

Facultatea de electronică, telecomunicații și tehnologia informației
Universitatea Tehnică “Gheorghe Asachi” din Iași
Specializarea: Electronică aplicată

Lucrare de licență

Coordonator științific:

Asist. Dr. Ing. Amăriuței Roxana Daniela

Student:

Popa Nicolae Dănuț

Iași, 2018

Facultatea de electronică, telecomunicații și tehnologia informației
Universitatea Tehnică “Gheorghe Asachi” din Iași
Specializarea: Electronică aplicată

**Sistem autonom de transformare a energiei solare în
energie electrică**
Conversie, stocare și interfață utilizator

Coordonator științific:

Asist. Dr. Ing. Amăriuței Roxana Daniela

Student:

Popa Nicolae Dănuț

Iași, 2018

Cuprins

1. Introducere

- 1.1 Memoriu justificativ
- 1.2 Descriere generală a aplicației
- 1.3 Schema bloc
- 1.4 Descriere generală a blocurilor
- 1.5 Implementare practică
- 1.6 Aplicații utilizate

2. Module

- 2.1 Interfața cu utilizatorul
 - 2.1.1 Generalități
 - 2.1.2 Display
 - 2.1.3 Buton meniu
 - 2.1.4 Joystick
- 2.2 Control încărcare
 - 2.2.1 Generalități
 - 2.2.2 Panoul solar
 - 2.2.3 Convertorul Buck MPPT
 - 2.2.4 Bateria
 - 2.2.5 Dioda
 - 2.2.6 Releul
 - 2.2.7 Disjunctorul

3. Microcontrolerul

- 3.1 Generalități
- 3.2 ADC
- 3.3 Software

5. Analize și rezultate

- 5.1 Convertor
- 5.2 Interfață utilizator

6. Concluzie

7. Anexă

1. Introducere

1.1 Memoriu justificativ

Economia de energie este la ora actuală o prioritate mondială, protejarea planetei și a resurselor ei devenind obiective principale internaționale. Contextul energetic mondial, conduce către o preocupare intensă în domeniul energiilor neconvenționale.

Energiile regenerabile sunt considerate energii care provin din surse care se regenerează de la sine în scurt timp sau care sunt surse practic inepuizabile.

Printre sursele regenerabile de energie se enumera:

1. Energia eoliană;
2. Energia solară;
3. Energia apei;
4. Energia geotermică;
5. Energia de biomasă;

Energia solară ocupă un loc important pe plan mondial. Aceasta poate fi captată și transformată fie în energie electrică prin utilizarea tehnologiilor fotovoltaice, fie în energie termică, prin utilizarea diferitelor tipuri de panouri solare termice. În acest context, lucrarea are la bază producerea energiei electrice cu ajutorul panourilor fotovoltaice și transferul termic ce are loc la nivelul acestora.

Panourile fotovoltaice sunt dispozitive ce transformă energia solară în energie electrică. O problemă existentă în producerea energiei este datea de fenomenologia ce decurge din caracterul sursei solare și al variațiilor meteorologice imprevizibile. Pe de altă parte sistemele fotovoltaice nu utilizează decât o mică parte din radiația solară și anumite lungimi de undă, pentru a produce energie electrică. Restul energiei primite este transformată în căldură, ce conduce la creșterea temperaturii celulelor componente și la scăderea randamentului lor.

În consecință, creșterea productivității energetice a acestor instalații presupune atât eficientizarea funcționării lor în domeniul electric, cât și studiul fenomenelor termice care au loc.

1.2 Descriere generală a aplicației

Aplicația prezentată este un sistem autonom care convertește energia luminii solare în electricitate cu ajutorul unui panou solar.

Pentru a funcționa cu o eficiență maximă, sistemul determină prin intermediul senzorului de lumină poziția optimă a panoului solar astfel încât să capteze cât mai multă energie. Mișcarea de rotație și înclinare a panoului este realizată cu ajutorul unui motor pas cu pas și a unui motor de curent continuu.

Scopul acestei aplicații este de a încarcă eficient o baterie de 12 V dar și pentru a oferi protecție la supratensiune, supraîncărcare, supracurent și descărcare accidentală a bateriei.

1. Protecție la supratensiune: în momentul în care panoul produce o tensiune prea mare, aceasta este reglata la un nivel acceptabil pentru baterie prin intermediul unui convertor de tip coborâtor (convertor buck).



Figura 1.2.1

2. Protecție la supraîncărcare: dacă tensiunea de la bornele bateriei depășește pragul maxim admis, se întrerupe procesul de încărcare prin acționarea unui releu ce realizează legătura electrică între baterie și convertor.



Figura 1.2.2

3. Protecție la supracurent: se utilizează un disjuncțor montat la bornele bateriei cu pragul de activare setat la valoarea de 10 A;



Figura 1.2.3

4. Protecție împotriva descărcării accidentale a bateriei: se utilizează o diodă care împiedică trecerea curentului electric de la baterie către convertor;



Figura 1.2.4

1.3 Schema bloc

Schema bloc a aplicației:

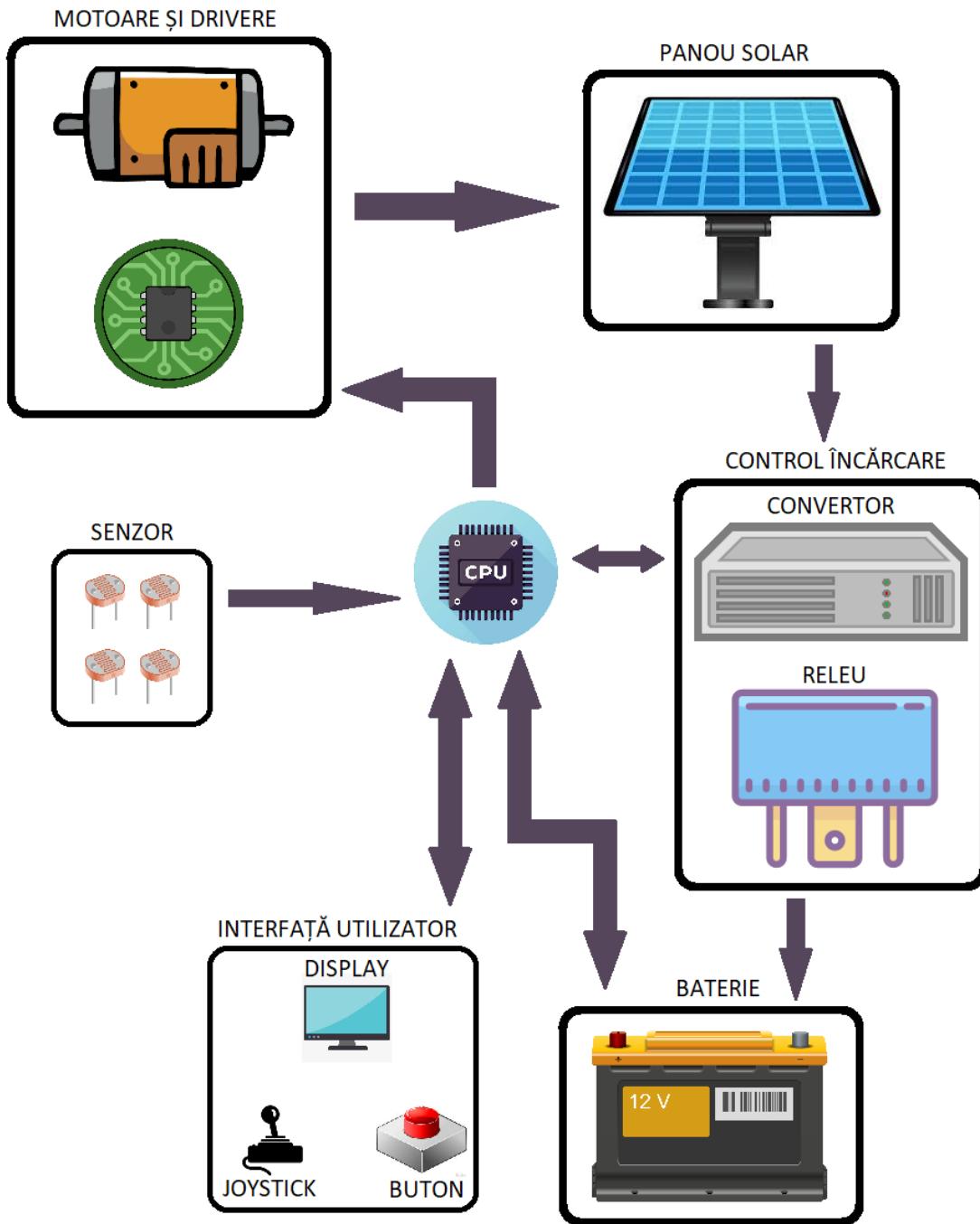


Figura 1.3.1

1.4 Descriere generală a blocurilor

MICROCONTROLERUL

Microcontrolerul are rolul de unitate principală de logică și control. Acesta preia datele de intrare necesare (în acest caz intensitatea luminoasă) și acționează corespunzător modului de funcționare calculat pe baza datelor: acționare motoare, pornire/oprire proces de încărcare a bateriei, interacțiune cu utilizatorul, etc.



Figura 1.4.1

PANOUL SOLAR

Panoul solar este blocul cheie al acestei aplicații. Rolul său este a prelua energia luminoasă de la soare și de a o transforma în energie electrică.



Figura 1.4.2

SENZOR

Blocul senzor, montat pe panoul solar, este format din patru senzori de citire a intensității luminoase (patru fotorezistențe) și are rolul de a oferi microcontrolerului datele de intrare necesare detectării poziției optime, astfel încât panoul solar să capteze cantitatea maximă de energie posibilă. Cele patru fotorezistențe sunt despărțite între ele printr-un scut opac de plastic ce formează patru zone de interes din care se pot citi intensitățile luminoase.

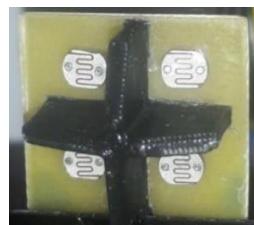


Figura 1.4.3

MOTOARE ȘI DRIVERE

Pentru a funcționa cu eficiență maximă, sistemul poate orienta panoul solar pe două axe astfel încât razele solare să ajungă perpendicular pe acesta. Blocul motoare și drivere este responsabil cu mișcările de rotație și de inclinație a panoului. Mișcarea de rotație este asigurată de un motor de curent continuu, în timp ce mișcarea de înclinare se realizează cu ajutorul unui motor de tip pas cu pas. Motoarele sunt acționate cu ajutorul driverelor specifice, care la rândul lor sunt controlate de microcontroler.

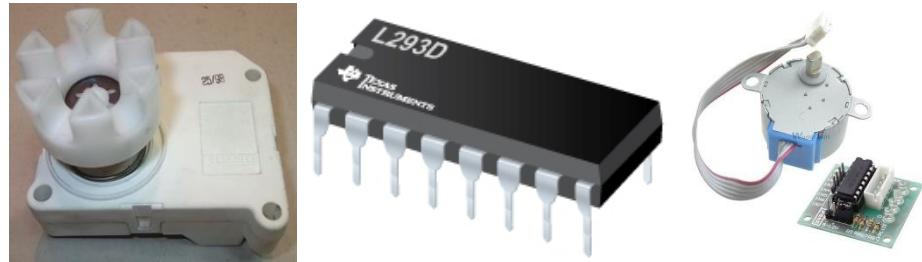


Figura 1.4.4

BATERIE

Bateria are rolul de a stoca energia generată de panoul solar pentru a putea fi utilizată ulterior. De asemenea, bateria reprezintă sursa principală de alimentare a sistemului.



Figura 1.4.5

CONTROL ÎNCĂRCARE

Elementele principale ce alcătuiesc blocul control încărcare sunt convertorul și releul. Cu ajutorul convertorului, tensiunea generată de panoul solar (aproximativ 18V) este adusă la un nivel potrivit încărcării bateriei (aproximativ 15V). Releul joacă rolul de întrerupător al alimentării bateriei de către panou.

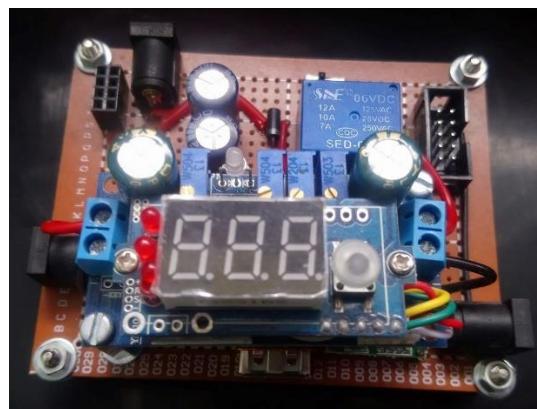


Figura 1.4.6

INTERFĂ UTILIZATOR

Blocul interfață utilizator permite comunicarea dintre sistem și utilizatorul uman.

Ecranul utilizat (display) permite utilizatorului să vadă în timp real modul de funcționare al sistemului, datele procesate și alte informații utile (tensiune baterie, nivel de încărcare, tensiune convertor, etc.). Prin intermediul butonului se poate schimba manual modul de funcționare al sistemului. Joystick-ul este utilizat în modul de funcționare control manual pentru a permite utilizatorului acționarea manuală a motoarelor.



Figura 1.4.7

1.5 Implementare practică



Figura 1.5.1

1.6 Aplicații utilizate

În procesul de realizare a proiectului, s-au folosit mai multe aplicații software. Acestea au avut numeroase roluri printre care:

- scriere și generare cod sursă
- versionare evoluție cod aplicație
- încărcare cod mașină pe microcontroler
- design hardware: schematic & layout
- design model 3D
- printare model 3D
- design workflow & scheme bloc

Atmel Studio 7.0

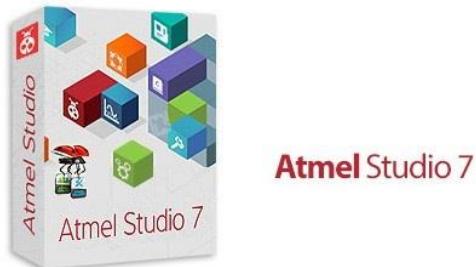


Figura 1.6.1

Atmel Studio 7 este o platformă de dezvoltare integrată (IDP) construită pentru dezvoltarea și depanarea tuturor aplicațiilor specifice microcontrolerelor de tip AVR și SAM.

Caracteristici principale:

- oferă suport pentru peste 500 de dispozitive AVR și SAM;
- oferă o vastă librărie de coduri sursă, inclusiv drivere, stive de comunicare, peste 1.600 de exemple de proiecte cu cod sursă, servicii grafice și funcționalitate prin atingere utilizând Advanced Software Framework (ASF);
- extensii IDE prin intermediul Atmel Gallery și magazin de aplicații online pentru instrumente de dezvoltare și software încorporat de la Microchip;
- monitorizare în timp real al consumului de energie și vizualizare de date în timp real cu posibilitatea de a genera grafice utilizând Atmel QTouch Composer;
- scriere și depanare cod C / C ++ și cod asamblare cu compilatorul integrat;
- funcții avansate de depanare incluzând întreruperea datelor, monitorizare întrerupere, urmărire a datelor interogate;
- editor integrat cu asistență vizuală;
- simulare completă a cipurilor pentru modele specifice de microcontrolere, întreruperi, periferice și stimuli externi

GitHub Desktop



Figura 1.6.2

GitHub Desktop este o aplicație de tip GUI (Graphical User Interface) ce permite utilizarea cu ușurință a funcționărilor de versionare Git. Git este un sistem de control al versiunii pentru urmărirea modificărilor în fișierele pe computer și coordonarea activității asupra acestor fișiere în rândul mai multor persoane. Acesta este utilizat în principal pentru gestionarea codului sursă în dezvoltarea de programe software, dar poate fi folosit pentru a urmări schimbările în orice set de fișiere. Ca sistem de control revizuit și distribuit, acesta vizează viteza, integritatea datelor și suport pentru fluxurile de lucru distribuite, neliniare.

Git a fost creat de Linus Torvalds în 2005 pentru dezvoltarea kernel-ului Linux, alți dezvoltatori de kernel-uri contribuind la dezvoltarea sa inițială.

Ca în majoritatea sistemelor de control distribuite și, spre deosebire de cele mai multe sisteme client-server, fiecare director Git de pe fiecare calculator este un depozit complet cu istorie completă și abilități de urmărire a versiunii complete, independent de accesul la rețea sau un server central.

Git este software gratuit și open-source, distribuit în termenii licenței GNU General Public Version 2.

Versionarea și codul sursă ale acestui proiect sunt valabile la următorul link:

https://github.com/scorpionipx/DD_IPX/tree/develop/licenta/cod/SunTracking32A/SunTracking32A/SunTracking32A

AVR Extreme Burner:



Figura 1.6.3

AVR Extreme Burner este un program software de tip GUI (Graphical User Interface) dedicat programatoarelor pentru microcontrolere AVR și care folosesc la bază protocolul USB ASP (Universal Serial Bus – AVR Serial Programmer).

Aceasta diminuează efortul depus de utilizatorul în procesul de scriere a memorilor EEPROM și flash pe microcontrolere.

Aplicația a fost utilizată pentru a încărca pe microcontroler codul sursă dezvoltat.

Pentru a flash-ui codul mașină pe microcontroler s-a utilizat un programator de tipul Atmel USBASP.



Figura 1.6.4

EAGLE



Figura 1.6.5

EAGLE este o aplicație software de automatizare a proiectării electronice, care permite creare schematicelor, layout-urilor pentru circuite imprimate (PCB), caracteristici de auto-router și asamblare automată. EAGLE (Easily Applicable Graphical Layout Editor) este un editor ușor de utilizat în format grafic și este dezvoltat de CadSoft Computer GmbH.

Caracteristici principale:

- conține un editor schematic, pentru proiectarea schemelor de circuit. Schematicul este stocat în fișiere cu extensie .SCH, iar componentele sunt definite în bibliotecile de dispozitive cu extensia .LBR. Componentele pot fi plasate pe mai multe coli și conectate împreună prin porturi;
- editorul de layout PCB stochează fișierele de tip board (plăcuță) cu extensia .BRD. Aceasta permite adnotarea înapoi la schema și rutarea automată pentru a conecta în mod automat componentele pe baza conexiunilor definite în schematic;
- EAGLE salvează fișierele de tip board generate sub forma Gerber și PostScript, iar fișierele de găurit sub format Excellon și Sieb & Meyer. Acestea sunt formatele de fișiere standard acceptate de companiile de fabricare a PCB-urilor, însă având în vedere faptul că cei mai mulți utilizatori EAGLE sunt reprezentanți de firmele mici și pasionați, mulți fabricanți de PCB și magazine de asamblare acceptă și fișiere de bord EAGLE (cu extensia .BRD).
- EAGLE oferă o interfață grafică de tip windows și un sistem de meniuri pentru editare, management de proiect și pentru personalizarea interfeței și a parametrilor de

proiectare. Sistemul poate fi controlat prin intermediul mouse-ului, scurtăturilor de tastatură sau prin introducerea unor comenzi specifice într-o linie de comandă încorporată. Comenziile repetitive multiple pot fi combinate în fișierele de tip script (cu extensia de fișiere .SCR). Este, de asemenea, posibil să se exploreze fișiere de proiectare utilizând un limbaj de programare orientat pe obiecte specific EAGLE (cu extensia .ULP).

VECTARY



Figura 1.6.6

Vectary este un instrument 3D online, ușor de folosit, unde se pot realiza modele 3D. Aplicația permite personalizarea şablonelor sau începerea propriului proiect prin asamblarea elementelor dintr-o imensă bibliotecă de materiale 2D și 3D de la Google Poly și alte arhive ce permit căutarea materialelor.

Cu ajutorul aplicației VECTARY s-a realizat designul 3D al scutului opac ce desparte cele patru fotorezistențe din alcătuirea senzorului de intensitate luminoasă.

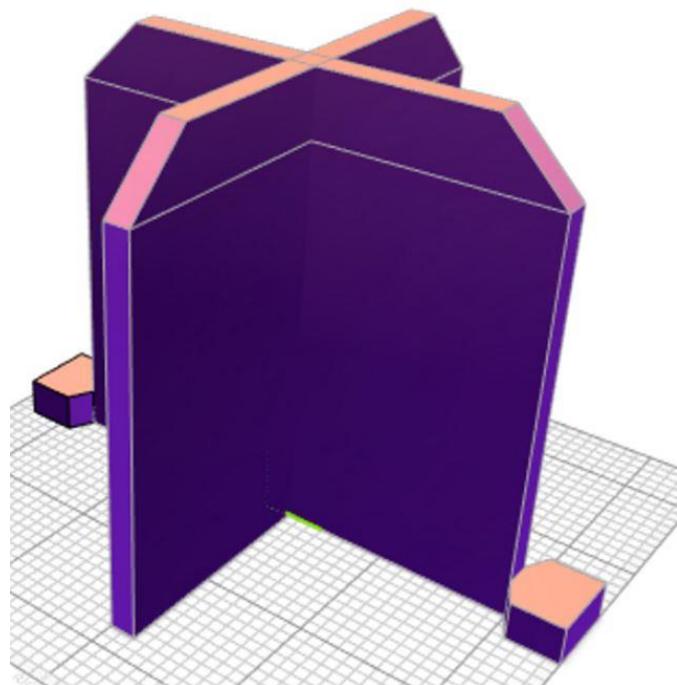


Figura 1.6.7

Modelul 3D poate fi găsit accesând link-ul următor:

<https://www.vectary.com/model3d/5dae01d1-ef5c-4821-adae-3c206c1e17d4>

OctoPrint



Figura 1.6.8

OctoPrint este o aplicație ce permite controlul imprimantelor 3D prin intermediul unui browser web. Această aplicație a fost folosită pentru a printa 3D modelul scutului creat.

Imprimanta 3D folosită este una modelul Anycubic 3D Printer I3 Mega.



Figura 1.6.9

Microsoft Visio



Figura 1.6.10

Visio este un program software ce ajută la crearea în mod simplu și intuitiv a schemelor logice, diagramelor de rețea, organigramelor, planurilor de nivel, proiectelor de inginerie și altele utilizând forme și şabloane moderne. A fost utilizat pentru crearea schemelor bloc ale proiectului.

2. Module

2.1 Interfață cu utilizatorul

2.1.1 Generalități

O interfață este o parte a unui sistem care servește comunicării. În tehnologia calculatoarelor, termenul de interfață se referă la un punct de interacțiune dintre două unități, dispozitive componente ale unui sistem, care este compatibil din punct de vedere hardware și software.

Interacțiunea și schimbul de informații printr-o interfață, între două unități de sistem, se realizează prin mărimi fizice precum tensiune, curent electric sau prin mărimi logice care se pot prezenta sub formă de semnale analogice (continue) sau semnale digitale (discontinue, discrete).

Interfață cu utilizatorul

Majoritatea sistemelor dețin o interfață prin intermediul căreia realizează comunicarea cu operatorul uman. În cazul de față, sistemul implementat dispune de o interfață cu utilizatorul atât hardware cât și grafică.

Interfață hardware

Exemplu: tastatura unui mic calculator de buzunar poate fi considerată o interfață hardware.

Interfața hardware a sistemului este formată din butoanele de meniu ce permit comutarea modului de funcționare și joystick-ul utilizat pentru comanda manuală a motoarelor.

Interfață grafică

Interfața grafică cu utilizatorul (GUI – Graphical User Interface) se situează din punct de vedere funcțional între utilizator și dispozitive electronice cum ar fi computere, dispozitive personale de tip hand-held (playere MP3, playere media portabile, dispozitive de jucăt), aparate electrocasnice și unele echipamente de birou.

2.1.2 Display

Elementul principal al interfeței grafice a sistemului construit este display-ul LCD (Liquid Crystal Display) HX1230, o copie chinezească, dar îmbunătățită a modelului NOKIA 5110.

Afișajul cu cristale lichide sau prescurtat LCD este un dispozitiv de afișare pentru litere, cifre, grafică și imagini, fiind constituit dintr-o matrice de celule lichide care devin opace sau își schimbă culoarea sub influența unui curent sau câmp electric.

Un afișaj LCD se prezintă sub forma unui ecran afișor (display) care este comandat electronic printr-un decodificator de caractere numerice și alfabetice.

Display-ul Nokia 5110

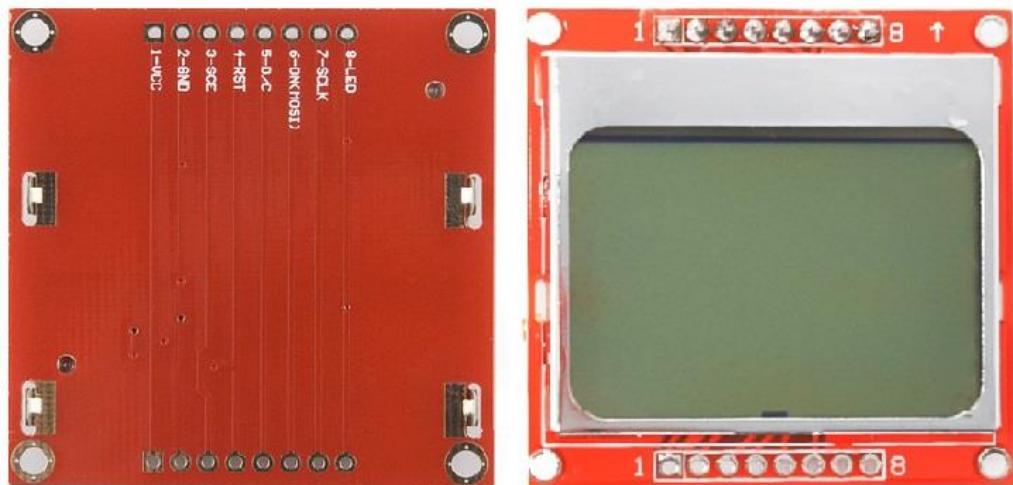


Figura 2.1.2.1

Display-ul HX1230



Figura 2.1.2.2

Descriere configurație pini:

1. Reset - acest pin resetează modulul în momentul în care i se aplică o tensiune de 0V, ducând sistemul într-o stare sigură, implicită și cunoscută;
2. Chip Enable (CE) – prin intermediul acestui pin se selectează display-ul ca dispozitiv comandat (de interes) pe magistrala de date și clock a interfeței SPI;
3. Data/Command (DC) - acest pin este utilizat pentru comutarea între modul de transmitere al datelor și modul transmitere al comenziilor; copia chinezeasca (HX1230) nu are implementată această funcționalitate. În schimb modulul său SPI este construit astfel încât să recepționeze date în pachete de 9 biți (față de pachetul standard de 8 biți). Astfel, primul bit transmis servește drept selecție al modului de interpretare a datelor receptate de display: 0 = comandă, 1 = date;
4. Serial Input (DIN) - acesta este pinul de intrare prin care sunt trimise datele seriale de către microcontroler;
5. Clock (CLK) – acestui pin i se furnizează un semnal sursă de ceas (clock) prin care se sincronizează dispozitivele SPI conectate pe magistrală;
6. Power (Vcc) – acest pin este folosit pentru alimentarea afișajului, tensiunea de alimentare poate varia între 2,7V și 3,3V;
7. Back Light (BL) - acest este folosit pentru acționarea luminii de fundal a afișajului ;
8. Ground (Gnd) – punctul de referință;

Caracteristici:

- Tensiunea de operare cuprinsă între 2.7V și 3.3V;
- Consumul de curent redus de aproximativ 6 mA;
- Conține 68 de rânduri și 96 de coloane (96 x 68) pixeli monocromi (modelul Nokia 5110 dispunea de numai 84 × 48 pixeli);
- Funcționează utilizând interfața SPI;
- Controlat cu driver-ul Philips PCD8544 pentru o interfață ușoară;
- Poate fi ușor interfațat cu Arduino;
- Suportă grafica imaginilor în format bitmap;
- Interfață serială de maximum 8 Mb/s;
- Consum redus de energie, potrivit pentru funcționarea cu baterii;
- Interval de temperatură: -25 până la +70 ° C.

Display-ul este controlat de microcontroler prin intermediul unui driver dedicat. În domeniul electronicii, un driver de afişaj este de obicei un circuit integrat de tip semiconductor (dar poate cuprinde, în mod alternativ, o maşină de stări(state machine)) care oferă o interfaţă (legătură) între un microprocesor, microcontroler, ASIC sau o interfaţă periferică generală și un anumit tip de dispozitiv de afişare, de ex LCD, LED, OLED, ePaper, CRT, Fluorescent vid sau Nixie.

Driverul de afişare va accepta în mod obișnuit comenzi și date utilizând o interfaţă standard serială sau paralelă, cum ar fi TTL, CMOS, RS232, SPI, I2C etc. și generează semnale de tensiune, curent, temporizare și demultiplexare adecvate pentru a controla dispozitivul de afişare să expună textul sau imaginea dorită.

Driverul de afişare poate fi în sine un microcontroler specific aplicației și poate include memoria RAM, memoria flash, EEPROM și / sau ROM. Memoria ROM-ul poate conține firmware-ul și diferite fonturi de afişare.

Un exemplu notabil al unui IC (integrated circuit) driver de afişare este controlerul LCD Hitachi HD44780.

În cazul de față, display-ul HX1230 utilizează driver-ul PCD8544

Driver display Philips PCD8544:

PCD8544 este un controler / driver CMOS cu putere redusă, conceput pentru a conduce un afişaj grafic de 68 de rânduri și 96 coloane.

Funcțiile necesare pentru afişare sunt furnizate într-un singur chip, inclusiv generarea pe chip a alimentării LCD-ului, astfel rezultând un minim de componente externe necesare și un consum redus de energie.

Pentru a comunica cu microcontrolerul, se folosește interfața serială SPI, cu lungimea cuvintelor de cod de 9 biți.

Schema bloc driver PCD8544:

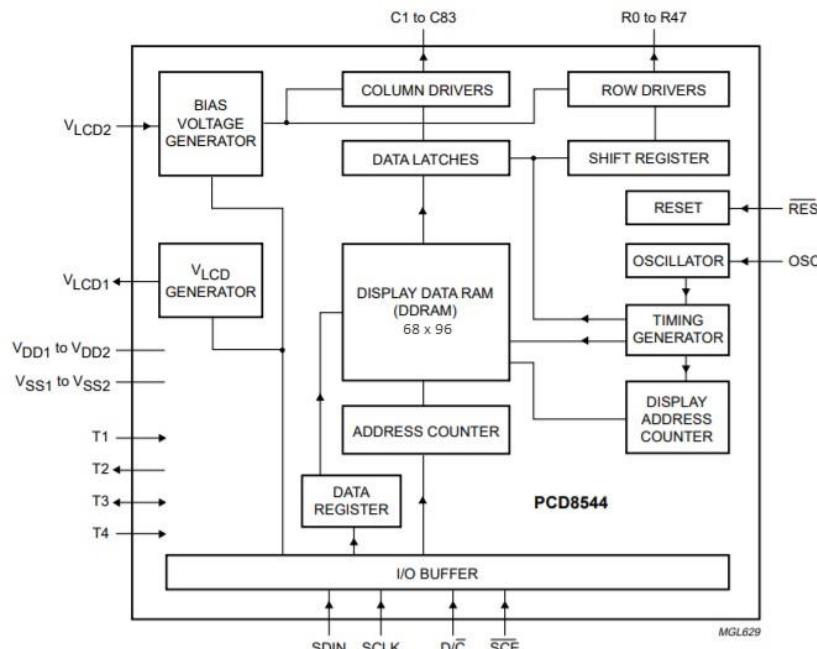


Figura 2.1.2.3

Interfața serială SPI

Interfața serială SPI (Serial Peripheral Interface) este o interfață sincronă standard de mare viteză, ce operează în mod full duplex.

Este folosită pentru transmiterea de date, unde circuitele sunt interconectate utilizând principiul master-slave. Modul master/slave semnifică faptul că circuitul digital master inițiază transmiterea de date. Este permisă conectarea mai multor dispozitive digitale slave cu selecție individuală.

SPI-ul are patru semnale logice specifice:

- **SCLK** - Ceas serial (ieșire din master).
- **MOSI/SIMO** - Master Output, Slave Input (ieșire master, intrare slave).
- **MISO/SOMI** - Master Input, Slave Output (intrare master, ieșire slave).
- **SS** - Slave Select (active low, ieșire din master).

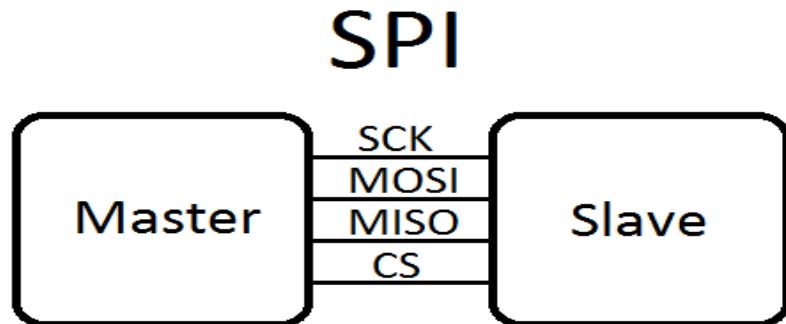


Figura 2.1.2.4

Mod de funcționare:

Ca mod de operare, interfața SPI utilizează un singur dispozitiv master și unul sau mai multe dispozitive slave.

Pentru a realiza transmisia, master-ul configurează mai întâi semnalul de ceas (clock) folosind o frecvență mai mică sau egală cu cea suportată de dispozitivele slave.

În timpul fiecărui ciclu de ceas SPI, apare o transmisie full duplex:

- master-ul trimite un bit pe linia MOSI; slave-ul îl citește de pe aceeași linie;
- slave-ul trimite un bit pe linia MISO; master-ul îl citește de pe aceeași linie.

Comunicația pe SPI implică de obicei existența a doi registri de shiftare (Shift Registers), unul aparținând dispozitivului master și unul aparținând dispozitivului slave, conectați în ciclu.

Principiul de transfer al datelor specific comunicației SPI

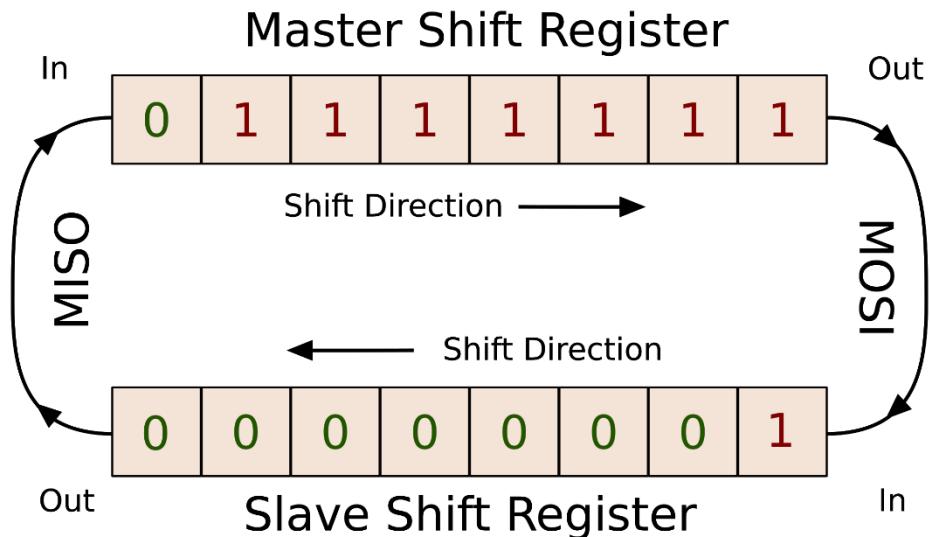


Figura 2.1.2.5

De obicei primul bit shiftat pe liniile de MISO/MOSI este bitul cel mai semnificativ, în timp ce un nou bit este adăugat pe poziția cea mai puțin semnificativă din registru. După ce întregul cuvânt a fost trimis, prin shiftare, masterul și slave-ul au interschimbat valorile din cei doi regiștri de shift. Dacă mai există date de transmis, procesul este reluat. Când nu mai există date de transmis, masterul întrerupe generarea semnalului de clock și, în general, pune valoarea de 1 logic pe linia de SS asociată dispozitivului slave. Chipurile slave care nu au fost selectate vor ignora semnalele de pe SCK și MOSI și nu vor genera nimic pe linia MISO.

Control modul display

Pentru a putea utiliza modulul display HX1230, acesta necesită o tensiune de alimentare de 3.3V. Microcontrolerul utilizat în cadrul proiectului funcționează la tensiunea nominală de 5V. Pentru a reduce nivelul de tensiune de 5V la cel util afișajului, 3V3, se utilizează regulatorul liniar de tip LDO (Low Dropout Regulators) KA78R33.

KA78R33



Figura 2.1.2.6

Schema bloc internă a integratului KA78R33

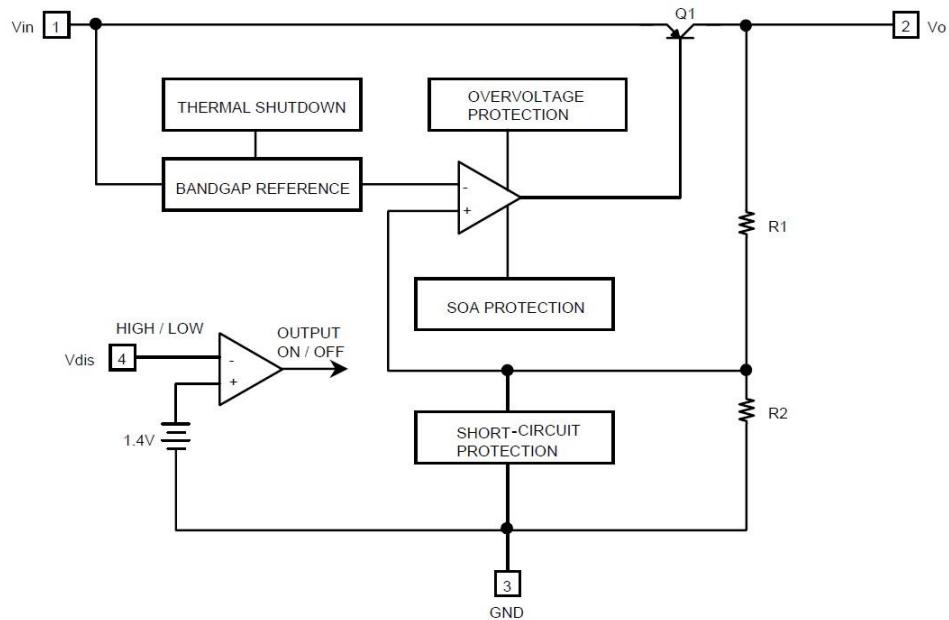


Figura 2.1.2.7

Din moment ce comunicația dintre microcontroler și display este unidirecțională (microcontrolerul doar trimite date, iar display-ul doar recepționează date), pentru a reduce nivelul de tensiune al semnalelor transmise de microcontroler (5V) către display (3V3) este suficientă utilizarea unor divizori rezistivi cu factorul de divizare 2/3.

Principiu divizor rezistiv

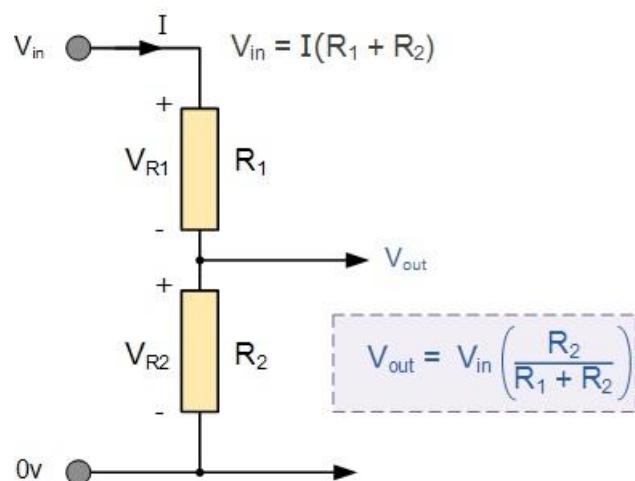


Figura 2.1.2.8

În dezvoltarea sistemului s-au utilizat rezistențe de valori $1K8\Omega$ (R1) și $3K6\Omega$ (R2).

Calcul tensiune divizor

$$V_{in} = 5V$$

$$R_1 = 1K8\Omega$$

$$R_2 = 3K6\Omega$$

$$V_o = V_{in} \cdot \frac{R_2}{R_1 + R_2} = 5V \cdot \frac{3600}{1800+3600} = 5V \cdot \frac{3600}{5400} = 5V \cdot 0.66 = 3.33V$$

Implementare hardware – schematic

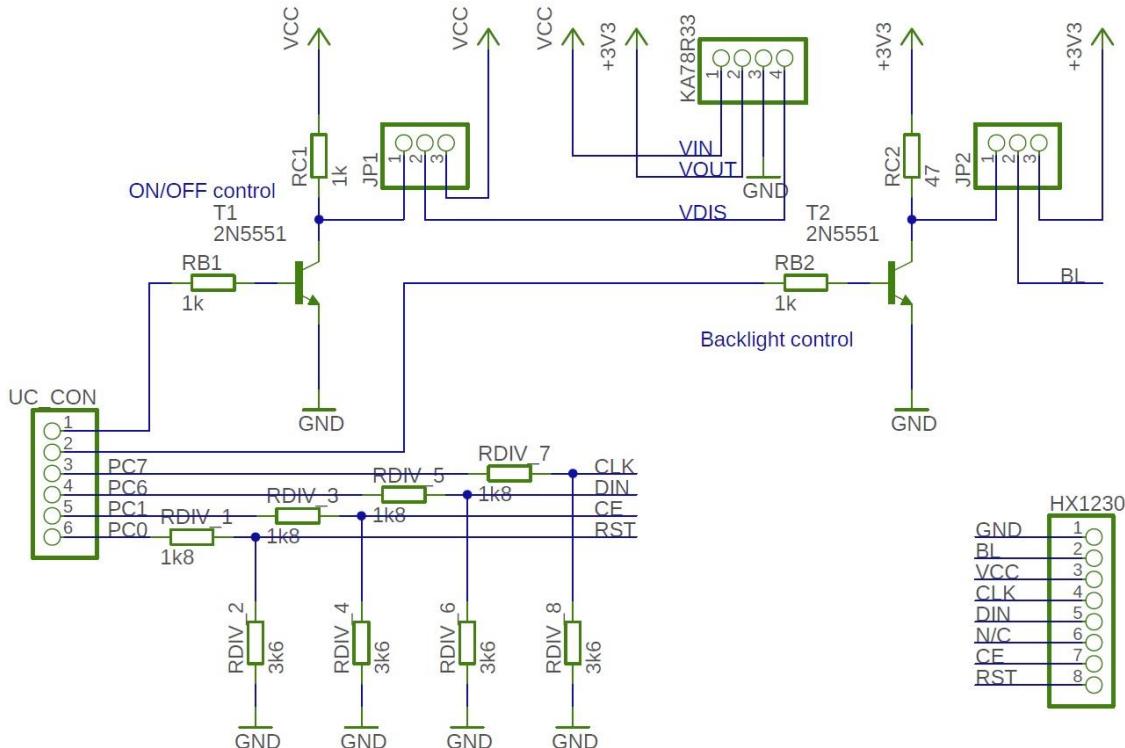


Figura 2.1.2.9

Adaptorul pentru driver-ul display-ului a fost conceput pentru a putea controla cu ajutorul microcontrolerului alimentarea și lumina de fundal a afișorului, dar și pentru a permite funcționarea continuă a acestuia, fără nicio intervenție externă.

Tranzistorul T1 este folosit pentru a porni și pentru a opri alimentarea generală a afișajului. Un semnal de 0 logic aplicat acestuia reprezintă pornirea alimentării modului display prin activarea pinului Vdis (pin cu funcționalitate de enable) a regulatorului de tensiune KA78R33. Un semnal de 1 logic în baza tranzistorului T1 va furniza un semnal de 0 logic pinului Vdis, astfel întrerupând alimentarea display-ului.

Tranzistorul T2 permite pornirea și oprirea luminii de fundal a display-ului. Un semnal de 0 logic aplicat acestuia reprezintă aprinderea luminii de fundal, în timp ce un semnal de 1 logic ar cauza stingerea luminii de fundal. Pentru a regla intensitatea luminii de fundal a afișajului, baza tranzistorului T2 poate fi acționate de un semnal PWM (Pulse-width modulation = semnal modulat în durată). Astfel, intensitatea luminii de fundal va fi direct proporțională cu factorul de umplere al semnalului de comandă.

Conecțorii JP1 și JP2 sunt utilizati pentru a selecta modul de comandă al alimentării display-ului și controlul luminii de fundal al acestuia. În implementarea practică se utilizează modul always-on (mereu pornit). Acest mod este selectat prin intermediul unor jumpere care conectează electric pinul Vdis al regulatorului KA78R33 la tensiunea de alimentare generală (Vcc, 5V), respectiv pinul BL (Back Light) al display-ului la tensiunea de alimentare de 3V3.

Adaptor microcontroler – display

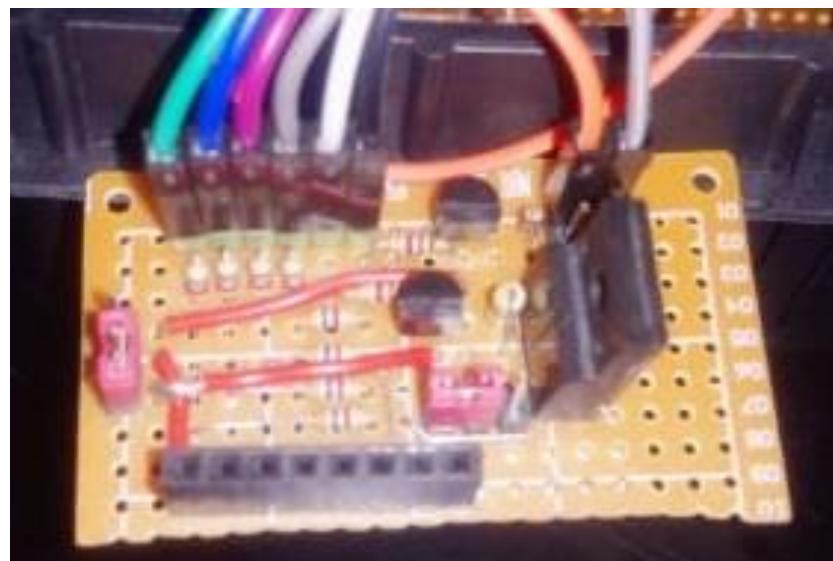


Figura 2.1.2.10

Utilizare display:

Cu ajutorul display-ului HX1230 se arată utilizatorului modul curent de funcționare al sistemului, dar și date interne în timp real. Tranzitia de la un mediu de funcționare la altul se face prin intermediul butonului atașat.

Modul inițializare

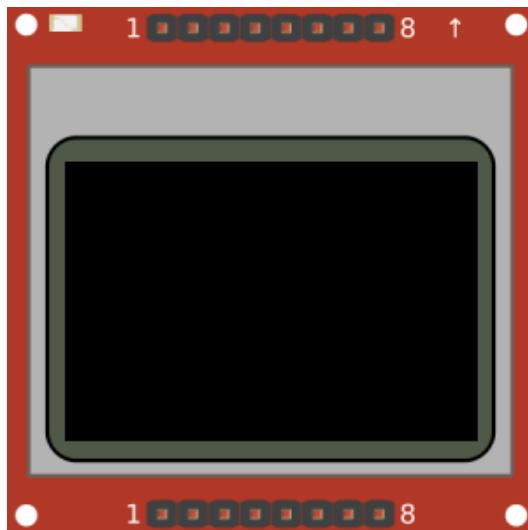


Figura 2.1.2.11

Modul de funcționare **inițializare** pregătește sistemul activând drivelele interne și perifericele necesare funcționării și durează aproximativ 850 milisecunde de la pornirea alimentării. În tot acest timp, display-ul menține activați toți pixelii display-ului.

Modul tracking / debugging

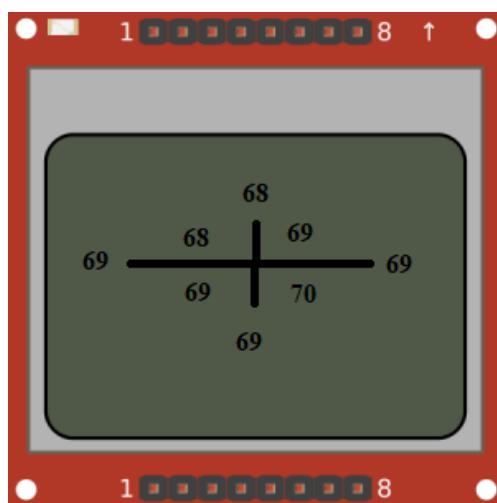


Figura 2.1.2.12

Modul de funcționare **tracking / debugging** activează funcționalitatea de găsire automată a poziției optime a panoului astfel încât acesta să capteze cantitatea maximă de energie posibilă. În plus, pe display se afișează datele preluate de microcontroler prin intermediul senzorului și informațiile calculate ulterior.

Desenul grafic de pe display ilustrează senzorul (format din patru fotorezistențe și un scut opac din plastic) atașat panoului solar și intensitatea luminoasă preluată de fiecare senzor în parte, sub formă procentuală (unde 0% înseamnă întuneric total, iar 100% înseamnă intensitate luminoasă maximă). Deasupra, dedesubt și de o parte și de alta a reprezentării grafice a senzorului se afișează mediile intensităților luminoase calculate pentru fiecare zonă de interes (sus, jos, stânga, dreapta).

Pe baza informațiilor calculate, microcontrolerul va roti și / sau înclina panoul prin intermediul motoarelor controlate de drivere.

Modul idle

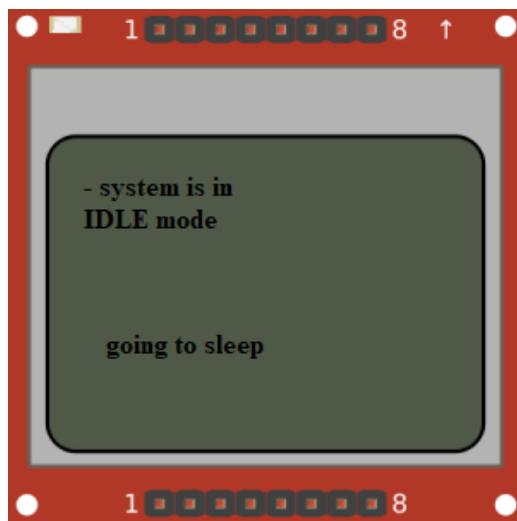


Figura 2.1.2.13

Modul idle dezactivează toate funcționalitățile și pregătește sistemul să intre în modul sleep.

Modul sleep

Modul sleep are rolul de a reduce la minim consumul sistemului. Toate funcționalitățile acestuia sunt dezactivate cu excepția celei de citire a butonului. Se așteaptă intervenția utilizatorului pentru a schimba modul de funcționare.

În acest timp, display-ul este dezactivat.

Modul monitorizare

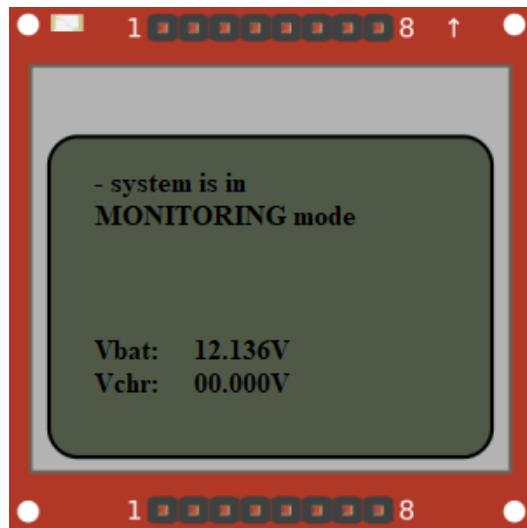


Figura 2.1.2.14

Modul monitorizare permite utilizatorului să urmărească în timp real tensiunea de la bornele bateriei și tensiunea de ieșire a convertorului buck. Datele sunt preluate prin intermediul modulului ADC al microcontrolerului, interpretate și apoi afișate pe display.

Modul manual

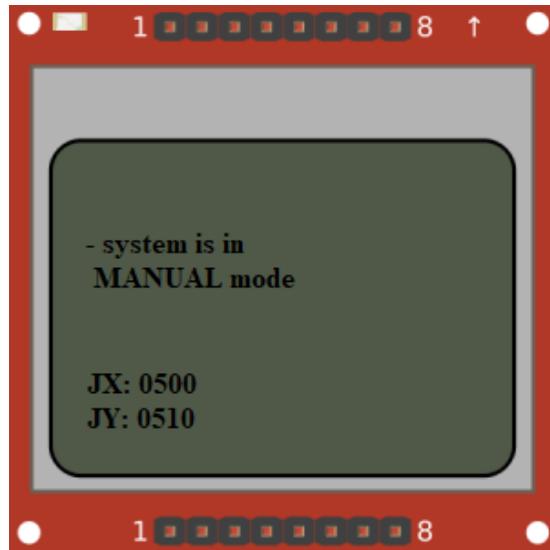


Figura 2.1.2.15

Modul manual permite utilizatorului control manual al motoarelor ce rotesc și inclină panoul solar. Pe display sunt afișate în timp real datele transmise microcontrolerului de către joystick.

Implementare software – cod sursă

Codul sursă utilizat pentru controlul display-ului este dezvoltat în fișierele hx1230.h și hx1230.c. În plus, în fișierul hx_8x6_characters.h sunt definite reprezentările simbolurilor tabelului ASCII (American Standard Code for Information Interchange) sub formă de vectori bidimensionali.

Pentru controlul display-ului HX1230 se utilizează numai patru pini ai microcontrolerului (PC0, PC1, PC6 și PC7) configurați astfel încât să îndeplinească funcțiile unui modul SPI.

Configurare pini microcontroler

```
#define HX1230_PORT PORTC //port used to control hx1230
#define HX1230_DDR DDRC //data direction register used for hx1230

#define HX_RST PC0 //external reset input
#define HX_CE PC1 //chip enable
#define HX_DIN PC6 //serial data input
#define HX_CLK PC7 //serial clock input

#define SET_HX_RST (HX1230_PORT |= (1 << HX_RST))
#define SET_HX_CE (HX1230_PORT |= (1 << HX_CE))
#define SET_HX_DIN (HX1230_PORT |= (1 << HX_DIN))
#define SET_HX_CLK (HX1230_PORT |= (1 << HX_CLK))

#define CLEAR_HX_RST (HX1230_PORT &= ~(1 << HX_RST))
#define CLEAR_HX_CE (HX1230_PORT &= ~(1 << HX_CE))
#define CLEAR_HX_DIN (HX1230_PORT &= ~(1 << HX_DIN))
#define CLEAR_HX_CLK (HX1230_PORT &= ~(1 << HX_CLK))
```

Figura 2.1.2.16

S-au dezvoltat un număr restrâns de funcții cu ajutorul cărora se controlează modulul display.

Funcții de control al display-ului

```
void init_hx1230_control(void);
void hx_send_data(unsigned char _data);
void hx_send_command(unsigned char _command);
void hx_set_coordinates(unsigned char _x, unsigned char _y);
void hx_clear_screen(void);
void hx_fill_screen(void);
void hx_write_char(const unsigned char _character);
void hx_write_string(const char *_characters_array);
```

Figura 2.1.2.17

Descriere funcții

Funcția **void init_hx1230_control(void)** pregătește configurarea hardware a microcontrolerului necesară pentru a controla display-ul.

```
void init_hx1230_control(void)
{
    // set required pins as output
    HX1230_DDR |= ((1 << HX_RST) | (1 << HX_CE) | (1 << HX_DIN) | (1 << HX_CLK));

    // set idle state
    CLEAR_HX_CLK;
    #ifdef HX_DELAY_ENABLED
    _delay_us(HX_DELAY_US);
    #endif
    CLEAR_HX_RST;
    #ifdef HX_DELAY_ENABLED
    _delay_us(HX_DELAY_US);
    #endif
    _delay_ms(50);
    SET_HX_RST;
    #ifdef HX_DELAY_ENABLED
    _delay_us(HX_DELAY_US);
    #endif
    CLEAR_HX_CE;
    #ifdef HX_DELAY_ENABLED
    _delay_us(HX_DELAY_US);
    #endif
    _delay_ms(1);
    SET_HX_CE;
    #ifdef HX_DELAY_ENABLED
    _delay_us(HX_DELAY_US);
    #endif
    _delay_ms(1);

    // commands needed to initialize hx1230 display
    // found within a chinese data sheet

    hx_send_command(0x2f);
    hx_send_command(0x90);
    hx_send_command(0xa6);
    hx_send_command(0xa4);
    hx_send_command(0xaf);

    hx_send_command(0x40);
    hx_send_command(0xb0);
    hx_send_command(0x10);
    hx_send_command(0x00);
}
```

Figura 2.1.2.18

Seria de comenzi necesare inițializării modulului display HX1230 a fost preluată din interiorul datasheet-ului dispozitivului.

Link către datasheet:

<https://cdn.instructables.com/ORIG/F4Y/Z9ZU/J76GKXJX/F4YZ9ZUJ76GKXJX.pdf>

Functia **void hx_send_data(unsigned char _data)** transmite către display un cuvânt de 8 biți (specificat prin intermediul parametrului **_data**, de tip **unsigned char**) sub formă de date. În prealabil se transmite un bit de nivelul 1 logic, pentru a specifica transferul unui cuvânt de tip date.

```

void hx_send_data(unsigned char _data)
{
    // activate hx1230
    CLEAR_HX_CE;
    #ifdef HX_DELAY_ENABLED
    _delay_us(HX_DELAY_US);
    #endif

    // configure communication for data transfer
    SET_HX_DIN;
    #ifdef HX_DELAY_ENABLED
    _delay_us(HX_DELAY_US);
    #endif

    // toggle clock
    SET_HX_CLK;
    #ifdef HX_DELAY_ENABLED
    _delay_us(HX_DELAY_US);
    #endif
    CLEAR_HX_CLK;
    #ifdef HX_DELAY_ENABLED
    _delay_us(HX_DELAY_US);
    #endif

    // send the actual data, MSB fiHX_RST
    for(int bit_position = 7; bit_position >= 0; bit_position--)
    {
        // calculate bit to be send
        if(((_data >> bit_position) & 1) == 1)
        {
            SET_HX_DIN;
            #ifdef HX_DELAY_ENABLED
            _delay_us(HX_DELAY_US);
            #endif
        }
        else
        {
            CLEAR_HX_DIN;
            #ifdef HX_DELAY_ENABLED
            _delay_us(HX_DELAY_US);
            #endif
        }

        // toggle clock
        SET_HX_CLK;
        #ifdef HX_DELAY_ENABLED
        _delay_us(HX_DELAY_US);
        #endif
        CLEAR_HX_CLK;
        #ifdef HX_DELAY_ENABLED
        _delay_us(HX_DELAY_US);
        #endif
    }

    // deactivate hx1230
    SET_HX_CE;
}

```

Figura 2.1.2.19

Functia **void hx_send_command(unsigned char _command)** transmite către display un cuvânt de 8 biți (specificat prin intermediul parametrului `_command`, de tip `unsigned char`) sub formă de comandă. În prealabil se transmite un bit de nivelul 0 logic, pentru a specifica transferul unui cuvânt de tip comandă.

```

void hx_send_command(unsigned char _command)
{
    // activate hx1230
    CLEAR_HX_CE;
    #ifdef HX_DELAY_ENABLED
    _delay_us(HX_DELAY_US);
    #endif

    // configure communication for command transfer
    CLEAR_HX_DIN;

    // toggle clock
    SET_HX_CLK;
    #ifdef HX_DELAY_ENABLED
    _delay_us(HX_DELAY_US);
    #endif
    CLEAR_HX_CLK;
    #ifdef HX_DELAY_ENABLED
    _delay_us(HX_DELAY_US);
    #endif

    // send the actual command, MSB fiHX_RST
    for(int bit_position = 7; bit_position >= 0; bit_position--)
    {
        // calculate bit to be send
        if(((_command >> bit_position) & 1) == 1)
        {
            SET_HX_DIN;
            #ifdef HX_DELAY_ENABLED
            _delay_us(HX_DELAY_US);
            #endif
        }
        else
        {
            CLEAR_HX_DIN;
            #ifdef HX_DELAY_ENABLED
            _delay_us(HX_DELAY_US);
            #endif
        }

        // toggle clock
        SET_HX_CLK;
        #ifdef HX_DELAY_ENABLED
        _delay_us(HX_DELAY_US);
        #endif
        CLEAR_HX_CLK;
        #ifdef HX_DELAY_ENABLED
        _delay_us(HX_DELAY_US);
        #endif
    }

    // deactivate hx1230
    SET_HX_CE;
}

```

Figura 2.1.2.20

Funcția **void hx_set_coordinates(unsigned char _x, unsigned char _y)** transmite către display comanda de mută cursorul la coordonatele indicate. Coordonata carteziană X este specificată prin intermediul parametrului *_x* de tip unsigned char, iar coordonată carteziană Y este specificată prin intermediul parametrului *_y* de tip unsigned char.

Pozitia cursorului reprezinta zona fizica unde va fi afisata urmatoarea data transmisă de microcontroler către display. Coordonata (0, 0) reprezinta pixelul aflat în colțul din stanga sus al ecranului.

```
void hx_set_coordinates(unsigned char _x, unsigned char _y)
{
    // 0, 0 is the upper left corner

    hx_send_command(0xB0 + _y);
    hx_send_command(0x10 | ((_x & 0x7F) >> 4));
    hx_send_command(0x0F & _x);
}
```

Figura 2.1.2.21

Funcția **void hx_clear_screen(void)** transmite comenzile și datele necesare stergerii complete a ecranului (se sterg toți pixelii).

```
void hx_clear_screen(void)
{
    unsigned char col, row;

    hx_set_coordinates(0, 0);

    for(row = 0; row <= HX_MAX_ROW_ROOT; row++)
    {
        for(col = 0; col <= HX_MAX_COL; col++)
        {
            hx_send_data(0x00);
        }
    }
}
```

Figura 2.1.2.22

Funcția **void hx_fill_screen(void)** transmite comenzi și datele necesare umplerii complete a ecranului (se aprind toți pixelii).

```
void hx_fill_screen(void)
{
    unsigned char col, row;

    hx_set_coordinates(0, 0);

    for(row = 0; row < 9; row++)
    {
        for(col = 0; col < 96; col++)
        {
            hx_send_data(0xFF);
        }
    }
}
```

Figura 2.1.2.23

Funcția **void hx_write_char(const unsigned char _character)** transmite datele și comenzi necesare afișării pe ecran a unui caracter ASCII. Caracterul este specificat prin intermediul parametrului **_character** de tip const unsigned char.

Valoarea parametrului este codul ASCII corespunzător caracterului dorit. În realitate acesta reprezintă indexul vectorului de date necesare a fi transmise pentru a reprezenta caracterul pe ecran. Vectorii de date sunt definiți în fișierul **hx_8x6_characters.h**.

```
void hx_write_char(const unsigned char _character)
{
    for(int row_index = 0; row_index < 6; row_index++)
    {
        hx_send_data(HX_character[_character][row_index]);
    }
}
```

Figura 2.1.2.24

Vectorul bidimensional ce conține datele necesare afișării unui caracter ASCII:

```
static const unsigned char HX_character[][6] = {
    {0x00,0x00,0x00,0x00,0x00,0x00}, // 0x 0 0
    {0x00,0x64,0x18,0x04,0x64,0x18}, // 0x 1 1
    {0x00,0x3c,0x40,0x40,0x20,0x7c}, // 0x 2 2
    {0x00,0x0c,0x30,0x40,0x30,0x0c}, // 0x 3 3
    {0x00,0x3c,0x40,0x30,0x40,0x3c}, // 0x 4 4
    {0x00,0x00,0x3e,0x1c,0x08,0x00}, // 0x 5 5
    {0x00,0x04,0x1e,0x1f,0x1e,0x04}, // 0x 6 6
    {0x00,0x10,0x3c,0x7c,0x3c,0x10}, // 0x 7 7
    {0x00,0x20,0x40,0x3e,0x01,0x02}, // 0x 8 8
    {0x00,0x22,0x14,0x08,0x14,0x22}, // 0x 9 9
    {0x00,0x00,0x38,0x28,0x38,0x00}, // 0x a 10
    {0x00,0x00,0x10,0x38,0x10,0x00}, // 0x b 11
    [...]
    {0x00,0x24,0x2a,0x7f,0x2a,0x12}, // à 0xf2 242
    {0x00,0x04,0x1e,0x1f,0x1e,0x04}, // ó 0xf3 243
    {0x00,0x00,0x00,0x00,0x00,0x00}, // ô 0xf4 244
    {0x00,0x00,0x00,0x00,0x00,0x00}, // õ 0xf5 245
    {0x00,0x00,0x00,0x00,0x00,0x00}, // ö 0xf6 246
    {0x00,0x00,0x00,0x00,0x00,0x00}, // ÷ 0xf7 247
    {0x00,0x00,0x00,0x00,0x00,0x00}, // ø 0xf8 248
    {0x00,0x00,0x00,0x00,0x00,0x00}, // ù 0xf9 249
    {0x00,0x00,0x00,0x00,0x00,0x00}, // ú 0xfa 250
    {0x00,0x00,0x00,0x00,0x00,0x00}, // û 0xfb 251
    {0x00,0x00,0x00,0x00,0x00,0x00}, // ü 0xfc 252
    {0x00,0x00,0x00,0x00,0x00,0x00}, // ý 0xfd 253
    {0x00,0x00,0x00,0x00,0x00,0x00}, // þ 0xfe 254
    {0x00,0x00,0x00,0x00,0x00,0x00} // ÿ 0xff 255
};
```

Figura 2.1.2.25

Fiecare caracter este reprezentat grafic pe display utilizând 48 de pixeli (8 rânduri și 6 coloane). Vectorul bidimensional **static const unsigned char HX_character[][6]** conține valorile reprezentării grafice a tuturor caracterelor din tabelul ASCII de bază (256 de caractere).

În partea dreaptă a fiecărui set de valori este scris sub formă de comentariu caracterul descris și codul său ASCII sub forma hexazecimală și decimală.

Funcția **void hx_write_string(const unsigned char *_characters_array)** transmite datele și comenzi necesare afișării pe ecran a unui sir de caracter ASCII (string). Sirul de caractere este specificat prin intermediul parametrului ***_characters_array** de tip const unsigned char (pointer).

```
void hx_write_string(const char *_characters_array)
{
    int string_length = strlen(_characters_array);
    for(int char_index = 0; char_index < string_length; char_index++)
    {
        hx_write_char((const unsigned char) (_characters_array[char_index]));
    }
}
```

Figura 2.1.2.26

2.1.3 Buton meniu

În aplicația prezentata se folosește un buton tactil pentru a putea schimba starea de funcționare a sistemului și implicit setul de informații oferite utilizatorului prin intermediul ecranului.

Butonul este un tip de comutator utilizat pentru controlul unor aparate sau numai a unor funcții ale acestora. Butoanele sunt de diferite forme și dimensiuni și se găsesc în tot felul de dispozitive, deși în principal în echipamentele electrice și electronice.

Acestea sunt în general activate atunci când sunt apăsate și permit trecerea unui fluxul de curent. Când nu mai sunt apăsate, revin în poziția inițială.

Un buton poate fi, prin construcția sa mecanică, de tipul normal deschis - ND sau NO (Normally Open în limba engleză), sau normal închis – NI sau NC (Normally Closed în engleză).

Butonul unui dispozitiv electronic, de obicei, funcționează ca un intrerupător electric. În interiorul său se află două contacte separate din punct de vedere electric ce se conectează în momentul acționării butonului.

În proiectul implementat se utilizează un buton de tipul **push switch** de pe un shield dedicat Arduino.



Figura 2.1.3.1

Funcționare:

Mecanismul acestuia constă din butonul propriu-zis acționat prin pulsare, o lamelă conductoare care stabilește contactele celor două terminale apăsând butonul propriu-zis, și un arc care face ca lamela să își recâștige poziția inițială după ce butonul nu mai este acționat.

Shield-ul nRF24L01 Nk 5110:



Figura 2.1.3.2

Din cadrul shield-ului anterior prezentat se utilizează butonul albastru din partea dreaptă. Acesta este conectat la tensiunea de alimentare (VCC, 5V) prin intermediul unei rezistențe de pull-up. În momentul în care butonul este acționat (apăsat), terminalul ce realizează conexiunea electrică spre microcontroler se conectează la ground. Astfel, starea implicită a butonului este high.

Implementare hardware – schematic echivalent

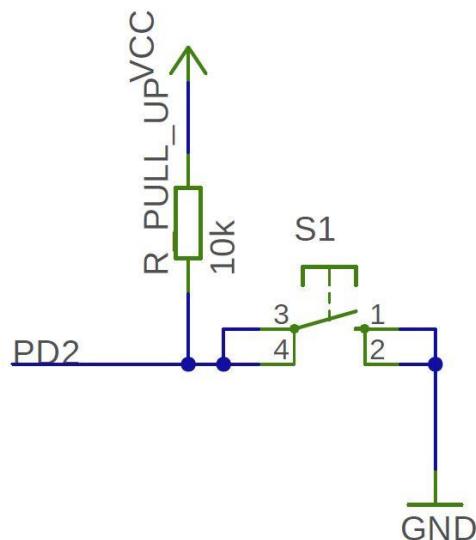


Figura 2.1.3.3

Implementare software – cod sursă

Codul sursă utilizat pentru interpretarea stării butonului este dezvoltat în fișierele `user_interface.h` și `user_interface.c`.

Pentru a detecta apăsarea butonului, s-a implementat o rutină de intrerupere ce se declanșează în momentul detecției unui front descrescător pe pinul PD2 (unde este conectat butonul) al microcontrolerului.

Rutina de intrerupere demarează procesele necesare schimbării modului de funcționare al sistemului. Pentru a elimina situațiile de acționare nedorită sau pentru a elimina riscul de a confunda o apăsare mai îndelungată cu apăsări multiple ale butonului, s-a implementat o metodă software de debouncing: se dezactivează temporar intreruperea specifică butonului, se face schimbarea modului de funcționare, se reactivează intreruperea și se așteaptă un interval de 250 milisecunde.

Configurare microcontroler

```
#define BUTTON_1_DRR DDRD  
#define BUTTON_1_PORT PORTD  
#define BUTTON_1_PIN 2 // PD2
```

Figura 2.1.3.3

Funcția void **init_next_state_button(void)** pregătește configurația hardware a microcontrolerului necesară interpretării stării butonului.

```
void init_next_state_button(void)  
{  
    BUTTON_1_DRR &= ~(1 << BUTTON_1_PIN); // PD2 is input  
  
    BUTTON_1_PORT |= (1 << BUTTON_1_PIN); // turn on the pull-up resistor  
    // PD2 is now an input with pull-up enabled  
  
    MCUCR &= ~(1 << ISC00 | 1 << ISC01); // low level of INT0 generates an interrupt request: when BUTTON_1 is pressed  
    GICR |= (1 << INT0); // turns on INT0  
}
```

Figura 2.1.3.4

Rutina de întrerupere este definită în vectorul de întreruperi **ISR (INT0_vect)**, specific pinului PD2, unde este conectat butonul.

```
ISR (INT0_vect)  
{  
    cli(); // temporarily disable interrupts  
    switch(STATE)  
    {  
        case STATE_IDLE:  
        {  
            STATE = STATE_MANUAL;  
            break;  
        }  
        case STATE_MANUAL:  
        {  
            STATE = STATE_TRACKING;  
            break;  
        }  
        case STATE_TRACKING:  
        {  
            STATE = STATE_MONITORING;  
            break;  
        }  
        case STATE_MONITORING:  
        {  
            STATE = STATE_IDLE;  
            break;  
        }  
        default:  
        {  
            STATE = STATE_IDLE;  
            break;  
        }  
    }  
}
```

Figura 2.1.3.5

Variabila **STATE** indică modul de funcționare curent al sistemului (state machine). Schimbarea modului de funcționare se face ciclic, succesiv, definit de ordinea: STATE_IDLE, STATE_MANUAL, STATE_TRACKING, STATE_MONITORING. Ca măsură de siguranță, în cazul în care printr-o eroare se cere trecerea la o stare care nu a fost definită în prealabil, sistemul va comuta pe starea IDLE.

Definirea stărilor

```
#define STATE_INIT 0
#define STATE_IDLE 1
#define STATE_MANUAL 2
#define STATE_TRACKING 3
#define STATE_MONITORING 4
```

Figura 2.1.3.6

2.1.4 Joystick

Un joystick este un periferic al computerului personal sau un dispozitiv de comandă ce constă într-o manetă care pivotează și transmite apoi unghiul său în două sau trei dimensiuni unui computer. Aceste dispozitive sunt folosite de obicei la controlul jocurilor video.

Cu același design de bază, dar cu o importanță mult mai mare, este și instrumentul principal de control al aparatelor de zbor (elicopter, avion). Se bazează pe același principiu al joystick-ului pentru PC, transmitându-și mișcarea 3D unui computer central ce îi transformă mișările în diferite acțiuni (virare, urcare, coborâre).

Ca principiu de funcționare, dispozitivele joystick au la bază două potențiometre circulare, ale căror rezistențe sunt modificate prin intermediul manetei atașate.

Schema electrică de principiu a unui joystick

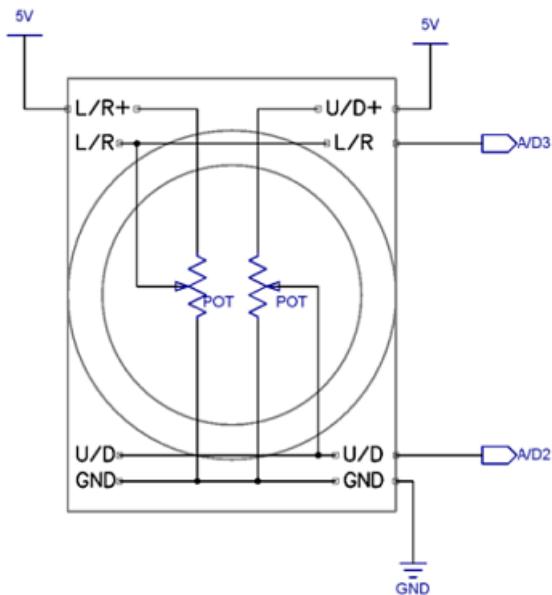


Figura 2.1.4.1

Fiecare potențiometru formează (cu ajutorul unor rezistențe fixe conectate intern) câte un divizor rezistiv. Tensiunile divizate rezultate sunt direct proporționale cu poziția de acționare a manetei. Astfel, cu ajutorul primului potențiometru montat se poate determina poziția manetei pe axa OX, în timp ce cu ajutorul celui de al doilea potențiometru se poate detecta poziția manetei față de axa OY.

Construcția fizică a unui joystick

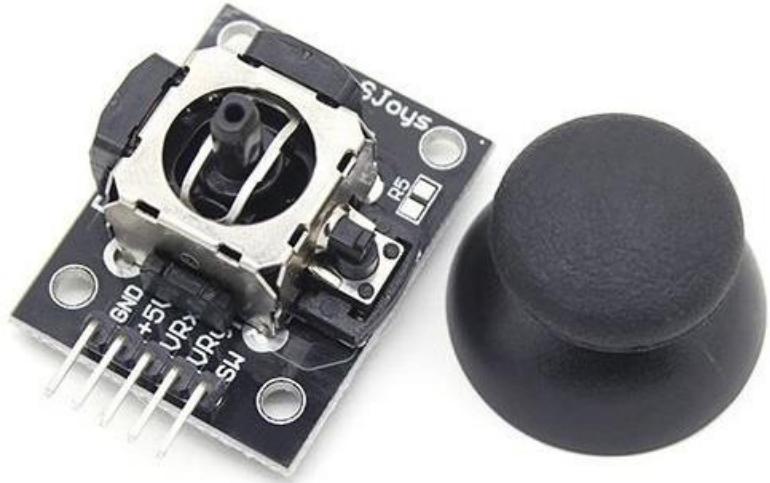


Figura 2.1.4.2

În cadrul proiectului se utilizează joystick-ul atașat shield-ului nRF24L01 Nk 5110.

Shield-ul nRF24L01 Nk 5110



Figura 2.1.4.3

Joystick-ul este utilizat pentru acționarea manuală a motoarelor de către utilizator. Această funcționalitate este disponibilă în modul de funcționare **STATE_MANUAL**.

Implementare hardware – schemă electrică echivalentă

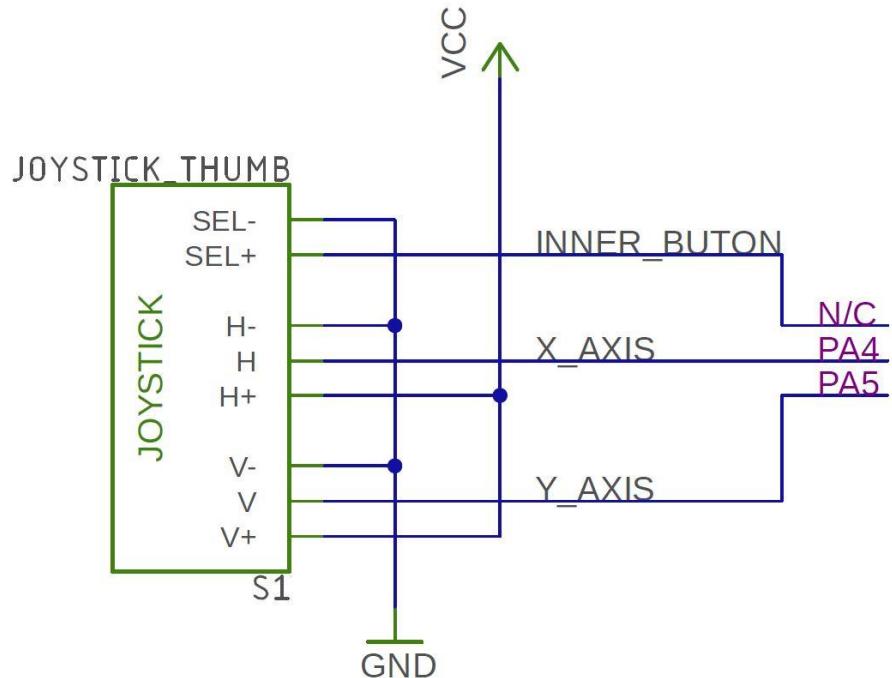


Figura 2.1.4.4

Implementare software – cod sursă

Codul sursă utilizat pentru interpretarea stării butonului este dezvoltat în fișierele joystick_driver.h și joystick_driver.c.

Pentru a putea determina poziția manetei de acționare a dispozitivului joystick, se utilizează două canale ADC (analog-to-digital converter) ale microcontrolerului (canalul 4 pentru axa X, respectiv canalul 5 pentru axa Y).

Configurare microcontroler

```
#define ADC_CHANNEL_X_AXIS 4  
#define ADC_CHANNEL_Y_AXIS 5
```

Figura 2.1.4.5

Funcția **void manual_control(void)** este funcția principală utilizată în modul **STATE_MANUAL** (control manual). Aceasta are rolul de a actiona motoarele ce rotesc și înclină panoul solar, în funcție de datele recepționate de microcontroler de la joystick.

```

void manual_control(void)
{
    unsigned int x = ADC_get_value(ADC_CHANNEL_X_AXIS);
    unsigned int y = ADC_get_value(ADC_CHANNEL_Y_AXIS);

    display_joystick_data(x, y);

    if(y > (JOYSTICK_IDLE_VALUE + JOYSTICK_DEAD_ZONE))
    {
        unipolar_01_step_backward(UNIPOLAR_01_CURRENT_STEP);
    }
    else if(y < (JOYSTICK_IDLE_VALUE - JOYSTICK_DEAD_ZONE))
    {
        unipolar_01_step_forward(UNIPOLAR_01_CURRENT_STEP);
    }
    else
    {
        unipolar_01_clear_steps();
    }

    if(x > (JOYSTICK_IDLE_VALUE + JOYSTICK_DEAD_ZONE))
    {
        1293d_hb2_rotate_right();
    }
    else if(x < (JOYSTICK_IDLE_VALUE - JOYSTICK_DEAD_ZONE))
    {
        1293d_hb2_rotate_left();
    }
    else
    {
        1293d_hb2_stop();
    }

    _delay_ms(25);
}

```

Figura 2.1.4.6

Pentru a elimina situațiile în care motoarele sunt acționate eronat din cauza unor erori de măsurare a poziției manetei de acționare a joystick-ului, funcționalitatea de control manual admite o marjă de eroare, similară cu cea utilizată de dezvoltatorii de jocuri video. În domeniu, această marjă de eroare poartă numele de **dead zone**.

Valorile recepționate de microcontroler sunt cuprinse între 0 și 1023 (ADC-ul are o rezoluție de 10 biți). Valoarea 0 reprezintă limita inferioară a poziției manetei de acționare a joystick-ului (atât pentru axa X, cât și pentru axa Y), iar valoarea 1023 reprezintă limita superioară. Astfel, valoarea 0 reprezintă poziția **stânga maxim** pe axa X și **jos maxim** pe axa Y, în timp ce valoarea 1023 reprezintă poziția **dreapta maxim** pe axa X, respectiv **sus maxim** pe axa Y. În consecință, valoarea de 512 reprezintă poziția de mijloc a manetei joystick-ului. Aceasta este poziția implicită (default, idle) a manetei – când nu este acționată.

Din cauza zgomotelor apărute în sistem, a erorilor de măsurare ale blocului ADC, dar și a construcției mecanice a dispozitivului joystick, poziția implicită a manetei de acționare nu este întotdeauna caracterizată de valoarea 512. Aceasta poate varia cu până la 50 de unități. Pentru a combate acest neajuns, se utilizează o marjă de eroare (**dead zone**) de aproximativ 10% (în jocurile video, marja de eroare standard este cuprinsă între 0% și 20%).

```
#define JOYSTICK_RESOLUTION 10  
#define JOYSTICK_IDLE_VALUE 512  
#define JOYSTICK_DEAD_ZONE 100
```

Figura 2.1.4.7

2.2 Control încărcare

2.2.1 Generalități

Scopul principal al aplicației este de a încărca o baterie caracterizată de tensiunea nominală de 12V cu energia colectată de la soare prin intermediul panoului solar. Pentru aceasta sunt necesare implementarea unor blocuri funcționale dedicate și a unui algoritm de monitorizare și decizie.

Schema bloc a principiului de control al încărcării bateriei

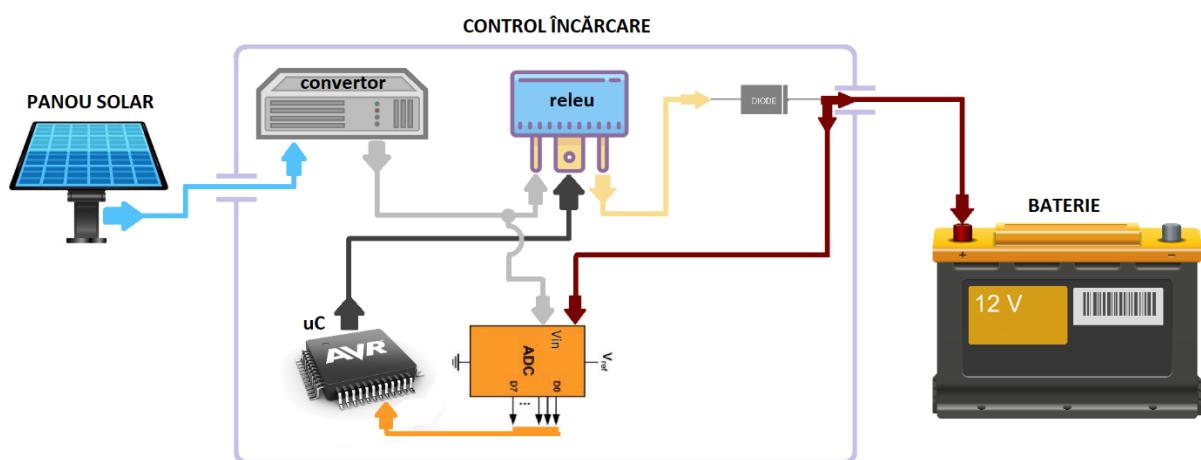


Figura 2.2.1.1

Schema bloc dedicată controlului încărcării este alcătuită din:

- convertorul buck, releu și dioda redresoare cu rol de blocuri funcționale;
- microcontrolerul cu rol de monitorizare și de factor de decizie în controlul blocurilor funcționale;

Funcționalitatea internă ADC a microcontrolerului este folosită pentru monitorizarea tensiunii de ieșire a convertorului și a tensiunii de la bornele bateriei.

Circuitul dedicat acestei funcționalități permite atât încărcarea bateriei cât și utilizarea la nevoie a energiei acumulate, oferind două niveluri de tensiuni de lucru:

- tensiunea de la bornele bateriei, aproximativ 12.4V, cu un consum de curent maxim admis de 10A (putere totală ~120W)
- tensiune continuă stabilizată de 5V, cu un consum de curent maxim admis de 2A (putere totală ~10W)

În plus, pe circuit este montat un port USB (Universal Serial Bus) ce permite alimentarea gadgeturilor casnice uzuale (telefon, tabletă, acumulatori externi, etc.).

Implementare hardware – schematic

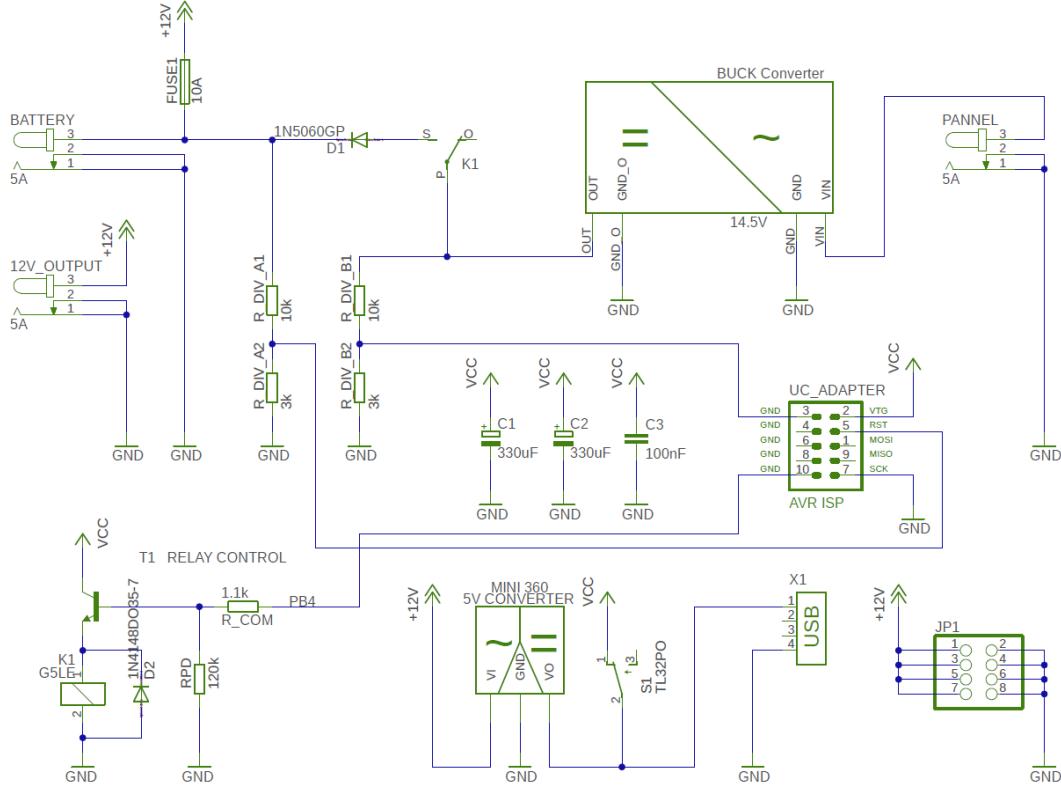


Figura 2.2.1.2

Butonul **S1** (TL32PO) permite alimentarea microcontrolerului de la baterie, prin intermediul convertorului buck **MINI 360**, reglat astfel încât să ofere o tensiune continuă de 5V. Transmisia până la microcontroler se face cu un cablu JTAG cu 10 pini conectat la mufa **UC_ADAPTER** de tip AVR ISP.

Convertorul buck MINI 360

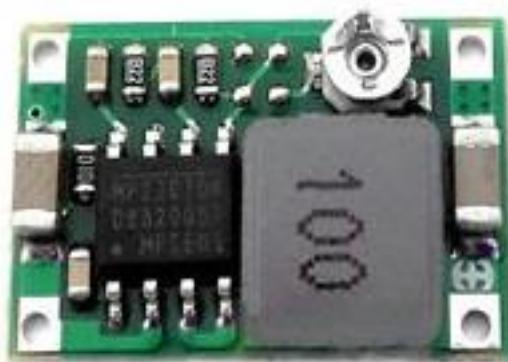


Figura 2.2.1.3

Modulul MINI 360 este un convertor buck special conceput pentru aplicării mobile, de dimensiuni reduse și consum relativ scăzut. Driverul principal al modului este MP2307, un regulator buck sincron monolitic.

Caracteristici MINI 360:

- voltaj de intrare: 4.75V – 23V
- voltaj de ieșire: 1V – 17V
- curent de ieșire: 2A (suportă și vârfuri de până la 3A)
- eficiență de conversie maximă: 96%
- frecvență de operare: 340KHz
- riplu redus: 30mV
- temperatură de operare: -40°C - +85°C
- dimensiuni reduse: 17mm * 11mm * 3.8mm
- protecție la polarizare inversă
- oprire automată la temperatura de 85°C

Tensiunea de ieșire este reglată cu ajutorul potențiometrului montat pe circuit.

Implementarea fizică

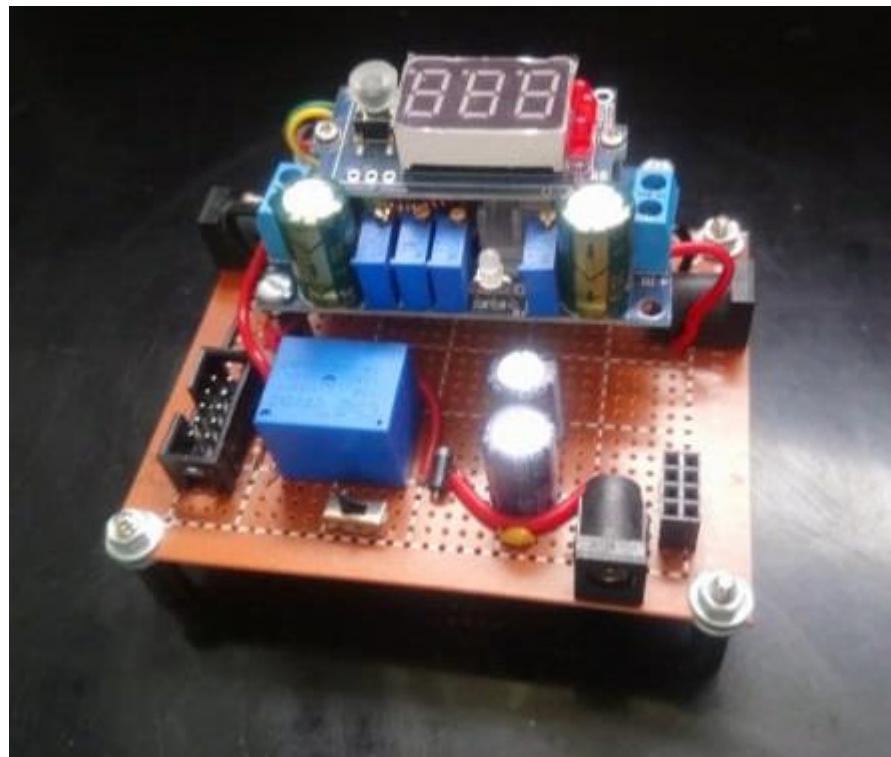


Figura 2.2.1.4

2.2.2 Panoul solar

Panoul solar este un dispozitiv capabil să transforme energia luminoasă din razele solare direct în energie electrică cu ajutorul celulelor fotovoltaice. Energia este de obicei stocată în baterii pentru a putea fi folosita atunci când este necesar. Conversia este statică și nepoluantă, tocmai de aceea constituie un mod ecologic de producere a energiei electric.

Panourile solare fotovoltaice sunt produse în diferite dimensiuni având puteri variate. Cele mai folosite panouri în gama rezidențială sunt cele de 50 și 75 W, iar pentru centrale fotovoltaice de puteri mari, panouri solare de 220W.



Figura 2.2.2.1

Principiu de funcționare

Când lumina lovește un atom, acesta este absorbit de către unul din electronii din jurul acestuia, stimulând energia electronului. Pe anumite materiale (metale, siliciu), această energie este suficient de puternică pentru a separa electronul de atom, permitând electronului să se deplaseze liber în structura cristalină a materialului.

Clasificare panouri solare

- panouri solare monocristaline;
- panouri solare policristaline;
- panouri solare Thin Film (film subțire);
- panouri solare amorfe;
- panouri solare tip țiglă;

Clasificarea panourilor solare este făcută în funcție de tipul celulelor din construcția sa. Celulele solare sunt de mai multe tipuri: monocristaline, policristaline, amorfice, film subțire, CIS și CdTe, CIGS, etc. Diferența între aceste celule constă în structura și modul cum sunt aranjați atomii compoziți. O celulă solară este alcătuită din două sau mai multe straturi de material semiconductor. Cel mai folosit dintre acestea este siliciul.

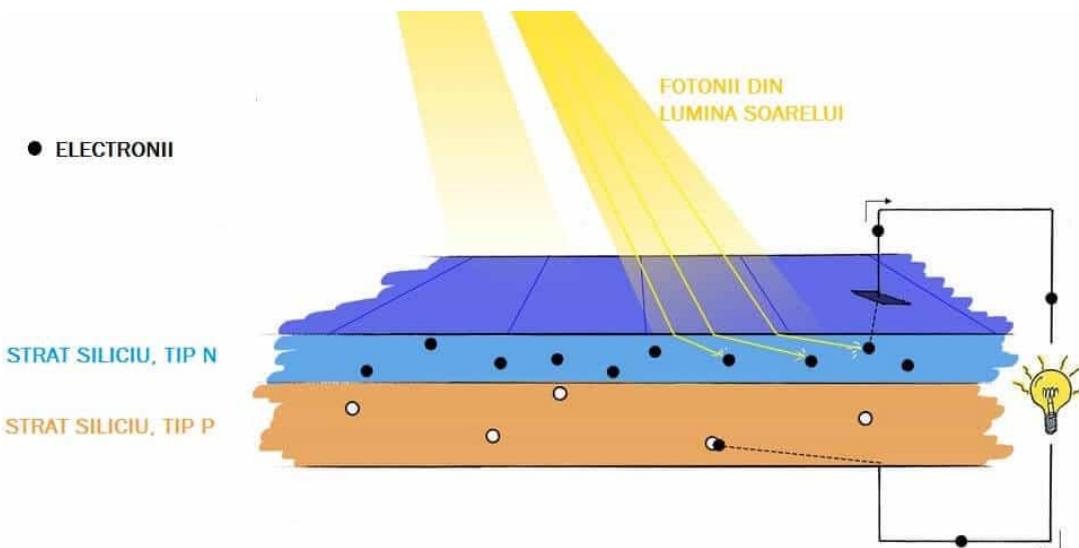


Figura 2.2.2.2

Panoul solar utilizat în proiect este unul potrivit alimentării unei baterii de 12V.

Ghid de selectare corectă a panoului solar în funcție de tensiunea nominală a bateriei

Tensiune baterie	Tensiune panou
1.2V	1.8V~2V
1.5V	2.2V-3.5V
2.4V	3.6V~4V
3.2V	4.5V-5V
3.7V	5.5V-6V
3.7V	5.5V-6V
6V	9V-10V
9V	12V-13.5V
12V	17.2~18V

Figura 2.2.2.3

Panoul solar utilizat este unul de tip monocristalin, fabricat și vândut de YucoSolar Store.

Caracteristici:

- tensiune nominală 18V;
- putere maxima 4.5W;
- dimensiuni 170 x 170 mm;



Figura 2.2.2.4

2.2.3 Convertor

Panourile solare pot genera la ieșire o tensiune prestabilită în etapa de fabricare. De asemenea, această tensiune este dependentă și de factorii externi de mediu: intensitatea luminii solare, gradul de incidență al razelor pe suprafața de contact, temperatură, etc. Din acest motiv, de cele mai multe ori această tensiune trebuie adusă la un anumit nivel în funcție de aplicația cerută. Astfel, tensiunea poate fi ridicată sau coborâtă la nivelul dorit.

Convertoarele de curent continuu care asigura conversia de curent continuu în curent continuu, au la intrare și la ieșire, curenți și tensiuni continue, cu valori diferite. Pentru a îmbunătăți performanțele convertoarelor de curent continuu, se urmăresc două obiective principale:

1. Reducerea dimensiunilor de gabarit

Pentru realizarea acestui obiectiv trebuie reduse dimensiunile dispozitivelor electronice de putere cu rol de comutator, dar și dimensiunile componentelor cu rol de stocare. Micșorarea dimensiunilor componentelor de stocare este permisă prin utilizarea unor frecvențe de lucru ridicate. Pe de altă parte, creșterea frecvenței de lucru conduce la creșterea pierderilor în comutație, iar pentru micșorarea lor se recurge la utilizarea comutațiilor soft care se realizează fie la curent zero, fie la tensiune zero.

2. Creșterea randamentului de conversie;

Realizarea convertoarelor de curent continuu implică utilizarea unui comutator ca și componentă de bază. Acesta trebuie să semene cât mai mult posibil cu un comutator ideal. Un comutator ideal este caracterizat prin curent nul de blocare, cădere nulă de tensiune în comutație și timpi nuli de comutație.

Convertorul coborâtor Buck

Tensiunea generată de panoul solar (18V) este mai mare decât tensiunea necesară încărcării bateriei alese (aproximativ 14.5V). Din acest motiv, în cadrul proiectul se utilizează un convertor de tip buck (coborâtor).

Buck Converter sau Step-Down SMPS este o sursă în comutație de tip coborâtor, unde tensiunea nestabilizată de la intrare este micșorată pentru a produce o tensiune continuă și stabilizată la ieșire.

Schema de bază a unui convertor buck

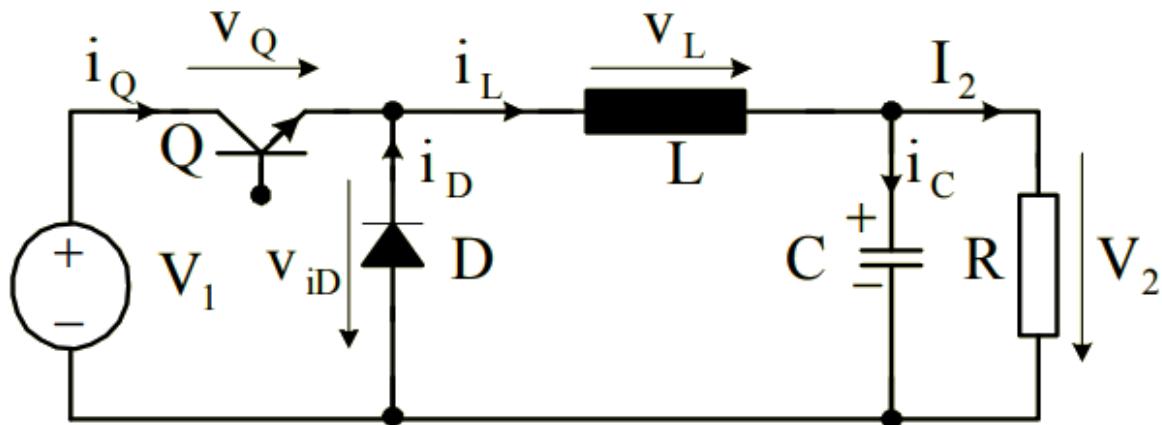


Figura 2.2.3.1

În general, o sursă de putere în comutație constă din cinci componente de bază:

1. Un circuit de control cu modulația impulsurilor în durată(PWM Controller);
2. Un tranzistor cu rol de comutator;
3. O inductanță;
4. O capacitate;
5. O dioda;

Circuitul de control cu modulația impulsurilor în durată este un integrat specializat și are rolul de a furniza semnale de comanda adecvate în scopul reglării și menținerii tensiunii de ieșire la o valoare constantă.

Tranzistorul, pe post de comutator, are rolul de a controla puterea transmisă sarcinii. Acestea pot fi de tip MOS sau bipolar, iar alegerea lor se face în funcție de putere și frecvență.

Inductoarele sunt utilizate cu rol de filtru pentru a reduce răspândirea de curent. Reducerea răspândirii de curent prin inductor nu poate fi schimbată instantaneu. Când curentul prin inductor tinde să scadă, inductorul tinde să-l mențină, având rolul de sursă de energie. Inductoarele utilizate în aceste conversioare sunt înfășurate de obicei pe miezuri toroidale, din fier așchiat cu pierderi reduse la frecvențe înalte.

Capacitatea este utilizată cu rol de filtru pentru a reduce răspândirea de tensiune. Aceasta trebuie aleasă cu pierderi minime. Pierderile din capacitate sunt datorate rezistenței serie și inductanței proprii. Tipul capacitatii este ales după rezistența serie efectivă (ESR). Cele mai indicate capacitate sunt cele din tantal. Uneori, pentru creșterea performanței regulatorului, se leagă în paralel câteva capacitate de valoare mai mică pentru a micșora rezistența serie efectivă.

Dioda folosită este de circulație libera (free-wheeling). Aceasta nu are rol de redresor, ci are funcția de a direcționa corect calea de curent prin inductor. Este important ca dioda să comute în starea de blocat foarte rapid, de aceea se vor folosi diode rapide de recuperare sau diode schottky, care sunt cele mai indicate.

Comanda convertorului

Tranzistorul se comandă cu frecvență $f=1/T$, menținându-se saturat pe o durată dT și blocat pe o durată $(1-d)T$, unde d reprezintă factorul de umplere al semnalului de comandă.

Controlul convertorului buck poate fi făcut în două moduri:

1. Funcționarea la frecvență constantă, sau controlul prin modularea impulsurilor în durată(PWM);
2. Funcționarea la frecvențe variabile, sau controlul prin modularea în frecvență;

Funcționarea convertorului trebuie analizată în două intervale distincte de timp:

- intervalul I, în care tranzistorul Q conduce la saturatie, iar dioda D este blocată, fiind polarizată invers. Considerând originea de timp în momentul comutației directe a lui Q, acest prim interval va fi: $t \in [0, dT]$.

Schemă electrică echivalentă:

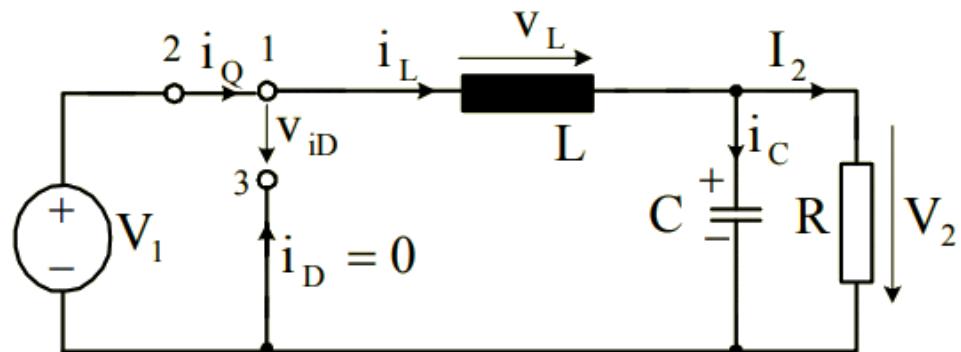


Figura 2.2.3.2

Valori calculate:

$$v_L = V_1 - V_2 = L \frac{di_L}{dt}, t \in [0, dT]$$

$$i_Q = i_L = I_{Lm} + \frac{V_1 - V_2}{L} t, t \in [0, dT] \quad (1)$$

Figura 2.2.3.3

- intervalul II, în care tranzistorul Q este blocat, iar dioda D conduce, asigurând închiderea curentului i_L menținut de la inductanță L.

Schemă electrică echivalentă:

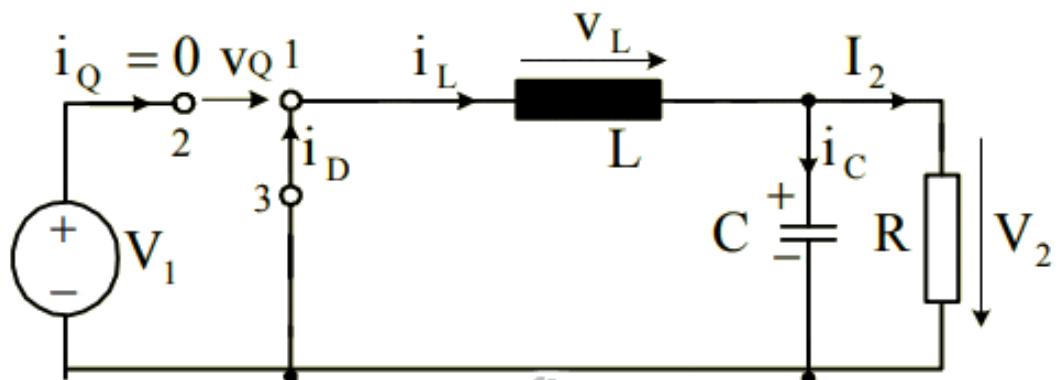


Figura 2.2.3.4

Valori calculate:

$$v_L = -V_2 = L \frac{di_L}{dt}, \quad t \in [dT, T]$$

$$i_D = i_L = I_{LM} - \frac{V_2}{L}(t - dT) \\ t \in [dT, T]$$

Figura 2.2.3.5

Pe baza relațiilor deduse (figura 2.2.3.4 și figura 2.2.3.5) au fost trasate forme de undă. Forma de undă a tensiunii v_L ne permite să deducem caracteristica de reglaj a convertorului. Deoarece valoarea medie a tensiunii pe inductanță L este nulă ($V_{Lavr} = 0$).

Forme de undă specifice convertorului:

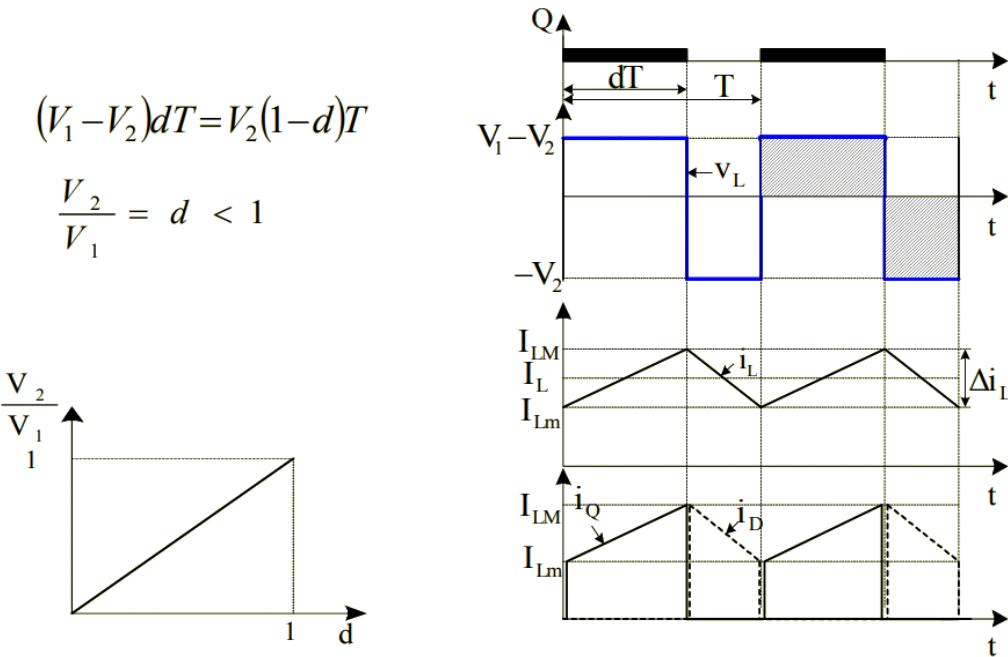


Figura 2.2.3.6

Convertorul buck utilizat în proiect este unul model **YH11087A**. Acesta este prevăzut cu o interfață de comunicare cu utilizatorul formată din:

- un afişaj numeric cu 3 dígi de tip 7 segmente, prin care se indică tensiunea de intrare (se aprinde primul LED din partea stânga), tensiunea de ieşire (se aprinde cel de al doilea LED din partea stângă) sau currentul de ieşire (se aprinde cel de al treilea LED din partea stângă);
- un buton tactil de tip push. La o simplă apăsare, acesta selectează informaţia care va fi afişată pe cei 3 dígi (tensiunea de intrare, tensiunea de ieşire sau currentul de ieşire). La o apăsare de 3 secunde, se va comanda rotirea perspectivei de afişare a datelor de pe afişaj. La o apăsare de 5 secunde, se opreşte funcţionalitatea de monitorizare a tensiunilor şi currentului de ieşire şi se sting cei trei dígi;
- 3 LED-uri de indicare a mărimei afişate pe afişaj;
- un potenţiometru (primul de la stânga la dreapta) prin intermediul căruia se modifică coeficientul lucru al algoritmului MPPT;
- un potenţiometru (al doilea de la stânga la dreapta) prin intermediul căruia se modifică valoarea currentului de ieşire la care se aprinde un LED de avertizare;

- un LED de avertizare (mijloc) ce anunță depășirea unui prag a valorii curentului de ieșire;
- un potențiometru (al treilea de la stânga la dreapta) prin care se reglează valoarea curentului constant de ieșire;
- un potențiometru (al patrulea de la stânga la dreapta) prin care se reglează valoarea tensiunii constante de ieșire;

Convertorul buck MPPT YH11087A



Figura 2.2.3.7

Specificații convertor:

- tensiune de intrare: 6V – 36V;
- tensiune de ieșire: 1.25 – 32V;
- curent de ieșire: 0.05A – 5A;
- prag avertizare depășire curent: 0.01A – 5A;
- interfață cu utilizatorul;
- temperatură de lucru: -40°C - +85°C;
- frecvență de lucru: 180KHZ;
- eficiență maximă: 95%;
- protecție la scurtcircuit

Convertorul poate atinge eficiențe foarte mari, de până la 95% prin utilizarea algoritmului de lucru MPPT (maximum power point tracking – urmărirea punctului de putere maximă). Acest algoritm are ca scop maximizarea puterii extrase. Față de tehnologia precedentă, cea care folosește modul de lucru PWM, tehnica MPPT poate da un randament cu până la 20% mai mare; un procent foarte ridicat când în ceea ce privește generarea energiei electrice nepoluante.

Comparație între regimurile de lucru PWM și MPPT

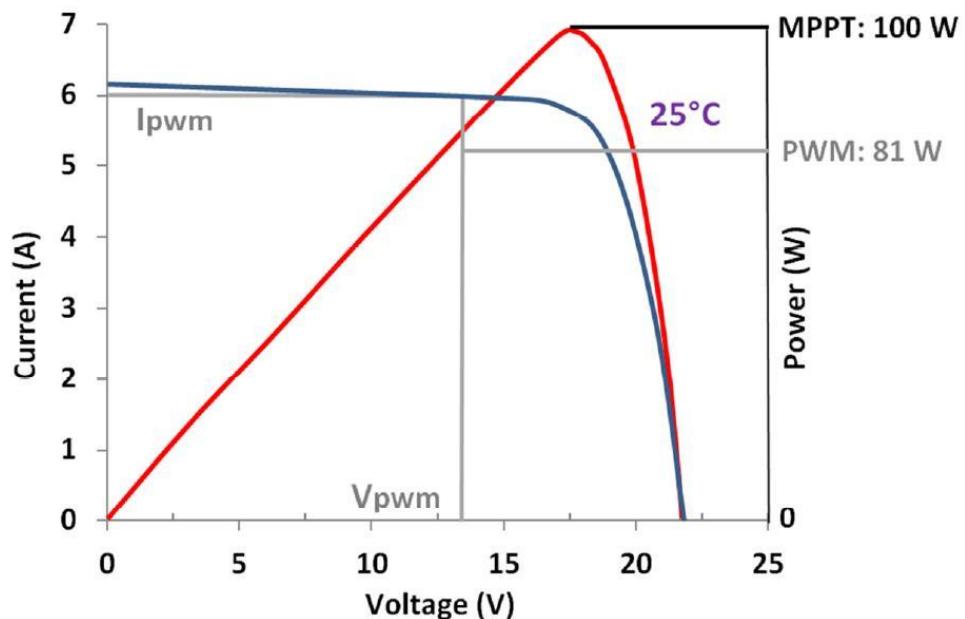


Figura 2.2.3.8

Ieșirea panoului solar este conectată la intrarea convertorului buck print intermediul unei mufe de tip jack 2.1mm x 5mm.



Figura 2.2.3.9

Implementare hardware – schematic

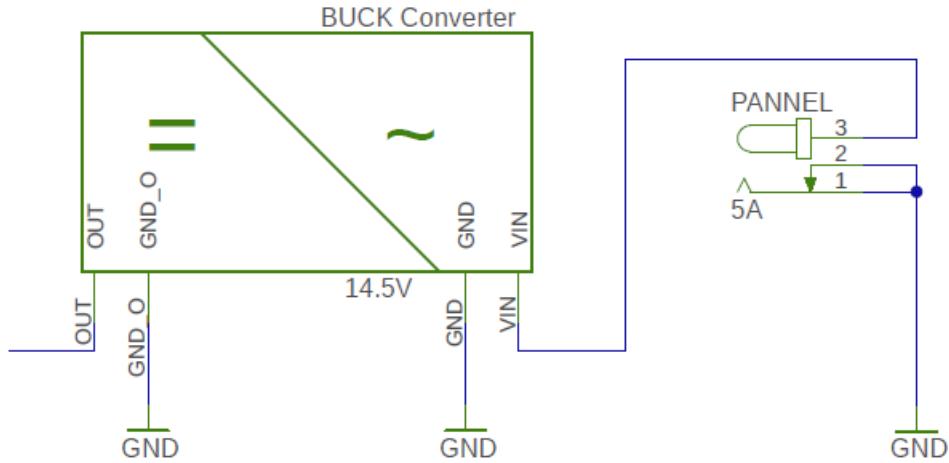


Figura 2.2.3.10

Între ieșirea convertorului și bornele bateriei este conectată o diodă, pentru a respecta indicațiile producătorului.

Implementare hardware – schematic

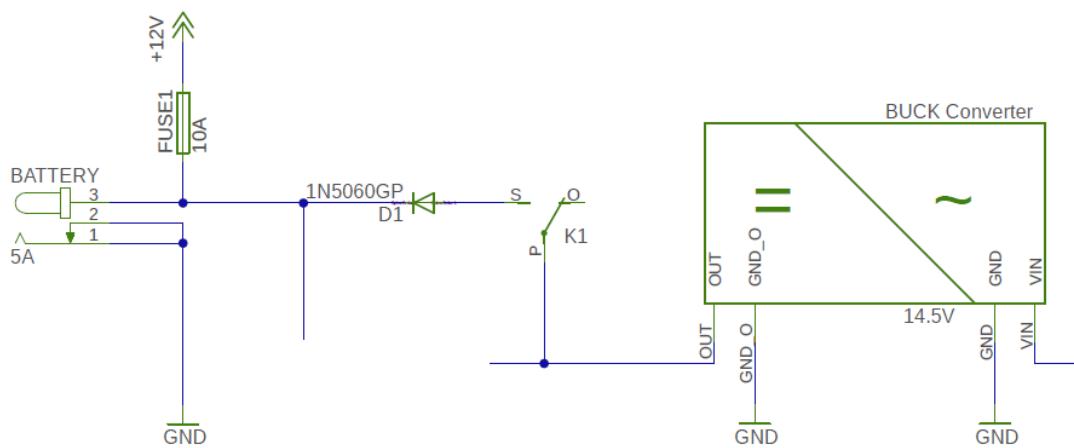


Figura 2.2.3.11

Convertorul buck este setat să ofere o tensiune de ieșire de 15V. Astfel, bateria se încarcă într-un mod similar cu procesul de încărcare al bateriilor auto. La majoritatea vehiculelor, un alternator și un circuit specializat generează o tensiune continuă și stabilizată de 14.4V. Astfel bateria se încarcă într-un mod eficient fără a fi nevoie să se monitorizeze tensiunile și curentii din sistem.

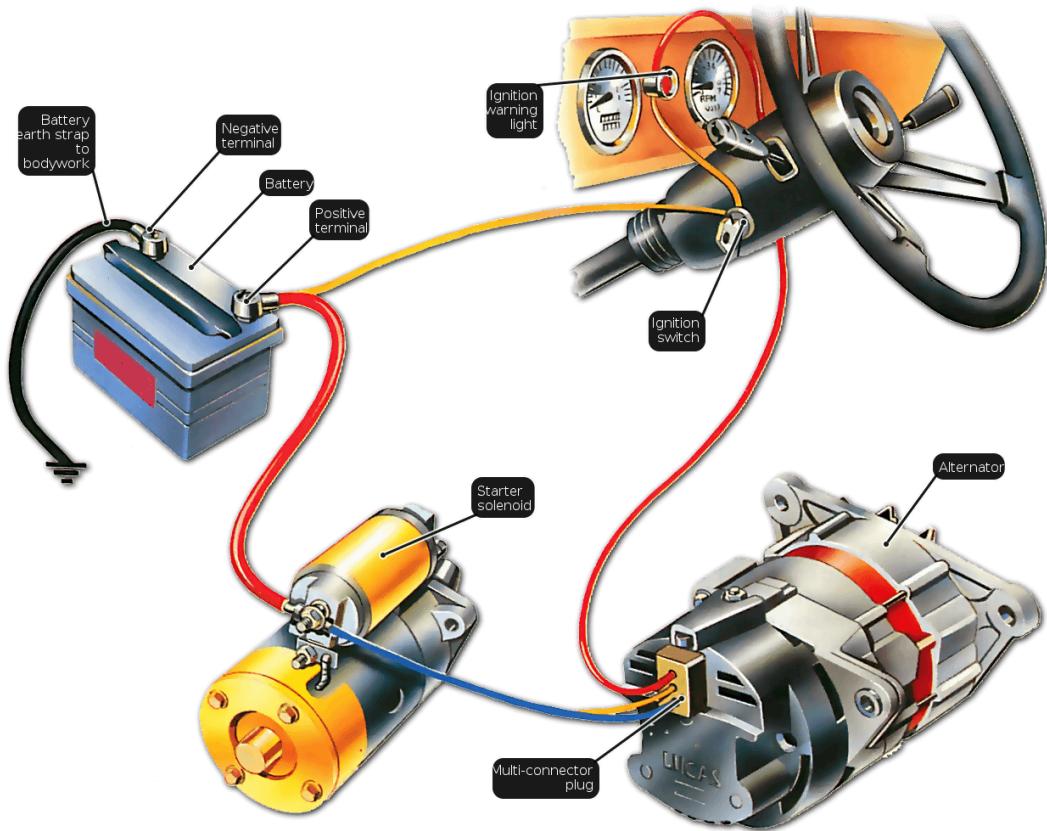


Figura 2.2.3.12

Tensiunea de ieșire a convertorului este monitorizată de microcontroler prin intermediul canalului 7 al blocului ADC intern. Diapazonul de intrare al blocului ADC este de 0V – 5V. Pentru a putea măsura tensiunile generate de convertor, s-a implementat un divizor rezistiv cu factor de divizare ~0.23. Astfel se permit măsurători ale tensiunilor de până la 20V fără a pune în pericol blocul ADC.

Implementare hardware – schematic

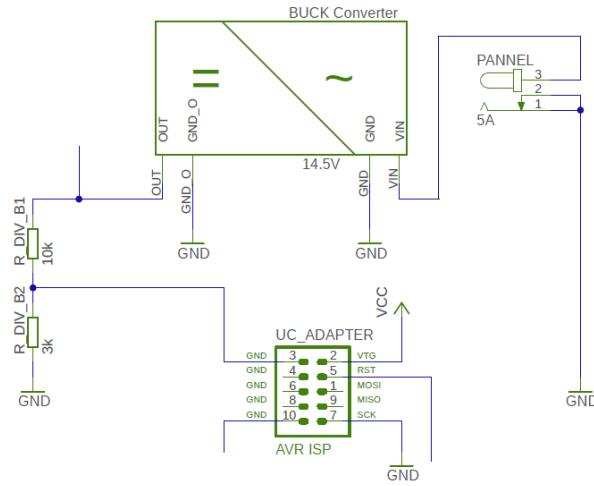


Figura 2.2.3.13

Pentru implementarea divizorului rezistiv s-au utilizat rezistențe cu valorile de $10\text{K}\Omega$, respectiv $3\text{K}\Omega$.

Calcul tensiune maximă divizor

$$V_{in} = 20\text{V} \text{ maxim}$$

$$R_1 = 10\text{K}\Omega$$

$$R_2 = 3\text{K}$$

$$V_o = V_{in} \cdot \frac{R_2}{R_1 + R_2} = 20V \cdot \frac{3000}{10000+3000} = 20V \cdot \frac{3000}{13000} = 20V \cdot 0.23 = 4.61V$$

Implementare software – cod sursă

Configurare microcontroler

```
#define BATTERY_ADC_CHANNEL 6
#define CONVERTER_ADC_CHANNEL 7

#define BATTERY_MAX_MV 20000 // ADC channel uses a 1/4 voltage divider
#define CONVERTER_MAX_MV 20000 // ADC channel uses a 1/4 voltage divider

#define V_BAT_GAIN 975 // as unit of 1000
#define V_CHR_GAIL 975 // as unit of 1000
```

Figura 2.2.3.14

Funcția unsigned int **get_converter_voltage(void)** preia valoarea calculată în blocul ADC de pe canalul 7 și o convertește în valoarea tensiunii oferite de ieșirea convertorului buck în unități de milivolti. Astfel valoarea calculată va fi cuprinsă între 0 și 20000 milivolti. Pentru a combate erorile de măsurare, valoarea este modificată cu un câștig calculat în prealabil. Valoarea câștigul a fost determinată experimental.

```
unsigned int get_converter_voltage(void)
{
    unsigned int battery_voltage = ADC_get_value(CONVERTER_ADC_CHANNEL);
    battery_voltage = battery_voltage*((long)CONVERTER_MAX_MV)/ADC_MAX;
    battery_voltage = battery_voltage*((long) V_CHR_GAIL) / 1000;
    return battery_voltage;
}
```

Figura 2.2.3.15

2.2.4 Bateria

Bateria electrică este un dispozitiv de stocare a energiei electrice sub formă de energie chimică. Procesul este reversibil, astfel că, la conectarea unui consumator la bornele bateriei, energia chimică se eliberează sub formă de energie electrică. Bateria electrică primară este bateria de unică folosință și nu se poate reîncărca, fiind folosită până la epuizare. Bateria electrică secundară numită și acumulator este reîncărcabilă. Bateria poate avea structura bazată pe una sau mai multe celule. Bateriile mari sunt compuse din pachete de baterii mai mici care sunt conectate în serie pentru a obține o tensiune ridicată sau sunt conectate în paralel pentru a debita curent mare.

Principiu de funcționare

Substanțele chimice din interiorul bateriei produc electroni prin transformarea energiei chimice în energie electrică. Atunci când conectăm o baterie la un gadget, electronii se deplasează de la polul negativ al bateriei, prin dispozitivul pe care îl deservește, închizând circuitul prin polul pozitiv al bateriei.

Diferențele dintre diferitele tipuri de baterii sunt date de substanțe chimice aflate în interiorul acestora. Fiecare tip de baterie este conceput astfel încât să deservească unei aplicații dependente de anumite criterii: preț, capacitate, curent maxim, greutate, volum etc.

Tipuri de baterii:

1. Baterii alcaline:

- sunt baterii clasice ce conțin o cantitate rezonabilă de energie;
- acestea se epuizează repede;
- nu sunt reîncărcabile;
- au un cost redus;



Figura 2.2.4.1

2. Bateriile cu oxid de argint (AG):

- sunt baterii cu o viață lungă, dat fiind faptul că acestea conțin o cantitate mare de energie;
- se găsesc sub formă de pastile (nasturi), în ceasurile cu cuarț, dar și în submarine;
- conțin mercur;



Figura 2.2.4.2

3. Bateriile alcaline reîncărcabile:

- după epuizare, aceste baterii se pot încarcă cu ajutorul încărcătorului;
- își pierd câte puțin din capacitatea de stocare a energiei electrice cu fiecare ciclu de reîncărcare;



Figura 2.2.4.3

4. Bateriile reîncărcabile nichel-cadmiu (Ni-Cd):

- sunt baterii reîncărcabile;
- dacă sunt reîncărcate înainte de descărcarea completă, se vor descărca din ce în ce mai repede pe viitor;
- cost mediu;



Figura 2.2.4.5

5. Bateriile nichel-metal hibride (Ni-MH):

- înlocuiesc bateriile nichel-cadmiu care ajuta la stocarea în același spațiu cu până la 40% mai multă energie.



Figura 2.2.4.6

6. Bateriile litiu-ion:

- constituie noul standard în ceea ce privește bateriile pentru gadgeturi, reprezentând un progres considerabil sub aspectul energiei stocate;
- este cel mai întâlnit tip de baterie în telefoanele mobile;



Figura 2.2.4.7

7. Bateriile litiu-polimer (LiPo):

- constituie cel mai înalt standard în ceea ce privește bateriile pentru gadgeturi;
- capacitate de stocare mare;
- volum redus;
- greutate redusă;
- cost ridicat;
- necesită condiții speciale de stocare, utilizare și reîncărcare;
- pericol de explozie sau incendiu;



Figura 2.2.4.8

În realizarea proiectului se utilizează o baterie cu plăci de plumb și acid sulfuric.

8. Bateriile cu plăci de plumb și acid:

- suportă un număr crescut de cicluri încărcare-descărcare;
- furnizează curenți mari;
- sunt rezistente la stres mecanic și electric;
- funcționează la temperaturi extreme;
- circuitele de încărcare sunt relativ simple;
- risc redus de incendiu / explozii;
- cost minim;



Figura 2.2.4.9

Ca și dezavantaje, bateriile cu plăci de plumb și acid sunt foarte grele și ocupă un volum destul de mare. Însă cum proiectul nu este unul mobil sau care să impună restricții de spațiu sau greutate, punctele forte ale acestui tip de baterie o fac alegerea perfectă. Acest tip de baterii este în general utilizat în domeniul auto și pentru UPS-uri (Uninterruptible Power Supply).

Etape de încărcare a unei baterii cu plăci de plumb și acid (în mod ideal)

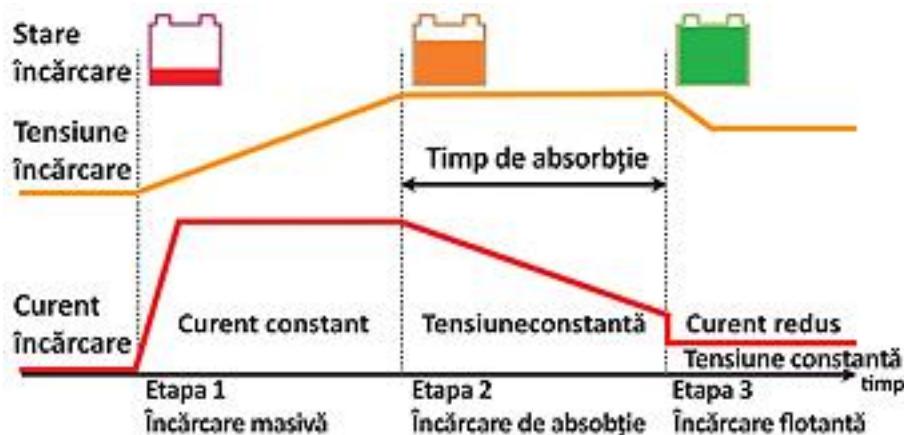


Figura 2.2.4.10

În funcție de nivelul tensiunii bateriei, se ajustează nivelul curentului de încărcare. Dacă bateria este descărcată, în Etapa 1 (bulk) se injectează masiv curent constant până se ajunge la încărcarea de aproximativ 80% și se trece în Etapa 2 când bateria este încărcată spre 99%, iar tensiunea bateriei cuplată la încărcător ajunge până la max. 14,4V. Etapa 3 menține flotant nivelul încărcării aproape de 100%, prin mici injectări de curent, dacă tensiunea de la bornele bateriei scade.

Caracteristicile unui acumulator de calitate:

- curba de descărcare să aibă un platou cât mai mare (sa poată furniza aceeași tensiune de-a lungul întregii perioade de funcționare);
- să poată fi încărcat de cât mai multe ori;
- durata de încărcare cât mai redusă;
- să nu se auto descarce în timp (dacă nu este folosit)
- să fie cât mai ușor;
- să ocupe cât mai puțin volum;
- să conțină cât mai puține substanțe toxice și să fie reciclabil;

În proiectul implementat bateria este utilizată ca sursă principală de alimentare a sistemului. În plus, tensiunea de la bornele bateriei este oferită spre utilizare prin intermediul a patru pini și a unei mufe de tip jack 2.1mm x 5mm.

Tensiunea de la bornele este monitorizată de microcontroler prin intermediul canalului 6 al blocului ADC intern. Diapazonul de intrare al blocului ADC este de 0V – 5V. Ca și în cazul

convertorului buck, pentru a putea măsura tensiunea bateriei, s-a implementat un divizor rezistiv cu factor de divizare ~ 0.23 , pentru a permite măsurători ale tensiunilor de până la 20V fără a pune în pericol blocul ADC.

Implementare hardware – schematic

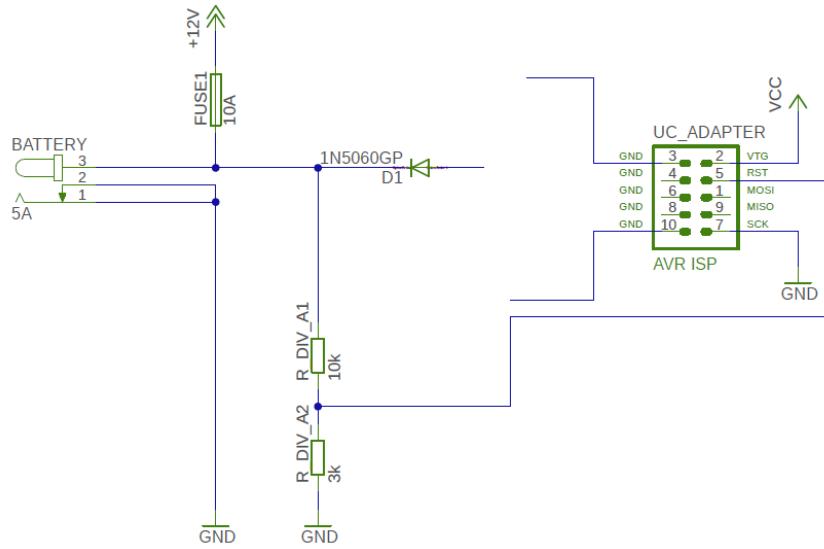


Figura 2.2.4.11

Pentru implementarea divizorului rezistiv s-au utilizat rezistențe cu valorile de $10K\Omega$, respectiv $3K\Omega$.

Calcul tensiune maximă divizor

$$V_{in} = 20V \text{ maxim}$$

$$R_1 = 10K\Omega$$

$$R_2 = 3K$$

$$V_o = V_{in} \cdot \frac{R_2}{R_1 + R_2} = 20V \cdot \frac{3000}{10000 + 3000} = 20V \cdot \frac{3000}{13000} = 20V \cdot 0.23 = 4.61V$$

Implementare software – cod sursă

Configurare microcontroler

```
#define BATTERY_ADC_CHANNEL 6
#define CONVERTER_ADC_CHANNEL 7

#define BATTERY_MAX_MV 20000 // ADC channel uses a 1/4 voltage divider
#define CONVERTER_MAX_MV 20000 // ADC channel uses a 1/4 voltage divider

#define V_BAT_GAIN 975 // as unit of 1000
#define V_CHR_GAIN 975 // as unit of 1000
```

Figura 2.2.4.12

Funcția **unsigned int get_battery_voltage(void)** preia valoarea calculată în blocul ADC de pe canalul 6 și o convertește în valoarea tensiunii de la bornele bateriei în unități de milivolți. Astfel valoarea calculată va fi cuprinsă între 0 și 20000 milivolți. Pentru a combate erorile de măsurare, valoarea este modificată cu un câștig calculat în prealabil. Valoarea câștigului a fost determinată experimental.

```
unsigned int get_battery_voltage(void)
{
    unsigned int battery_voltage = ADC_get_value(BATTERY_ADC_CHANNEL);
    battery_voltage = battery_voltage*((long)BATTERY_MAX_MV) / ADC_MAX;
    battery_voltage = battery_voltage*((long) V_BAT_GAIN) / 1000;
    return battery_voltage;
}
```

Figura 2.2.4.13

2.2.5 Dioda

Dioda este un dipol neliniar și polarizat (sau asymmetric) ce permite trecerea curentului doar într-o singură direcție. Cea mai folosită dioda în circuitele electronice este cea redresoare.

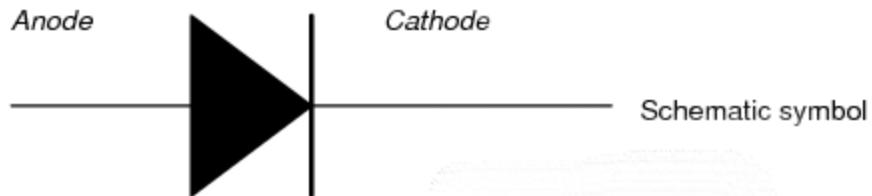


Figura 2.2.5.1

Clasificarea diodelor

- După natura materialului semiconductor folosit sunt:
 - diode cu siliciu;
 - diode cu germaniu.
- După tehnologia de fabricație pot exista în principiu diode:
 - cu contact punctiform;
 - aliate;
 - difuzate;
 - epitaxiale.
- După utilizare se diferențiază următoarele tipuri de diode:
 - redresoare (DR), utilizate pentru conversia de energie din curent alternativ în curent continuu;
 - de comutare (DC), care realizează trecerea rapidă de la starea de conducție la cea de blocare;
 - de semnal (DS), utilizate în circuite de extragere a informațiilor conținute într-un semnal electric care variază în timp, ca de exemplu de detecție și amestec;
 - stabilizatoare de tensiune sau diode Zenner (DZ), care asigură între terminalele lor o tensiune constantă, într-o gamă de curenți specificată;

- varicap (DV), denumite și diode cu capacitate variabilă, la care capacitatea variază cu tensiunea aplicată;
- traductoare, care cuprind: fotodiode (F), diode electroluminiscente (LED);
- speciale, incluzând diode tunel, Schottky, Gunn.

d. După putere, diodele se pot clasifica în diode:

- de putere mica, pentru curenți medii redresați mai mici de 3A;
- de putere medie, pentru curenți cuprinși între 3A și 30A;
- de putere, pentru curenți cuprinși între 30A și 200A;
- de mare putere, pentru curenți peste 200A.

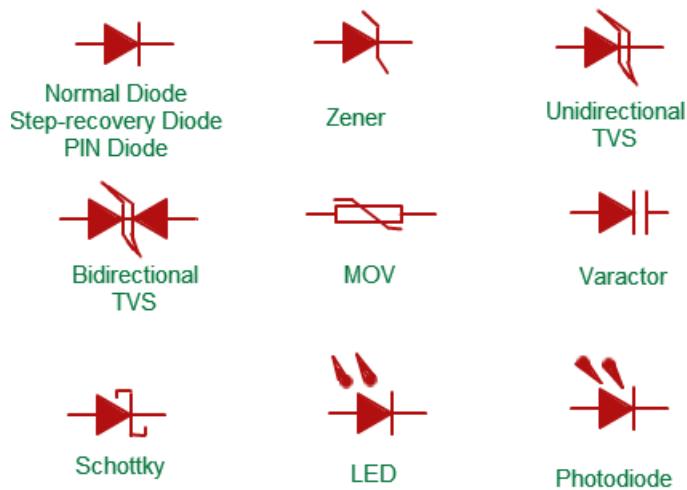


Figura 2.2.5.2

Principiu de funcționare

Dioda permite trecerea unui curent direct (I_d) doar atunci când este polarizată direct. Dacă este polarizată invers, se admite doar trecere unui curent invers minuscul (I_i).

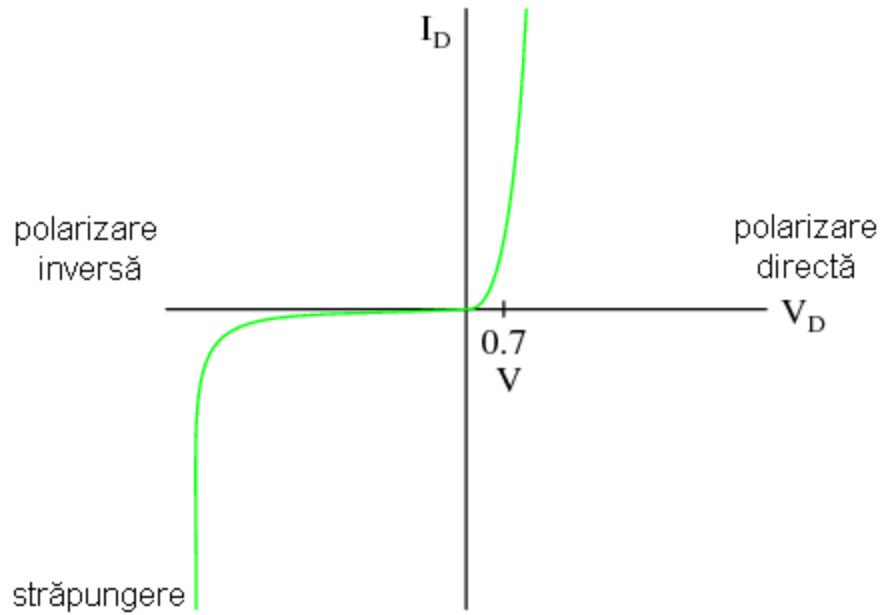


Figura 2.2.5.3

În proiectul realizat, dioda conectată între ieșirea convertorului buck și terminalul pozitiv al bateriei are rolul de a împiedica descărcarea accidentală a bateriei când aceasta nu este în procesul de încărcare sau nu este utilizată, dar și pentru a satisface cerințele producătorului convertorului.

Implementare hardware – schematic

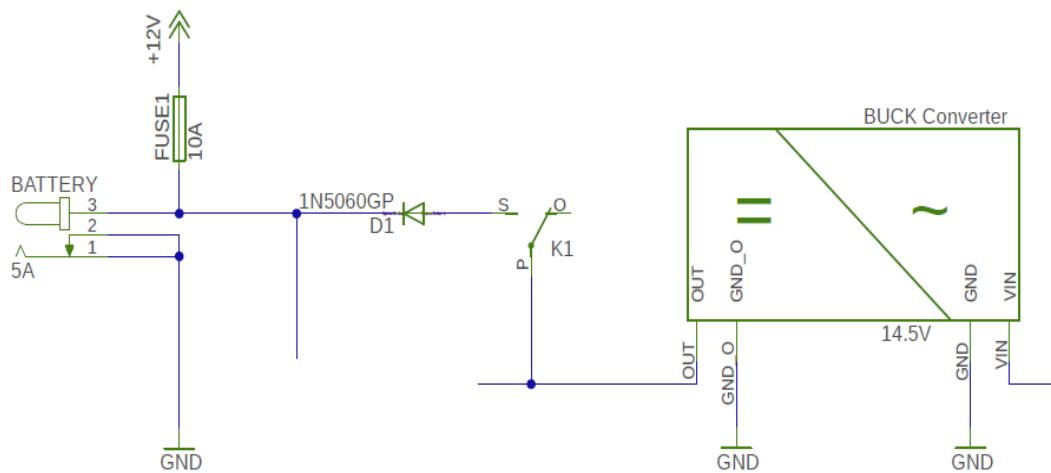


Figura 2.2.5.4

Dioda folosită este una redresoare, de tipul 1N5062. Principalele caracteristici: tensiune de străpungere maxima de 1.15V, tensiune maximă admisă 800V, curent maxim admis 50A.



Figura 2.2.5.3

2.2.6 Releul

Releul este un aparat care, sub acțiunea mărimii de intrare, realizează o variație în salt a mărimii de ieșire, în scopul comenziilor altor elemente.

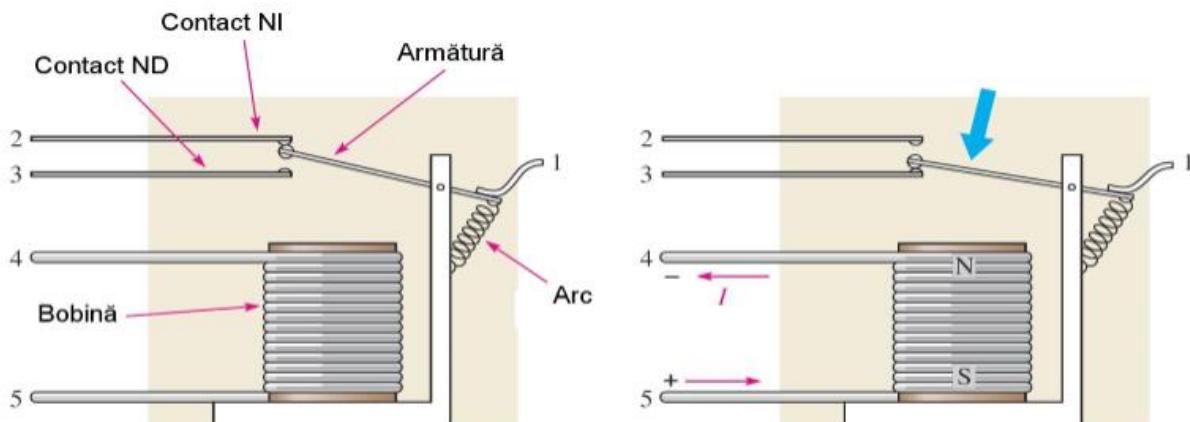


Figura 2.2.6.1

Releele electromagnetice au ca element principal de acționare un electromagnet cu armătură mobilă (ce poate fi basculantă, rotitoare sau cu mișcare de translație). Aceasta acționează asupra contactelor electrice, modificând circuitul electric în care releul este introdus.

Releu neacționat – contact electric între terminalele 1 și 2;

Releu energizat – contact electric între terminalele 1 și 3;

Contactele unui releu electromecanic pot fi contacte normal deschise (ND) sau contacte normal închise (NI). Contactele sunt reținute în aceste poziții de arcuri sau uneori de forță gravitațională.

Brațul mobil, pârghia unui releu, se numește armătură. Armătura este realizată din material magnetic și este acționată de electromagnetul format de bobină. Prin acțiunea asupra

armăturii se deplasează mecanic contactele releului, care se închid (ND) respectiv se deschid (NI) atunci când releul este alimentat. Se mai spune că releul este energizat.

Tipuri de relee:

- Relee electromecanice;
- Relee electromagnetice;
- Relee termice;
- Relee „Reed”;

În acest proiect este folosit un releu electromecanic, cu rolul de a conecta și deconecta ieșirea convertorului buck și bornele bateriei. Releu utilizat este unul model LEG 5.



Figura 2.2.6.2

Implementare hardware – schematic

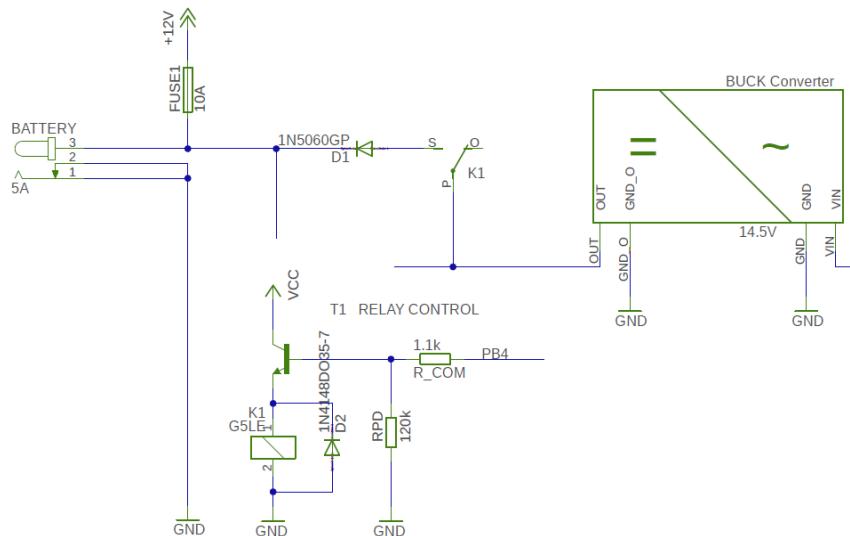


Figura 2.2.6.3

Pentru a comanda releul, se utilizează tranzistorul NPN **T1** (de tip 2N5551). Baza acestuia este controlată prin intermediul pinului PB4 al microcontrolerului. Rezistența **R_COM** are rolul de a proteja baza tranzistorului față de un exces de curent. Prin rezistența **RPD** se asigură ca punctul de comandă nu este flotant, iar tranzistorul nu se va deschide în mod accidental.

Dioda D2 (de tip 1N4148) este folosită pe post de cale de circulație liberă (free-wheeling) și are rolul de a direcționa corect calea de curent prin inductanță internă releului. În lipsa acestei diode, sistemul ar putea primi accidental impulsuri de tensiune cu valori foarte ridicate, periculoase pentru elementele componente sensibile, de exemplu microcontrolerul.

Caracteristicile releului LEG 5

- tensiune de comandă: 5V
- rezistență de izolare: 100MΩ
- rezistență de contact: 100mΩ
- curent admis: 10A
- număr de operații mecanice suportate: 10 000 000

Implementare software – cod sursă

Configurare microcontroler

```
#define RELAY_PIN PINB4
#define RELAY_PORT PORTB
#define RELAY_DDR DDRB
#define TURN_ON_RELAY (RELAY_PORT |= 1 << RELAY_PIN)
#define TURN_OFF_RELAY (RELAY_PORT &= ~(1 << RELAY_PIN))
```

Figura 2.2.6.4

Funcția **void init_charge_control(void)** are rolul de pregăti configurația hardware a microcontrolerului necesară controlului releului.

```
void init_charge_control(void)
{
    RELAY_DDR |= 1 << RELAY_PIN; // Relay pin as output
    RELAY_PORT &= ~(1 << RELAY_PIN); // Relay default state OFF
}
```

Figura 2.2.6.5

Prin intermediul funcțiilor void **turn_on_charging(void)** și void **turn_off_charging(void)** se pornește, respectiv se oprește procesul de încărcare al bateriei prin comanda releului.

```
void turn_on_charging(void)
{
    TURN_ON_RELAY;
}

void turn_off_charging(void)
{
    TURN_OFF_RELAY;
}
```

Figura 2.2.6.6

2.2.7 Disjuncțorul

Disjuncțorul este un aparat electric care întrerupe pe cale mecanică, în mod automat, curentul, conform cu cerințele unei funcționări prestabilite.

În acest proiect se utilizează un disjuncțor cu pragul de activare de 10A, montat pe borna pozitivă a bateriei. Scopul acestuia este de a întrerupe alimentarea principală în cazul apariției unui scurtcircuit. Prin acest montaj, puterea totală ce poate fi oferită de baterie este limitată la aproximativ 120W.



Figura 2.2.7.1

3. Microcontrolerul

3.1 Generalități:

Un microcontroler este un tip de circuit programabil care integrează o unitate de procesare și alte dispozitive periferice într-un singur chip, punându-se accent pe un cost redus de producție și un consum redus de energie electrică.

Elementele componente ale unui microcontroler:

- unitatea centrală de procesare (μ P core) cu o arhitectură care poate lucra pe 8, 16, 32 sau 64 de biți;
- memorie de date volatila (RAM) sau nevolatilă pentru date sau program (Flash sau EEPROM);
- porturi digitale de intrare-iesire;
- interfețe seriale (RS232, SPI, I2C, CAN, RS485).
- temporizatoare, generatoare de PWM sau Watch dog.
- Convertoare analog-digitale.
- Suport pentru programare și debugging (ISP).

În cadrul proiectului s-a utilizat microcontrolerul **Atmega32A**, dezvoltat de compania Atmel, ulterior cumpărată de Microchip.



figura 3.1.1

Microcontrolerul AVR are la bază un procesor RISC cu o arhitectură Harvard (unitatea centrală de procesare are memorie de program și memorie de date separate). Pe baza acestui nucleu RISC firma Atmel a dezvoltat mai multe familii de microcontrolere, cu diferite structuri de memorie și de interfețe I/O (input / output), destinate diferitelor clase de aplicații. Aceste microcontrolere sunt destinate aplicațiilor simple cum ar fi: controlul motoarelor, controlul fluxului de informație pe portul USB, controlul accesului de la distanță (Remote Access Control).

Caracteristici Atmega32A:

- Arhitectura RISC avansată
 - 131 Instrucțiuni ;
 - 32×8 registre de lucru cu scop general;
 - Operație complet statică;
 - Până la 16MIPS, la frecvența de lucru de 16MHz;
- Segmente de memorie nevolatile de înaltă rezistență
 - 32 Kbytes de memorie de program Flash auto-programabilă în sistem;
 - 1024 biți EEPROM;
 - 2Kbytes SRAM intern;
 - Reținerea datelor: 20 de ani la 85°C / 100 de ani la 25°C;
- Operație True Read-While-Write
 - Blocarea programării pentru securitatea software-ului;
- Interfață JTAG
 - Programare Flash, EEPROM, siguranțe și biți de blocare prin interfața JTAG;
- Caracteristici periferice
 - Două timere / contoare de 8 biți;
 - Un timer / contor de 16 biți cu prescaler separat;
 - Contor de timp real cu Oscilator separat;
 - Patru canale PWM;
 - ADC pe 8 canale de rezoluție 10 biți;
- 2 canale diferențiale cu câștig programabil la 1x, 10x sau 200x
 - interfață serială orientată pe două fire;
 - Serial USART programabil;
 - Interfață serială Master / Slave SPI;
 - Timer programabil de supraveghere cu Oscilator separat pe cip;
 - comparator analogic pe cip;
- Caracteristici speciale de microcontroler
 - Resetare la pornire și detectare programabilă de marcare;
 - Oscilatorul RC calibrat intern;
 - Surse externe și interne de întrerupere;
 - Șase moduri de repaus: inactiv, reducerea zgomotului ADC, economie de energie, pornire, oprire și extindere;
- I / O și pachete
 - 32 linii I / O programabile;
 - PDIP cu 40 de pini, TQFP cu 44 de conductori și QFN / MLF cu 44 de cadre;

- Tensiuni de operare
 - 2.7V - 5.5V;
- Frecvențe de operare
 - 0 - 16MHz;
- Consum de putere la 1MHz, 3V, 25°C
 - Activ: 0.6mA;
 - Mod inactiv: 0,2 mA;
 - Mod de pornire: <1µA;

Schema bloc a microcontrolerului

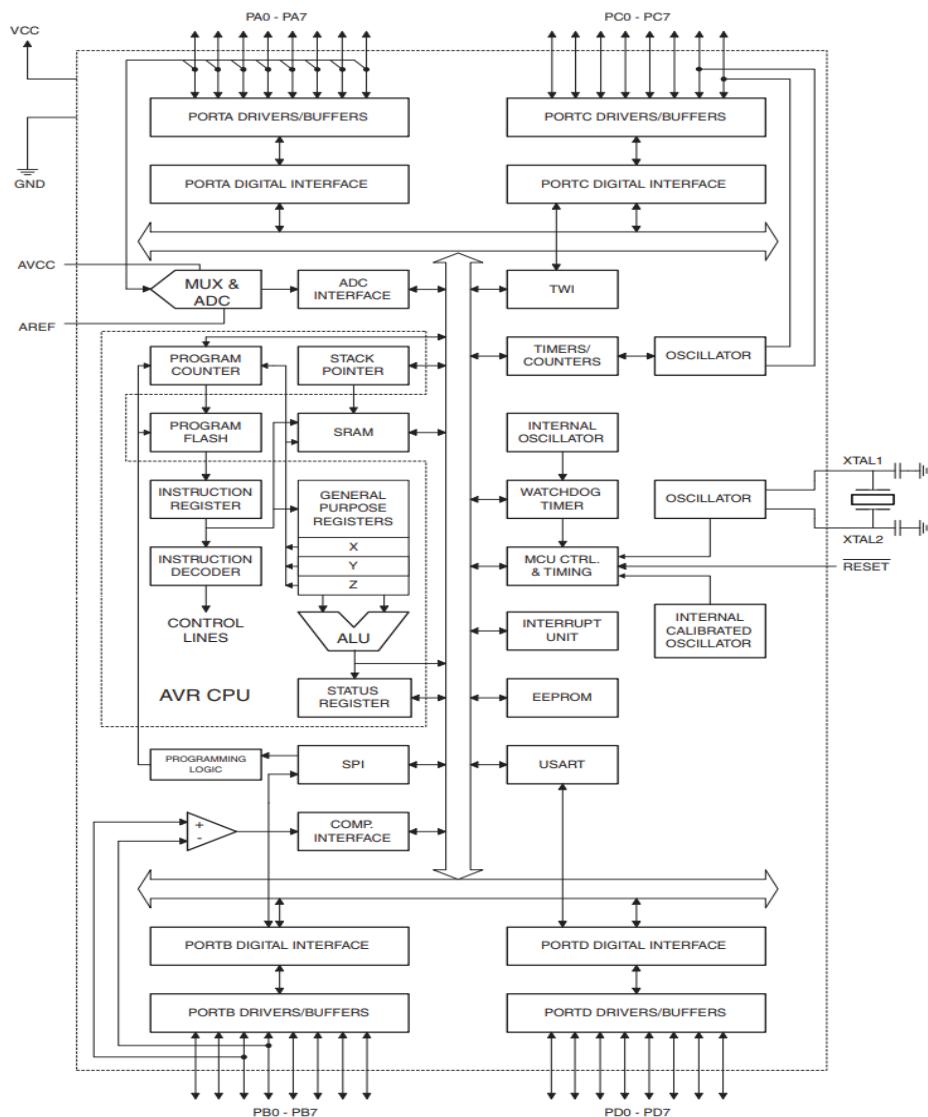


Figura 3.1.2

Descriere pini:

VCC: Tensiune de alimentare digitală;

GND: ground, nivel de referință;

Port A: acest port servește ca intrări analogice al convertorului A / D. De asemenea, servește ca port I / O bidirectional pe 8 biți când convertorul A / D nu este utilizat.

Port B: Este un port I / O bidirectional pe 8 biți. Bufferele sale de ieșire au caracteristici simetrice de acționare, atât cu capacitate ridicată, cât și cu sursa.

Port C: Este un port I / O bidirectional pe 8 biți. Dacă interfața JTAG este activată, rezistențele pull-up de la pinii PC5 (TDI), PC3 (TMS) și PC2 (TCK) vor fi activate.

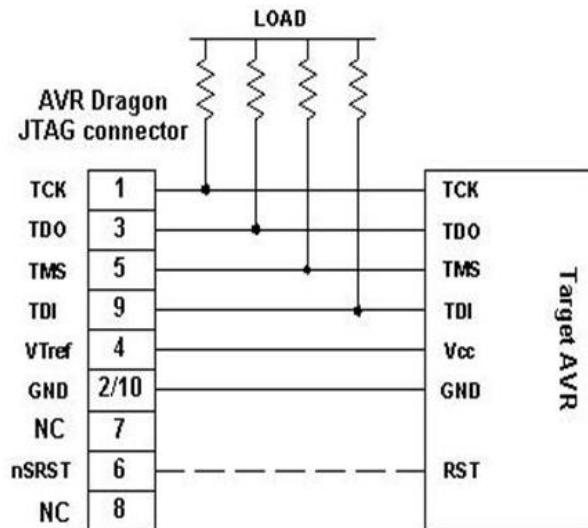


Figura 3.1.3

Reset: resetare microcontroler.

XTAL1: Este o intrare a amplificatorului de oscilator inversat și intrare în circuitul de funcționare a ceasului intern.

XTAL2: Este o ieșire de la amplificatorul oscilator inversor.

AVCC: Aceasta este pinul de tensiune de alimentare pentru Port A și convertorul A / D

AREF: AREF este pinul analogic de referință pentru convertorul A / D.

Implementare hardware – schematic

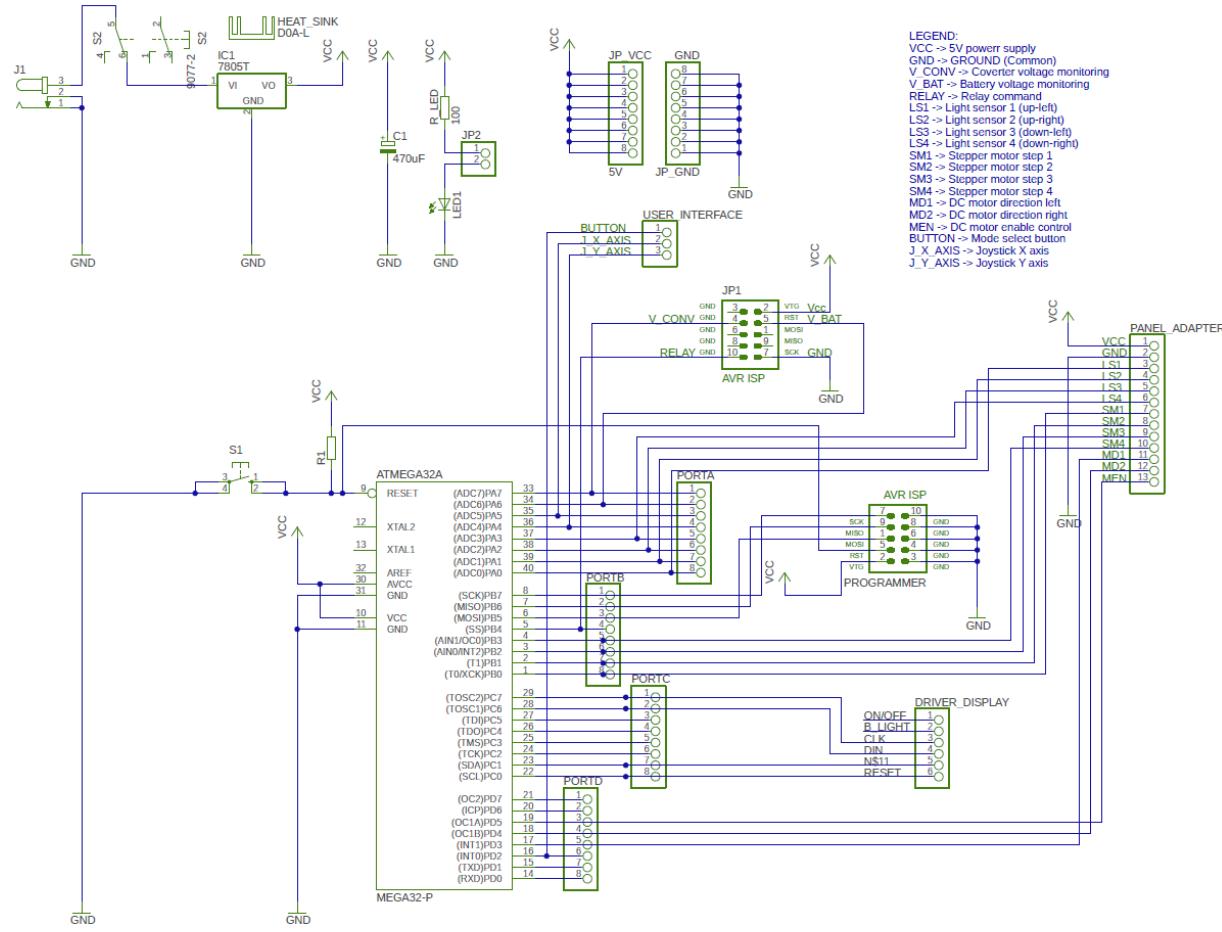


Figura 3.1.4

Pentru a reduce costul total al proiectului și pentru a diminua eforturile de implementare, microcontrolerul folosește oscilatorul RC intern (se elimină necesitatea montării unui cristal de quartz și a celor doi condensatori de filtrare specifici). Acest oscilator este setat să furnizeze un semnal de clock cu frecvența de 8MHz.

Alimentarea microcontrolerului se poate face prin mufa **J1** de tip Jack 2.1mm x 5mm. Tensiunea furnizată (7V – 20V) este adusă la nivel dorit de 5V prin intermediul regulatorului liniar de tensiune 7805.

Întreruperea alimentării se asigură prin acționarea butonului **S2**. Ledul **LED1** indică dacă sistemul este pornit sau oprit. Pe conectorul **JP2** este montat un jumper ce poate fi scos pentru a reduce consumul sistemului, cu mențiunea că elimină funcționalitatea mai sus amintită.

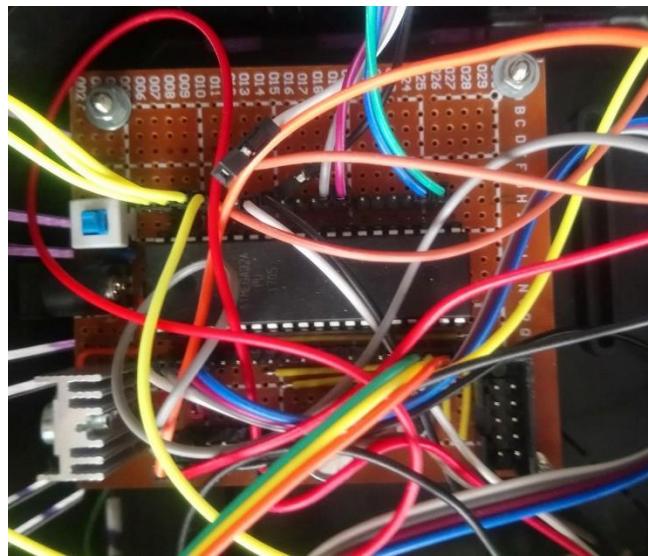


Figura 3.1.5

În plus, microcontrolerul poate fi alimentat de la bateria sistemului prin intermediul convertorului buck **MINI 360**, descris în capitolul 2.2.1.

Pentru a preveni resetarea accidentală a microcontrolerului, pinul său de reset este conectat prin intermediul rezistenței de pull-up **R1**. Pentru a forța resetarea microcontrolerului se acționează butonul **S1**.

Programarea microcontrolerului se realizează print interfața **AVRISP 10 PIN**.
Programatorul se conectează la microcontroler prin intermediul soclului **PROGRAMMER**.

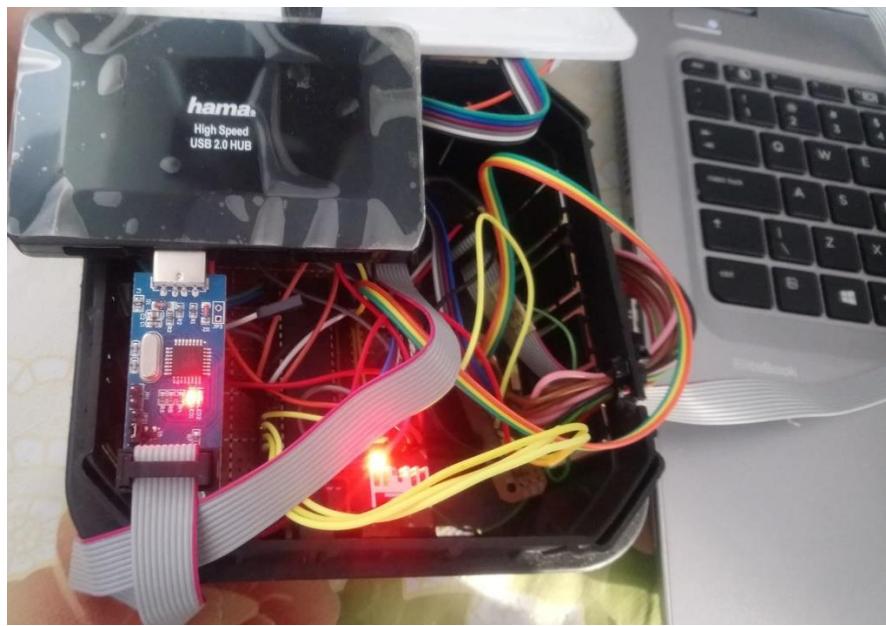


Figura 3.1.6

3.2 ADC

Convertorul analogic-digital (ADC) este un circuit electronic integrat folosit pentru a converti semnalele analogice, cum ar fi tensiunile, în formă digitală sau binară.

În principal, există doi pași pentru conversia analog-digital:

S / H: Eșantionare și reținere

Q / E: cuantificarea și codificarea

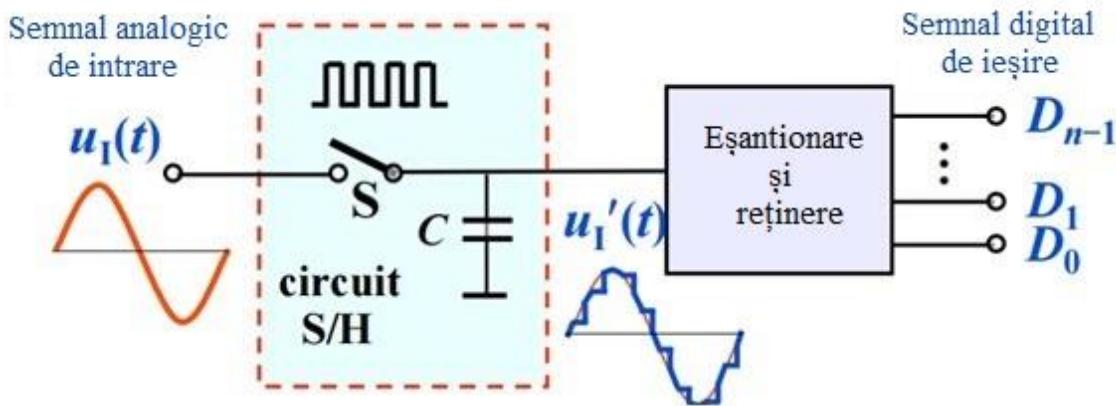


Figura 3.2.1

S / H: Eșantionare și reținere:

Un semnal analog se schimbă continuu în timp. Pentru a măsura semnalul, acesta trebuie păstrat constant pentru o scurtă durată, astfel încât să poată fi eșantionată.. Aceasta se face printr-un circuit de eșantionare și de așteptare.

Q / E: cuantificarea și codificarea:

Pe ieșirea (S / H), există un anumit nivel de tensiune. Acestuia i se atribuie o valoare numerică. Se caută cea mai apropiată valoare, în corespondență cu amplitudinea eșantionării și a semnalului de reținere. După identificarea celei mai apropiate valori, îi este atribuită o valoare numerică și este codificată sub forma unui număr binar.

În sistemul implementat există numeroase mărimi fizice ce trebuie interpretate cu ajutorul unui bloc funcțional ADC. Printre acestea, se enumera: intensitatea luminoasă, poziția manetei de acționare a joystick-ului, tensiunea de la bornele bateriei și tensiunea de la ieșirea convertorului buck.

Microcontrolerul **Atmega32A** este echipat cu un circuit ADC cu referință de tensiune configurabilă. Acesta funcționează prin aproximări succesive și are o rezoluție configurabilă de până la 10 biți. Intrarea blocului ADC este conectată la un multiplexor analog ce permite citirea a 8 canale diferite, fiecare conectată pe pinii PA0, PA1, ..., PA7 ai portului A.

Implementare software – cod sursă

Cu ajutorul funcției **void ADC_init(void)** se pregătește configurația hardware necesară microcontrolerului pentru a utiliza blocul ADC intern.

```
void ADC_init(void)
{
    DDRA = 0x00;
    // AREF = AVcc
    ADMUX = (1<<REFS0);

    // ADC Enable and prescaler of 128
    // 16000000/128 = 125000
    ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
}
```

Figura 3.2.2

Prin intermediul registrului **DDRA**, portul A este configurat ca port de intrare și este conectat la multiplexorul analogic al blocului ADC. În registrul **ADMUX** se setează bitul **REFS0** pentru a indica pinul AVCC ca referință de tensiune. Prin registrul **ADCSRA** se activează blocul ADC și se configerează un prescaler de 128.

Referințe din catalogul tehnic al microcontrolerului

REFS[1:0]	Voltage Reference Selection
00	AREF, Internal V_{ref} turned off
01	AV_{CC} with external capacitor at AREF pin
10	Reserved
11	Internal 2.56V Voltage Reference with external capacitor at AREF pin

Figura 3.2.3

Bit	7	6	5	4	3	2	1	0
Access	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
Reset	0	0	0	0	0	0	0	0

Bit 7 – ADEN: ADC Enable

Writing this bit to one enables the ADC. By writing it to zero, the ADC is turned off. Turning the ADC off while a conversion is in progress, will terminate this conversion.

Figura 3.2.4

ADPS[2:0]	Division Factor
000	2
001	2
010	4
011	8
100	16
101	32
110	64
111	128

Figura 3.2.5

Funcția **uint16_t ADC_get_value(uint8_t ch)** este utilizată pentru a porni procesul de conversie al blocului ADC și pentru a prelua valoarea returnată de acesta.

```
// read ADC value
uint16_t ADC_get_value(uint8_t ch)
{
    // select the corresponding channel 0-7
    // ANDing with '7' will always keep the value
    // of 'ch' between 0 and 7
    ch &= 0b00000111; // AND operation with 7
    ADMUX = (ADMUX & 0xF8)|ch; // clears the bottom 3 bits before ORing

    // start single conversion
    // write '1' to ADSC
    ADCSRA |= (1<<ADSC);

    // wait for conversion to complete
    // ADSC becomes '0' again
    // till then, run loop continuously
    while(ADCSRA & (1<<ADSC));

    return (ADC);
}
```

Figura 3.2.6

Prin intermediul parametrului **ch**, de tip `uint8_t` (`unsigned char`), se indică canalul multiplexorului analogic ce va fi conectat la ADC în momentul conversiei. Valoarea returnată este de tip `int` (întreg) de dimensiune 16 biți.

3.3 Software

Codul sursă pentru această aplicație a fost scris în mediul de dezvoltare Atmel Studio 7, oferit de Microchip, utilizând ca limbaj de programare C/C++. Ca și complexitate, partea logică a aplicației se încadrează la un nivel mediu. Astfel a fost de preferat ca fiecare funcționalitate să fie implementată într-un grup de fișiere propriu; fiecare grup este constituit dintr-un fișier .c ce conține corpul funcțiilor dezvoltate și un fișier .h, (header) care cuprinde prototipurile funcțiilor definite în fișierul anterior menționat. Funcțiile implementate vor putea fi accesate în cod utilizând directiva de preprocesare **#include**.

Lista modulelor software implementate:

main.c	
adc_driver.c	
adc_driver.h	monitoring.c
charge_driver.c	monitoring.h
charge_driver.h	pwm_driver.c
global.h	pwm_driver.h
graphics.c	sg90_driver.c
graphics.h	sg90_driver.h
hx1230.c	state_handler.c
hx1230.h	state_handler.h
hx_8x6_characters.h	tracking.c
joystick_driver.c	tracking.h
joystick_driver.h	user_interface.c
light.c	user_interface.h
light.h	

Figura 3.3.1

După dezvoltarea codului C/C++, acesta este verificat, compilat și ulterior se generează codul mașină util microcontrolerului în format HEX. Codul în format HEX este cel încărcat pe microcontroler cu ajutorul programatorului USBASP.

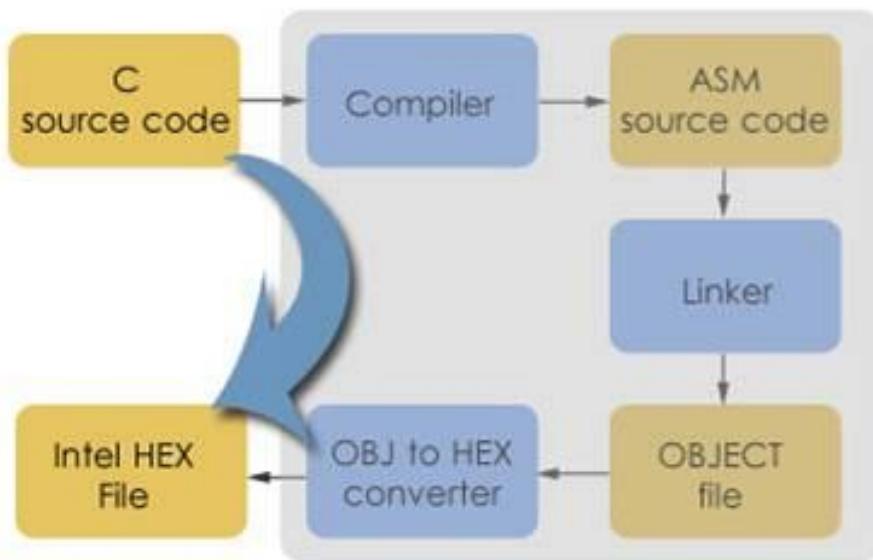


Figura 3.3.2

5. Analize si rezultate

Pentru a asigura o bună funcționare a sistemului, de-a lungul dezvoltării acestuia s-au efectuat diferite teste de funcționalitate. Probele de osciloscop au fost obținute cu ajutorul osciloscopului Hantek DSO 5102P.

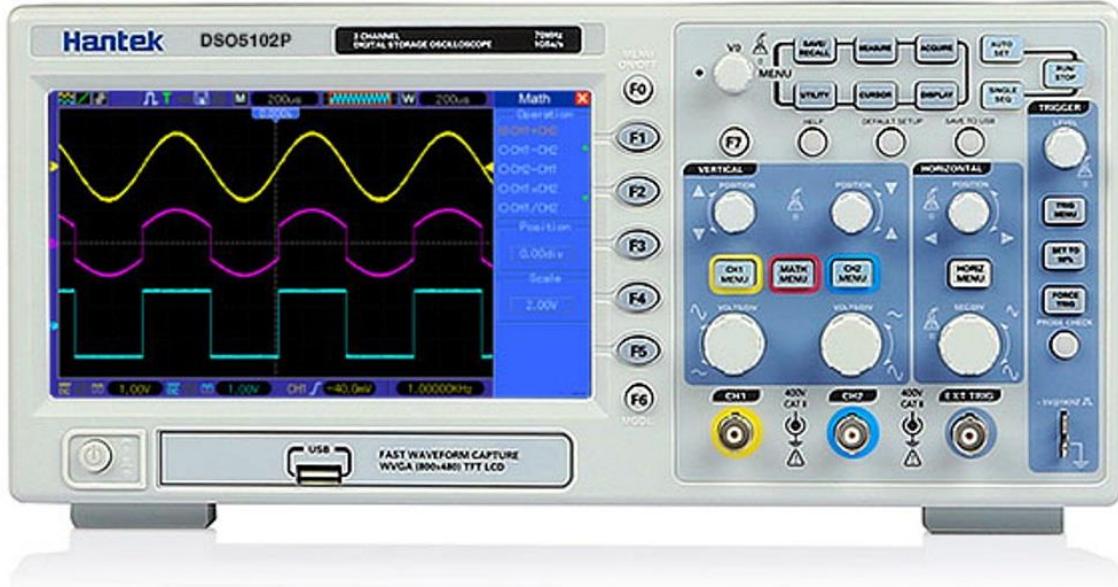


Figura 5.1

5.1 Convertor

Pentru convertorul buck **YH11087A** s-a urmărit capacitatea acestuia de a furniza o tensiune constantă la ieșire, indiferent de fluctuațiile apărute din partea panoului solar. Pentru a simula tensiunea generată de panoul solar s-a utilizat o sursă de laborator.

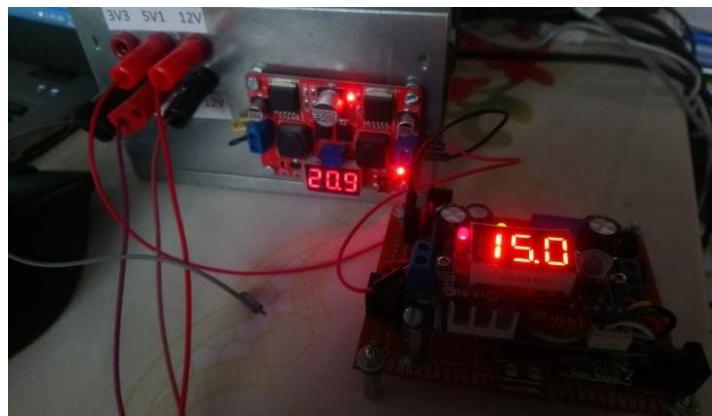


Figura 5.1.1

În cadrul acestui test, tensiunea de ieșire a convertorului buck a fost setată la valoarea de 15V. Tensiunea generată de sursa de laborator a fost modificată între valorile 15.5V și 23V.

Proba de osciloscop:

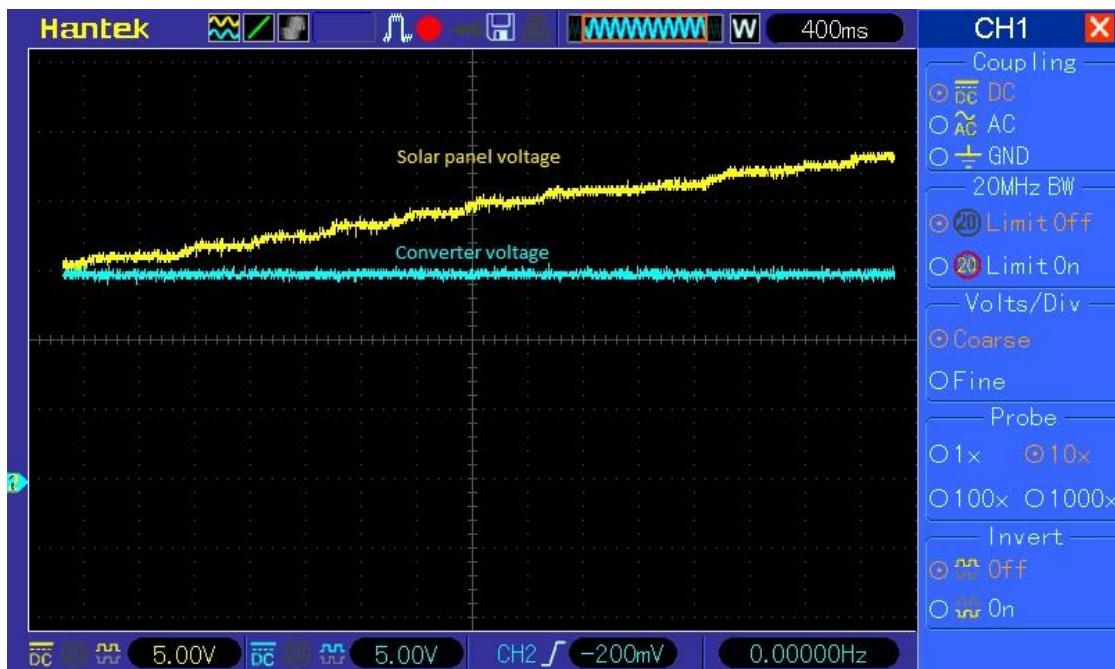


Figura 5.1.2

În rezultatele obținute (figura 5.1.2) se poate observa că performanțele convertorului îndeplinesc criteriile cerute.

În strânsă legătură cu performanțele convertorului, s-a mai rulat un test de verificare a reacției sistemului la generarea unei tensiuni de ieșire (de către convertor) a unei tensiuni mai mici decât cea necesară încărcării bateriei.

În cadrul acestui test, tensiunea de ieșire a convertorului buck a fost setată la valoarea de 15V. Tensiunea generată de sursa de laborator a fost modificată descrescător de la valoarea de 18.5V până 10V.

Conform cerințelor, sistemul trebuie să detecteze o tensiune de încărcare mai mică decât cea necesară și să întrerupă procesul de alimentare al bateriei prin intermediul releului.

Pe parcursul testului s-a monitorizat pinul PB4 al microcontrolerului, pin responsabil de acționarea releului.

Proba de osciloscop:

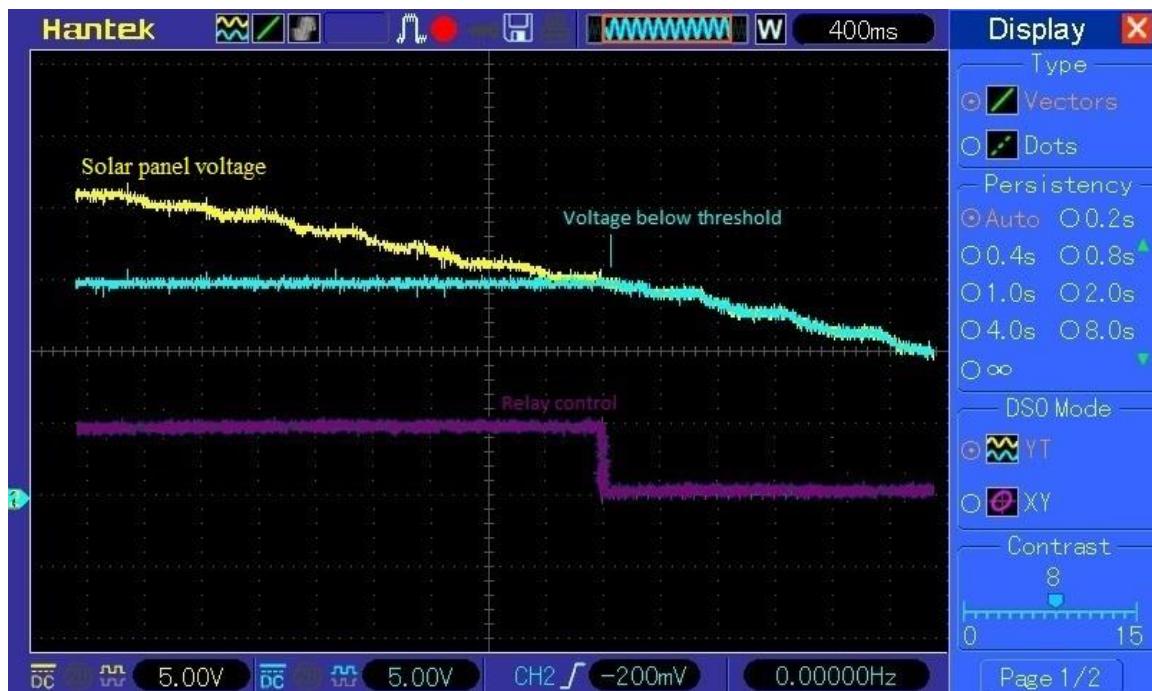


Figura 5.1.3

În rezultatele obținute (figura 5.1.3) se poate observa că sistemul reacționează corect, conform cerințelor.

5.2 Interfață utilizator

În cadrul testelor dedicate interfeței cu utilizatorul, sau urmărit performanțele algoritmilor responsabili de controlul display-ului și cei de interpretare a poziției manetei de actionare a joystick-ului.

Pentru a verifica corectitudinea datelor și comenziilor transmise către display, s-a comandat afișarea unei imagini complexe, compusă din pixeli activați la coordonate specifice.

Imaginea afișată de către display este cea dorită (figura 5.2.1).

Imaginea obținută:



Figura 5.2.1

Un alt test s-a rulat pentru a verifica performanțele algoritmului de interpretare a poziției manetei de acționare a joystick-ului, pe baza datelor primite prin intermediul blocului ADC de la canalele 4, respectiv 5.

Datele interpretare au fost monitorizate în paralel cu ajutorul osciloscopului.

Datele afișate de display:



Figura 5.2.2

Conform indicațiilor producătorului joystick-ului, tensiunile de ieșire de pe fiecare canal (axa X și axa Y) au valorile egale cu jumătate din valoarea tensiunii de alimentare a modului.

Pentru fiecare axă, acționarea manetei joystick-ului va determina generarea unei tensiuni direct proporționale cu poziția acesteia.

Proba de osciloscop:

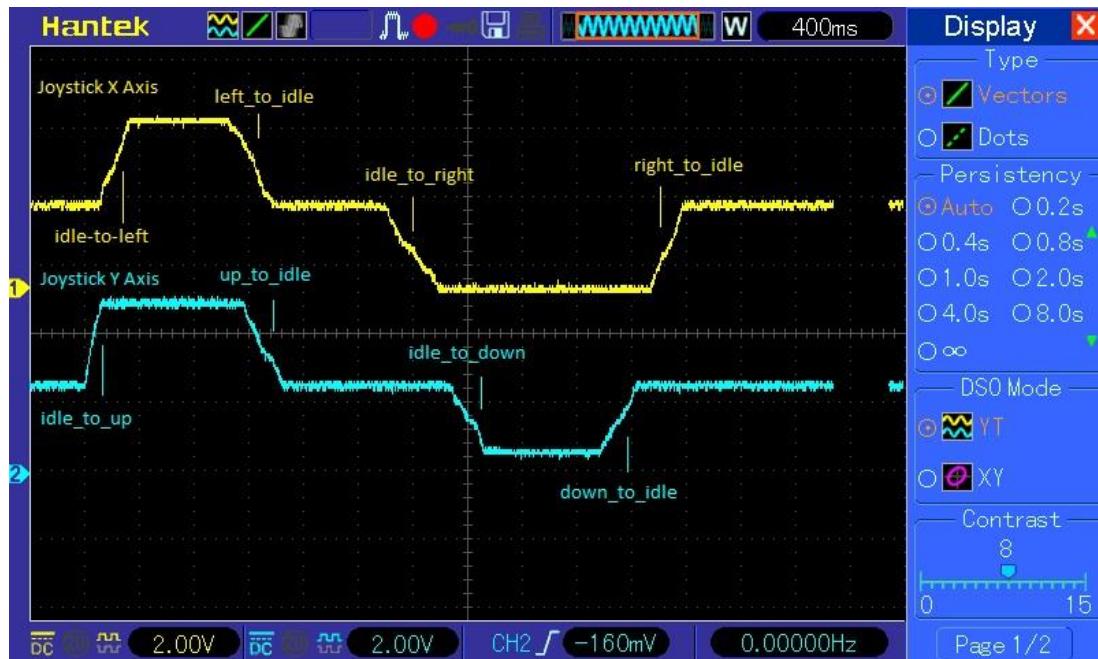


Figura 5.2.3

Conform rezultatelor obținute (figura 5.2.2 și figura 5.2.3), joystick-ul respectă specificațiile producătorului și algoritmul de interpretare a poziției manetei de acționare funcționează corect.

6.Concluzie:

Proiectul de licență realizat pune în evidență etapele parcurse în vederea construirii unui sistem autonom care convertește energia luminii solare în electricitate cu ajutorul unui panou solar.

De-a lungul lucrării s-au prezentat rând pe rând blocurile componente ce au fost utilizate în cadrul aplicației.

Cu ajutorul senzorilor utilizați, sistemul este capabil să determine poziția optimă a panoului solar astfel încât să capteze cât mai multă energie luminoasă. Pentru a poziționa panoul solar, două motoare sunt folosite în cadrul aplicației: un motor pas cu pas și un motor de curent continuu.

Energia electrică furnizată de panoul solar este utilizată pentru încărcarea eficientă a unei baterii de 12V. În plus s-au dezvoltat o serie de măsuri de protecție pentru sistem și anume: protecție la supratensiune, supraîncărcare, supracurent și descărcare accidentală a bateriei.

Pentru a putea monitoriza funcționarea sistemului, s-a pus la dispoziție un display care afișează parametrii de interes în timpul funcționării.

Funcționarea corecta a aplicației este pusă în evidență pe parcursul lucrării prin intermediul capturilor de osciloscop ce au fost realizate în diverse stări de funcționare a sistemului.

Bibliografie

- https://ro.wikipedia.org/wiki/Interfa_grafica
<http://labs.cs.upt.ro/labs/so/html/so1.html>
<http://extremeelectronics.co.in/avr-tutorials/gui-software-for-usbasb-based-usb-avr-programmers/>
https://en.wikipedia.org/wiki/Display_driver
<http://www.rasfoiesc.com/inginerie/electronica/SIGURANTE-FUZIBILE-Principiul-43.php>
<https://ro.wikipedia.org/wiki/Interfa%C8%9B%C4%83>
https://ro.wikipedia.org/wiki/Afi%C8%99aj_cu_cristale_lichide
<https://components101.com/nokia-5110-lcd>
https://ro.wikipedia.org/wiki/Interfa%C8%9Ba_serial%C4%83_SPI
<https://ro.wikipedia.org/wiki/Buton>
<http://alternativepureenergy.ro/despre/panouri-fotovoltaice/>
<http://energie-verde.ro/produse/panouri-fotovoltaice-2/>
<http://solarcenter.ro/blog/cum-functioneaza-panourile-solare-fotovoltaice/>
<http://www.esolar.ro/sfaturi-utile/toate-tipurile-de-panouri-solare-avantajele-si-dezavantajelelor.html>
<http://ep/etc.tuiasi.ro/site/Electronica%20Industriala/referate%20laborator/L09%20-%20Convertor%20buck%202007.pdf>
<http://www.scientia.ro/tehnologie/gadgeturi/1091-scurt-ghid-despre-baterii.html>
<http://electronica-azi.ro/2013/12/11/bateriile-electrice-reincarcabile-pb-acid/>
<https://biblioteca.regielive.ro/referate/electronica/dioda-electrotehnica-fizica-220222.html>
<https://hobbytronica.ro/dioda/>
<http://www.ac.tuiasi.ro/~lmastacan/wp-content/uploads/L3-Relee-si-contactoare.pdf>
<http://microcontrollerslab.com/analog-to-digital-adc-converter-working/>
<https://www.elprocus.com/analog-to-digital-adc-converter/>
<https://www.elprocus.com/types-of-avr-microcontroller-atmega32-and-atmega8/>
https://www.mouser.com/ds/2/268/atmel_atmel-8155-8-bit-microcontroller-avr-atmega3-1180575.pdf
<http://andrei.clubcisco.ro/cursuri/3pm/lab1.pdf>
<https://products.office.com/ro-ro/visio/flowchart-software>
<https://ro.wikipedia.org/wiki/Joystick>
https://en.wikipedia.org/wiki/Maximum_power_point_tracking
<https://www.sciencedirect.com/science/article/pii/S1364032116302842>

Anexă

adc_driver.c

```
/*
 * adc_driver.c
 *
 * Created: 18-Oct-17 9:51:05 AM
 * Author: ScorpionIPX
 */

#include <avr/io.h>

void ADC_init(void)
{
    DDRA = 0x00;
    // AREF = AVcc
    ADMUX = (1<<REFS0);

    // ADC Enable and prescaler of 128
    // 16000000/128 = 125000
    ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
}

// read ADC value
uint16_t ADC_get_value(uint8_t ch)
{
    // select the corresponding channel 0~7
    // ANDing with '7' will always keep the value
    // of 'ch' between 0 and 7
    ch &= 0b00000111; // AND operation with 7
    ADMUX = (ADMUX & 0xF8)|ch; // clears the bottom 3 bits before ORing

    // start single conversion
    // write '1' to ADSC
    ADCSRA |= (1<<ADSC);

    // wait for conversion to complete
    // ADSC becomes '0' again
    // till then, run loop continuously
    while(ADCSRA & (1<<ADSC));

    return (ADC);
}
```

adc_driver.h

```
/*
 * adc_driver.h
 *
 * Created: 18-Oct-17 9:58:32 AM
 * Author: uidq6025
 */

#ifndef ADC_DRIVER_H_
#define ADC_DRIVER_H_

#include <avr/io.h>

#define ADC_MAX 1023
#define ADC_HALF 512

void ADC_init(void);
uint16_t ADC_get_value(uint8_t ch);

#endif /* ADC_DRIVER_H_ */
```

charge_driver.c

```
/*
 * charge_driver.c
 *
 * Created: 24-Mar-18 14:39:59
 * Author: ScorpionIPX
 */

#include "charge_driver.h"
#include "adc_driver.h"

void init_charge_control(void)
{
    RELAY_DDR |= 1 << RELAY_PIN; // Relay pin as output
    RELAY_PORT &= ~(1 << RELAY_PIN); // Relay default state OFF
}

void turn_on_charging(void)
{
    TURN_ON_RELAY;
}

void turn_off_charging(void)
{
    TURN_OFF_RELAY;
}

unsigned int get_battery_voltage(void)
{
    unsigned int battery_voltage = ADC_get_value(BATTERY_ADC_CHANNEL);
    battery_voltage = battery_voltage*((long)BATTERY_MAX_MV) / ADC_MAX;
    battery_voltage = battery_voltage*((long) V_BAT_GAIN) / 1000;
    return battery_voltage;
}

unsigned int get_converter_voltage(void)
{
    unsigned int battery_voltage = ADC_get_value(CONVERTER_ADC_CHANNEL);
    battery_voltage = battery_voltage*((long)CONVERTER_MAX_MV)/ADC_MAX;
    battery_voltage = battery_voltage*((long) V_CHR_GAIL) / 1000;
    return battery_voltage;
}
```

charge_driver.h

```
/*
 * charge_driver.h
 *
 * Created: 24-Mar-18 14:40:12
 * Author: ScorpionIPX
 */

#include <avr/io.h>

#ifndef CHARGE_DRIVER_H_
#define CHARGE_DRIVER_H_

#define BATTERY_ADC_CHANNEL 6
#define CONVERTER_ADC_CHANNEL 7

#define BATTERY_MAX_MV 20000 // ADC channel uses a 1/4 voltage divider
#define CONVERTER_MAX_MV 20000 // ADC channel uses a 1/4 voltage divider

#define V_BAT_GAIN 975 // as unit of 1000
#define V_CHR_GAIL 975 // as unit of 1000

#define RELAY_PIN PINB4
#define RELAY_PORT PORTB
#define RELAY_DDR DDRB
#define TURN_ON_RELAY (RELAY_PORT |= 1 << RELAY_PIN)
#define TURN_OFF_RELAY (RELAY_PORT &= ~(1 << RELAY_PIN))

unsigned int get_battery_voltage(void);
unsigned int get_converter_voltage(void);
void init_charge_control(void);
void turn_on_charging(void);
void turn_off_charging(void);

#endif /* CHARGE_DRIVER_H_ */
```

global.h

```
/*
 * global.h
 *
 * Created: 06-Oct-17 11:42:12 AM
 * Author: ScorpionIPX
 */

#ifndef GLOBAL_H_
#define GLOBAL_H_

#define F_CPU 8000000UL

unsigned char STATE;
unsigned char OLD_STATE;

unsigned char UNIPOLEAR_01_CURRENT_STEP;

#define STATE_INIT 0
#define STATE_IDLE 1
#define STATE_MANUAL 2
#define STATE_TRACKING 3
#define STATE_MONITORING 4

#endif /* GLOBAL_H_ */
```

graphics.c

```
/*
 * graphics.c
 *
 * Created: 28-Oct-17 6:25:15 PM
 * Author: ScorpionIPX
 */

#include "hx1230.h"
#include "joystick_driver.h"
#include "sg90_driver.h"
#include <stdlib.h>

void display_title(void)
{
    hx_set_coordinates(0, 0);
    hx_write_string("AutoTracking LDR");
    hx_set_coordinates(0, 1);
    hx_write_string(" ScorpionIPX");
}

void display_data_menu(void)
{
    hx_set_coordinates(0, 3);
    hx_write_string("      ||");
    hx_set_coordinates(0, 4);
    hx_write_string("=====");
    hx_set_coordinates(0, 5);
    hx_write_string("      ||");
}

void display_light_sensor_data(uint8_t sensor, int data)
{
    //hx_set_coordinates(60, 3 + sensor);
    hx_set_coordinates(24 + 36 * (sensor & 1), 3 + 2 * (sensor >> 1));

    // hx_write_char('0' + ((data / 100) % 10));
    hx_write_char('0' + ((data / 10) % 10));
    hx_write_char('0' + (data % 10));

    hx_set_coordinates(18, 7);
    hx_write_char('0' + OCR1A / 100);
    hx_write_char('0' + (OCR1A / 10) % 10);
    hx_write_char('0' + OCR1A % 10);

    hx_set_coordinates(66, 7);
    hx_write_char('0' + OCR1B / 100);
    hx_write_char('0' + (OCR1B / 10) % 10);
    hx_write_char('0' + OCR1B % 10);
}

void display_idle_state_message(void)
{
    hx_set_coordinates(0, 3);
    hx_write_string("- system is in ");
```

```

        hx_set_coordinates(0, 4);
        hx_write_string("IDLE mode");
        hx_set_coordinates(0, 6);
        hx_write_string("going to sleep");
    }

void display_manual_state_message(void)
{
    hx_set_coordinates(0, 3);
    hx_write_string("- system is in ");
    hx_set_coordinates(0, 4);
    hx_write_string("MANUAL mode");

    hx_set_coordinates(0, 6);
    hx_write_string("JX:");
    hx_set_coordinates(54, 6);
    hx_write_string("B:");

    hx_set_coordinates(0, 7);
    hx_write_string("JY:");
    hx_set_coordinates(54, 7);
    hx_write_string("A:");
}

void display_joystick_data(unsigned int x, unsigned int y)
{
    hx_set_coordinates(24, 6);

    hx_write_char('0' + x / 1000);
    hx_write_char('0' + (x / 100) % 10);
    hx_write_char('0' + (x / 10) % 10);
    hx_write_char('0' + x % 10);

    hx_set_coordinates(72, 6);

    hx_write_char('0' + SG90_ROTATE_DUTY_CYCLE_REGISTER / 1000);
    hx_write_char('0' + (SG90_ROTATE_DUTY_CYCLE_REGISTER / 100) % 10);
    hx_write_char('0' + (SG90_ROTATE_DUTY_CYCLE_REGISTER / 10) % 10);
    hx_write_char('0' + SG90_ROTATE_DUTY_CYCLE_REGISTER % 10);

    hx_set_coordinates(24, 7);

    hx_write_char('0' + y / 1000);
    hx_write_char('0' + (y / 100) % 10);
    hx_write_char('0' + (y / 10) % 10);
    hx_write_char('0' + y % 10);

    hx_set_coordinates(72, 7);

    hx_write_char('0' + SG90_INCLINE_DUTY_CYCLE_REGISTER / 1000);
    hx_write_char('0' + (SG90_INCLINE_DUTY_CYCLE_REGISTER / 100) % 10);
    hx_write_char('0' + (SG90_INCLINE_DUTY_CYCLE_REGISTER / 10) % 10);
    hx_write_char('0' + SG90_INCLINE_DUTY_CYCLE_REGISTER % 10);
}

```

```

void display_monitoring_message(void)
{
    hx_set_coordinates(0, 3);
    hx_write_string("- system is in ");
    hx_set_coordinates(0, 4);
    hx_write_string("MONITORING mode");

    hx_set_coordinates(0, 6);
    hx_write_string("Vbat:");
    hx_set_coordinates(80, 6);
    hx_write_string("V");

    hx_set_coordinates(0, 7);
    hx_write_string("Vchr:");
    hx_set_coordinates(80, 7);
    hx_write_string("V");
}

void display_monitoring_data(unsigned int v_bat, unsigned int v_chr)
{
    hx_set_coordinates(42, 6);

    hx_write_char('0' + v_bat / 10000);
    hx_write_char('0' + (v_bat / 1000) % 10);
    hx_write_char('.');
    hx_write_char('0' + (v_bat / 100) % 10);
    hx_write_char('0' + (v_bat / 10) % 10);
    hx_write_char('0' + v_bat % 10);

    hx_set_coordinates(42, 7);

    hx_write_char('0' + v_chr / 10000);
    hx_write_char('0' + (v_chr / 1000) % 10);
    hx_write_char('.');
    hx_write_char('0' + (v_chr / 100) % 10);
    hx_write_char('0' + (v_chr / 10) % 10);
    hx_write_char('0' + v_chr % 10);
}

```

graphics.h

```
/*
 * graphics.h
 *
 * Created: 28-Oct-17 6:25:27 PM
 * Author: ScorpionIPX
 */

#ifndef GRAPHICS_H_
#define GRAPHICS_H_

void display_title(void);
void display_data_menu(void);
void display_light_sensor_data(unsigned char sensor, int data);
void display_idle_state_message(void);
void display_manual_state_message(void);
void display_joystick_data(unsigned int x, unsigned int y);
void display_monitoring_message(void);
void display_monitoring_data(unsigned int v_bat, unsigned int v_chr);
#endif /* GRAPHICS_H_ */
```

hx_8x6characters.h

```
/*
 * hx_8x6characters.h
 *
 * Created: 06-Oct-17 2:02:58 PM
 * Author: ScorpionIPX
 */

#ifndef HX_8X6CHARACTERS_H_
#define HX_8X6CHARACTERS_H_

static const unsigned char HX_character[][6] = {
    {0x00,0x00,0x00,0x00,0x00,0x00}, // 0x 0 0
    {0x00,0x64,0x18,0x04,0x64,0x18}, // - 0x 1 1
    {0x00,0x3c,0x40,0x40,0x20,0x7c}, // - 0x 2 2
    {0x00,0x0c,0x30,0x40,0x30,0x0c}, // - 0x 3 3
    {0x00,0x3c,0x40,0x30,0x40,0x3c}, // - 0x 4 4
    {0x00,0x00,0x3e,0x1c,0x08,0x00}, // - 0x 5 5
    {0x00,0x04,0x1e,0x1f,0x1e,0x04}, // - 0x 6 6
    {0x00,0x10,0x3c,0x7c,0x3c,0x10}, // - 0x 7 7
    {0x00,0x20,0x40,0x3e,0x01,0x02}, // - 0x 8 8
    {0x00,0x22,0x14,0x08,0x14,0x22}, // - 0x 9 9
    {0x00,0x00,0x38,0x28,0x38,0x00}, // - 0x a 10
    {0x00,0x00,0x10,0x38,0x10,0x00}, // - 0x b 11
    {0x00,0x00,0x00,0x10,0x00,0x00}, // - 0x c 12
    {0x00,0x08,0x78,0x08,0x00,0x00}, // - 0x d 13
    {0x00,0x00,0x15,0x15,0x0a,0x00}, // - 0x e 14
    {0x00,0x7f,0x7f,0x09,0x09,0x01}, // - 0x f 15
    {0x00,0x10,0x20,0x7f,0x01,0x01}, // - 0x10 16
    {0x00,0x04,0x04,0x00,0x01,0x1f}, // - 0x11 17
    {0x00,0x00,0x19,0x15,0x12,0x00}, // - 0x12 18
    {0x00,0x40,0x60,0x50,0x48,0x44}, // - 0x13 19
    {0x00,0x06,0x09,0x09,0x06,0x00}, // - 0x14 20
    {0x00,0x0f,0x02,0x01,0x01,0x00}, // - 0x15 21
    {0x00,0x00,0x01,0x1f,0x01,0x00}, // - 0x16 22
    {0x00,0x44,0x44,0x4a,0x4a,0x51}, // - 0x17 23
    {0x00,0x14,0x74,0x1c,0x17,0x14}, // - 0x18 24
    {0x00,0x51,0x4a,0x4a,0x44,0x44}, // - 0x19 25
    {0x00,0x00,0x00,0x04,0x04,0x04}, // - 0x1a 26
    {0x00,0x00,0x7c,0x54,0x54,0x44}, // - 0x1b 27
    {0x00,0x08,0x08,0x2a,0x1c,0x08}, // - 0x1c 28
    {0x00,0x7c,0x00,0x7c,0x44,0x7c}, // - 0x1d 29
    {0x00,0x04,0x02,0x7f,0x02,0x04}, // - 0x1e 30
    {0x00,0x10,0x20,0x7f,0x20,0x10}, // - 0x1f 31
    {0x00,0x00,0x00,0x00,0x00,0x00}, // - 0x20 32
    {0x00,0x00,0x00,0x6f,0x00,0x00}, // ! 0x21 33
    {0x00,0x00,0x07,0x00,0x07,0x00}, // " 0x22 34
    {0x00,0x14,0x7f,0x14,0x7f,0x14}, // # 0x23 35
    {0x00,0x00,0x07,0x04,0x1e,0x00}, // $ 0x24 36
    {0x00,0x23,0x13,0x08,0x64,0x62}, // % 0x25 37
    {0x00,0x36,0x49,0x56,0x20,0x50}, // & 0x26 38
    {0x00,0x00,0x00,0x07,0x00,0x00}, // ' 0x27 39
    {0x00,0x00,0x1c,0x22,0x41,0x00}, // ( 0x28 40
    {0x00,0x00,0x41,0x22,0x1c,0x00}, // ) 0x29 41
    {0x00,0x14,0x08,0x3e,0x08,0x14}, // * 0x2a 42
```

```

{0x00,0x08,0x08,0x3e,0x08,0x08}, // + 0x2b 43
{0x00,0x00,0x50,0x30,0x00,0x00}, // , 0x2c 44
{0x00,0x08,0x08,0x08,0x08,0x08}, // - 0x2d 45
{0x00,0x00,0x60,0x60,0x00,0x00}, // . 0x2e 46
{0x00,0x20,0x10,0x08,0x04,0x02}, // / 0x2f 47
{0x00,0x3e,0x51,0x49,0x45,0x3e}, // 0 0x30 48
{0x00,0x00,0x42,0x7f,0x40,0x00}, // 1 0x31 49
{0x00,0x42,0x61,0x51,0x49,0x46}, // 2 0x32 50
{0x00,0x21,0x41,0x45,0x4b,0x31}, // 3 0x33 51
{0x00,0x18,0x14,0x12,0x7f,0x10}, // 4 0x34 52
{0x00,0x27,0x45,0x45,0x45,0x39}, // 5 0x35 53
{0x00,0x3c,0x4a,0x49,0x49,0x30}, // 6 0x36 54
{0x00,0x01,0x71,0x09,0x05,0x03}, // 7 0x37 55
{0x00,0x36,0x49,0x49,0x49,0x36}, // 8 0x38 56
{0x00,0x06,0x49,0x49,0x29,0x1e}, // 9 0x39 57
{0x00,0x00,0x36,0x36,0x00,0x00}, // : 0x3a 58
{0x00,0x00,0x56,0x36,0x00,0x00}, // ; 0x3b 59
{0x00,0x08,0x14,0x22,0x41,0x00}, // < 0x3c 60
{0x00,0x14,0x14,0x14,0x14,0x14}, // = 0x3d 61
{0x00,0x00,0x41,0x22,0x14,0x08}, // > 0x3e 62
{0x00,0x02,0x01,0x51,0x09,0x06}, // ? 0x3f 63
{0x00,0x3e,0x41,0x5d,0x49,0x4e}, // @ 0x40 64
{0x00,0x7e,0x09,0x09,0x09,0x7e}, // A 0x41 65
{0x00,0x7f,0x49,0x49,0x49,0x36}, // B 0x42 66
{0x00,0x3e,0x41,0x41,0x41,0x22}, // C 0x43 67
{0x00,0x7f,0x41,0x41,0x41,0x3e}, // D 0x44 68
{0x00,0x7f,0x49,0x49,0x49,0x41}, // E 0x45 69
{0x00,0x7f,0x09,0x09,0x09,0x01}, // F 0x46 70
{0x00,0x3e,0x41,0x49,0x49,0x7a}, // G 0x47 71
{0x00,0x7f,0x08,0x08,0x08,0x7f}, // H 0x48 72
{0x00,0x00,0x41,0x7f,0x41,0x00}, // I 0x49 73
{0x00,0x20,0x40,0x41,0x3f,0x01}, // J 0x4a 74
{0x00,0x7f,0x08,0x14,0x22,0x41}, // K 0x4b 75
{0x00,0x7f,0x40,0x40,0x40,0x40}, // L 0x4c 76
{0x00,0x7f,0x02,0x0c,0x02,0x7f}, // M 0x4d 77
{0x00,0x7f,0x04,0x08,0x10,0x7f}, // N 0x4e 78
{0x00,0x3e,0x41,0x41,0x41,0x3e}, // O 0x4f 79
{0x00,0x7f,0x09,0x09,0x09,0x06}, // P 0x50 80
{0x00,0x3e,0x41,0x51,0x21,0x5e}, // Q 0x51 81
{0x00,0x7f,0x09,0x19,0x29,0x46}, // R 0x52 82
{0x00,0x46,0x49,0x49,0x49,0x31}, // S 0x53 83
{0x00,0x01,0x01,0x7f,0x01,0x01}, // T 0x54 84
{0x00,0x3f,0x40,0x40,0x40,0x3f}, // U 0x55 85
{0x00,0x0f,0x30,0x40,0x30,0x0f}, // V 0x56 86
{0x00,0x3f,0x40,0x30,0x40,0x3f}, // W 0x57 87
{0x00,0x63,0x14,0x08,0x14,0x63}, // X 0x58 88
{0x00,0x07,0x08,0x70,0x08,0x07}, // Y 0x59 89
{0x00,0x61,0x51,0x49,0x45,0x43}, // Z 0x5a 90
{0x00,0x3c,0x4a,0x49,0x29,0x1e}, // [ 0x5b 91
{0x00,0x02,0x04,0x08,0x10,0x20}, // \ 0x5c 92
{0x00,0x00,0x41,0x7f,0x00,0x00}, // ] 0x5d 93
{0x00,0x04,0x02,0x01,0x02,0x04}, // ^ 0x5e 94
{0x00,0x40,0x40,0x40,0x40,0x40}, // _ 0x5f 95
{0x00,0x00,0x00,0x03,0x04,0x00}, // ` 0x60 96
{0x00,0x20,0x54,0x54,0x54,0x78}, // a 0x61 97
{0x00,0x7f,0x48,0x44,0x44,0x38}, // b 0x62 98
{0x00,0x38,0x44,0x44,0x44,0x20}, // c 0x63 99

```

```

{0x00,0x38,0x44,0x44,0x48,0x7f}, // d 0x64 100
{0x00,0x38,0x54,0x54,0x54,0x18}, // e 0x65 101
{0x00,0x08,0x7e,0x09,0x01,0x02}, // f 0x66 102
{0x00,0x0c,0x52,0x52,0x52,0x3e}, // g 0x67 103
{0x00,0x7f,0x08,0x04,0x04,0x78}, // h 0x68 104
{0x00,0x00,0x44,0x7d,0x40,0x00}, // i 0x69 105
{0x00,0x20,0x40,0x44,0x3d,0x00}, // j 0x6a 106
{0x00,0x00,0x7f,0x10,0x28,0x44}, // k 0x6b 107
{0x00,0x00,0x41,0x7f,0x40,0x00}, // l 0x6c 108
{0x00,0x7c,0x04,0x18,0x04,0x78}, // m 0x6d 109
{0x00,0x7c,0x08,0x04,0x04,0x78}, // n 0x6e 110
{0x00,0x38,0x44,0x44,0x44,0x38}, // o 0x6f 111
{0x00,0x7c,0x14,0x14,0x14,0x08}, // p 0x70 112
{0x00,0x08,0x14,0x14,0x18,0x7c}, // q 0x71 113
{0x00,0x7c,0x08,0x04,0x04,0x08}, // r 0x72 114
{0x00,0x48,0x54,0x54,0x54,0x20}, // s 0x73 115
{0x00,0x04,0x3f,0x44,0x40,0x20}, // t 0x74 116
{0x00,0x3c,0x40,0x40,0x20,0x7c}, // u 0x75 117
{0x00,0x1c,0x20,0x40,0x20,0x1c}, // v 0x76 118
{0x00,0x3c,0x40,0x30,0x40,0x3c}, // w 0x77 119
{0x00,0x44,0x28,0x10,0x28,0x44}, // x 0x78 120
{0x00,0x0c,0x50,0x50,0x50,0x3c}, // y 0x79 121
{0x00,0x44,0x64,0x54,0x4c,0x44}, // z 0x7a 122
{0x00,0x00,0x08,0x36,0x41,0x41}, // { 0x7b 123
{0x00,0x00,0x00,0x7f,0x00,0x00}, // | 0x7c 124
{0x00,0x41,0x41,0x36,0x08,0x00}, // } 0x7d 125
{0x00,0x04,0x02,0x04,0x08,0x04}, // ~ 0x7e 126
{0x00,0x7f,0x6b,0x6b,0x6b,0x7f}, // 0x7f 127
{0x00,0x00,0x7c,0x44,0x7c,0x00}, // - 0x80 128
{0x00,0x00,0x08,0x7c,0x00,0x00}, // • 0x81 129
{0x00,0x00,0x64,0x54,0x48,0x00}, // , 0x82 130
{0x00,0x00,0x44,0x54,0x28,0x00}, // f 0x83 131
{0x00,0x00,0x1c,0x10,0x78,0x00}, // „ 0x84 132
{0x00,0x00,0x5c,0x54,0x24,0x00}, // ... 0x85 133
{0x00,0x00,0x78,0x54,0x74,0x00}, // † 0x86 134
{0x00,0x00,0x64,0x14,0x0c,0x00}, // ‡ 0x87 135
{0x00,0x00,0x7c,0x54,0x7c,0x00}, // ^ 0x88 136
{0x00,0x00,0x5c,0x54,0x3c,0x00}, // % 0x89 137
{0x00,0x78,0x24,0x26,0x25,0x78}, // Š 0x8a 138
{0x00,0x78,0x25,0x26,0x24,0x78}, // < 0x8b 139
{0x00,0x70,0x2a,0x29,0x2a,0x70}, // € 0x8c 140
{0x00,0x78,0x25,0x24,0x25,0x78}, // • 0x8d 141
{0x00,0x20,0x54,0x56,0x55,0x78}, // } 0x8e 142
{0x00,0x20,0x55,0x56,0x54,0x78}, // • 0x8f 143
{0x00,0x20,0x56,0x55,0x56,0x78}, // • 0x90 144
{0x00,0x20,0x55,0x54,0x55,0x78}, // ‘ 0x91 145
{0x00,0x7c,0x54,0x56,0x55,0x44}, // ’ 0x92 146
{0x00,0x7c,0x55,0x56,0x54,0x44}, // “ 0x93 147
{0x00,0x7c,0x56,0x55,0x56,0x44}, // ” 0x94 148
{0x00,0x7c,0x55,0x54,0x55,0x44}, // • 0x95 149
{0x00,0x38,0x54,0x56,0x55,0x18}, // - 0x96 150
{0x00,0x38,0x55,0x56,0x54,0x18}, // - 0x97 151
{0x00,0x38,0x56,0x55,0x56,0x18}, // ~ 0x98 152
{0x00,0x38,0x55,0x54,0x55,0x18}, // ™ 0x99 153
{0x00,0x00,0x44,0x7e,0x45,0x00}, // š 0x9a 154
{0x00,0x00,0x45,0x7e,0x44,0x00}, // > 0x9b 155
{0x00,0x00,0x46,0x7d,0x46,0x00}, // œ 0x9c 156

```

```

{0x00,0x00,0x45,0x7c,0x45,0x00}, // • 0x9d 157
{0x00,0x00,0x48,0x7a,0x41,0x00}, // ~ 0x9e 158
{0x00,0x00,0x49,0x7a,0x40,0x00}, // Ÿ 0x9f 159
{0x00,0x00,0x4a,0x79,0x42,0x00}, // 0xa0 160
{0x00,0x00,0x49,0x78,0x41,0x00}, // i 0xa1 161
{0x00,0x38,0x44,0x46,0x45,0x38}, // ¢ 0xa2 162
{0x00,0x38,0x45,0x46,0x44,0x38}, // £ 0xa3 163
{0x00,0x38,0x46,0x45,0x46,0x38}, // ₣ 0xa4 164
{0x00,0x38,0x45,0x44,0x45,0x38}, // ¥ 0xa5 165
{0x00,0x30,0x48,0x4a,0x49,0x30}, // ! 0xa6 166
{0x00,0x30,0x49,0x4a,0x48,0x30}, // § 0xa7 167
{0x00,0x30,0x4a,0x49,0x4a,0x30}, // " 0xa8 168
{0x00,0x30,0x49,0x48,0x49,0x30}, // ® 0xa9 169
{0x00,0x3c,0x40,0x42,0x41,0x3c}, // ª 0xaa 170
{0x00,0x3c,0x41,0x42,0x40,0x3c}, // « 0xab 171
{0x00,0x3c,0x42,0x41,0x42,0x3c}, // ¬ 0xac 172
{0x00,0x3c,0x41,0x40,0x41,0x3c}, // - 0xad 173
{0x00,0x3c,0x40,0x42,0x21,0x7c}, // ® 0xae 174
{0x00,0x3c,0x41,0x42,0x20,0x7c}, // - 0xaf 175
{0x00,0x38,0x42,0x41,0x22,0x78}, // ° 0xb0 176
{0x00,0x3c,0x41,0x40,0x21,0x7c}, // ± 0xb1 177
{0x00,0x4e,0x51,0x71,0x11,0x0a}, // ² 0xb2 178
{0x00,0x58,0x64,0x64,0x24,0x10}, // ³ 0xb3 179
{0x00,0x7c,0x0a,0x11,0x22,0x7d}, // ' 0xb4 180
{0x00,0x78,0x12,0x09,0x0a,0x71}, // µ 0xb5 181
{0x00,0x00,0x00,0x04,0x02,0x01}, // ¶ 0xb6 182
{0x00,0x01,0x02,0x04,0x00,0x00}, // • 0xb7 183
{0x00,0x00,0x02,0x00,0x02,0x00}, // . 0xb8 184
{0x00,0x30,0x48,0x45,0x40,0x20}, // ¹ 0xb9 185
{0x00,0x00,0x00,0x7b,0x00,0x00}, // º 0xba 186
{0x00,0x38,0x44,0x44,0x38,0x44}, // » 0xbb 187
{0x00,0x40,0x3e,0x49,0x49,0x36}, // % 0xbc 188
{0x00,0x08,0x04,0x08,0x70,0x0c}, // % 0xbd 189
{0x00,0x60,0x50,0x48,0x50,0x60}, // % 0xbe 190
{0x00,0x20,0x52,0x55,0x59,0x30}, // ð 0xbf 191
{0x00,0x38,0x54,0x54,0x54,0x00}, // À 0xc0 192
{0x00,0x00,0x00,0x7f,0x41,0x00}, // Á 0xc1 193
{0x00,0x40,0x22,0x14,0x18,0x60}, // Â 0xc2 194
{0x00,0x7c,0x20,0x20,0x1c,0x20}, // Â 0xc3 195
{0x00,0x44,0x3c,0x04,0x7c,0x44}, // Å 0xc4 196
{0x00,0x40,0x3c,0x12,0x12,0x0c}, // Å 0xc5 197
{0x00,0x41,0x63,0x55,0x49,0x41}, // Å 0xc6 198
{0x00,0x38,0x44,0x44,0x3c,0x04}, // Ç 0xc7 199
{0x00,0x08,0x04,0x3c,0x44,0x24}, // È 0xc8 200
{0x00,0x08,0x14,0x7f,0x14,0x08}, // É 0xc9 201
{0x00,0x4e,0x71,0x01,0x71,0x4e}, // Ê 0xca 202
{0x00,0x45,0x29,0x11,0x29,0x45}, // Ë 0xcb 203
{0x00,0xd,0x51,0x51,0x51,0x3d}, // Í 0xcc 204
{0x00,0x00,0x00,0x05,0x02,0x05}, // Í 0xcd 205
{0x00,0x40,0x00,0x40,0x00,0x40}, // Î 0xce 206
{0x00,0x00,0x08,0x1c,0x3e,0x00}, // Ï 0xcf 207
{0x00,0x1c,0x1c,0x1c,0x00,0x00}, // Đ 0xd0 208
{0x00,0x00,0x70,0x08,0x07,0x00}, // Ñ 0xd1 209
{0x00,0x00,0x08,0x08,0x08,0x00}, // ò 0xd2 210
{0x00,0x00,0x1d,0x15,0x17,0x00}, // Õ 0xd3 211
{0x00,0x00,0x07,0x05,0x07,0x00}, // Ô 0xd4 212
{0x00,0x00,0x11,0x15,0x0a,0x00}, // Õ 0xd5 213

```

```

{0x00,0x00,0x00,0x00,0x00,0x00}, // Ö 0xd6 214
{0x00,0x04,0x3c,0x41,0x20,0x00}, // × 0xd7 215
{0x00,0x7c,0x16,0x15,0x16,0x08}, // Ø 0xd8 216
{0x00,0x21,0x16,0x08,0x34,0x42}, // Ù 0xd9 217
{0x00,0x7f,0x09,0x1d,0x01,0x03}, // Ú 0xda 218
{0x00,0x38,0x54,0x54,0x14,0x08}, // Ü 0xdb 219
{0x00,0x00,0x00,0x7c,0x40,0x40}, // ÿ 0xdc 220
{0x00,0x7f,0x0e,0x1c,0x38,0x7f}, // Ÿ 0xdd 221
{0x00,0x41,0x22,0x5d,0x22,0x1c}, // þ 0xde 222
{0x00,0x1c,0x3e,0x1c,0x08,0x00}, // ß 0xdf 223
{0x00,0x7f,0x7f,0x7f,0x7f,0x7f}, // à 0xe0 224
{0x00,0x77,0x7b,0x01,0x7b,0x77}, // á 0xe1 225
{0x00,0x7f,0x43,0x75,0x43,0x7f}, // â 0xe2 226
{0x00,0x7f,0x6f,0x55,0x43,0x7f}, // ã 0xe3 227
{0x00,0x40,0x40,0x40,0x40,0x40}, // ä 0xe4 228
{0x00,0x44,0x42,0x5f,0x42,0x44}, // å 0xe5 229
{0x00,0x40,0x5e,0x45,0x5e,0x40}, // æ 0xe6 230
{0x00,0x40,0x48,0x55,0x5e,0x40}, // ç 0xe7 231
{0x00,0x00,0x04,0x08,0x10,0x20}, // è 0xe8 232
{0x00,0x03,0x07,0x0e,0x1c,0x38}, // é 0xe9 233
{0x00,0x01,0x03,0x07,0x0f,0x1f}, // ê 0xea 234
{0x00,0x7c,0x78,0x70,0x60,0x40}, // ë 0xeb 235
{0x00,0x08,0x08,0x1c,0x22,0x1c}, // ï 0xec 236
{0x00,0x00,0x1c,0x22,0x1c,0x00}, // í 0xed 237
{0x00,0x02,0x00,0x08,0x00,0x20}, // î 0xee 238
{0x00,0x04,0x3e,0x3f,0x3e,0x04}, // ï 0xef 239
{0x00,0x10,0x3e,0x7e,0x3e,0x10}, // ð 0xf0 240
{0x00,0x55,0x2a,0x55,0x2a,0x55}, // ñ 0xf1 241
{0x00,0x24,0x2a,0x7f,0x2a,0x12}, // ò 0xf2 242
{0x00,0x04,0x1e,0x1f,0x1e,0x04}, // ó 0xf3 243
{0x00,0x00,0x00,0x00,0x00,0x00}, // ô 0xf4 244
{0x00,0x00,0x00,0x00,0x00,0x00}, // õ 0xf5 245
{0x00,0x00,0x00,0x00,0x00,0x00}, // ö 0xf6 246
{0x00,0x00,0x00,0x00,0x00,0x00}, // ÷ 0xf7 247
{0x00,0x00,0x00,0x00,0x00,0x00}, // ø 0xf8 248
{0x00,0x00,0x00,0x00,0x00,0x00}, // ù 0xf9 249
{0x00,0x00,0x00,0x00,0x00,0x00}, // ú 0xfa 250
{0x00,0x00,0x00,0x00,0x00,0x00}, // û 0xfb 251
{0x00,0x00,0x00,0x00,0x00,0x00}, // ü 0xfc 252
{0x00,0x00,0x00,0x00,0x00,0x00}, // ý 0xfd 253
{0x00,0x00,0x00,0x00,0x00,0x00}, // þ 0xfe 254
{0x00,0x00,0x00,0x00,0x00,0x00} // ÿ 0xff 255
};

#endif /* HX_8X6CHARACTERS_H_ */

```

hx1230.c

```
/*
 * hx1230.c
 *
 * Created: 06-Oct-17 12:35:49 AM
 * Author: ScorpionIPX
 */

#include "global.h"
#include "hx1230.h"
#include <util/delay.h>
#include "hx_8x6_characters.h"
#include <string.h>

void init_hx1230_control(void)
{
    // set required pins as output
    HX1230_DDR |= ((1 << HX_RST) | (1 << HX_CE) | (1 << HX_DIN) | (1 << HX_CLK));

    // set idle state
    CLEAR_HX_CLK;
    #ifdef HX_DELAY_ENABLED
    _delay_us(HX_DELAY_US);
    #endif
    CLEAR_HX_RST;
    #ifdef HX_DELAY_ENABLED
    _delay_us(HX_DELAY_US);
    #endif
    _delay_ms(50);
    SET_HX_RST;
    #ifdef HX_DELAY_ENABLED
    _delay_us(HX_DELAY_US);
    #endif
    CLEAR_HX_CE;
    #ifdef HX_DELAY_ENABLED
    _delay_us(HX_DELAY_US);
    #endif
    _delay_ms(1);
    SET_HX_CE;
    #ifdef HX_DELAY_ENABLED
    _delay_us(HX_DELAY_US);
    #endif
    _delay_ms(1);

    // commands needed to initialize hx1230 display
    // found within a chinese data sheet

    hx_send_command(0x2f);
    hx_send_command(0x90);
    hx_send_command(0xa6);
    hx_send_command(0xa4);
    hx_send_command(0xaf);

    hx_send_command(0x40);
```

```

    hx_send_command(0xb0);
    hx_send_command(0x10);
    hx_send_command(0x00);
}

void hx_send_data(unsigned char _data)
{
    // activate hx1230
    CLEAR_HX_CE;
    #ifdef HX_DELAY_ENABLED
    _delay_us(HX_DELAY_US);
    #endif

    // configure communication for data transfer
    SET_HX_DIN;
    #ifdef HX_DELAY_ENABLED
    _delay_us(HX_DELAY_US);
    #endif

    // toggle clock
    SET_HX_CLK;
    #ifdef HX_DELAY_ENABLED
    _delay_us(HX_DELAY_US);
    #endif
    CLEAR_HX_CLK;
    #ifdef HX_DELAY_ENABLED
    _delay_us(HX_DELAY_US);
    #endif

    // send the actual data, MSB fiHX_RST
    for(int bit_position = 7; bit_position >= 0; bit_position--)
    {
        // calculate bit to be send
        if(((_data >> bit_position) & 1) == 1)
        {
            SET_HX_DIN;
            #ifdef HX_DELAY_ENABLED
            _delay_us(HX_DELAY_US);
            #endif
        }
        else
        {
            CLEAR_HX_DIN;
            #ifdef HX_DELAY_ENABLED
            _delay_us(HX_DELAY_US);
            #endif
        }

        // toggle clock
        SET_HX_CLK;
        #ifdef HX_DELAY_ENABLED
        _delay_us(HX_DELAY_US);
        #endif
        CLEAR_HX_CLK;
        #ifdef HX_DELAY_ENABLED
        _delay_us(HX_DELAY_US);
        #endif
    }
}

```

```

    }

    // deactivate hx1230
    SET_HX_CE;
}

void hx_send_command(unsigned char _command)
{
    // activate hx1230
    CLEAR_HX_CE;
    #ifdef HX_DELAY_ENABLED
    _delay_us(HX_DELAY_US);
    #endif

    // configure communication for command transfer
    CLEAR_HX_DIN;

    // toggle clock
    SET_HX_CLK;
    #ifdef HX_DELAY_ENABLED
    _delay_us(HX_DELAY_US);
    #endif
    CLEAR_HX_CLK;
    #ifdef HX_DELAY_ENABLED
    _delay_us(HX_DELAY_US);
    #endif

    // send the actual command, MSB fiHX_RST
    for(int bit_position = 7; bit_position >= 0; bit_position--)
    {
        // calculate bit to be send
        if((( _command >> bit_position) & 1) == 1)
        {
            SET_HX_DIN;
            #ifdef HX_DELAY_ENABLED
            _delay_us(HX_DELAY_US);
            #endif
        }
        else
        {
            CLEAR_HX_DIN;
            #ifdef HX_DELAY_ENABLED
            _delay_us(HX_DELAY_US);
            #endif
        }
    }

    // toggle clock
    SET_HX_CLK;
    #ifdef HX_DELAY_ENABLED
    _delay_us(HX_DELAY_US);
    #endif
    CLEAR_HX_CLK;
    #ifdef HX_DELAY_ENABLED
    _delay_us(HX_DELAY_US);
    #endif
}

```

```

    // deactivate hx1230
    SET_HX_CE;
}

void hx_set_coordinates(unsigned char _x, unsigned char _y)
{
    // 0, 0 is the upper left corner

    hx_send_command(0xB0 + _y);
    hx_send_command(0x10 | ((_x & 0x7F) >> 4));
    hx_send_command(0x0F & _x);
}

void hx_clear_screen(void)
{
    unsigned char col, row;

    hx_set_coordinates(0, 0);

    for(row = 0; row <= HX_MAX_ROW_ROOT; row++)
    {
        for(col = 0; col <= HX_MAX_COL; col++)
        {
            hx_send_data(0x00);
        }
    }
}

void hx_fill_screen(void)
{
    unsigned char col, row;

    hx_set_coordinates(0, 0);

    for(row = 0; row < 9; row++)
    {
        for(col = 0; col < 96; col++)
        {
            hx_send_data(0xFF);
        }
    }
}

void hx_write_char(const unsigned char _character)
{
    for(int row_index = 0; row_index < 6; row_index++)
    {
        hx_send_data(HX_character[_character][row_index]);
    }
}

void hx_write_string(const char *_characters_array)
{
    int string_length = strlen(_characters_array);
    for(int char_index = 0; char_index < string_length; char_index++)
    {

```

```
        hx_write_char((const unsigned char)({_characters_array[char_index]}));  
    }  
}
```

hx1230.h

```
/*
 * hx1230.h
 *
 * Created: 06-Oct-17 12:36:42 AM
 * Author: ScorpionIPX
 */

#include "global.h"
#include <avr/io.h>
#include "hx_8x6_characters.h"

#ifndef HX1230_H_
#define HX1230_H_

//##define HX_DELAY_ENABLED // if uC is too fast, HX1230 won't be able to read commands
#ifndef HX_DELAY_ENABLED
#define HX_DELAY_US 1
#endif

#define HX1230_PORT PORTC //port used to control hx1230
#define HX1230_DDR DDRC //data direction register used for hx1230

#define HX_RST PC0 //external reset input
#define HX_CE PC1 //chip enable
#define HX_DIN PC6 //serial data input
#define HX_CLK PC7 //serial clock input

#define SET_HX_RST (HX1230_PORT |= (1 << HX_RST))
#define SET_HX_CE (HX1230_PORT |= (1 << HX_CE))
#define SET_HX_DIN (HX1230_PORT |= (1 << HX_DIN))
#define SET_HX_CLK (HX1230_PORT |= (1 << HX_CLK))

#define CLEAR_HX_RST (HX1230_PORT &= ~(1 << HX_RST))
#define CLEAR_HX_CE (HX1230_PORT &= ~(1 << HX_CE))
#define CLEAR_HX_DIN (HX1230_PORT &= ~(1 << HX_DIN))
#define CLEAR_HX_CLK (HX1230_PORT &= ~(1 << HX_CLK))

#define HX_MAX_ROW 64
#define HX_MAX_ROW_ROOT 8
#define HX_MAX_COL 96

void init_hx1230_control(void);
void hx_send_data(unsigned char _data);
void hx_send_command(unsigned char _command);
void hx_set_coordinates(unsigned char _x, unsigned char _y);
void hx_clear_screen(void);
void hx_fill_screen(void);
void hx_write_char(const unsigned char _character);
void hx_write_string(const char *_characters_array);

#endif /* HX1230_H_ */
```

joystick_driver.c

```
/*
 * joystick_driver.c
 *
 * Created: 3/19/2018 11:34:43 PM
 * Author: uidq6025
 */

#include "global.h"
#include <util/delay.h>
#include "user_interface.h"
#include "adc_driver.h"
#include "graphics.h"
#include "joystick_driver.h"
#include "sg90_driver.h"
#include "unipolar_driver.h"
#include "l293d.h"

void manual_control(void)
{
    unsigned int x = ADC_get_value(ADC_CHANNEL_X_AXIS);
    unsigned int y = ADC_get_value(ADC_CHANNEL_Y_AXIS);

    display_joystick_data(x, y);

    if(y > (JOYSTICK_IDLE_VALUE + JOYSTICK_DEAD_ZONE))
    {
        unipolar_01_step_backward(UNIPOLAR_01_CURRENT_STEP);
    }
    else if(y < (JOYSTICK_IDLE_VALUE - JOYSTICK_DEAD_ZONE))
    {
        unipolar_01_step_forward(UNIPOLAR_01_CURRENT_STEP);
    }
    else
    {
        unipolar_01_clear_steps();
    }

    if(x > (JOYSTICK_IDLE_VALUE + JOYSTICK_DEAD_ZONE))
    {
        l293d_hb2_rotate_right();
    }
    else if(x < (JOYSTICK_IDLE_VALUE - JOYSTICK_DEAD_ZONE))
    {
        l293d_hb2_rotate_left();
    }
    else
    {
        l293d_hb2_stop();
    }

    _delay_ms(25);
}
```

```
signed int format_axis(unsigned int axis)
{
    signed int formated_axis;
    if(axis >= ADC_HALF)
    {
        formated_axis = (axis - ADC_HALF) / JOYSTICK_RESOLUTION;
    }
    else
    {
        formated_axis = axis / JOYSTICK_RESOLUTION;
        formated_axis = formated_axis * (-1);
    }
}

return formated_axis;
}
```

joystick_driver.h

```
/*
 * joystick_driver.h
 *
 * Created: 3/19/2018 11:34:57 PM
 * Author: uidq6025
 */

#ifndef JOYSTICK_DRIVER_H_
#define JOYSTICK_DRIVER_H_

#define JOYSTICK_RESOLUTION 10
#define JOYSTICK_IDLE_VALUE 512
#define JOYSTICK_DEAD_ZONE 100

void manual_control(void);
signed int format_axis(unsigned int axis);

#endif /* JOYSTICK_DRIVER_H_ */
```

main.c

```
#include "global.h"
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include "adc_driver.h"
#include "hx1230.h"
#include "hx_8x6_characters.h"
#include "graphics.h"
#include "light.h"
#include "pwm_driver.h"
#include "sg90_driver.h"
#include "tracking.h"
#include "user_interface.h"
#include "state_handler.h"
#include "joystick_driver.h"
#include "monitoring.h"
#include "unipolar_driver.h"
#include "l293d.h"

void uC_init(void);

int main(void)
{
    STATE = STATE_INIT;
    OLD_STATE = STATE_INIT;
    uC_init();

    STATE = STATE_IDLE;

    while (1)
    {
        if(STATE_CHANGED)
        {
            OLD_STATE = STATE; // update state
            go_to_state(STATE);
            _delay_ms(250);
            sei(); // enable interrupts
        }

        switch(OLD_STATE)
        {
            case STATE_TRACKING:
            {
                track();
                break;
            }
            case STATE_MANUAL:
            {
                manual_control();
                break;
            }
            case STATE_MONITORING:
```

```

        {
            monitor();
            break;
        }
        default:
        {
            break;
        }
    }
}

void uC_init(void)
{
    // Wait for system to get fully powered up
    _delay_ms(100);

    // initialize required modules
    ADC_init();
    _delay_ms(50);

    init_user_interface();
    _delay_ms(50);

    init_unipolar_control();
    _delay_ms(50);

    init_l293d_control();
    _delay_ms(50);

    init_hx1230_control();
    _delay_ms(50);
    hx_fill_screen();
    _delay_ms(500);
    hx_clear_screen();
    _delay_ms(50);

    display_title();
    display_idle_state_message();

    sei(); // enable global interrupts
}

```

monitoring.c

```
/*
 * monitoring.c
 *
 * Created: 24-Mar-18 15:06:41
 * Author: ScorpionIPX
 */

#include "global.h"
#include <util/delay.h>
#include "charge_driver.h"
#include "graphics.h"

void monitor(void)
{
    unsigned int battery_voltage = get_battery_voltage();
    unsigned int converter_voltage = get_converter_voltage();

    display_monitoring_data(battery_voltage, converter_voltage);
    _delay_ms(100);
}
```

monitoring.h

```
/*
 * monitoring.h
 *
 * Created: 24-Mar-18 15:06:54
 * Author: ScorpionIPX
 */
```

```
#ifndef MONITORING_H_
#define MONITORING_H_

void monitor(void);

#endif /* MONITORING_H_ */
```

state_handler.c

```
/*
 * state_handler.c
 *
 * Created: 3/19/2018 8:08:05 PM
 * Author: uidq6025
 */

#include "global.h"
#include <util/delay.h>
#include "graphics.h"
#include "state_handler.h"
#include "hx1230.h"
#include "unipolar_driver.h"
#include "1293d.h"

void go_to_state(unsigned char state)
{
    1293d_hb2_stop(); /* make sure motor control is turned off when changing states */
    unipolar_01_clear_steps(); /* make sure motor control is turned off when changing states */
    hx_clear_screen();
    switch(state)
    {
        case STATE_IDLE:
        {
            STATE = STATE_IDLE; // update global state
            idle_state_setup();
            break;
        }
        case STATE_MANUAL:
        {
            STATE = STATE_MANUAL; // update global state
            manual_state_setup();
            break;
        }
        case STATE_TRACKING:
        {
            STATE = STATE_TRACKING; // update global state
            tracking_state_setup();
            break;
        }
        case STATE_MONITORING:
        {
            STATE = STATE_MONITORING; // update global state
            monitoring_state_setup();
            break;
        }
    }
}

void idle_state_setup(void)
{
```

```
_delay_ms(200);
hx_clear_screen();
display_title();
display_idle_state_message();
}

void manual_state_setup(void)
{
    _delay_ms(200);
    hx_clear_screen();
    display_title();
    display_manual_state_message();
}

void tracking_state_setup(void)
{
    _delay_ms(200);
    hx_clear_screen();
    display_title();
    display_data_menu();
    _delay_ms(500);
}

void monitoring_state_setup(void)
{
    _delay_ms(200);
    hx_clear_screen();
    display_title();
    display_monitoring_message();
}
```

state_handler.h

```
/*
 * state_handler.h
 *
 * Created: 3/19/2018 8:10:15 PM
 * Author: uidq6025
 */

#include "global.h"

#ifndef STATE_HANDLER_H_
#define STATE_HANDLER_H_

#define STATE_CHANGED (!(OLD_STATE == STATE))

void go_to_state(unsigned char state);
void idle_state_setup(void);
void manual_state_setup(void);
void tracking_state_setup(void);
void monitoring_state_setup(void);

#endif /* STATE_HANDLER_H_ */
```

tracking.h

```
/*
 * tracking.h
 *
 * Created: 29-Oct-17 5:25:08 PM
 * Author: ScorpionIPX
 */

#ifndef TRACKING_H_
#define TRACKING_H_

#define INCLINE_TRACKING_TOLERANCE 2
#define ROTATE_TRACKING_TOLERANCE 2

void track(void);

#endif /* TRACKING_H_ */
```

user_interface.c

```
/*
 * user_interface.c
 *
 * Created: 3/19/2018 8:38:39 PM
 * Author: uidq6025
 */

#include "global.h"
#include <util/delay.h>
#include <avr/interrupt.h>
#include "user_interface.h"
#include "state_handler.h"
#include "adc_driver.h"

void init_user_interface(void)
{
    init_next_state_button();
}

void init_next_state_button(void)
{
    BUTTON_1_DRR &= ~(1 << BUTTON_1_PIN); // PD2 is input

    BUTTON_1_PORT |= (1 << BUTTON_1_PIN); // turn on the pull-up resistor
    // PD2 is now an input with pull-up enabled

    MCUCR &= ~(1 << ISC00 | 1 << ISC01); // low level of INT0 generates an
    interrupt request: when BUTTON_1 is pressed
    GICR |= (1 << INT0); // turns on INT0
}

ISR (INT0_vect)
{
    cli(); // temporarily disable interrupts
    switch(STATE)
    {
        case STATE_IDLE:
        {
            STATE = STATE_MANUAL;
            break;
        }
        case STATE_MANUAL:
        {
            STATE = STATE_TRACKING;
            break;
        }
        case STATE_TRACKING:

```

```
{  
    STATE = STATE_MONITORING;  
    break;  
}  
case STATE_MONITORING:  
{  
    STATE = STATE_IDLE;  
    break;  
}  
default:  
{  
    STATE = STATE_IDLE;  
    break;  
}  
}  
}
```

user_interface.h

```
/*
 * user_interface.h
 *
 * Created: 3/19/2018 8:38:53 PM
 * Author: uidq6025
 */

#include <avr/io.h>

#ifndef USER_INTERFACE_H_
#define USER_INTERFACE_H_

#define BUTTON_1_DRR DDRD
#define BUTTON_1_PORT PORTD
#define BUTTON_1_PIN 2 // PD2

#define ADC_CHANNEL_X_AXIS 4
#define ADC_CHANNEL_Y_AXIS 5

void init_next_state_button(void);
void init_user_interface(void);

#endif /* USER_INTERFACE_H_ */
```