGH;

gpioInitStruct.Pull = GPIO_PULLUP;

HAL_GPIO_Init(GPIOA, &gpioInitStruct);

__USART2_CLK_ENABLE();

gUSART2Handle.Instance = USART2;

gUSART2Handle.Init.BaudRate=
UART_BAUD_RATE;

gUSART2Handle.Init.WordLength =
USART_WORDLENGTH_8B;

gUSART2Handle.Init.StopBits =
USART_STOPBITS_1;

gUSART2Handle.Init.Parity =
USART_PARITY_NONE;

gUSART2Handle.Init.HwFlowCtl =
UART_HWCONTROL_NONE;

gUSART2Handle.Init.Mode = USART_MODE_RX |
USART_MODE_TX;

HAL_UART_Init(&gUSART2Handle);

__HAL_UART_ENABLE_IT(&gUSART2Handle,
USART_IT_RXNE);

HAL_NVIC_EnableIRQ(USART2_IRQn);

HAL_NVIC_SetPriority(USART2_IRQn,1,1);

__HAL_UART_ENABLE(&gUSART2Handle);}}

static void uartInitSW(void)

{fifoReset(&gUartTxFIFO);fifoReset(&gUartR
xFIFO);

gUartTxFIFO.bufSize = UART_TX_FIFO_SIZE;

gUartTxFIFO.pData =
malloc(UART_TX_FIFO_SIZE);

gUartRxFIFO.bufSize = UART_RX_FIFO_SIZE;

gUartRxFIFO.pData =
malloc(UART_RX_FIFO_SIZE);

if ((gUartTxFIFO.pData != NULL) &&
(gUartRxFIFO.pData != NULL))

{gUartInitDone = 1;}}

uint16_t uartInit(void)

{uartInitHW();uartInitSW();

return gUartInitDone;}

void uartReset(void)

{INTERRUPTS_DISABLE();

fifoReset(&gUartTxFIFO);

fifoReset(&gUartRxFIFO);

__HAL_UART_DISABLE_IT(&gUSART2Handle,
USART_IT_TXE);

__HAL_USART_CLEAR_FLAG(&gUSART2Handle,
USART_IT_RXNE | USART_IT_TXE);

INTStatusEnumType uartStatus(void)

{ERRUPTS_ENABLE();}

Uart

UartStatusEnumType status = UART_OK;

INTERRUPTS_DISABLE();

if (gUartTxFIFO.status != FIFO_OK)

{status |= (FIFO_EMPTY&gUartTxFIFO.status)

       ? UART_TX_BUFFER_EMPTY : 0;

status |= (FIFO_FULL & gUartTxFIFO.status)

       ? UART_TX_BUFFER_FULL : 0;

status |=(FIFO_OVERRUN &
gUartTxFIFO.status)

       ? UART_TX_BUFFER_OVERRUN : 0;}

if (gUartRxFIFO.status != FIFO_OK)

{status |=(FIFO_EMPTY &
gUartRxFIFO.status)

       ? UART_RX_BUFFER_EMPTY : 0;

status |= (FIFO_FULL & gUartRxFIFO.status)

       ? UART_RX_BUFFER_FULL : 0;

status |= (FIFO_OVERRUN &
gUartRxFIFO.status)

       ? UART_RX_BUFFER_OVERRUN : 0;}

       INTERRUPTS_ENABLE();

       return status;}

uint16_t uartWrite(char* pStr)

{uint16_t bytesWritten =
fifoIn(&gUartTxFIFO, pStr, strlen(pStr));

if (bytesWritten == strlen(pStr))

{__HAL_UART_ENABLE_IT(&gUSART2Handle,
USART_IT_TXE);}

       else

{bytesWritten = 0;}

return bytesWritten;}
```

```
uint16_t uartRead(char* pStr, uint16_t
maxLen)

{uint16_t bytesRead = 0;

*pStr = '\0';

if (gUartRxFIFO.linesCount)

{while (maxLen)

{maxLen--;

if (fifoOut(&gUartRxFIFO, pStr, 1))

{if ('\r' == *pStr)

{*pStr = '\0';break;}}

Else {break;}

pStr++;bytesRead++;}}

return bytesRead;}

void USART2_IRQHandler(void)

{uint16_t data = 0;

INTERRUPTS_DISABLE();

if ((USART2->SR & USART_FLAG_RXNE) !=
RESET)

{data = gUSART2Handle.Instance->DR &
(uint16_t)0xFF;

fifoIn(&gUartRxFIFO, (uint8_t*)&data, 1);}

if ((USART2->SR & USART_IT_TXE) != RESET)

       //if
(__HAL_USART_GET_IT_SOURCE(&gUSART1Handle,
USART_IT_TXE) != RESET)

{if (fifoOut(&gUartTxFIFO,
(char*)&data,1))

{gUSART2Handle.Instance->DR = data;}

else

{__HAL_UART_DISABLE_IT(&gUSART2Handle,
USART_IT_TXE);}}

INTERRUPTS_ENABLE();}
```

TIMER

```
#include "timer.h"

#include "cortexm/ExceptionHandlers.h"

#include "adc.h"

// Forward declarations.

void timerTick(void);

static volatile uint32_t tickCount;

static volatile uint32_t tickCountDelay;

static volatile uint32_t tickCountTimeout;

static volatile uint32_t tickCountAdc =
ADC_SAMPLE_PERIOD;

void timerStart(void)

{SysTick_Config(SystemCoreClock /
TIMER_FREQUENCY_HZ);}

void timerSleep(uint32_t ticks)

{tickCountDelay = ticks;

while (tickCountDelay);}

void timerTick(void)

{tickCount++;

if (tickCountDelay)

{--tickCountDelay;}

if (tickCountAdc)

{tickCountAdc--;}

else

{adcNewSample();

tickCountAdc = ADC_SAMPLE_PERIOD;}}

uint32_t timerGetSysTime(void)

{return tickCount;}

void timerTimeoutSet(uint32_t ticks)

{tickCountTimeout = tickCount + ticks;}

uint8_t timerTimeout(void)

{return (tickCount > tickCountTimeout);}

void SysTick_Handler(void)

{timerTick();}
```