Universitatea “Politehnica” din Bucureşti

Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei

Tablă de şah inteligentă

Proiect de diplomă

prezentat ca cerinţă parţială pentru obţinerea titlului de

Inginer în domeniul *Electronică şi Telecomunicaţii*

programul de studii de licenţă *Microelectronică, Optoelectronică şi Nanotehnologii*

Conducător ştiinţific

Conf. Dr. Ing. Horia CUCU

Absolvent

Victor-Valentin MOCANU

Bucureşti
2020

Universitatea "Politehnica" din Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
Departamentul **TEF**

Anexa 1

TEMA PROIECTULUI DE DIPLOMĂ
a studentului **MOCANU G.D. Victor-Valentin , 441E-MON.**

1. Titlul temei: Tablă de şah inteligentă

2. Descrierea contribuţiei originale a studentului (în afara părţii de documentare) şi specificaţii de proiectare:
Scopul lucrării de diplomă este proiectarea şi realizarea unei table de şah inteligente destinată învăţării rapide şi eficiente a jocului de şah. Proiectul este structurat în două etape(hardware şi software) prezentate pe scurt în continuare.
În ceea ce priveşte partea hardware, proiectul urmăreşte implementarea următoarelor funcţionalităţi: recunoaşterea câmpurilor (realizată cu ajutorul senzorilor magnetici cu efect Hall), semnalizarea diverselor tipuri de mutări posibile prin folosirea LED-urilor RGB, conversia modului de transmitere a datelor (paralel-serial şi serial-paralel) prin utilizarea registrelor de deplasare, integrarea unui ceas de şah pentru controlul timpului de joc.
Partea software presupune dezvoltarea unei aplicaţii firmware destinată implementării regulilor jocului de şah şi diferenţierii pieselor (tipul şi culoarea acestora) pe baza poziţiei iniţiale.

3. Resurse folosite la dezvoltarea proiectului:
Limbajul de programare C (Atmel Studio), placă de dezvoltare cu microcontroler din familia AVR (seria ATmega).

4. Proiectul se bazează pe cunoştinţe dobîndite în principal la următoarele 3-4 discipline:
Microcontrolere, CID, PC, SDA

5. Proprietatea intelectuală asupra proiectului aparţine: U.P.B.

6. Data înregistrării temei: 2019-11-25 14:43:55

Conducător(i) lucrare,
Conf. dr. ing. Horia CUCU

Student,

Director departament,
Conf. dr. ing. Marian VLĂDESCU

Decan,
Prof. dr. ing. Mihnea UDREA

Cod Validare: **17e31b4f53**

Declarație de onestitate academică

Prin prezenta declar că lucrarea cu titlul "*Tablă de şah inteligentă*", prezentată în cadrul Facultății de Electronică, Telecomunicații și Tehnologia Informației a Universității "Politehnica" din București ca cerință parțială pentru obținerea titlului de *Inginer* în domeniul *Electronică și Telecomunicații*, programul de studii *Microelectronică, Optoelectronică și Nanotehnologii (MON)* este scrisă de mine și nu a mai fost prezentată niciodată la o facultate sau instituție de învățământ superior din țară sau străinătate.

Declar că toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare, ca referințe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și fac referință la sursă. Reformularea în cuvinte proprii a textelor scrise de către alți autori face referință la sursă. Înțeleg că plagiatul constituie infracțiune și se sancționează conform legilor în vigoare.

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

București, *17.06.2020*

Absolvent *Victor-Valentin MOCANU*

(semnătura în original)

Cuprins

Listă de figuri

Listă de tabele

Listă de acronime

Acronime	Descriere	Scurtă explicaţie
ABS	Acrylonitrile Butadiene Styrene	Acrilonitril Butadien Stiren
AVR	-	Familie de microcontrolere
CPHA	Clock Phase	-
CPI	Clock Per Instruction	-
CPOL	Clock Polarity	-
CPU	Central Processing Unit	-
CTC	Clear Timer on Compare	Mod de funcţionare al timer-ului 1 intern al microcontrolerului ATmega2560
DGT	Digital Game Technology	-
DIN	Data Input	Pin de intrare de date
DIP	Dual In-line Package	-
DO sau DOUT	Data Output	Pin de ieşire de date
GND	Ground	-
I2C	Inter Integrated Circuit	-
LED	Light Emitting Diode	-
LSB	Less Significant Bit	-
MISO	Master Input – Slave Output	-
MOSI	Master Output – Slave Input	-
MSB	Most Significant Bit	-
NRZ	Not Return to Zero	-
PCB	Printed Circuit Board	Placă de circuit imprimat
PWM	Pulse Width Modulation	-
RGB	Red Green Blue	-
RISC	Reduced Instruction Set Computer	-
SCLK	Serial Clock	-
SMD	Surface-Mounted Device	-
SPCR	SPI Control Register	-
SPDR	SPI Data Register	-
SPI	Serial Peripheral Interface	-
SPIF	SPI Interrupt Flag	-
SPSR	SPI Status Register	-
SS / CS	Slave Select / Chip Select	-
THT	Through Hole Technology	-
USART	Universal Synchronous and Asynchronous Receiver and Transmitter	-
USB	Universal Serial Bus	-
VCC sau VDD	-	Tensiune pozitivă de alimentare
VSS	-	Tensiune nulă sau negativă de alimentare

Introducere

Motivaţie

Şahul reprezintă unul dintre cele mai populare sporturi practicate în zilele noastre, fiind apărut încă din a doua jumătate a secolului al VI - lea în zona fostelor imperii asiatice (India, China, Persia) şi având un set de reguli bine definit, cunoscut de către jucătorii actuali, abia în secolul al XV - lea. [1]

Fiind în egală măsură un joc, dar şi un sport, şahul aduce multiple beneficii persoanelor ce îl practică, printre acestea fiind enumerate: creşterea capacităţii de concentrare către o anumită activitate, dobândirea unei gândiri analitice şi a unui caracter competitiv, sporirea reuşitei în diverse situaţii ce necesită stabilirea unei strategii eficiente şi într-un interval de timp scurt.

Evoluţia tehnologică din ultimii 30 de ani a atras o dezvoltare puternică în domeniul şahului, persoanele practicante ale acestui sport având acces la o serie de unelte menite să le uşureze procesul de învăţare în obţinerea performanţei. Câteva realizări remarcabile în acest sens sunt:

motoarele de analiză (precum Deep Blue, Fritz, Rybka), bazele de date ce înglobează milioane de partide jucate de acum aproximativ 200 de ani şi până în prezent (bazele de date ale companiei ChessBase), tablele de şah electronice ce permit vizualizarea în timp real, prin intermediul unei pagini web, a partidelor desfăşurate în cadrul unor competiţii (tablele de şah electronice ale companiei DGT). Totuşi, uneltele prezentate anterior se adresează unor jucători de nivel intermediar, nefiind uşor accesibile persoanelor care fac primii paşi în lumea şahului, pentru care conceptele de bază ale acestui joc reprezintă o provocare. De asemenea, utilizarea acestor facilităţi implică existenţa unui sistem de calcul în care să fie instalate aplicaţiile specifice, iar în cazul motoarelor de analiză, acurateţea calculului este direct influenţată de către performanţele procesorului folosit.

Toate aspectele prezentate anterior au determinat alegerea temei prezentate în lucrarea de faţă, şi anume realizarea unei table de şah inteligente menite să ghideze utilizatorii în tainele şahului prin parcurgerea conceptelor de bază într-un mod interactiv şi introducerea unei atmosfere de competiţie (primul pas fiind utilizarea unui ceas de şah pentru gestionarea timpului de joc).

Obiective urmărite

Scopul lucrării constă în realizarea unei table de şah inteligente sub forma unui sistem incorporat (“embedded”), ce cuprinde parcurgerea următoarelor etape:

- dezvoltarea unei structuri fizice sub formă de tablă de şah inteligentă, capabilă să transmită informaţii despre poziţia curentă a pieselor şi să semnalizeze luminos diversele câmpuri de pe tabla de joc;
- extinderea structurii fizice a tablei de şah pentru integrarea unui cronometru ce poate fi folosit pentru contorizarea timpului de joc;
- dezvoltarea unui sistem embedded bazat pe microcontroler care să preia informaţiile despre piesele plasate pe tabla de şah şi să detecteze diversele mutari posibile ale piesei ridicate de către utilizator, precum şi a altor evenimente specifice jocului de şah;

- realizarea unei aplicaţii firmware pentru configurarea modulelor, implementarea comunicaţiei dintre acestea şi introducerea unor algoritmi care să asigure definirea şi respectarea setului de reguli specific jocului de şah;
- verificarea funcţionalităţii corecte a sistemului atât la nivel hardware, cât şi software.

Structura proiectului

Lucrarea de faţă este împărţită în şase capitole, după cum urmează:

- Capitolul 1 prezintă domeniul de activitate în care se încadrează tema lucrării, prin observarea principiilor de funcţionare ale unor circuite electronice deja existente în categoria jocurilor de societate;
- Capitolul 2 stabileşte o serie de specificaţii ce trebuie urmate în procesul de dezvoltare a unei table de şah inteligente;
- Capitolul 3 realizează o trecere către implementarea hardware a sistemului prin analizarea blocurilor funcţionale şi a componentelor ce stau la baza fiecărui bloc. De asemenea, în acest capitol se vor prezenta şi principalele caracteristici ale modulelor utilizate;
- În capitolul 4 sunt explicate metodele care au dus la realizarea aplicaţiei firmware, punându-se accent pe funcţiile care contribuie la implementarea regulilor jocului de şah şi pe modul în care se execută programul principal pe parcursul desfăşurării unei mutări;
- Capitolul 5 cuprinde procesul de proiectare mecanică, precum şi etapele parcurse în scopul realizării fizice a tablei de şah;
- În capitolul 6 sunt prezentate datele experimentale obţinute în urma testării atât a structurii hardware, cât şi a fişierelor de cod sursă ce alcătuiesc componenta software a sistemului.

Capitolul 1 - Starea artei în domeniul jocurilor de societate inteligente

Şahul este un sport complex, alcătuit dintr-un set de reguli bine definite şi înglobează mai bine de câteva milioane de combinaţii posibile ale poziţiilor pieselor ce pot apărea pe tabla de şah. Totuşi, fiind şi un joc al minţii, şahul se încadrează în categoria jocurilor de societate ce necesită utilizarea unor informaţii a priori cunoscute (cum ar fi primele 10-20 de mutări dintr-o partidă de şah, numite de către persoanele de specialitate "deschideri") şi legarea unor noţiuni de logică pentru a stabili o strategie ce va fi urmată în scopul câştigării jocului. Pe lângă jocul de şah, în această categorie se încadrează şi jocul de GO, rezolvarea cubului Rubik, precum şi alte jocuri ce favorizează gândirea profundă a mutărilor efectuate, în detrimentul norocului.

În continuare, vor fi prezentate câteva exemple de proiecte existente în această arie, folosind sisteme incorporate pentru realizarea implementării propriu-zise. Nu sunt de neglijat nici implementările pur software, însă nu reprezintă un subiect de discuţie în cadrul lucrării de faţă.

Primul exemplu constă în realizarea unei table de şah din lemn, ce are drept scop realizarea unei interfeţe fizice între un utilizator şi unul dintre cele mai bune motoare de analiză existente la momentul de faţă (programul Stockfish). Proiectul este realizat de către utilizatorul "MaxChess", iar pentru mai multe detalii se poate accesa link-ul indicat ca sursa [15] în cadrul bibliografiei.

Figura 1.1 – Tablă de şah ce permite interacţiunea cu motorul de analiză Stockfish. Sursa: [15].

Ca principiu de funcţionare, circuitul electronic este alcătuit dintr-o placă de dezvoltare Arduino UNO care controlează o serie de LED-uri verzi THT ce indică câmpurile unde programul Stockfish doreşte să amplaseze piesele de joc, utilizatorul având sarcina de a muta piesele de joc pe câmpurile indicate de către motorul de analiză, patru butoane şi un LCD 16x4 pentru setarea nivelului de joc al programului (21 de nivele, pornind de la începător şi până la cel mai avansat), a pieselelor cu care va juca utilizatorul şi a altor opţiuni prevăzute de către această aplicaţie [15]. Pentru a integra programul Stockfish pe tabla de şah, se foloseşte placa Raspberry Pi 1 Model B+ ce reprezintă un sistem de calcul introdus pe un cip [15].

Următoarea aplicaţie este realizată de către compania GoCube şi constă într-un cub Rubik asistat de către o aplicaţie software ce poate fi instalată pe dispozitivele mobile ce rulează sistemele de operare Android sau IOS [16].

Figura 1.2 – Prezentarea cubului Rubik GoCube. Sursa: [16].

Scopul aplicaţiei este de a urmări mişcările efectuate de către utilizator (prin intermediul unor senzori ce determină prin măsurători poziţiile pătratelor de pe feţele cubului) şi trimiterea informaţiilor către aplicaţia software ce înglobează următoarele funcţionalităţi: contorizarea timpului de rezolvare a cubului Rubik, realizarea unor puzzle-uri pentru antrenarea utilizatorului şi posibilitatea de conectare la internet pentru a concura în mediul online cu alţi utilizatori ce deţin acest cub Rubik [16].

Ultimul exemplu prezentat în continuare aparţine de asemenea companiei GoCube şi constă în realizarea unui dispozitiv inteligent de dimensiuni extrem de reduse, şi anume un set de zaruri inteligente. Deşi zarul nu intra în categoria jocurilor de strategie, reprezintă un element nelipsit din majoritatea jocurilor de societate în care norocul joaca un rol esenţial în câştigarea acestora.

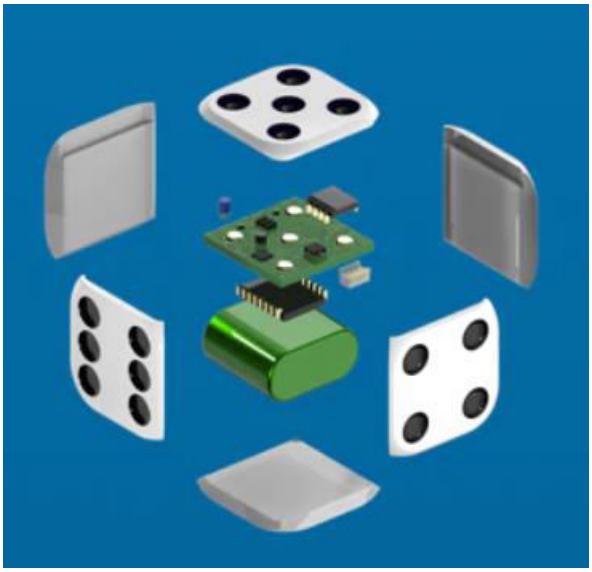


Figura 1.3 – Prezentarea zarului inteligent GoDice al companiei GoCube. Sursa: [17]

Zarul conţine accelerometre care măsoara poziţia actuală a zarului pentru a extrage informaţia referitoare la numărul vizualizat pe suprafaţa superioară a acestuia şi un modul bluetooth care permite trimiterea acestor informaţii către o aplicaţie instalată pe mobil. Valorile obţinute în urma aruncării zarului pot fi salvate în cadrul acestei aplicaţii software în timpul desfăşurării unui joc de societate cu alte persoane aflate în aceiaşi încăpere sau cu persoane din mediul online. Astfel, acesta este un bun exemplu de sistem incorporat de dimensiuni mici, a cărui funcţionare poate dura până la o oră de utilizare, datorită unui supercondensator integrat în cadrul acestui dispozitiv a cărui durată de încărcare este de aproximativ zece secunde [17].

Acestea sunt doar trei exemple de sisteme incorporate şi inteligente existente în prezent. În continuarea lucrării, se prezintă atât modul de funcţionare (hardware şi software) al tablei de şah inteligente, cât şi modul de utilizare al acesteia, observând astfel apartenenţa acestui sistem în domeniul de activitate ilustrat în cadrul acestui capitol.

Capitolul 2 - Cerinţe şi soluţii în implementarea unei table de şah inteligente

Prima etapa în dezvoltarea unui produs este reprezentată de stabilirea cerinţelor utilizatorilor vizaţi, deoarece în funcţie de aceste aspecte se determină şi eventualele specificaţii de proiectare ale sistemului. Astfel, grupul ţintă al acestui produs este reprezentat de către persoanele care sunt la început de drum şi interacţionează pentru prima dată cu regulile şi rigorile jocului de sah. Printre acestea, se pot enumera: grupele începătoare din cadrul cluburilor de şah, elevii ce studiază şahul ca materie introdusă în programa şcolară şi nu în ultimul rând, persoanele pasionate de şah, de orice vârstă.

În urma unei analize amănunţite a domeniului de activitate, s-a stabilit un set de posibile cerinţe ale grupului ţintă în ceea ce priveşte utilizarea unei table de şah inteligente, prezentate în figura 2.1:

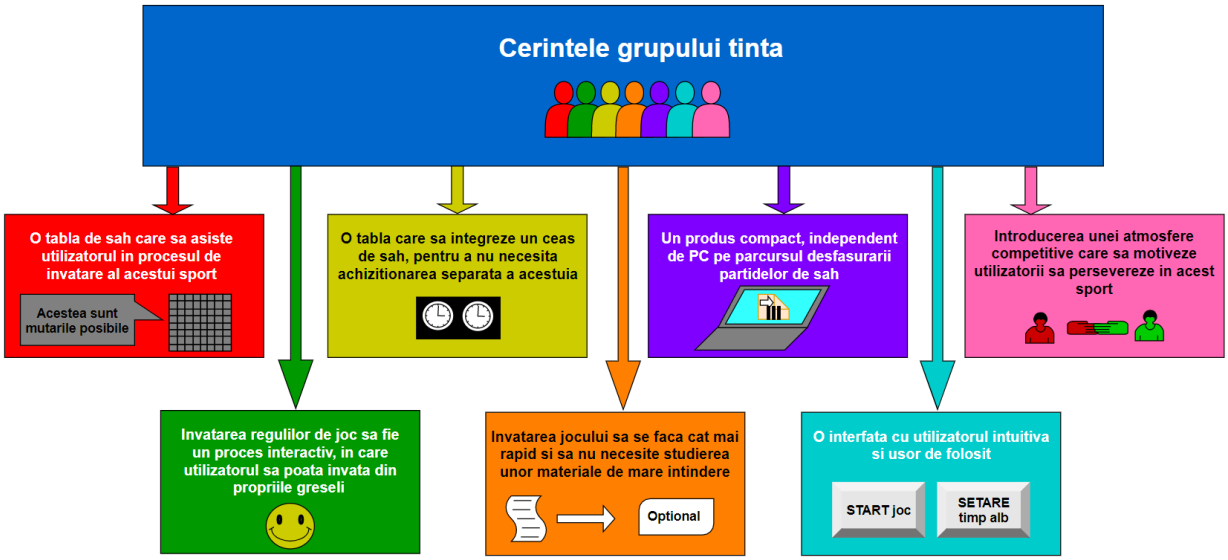


Figura 2.1 – Specificaţiile tablei de şah inteligente din perspectiva utilizatorului

Următoarea etapă constă în detalierea comportamentului produsului ce urmează a fi proiectat, pentru a se forma o imagine de ansamblu asupra elementelor ce vor face parte din componenţa tablei de şah inteligente.

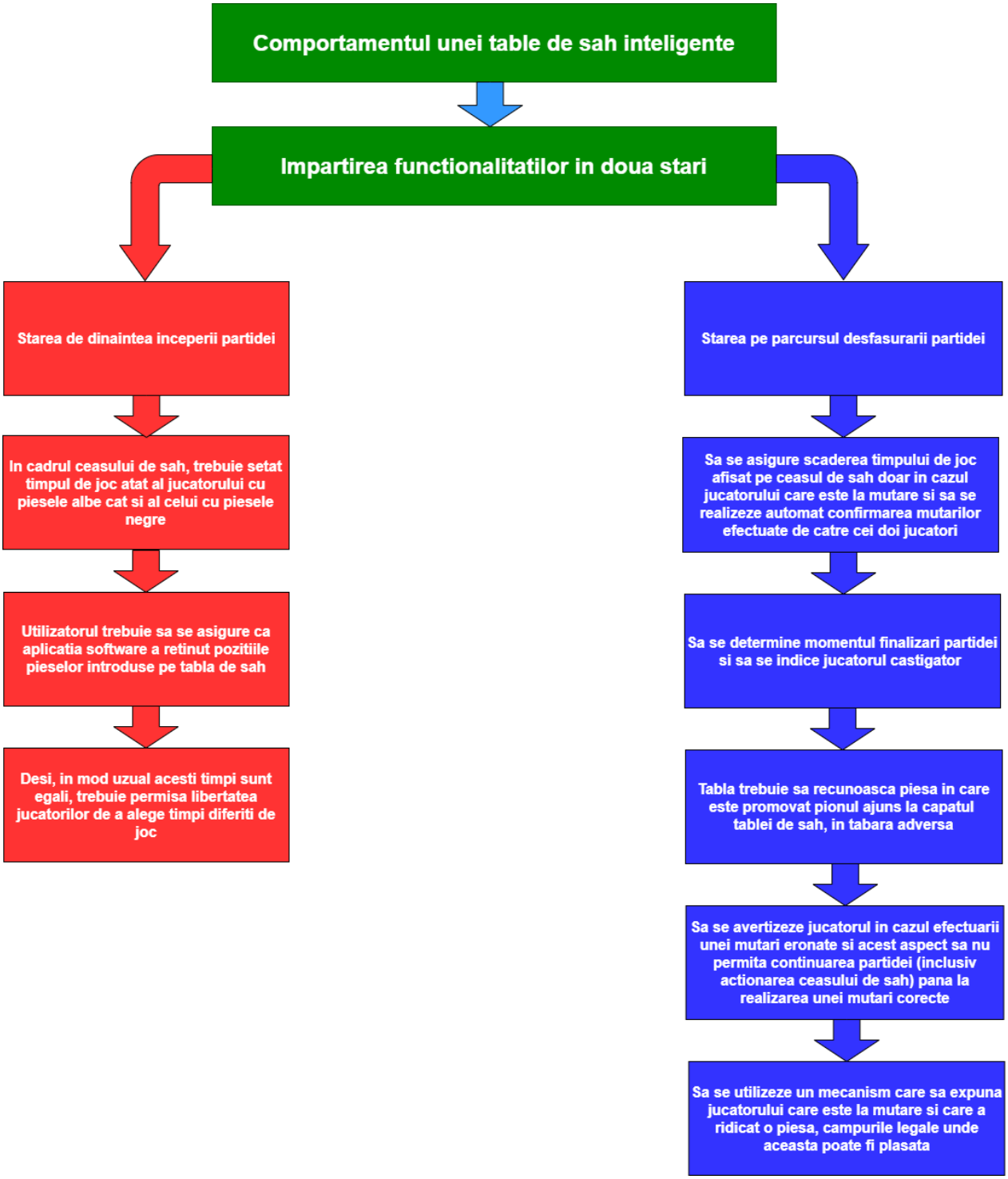


Figura 2.2 – Stabilirea comportamentului tablei de şah inteligente

În ceea ce priveşte procesul de proiectare, se rezumă toate funcţionalităţile parţiale (descrise în figura 2.2) ale tablei de şah inteligente, afirmând că funcţia îndeplinită de către acest sistem constă în asistarea utilizatorilor în timpul desfăşurării partidei prin propunerea unor mutări posibile ale pieselor

în conformitate cu regulile jocului de şah şi contorizarea timpului de joc. Deoarece, funcţionalitatea urmărită este complexă şi imposibil de atins prin folosirea unui circuit electronic simplu, este necesară împărţirea sarcinilor sistemului în mai multe funcţii simple a căror interacţiune să ducă la obţinerea rezultatului dorit (utilizarea principiului “divide et impera”). În acest capitol se delimitează aceste funcţii, iar în capitolele 3 şi 4 se prezintă modulele electronice şi componentele software care realizează implementarea propriu-zisă a acestor funcţii.

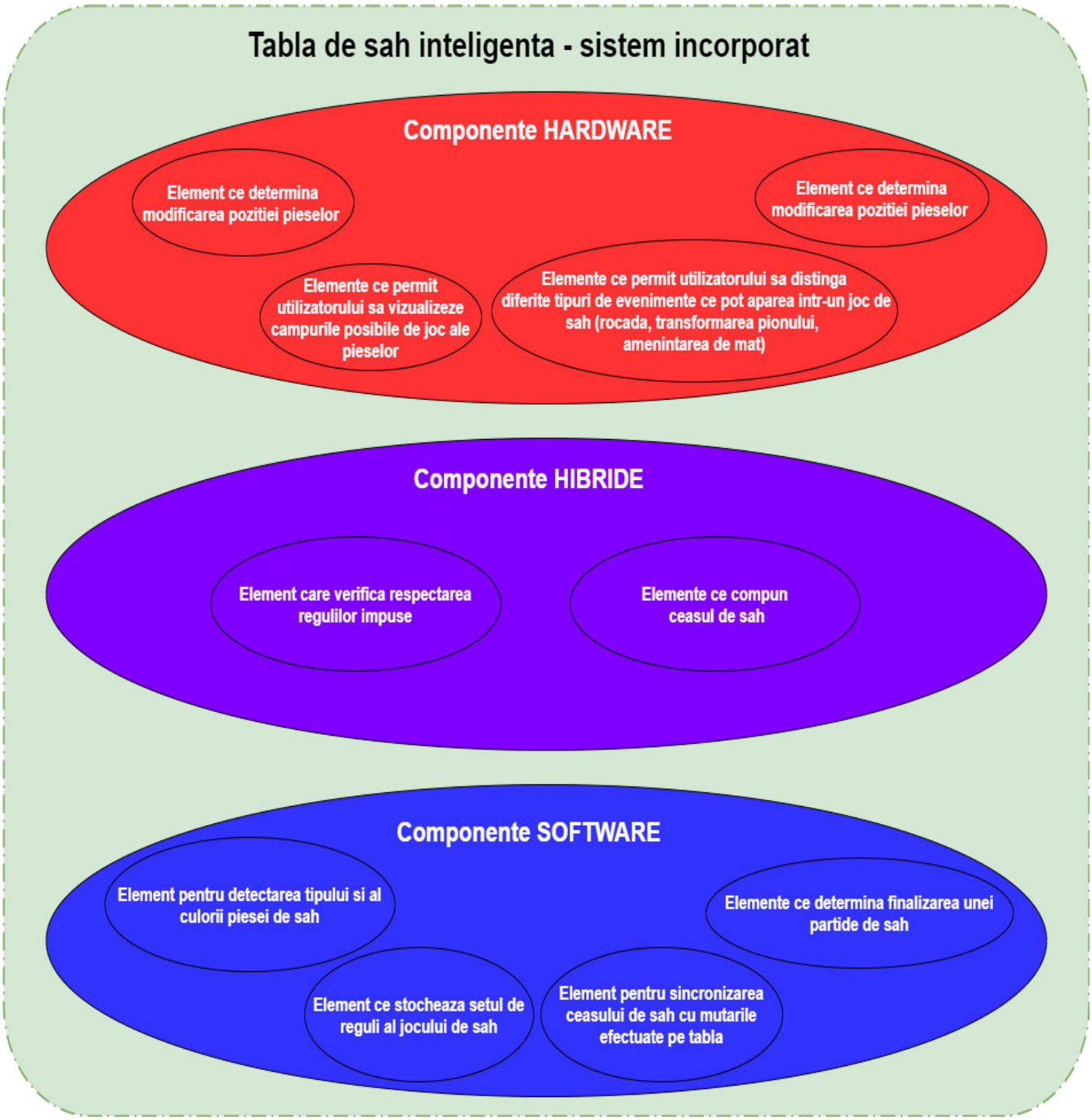


Figura 2.3 – Funcţionalitatea tablei de şah împărţită în funcţii simple

În continuare vor fi precizate soluţiile adoptate pentru a îndeplini toate cerinţele şi specificaţiile de proiectare prezentate în paragrafele anterioare.

Pentru a permite utilizatorului învăţarea noţiunilor de bază ale jocului de şah printr-o metodă interactivă şi uşor de urmărit, s-au plasat şaizeci şi patru de LED-uri RGB SMD WS2812C sub cele 64 de câmpuri de pe tabla de joc şi s-au utilizat pentru a evidenţia câmpurile conform regulilor jocului de şah. Prin schimbarea culorilor acestor LED-uri, utilizatorul poate determina ce eveniment specific acestui joc se regăseşte în poziţia actuală de pe tabla de şah şi dacă regulile impuse sunt respectate. Fiind un proces de învăţare bazat pe exerciţiu şi pe experimentarea diverselor situaţii ce pot interveni într-o partidă de şah, utilizatorii îsi pot verifica noţiunile asimilate într-un timp mult mai scurt decât cel în care utilizatorul ar fi urmat acelaşi proces în urma studierii acestor noţiuni dintr-un material ce cuprinde un număr mare de pagini.

Pentru a gestiona poziţiile actuale ale pieselor, acestea sunt urmărite de către şaizeci şi patru de senzori magnetici cu efect Hall TLE4905 ce trimit continuu informaţii referitoare la aşezarea pieselor pe tabla de şah.

Interfaţa cu utilizatorul este alcătuită din şapte butoane ce permit utilizatorului să selecteze timpii de joc ai celor doi jucători pe ceasul de şah, să determine momentul începerii partidei de şah şi al resetării acesteia, precum şi piesa ce va înlocui un pion ajuns pe câmpul de transformare.

Placa de dezvoltare cu microcontroler ATmega2560 asigură controlul întregului sistem prin intermediul aplicaţiei firmware încărcate în memoria flash a microcontrolerului. Astfel, această aplicaţie realizează majoritatea sarcinilor urmărite în procesul de proiectare al unei table de şah inteligente, cum ar fi: determinarea tipului de piesă şi al culorii acesteia, reţinerea setului de reguli ce defineşte jocul de şah şi aplicarea acestora, sincronizarea ceasului de şah cu mutările pieselor efectuate pe tabla de joc, stabilirea celor două stări ale tablei de şah inteligente menţionate anterior şi în acelaşi timp asigură posibilitatea realizării unei table de şah independente de o aplicaţie software instalată pe PC.

Sincronizarea unui volum mare de date preluate de la senzori şi aducerea acestora într-o formă compactă sunt funcţii realizate de către un grup de opt registre de deplasare SN74HC165N.

Integrarea ceasului de şah în tabla de joc este posibilă datorită modulului MAX7219 care permite vizualizarea timpului de joc al celor doi jucători pe un afişaj cu şapte segmente şi opt digiţi, funcţionalitatea ceasului fiind implementată în cadrul aplicaţiei firmware.

În acest capitol au fost rezumate cerinţele utilizatorului şi soluţiile propuse, detaliile de implementare hardware, software şi respectiv fizică fiind subiectele de discuţie ale capitolelor 3, 4 şi 5.

Capitolul 3 - Structura hardware a tablei de şah

3.1) Schema bloc a sistemului

În realizarea hardware a tablei de şah inteligente s-a folosit schema bloc prezentată în figura 3.1.

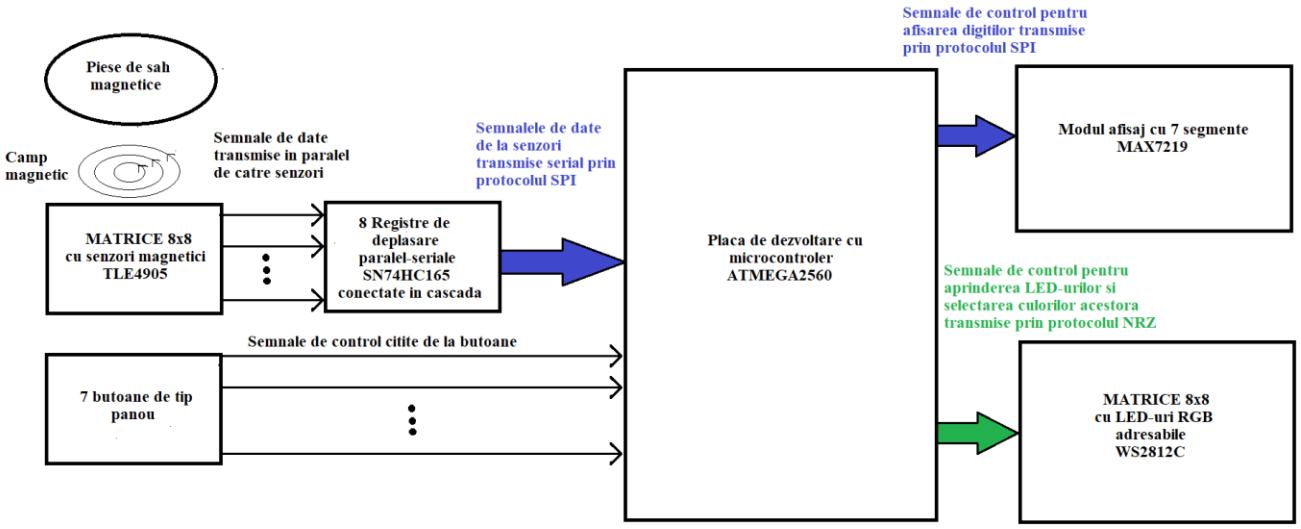


Figura 3.1 – Schema bloc a tablei de şah inteligente

În schema bloc sunt evidenţiate atât modulele ce constituie sistemul încorporat, cât şi principalele semnale ce determină interacţiunea dintre acestea. Senzorii magnetici TLE4905 împreună cu butoanele reprezintă blocul de intrare al sistemului şi asigură interfaţa acestuia cu utilizatori. Astfel, setările disponibile ale tablei de şah pot fi selectate prin intermediul butoanelor. Registrele de deplasare SN74HC165 asigură conversia paralel-serială a datelor citite de la senzori, reprezentând un bloc intermediar în cadrul sistemului. Placa de dezvoltare determină blocul de procesare şi control, deoarece pe baza datelor primite, ea trebuie să ia decizii şi să transmită semnale de control în scopul aprinderii şi selectării culorilor corespunzătoare LED-urilor RGB, precum şi afişarea segmentelor în cadrul modulului MAX7219. Blocul de ieşire este asigurat de către LED-urile adresabile WS2812C care indică utilizatorilor câmpurile legale în timpul desfăşurării unei mutări , iar afişajul cu 7 segmente furnizează timpul de joc rămas pentru fiecare jucător în parte.

De menţionat că, toate modulele sunt alimentate folosind o sursă în comutaţie cu tensiunea de ieşire de 5V, caracteristicile sursei fiind detaliate în subcapitolul 3.6.

Mai multe detalii în ceea ce priveşte principiul de funcţionare al modulelor şi modul de prelucrare al datelor se regăsesc în următoarele paragrafe din acest capitol.

3.2) Blocul de intrare

3.2.1) Scurtă prezentare a senzorului TLE4905

TLE4905 reprezintă un senzor magnetic unipolar, cu efect Hall şi ieşire digitală realizat de către compania Infineon Technologies (companie cu care am avut onoarea să colaborez în cadrul lucrării de faţă), având numeroase aplicaţii în domeniile industriale şi automotive. [3]

Astfel, TLE4905 este conceput ca fiind un circuit integrat cu trei pini, prezentaţi în următoarea figură:

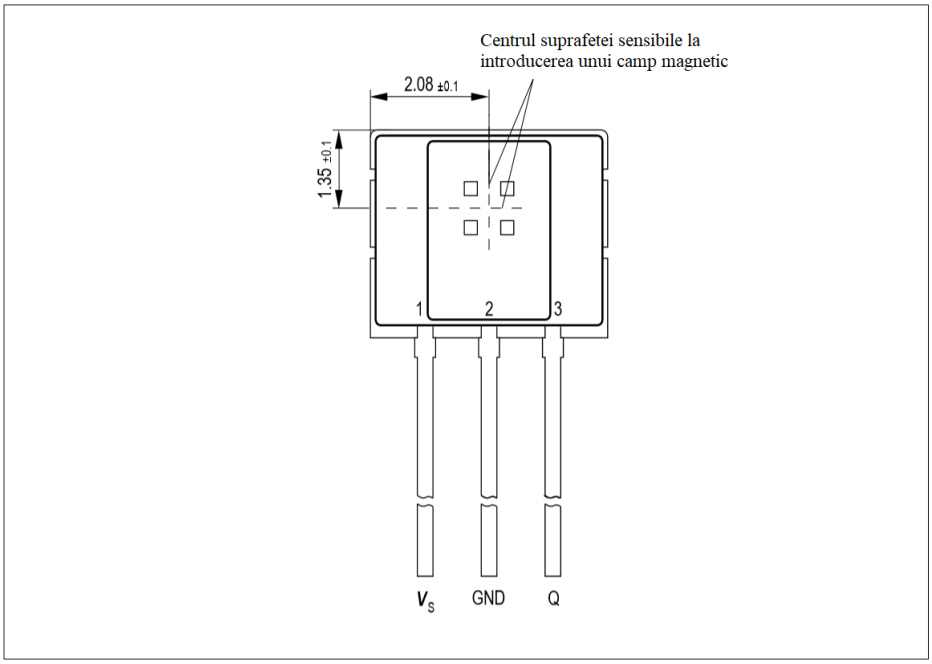


Figura 3.2 – Structura pinilor senzorului TLE4905. Sursa: [3]

- Vs reprezintă pinul de alimentare;
- GND reprezintă pinul de masă;
- Q reprezintă ieşirea digitală a senzorului.

Principalele caracteristici care au dus la alegerea senzorului TLE4905 sunt următoarele:

Parametru	Minim	Maxim
Vs (tensiunea de alimentare)	3.8 V	24 V
Is (consum de curent)	-	8 mA
Gama de temperatură	-40 ºC	150 ºC
Intervalul de valori ale inducţiei magnetice pentru care ieşirea senzorului nu comută	5 mT	18 mT

Tabel 3.1 – Principalele caracteristici ale senzorului TLE4905. Sursa: [3]

3.2.2) Principiul de funcţionare al senzorilor cu efect Hall

Efectul Hall, descoperit în anul 1879 de către fizicianul Edwin Herbert Hall, reprezintă un fenomen ce stă la baza funcţionării unor clase de senzori magnetici (precum TLE4905 prezentat anterior). Acest efect constă în apariţia unei diferenţe de potenţial (perpendiculare pe direcţia de curgere a curentului electric şi pe cea a liniilor de câmp) ca urmare a exercitării unei forţe transversale (forţă Lorentz) asupra unui semiconductor parcurs de un curent electric şi aflat sub influenţa unui câmp magnetic. Diferenţa de potenţial poartă numele de tensiune Hall (notată uzual cu V_H). [4] Acest efect este ilustrat în figura 3.3:

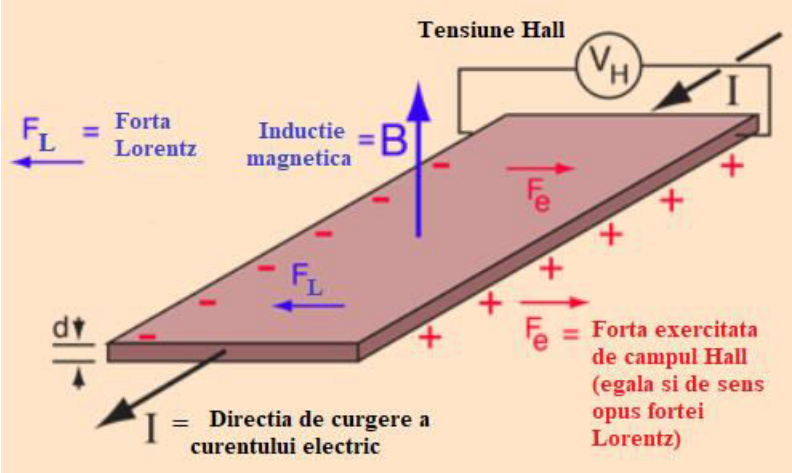


Figura 3.3 – Efectul Hall exercitat asupra unui material semiconductor. Sursa: [4]

Rezumând informaţiile prezentate anterior, un senzor cu efect Hall reprezintă un circuit integrat care converteşte o mărime magnetică (inducţia magnetică) într-o mărime electrică (tensiune electrică). În cazul senzorilor analogici, dependenţa dintre cele două mărimi este liniară în zona în care operează senzorul, în timp ce în cazul senzorilor digitali, tensiunea de ieşire poate lua doar două valori logice situate la nivele de tensiune predefinite (Exemplu : tensiunea de 5V reprezintă “1” logic, iar tensiunea de 0V reprezintă “0” logic), valori ce sunt atinse atunci când inducţia magnetică depăşeşte pragurile de operare (“operating point”, respectiv “release point”). Pentru a îndeplini condiţiile aplicaţiei impuse,

senzorul Hall este realizat ca un circuit integrat, având în componenţă atât un element sensibil, cât şi circuite de condiţionare a semnalului [5], structura internă a senzorului fiind astfel alcătuită din:

- un stabilizator de tensiune pentru a regla tensiunea de alimentare a celorlalte componente din structura internă a senzorului;
- un element sensibil la variaţiile inducţiei magnetice a câmpului aplicat pe suprafaţa acestui element, a cărui tensiune de ieşire (generate prin efect Hall) este de ordinul unităţilor până la zeci de mV. În literatura de specialitate această componentă poartă denumirea de element Hall;
- un amplificator care să scaleze tensiunea de ieşire a elementului Hall la valori de ordinul volţilor urmărite a fi obţinute la ieşirea unui senzor Hall;
- o referinţă de tensiune şi specific unui senzor Hall digital, a unui declanşator Schmitt ("trigger Schmitt") în scopul comparării tensiunii de ieşire a elementului Hall în urma parcurgerii etajului de amplificare cu tensiunea de referinţă, la ieşirea declanşatorului Schmitt obţinându-se două nivele logice ce vor comanda baza unui tranzistor NPN;
- un tranzistor bipolar NPN ce operează în regimul de saturaţie, având colectorul conectat la ieşirea senzorului. Tranzistorul acţionează ca un comutator, având rolul de a aduce tensiunea de ieşire a senzorului la valoarea de 0V (când tranzistorul este în saturaţie) , respectiv la valoarea tensiunii de alimentare (5V, în cazul senzorului TLE4905, atunci când tranzistorul este blocat).

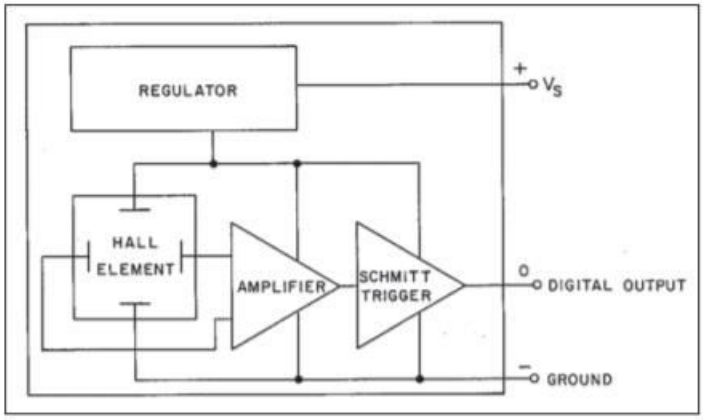


Figura 3.4 – Structura internă a unui senzor magnetic digital cu efect Hall. Sursa: [5]

3.2.3) Rolul senzorilor TLE4905 şi al butoanelor în cadrul sistemului

O etapă preliminară în realizarea unei table de şah inteligente constă în alegerea unui mecanism care să detecteze un set de date pe baza unor acţiuni externe, asigurând compatibilitatea din punct de vedere electric cu celelalte componente electronice. Aceaste afirmaţii precum şi analiza caracteristicilor prezentate în tabelul 3.1 au dus la alegerea senzorilor TLE4905.

În lucrarea de faţă s-au folosit şaizeci şi patru de senzori TLE4905, numărul fiind ales pe baza numărului de câmpuri existente pe o tablă de şah.

Rolul senzorilor TLE4905 constă în oferirea informaţiei privitoare la prezenţa sau absenţa unei piese de şah pe un anumit câmp. În cadrul capitolului 4 se va prezenta modul în care se pot diferenţia piesele în funcţie de tipul (Exemplu: cal, nebun, rege) şi culoarea (Exemplu: alb, negru) acestora, folosind procedee software. Toate aceste informaţii formează datele de intrare ale sistemului ce vor fi prelucrate, folosind dispozitivele prezentate în continuarea acestui capitol, procesarea şi controlul datelor de ieşire fiind sarcini atribuite firmware-ului, detalierea acestora fiind unul din subiectele capitolului 4.

Senzorul TLE4905 acţionează ca un comutator, oferind la ieşire semnalul “1” logic (la tensiunea de 5V) în cazul în care câmpul atribuit senzorului este liber, iar semnalul de “0” logic (la tensiunea de 0V) corespunde prezenţei unei piese pe câmpul respectiv. În urma finalizării detecţiei tuturor pieselor, şaizeci şi patru de biţi vor fi trimişi în paralel către registrele de deplasare SN74HC165. Pentru a garanta corectitudinea informaţiei transmise, piesele sunt prevăzute la baza lor cu magneţi de tip disc din neodim , senzorii fiind sensibili la valorile câmpului magnetic dezvoltat de către magneţi. De menţionat că senzorul TLE4905 este un senzor unipolar, fiind critică determinarea suprafeţei sensibile a acestuia, precum şi polaritatea corespunzătoare a magnetului.

Figura 3.5 – Set de magneţi de tip disc din neodim. Sursa: [14]

Pentru realizarea setărilor înainte de începerea unei partide, tabla de şah este prevăzută cu şapte butoane. Fiecare buton este conectat la câte un pin digital al plăcii de dezvoltare pentru diferenţierea acestora. În starea inactivă, butoanele trimit câte un semnal “1” logic către placa de dezvoltare. La acţionarea unui buton, se trimite un semnal “0 logic” către placa de dezvoltare, citirea acestor valori şi luarea deciziilor pe baza lor fiind realizate în cadrul firmware-ului.

De asemenea, atât în cazul senzorilor, cât şi în cazul butoanelor, asigurarea unei valori logice stabile în starea inactivă este asigurată atât de către rezistoare de pull-up externe de 10KΩ, logica de interconectare a senzorilor şi a butoanelor fiind prezentată în figura 3.6:

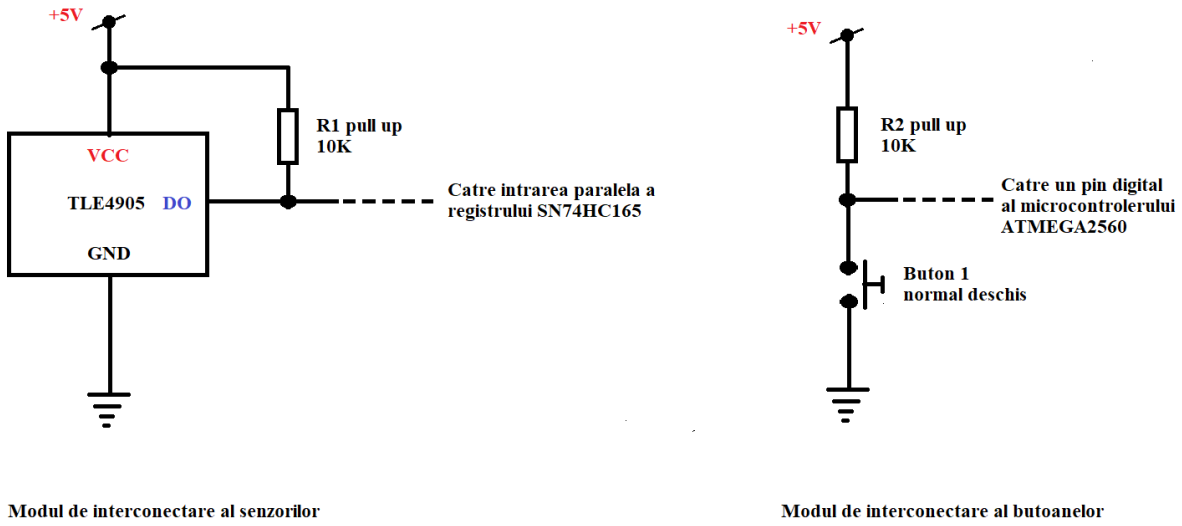


Figura 3.6 – Interconectarea senzorilor TLE4905 şi a butoanelor

Pe scurt, funcţionalităţile butoanelor sunt definite prin convenţia “Modul meniu / Modul de joc”, o introducere a acestora fiind prezentată mai jos (de asemenea, trebuie menţionat că în urma alimentării tablei de şah, primul mod de joc va fi întotdeauna “modul meniu”):

- Butonul 1 – Selectează modul de joc (începerea unui nou joc din poziţia iniţială) la o apăsare scurtă a butonului / Selectează modul meniu (resetarea poziţiei curente la poziţia initială şi posibilitatea de setare a altor timpi de joc pe ceasul de şah) la o apăsare scurtă a butonului în cazul în care pe tabla de joc nu apare o situaţie de mat şi în caz contrar la două apăsări scurte ale butonului (prima apăsare duce la stingerea tuturor LED-urilor, iar a doua duce la aprinderea LED-urilor cu culoarea portocalie pe câmpurile unde sunt prezente piesele de joc).
- Butonul 2 – Setarea timpului de joc al albului / Nedefinită. La o apăsare lungă a acestui buton în “modul meniu” se permite incrementarea timpului de joc al albului cu mai multe secunde.
- Butonul 3 – Setarea timpului de joc al negrului / Nedefinită. La o apăsare lungă a acestui buton în “modul meniu” se permite incrementarea timpului de joc al negrului cu mai multe secunde.
- Butoanele [4, 7] – Nedefinită / Selectarea piesei dorite la promovarea pionului pe câmpul de transformare la o apăsare scurtă a butonului.

Figura 3.7 – Identificarea butoanelor pe tabla de şah inteligentă

Număr buton	La mutarea albului – piesă ce înlocuieşte pionul alb	La mutarea negrului – piesă ce înlocuieşte pionul negru
4	Cal alb	Cal negru
5	Nebun alb	Nebun negru
6	Turn alb	Turn negru
7	Damă albă	Damă neagră

Tabel 3.2 – Acţionarea butoanelor pentru promovarea pionului

3.3) Blocul de procesare şi control

3.3.1) Scurtă prezentare a plăcii de dezvoltare cu microcontroler ATmega2560

În lucrarea de faţă s-a folosit o placă de dezvoltare cu microcontroler ATmega2560, pe 8 biţi, cu arhitectură Harvard, ce face parte din familia de microcontrolere AVR deţinute în prezent de către compania Microchip.

ATmega2560 reprezintă un circuit integrat ce înglobează multiple funcţionalităţi pe un singur cip. Din punct de vedere al blocurilor, ATmega2560 cuprinde:

- un CPU cu arhitectură RISC, având o frecvenţă a ceasului programabilă până la 16MHz, datorită arhitecturii putând să efectueze majoritatea instrucţiunilor într-o singură perioada de ceas (1 CPI) [6];

- o memorie EEPROM de 4KBytes, una SRAM de 8KBytes şi cea mai importantă, o memorie Flash programabilă de 256KBytes care limitează dimensiunea programelor ce pot fi încărcate de către utilizator [6];
- multiple periferice, dintre care vor fi amintite: două timere pe 8 biţi, patru timere pe 16 biţi, interfeţe seriale pentru stabilirea comunicaţiilor prin SPI, I2C, USART, un comparator analogic şi un convertor analog-digital pe 10 biţi [6].

Placa de dezvoltare cuprinde un număr de 70 de pini de uz general, dintre care 16 sunt pini analogici (A0-A15) şi 54 sunt pini digitali (numerotaţi de la 0 la 53). Prin setarea biţilor din cadrul unor registre de control, pot fi activate şi alte funcţionalităţi pentru anumiţi pini, cum ar fi: transmiterea unor semnale de ceas şi a datelor folosind protocoalele de comunicaţie (SPI, I2C, USART), generarea unor semnale PWM, introducerea semnalelor în vederea unei conversiei analog-digitale. În figura de mai jos este prezentată atât placa de dezvoltare, cât şi structura pinilor acesteia:

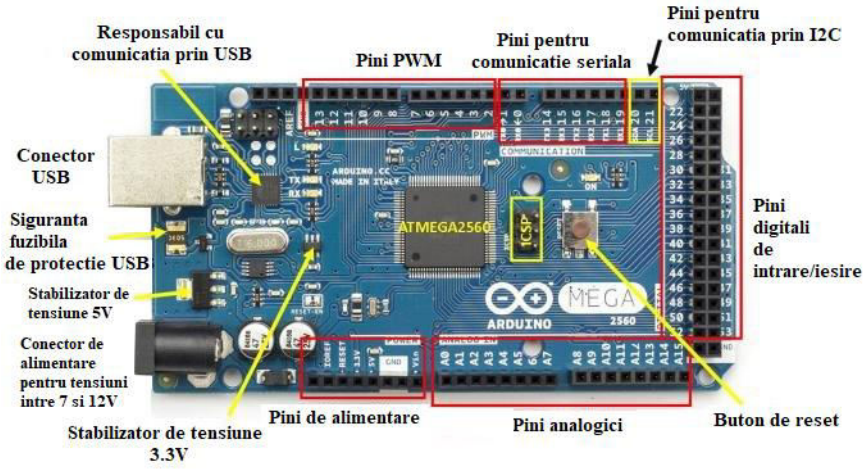


Figura 3.8 – Structura pinilor plăcii de dezvoltare cu microcontroler ATmega2560. Sursă: [7]

De menţionat că placa se poate alimenta la tensiunile de 3.3V, respectiv 5V, curentul maxim pe care îl poate absorbi/debita fiind de 200mA în cazul pinilor de alimentare şî GND, respectiv 20mA în cazul celorlalţi pini, programarea plăcii fiind realizată prin intermediul cablului USB.

3.3.2) Rolul microcontrolerului ATmega2560 în procesarea datelor şi controlul sistemului

Componenta cheie a tablei de şah inteligente este dată de către placa de dezvoltare cu microcontroler ATmega2560, deoarece aplicaţia firmware încărcată în memoria flash a microcontrolerului realizează cele mai grele operaţii, printre care se numără: configurarea interfeţei SPI pentru generarea semnalului de ceas şi selectarea dispozitivului cu care se doreşte iniţierea transferului de date (comunicaţiile ATmega2560 – SN74HC165N, respectiv ATmega2560 – MAX7219) şi a timer-ului 1 pentru implementarea ceasului de şah, recunoaşterea tipului şi a culorii pieselor de şah, implementarea regulilor jocului de şah în vederea indicării mutărilor posibile ale pieselor şi verificarea respectării acestor reguli.

Din punct de vedere al structurii hardware, microcontrolerul ATmega2560 are rolul de a prelua o valoare pe 64 de biţi provenită de la cei şaizeci şi patru de senzori magnetici TLE4905 şi transmisă serial de către registrele de deplasare SN74HC165N în scopul determinării modificării poziţiei pieselor pe tabla de şah prin procesarea acestei valori în aplicaţia firmware şî de a trimite informaţia prelucrată către cele şaizeci şi patru de LED-uri WS2812C pentru vizualizarea câmpurilor unde poate fi mutată piesa ridicată în poziţia curentă şi către modulul MAX7219 pentru a actualiza timpul de joc corespunzător fiecărui jucator în parte.

3.4) Stabilirea comunicaţiei între blocul de intrare şi blocul de procesare şi control

În cadrul subcapitolului 3.2 a fost descris rolul senzorilor TLE4905 în cadrul sistemului şi anume acela de a obţine informaţii referitoare la mutările pieselor ce au fost efectuate de către utilizatori prin trimiterea a 64 de biţi către microcontrolerul ATmega2560 în vederea procesării acestora. Transmiterea biţilor se poate realiza în mod direct, însă ar reprezenta o metodă total ineficientă, fiind necesari 64 de pini individuali pentru citirea informaţiei. De asemenea, din punct de vedere software, necesită un efort suplimentar aducerea datelor într-o formă compactă, pentru a uşura etapa de procesare. Toate aceste probleme sunt eliminate prin introducerea unui bloc hardware intermediar format din 8 registre de deplasare paralel-seriale SN74HC165N şi prin folosirea protocolului de comunicaţie SPI, aceste aspecte reprezentând subiecte ce vor fi discutate în continuare.

3.4.1) Scurtă prezentare a registrului de deplasare paralel-serial SN74HC165N

SN74HC165N reprezintă un registru de deplasare cu opt intrări paralele asincrone şi o ieşire serială sincronă, folosit pentru transmiterea compactă a datelor în grupuri de câte un byte.

Structura pinilor în cazul registrului realizat în capsulă DIP16 este prezentată în figura 3.9.

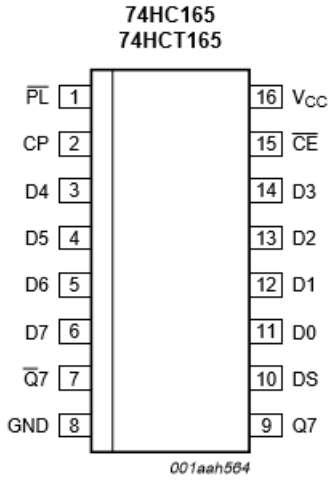


Figura 3.9 – Structura pinilor registrului de deplasare SN74HC165N. Sursă: [8]

În continuare se va face o scurtă prezentare a pinilor evidenţiaţi în figura 3.9:

- PL/LD (“Parallel Load”/”Load Data” [8]) – pin de intrare, în momentul în care acest semnal este activ (trimiterea valorii “0” logic), datele de pe intrarea paralelă (D7-D0) sunt încărcate într-un registru intern al modului SN74HC165N;
- CP (“Clock Input” [8]) – pin de intrare, în cadrul acestui pin se trimite semnalul de ceas partajat de către circuitele secventiale interne modulului SN74HC165N;
- D7-D0 – pini de intrare, reprezintă intrările de date paralele ale registrului;
- Q7 – pin de ieşire, reprezintă ieşirea serială a registrului , datele încărcate în pinii D7-D0 sunt trimise serial prin intermediul pinului Q7, în urma încărcării datelor în registru (aplicarea valorii “0” logic pe intrarea PL) şi activarea semnalului de ceas (trimiterea valorii “0” logic pe pinul CE). Pinul 7 al registrului trimite un semnal de ieşire rezultat prin negarea tuturor biţilor de pe ieşirea pinului Q7.
- CE/INH (“Clock Enable Input”/”Clock Inhibit” [8]) – pin de intrare, activează intrarea CP pe durata trimiterii valorii “0” logic pe pinul CE;
- DS (“Serial Data Input” [8]) – pin de intrare, permite introducerea serială a datelor de intrare, modulul SN74HC165N devenind în acest caz un registru serial-serial;
- V_{CC} – reprezintă pinul de alimentare;
- GND – reprezintă pinul de masă.

Funcţionalităţile pinilor prezentaţi anterior au fost determinate pe baza foii de catalog a registrului SN74HC165N prin citirea descrierii pinilor şi vizualizarea diagramei de timp prezentate în figura 3.10.

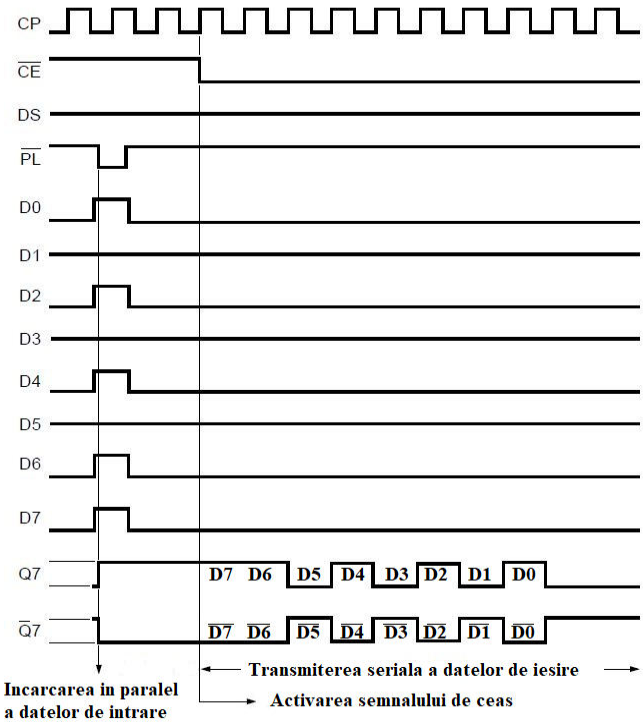


Figura 3.10 – Diagrama de timp a registrului SN74HC165N. Sursa: [8]

Principalele caracteristici ale registrului SN74HC165N sunt:

Parametru	Minim	Maxim
Vcc (tensiunea de alimentare)	-0.5 V	7 V
Icc (consum de curent)	-	50 mA
Io (curent de ieşire)	-	25 mA
Gama de temperatură	-40 ºC	125 ºC
f (frecvenţă de lucru)	-	51 MHz

Tabel 3.3 – Principalele caracteristici ale registrului SN74HC165N. Sursa: [8]

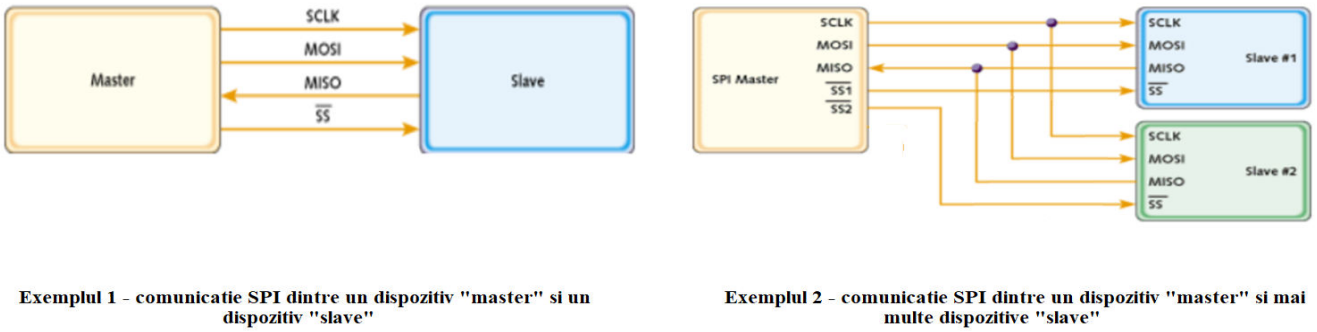
3.4.2) Protocolul SPI

SPI reprezintă un protocol de comunicaţie sincron, full-duplex (dispozitivele transmit şi recepţionează date simultan) introdus de către compania Motorola în scopul realizării unui transfer rapid de date (la frecvenţe de până la zeci de MHz [9]) între multiple dispozitive.

Pentru realizarea transferului de date, protocolul SPI introduce patru fire suplimentare în cadrul circuitului, terminalele la care sunt legate aceste fire purtând următoarele denumiri standardizate:

- SCLK (“Serial clock”) reprezintă pinul pe care se transmite semnalul de ceas utilizat pentru sincronizarea datelor;
- MISO (“Master Input – Slave Output) reprezintă pinul pe care se transmit semnale de date de la dispozitivul “slave” către dispozitivul “master”;
- MOSI (“Master Output – Slave Input) reprezintă pinul pe care se transmit semnale de date de la dispozitivul “master” către dispozitivul “slave”;
- SS (“Slave Select”) reprezintă pinul pe care se transmite semnalul de selecţie către dispozitivului “slave” cu care se doreşte iniţierea comunicaţiei.

Fiind un protocol de tip “single master-multi slave”, SPI permite realizarea comunicaţiei între un număr oricât de mare de dispozitive cu condiţia ca un singur dispozitiv să fie la un anumit moment de timp declarat “master”. Două exemple concrete de interconectare a dispozitivelor sunt prezentate în figura 3.10.



Exemplul 1 - comunicatie SPI dintre un dispozitiv "master" si un dispozitiv "slave"

Exemplul 2 - comunicatie SPI dintre un dispozitiv "master" si mai multe dispozitive "slave"

Figura 3.11 – Conectarea pinilor specifici protocolului SPI. Sursa: [9]

În lucrarea de faţă s-a folosit schema de interconectare din exemplul 2 al figurii de mai sus, microcontrolerul ATmega2560 fiind dispozitivul “master”, iar modulul MAX7219, respectiv cele 8 registre de deplasare SN74HC165 reprezentând dispozitivele “slave”.

Pentru a iniţia comunicaţia prin SPI, dispozitivul “master” trebuie să seteze parametrii semnalului de ceas pe pinul SCLK, având pe pinul SS valoarea “1” logic. De asemenea, dispozitivele “master” şi “slave” trebuie să încarce datele care urmează a fi transmise în registrele de deplasare ale interfeţei SPI. Începutul comunicaţiei se stabileşte prin setarea valorii “0” logic pe pinul SS de către dispozitivul “master”, transferul de date realizându-se în grupuri de câte opt biţi pe pinii MOSI şi MISO. După transmiterea a unuia sau mai multor grupuri de 8 biţi, dispozitivul “master” trebuie să seteze valoarea “1” logic pe pinul SS, indicând finalizarea comunicaţiei dintre dispozitive. Toate aceste aspecte sunt ilustrate în figura 3.12.

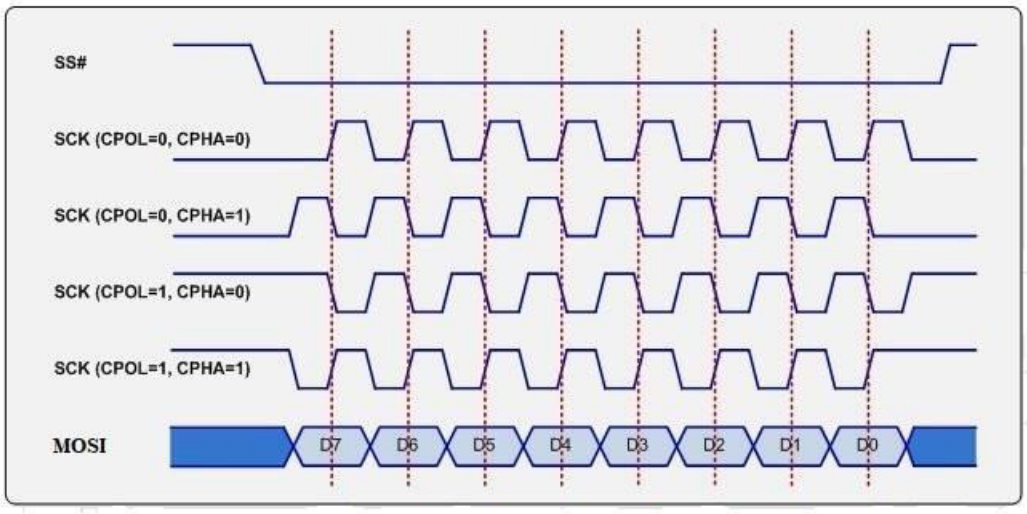


Figura 3.12 – Transmiterea unui pachet de date prin protocolul SPI. Sursa: [10]

. În figura 3.12 se observă, pe lângă informatiile expuse anterior, prezenţa unor biţi suplimentari, şi anume CPOL şi CPHA. În funcţie de valorile acestor biţi, se stabilesc ultimele detalii în ceea ce priveşte transferul datelor, cu liniile punctate fiind reprezentate fronturile active ale semnalului de ceas în cele patru moduri de funcţionare. Astfel, CPOL setează polaritatea semnalului de ceas (pentru CPOL = 0, semnalul de ceas porneşte cu valoarea “0” logic, iar pentru CPOL = 1, semnalul de ceas porneşte cu valoarea “1” logic), iar CPHA setează faza semnalului de ceas (pentru CPHA = 0, semnalul de ceas are defazajul nul, iar pentru CPHA = 1, defazajul este de 180º). Biţii CPOL şi CPHA se găsesc într-un registru de control intern al interfeţei SPI, fiind stabiliţi de către utilizator.

În vederea implementării comunicaţiei prin SPI în cadrul acestei lucrări, se folosesc interfeţele SPI ale modulelor, configurarea registrelor in scopul generării semnalului de ceas, respectiv a semnalelor de pe pinii SS fiind realizate de către aplicaţia firmware prezentată în cadrul capitolului 4, în acest capitol fiind evidenţiată schema bloc a interfeţei SPI în figura 3.13.

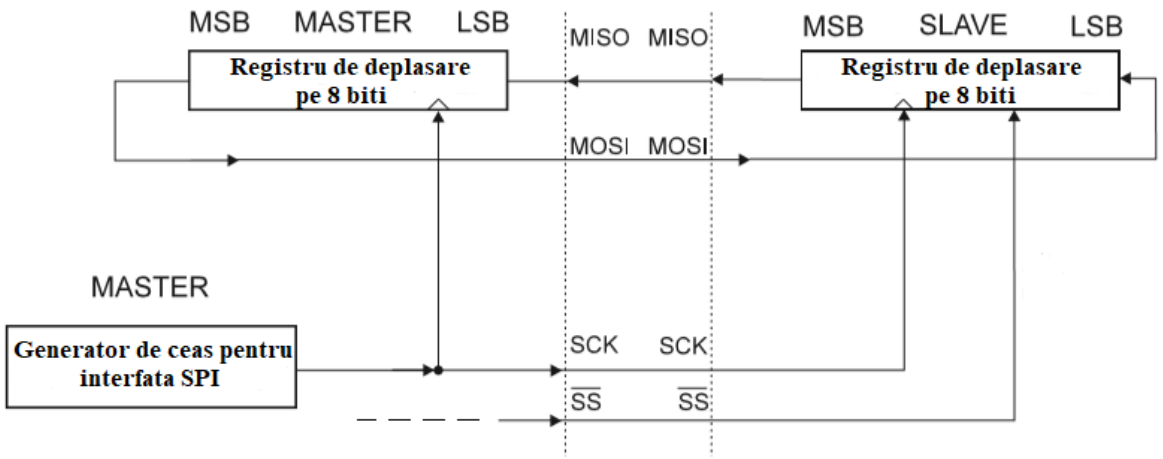


Figura 3.13 – Schema bloc a interfeţei SPI. Sursa: [6]

3.4.3) Rolul registrelor de deplasare SN74HC165N şi al protocolului SPI în cadrul sistemului

În primul paragraf al subcapitolului 3.4 au fost expuse principalele probleme care apar în cazul implementării lucrării de faţă în lipsa registrelor de deplasare SN74HC165N şi a unui protocol de comunicaţie menit să organizeze datele ce vor fi livrate către microcontrolerul ATmega2560, în acest paragraf fiind prezentat modul în care aceste probleme au fost rezolvate.

Astfel, pentru a uşura sarcina firmware-ului privind manipularea celor 64 de biţi citiţi de la senzori, soluţia adoptată a constat în concatenarea acestora şi asigurarea recepţiei corecte a datelor de către ATmega2560.

Principalul rol al registrului de deplasare SN74HC165N este de a converti un grup de 8 biţi transmişi în paralel de la senzorii magnetici TLE4905 într-un pachet de 8 biţi ce va fi transmis serial către placa de dezvoltare. Astfel, prin utilizarea a 8 registre de deplasare, pot fi trimise simultan toate informaţiile legate de poziţia curentă a pieselor pe tabla de şah.

Cele 8 registre de deplasare SN74HC165N au fost conectate în cascadă pentru a garanta recepţia secvenţială a celor 64 de biţi proveniţi de la senzori pe intrările paralele ale registrelor şi pentru a reduce numărul de interconexiuni către placa de dezvoltare de la şaizeci şi patru la o singură interconexiune. Deşi printre pinii registrelor de deplasare nu se găsesc notaţiile standard ale protocolului SPI (SCLK, SS, MOSI, MISO), prin analizarea amănunţită a princpiului de funcţionare corespunzător acestor registre, s-a constatat că se pot utiliza pinii CP, CE, respectiv Q7 pentru a realiza o interfaţă cu blocul de SPI intern al microcontrolerului ATmega2560, astfel:

- prin conectarea pinului CP, pentru fiecare din cele opt registre de deplasare SN74HC165N, la pinul SCLK (pinul 52) al plăcii de dezvoltare. Astfel, se rezolvă problema generării manuale a semnalului de ceas (prin introducerea întârzierilor sau configurarea unui timer din structura

internă a microcontrolerului), deoarece se utilizează implicit generatorul de ceas pentru interfaţa SPI a microcontrolerului (prezentate în figura 3.12);

- prin conectarea pinului CE, pentru fiecare din cele opt registre de deplasare SN74HC165N, la pinul 23 (definit ca pin de "slave select") al plăcii de dezvoltare. Prin această conexiune, ATmega2560 va putea decide momentul de timp în care să achiziţioneze grupul serial de 64 de biţi în vederea procesării acestuia;
- prin conectarea ieşirii seriale a ultimului registru SN74HC165N parcurs de date (pinul Q7) la pinul MISO (pinul 50) al plăcii de dezvoltare.

3.5) Blocuri de ieşire

3.5.1) Scurtă prezentare a LED-ului WS2812C şi a protocolului NRZ

WS2812C reprezintă un circuit integrat SMD realizat de către compania Worldsemi, alcătuit din trei LED-uri cu culorile roşu, verde, albastru (aflate în componenţa oricărui LED RGB) şi un circuit de control care permite limitarea curentului ce trece prin LED-uri, având la dispoziţie 256 de nivele de luminozitate pentru cele trei culori de bază, obţinându-se în total un număr de aproximativ 16 milioane de nuanţe de culori [11].

Acest modul este alcătuit din patru pini, doi pini fiind folosiţi pentru alimentarea cipului (VDD şi GND), iar ceilalţi doi pentru realizarea transferului de date cu alte dispozitive prin intermediul protocolului NRZ.

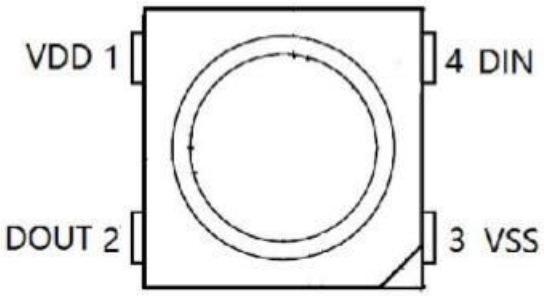
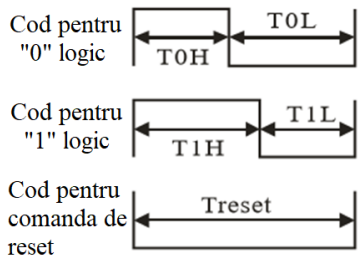


Figura 3.14 – Structura pinilor LED-ului WS2812C. Sursa: [11]

Protocolul NRZ constă în codarea informaţiei prin intermediul duratei lăţimilor corespunzătoare pulsurilor transmise. Astfel, în cazul LED-ului WS2812C sunt prezentate valorile celor două paliere, informaţia decodificată de către WS2812C putând fi: transmiterea unui "0" logic, transmiterea unui "1" logic sau o comandă de reset, după cum se poate observa şi din figura 3.15.



T0H	Durata palierului superior corespunzator codului "0" logic	220ns~380ns
T1H	Durata palierului superior corespunzator codului "1" logic	580ns~1μs
T0L	Durata palierului inferior corespunzator codului "0" logic	580ns~1μs
T1L	Durata palierului inferior corespunzator codului "1" logic	220ns~420ns
RES	Durata palierului inferior corespunzator codului pentru comanda de reset	>280μs

Figura 3.15 – Diagrama de timp a protocolului NRZ în cazul LED-ului WS2812C. Sursa: [11]

Selectarea culorii dorite se realizează prin trimiterea unui pachet de 24 de biţi prin intermediul protocolului NRZ pe pinul de intrare al modulului WS2812C (pinul DIN). Primii 8 biţi transmişi reprezintă nivelul de luminozitate al culorii verde, următorii 8 biţi determină nivelul de luminozitate al culorii roşu, iar ultimii 8 biţi selectează nivelul de luminozitate al culorii albastru. Prin combinarea celor trei culori cu diferite nivele de luminozitate se pot obţine cele 16 milioane de nuanţe de culori amintite anterior. În cadrul pachetului, biţii cei mai semnificativi se transmit primii, iar biţii cei mai puţin semnificativi se transmit ultimii.

3.5.2) Rolul LED-urilor WS2812C în cadrul sistemului

În cadrul lucrării de faţă, s-au folosit şaizeci şi patru de LED-uri WS2812C, corespunzător celor şaizeci şi patru de câmpuri de pe tabla de şah.

Toate LED-urile au fost conectate în cascadă pentru a putea fi comandate simultan prin intermediul unei valori pe 1536 de biţi (24 de biţi pentru fiecare LED în parte), transmiterea datelor fiind realizată folosind un singur fir situat între pinul 24 al plăcii de dezvoltare şi pinul DIN corespunzător primului LED din lanţ. Pentru a facilita controlul culorilor ce trebuie afişate de către LED-uri, în cadrul aplicaţiei firmware s-a introdus o structură în care pot fi încărcate nivelele de luminozitate (între 0 şi 255) pentru cele trei culori ale unui LED, întregul lanţ fiind controlat printr-un vector de şaizeci şi patru de elemente al cărui tip de date este structura enunţată anterior.

Modul de interconectare al dispozitivelor WS2812C (L1, L2, …, L64) este prezentat în figura 3.16, folosind condensatoare (C1, C2, …, C64) cu capacitatea de 100nF pentru filtrarea zgomotului ce ar putea apărea pe pinul de alimentare şi ar introduce erori în transmiterea datelor, aspect recomandat şi în foaia de catalog a modulului WS2812C.

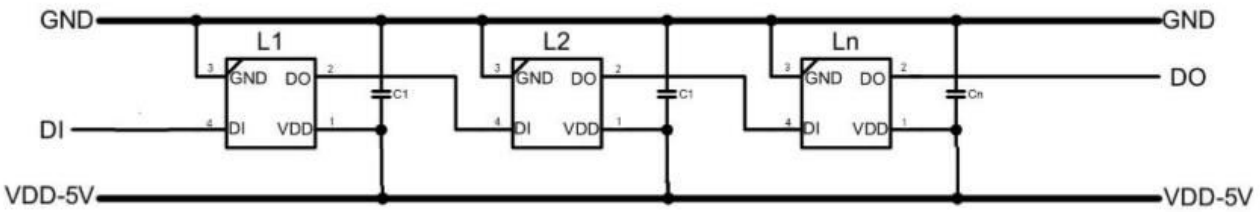


Figura 3.16 – Interconectarea în cascadă a celor 64 de module WS2812C. Sursa: [11]

Fiecare culoare a LED-ului afişată pe tabla de şah are rolul de a marca o situaţie ce poate interveni în cadrul partidei, convenţia de culori şi evenimentele marcate sunt specificate în tabelul 3.4.

Culoare LED	Mod de aprindere	Eveniment marcat
Portocaliu	Continuu	În modul meniu, aprinderea LED-urilor este declanşată de introducerea pieselor pe câmpurile respective pentru a verifica că microcontrolerul detectează prezenţa pieselor pe tabla de șah.
Verde	Continuu	În modul de joc, la ridicarea unei piese se indică câmpurile posibile unde poate fi plasată piesa ridicată.
Roşu	O singură dată, scurt.	În modul de joc, la plasarea piesei pe un câmp eronat (diferit de cele aprinse cu culoarea verde).
Albastru	O singură dată, scurt.	În modul de joc, la plasarea piesei pe un câmp indicat cu culoarea verde, pentru a confirma mutarea efectuată.
Albastru	Continuu	În modul de joc, marchează piesa care a generat situaţia de mat sau faptul că un pion ajunge pe câmpul de transformare.
Violet	O singură dată, scurt.	În modul de joc, confirmă apăsarea butonului pentru transformarea pionului ajuns pe câmpul de promovare.
Galben	Intermitent, până la resetarea partidei.	În modul de joc, se aprinde LED-ul de pe câmpul unde este situat regele negru care a primit mat, marcând faptul că jucătorul cu piesele albe a câştigat partida.
Turcoaz	Intermitent, până la resetarea partidei.	În modul de joc, se aprinde LED-ul de pe câmpul unde este situate regele alb care a primit mat, marcând faptul că jucătorul cu piesele negre a câştigat partida.

Tabel 3.4 – Convenţia culoare – eveniment marcat

3.5.3) Scurtă prezentare a modulului MAX7219

MAX7219 este un circuit integrat realizat de către compania Maxim Integrated, alcătuit dintr-un cip MAX7219 ce reprezintă un driver cu intrare serială şi ieşire serială utilizat pentru a interfaţa un

afişaj cu şapte segmente cu o placă de dezvoltare, în scopul asigurării unui control precis asupra modului de reprezentare al datelor şi un afişaj cu şapte segmente şi opt digiţi, având LED-uri în conexiunea catod comun [12].

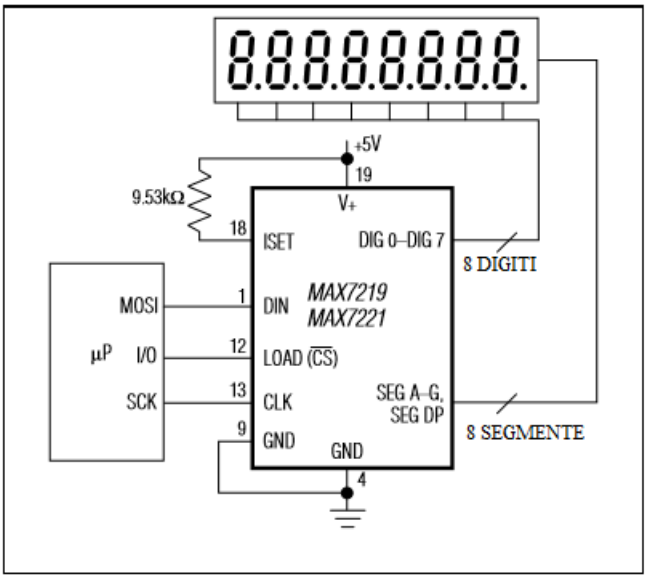


Figura 3.17 – Schema bloc a modului MAX7219 şi modul de interconectare al pinilor SPI la un microprocesor. Sursa: [12]

Controlul afişajului cu şapte segmente şi opt digiţi este uşor de realizat, deoarece modulul MAX7219 conţine o interfaţă ce permite comunicarea cu alte dispozitive prin intermediul protocolului SPI. De menţionat că MAX7219 conţine un set de treisprezece registre pe opt biţi, pentru accesarea acestora fiind necesară trimiterea unui pachet de 16 biţi, primul octet pentru selectarea adresei registrului şi al doilea octet pentru introducerea noii valori în acel registru (configurarea modului MAX7219 fiind detaliată în capitolul 4).

REGISTER	ADDRESS					HEX CODE
	D15–D12	D11	D10	D9	D8	
No-Op	X	0	0	0	0	0xX0
Digit 0	X	0	0	0	1	0xX1
Digit 1	X	0	0	1	0	0xX2
Digit 2	X	0	0	1	1	0xX3
Digit 3	X	0	1	0	0	0xX4
Digit 4	X	0	1	0	1	0xX5
Digit 5	X	0	1	1	0	0xX6
Digit 6	X	0	1	1	1	0xX7
Digit 7	X	1	0	0	0	0xX8
Decode Mode	X	1	0	0	1	0xX9
Intensity	X	1	0	1	0	0xXA
Scan Limit	X	1	0	1	1	0xXB
Shutdown	X	1	1	0	0	0xXC
Display Test	X	1	1	1	1	0xXF

Tabel 3.5 – Maparea registrelor în cadrul modulului MAX7219. Sursa: [12]

Astfel, în registrele "Digit 0 – Digit7" sunt stocate valorile afişate pe cei opt digiţi, registrul "Decode Mode" activează decodorul intern al modului MAX7219 pentru a nu mai fi necesară aprinderea individuală a fiecărui LED din cadrul unui digit, valoarea stocată în registrul "Intensity" reglează intensitatea luminoasă a LED-urilor din componenţa afişajului cu şapte segmente, registrul "Scan Limit" activează/dezactivează conţinutul afişat pe digiţi, registrul "Shutdown" selectează modul de operare (mod normal sau mod "Shutdown" cu consum redus de putere), iar registrul "Display Test" aprinde toate LED-urile afişajului pentru testarea funcţionării acestora.

3.5.4) Rolul modulului MAX7219 şi al protocolului SPI în implementarea unui ceas de şah

În lucrarea de faţă, principalul rol al modului MAX7219 constă în afişarea timpului de joc rămas în cadrul partidei de şah. Digiţii 7,6,5,4 permit vizualizarea timpului de joc al jucătorului cu piesele albe, în timp ce digiţii 3,2,1,0 permit vizualizarea timpului de joc al jucătorului cu piesele negre.

Se stabileşte comunicaţia prin SPI, prin configurarea microcontrolerului ATmega2560 ca dispozitiv "master" şi a modului MAX7219 ca dispozitiv "slave" şi prin interconectarea pinilor ca în tabelul 3.6.

ATmega2560	MAX7219
SCLK (pinul 52)	CLK
SS (pinul 22)	CS
MOSI (pinul 51)	DIN

Tabel 3.6 – Corespondenţa pinilor interfeţei SPI în cazul comunicaţiei dintre ATmega2560 şi MAX7219

Prin configurarea timer-ului 1 pe 16 biţi al microcontrolerului ca bază de timp, implementarea unei rutine de deservire a întreruperii (întrerupere activată atunci când timer-ul atinge o valoare prestabilită) care actualizează timpul de joc şi afişarea informaţiilor cu ajutorul modului MAX7219 se realizează ceasul integrat în tabla de şah. Mai multe detalii despre timer şi rutina de deservire a întreruperii se găsesc în cadrul capitolului 4.

Afişarea timpului de joc pe ceasul de şah se realizează prin actualizarea registrelor "Digit 0-Digit 7" la intervale de o secundă, iar prin intermediul protocolului SPI se transmit pachetele de date pentru actualizarea digiţilor, formatul serial al datelor fiind prezentat în figura 3.18.

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	X	Biti de adresa				MSB	Biti de date						LSB

Figura 3.18 – Formatul pachetului de date transmis către modulul MAX7219. Sursa: [12]

3.6) Sursa de alimentare

Pentru a asigura funcţionalitatea unui număr mare de componente (mai precis 138, fără a include în acest calcul numărul de componente pasive), tabla de şah inteligentă trebuie alimentată folosind o sursă de tensiune cu o capabilitate mare în curent. Astfel, în lucrarea de faţă s-a utilizat sursa de tensiune stabilizată în comutaţie SP-50-5V cu următoarele caracteristici principale:

- Tensiune de ieşire: 5V;
- Curent maxim de ieşire: 10A;
- Intervalul de temperaturi de operare: [-10, 50]ºC;
- Protecţie la scurtcircuit şi la supracurent.

Sursa de tensiune SP-50-5V este prezentată în figura 3.19:

Figura 3.19 – Sursă de tensiune stabilizată în comutaţie SP-50-5V. Sursa: [13]

Capitolul 4 – Structura software a tablei de şah

4.1) Schema bloc a structurii software

În acest capitol se realizează o prezentare a aplicaţiei firmware dezvoltate pentru tabla de şah inteligentă. Firmware-ul reprezintă componenta software a sistemului şi are un rol esenţial, având de rezolvat cele mai grele sarcinii în vederea obţinerii funcţionalităţii dorite, toate acestea fiind prezentate pe scurt în structura fişierelor din figura 4.1 şi detaliate în continuarea acestui capitol.

Limbajul de programare folosit în cadrul acestei lucrării este limbajul C, un limbaj compilat ce urmează conceptele programării procedurale, prin împărţirea unui program în funcţii menite să realizeze sarcinile mai simple dintr-o structură de cod complexă. Aplicaţia firmware a fost împărţită în două structuri de cod, numite în continuare “etapa de configurare” realizată cu ajutorul mediului de dezvoltare Atmel Studio 7, respectiv “etapa de descriere a regulilor jocului de şah” realizată în mediul de dezvoltare Visual Studio 2019, în final realizându-se concatenarea fişierelor şi a instrucţiunilor în cadrul programului principal.

În etapa de configurare, interacţiunea cu hardware-ul sistemului este preponderentă, în timp ce în etapa de descriere a regulilor jocului de sah, accesarea hardware-ului se realizează prin intermediul funcţiilor realizate în etapa de configurare, exemple fiind citirea butoanelor şi aprinderea LED-urilor WS2812C. Schema bloc a structurii software este prezentată în figura 4.1.

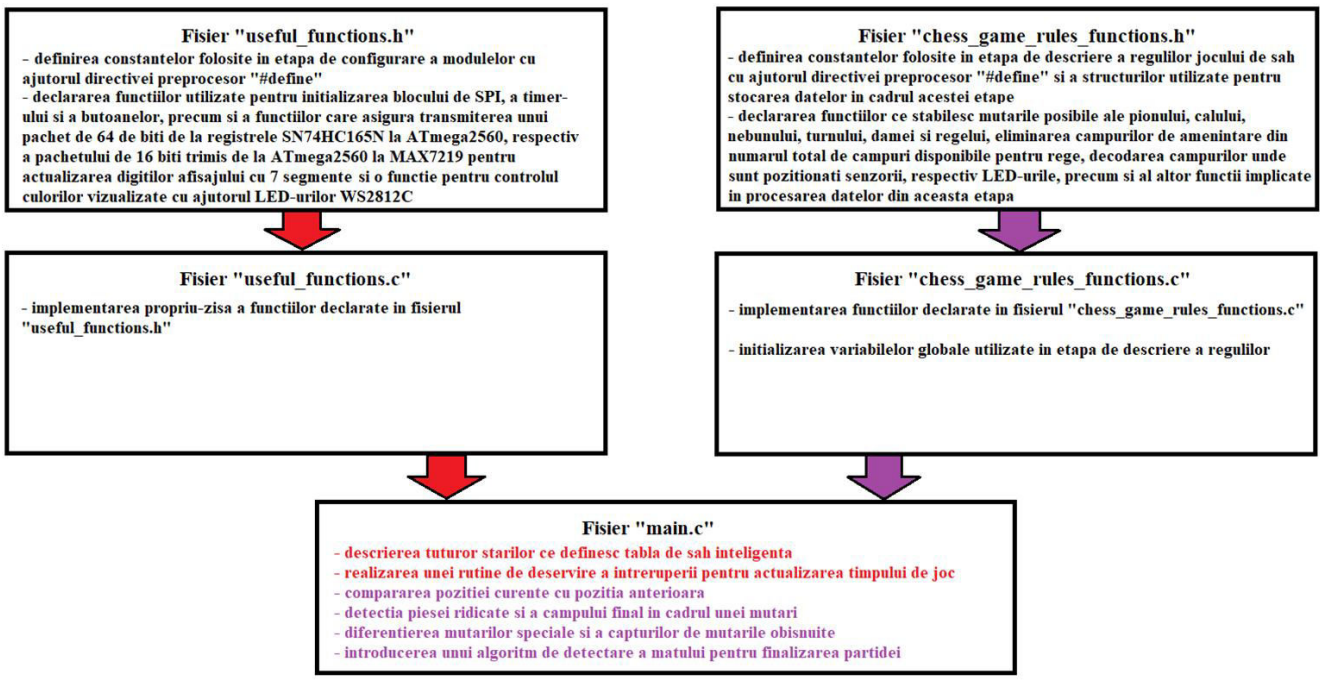


Figura 4.1 – Schema bloc a structurii software corespunzătoare tablei de şah inteligente

4.2) Structura de cod pentru configurarea modulelor si implementarea protocoalelor de comunicaţie

4.2.1) Implementarea software a protocolului SPI (funcţii de iniţializare, transmiterea datelor către modulul MAX7219 şi recepţionarea datelor de la registrele de deplasare SN74HC165)

În subcapitolul 3.4.2 au fost prezentate noţiunile teoretice care descriu principiul de funcţionare al protocolului SPI. În acest subcapitol, se prezintă implementarea acestui standard în firmware-ul tablei de şah inteligente.

În lucrarea de faţă se evidenţiază două linii de comunicaţie stabilite prin SPI, şî anume:

- între ATmega2560 (dispozitiv "master") şi cele 8 registre SN74HC165N (dispozitiv "slave"), în scopul transmiterii seriale a 64 de biţi în pachete de câte 8 biţi către microcontroler;
- între ATmega2560 (dispozitiv "master") şi modulul MAX7219 (dispozitiv "slave"), pentru transmiterea datelor în pachete de câte doi octeţi, primul folosit pentru selectarea registrului intern al modului MAX7219 şi al doilea pentru stabilirea datelor ce vor fi încărcate în registrul ales.

Prima funcţie implicată în acest proces este "SPI_masterInit()" şi are rolul de a configura registrele interne ale interfeţei SPI corespunzătoare microcontrolerului ATmega2560.

```
void SPI_masterInit(void)
{
	DDR_SPI_extension |= ((1 << DDR_SS_transmit) | (1 << DDR_SS_receive_control));	/* Set
master SS_transmit and SS_receive_control as output */
	DDR_SPI |= ((1 << DDR_SS_receive) | (1 << DDR_SCK) | (1 << DDR_MOSI));		/* Set
SCK and MOSI as output, SS_receive and MISO as input */
	PORT_SPI_extension |= ((1 << PIN_SS_transmit) | (1 << PIN_SS_receive_control));	/*
Disable master SS_transmit and SS_receive_control pins (both active low) */
	SPCR |= (1 << SPE) | (1 << MSTR) | (1 << SPR0);				/*
Enable SPI, configure ATmega2560 as master device, set the clock rate at f_sys_ck/16 */
	SPCR |= (1 << CPOL);
	SPCR &= ~(1 << CPHA);
	/* Take data on the rising edge of ck(SPI data format), data order is MSB first */
}
```

Figura 4.2 – Funcţia "SPI_masterInit()"

Primele trei linii de cod din cadrul acestei funcţii configurează pinii porturilor A şi B ale microcontrolerului, practic pinii SS_transmit (pinul SS care selectează modulul MAX7219), SS_receive_control (pinul SS care selectează registrele de deplasare SN74HC165N), SCLK, MISO şi MOSI prin intermediul cărora se vor realiza comunicaţiile prin SPI între ATmega2560 şi cele două module externe. Pinilor SS_transmit, respectiv SS_receive control li se atribuie valorile "1" logic, pentru a marca starea inactivă a acestora.

Următoarele trei linii de cod realizează setările interfeţei SPI a microcontrolerului ATmega2560 prin intermediul registrului de control SPCR, pinii evidenţiaţi ai acestui registru având următoarele semnificaţii:

- SPE (“SPI Enable” [6]) reprezintă bitul care activează blocul de SPI la scrierea valorii “1” logic;
- MSTR (“Master/Slave Select” [6]) selectează microcontrolerul ATmega2560 ca dispozitiv “master” la introducerea valorii “1” logic, respectiv ca dispozitiv “slave” la scrierea valorii “0” logic;
- grupul de biţi SPI2X (“Double SPI speed bit” [6]), SPR1 şi SPR0 (“SPI Clock Rate Select 1 and 0” [6]) determină frecvenţa semnalului de ceas folosit în cadrul protocolului SPI ({SPI2X, SPR1 şi SPR0} = {0,0,1} setează frecvenţa semnalului de ceas la 1MHz). Pentru mai multe detalii se poate consulta foaia de catalog a microcontrolerului ATmega2560, secţiunea “Serial Peripheral Interface”;
- biţii CPOL şî CPHA au fost discutaţi în cadrul subcapitolului 3.4.2. În această lucrare, s-a folosit combinaţia {CPOL, CPHA} = {1, 0} , achiziţia datelor fiind realizată pe frontul negativ al semnalului de ceas.

Funcţiile responsabile pentru trimiterea celor 64 de biţi de la cele 8 registre de deplasare SN74HC165N la placa de dezvoltare sunt “SN74HC165_SPI_masterReceive()” şi “SN74HC165_SPI_masterReceive_eightBytes()”.

```
char SN74HC165_SPI_masterReceive(void)
{
	char SPI data received;
	PORT_SPI_extension &= ~(1 << PIN_SS_receive_control);		/* Start SPI
reception, master pull SS low */
	/* (emulating this command because the slave device does not have a dedicated SPI
interface) */
	SPDR = 0x00;							/* Send dummy data
to take the received useful data */
	while(!(SPSR & (1 << SPIF)));					/* Wait for
transmission complete */
	SPI_data_received = SPDR;					/* Temporary
storage of the received data */
	PORT_SPI_extension |= (1 << PIN_SS_receive_control);		/* End SPI
reception, master pull SS high  */
	return SPI_data_received;					/* Return received
data */
}
```

Figura 4.3 – Funcţia “SN74HC165_SPI_masterReceive()”

Rolul funcţiei “SN74HC165_SPI_masterReceive()” este de a trimite un octet de date de la SN74HC165N la microcontrolerul ATmega2560. Pentru realizarea acestei sarcini, se realizează următoarele operaţii: se trimite “0” logic pe pinul SS_receive_control pentru selectarea registrelor SN74HC165N ca dispozitiv “slave”, se introduce o valoare neutră în registrul SPDR (fiind un protocol full-duplex, pentru a recepţiona un octet, trebuie de asemenea transmis simultan un octet de date), registru ce stochează valorile transmise şi recepţionate prin intermediul interfeţei SPI (de asemenea, scrierea în registrul SPDR declanşează procesul de transmitere a datelor), se asteaptă finalizarea trimiterii/recepţionării octetului de date (indicat prin introducerea valorii “0” logic pe bitul SPIF al registrului SPSR), se citeşte octetul primit de la registrele de deplasare, valoare ce va fi returnată de către funcţie, se trimite “1” logic pe pinul SS_receive_control pentru a marca finalizarea comunicaţiei.

În cadrul funcţiei “SN74HC165_SPI_masterReceive_eightBytes()” se încarcă cei 64 de biţi citiţi de la senzori pe intrările paralele ale registrelor SN74HC165N (prin trimiterea unui puls de valoare “0” logic timp de 10us pe intrările PL/LD ale registrelor de deplasare), se apelează de opt ori funcţia “SN74HC165_SPI_masterReceive()”, se concatenează la fiecare iteraţie octetul recepţionat într-o variabilă pe 64 de biţi şi se returnează valoarea stocată în această variabilă.

În cazul modulului MAX7219 se remarcă funcţiile “MAX7219_SPI_masterTransmit()”, “MAX7219_SPI_configureDisplay_decoded()”, “MAX7219_SPI_displayWhiteTime()”, respectiv “MAX7219_SPI_displayBlackTime()”.

```
/* 2.3) */
void MAX7219_SPI_masterTransmit(uint8_t address_SPI, uint8_t data_SPI)
{
	PORT_SPI_extension &= ~(1 << PIN_SS_transmit);
/* Start SPI transmission, master pull SS low */
	SPDR = address_SPI;                                          /* Transmit address
byte */
	while(!(SPSR & (1 << SPIF)));                                /* Wait for
transmission complete */
	SPDR = data_SPI;                                             /* Transmit data
byte */
	while(!(SPSR & (1 << SPIF)));                                /* Wait for
transmission complete */
	PORT_SPI_extension |= (1 << PIN_SS_transmit);
/* End SPI transmission, master pull SS high(other devices can be the master device) */
}

/* 2.4) */
void MAX7219_SPI_configureDisplay_decoded(void)
{
	SPI_masterInit();
	MAX7219_SPI_masterTransmit(0x0C, 0x01);                      //Configure shutdown
register to normal mode
	_delay_ms(10);
	MAX7219_SPI_masterTransmit(0x0A, 0x07);                      //Configure intensity
register to obtain ((15/32) * maximum intensity) of the display
	_delay_ms(10);
	MAX7219_SPI_masterTransmit(0x0B, 0x07);                      //Configure scan-limit
register to display all digits
	_delay_ms(10);
	MAX7219_SPI_masterTransmit(0x09, 0xFF);                      //Configure decode
register to enable decode mode
	_delay_ms(10);
	MAX7219_SPI_masterTransmit(0x0F, 0x00);                      //Configure display
test register to disable test mode
	_delay_ms(10);
}
```

Figura 4.4 – Funcţiile “MAX7219_SPI_masterTransmit() şi “MAX7219_SPI_configureDisplay_decoded()”

Funcţia “MAX7219_SPI_masterTransmit()” realizează trimiterea datelor prin SPI de la ATmega2560 către MAX7219. Se observă că modul în care este implementată comunicaţia este asemănător cu cel prezentat în cadrul funcţiei “SN74HC165_SPI_masterReceive()”. Faţă de cazul funcţiei descrise anterior, această funcţie utilizează un alt pin al plăcii de dezvoltare pentru selectarea cipului “slave” şi anume pinul SS_transmit, iar datele încărcate în registrul SPDR nu mai sunt neutre (în scopul recepţionării altor date de la dispozitivul “slave”), ci reprezintă un grup de doi octeţi, primul fiind folosit pentru identificarea registrului intern al modului MAX7219 (octet de adresă), iar al doilea reprezintă datele ce vor fi încărcate în registrul selectat (octet de date). Funcţia “MAX7219_SPI_configureDisplay_decoded()” realizează setările iniţiale ale modului MAX7219 prin

apelarea funcţiei “MAX7219_SPI_masterTransmit()”, aceste setării fiind: activarea modului normal de operare, selectarea intensităţii luminoase a segmentelor la 15/32 din valoarea maximă a acesteia, selectarea tuturor celor 8 digiţi pentru afişarea conţinutului acestora, activarea decodorului intern al modulului MAX7219, dezactivarea modului de test.

```
/* 6.5) */
void MAX7219_SPI_displayWhiteTime(uint8_t main_global_white_minutes, uint8_t main_global_white_seconds)
{
	MAX7219_SPI_masterTransmit(0x05, (main_global_white_seconds % 10));
	MAX7219_SPI_masterTransmit(0x06, ((main_global_white_seconds / 10) % 10));
	MAX7219_SPI_masterTransmit(0x07, (0x80 | (main_global_white_minutes % 10)));
	MAX7219_SPI_masterTransmit(0x08, ((main_global_white_minutes / 10) % 10));
}
/* 6.6) */
void MAX7219_SPI_displayBlackTime(uint8_t main_global_black_minutes, uint8_t main_global_black_seconds)
{
	MAX7219_SPI_masterTransmit(0x01, (main_global_black_seconds % 10));
	MAX7219_SPI_masterTransmit(0x02, ((main_global_black_seconds / 10) % 10));
	MAX7219_SPI_masterTransmit(0x03, (0x80 | (main_global_black_minutes % 10)));
	MAX7219_SPI_masterTransmit(0x04, ((main_global_black_minutes / 10) % 10));
}
```

Figura 4.5 – Funcţiile “MAX7219_SPI_displayWhiteTime()” şi “MAX7219_SPI_displayBlackTime()”

Prin folosirea unui mecanism descris în subcapitolul 4.2.2, se obţin valorile minutelor şi ale secundelor rămase celor doi jucători în partida de şah, rolul funcţiilor “MAX7219_SPI_displayWhiteTime()” şî “MAX7219_SPI_displayBlackTime()” fiind acela de a permite vizualizarea acestor timpi pe ceasul de şah. Valorile minutelor şi ale secundelor vor fi stocate în registrele “Digit 7 – Digit 4” în cazul albului, respectiv în registrele “Digit 3 – Digit 0“ în cazul negrului. Operaţia SAU-logic pe biţi cu constanta “0x80” are rolul de a activa punctul zecimal ce separă minutele de secunde pe ceasul de şah.

4.2.2) Configurarea timer-ului intern al microcontrolerului ATmega2560 şi implementarea ceasului de şah

În subcapitolele 3.5.4 şi 4.2.1 s-a descris modulul MAX7219 şi rolul acestuia în afişarea timpului de joc pe ceasul de şah. În acest subcapitol se va prezenta rolul timer-ului 1 al microcontrolerului ATmega2560 şi al firmware-ului în completarea realizării unui ceas de şah integrat în tablă.

Timer 1 reprezintă blocul intern al microcontrolerului ATmega2560 care se ocupă de măsurarea timpului pentru gestionarea diferitelor evenimente, generarea semnalelor şi a întârzierilor.

În lucrarea de faţă, rolul timer-ului 1 este de a genera o bază de timp pentru a actualiza la fiecare secundă timpul de joc al jucătorului care este la mutare. În continuare se va demonstra că această funcţionalitate se poate obţine prin configurarea corectă a timer-ului (generarea bazei de timp) şi utilizarea unei întreruperi declanşate la apariţia unui eveniment specific timer-ului (rutina de deservire a întreruperii asigurând actualizarea timpului de joc la fiecare secundă).

```
/* 6) Timer1 and chess clock implementation - data types and functions: */
/* 6.2) */
void ATMEGA2560_TIMER1_init(void)
{
	TCCR1B |= (1 << WGM12); /* Configure timer1 in CTC mode */
	TCCR1B &= ~((1 << CS12) | (1 << CS11) | (1 << CS10)); /* Configure timer1 clock to no
clock source in order to disable timer1 */
	TCNT1 = 0; /* Initialize timer1 register. */
	OCR1A = 19999; /* Configure timer1 output compare match A register with the converted
value */
	TIMSK1 |= (1 << OCIE1A); /* Enable output compare match A interrupt */
}
/* 6.3) */
void ATMEGA2560_TIMER1_start(void)
{
	TCCR1B |= (1 << CS11); /* Configure timer1 with internal clock and prescaler = 8 in order
to enable timer1 */
}
/* 6.4) */
void ATMEGA2560_TIMER1_stop(void)
{
	TCCR1B &= ~((1 << CS12) | (1 << CS11) | (1 << CS10)); /* Configure timer1 clock to no
clock source in order to disable timer1 */
}
```

Figura 4.6 – Funcţii pentru configurarea, pornirea şi oprirea timer-ului 1

În cadrul funcţiei “ATmega2560_TIMER1_init()” se introduc valori în registrele microcontrolerului ATmega2560, în scopul realizării setărilor iniţiale ale timer-ului 1. În registrul TCCR1B (“Timer Control Register B” [6]) se setează bitul WGM12 (“Waveform Generation Mode Bit 2 for Timer 1” [6]), deoarece grupul {WGM12, WGM11, WGM10} = {1, 0, 0} configurează timer-ul în modul de operare CTC care presupune resetarea timer-ului atunci când acesta ajunge la o valoare prestabilită (în acest caz, 19999) setată iniţial în registrul OCR1A (“Output Compare Register 1 A” [6]). TCNT1 (“Timer 1 Register” [6]) reprezintă registrul în care sunt stocate valorile numerate de către timer, acesta fiind iniţializat cu valoarea 0, iar resetarea acestui registru la valoarea 0 având loc de fiecare dată când timer-ul numără valoarea 19999. Bitul OCIE1A (“Output Compare A Match Interrupt Enable” [6]) al registrului TIMSK1 (“Timer 1 Interrupt Mask Register” [6]) activează întreruperea declanşată de atingerea valorii 19999 în registrul TCNT1, iar în programul principal se activează întreruperile la nivel global prin intermediul metodei “sei()”. Grupul de biţi {CS12, CS11,CS10} = {0, 1, 1} (“Clock Select Bits [6]”) setează frecvenţa de lucru a timer-ului 1 la valoarea de 250kHz, prin selectarea unui factor de scalare (“prescaler”) faţă de frecvenţa maxima de operare a CPU-ului microcontrolerului ATmega2560 (ce are valoarea de 16MHz). Funcţiile “ATmega2560_TIMER1_start()” şi “ATmega2560_TIMER1_stop()” realizează pornirea şi oprirea timer-ului prin activarea (setarea {CS12, CS11,CS10} = {0, 1, 1}) , respectiv întreruperea (setarea {CS12, CS11, CS10} = {0, 0, 0}) liniei ce furnizează semnalul de ceas cu frecvenţa de 250kHz. Prin aceste două funcţii se asigură trecerea de la contorizarea timpului de joc al albului la cel al negrului şi invers.

Valoarea stocata in registrul OCR1A	f_CPU [Hz]	Valoarea factorului de scalare	Perioada timer-ului [s]	Numar de iteratii al ciclilor de numarare = 1s / Perioada timer-ului [s]
0	16000000	1	6.25E-08	16000000
0	16000000	8	0.0000005	2000000
0	16000000	64	0.000004	250000
0	16000000	256	0.000016	62500
0	16000000	1024	0.000064	15625
999	16000000	1	0.0000625	16000
999	16000000	8	0.0005	2000
999	16000000	64	0.004	250
999	16000000	256	0.016	62.5
999	16000000	1024	0.064	15.625
9999	16000000	1	0.000625	1600
9999	16000000	8	0.005	200
9999	16000000	64	0.04	25
9999	16000000	256	0.16	6.25
9999	16000000	1024	0.64	1.5625
19999	16000000	1	0.00125	800
19999	16000000	8	0.01	100
19999	16000000	64	0.08	12.5
19999	16000000	256	0.32	3.125
19999	16000000	1024	1.28	0.78125
65535	16000000	1	0.004096	244.140625
65535	16000000	8	0.032768	30.51757813
65535	16000000	64	0.262144	3.814697266
65535	16000000	256	1.048576	0.953674316
65535	16000000	1024	4.194304	0.238418579
999	16000000	8	0.0005	2000

Tabel 4.1 – Determinarea valorii încărcate în registrul OCR1A şi a factorului de scalare al frecvenţei de lucru corespunzătoare timer-ului 1

În vederea obţinerii bazei de timp de o secundă, în tabelul 4.1 s-a realizat cu ajutorul programului Excel o serie de calcule pornind de la câteva valori date pentru registrul OCR1A şi calculând perioada de lucru a timer-ului 1 pentru cele cinci valori ale factorului de scalare puse la dispoziţie de către ATmega2560 (afişate în tabel). S-a ajuns la concluzia că valoarea de o secundă nu poate fi obţinută cu precizia dorită în mod direct, fiind necesară alegerea unei perioade a cărei valori să fie submultiplu de o secundă (în acest caz 10ms) şi repetarea procesului de numărare de o sută de ori pentru a obţine baza de timp specificată anterior. Formulele introduse pentru calculul perioadei (T_timer1), respectiv al numărului de iteraţii al ciclilor de numărare sunt prezentate mai jos.

$$T\_CPU\_scalat = \frac{factor\_de\_scalare}{f\_CPU}$$

$$T\_timer1 = T\_CPU\_scalat * (OCR1A + 1)$$

$$număr\_de\_iteraţii\_al\_ciclilor\_de\_numărare = \frac{1}{T\_timer1}$$

În programul principal au fost definite instrucţiunile ce trebuie exectutate în cadrul rutinei de deservire a întreruperii declanşate de către timer-ul 1 în interiorul blocului denumit “ISR(TIMER1_COMPA_vect)”. În cadrul acestui bloc, se determină cele o sută de iteraţii calculate anterior (stocate în variabila “counted_ten_milliseconds”) şi se actualizează timpul de joc al jucătorului care este la mutare după fiecare secundă, prin apelul funcţiilor “MAX7219_SPI_displayWhiteTime()” şî “MAX7219_SPI_displayBlackTime()”.

4.3) Structura de cod pentru implementarea regulilor jocului de şah, recunoaşterea pieselor (tip şi culoare) şi sugerarea mutărilor posibile ale acestora în conformitate cu aceste reguli

4.3.1) Recunoaşterea tipului de piesă prin procedee software

Diferenţierea pieselor de şah după cele două criterii, tip şi culoare, s-a realizat prin definirea unor valori întregi unice pentru fiecare piesă în parte, utilizând directiva preprocesor "#define". De asemenea, trebuie introdusă o valoare unică şi pentru câmpurile ce rămân libere pe parcursul desfăşurării partidei de şah, toate aceste aspecte fiind prezentate în figura de mai jos:

```
/*----------------------3--------------------*/
/*----------Chess pieces definition----------*/
/*----------Definirea pieselor de sah----------*/

#define NO_PIECE 0

#define WHITE_PAWN 1
#define WHITE_KNIGHT 2
#define WHITE_BISHOP 3
#define WHITE_ROOK 4
#define WHITE_QUEEN 5
#define WHITE_KING 6

#define BLACK_PAWN 11
#define BLACK_KNIGHT 12
#define BLACK_BISHOP 13
#define BLACK_ROOK 14
#define BLACK_QUEEN 15
#define BLACK_KING 16

/*----------Definirea pieselor de sah----------*/
/*----------Chess pieces definition----------*/
/*----------------------3--------------------*/
```

Figura 4.7 – Definirea pieselor de şah prin procedee software

De asemenea, alegerea acestor numere permite uşurarea procesului de identificare al pieselor în cadrul aplicaţiei, prin testarea intervalului [1,6] pentru identificarea pieselor albe, respectiv al intervalului [11, 16] pentru identificarea pieselor negre, intervale folosite atât în funcţiile ce definesc mutările posibile ale pieselor, cât şi în cadrul programului principal.

În continuare, sunt prezentate matricile "initialPiecesPositionMatrix" şi "piecesPositionMatrix" iniţializate în fişierul "chess_game_rules_functions.c" şi utilizate pentru setarea poziţiei iniţiale, respectiv pentru actualizarea poziţiei curente a pieselor de şah pe parcursul desfăşurării partidei. Toate matricile prezentate în continuare au opt linii şi opt coloane, iar legătura dintre acestea se stabileşte pe baza indexării identice a liniilor şi a coloanelor (linia 0 şi coloana 0 în cod corespund informaţiilor referitoare la câmpul numerotat A1 pe tabla de şah, iar linia 7 şi coloana 7 în cod corespund informaţiilor referitoare la câmpul numerotat H8).

```
/* Matrice folosita pentru stocarea pozitiilor initiale ale pieselor. Folosita la inceputul
jocului de sah si dupa resetarea tablei de sah. */
Matrix_uint8_t initialPiecesPositionMatrix =
{
	WHITE_ROOK, WHITE_KNIGHT, WHITE_BISHOP, WHITE_QUEEN, WHITE_KING, WHITE_BISHOP,
WHITE_KNIGHT, WHITE_ROOK,
	WHITE_PAWN, WHITE_PAWN,   WHITE_PAWN,   WHITE_PAWN,  WHITE_PAWN, WHITE_PAWN,
WHITE_PAWN,   WHITE_PAWN,
	NO_PIECE,   NO_PIECE,     NO_PIECE,     NO_PIECE,    NO_PIECE,   NO_PIECE,     NO_PIECE,
NO_PIECE,
	NO_PIECE,   NO_PIECE,     NO_PIECE,     NO_PIECE,    NO_PIECE,   NO_PIECE,     NO_PIECE,
NO_PIECE,
	NO_PIECE,   NO_PIECE,     NO_PIECE,     NO_PIECE,    NO_PIECE,   NO_PIECE,     NO_PIECE,
NO_PIECE,
	NO_PIECE,   NO_PIECE,     NO_PIECE,     NO_PIECE,    NO_PIECE,   NO_PIECE,     NO_PIECE,
NO_PIECE,
	BLACK_PAWN, BLACK_PAWN,   BLACK_PAWN,   BLACK_PAWN,  BLACK_PAWN, BLACK_PAWN,
BLACK_PAWN,   BLACK_PAWN,
	BLACK_ROOK, BLACK_KNIGHT, BLACK_BISHOP, BLACK_QUEEN, BLACK_KING, BLACK_BISHOP,
BLACK_KNIGHT, BLACK_ROOK
};

/* Matrice folosita pentru stocarea pozitiilor curente ale pieselor. Aceasta matrice permite
diferentierea tipului de piesa. */
Matrix_uint8_t piecesPositionMatrix =
{
	WHITE_ROOK, WHITE_KNIGHT, WHITE_BISHOP, WHITE_QUEEN, WHITE_KING, WHITE_BISHOP,
WHITE_KNIGHT, WHITE_ROOK,
	WHITE_PAWN, WHITE_PAWN,   WHITE_PAWN,   WHITE_PAWN,  WHITE_PAWN, WHITE_PAWN,
WHITE_PAWN,   WHITE_PAWN,
	NO_PIECE,   NO_PIECE,     NO_PIECE,     NO_PIECE,    NO_PIECE,   NO_PIECE,     NO_PIECE,
NO_PIECE,
	NO_PIECE,   NO_PIECE,     NO_PIECE,     NO_PIECE,    NO_PIECE,   NO_PIECE,     NO_PIECE,
NO_PIECE,
	NO_PIECE,   NO_PIECE,     NO_PIECE,     NO_PIECE,    NO_PIECE,   NO_PIECE,     NO_PIECE,
NO_PIECE,
	NO_PIECE,   NO_PIECE,     NO_PIECE,     NO_PIECE,    NO_PIECE,   NO_PIECE,     NO_PIECE,
NO_PIECE,
	BLACK_PAWN, BLACK_PAWN,   BLACK_PAWN,   BLACK_PAWN,  BLACK_PAWN, BLACK_PAWN,
BLACK_PAWN,   BLACK_PAWN,
	BLACK_ROOK, BLACK_KNIGHT, BLACK_BISHOP, BLACK_QUEEN, BLACK_KING, BLACK_BISHOP,
BLACK_KNIGHT, BLACK_ROOK
};
```

Figura 4.8 – Matrici pentru iniţializarea şi urmărirea poziţiei pieselor de şah

4.3.2) Prezentarea funcţiilor pentru prelucrarea datelor (operaţii matriceale) şi afişarea mutărilor posibile în funcţie de piesa selectată

În acest subcapitol se evidenţiază funcţiile realizate pentru a defini câmpurile corespunzătoare mutărilor celor şase tipuri de piese de şah (pion, cal, nebun, turn, damă, rege) precum şi variabilele implicate în acest proces. Astfel, funcţiile discutate în continuare sunt cele din figura 4.9.

```
/* Functie de detectare (valori stocate in currentPiecePossibleMove si newLedMatrix) a campurilor
posibile pe care poate fi mutat un pion. */
void setLedsForPossibleMovesPawn(uint8_t local_currentPieceMove, uint8_t local_currentPieceRow,
uint8_t local_currentPieceColumn);

/* Functie de detectare (valori stocate in currentPiecePossibleMove si newLedMatrix) a campurilor
posibile pe care poate fi mutat un cal. */
void setLedsForPossibleMovesKnight(uint8_t local_currentPieceMove, uint8_t local_currentPieceRow,
uint8_t local_currentPieceColumn);

/* Functie de detectare (valori stocate in currentPiecePossibleMove si newLedMatrix) a campurilor
posibile pe care poate fi mutat un nebun. */
extern void setLedsForPossibleMovesBishop(uint8_t local_currentPieceMove, uint8_t
local_currentPieceRow, uint8_t local_currentPieceColumn);

/* Functie de detectare (valori stocate in currentPiecePossibleMove si newLedMatrix) a campurilor
posibile pe care poate fi mutat un turn. */
void setLedsForPossibleMovesRook(uint8_t local_currentPieceMove, uint8_t local_currentPieceRow,
uint8_t local_currentPieceColumn, uint8_t local_whiteCastleInProgress, uint8_t
local_blackCastleInProgress);

/* Functie de detectare (valori stocate in currentPiecePossibleMove si newLedMatrix) a campurilor
posibile pe care poate fi mutata o dama. */
void setLedsForPossibleMovesQueen(uint8_t local_currentPieceMove, uint8_t local_currentPieceRow,
uint8_t local_currentPieceColumn);

/* Functie de detectare (valori stocate in currentPiecePossibleMove si newLedMatrix) a campurilor
posibile pe care poate fi mutat un rege. */
void setLedsForPossibleMovesKing(uint8_t local_currentPieceMove, uint8_t local_currentPieceRow,
uint8_t local_currentPieceColumn, uint8_t local_isCurrentKing);
```

Figura 4.9 – Funcţii pentru definirea mutărilor corespunzătoare celor şase tipuri de piese de şah

Principalele variabile folosite în aceste funcţii sunt matricile “currentPiecePossibleMove” şi “newLedMatrix” care indică câmpul curent al piesei ridicate, precum şi câmpurile legale unde poate fi mutată piesa, respectiv care determină câmpurile ce trebuie aprinse de către LED-uri, valorile stocate în aceste variabile fiind: “1” pentru câmpuri posibile, “0” pentru câmpuri ilegale şi “103” pentru câmpul curent al piesei ridicate.

Deoarece în cazul pieselor negre se foloseşte o descriere similară, diferenţele principale fiind valorile câmpurilor de plecare şi de sosire, precum şi inversarea culorilor, în continuare se va pune accentul pe evidenţierea funcţiilor din figura 4.9 în cazul pieselor albe.

Câmpurile posibile la mutarea unui pion

Conform regulilor jocului de şah, pionul poate muta doar câte un câmp ”înainte” (către piesele adverse), neavând posibilitatea de a se întoarce pe câmpurile anterioare. Pionul poate captura piesele oponentului prin înaintarea pe diagonală cu un câmp. Două aspecte se abat de la regulile prezentate anterior: singura situaţie în care un pion poate muta două câmpuri înainte în loc de unul este aceea în care pionul se găseşte pe poziţia initială şi un pion amplasat pe ultimul câmp din tabăra adversă se “promovează / transformă” în una din cele patru categorii de piese (cal, nebun, damă, turn) la alegerea jucătorului.

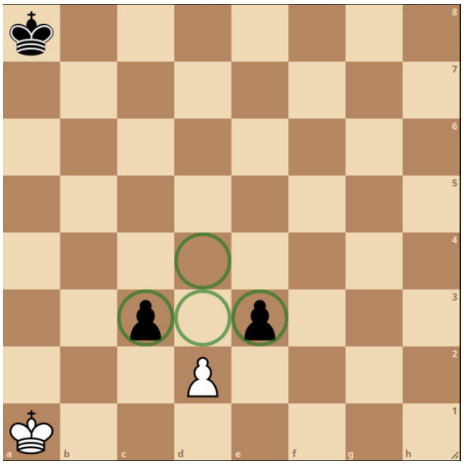


Figura 4.10 – Câmpuri legale pentru pionul alb aflat pe poziţia inițială

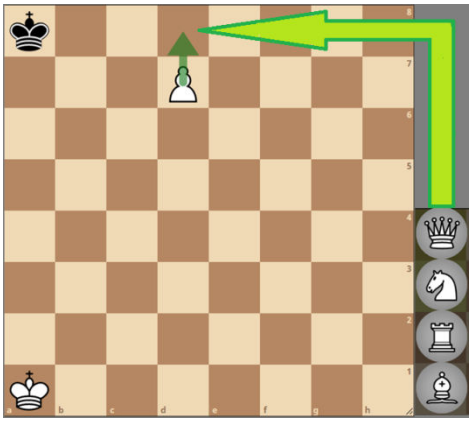


Figura 4.11 – Promovarea pionului alb

Toate tipurile de mutări prezentate anterior sunt implementate cu ajutorul funcţiei “setLedsForPossibleMovesPawn()” ce primeşte ca parametrii: valoarea de identificare a piesei curente, linia şi coloana pe care se găseşte această piesă. Funcţia verifică dacă piesa curentă este un pion alb sau negru (existând mici diferenţe de implementare în funcţie de culoarea acestuia) şi setează câmpurile legale unde poate fi mutat pionul (prezentate anterior) în matricile “currentPiecePossibleMove” şi “newLedMatrix”, pornind de la linia şi coloana corespunzătoare poziţiei curente a piesei. Principala diferenţă dintre valorile matricilor “currentPiecePossibleMove” şi “newLedMatrix” este că prima matrice menţionată setează valoarea “1” şi pe alte câmpuri suplimentare, pentru a indica posibilitatea de capturare a pieselor adverse şi de protejare a pieselor proprii, în cea de-a doua matrice fiind setate valorile “1” strict pe câmpurile aprinse de către LED-uri.

Promovarea pionului este implementată în cadrul programului principal, prin testarea liniei curente pe care se poziţionează pionul, iar dacă aceasta corespunde cu prima linie (linia 0 în cod) pentru pionul negru, respectiv ultima linie (linia 7 în cod) pentru pionul alb se asteaptă în interiorul unui bloc “do-while” apăsarea unui buton în conformitate cu piesa care va înlocui pionul, funcţionalităţile celor patru butoane fiind prezentate în tabelul 3.2, iar structurile decizionale executate

în interiorul blocului “do-while” sunt evidenţiate în figura 4.12 în cazul turnului şi al damei, fiind în mod similar scrise şi cele pentru nebun şi cal. Etichetele “BUTTON_ROOK_PRESSED”, respectiv “BUTTON_QUEEN_PRESSED” conţin valorile citite direct de la cele două butoane (ce trimit valori “0” logic la acţionarea acestora).

```
if (BUTTON_ROOK_PRESSED) /* Read Rook promote button. */
{

        piecesPositionMatrix.values[currentPieceRow][currentPieceColumn] = NO_PIECE;
        if (currentPieceMove == WHITE_PAWN)
        {

                piecesPositionMatrix.values[i][j] = WHITE_ROOK; /* Promote white pawn to
rook.*/
        }
        if (currentPieceMove == BLACK_PAWN)
        {

                piecesPositionMatrix.values[i][j] = BLACK_ROOK; /* Promote black pawn to rook. */
        }
        break;
}
if (BUTTON_QUEEN_PRESSED) /* Read Queen promote button. */
{

        piecesPositionMatrix.values[currentPieceRow][currentPieceColumn] = NO_PIECE;
        if (currentPieceMove == WHITE_PAWN)
        {

                piecesPositionMatrix.values[i][j] = WHITE_QUEEN; /* Promote white pawn to queen.*/
        }
        if (currentPieceMove == BLACK_PAWN)
        {

        piecesPositionMatrix.values[i][j] = BLACK_QUEEN; /* Promote black pawn to queen. */
        }
        break;
}
```

Figura 4.12 – Cod pentru promovarea pionului în turn, respectiv damă

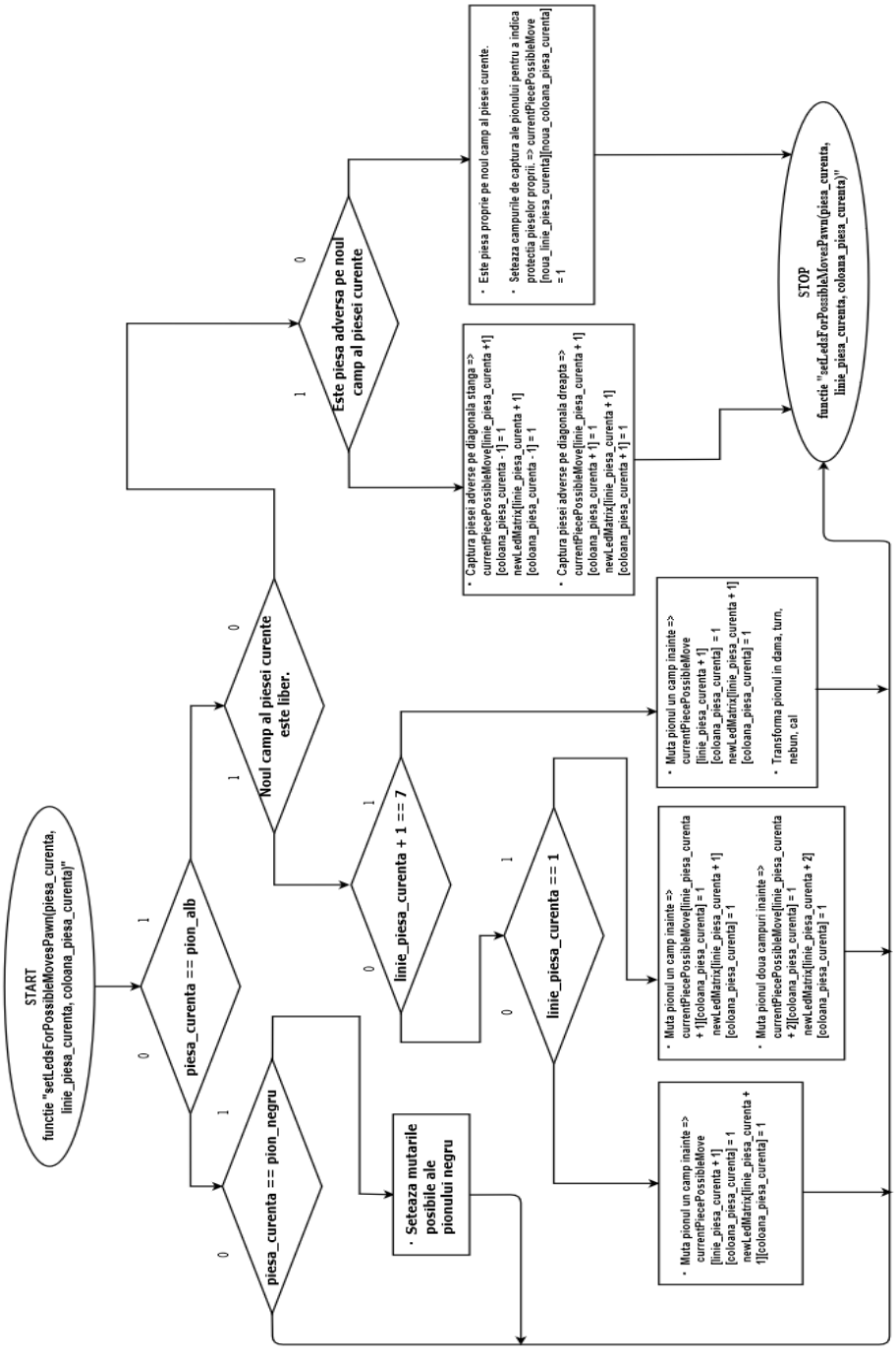


Figura 4.13 – Organigrama funcţiei “setLedsForPossibleMovesPawn()”

Câmpurile posibile la mutarea unui cal

Calul reprezintă o piesă specială în jocul de şah, deoarece această piesă poate muta numai în forma literei “L”. Datorită acestei proprietăţi, valoarea acestei piese variază în funcţie de poziţia unde este amplasată. Astfel, un cal situat la marginea tablei, va avea la dispoziţie doar două câmpuri permise, iar în centrul tablei, numărul de câmpuri legale este opt, reprezentând şi numărul maxim de câmpuri ce pot fi acoperite de către un cal.

Astfel, conform paragrafului anterior, realizarea funcţiei “setLedsForPossibleMovesKnight()” porneşte de la premisa că, indiferent de câmpul de poziţionare al calului, există opt cazuri ce trebuie acoperite, acestea fiind implicit eliminate dacă numărul liniei sau al coloanei depăşeşte dimensiunile maxime ale unei matrici 8 x 8. Deoarece, cele opt cazuri diferă semnificativ, ele trebuie rezolvate individual, nefiind posibilă înglobarea acestora în cadrul unei structuri repetitive “for”. Ca şi în cazul funcţiei anterioare, valorile celor opt câmpuri sunt determinate pe baza poziţiei curente a calului prin setarea valorii “1” în matricile “currentPiecePossibleMove” şi “newLedMatrix”.

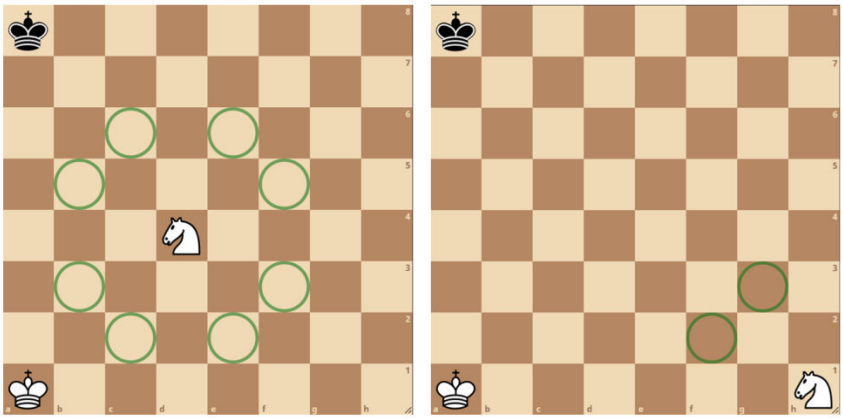


Figura 4.14 – Comparaţia câmpurilor posibile între un cal situat în centrul tablei şi unul situat la marginea tablei

Câmpurile posibile la mutarea unui nebun

Nebunul reprezintă prima piesă ce poate fi mutată cu un număr de câmpuri oricât de mare, dar doar pe diagonală. Astfel, acesta reprezintă primul caz în care se pune problema întâlnirii unei piese proprii sau adverse situate pe diagonala nebunului, piese care întrerup raza de acţiune a acestuia, deoarece nicio piesă (excepţie fiind calul) nu poate sări peste piesele proprii sau peste cele ale oponentului.

Funcţia “setLedsForPossibleMovesBishop()” este cea care implementează regulile jocului de şah în cazul nebunului, pornind de la concepte prezentate anterior, cum ar fi: validarea piesei curente ca fiind nebun alb sau negru şi setarea câmpurilor de joc specifice nebunului în matricile “currentPiecePossibleMove” şi “newLedMatrix” în scopul afişării acestor câmpuri prin aprinderea LED-urilor WS2812C.

În cazul acestei piese, mutările posibile se împart în patru cazuri (câte două pentru fiecare diagonală), pornind de la poziţia unui nebun situat în centrul tablei, Astfel, pentru fiecare caz se introduce câte o structură repetitive “for”, punctul de plecare în cele patru cazuri fiind poziţia curentă a nebunului. Pentru a scrie toate structurile repetitive, se observă modificările liniei şi a coloanei pentru fiecare jumătate de diagonală disponibilă. Astfel, mutând nebunul cu un singur câmp, se observă următoarele modificări faţă de poziţia curentă a acestuia:

- Cazul 1: se decrementează atât numărul liniei curente, cât şi al coloanei curente;
- Cazul 2: se decrementează numărul liniei curente şi se incrementează numărul coloanei curente;
- Cazul 3: se incrementează numărul liniei curente şi se decrementează numărul coloanei curente;
- Cazul 4: se incrementează atât numărul liniei curente, cât şi al coloanei curente.

În figura de mai jos, sunt prezentate câmpurile posibile ale nebunului cu culoare verde, iar cele ilegale cu culoare roşie.

Figura 4.15 – Implementarea regulilor pentru mutarea nebunului

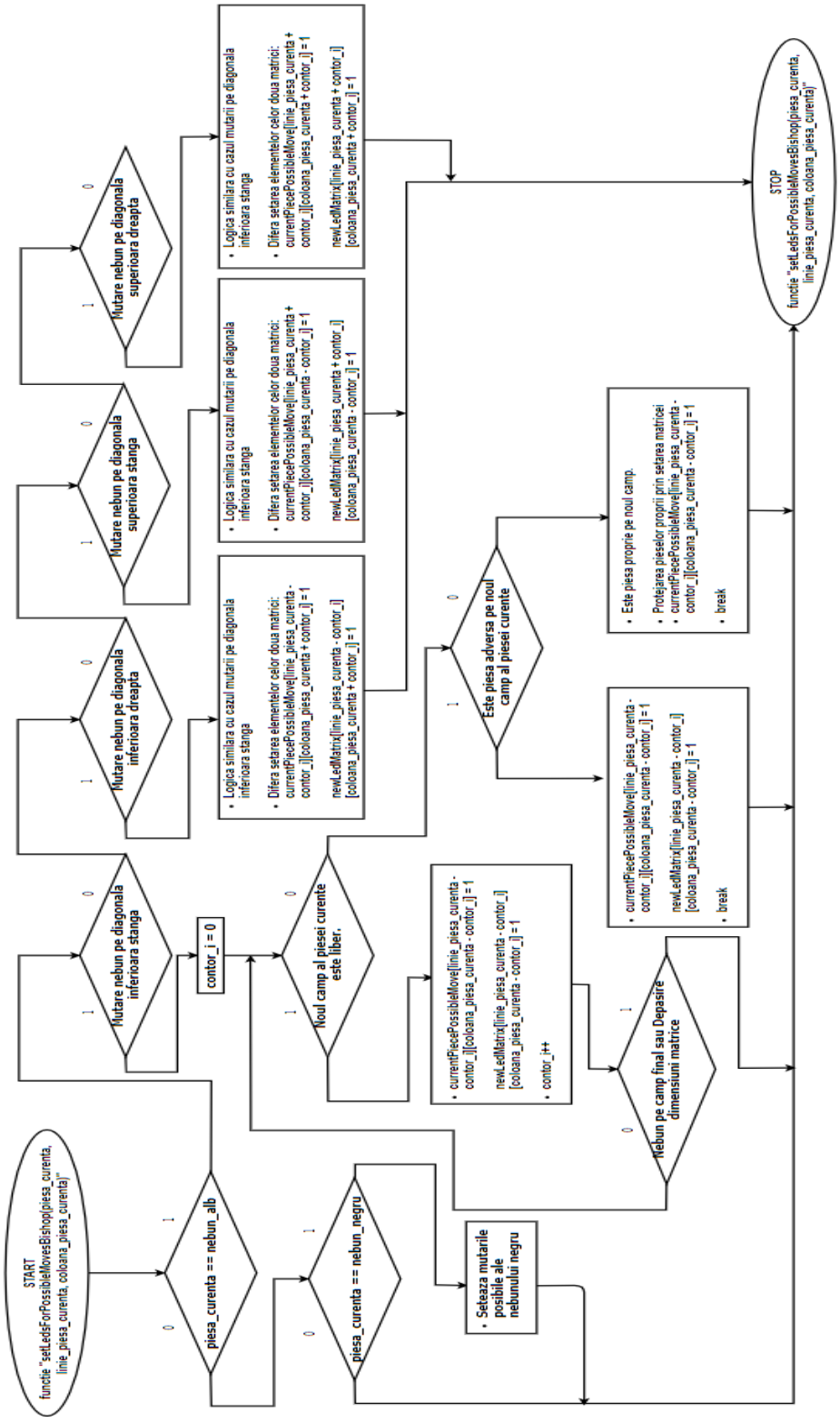


Figura 4.16 – Organigrama funcției “setLedsForPossibleMovesBishop()”

Câmpurile posibile la mutarea unui turn

Turnul este a doua piesă capabilă într-o singură mutare să ajungă de la un capăt la celălalt al tablei de şah, prin mutarea pe verticală, respectiv pe orizontală a acestuia.

Funcţia responsabilă pentru indicarea câmpurilor posibile de deplasare ale unui turn este “setLedsForPossibleMovesRook()”. Spre deosebire de funcţiile prezentate anterior, se observă prezenţa a doi parametrii de intrare suplimentari, “whiteCastleInProgress” şi “blackCastleInProgress” folosiţi pentru implementarea rocadei, o mişcare specifică jocului de şah ce va fi descrisă în secţiunea de mutări speciale din acest capitol.

În cadrul funcţiei, pornind de la câmpul curent al turnului, mutările posibile ale acestei piese se impart de asemenea în patru cazuri implementate cu ajutorul structurii repetitive “for”:

- Cazul 1: se decrementează numărul liniei curente, păstrând numărul coloanei curente constant;
- Cazul 2: se incrementează numărul liniei curente, păstrând numărul coloanei curente constant;
- Cazul 3: se păstrează numărul liniei curente constant şi se decrementează numărul coloanei curente;
- Cazul 4: se păstrează numărul liniei curente constant şi se incrementează numărul coloanei curente.

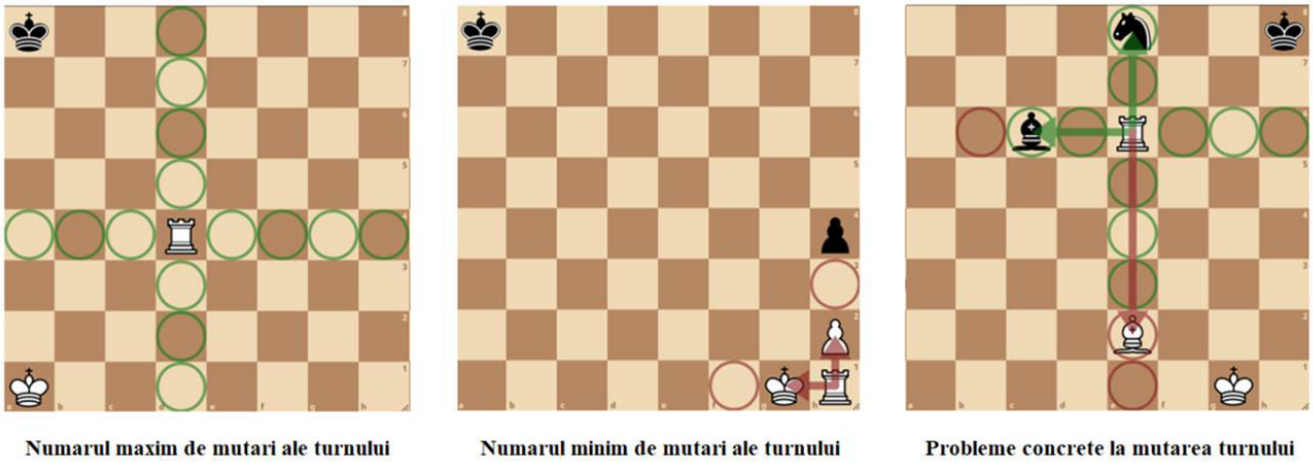


Figura 4.17 – Implementarea regulilor pentru mutarea turnului

Câmpurile posibile la mutarea unui dame

Dama este cea mai puternică piesă, deoarece are cea mai mare libertate de mişcare pe câmpurile tablei de şah, având posibilitatea de a se deplasa pe un număr oricât de mare de câmpuri pe verticală, pe orizontală şi pe diagonală.

Situată în centrul tablei şi în lipsa altor piese, dama are o rază de acţiune ce acoperă aproape jumătate din câmpurile totale ale tablei de şah, iar prin natura tipurilor de mutări ale acesteia, trebuie rezolvate atât probleme întâlnite în cazul turnului, cât şi cele evidenţiate în cazul nebunului.

Pornind de la afirmaţia că, în cazul unei dame, câmpurile posibile de joc ale acesteia reprezintă o îmbinare a câmpurilor posibile de joc pentru un turn şi cele ale unui nebun, se implementează funcţia "setLedsForPossibleMovesQueen()" a cărei rol este să determine numărul de câmpuri valide ale damei prin apelarea funcţiilor "setLedsForPossibleMovesRook()" şi "setLedsForPossibleMovesBishop()" prezentate anterior în cadrul acestui capitol. Astfel, se actualizează matricile definite ca variabile globale, "currentPiecePossibleMove" şi "newLedMatrix" prin setarea atât a valorilor obţinute în cazul unui turn situat pe poziţia curentă a damei, cât şi a celor corespunzătoare unui nebun.

Figura 4.18 – Implementarea regulilor pentru mutarea damei

Câmpurile posibile la mutarea regelui

Regele reprezintă cea mai importantă piesă de pe tabla de şah, fiind o piesă unică atât în tabăra albului, cât şi în cea a negrului. Astfel, scopul unei partide de şah constă în atacarea regelui advers şi eliminarea tuturor mutărilor posibile ale acestuia, protejând în acelaşi timp regele propriu de ameninţările oponentului.

Până în acest moment au fost prezentate principalele probleme ce pot apărea la definirea tipurilor de mutări posibile ale pieselor de şah, printre acestea fiind amintite: coliziunea piesei curente cu celelalte piese, implementarea unui mecanism care să asigure protejarea celorlalte piese aliate, capturarea pieselor oponente şi evitarea unui posibil salt al piesei curente peste celelalte piese. În cadrul implementării regulilor jocului de şah, în cazul regelui se regăsesc toate aspectele prezentate anterior, dar se introduc şi alte probleme specifice acestei piese, cum ar fi: restricţionarea câmpurilor posibile pentru amplasarea regelui din cauza ameninţărilor de "şah" ale pieselor advers, forţarea mutării regelui, capturarea piesei atacatoare sau introducerea unei piese proprii în calea piesei atacatoare în situaţia în

care câmpul curent al regelui este ameninţat, detectarea matului ce defineşte finalizarea partidei, testarea posibilităţii de realizare a rocadei.

Astfel, deoarece descrierea regulilor de şah aplicate în cazul regelui este un proces complex, majoritatea problemelor de mai sus sunt rezolvate prin implementarea funcţiilor “setLedsForPossible MovesKing()”, “Matrix_uint8_t_operation_substract()” şi “eliminatePossibleChecksForCurrent King()”.

Funcţia “setLedsForPossibleMovesKing()” defineşte toate mutările ce pot fi efectuate în cazul regelui, pornind de la poziţia curentă a acestuia (determinată de numărul liniei curente, respectiv de cel al coloanei curente) şi actualizând valorile matricilor “currentPiecePossibleMove” şi “newLedMatrix”. În cazul regelui, se pot distinge nouă câmpuri posibile (inclusiv câmpul curent), prin incrementarea/decrementarea liniei curente, a coloanei curente sau a ambelor dimensiuni ale matricii 8 x 8 corespunzătoare tablei de şah.

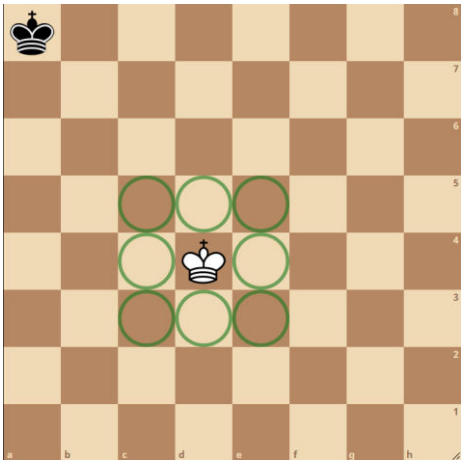
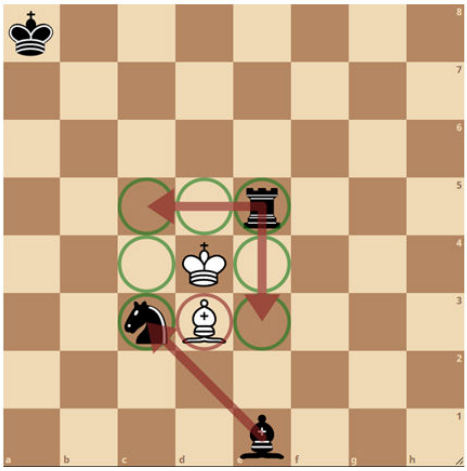
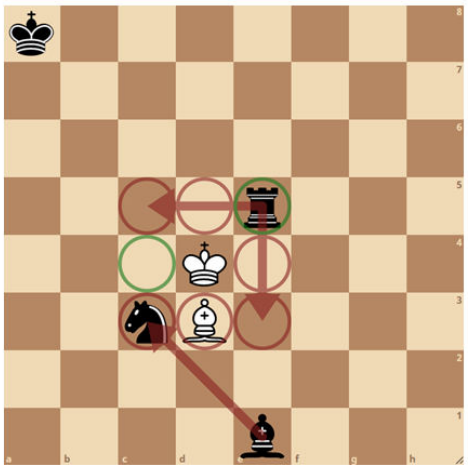


Figura 4.19 - Mutările regelui alb implementate cu funcţia “setLedsForPossibleMovesKing()”

În cadrul funcţiei “eliminatePossibleChecksForCurrentKing()” se definesc trei matrici care stochează câmpurile posibile de joc ale: regelui care nu tine cont de ameninţările de şah (matricea “currentKingPossibleMoves_withoutChecks”, obţinută prin apelarea funcţiei “setLedsForPossible MovesKing()”), regelui care tine cont de ameninţările de şah (matricea “currentKingPossibleMoves_withChecks”) şi ale piesei adverse testate la iteraţia curentă (“opositePiecePossibleMoves”). Astfel, prin folosirea a două structuri repetitive “for” imbricate şi prin aplicarea înmulţirii element cu element a matricilor “currentKingPossibleMoves_withoutChecks” şi “opositePiecePossibleMoves”, se identifică toate piesele adverse care atacă unul sau mai multe dintre câmpurile posibile de joc ale regelui ce se doreşte a fi mutat în poziţia curentă şi se elimină printr-o operaţie de scădere element cu element aceste câmpuri din matricea regelui care ţine cont de ameninţările de şah ale piesei oponente (şi care este iniţial setată cu valorile matricei “currentKing PossibleMoves_withoutChecks”). Cele două operaţii matriceale sunt implementate în cadrul funcţiei “Matrix_uint8_t_operation_substract()”, funcţia “eliminatePossibleChecksForCurrentKing()” returnând valorile matricei “currentKingPossibleMoves_withChecks” după testarea tuturor pieselor adverse.



Mutarile regelui alb dupa apelarea functiei "setLedsForPossibleMovesKing()"



Mutarile regelui alb dupa apelarea functiei "eliminatePossibleChecksForCurrentKing()"

Figura 4.20 – Comparaţie între funcţiile “setLedsForPossibleMovesKing()” şi “eliminatePossibleChecksForCurrentKing()” în cazul regelui alb

Din figura 4.20, se observă că doar prin apelarea funcţiei “setLedsForPossibleMovesKing()”, regele alb are la dispoziţie cinci câmpuri atacate de către piesele oponentului (marcate cu săgeţi roşii). Această eroare se corectează prin apelarea funcţiei “eliminatePossibleChecksForCurrentKing()”, câmpurile posibile de mutare ale regelui alb fiind în acest caz în conformitate cu regulile jocului de şah.

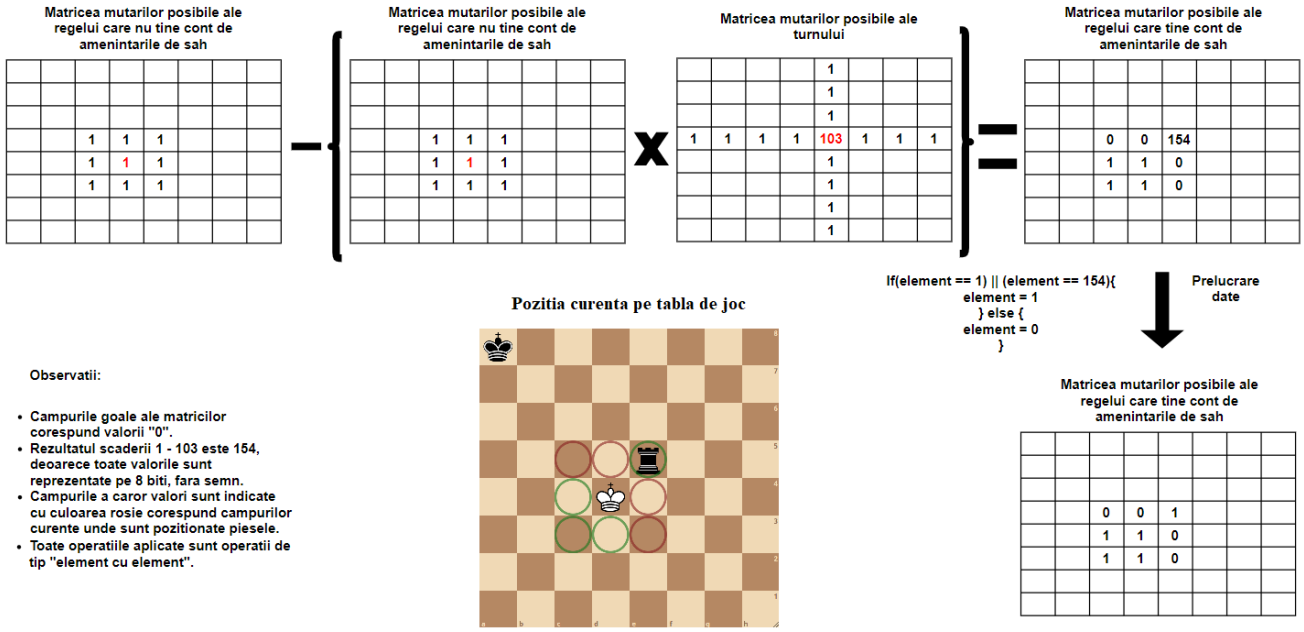


Figura 4.21 – Operaţiile aplicate pentru obţinerea matricei regelui care ţine cont de ameninţările de şah

Mutare specială în partida de şah: rocada

Rocada reprezintă o mutare compusă şi implică mutarea regelui cu două câmpuri în direcţia în care se doreşte efectuarea rocadei şi plasarea turnului lângă rege, conform figurii 4.22.

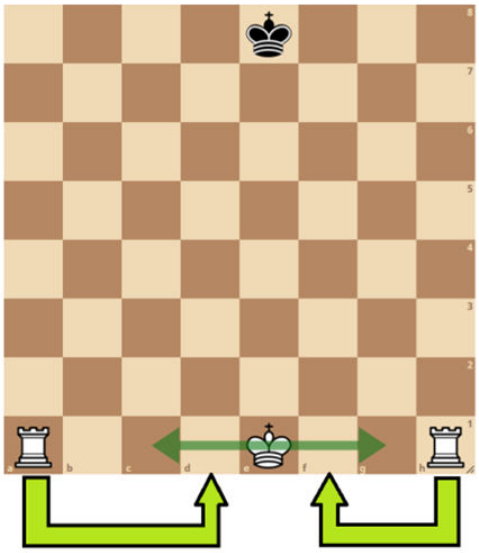


Figura 4.22 – Mutări pentru efectuarea rocadei

Pentru implementarea rocadei, trebuie urmărite câteva aspecte importante în conformitate cu regulile jocului de şah:

- rocada nu poate fi efectuată dacă atât câmpul curent al regelui, cât şi celelalte două câmpuri parcurse de acesta sunt atacate de către piesele adverse;
- orice mutare a regelui de pe câmpul iniţial duce la imposibilitatea realizării rocadei pe tot parcursul partidei;
- orice mutare a unui turn de pe poziţia iniţială duce la imposibilitatea realizării rocadei în direcţia acelui turn.

Prima regulă este implicit rezolvată prin implementarea funcţiilor care definesc câmpurile posibile de mutare ale regelui, descrise anterior. Celelalte două reguli sunt asigurate prin introducerea variabilelor “first_white_king_move / first_black_king_move” (stochează valoarea “1” dacă regele nu a fost deplasat de pe câmpul iniţial şi valoarea “0” în caz contrar), “first_a_column_white_rook_move / first_a_column_black_rook_move” (păstrează valoarea “1” cât timp turnurile de pe coloana A nu au fost mutate), “first_h_column_white_rook_move / first_h_column_black_rook_move” (similar cu variabila anterioară, dar se ocupă de turnurile de pe coloana H).

La mutarea regelui cu două câmpuri, nefiind în setul de mutări obişnuite ale acestuia, programul decide prin intermediul unor structuri decizionale “if-else” că urmează realizarea rocadei si setează variabila “whiteCastleInProgress / blackCastleInProgress”, iar poziţia regelui după mutarea acestuia indică tipul de rocadă efectuat. Astfel, tot partea care a mutat regele va fi la mutare şi va avea ca unică

mutare corectă plasarea turnului pe poziţia care finalizează efectuarea rocadei, prin trimiterea variabilei “whiteCastleInProgress / blackCastleInProgress” ca parametru de intrare al funcţiei “setLedsFor PossibleMoveRook()”.

Problema detectării matului

Matul reprezintă acea situaţie dintr-o partidă de şah în care unul din regi este atacat de către o piesă adversă, iar acesta nu poate muta pe niciun alt câmp, nicio piesă proprie nu poate proteja ameninţarea piesei adverse şi piesa ameninţătoare nu poate fi capturată. Astfel, în urma ameninţării de mat se încheie partida de şah.

Fiind implicate foarte multe variabile în algoritmul de detectare a matului, acesta a fost implementat direct în programul principal, urmărind organigrama din figura 4.23.

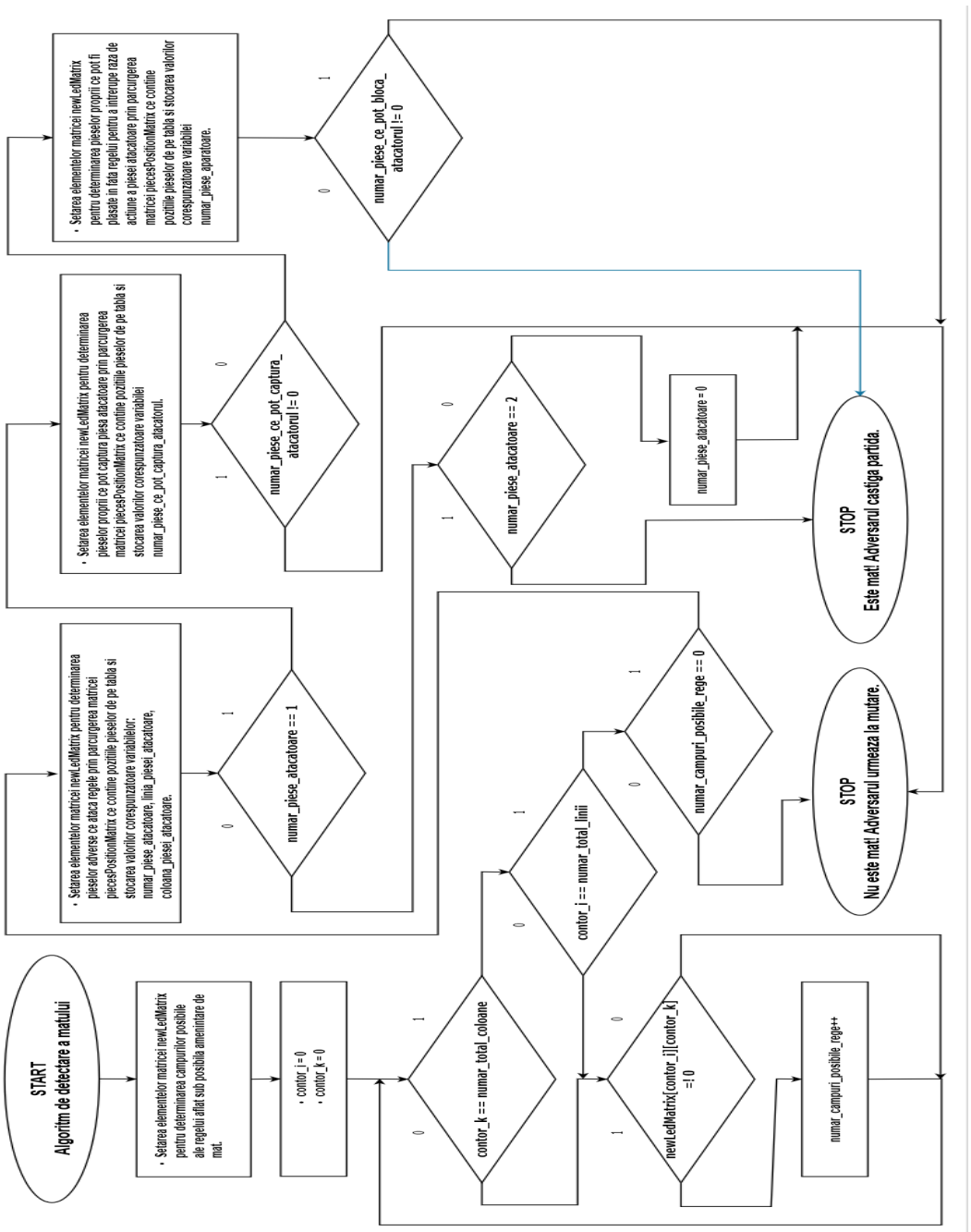


Figura 4.23 – Organigrama algoritmului de detectare a matului

.

Capitolul 5 - Structura fizică a tablei de şah

5.1) Proiectarea mecanică a tablei de şah fizice care integrează sistemul incorporat

5.1.1) Determinarea dimensiunilor componentelor electronice şi poziţionarea acestora în interiorul tablei fizice

Prima etapă în procesul de proiectare mecanică a tablei de şah constă în stabilirea dimensiunilor corespunzătoare componentelor electronice alese în cadrul proiectului şi prezentate în capitolul 3. Astfel, prin măsurarea cu ajutorul şublerului şi prin consultarea foilor de catalog (acolo unde sunt specificate dimensiunile), s-au obţinut următoarele dimensiuni fizice ale componentelor:

- placa de dezvoltare cu microcontroler ATmega2560: lungime = 110mm, lăţime = 55mm, înălţime = 15mm; dimensiunile portului USB: lungime = 17.5mm, lăţime = 12mm, înălţime = 15mm;
- capsula senzorului magnetic cu efect Hall TLE4905: lungime = 5mm, lăţime = 1mm, înălţime = 4mm; înălţimea totală, luând în considerare înălţimea capsulei şi cea a terminalului cel mai lung este de 20mm;
- capsula registrului de deplasare SN74HC165N: lungime = 20mm, lăţime = 10mm, înălţime = 10mm;
- afişajul cu şapte segmente şi opt digiţi şi PCB-ul ce alcătuiesc modulul MAX7219: lungime = 60mm, lăţime = 15mm, înălţime = 10mm;
- dimensiunile LED-urilor SMD WS2812C sunt foarte mici şi datorită poziţiei acestora, ele nu influenţează determinarea cotelor tablei de şah;
- dimensiunile butoanelor: diametru = 18mm, înălţime = 25mm;
- secţiunea cablului de alimentare: diametru = 20mm.

Următoarea etapă presupune încadrarea componentelor electronice în interiorul structurii fizice ce trebuie realizate, ţinând cont de multe aspecte, printre care se amintesc: modul de interconectare al dispozitivelor, dimensiunile fizice ale acestora, protejarea componentelor sensibile la şocuri mecanice şi aspectul produsului final din perspectiva utilizatorului. Proiectarea tablei de şah inteligente s-a realizat cu ajutorul mediului de proiectare 3D FreeCAD, plasarea componentelor putând fi observată în figurile 5.1, 5.2 şi 5.3.

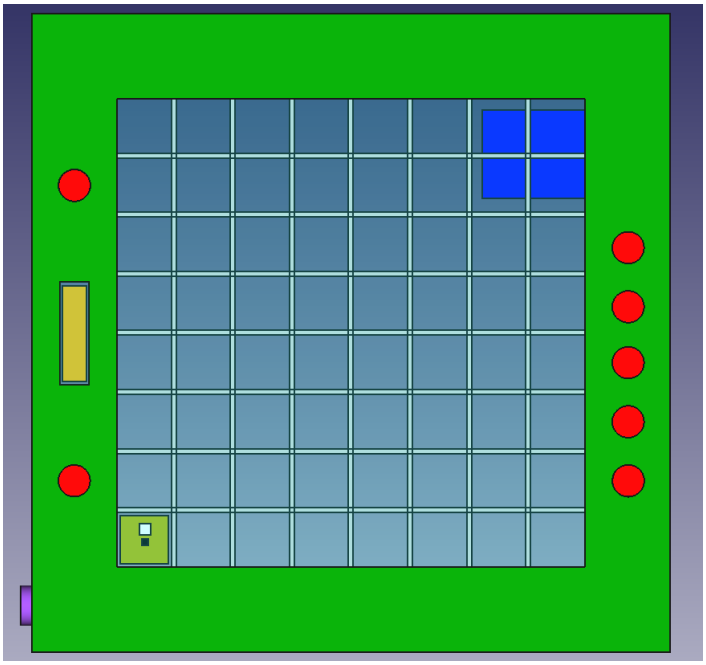


Figura 5.1 – Tabla de şah proiectată în mediul FreeCAD – vedere din faţă

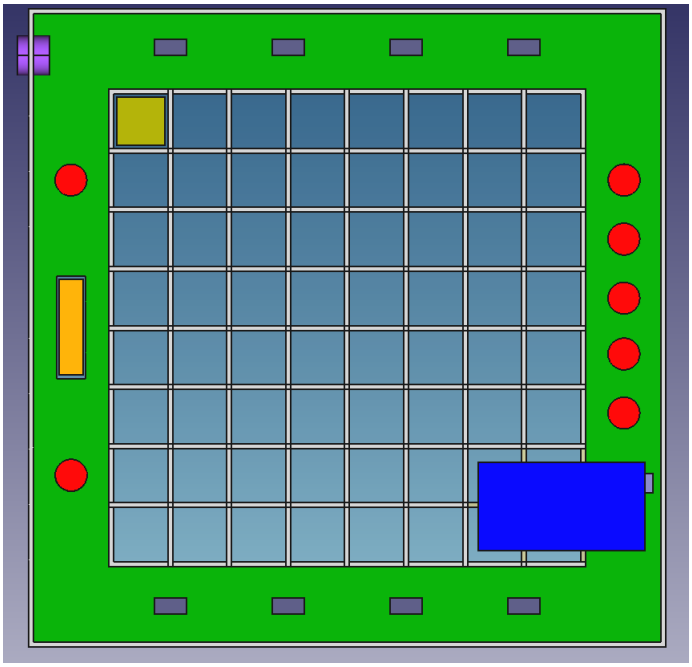


Figura 5.2 – Tabla de şah proiectată în mediul FreeCAD – vedere din spate

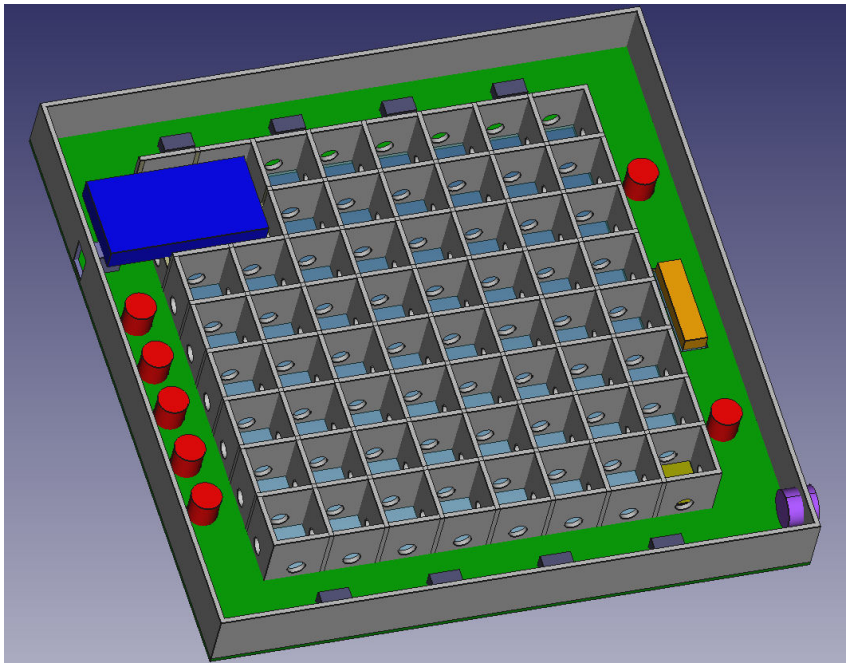


Figura 5.3 – Tabla de şah proiectată în mediul FreeCAD – vedere laterală

Astfel, pentru modelarea componentelor în cadrul mediului de proiectare FreeCAD s-au folosit cilindre pentru reprezentarea butoanelor şi a conectorului pentru cablul de alimentare, precum şi paralelipipede dreptunghice pentru vizualizarea locurilor ocupate de către placa de dezvoltare (împreună cu conectorul USB), registrele de deplasare, şi modulul MAX7219

5.1.2) Stabilirea cotelor tablei de şah

Pornind de la figurile 5.1, 5.2 şi 5.3, în care este reprezentată tabla de şah deja proiectată din punct de vedere mecanic, se vor reprezenta componentele din care este alcătuită structura fizică a tablei de şah, vizualizând şi cotele alese pe baza dimensiunilor fizice ale dispozitivelor electronice.

Prima componentă a tablei fizice este o structură sub formă de fagure ce delimitează cele şaizeci şi patru de câmpuri de joc şi opturează trecerea radiaţiei optice emise de către LED-urile destinate iluminării unor câmpuri pe câmpurile vecine, lucru care ar putea duce la afişarea unor culori eronate din punct de vedere al convenţiei de culori stabilite în tabelul 3.4. Secţiunea circulară realizată în cadrul fiecărui câmp este destinată firelor de legătură folosite la interconectarea atât a componentelor electronice din interiorul fagurelui, cât şi a acestora cu dispozitivele din exterior. Dimensiunile alese asigură plasarea unei plăci de perfoboard echipate cu un senzor TLE4905, un LED WS2812C şi cu componentele pasive ce asigură funcţionalitatea corectă a acestora, iar decuparea celor patru câmpuri de joc vizualizată în figura 5.3 şi cu cotele prezentate în figura 5.4 permite introducerea plăcii de dezvoltare în interiorul tablei de şah.

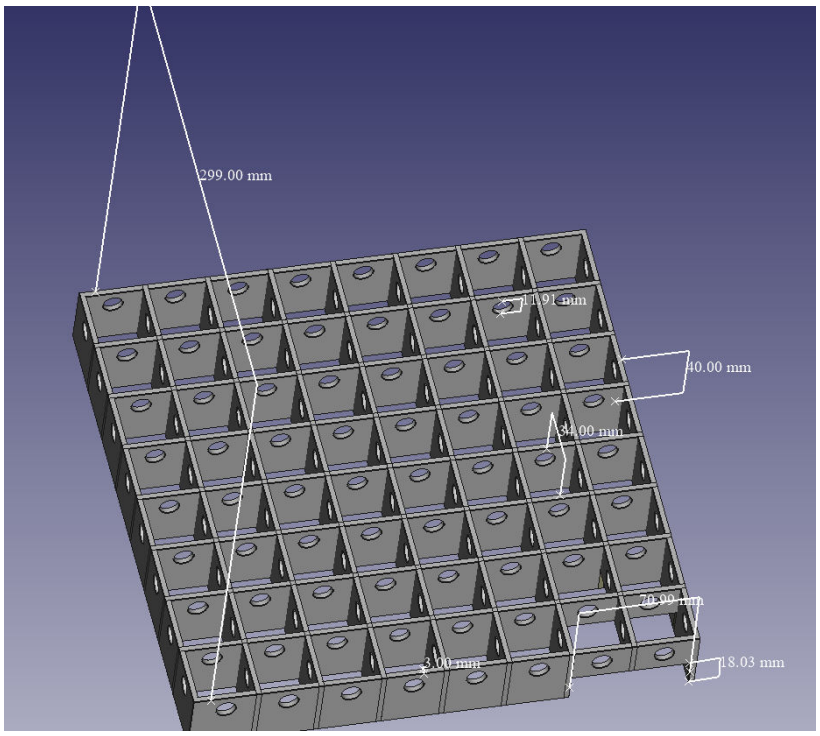


Figura 5.4 – Vizualizarea cotelor pentru structura de fagure

Următoarea componentă stabileşte perimetrul tablei de şah, spaţiul interior tablei de şah şi exterior fagurelui şi dimensiunile găurilor în care se va conecta sursa de alimentare, respectiv cablul USB pentru programarea plăcii de dezvoltare (pentru depanare şi îmbunătăţire a firmware-ului).

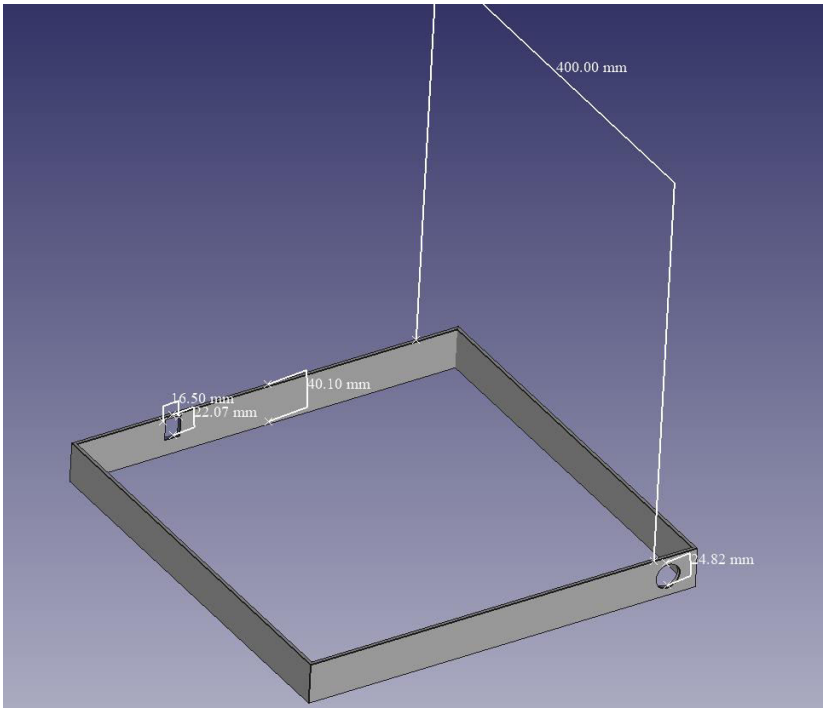


Figura 5.5 – Vizualizarea cotelor pentru structura ce delimitează perimetrul tablei de şah

Ultima componentă importantă este reprezentată de structura transparentă din plexiglas ce defineşte suprafaţa tablei de şah şi permite vizualizarea câmpurilor aprinse de LED-uri de către utilizator, porţiunea reprezentată cu verde închis, fiind spaţiul interior în care sunt poziţionate registrele de deplasare. La determinarea cotelor din figura 5.6 se ţine cont atât de dimensiunile structurilor prezentate anterior, cât şi de dimensiunile butoanelor şi ale ceasului de şah integrate în tablă.

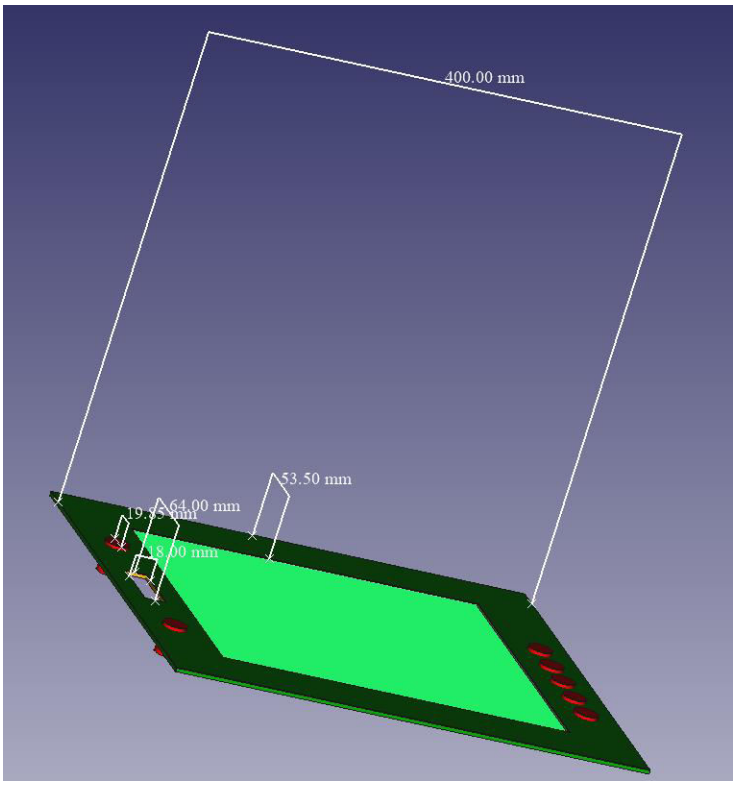


Figura 5.6 – Vizualizarea cotelor pentru determinarea suprafeţei tablei de şah

5.2) Prezentare sumară a etapelor de tehnologie în succesiunea impusă de proiectare şi vizualizarea produsului final

Realizarea efectivă a construcţiei fizice a tablei de şah inteligente impune alegerea următoarelor materiale şi scule:

MATERIALE:

- Placă din plastic alb mat de tip ABS;
- Placă din plexiglas transparent;
- Două tipuri de autocolant alb şi fumuriu semitransparent;
- Fludor;
- Plăci de perfoboard;
- Varnish constrictor;
- Vopsea neagră tip spray;
- Şmirghel fin.

SCULE ŞI DISPOZITIVE:

- Polizor unghiular;
- Fierăstrău pendular cu lamă de tăiere specifică pentru plastic;
- Disc de tăiere pentru plastic;
- Markere subţiri;
- Letcon termostatat;
- Multimetru analogic SANWA;
- Cutter;
- Cleşte pentru tăierea şi dezizolarea firelor;
- Pistol cu aer cald.

Etapele tehnologice care au dus la realizarea fizică a tablei sunt următoarele:

1. Tăierea plăcii din plexiglas transparent la dimensiunea totală a tablei de joc (suprafaţă pătratică cu latura de 40 cm) şi decuparea găurilor pentru introducerea butoanelor şi a ceasului de şah.

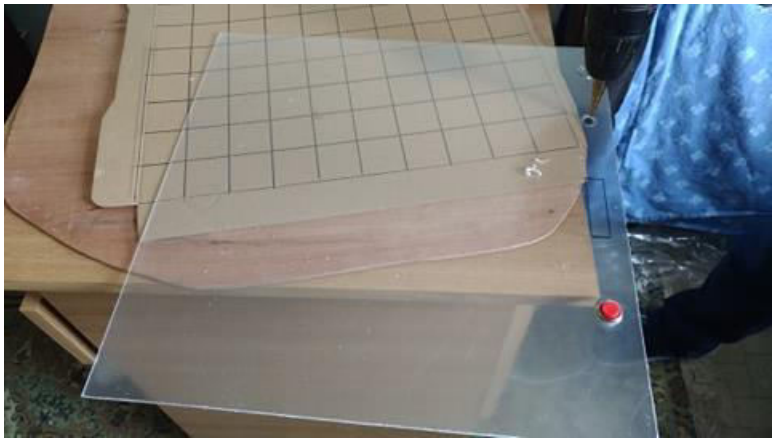


Figura 5.7 – Tăierea plăcii din plexiglas transparent

2. Caroierea plăcii din plexiglas prin zgârierea liniilor cu cutter-ul şi marcarea acestora cu vopsea neagră.

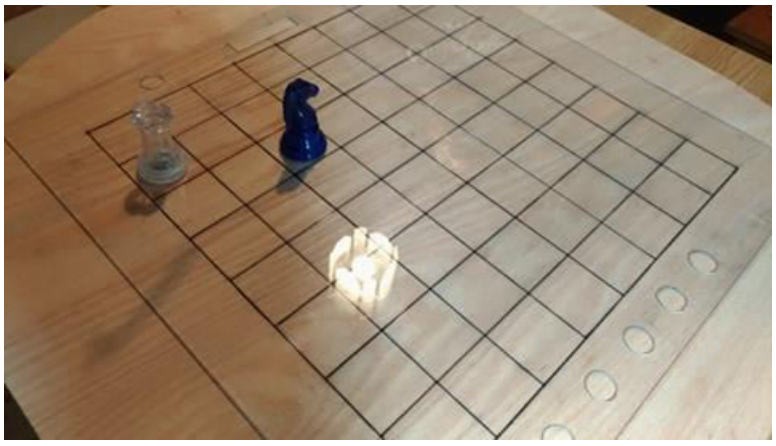


Figura 5.8 – Realizarea caroiajului pentru delimitarea câmpurilor de joc

3. Diferenţierea culorii câmpurilor prin lipirea celor două tipuri de autocolant semitransparent în vederea realizării contrastului.

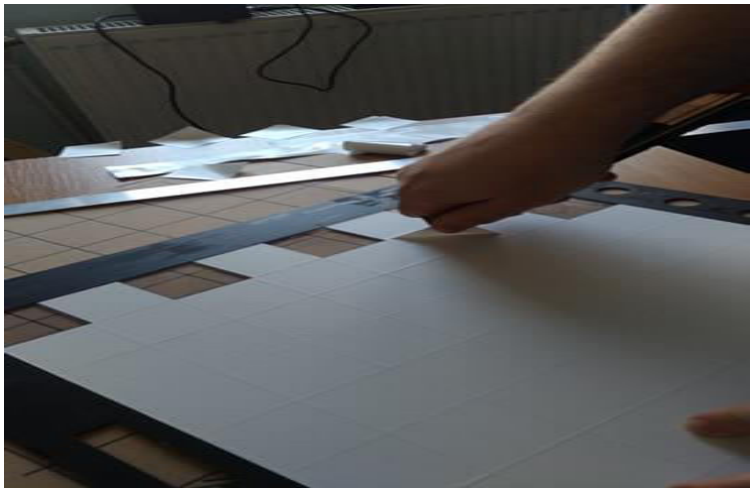


Figura 5.9 – Procesul de lipire şi dezlipire a autocolantului

4. Tăierea plăcii de plastic alb tip ABS în benzi folosite pentru stabilirea perimetrului tablei de şah şi a celulelor fagurelui interior (prin îmbinarea a 7 x 7 benzi în fante decupate până la jumătatea lăţimii), îmbinarea benzilor din ABS fiind realizată cu ajutorul unui pistol de lipit cu baghete de plastic.

Figura 5.10 – Structură tip fagure realizată din benzi de plastic alb ABS

5. Îmbinarea finală a fagurelui din plastic ABS cu placa prelucrată din plexiglas ce determină suprafaţa tablei de şah.

Figura 5.11 – Lipirea fagurelui de suprafaţa din plexiglas

6. Decuparea zonelor destinate conectorului pentru alimentarea tablei de şah şi a portului USB pentru programarea plăcii de dezvoltare şi introducerea benzilor ce definesc perimetrul tablei de joc în interiorul structurii descrise în etapa anterioară.

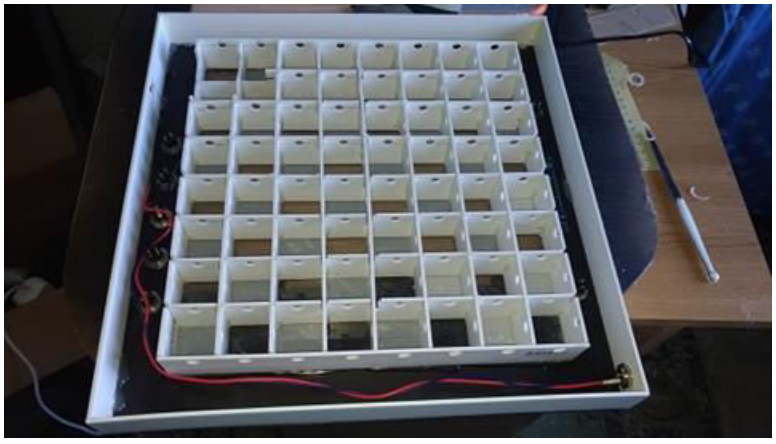


Figura 5.12 – Introducerea benzilor ce definesc perimetrul tablei de joc

7. Montarea pe partea interioară a tablei a senzorilor magnetici în cadrul fiecărui câmp de joc, prin lipirea acestora cu adeziv instant.

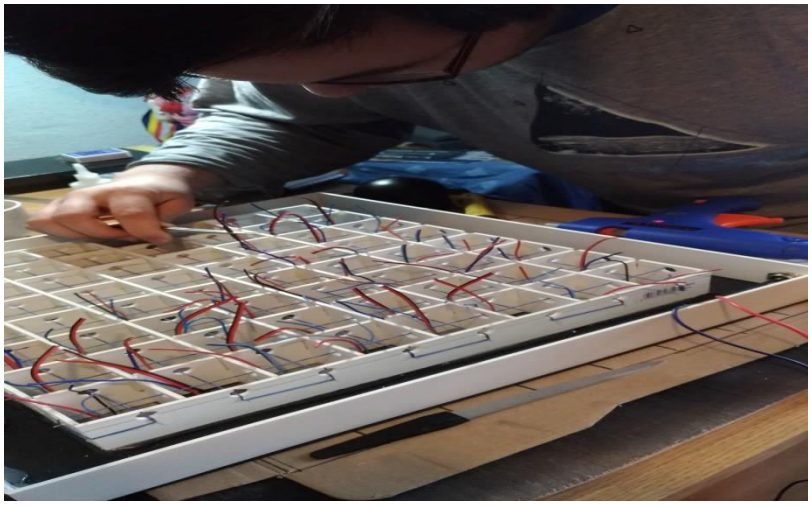


Figura 5.13 - Introducerea senzorilor pe partea inferioara a placii de plexiglas

8. Echiparea a şaizeci şi patru de plăci de perfoboard cu câte un senzor magnetic TLE4905, un LED SMD WS2812C şi a componentelor pasive (rezistor de pull-up pentru ieşirea senzorului şi a condensatorului de decuplare între pinul de alimentare şi cel de masă al LED-ului), precum şi a două plăci de perfoboard în care sunt conectate câte patru registre de deplasare SN74HC165N, aşezarea componentelor în cadrul tablei de şah respectând cu stricteţe poziţia stabilită din etapa de proiectare mecanică.

Figura 5.14 – Lipirea componentelor electronice şi introducerea plăcilor de perfoboard echipate în interiorul tablei de şah

9. Finalizarea conexiunilor electrice între componentele situate in interiorul fagurelui şi cele din exteriorul acestuia, precum şi a tuturor componentelor la terminalele de alimentare şi de masă ale sursei de alimentare.

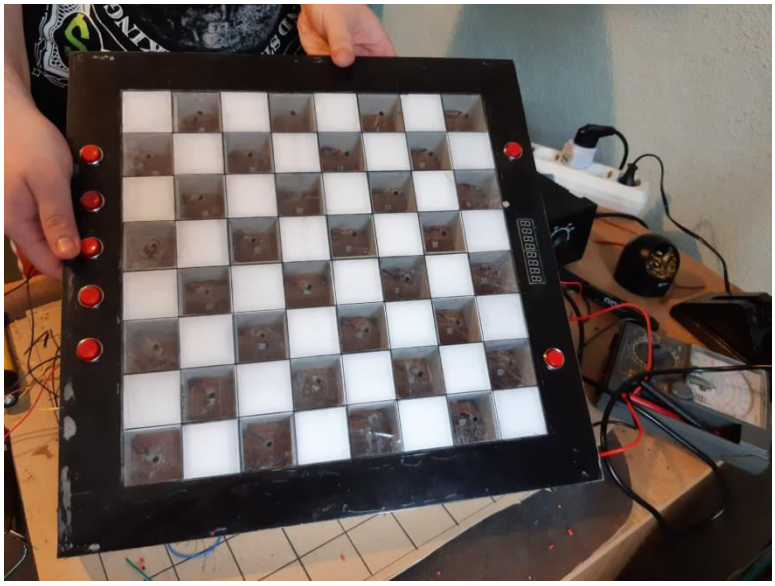


Figura 5.15 – Prezentarea produsului realizat – vedere din faţă

Figura 5.16 – Prezentarea produsului realizat – vedere din spate

Capitolul 6 - Experimente şi rezultate obţinute

6.1) Teste şi măsurători realizate pe circuitul electronic al tablei de şah

Pornind de la schema hardware stabilită pentru tabla de şah inteligentă (prezentată în anexa 1), procesul de testare al sistemului a pornit de la verificarea individuală a modulelor funcţionale ce alcătuiesc sistemul, fiecare din aceste dispozitive având alocate una sau mai multe funcţii prezentate în fişierele de configurare ale tablei de şah (detaliate în capitolul 4).

Astfel, testarea senzorilor s-a realizat atât înainte cât şi după introducerea acestora în structura fizică a tablei de şah prin măsurarea tensiunii de pe terminalul de ieşire al senzorului, urmărind obţinerea valorii de 0V în prezenţa magnetului din neodim pe suprafaţa sensibilă a senzorului şi a valorii de 5V în absenţa acestuia.

În cazul butoanelor, s-a urmărit citirea valorilor corecte de către placa de dezvoltare la acţionarea butoanelor, prin introducerea unei întârzieri în cod care să capteze modificarea valorii logice a bitului în urma apăsării butonului, la scurt timp după stabilizarea acestei valori şi să evite problemele de natură fizică (fenomenul “bounce” în limba engleză) care ar putea duce la citirea unei valori eronate.

Testele registrelor SN74HC165N şi ale modului MAX7219 s-au realizat cu ajutorul programului Proteus ce permite simularea comportamentului sistemelor incorporate. Dupa finalizarea testelor în Proteus, s-au realizat montaje asemănătoare cu cele de test descrise mai jos şi pe breadboard, urmărind obţinerea comportamentului stabilit încă din simulare.

În cazul registrelor SN74HC165N, s-a testat câte un registru de deplasare prin scrierea unei secvenţe de cod ce permite citirea a opt biţi primiţi de la senzori pe intrările paralele ale registrului (apropierea magnetului de senzorii cu efect Hall fiind un comportament modelat în simulator cu ajutorul unor comutatoare care să selecteze una dintre valorile de tensiune prezente la ieşirea senzorului) şi trimiterea serială a octetului rezultat prin concatenarea biţilor către placa de dezvoltare cu microcontroler ATmega2560. Citirea corectă a datelor de către Atmega2560 se verifică prin urmărirea continuă a octetului primit de la registrul de deplasare şi prin controlul unui grup de opt diode LED (conectate la opt pini digitali, configuraţi ca pini de ieşire, ai microcontrolerului) în conformitate cu valorile citite (schema de test fiind prezentată în figura 6.1). În urma acestui test, s-au verificat în mod gradual şi următoarele cazuri: testarea a două registre de deplasare conectate în cascadă, testarea a patru registre de deplasare şi în final testarea celor opt registre de deplasare, urmărind citirea corectă a opt octeţi, făcând la fiecare iteraţie modificările corespunzătoare în schema hardware de test şi în aplicaţia firmware.

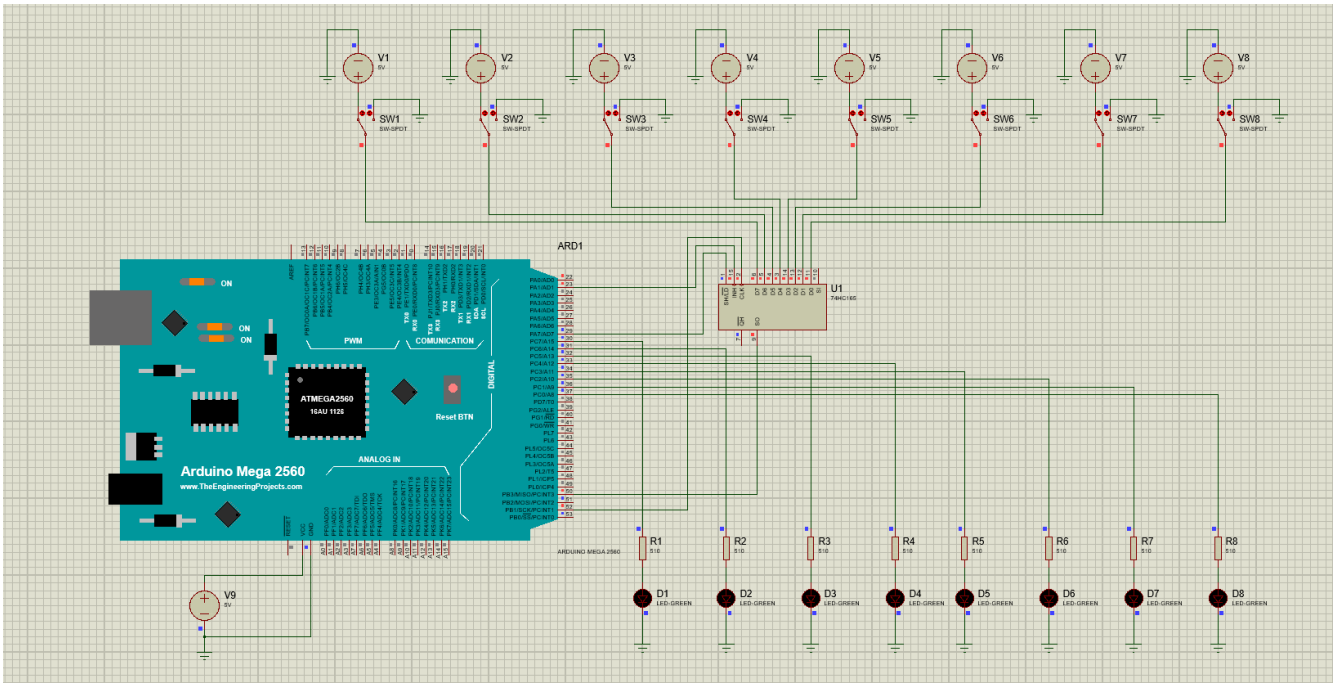


Figura 6.1 – Verificarea registrului SN74HC165N - trimiterea octetului 11111111 prin protocolul SPI

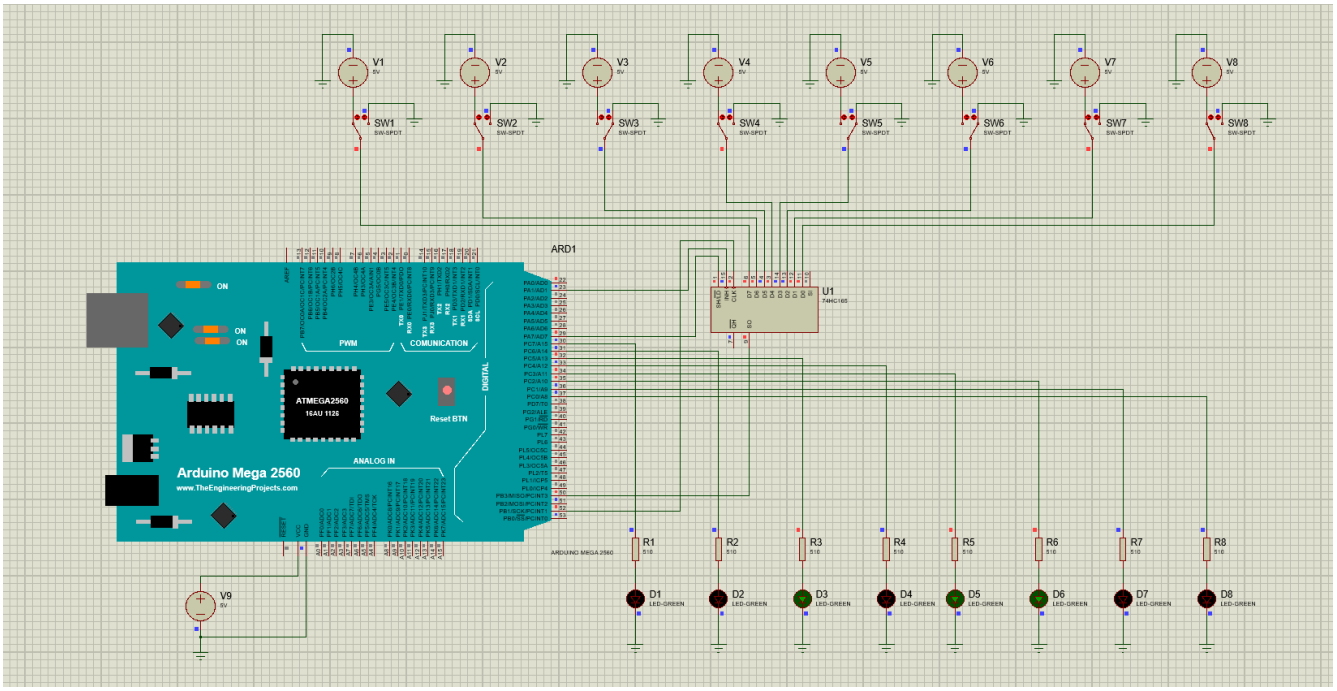


Figura 6.2 – Verificarea registrului SN74HC165N - trimiterea octetului 11010011 prin protocolul SPI

Pentru modulul MAX7219, montajul de test este format din placa de dezvoltare cu microcontroler ATmega2560 şi dispozitivele interne ale modulului MAX7219 (afişajul cu şapte

segmente şi opt digiţi şi driver-ul folosit pentru aprinderea segmentelor). Scopul acestui test este de a verifica corectitudinea comenzilor trimise prin protocolul SPI de la placa de dezvoltare la modulul MAX7219, în vederea vizualizării unor caractere alfanumerice pe cei opt digiţi ai afişajului.

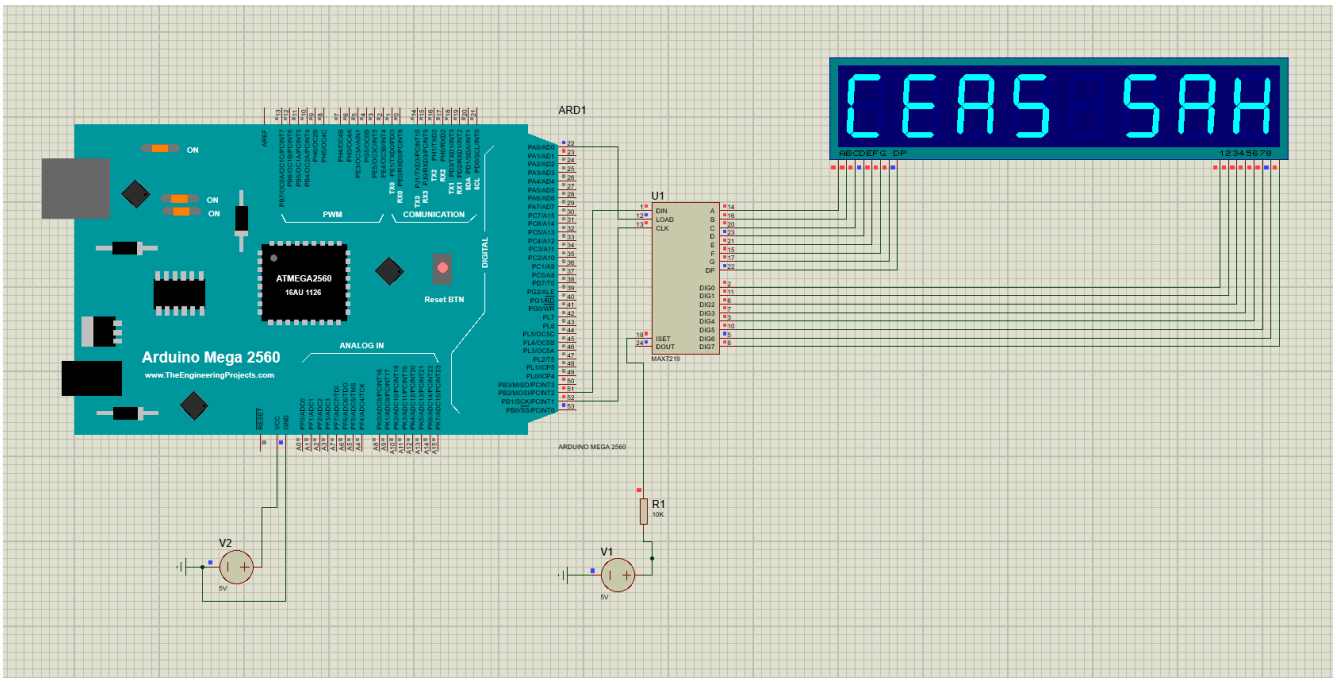


Figura 6.3 – Trimiterea comenzilor prin SPI pentru afişarea textului “ceas şah”

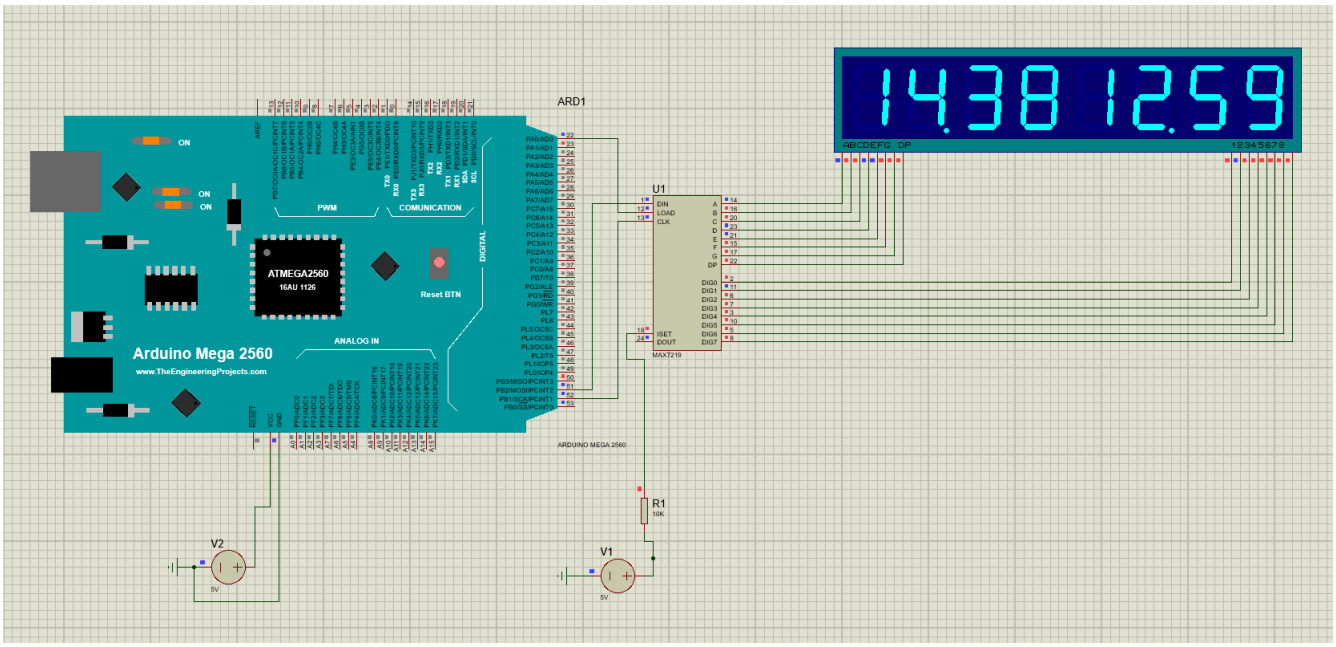


Figura 6.4 – Trimiterea comenzilor prin SPI pentru afişarea timpului de joc pe ceasul de şah

Pentru testarea LED-urilor WS2812C, s-a folosit iniţial o matrice de şaisprezece LED-uri conectate în cascadă, urmărind scrierea unui cod care să permită controlul individual al LED-urilor şi al culorilor afişate, în vederea stabilirii diverselor evenimente ce pot apărea într-o partidă de şah.

După verificarea individuală a modulelor, se realizează schema hardware a întregului sistem şi se confirmă cu ajutorul osciloscopului că se implementează corect comunicaţiile prin SPI (între ATmega2560 şi registrele de deplasare SN74HC165N / modulul MAX7219) şi NRZ (între ATmega2560 şi cele şaizeci şi patru de LED-uri WS2812C conectate în cascadă).

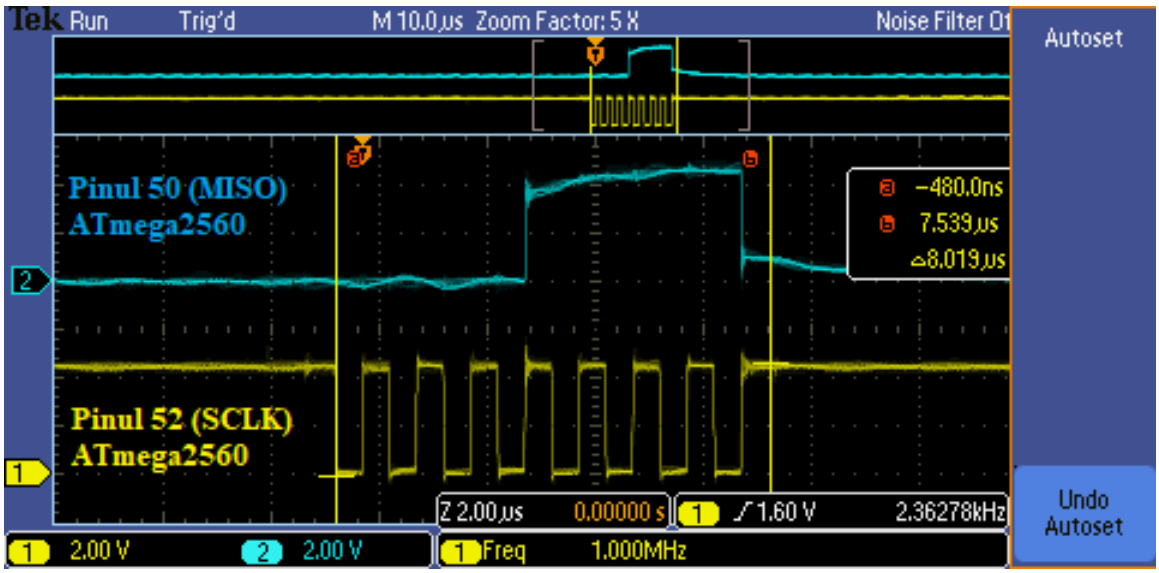


Figura 6.5 – Formatul pachetului de date (octetul “00001111”) trimis prin protocolul SPI

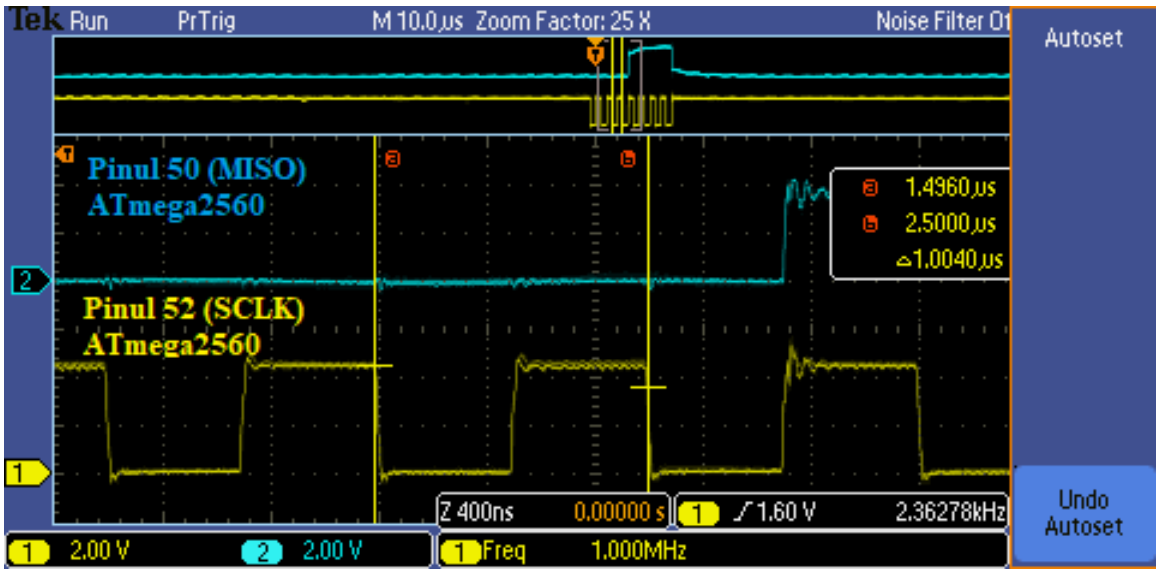


Figura 6.6 – Evidenţierea perioadei semnalului de ceas (vizualizat pe pinul SCLK al microcontrolerului, reprezentat cu culoarea galbenă) folosit în cadrul protocolului SPI

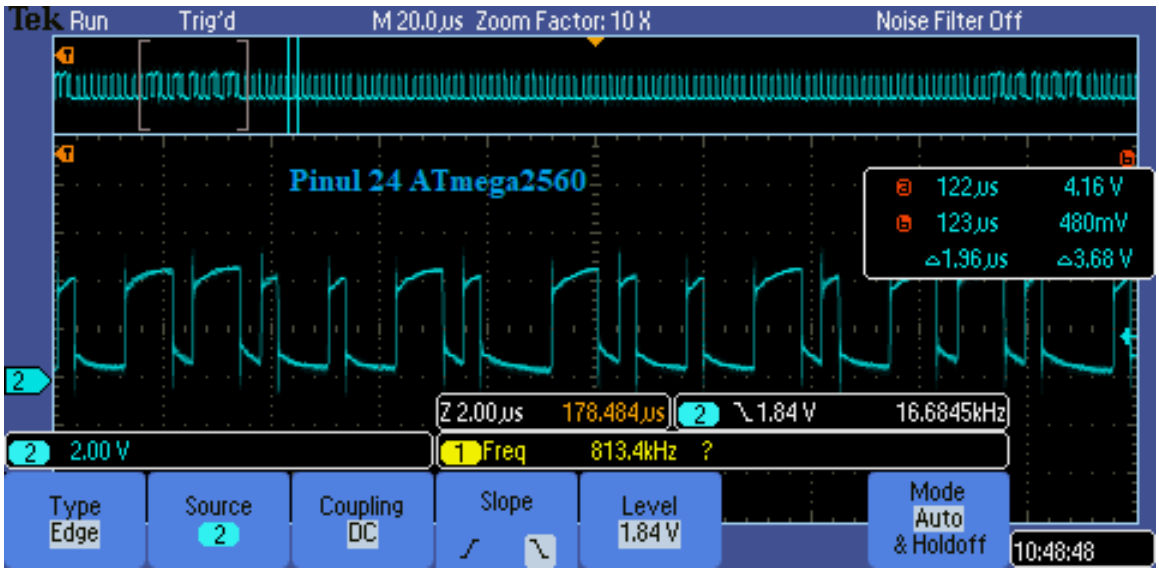


Figura 6.7 – Formatul pachetului de date (codul "1100110010110") trimis prin protocolul NRZ vizualizat la intrarea primul LED din grupul de 64 de LED-uri WS2812 conectate în cascadă

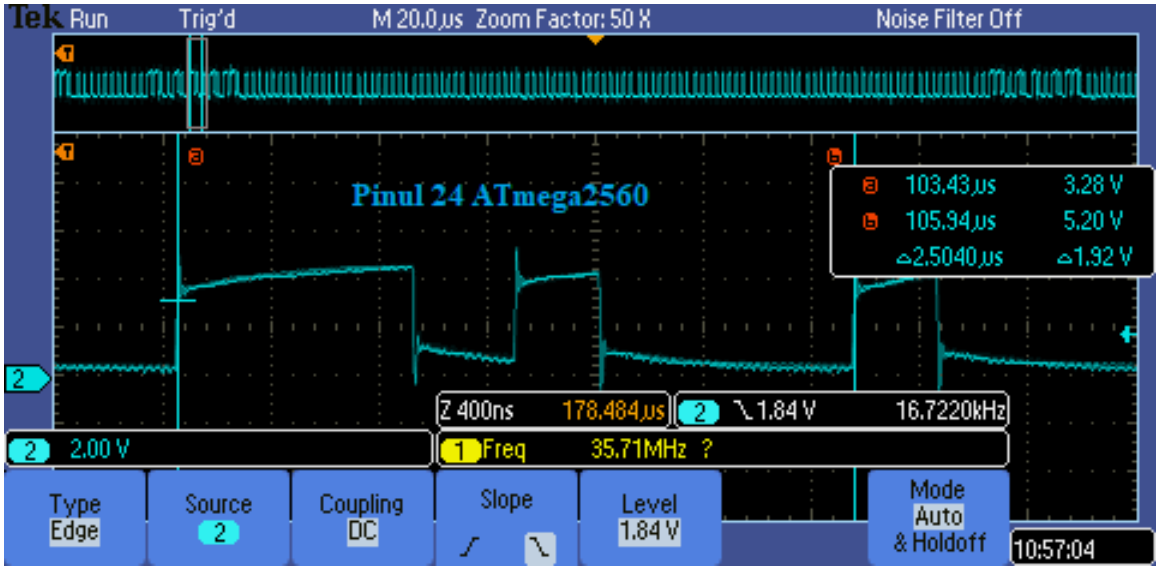


Figura 6.8 – Verificarea duratei codului "10" trimis prin protocolul NRZ vizualizat la intrarea primul LED din grupul de 64 de LED-uri WS2812 conectate în cascadă

În următorul subcapitol se prezintă metoda de depanare software a codului ce implementează regulile jocului de şah şi se arată că această metodă a dus la obţinerea funcţionalităţii dorite a sistemului incorporat realizat fizic.

6.2) Testarea software a codului ce implementează regulile jocului de şah

O problemă principală în cadrul lucrării de faţă a reprezentat-o testarea codului ce descrie regulile jocului de şah, deoarece doar această porţiune din întreaga aplicaţie firmware însumează mai bine de 3500 de linii de cod.

Astfel, s-a ajuns la concluzia că depanarea codului nu poate fi realizată doar prin utilizarea unui depanator pus la dispoziţie de către mediile de dezvoltare Atmel Studio 7 sau Visual Studio 2019, cheia constând în afişarea valorilor unui set de variable într-o manieră intuitivă şi uşor de descifrat, a căror interpretare să ducă la depanarea codului, iar modul de vizualizare al rezultatelor obţinute să fie cât mai apropiat de cel dorit a fi obţinut în cadrul tablei de şah inteligente realizate fizic.

În continuare sunt prezentate rezultatele obţinute în urma implementării regulilor şahului, confirmând testarea software realizată direct în consola programului cu reprezentarea unor situaţii similare obţinute pe tabla de şah fizică.

Se reamintesc semnificaţiile valorilor variabilelor afişate în consolă (majoritatea regăsindu-se şi în cadrul capitolului 4):

- "nrOfPiecesHeld" – reţine numărul de piese ridicate de către jucătorul care este la mutare;
- "newLedMatrix" – matrice ce determină aprinderea LED-urilor (în cazul valorilor nenule) în conformitate cu câmpurile unde poate fi plasată piesa de şah ridicată la mutarea curentă;
- "piecesPositionMatrix" – matrice ce stochează poziţia curentă a pieselor pe tabla de şah, fiecare piesă de şah fiind unic identificată printr-o valoare numerică, valorile exacte fiind prezentate în figura 4.7;
- "whoMoves" – determină partea care este la mutare (reţine valoarea "1" în cazul albului şi "0" în cazul negrului);
- "chessClockPressEnable" – setarea valorii "1" a acestei variabile reprezintă momentul în care jucătorul aflat la mutare realizează o mutare corectă, ceea ce permite acţionarea automata a ceasului de şah;
- "whiteCastleInProgress / blackCastleInProgress" – setarea valorii "1" în cadrul acestor variabile determină schimbarea modului în care poate fi mutat turnul (deoarece, în timpul realizării rocadei, turnul trebuie poziţionat lângă rege);
- "first_white_king_move/first_black_king_move" – variabile ce determină dacă cei doi regi sunt deplasaţi pentru prima dată de pe poziţiile iniţiale, caz în care se poate efectua rocada;
- "first_a_column_white_rook_move", precum şi celelalte variabile denumite asemănător verifică dacă turnurile au fost sau nu deplasate de pe poziţiile iniţiale în vederea realizării rocadei.
- "sensorValues" – variabilă ce stochează cei şaizeci şi patru de biţi citiţi de la senzori, valorile date acestei variabile, de la tastatură, în cadrul consolei urmăresc diagrama de decodare prezentată în anexa 2.

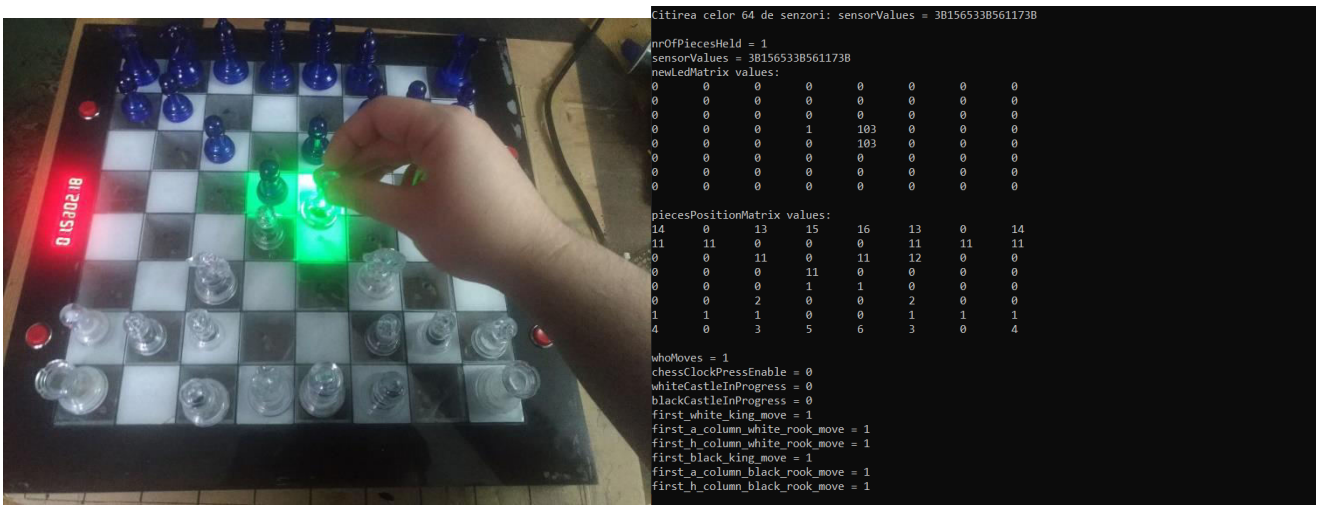


Figura 6.9 – Verificarea mutării pionului – ridicarea pionului

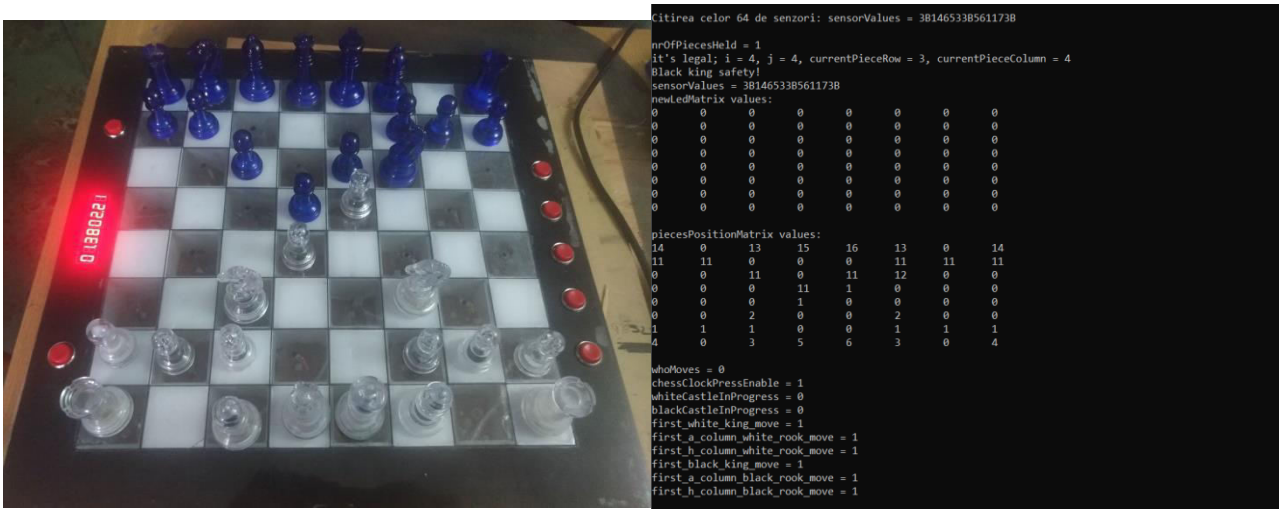


Figura 6.10 – Verificarea mutării pionului – plasarea pionului pe noul câmp

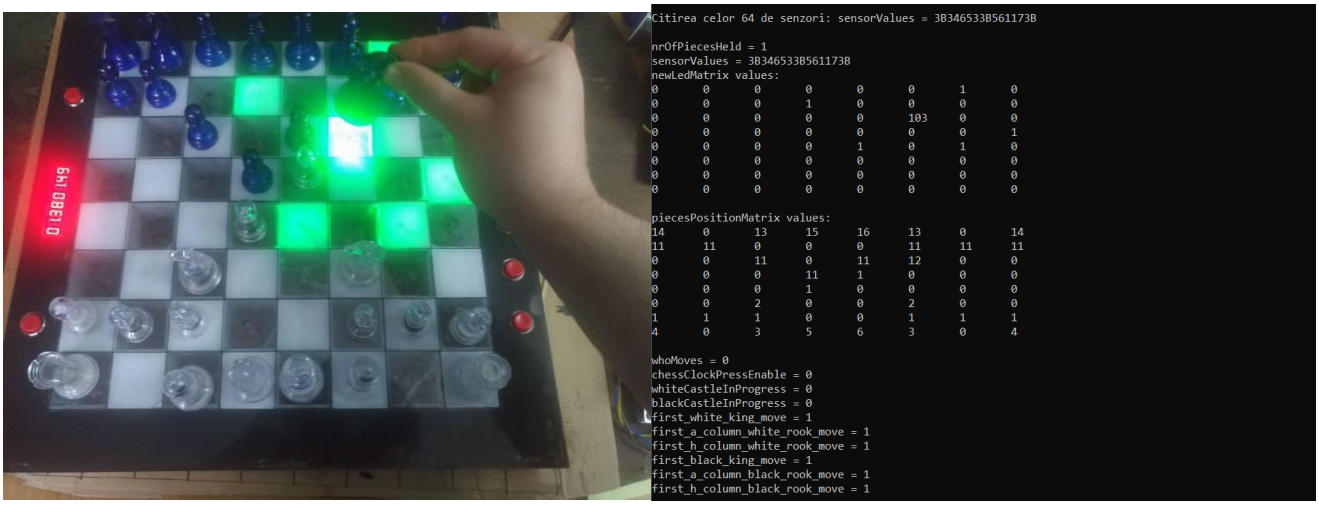


Figura 6.11 – Verificarea mutării calului – ridicarea calului

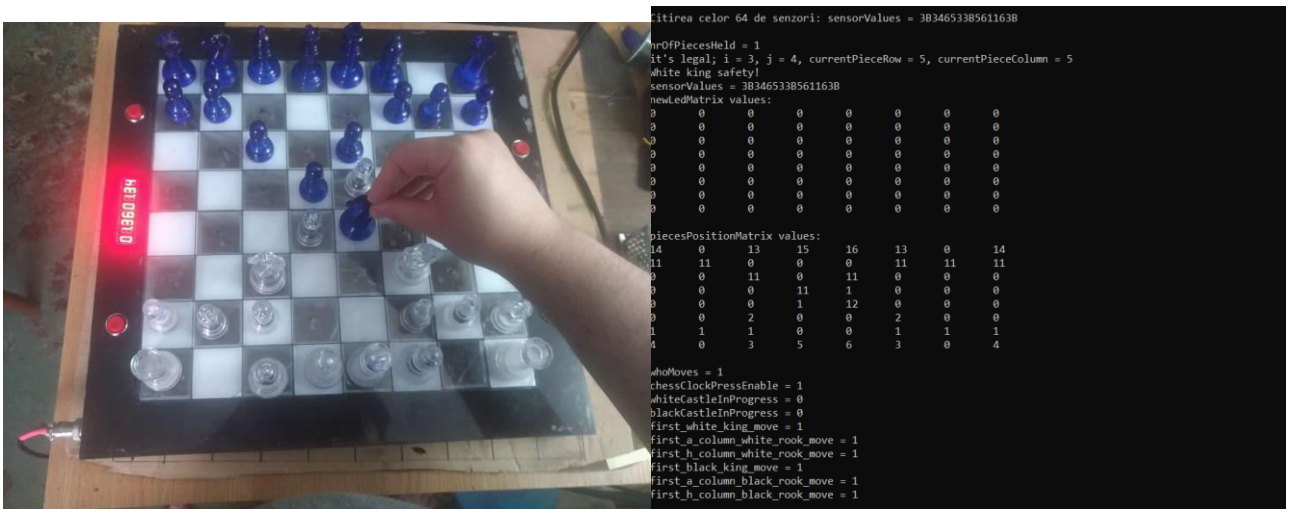


Figura 6.12 – Verificarea mutării calului – plasarea calului pe noul câmp

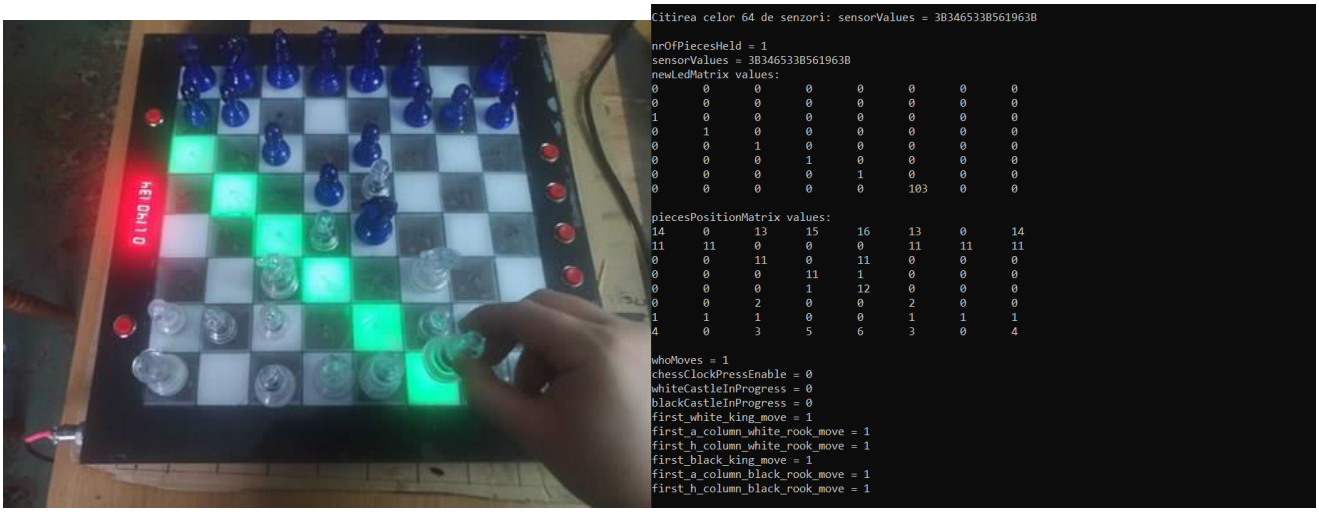


Figura 6.13 – Verificarea mutării nebunului – ridicarea nebunului

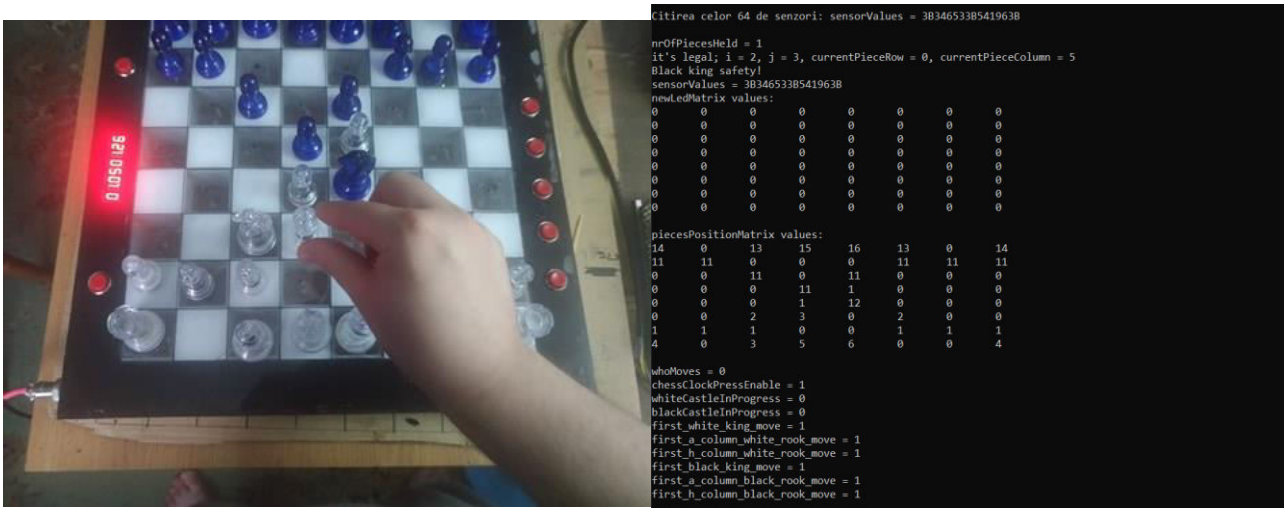
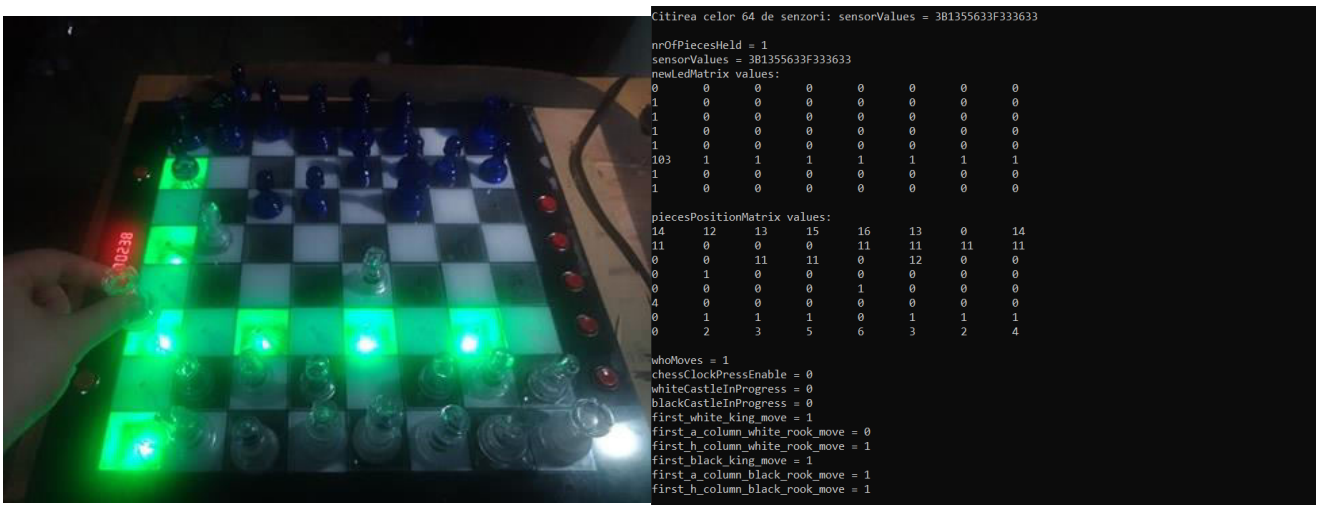


Figura 6.14 – Verificarea mutării nebunului – plasarea nebunului pe noul câmp



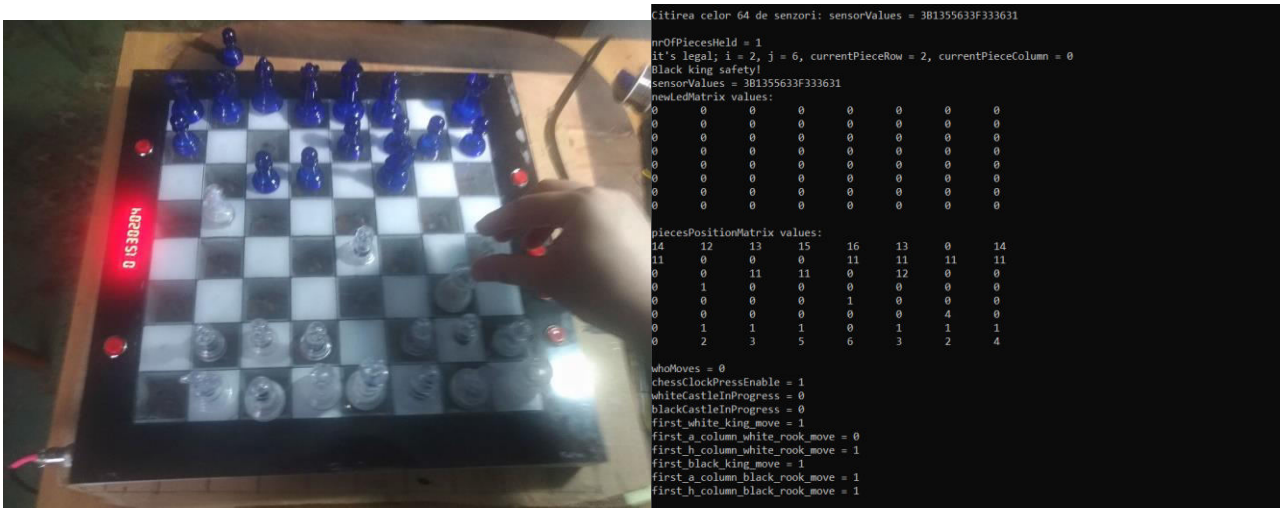
```
Citirea celor 64 de senzori: sensorValues = 3B1355633F333633

nrOfPiecesHeld = 1
sensorValues = 3B1355633F333633
newLedMatrix values:
0       0       0       0       0       0       0       0
1       0       0       0       0       0       0       0
1       0       0       0       0       0       0       0
1       0       0       0       0       0       0       0
1       0       0       0       0       0       0       0
103     1       1       1       1       1       1       1
1       0       0       0       0       0       0       0
1       0       0       0       0       0       0       0

piecesPositionMatrix values:
14      12      13      15      16      13      0       14
11      0       0       0       11      11      11      11
0       0       11      11      0       12      0       0
0       1       0       0       0       0       0       0
0       0       0       0       1       0       0       0
4       0       0       0       0       0       0       0
0       1       1       1       0       1       1       1
0       2       3       5       6       3       2       4

whoMoves = 1
chessClockPressEnable = 0
whiteCastleInProgress = 0
blackCastleInProgress = 0
first_white_king_move = 1
first_a_column_white_rook_move = 0
first_h_column_white_rook_move = 1
first_black_king_move = 1
first_a_column_black_rook_move = 1
first_h_column_black_rook_move = 1
```

Figura 6.15 – Verificarea mutării turnului – ridicarea turnului



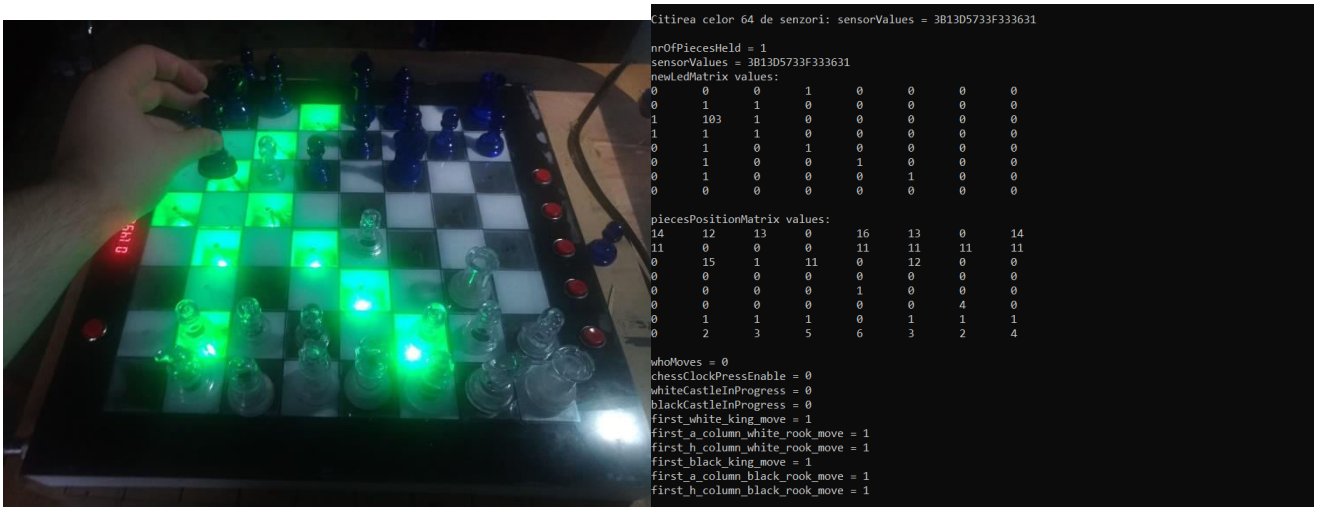
```
Citirea celor 64 de senzori: sensorValues = 3B1355633F333631

nrOfPiecesHeld = 1
it's legal; i = 2, j = 6, currentPieceRow = 2, currentPieceColumn = 0
Black king safety!
sensorValues = 3B1355633F333631
newLedMatrix values:
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0

piecesPositionMatrix values:
14      12      13      15      16      13      0       14
11      0       0       0       11      11      11      11
0       0       11      11      0       12      0       0
0       1       0       0       0       0       0       0
0       0       0       0       1       0       0       0
0       0       0       0       0       0       4       0
0       1       1       1       0       1       1       1
0       2       3       5       6       3       2       4

whoMoves = 0
chessClockPressEnable = 1
whiteCastleInProgress = 0
blackCastleInProgress = 0
first_white_king_move = 1
first_a_column_white_rook_move = 0
first_h_column_white_rook_move = 1
first_black_king_move = 1
first_a_column_black_rook_move = 1
first_h_column_black_rook_move = 1
```

Figura 6.16 – Verificarea mutării turnului – plasarea turnului pe noul câmp



```
Citirea celor 64 de senzori: sensorValues = 3B13D5733F333631

nrOfPiecesHeld = 1
sensorValues = 3B13D5733F333631
newLedMatrix values:
0       0       0       1       0       0       0       0
0       1       1       0       0       0       0       0
1       103     1       0       0       0       0       0
1       1       1       0       0       0       0       0
0       1       0       1       0       0       0       0
0       1       0       0       1       0       0       0
0       1       0       0       0       1       0       0
0       0       0       0       0       0       0       0

piecesPositionMatrix values:
14      12      13      0       16      13      0       14
11      0       0       0       11      11      11      11
0       15      1       11      0       12      0       0
0       0       0       0       0       0       0       0
0       0       0       0       1       0       0       0
0       0       0       0       0       0       4       0
0       1       1       1       0       1       1       1
0       2       3       5       6       3       2       4

whoMoves = 0
chessClockPressEnable = 0
whiteCastleInProgress = 0
blackCastleInProgress = 0
first_white_king_move = 1
first_a_column_white_rook_move = 1
first_h_column_white_rook_move = 1
first_black_king_move = 1
first_a_column_black_rook_move = 1
first_h_column_black_rook_move = 1
```

Figura 6.17 – Verificarea mutării damei – ridicarea damei

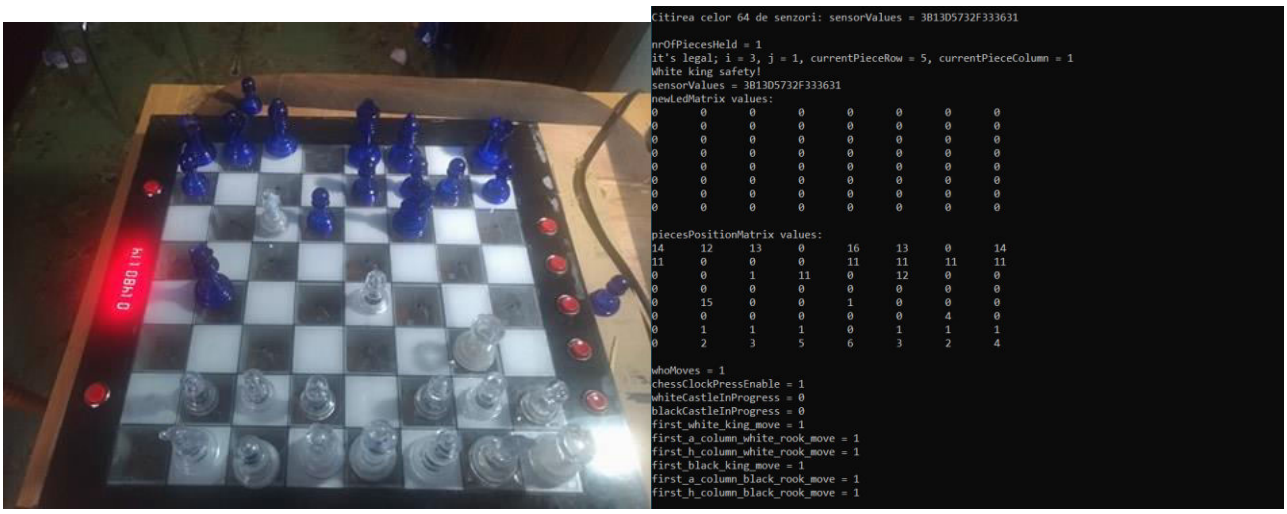


Figura 6.18 – Verificarea mutării damei – plasarea damei pe noul câmp

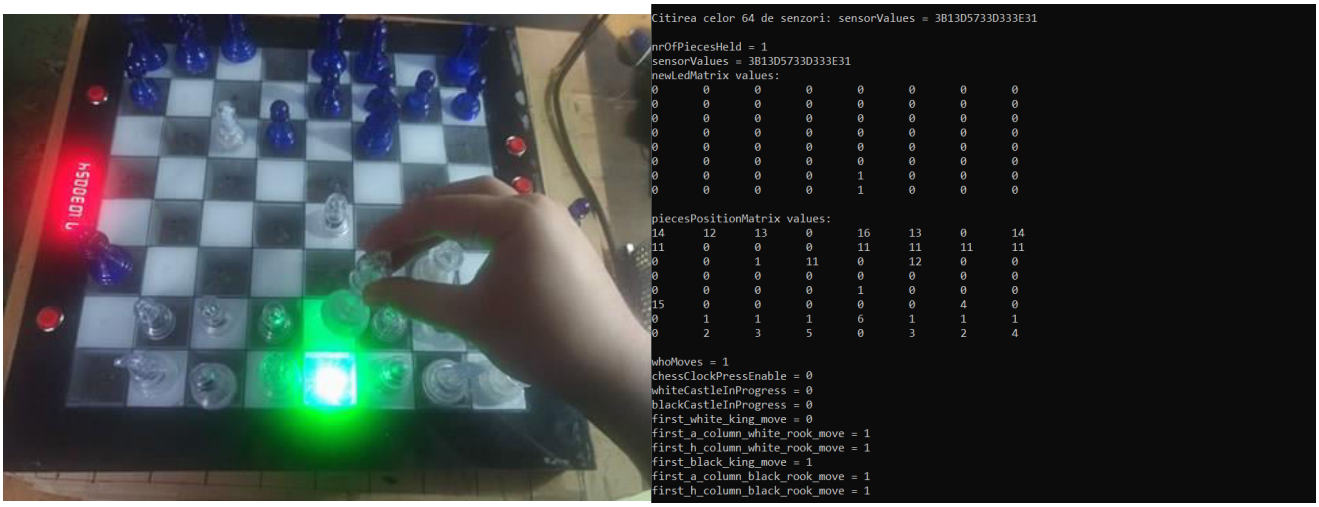


Figura 6.19 – Verificarea mutării regelui – ridicarea regelui

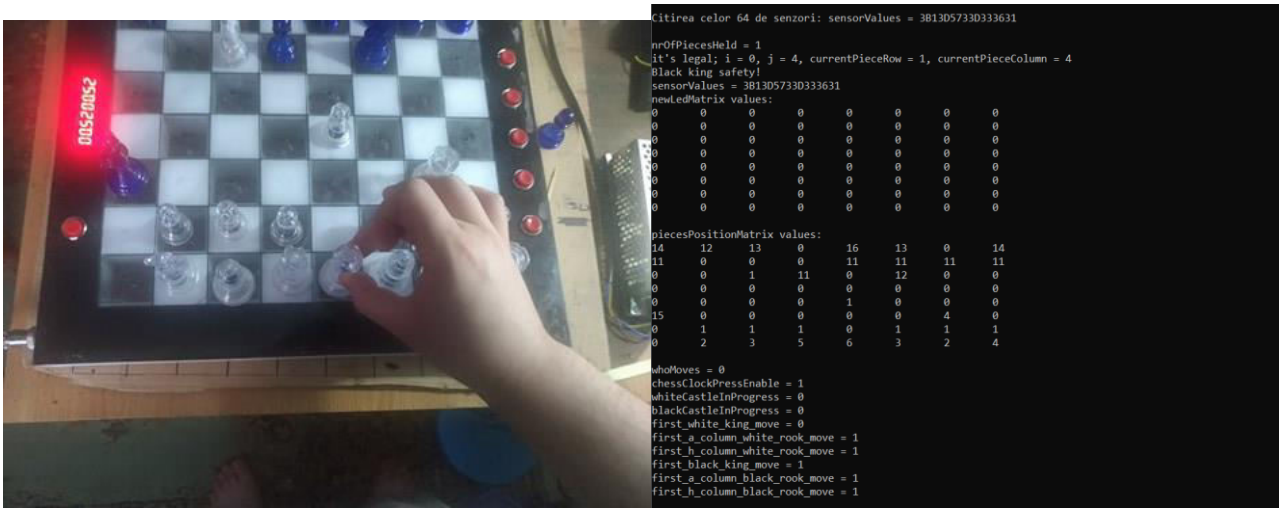


Figura 6.20 – Verificarea mutării regelui – plasarea regelui pe noul câmp

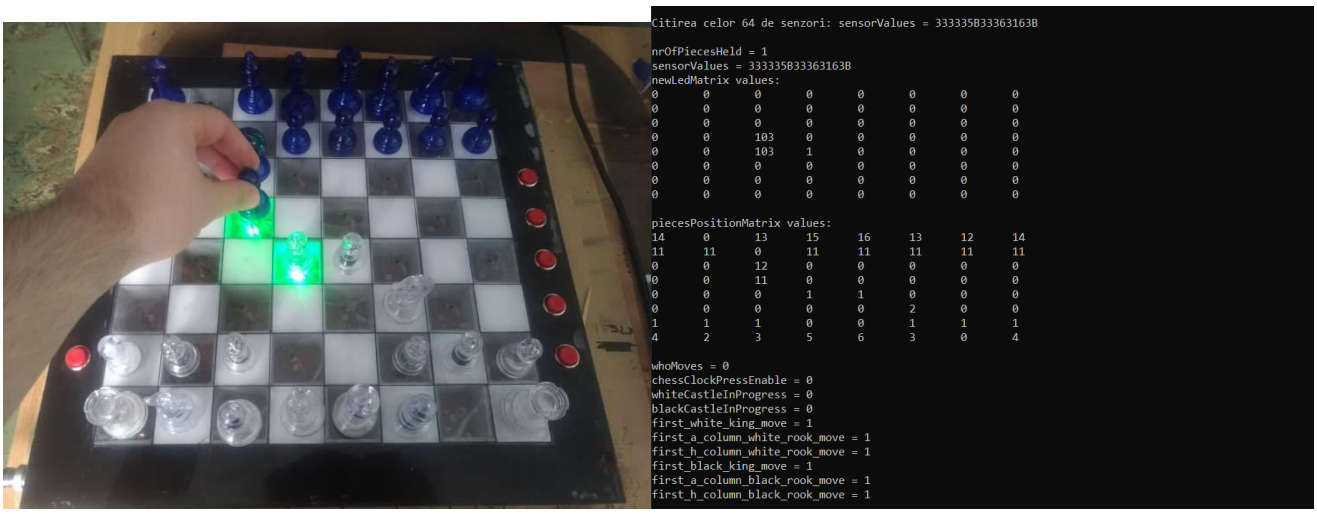


Figura 6.21 – Verificarea unei capturi de piesă – ridicarea piesei atacatoare

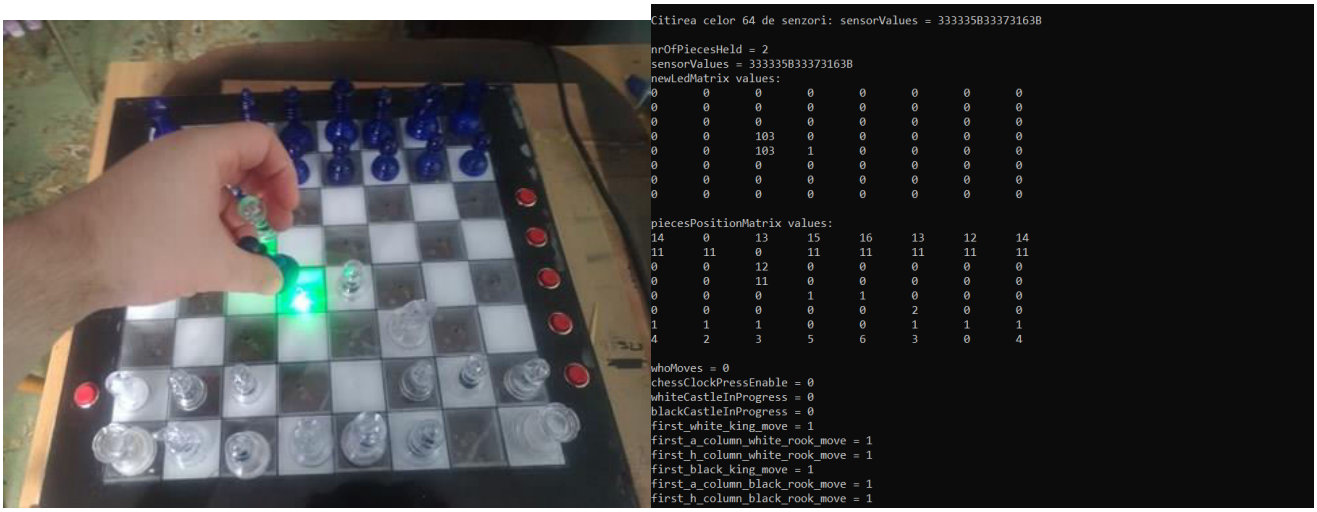


Figura 6.22 – Verificarea unei capturi de piesă – ridicarea piesei atacate

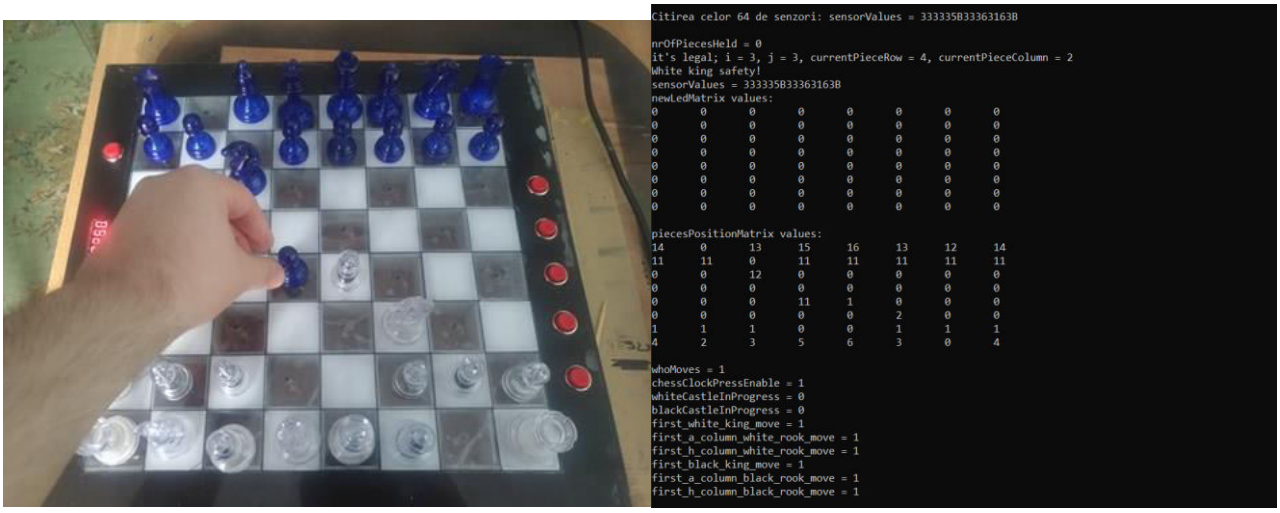
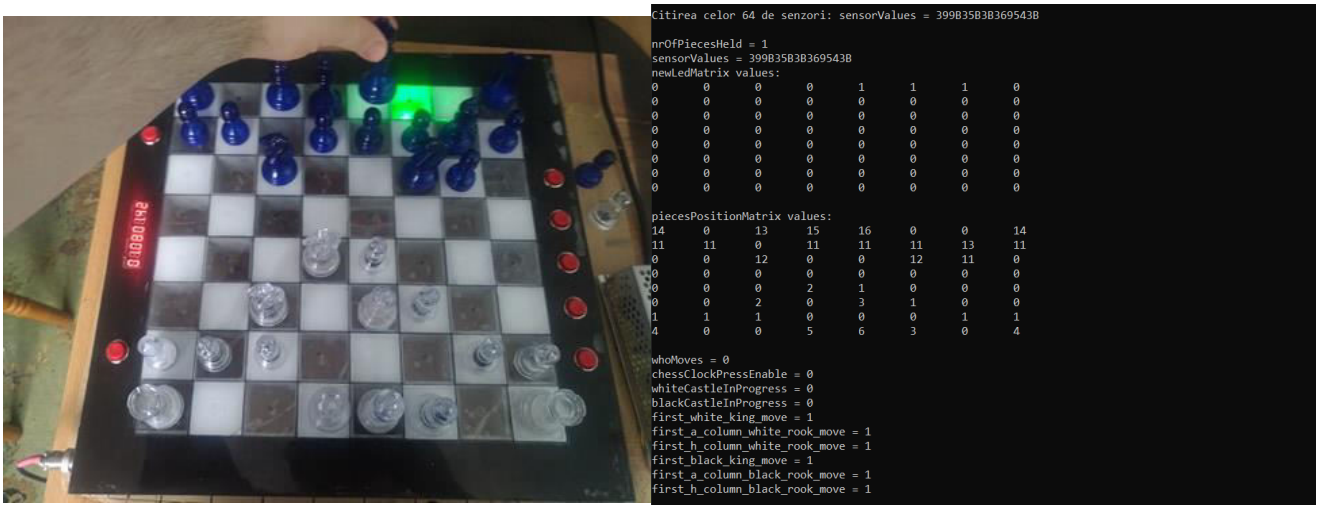


Figura 6.23 – Verificarea unei capturi de piesă – plasarea piesei atacatoare pe noul câmp



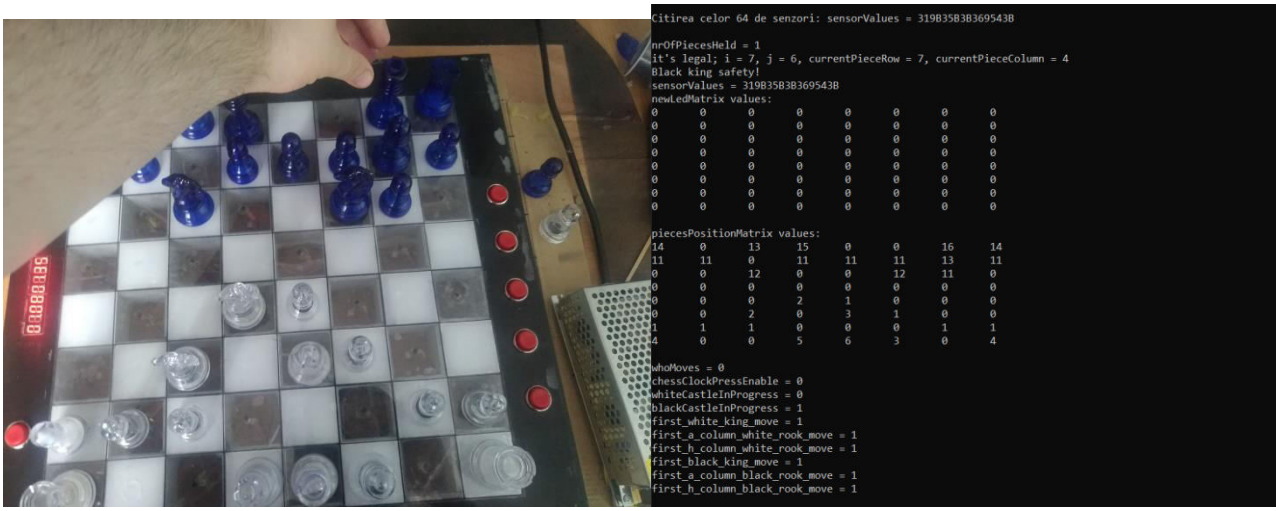
```
Citirea celor 64 de senzori: sensorValues = 399B35B3B369543B

nrOfPiecesHeld = 1
sensorValues = 399B35B3B369543B
newLedMatrix values:
0       0       0       0       1       1       1       0
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0

piecesPositionMatrix values:
14      0       13      15      16      0       0       14
11      11      0       11      11      11      13      11
0       0       12      0       0       12      11      0
0       0       0       0       0       0       0       0
0       0       0       2       1       0       0       0
0       0       2       0       3       1       0       0
1       1       1       0       0       0       1       1
4       0       0       5       6       3       0       4

whoMoves = 0
chessClockPressEnable = 0
whiteCastleInProgress = 0
blackCastleInProgress = 0
first_white_king_move = 1
first_a_column_white_rook_move = 1
first_h_column_white_rook_move = 1
first_black_king_move = 1
first_a_column_black_rook_move = 1
first_h_column_black_rook_move = 1
```

Figura 6.24 – Verificarea rocadei – ridicarea regelui



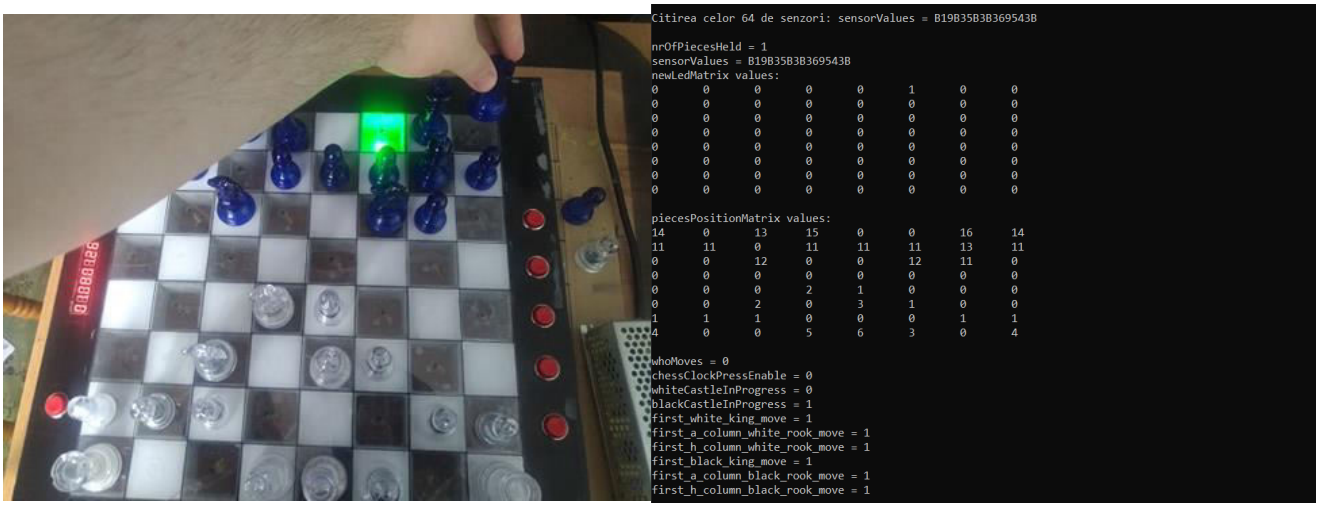
```
Citirea celor 64 de senzori: sensorValues = 319B35B3B369543B

nrOfPiecesHeld = 1
it's legal; i = 7, j = 6, currentPieceRow = 7, currentPieceColumn = 4
Black king safety!
sensorValues = 319B35B3B369543B
newLedMatrix values:
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0

piecesPositionMatrix values:
14      0       13      15      0       0       16      14
11      11      0       11      11      11      13      11
0       0       12      0       0       12      11      0
0       0       0       0       0       0       0       0
0       0       0       2       1       0       0       0
0       0       2       0       3       1       0       0
1       1       1       0       0       0       1       1
4       0       0       5       6       3       0       4

whoMoves = 0
chessClockPressEnable = 0
whiteCastleInProgress = 0
blackCastleInProgress = 1
first_white_king_move = 1
first_a_column_white_rook_move = 1
first_h_column_white_rook_move = 1
first_black_king_move = 1
first_a_column_black_rook_move = 1
first_h_column_black_rook_move = 1
```

Figura 6.25 – Verificarea rocadei – plasarea regelui pe câmpul corespunzător rocadei



```
Citirea celor 64 de senzori: sensorValues = B19B35B3B369543B

nrOfPiecesHeld = 1
sensorValues = B19B35B3B369543B
newLedMatrix values:
0       0       0       0       0       1       0       0
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0

piecesPositionMatrix values:
14      0       13      15      0       0       16      14
11      11      0       11      11      11      13      11
0       0       12      0       0       12      11      0
0       0       0       0       0       0       0       0
0       0       0       2       1       0       0       0
0       0       2       0       3       1       0       0
1       1       1       0       0       0       1       1
4       0       0       5       6       3       0       4

whoMoves = 0
chessClockPressEnable = 0
whiteCastleInProgress = 0
blackCastleInProgress = 1
first_white_king_move = 1
first_a_column_white_rook_move = 1
first_h_column_white_rook_move = 1
first_black_king_move = 1
first_a_column_black_rook_move = 1
first_h_column_black_rook_move = 1
```

Figura 6.26 – Verificarea rocadei – ridicarea turnului

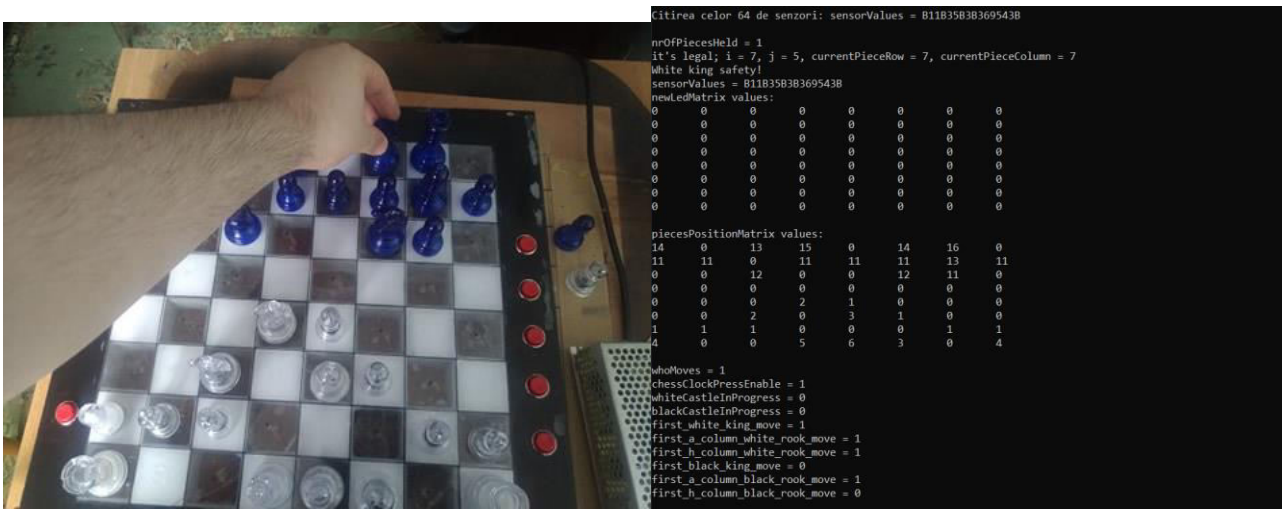


Figura 6.27 – Verificarea rocadei – plasarea turnului pe câmpul corespunzător rocadei

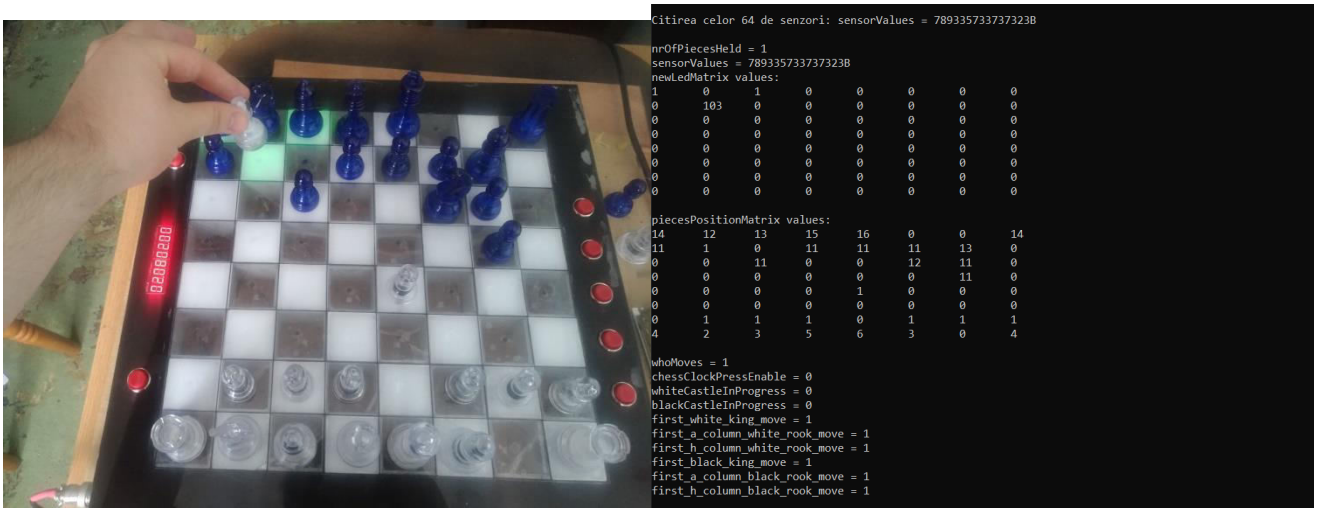


Figura 6.28 – Promovarea pionului – ridicarea pionului

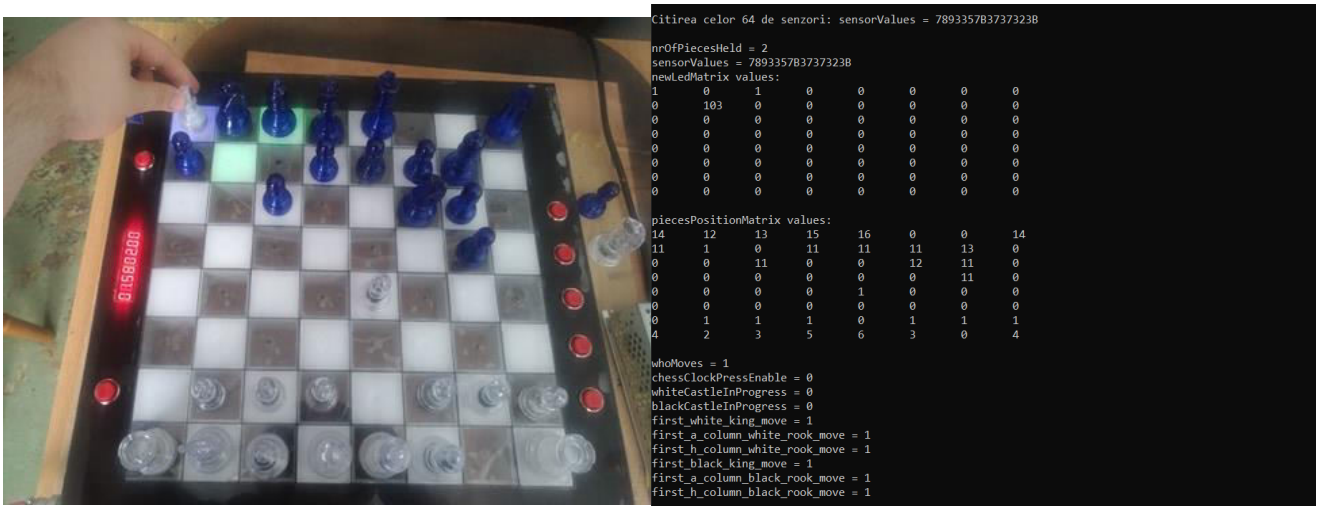


Figura 6.29 – Promovarea pionului – plasarea pionului pe câmpul de transformare

```
Citirea celor 64 de senzori: sensorValues = 7893357337373238

nrOfPiecesHeld = 0
it's legal; i = 7, j = 0, currentPieceRow = 6, currentPieceColumn = 1
Give BUTTON_KNIGHT_PRESSED =  1

Give BUTTON_BISHOP_PRESSED =  0

Give BUTTON_ROOK_PRESSED =  0

Give BUTTON_QUEEN_PRESSED =  0

Black king safety!
sensorValues = 789335733737323B
newLedMatrix values:
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0

piecesPositionMatrix values:
2       12      13      15      16      0       0       14
11      0       0       11      11      11      13      0
0       0       11      0       0       12      11      0
0       0       0       0       0       0       11      0
0       0       0       0       1       0       0       0
0       0       0       0       0       0       0       0
0       1       1       1       0       1       1       1
4       2       3       5       6       3       0       4

whoMoves = 0
chessClockPressEnable = 1
whiteCastleInProgress = 0
blackCastleInProgress = 0
first_white_king_move = 1
first_a_column_white_rook_move = 1
first_h_column_white_rook_move = 1
first_black_king_move = 1
first_a_column_black_rook_move = 1
first_h_column_black_rook_move = 1
```

Figura 6.30 – Promovarea pionului – transformarea pionului în cal

```
sensorValues = 3BB635339539723A
newLedMatrix values:
0       0       0       0       0       1       0       0
0       0       0       0       1       0       0       0
0       1       0       1       0       0       0       0
0       0       103     0       0       0       0       0
0       1       0       1       0       0       0       0
1       0       0       0       1       0       0       0
0       0       0       0       0       1       0       0
0       0       0       0       0       0       0       0

piecesPositionMatrix values:
14      12      13      15      16      0       0       14
11      11      0       11      0       11      11      11
0       0       11      0       0       0       0       0
0       0       13      0       11      0       0       0
0       0       0       0       1       0       12      0
1       1       2       0       0       0       0       0
0       3       1       1       2       1       1       1
4       0       0       5       6       3       0       4

whoMoves = 0
chessClockPressEnable = 1
whiteCastleInProgress = 0
blackCastleInProgress = 0
first_white_king_move = 1
first_a_column_white_rook_move = 1
first_h_column_white_rook_move = 1
first_black_king_move = 1
first_a_column_black_rook_move = 1
first_h_column_black_rook_move = 1

Citirea celor 64 de senzori: sensorValues = 3BB635339539323A

nrOfPiecesHeld = 0
it's legal; i = 1, j = 5, currentPieceRow = 4, currentPieceColumn = 2
White king safety!

Congratulation! Black won!

piecesPositionMatrix values:
14      12      13      15      16      0       0       14
11      11      0       11      0       11      11      11
0       0       11      0       0       0       0       0
0       0       0       0       11      0       0       0
0       0       0       0       1       0       12      0
1       1       2       0       0       0       0       0
0       3       1       1       2       13      1       1
4       0       0       5       6       3       0       4
```

Figura 6.31 – Situaţie de mat – finalizarea partidei de şah (câstigator jucătorul cu piesele negre)

Concluzii, contribuţii personale şi îmbunătăţiri ulterioare

Concluzii

Lucrarea de faţă a avut drept scop proiectarea şi realizarea fizică unei table de şah inteligente menite să ghideze utilizatorii acesteia în procesul de învăţare al jocului de şah, prin asigurarea unor funcţionalităţi care să răspundă în timp real la stimulii aplicaţi (indicarea câmpurilor posibile de joc la ridicarea unei piese, acţionarea automata a ceasului de şah în urma efectuării unei mutări corecte şi avertizarea utilizatorului în urma realizării unei mutări eronate, afişarea prin culori distincte a LED-urilor a unor evenimente specifice jocului de şah, precum: efectuarea rocadei, transformarea pionului, ameninţarea de mat). Pentru a ajunge la produsul finit, a fost necesară parcurgerea următoarelor etape: proiectarea şi implementarea schemei hardware din anexa 1 alcătuită din blocurile funcţionale prezentate în capitolul 3, realizarea unei aplicaţii firmware care să cuprindă atât funcţii pentru configurarea blocurilor hardware, cât şi funcţii pentru implementarea regulilor jocului de şah (subiectul capitolului 4), proiectarea şi realizarea unei structurii fizice care să încapsuleze circuitul electronic al tablei de şah (evidenţiate în capitolul 5), oferind astfel utilizatorilor o experienţă unică pe parcursul desfăşurării partidelor de şah, precum şi testarea componentelor hardware şi software pe parcursul dezvoltării tablei de şah inteligente (procese prezentate în capitolul 6).

Contribuţii personale

- Realizarea şi implementarea schemei hardware a întregului sistem, care să asigure citirea corectă a datelor de intrare din aplicaţia firmware şi trimiterea datelor prelucrate software către dispozitivele de ieşire;
- Scrierea unei biblioteci de funcţii care să permită configurarea blocurilor ce alcătuiesc întregul sistem, pornind de la secvenţele de cod C prezentate în foaia de catalog a microcontrolerului ATmega2560 (sursa [6]) şi folosind fişierele “light_ws2812.c”, “light_ws2812.h”, “ws2812_config.h” pentru controlul LED-urilor WS2812C, preluate de la sursa [18];
- Realizarea unei biblioteci de funcţii ce descrie regulile jocului de şah aplicate celor şase tipuri de piese existente (pion, cal, nebun, turn, damă, rege) şi a programului principal care apelează funcţiile definite în cele două biblioteci în vederea obţinerii funcţionalităţii finale a întregului sistem;
- Realizarea unei structuri fizice care să evidenţieze că aplicaţia finală a circuitului electronic realizat este cea de tablă de şah inteligentă.
- Efectuare testelor descrise în capitolul 6 cu ajutorul multimetrului şi al simulatorului Proteus în cazul modulelor electronice şi introducerea unei metode rapide de depanare a codului ce implementează regulile jocului de şah.

Îmbunătăţiri ulterioare

Deoarece lucrarea “Tablă de şah inteligentă” pune accentul pe gestionarea diverselor situaţii ce pot apărea înainte sau în timpul unei partide de şah, majoritate îmbunătăţirilor ce pot fi aduse vizează procesul de salvare şi gestionare facilă a partidelor de şah jucate, în vederea analizării acestora de către utilizatori. O soluţie propusă în această direcţie constă în realizarea unor funcţii care să convertească mutările efectuate pe tabla de şah în format text, acesta fiind ulterior trimis prin intermediul interfeţei seriale USART a microcontrolerului ATmega2560 către o aplicaţie realizată pe un dispozitiv PC. În cadrul aplicaţiei, partidele pot fi salvate într-un fişier text şi apoi vor fi convertite într-un format suportat de către bazele de date deja existente.

O altă îmbunătăţire poate consta în introducerea unui mod de setare al ceasului de şah care să permită incrementarea timpului de joc cu un număr fix de minute sau secunde după efectuarea unei mutări.

Bibliografie:

[1] “Originea şahului” disponibil la : https://adevarul.ro/locale/iasi/legendele-spatele-jocului-sah-inventat-fapt-mai-celebru-mai-raspandit-joc-mintii-1_56cc6e9c5ab6550cb81f808f/index.html , accesat la data de : 24.05.2020.

[2] “Structura sistemelor incorporate” disponibil la : https:// internetofthingsagenda.techtarget.com/ definition/embedded-system , accesat la data de: 25.05.2020.

[3] “Foaie de catalog TLE4905 – Infineon” disponibil la : https://ro.mouser.com/ Semiconductors/ Sensor-ICs /TLE4905-Series/Datasheets/ /N-6gixy?P=1yuy2ng , accesat la data de: 24.11.2019.

[4] “Prezentarea efectului Hall” disponibil la : http://hyperphysics.phy-astr.gsu.edu/hbase/ magnetic/ Hall.html , accesat la data de : 27.05.2020.

[5] “Principiul de funcţionare al senzorilor magnetici cu efect Hall” disponibil la : https://sensing.honeywell.com/honeywell-sensing-sensors-magnetoresistive-hall-effect-applications-005715-2-en2.pdf , accesat la data de : 27.05.2020.

[6] ”Foaie de catalog ATmega2560 – Microchip” disponibil la : https://ww1.microchip.com/downloads/en/devicedoc/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf , accesat la data de : 22.11.2019.

[7] “Structura pinilor plăcii de dezvoltare cu microcontroler ATmega2560” disponibil la : https://rees52.com/arduino-boards/2-aa004 , accesat la data de : 28.05.2020.

[8] “Foaie de catalog SN74HC165N – NXP” disponibil la : https://assets.nexperia.com/ documents/data-sheet/74HC_HCT165.pdf , accesat la data de: 29.11.2019.

[9] “Conectarea dispozitivelor ce comunica prin protocolul SPI” disponibil la : http://andrei.clubcisco.ro/cursuri/3pm/lab6.pdf , accesat la data de : 30.05.2020.

[10] “Tutorial – protocol SPI” disponibil la : https://www.corelis.com/education/tutorials/spi-tutorial/ , accesat la data de : 30.05.2020.

[11] “Foaie de catalog WS2812C – Worldsemi” disponibil la : https://datasheet.lcsc.com/ szlcsc/1810231210_Worldsemi-WS2812C_C114587.pdf , accesat la data de : 02.12.2019.

[12] “Foaie de catalog MAX7219 – Maxim Integrated” https://html.alldatasheet.com/html-pdf/73745/MAXIM/MAX7219/126/1/MAX7219.html, accesat la data de : 30.11.2019.

[13] “Prezentarea sursei de tensiune SP-50-5V” disponibil la: https://www.optimusdigital.ro/ro/surse-ac-dc-de-5-v/1954-sursa-de-tensiune-in-comutaie-5v-10a-50-w.html , accesat la data de : 05.12.2019.

[14] “Prezentare magneţi disc din neodim” disponibil la: https://craftup.ro/magnet-neodim-disc-50-buc/?attribute_pa_marime=9x1mm&utm_source=Google%20Shopping&utm_campaign=Google%20Shopping%20-%20iunie1-_2020&utm_medium_=cpc&utm_term=5219&gclid=_CjwKCAjw2uf2BRBpEiwA31VZjx3n50V1lqvb6uoYV19tXXw-2CiGlJh_whDlQg_c0YV96wv6_KQpWY_BoCwyQQ_AvD_BwE , accesat la data de: 03.06.2020.

[15] “Tabla de sah ce incorporeaza motorul de analiză StockFish” disponibil la : https://create.arduino.cc/projecthub/Maxchess/wooden-chess-board-with-piece-recognition-872ffb?_ref=user&ref_id=46662&offset=1 , accesat la data de : 09.06.2020.

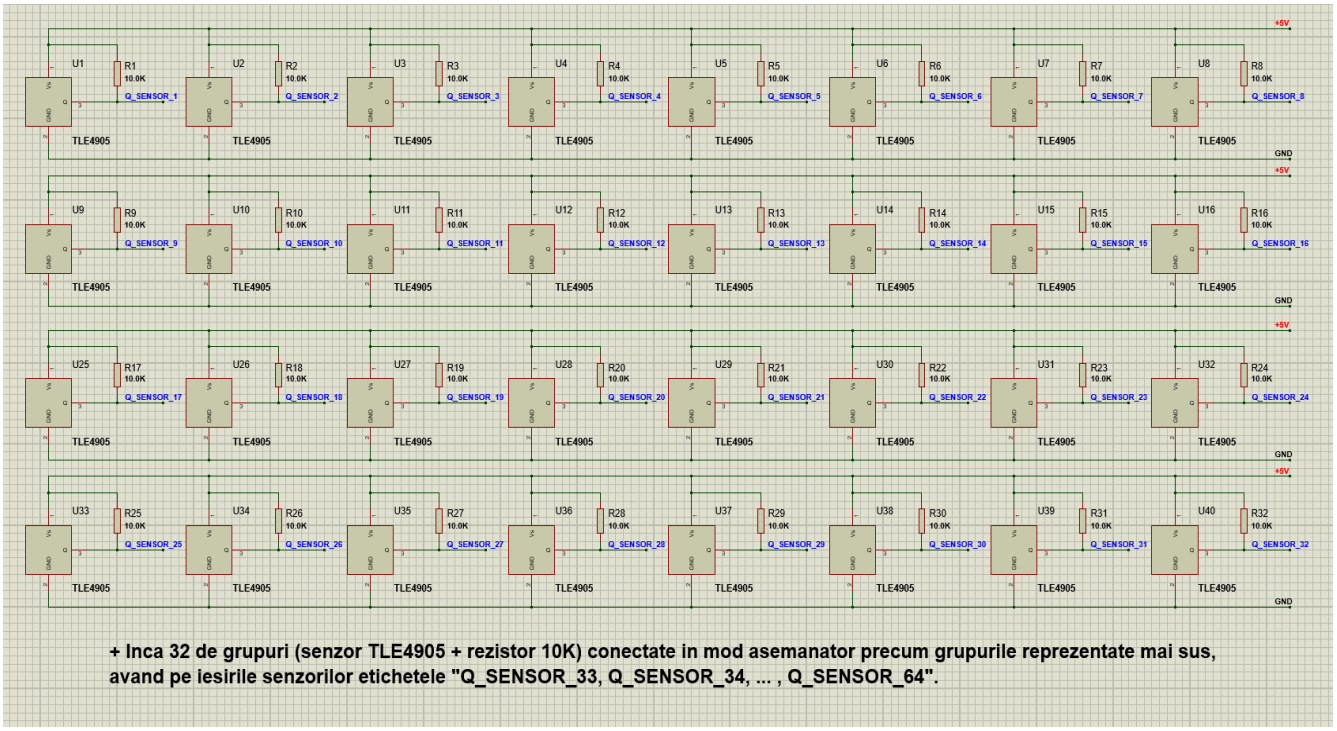
[16] “Prezentarea cubului electronic GoCube” disponibil la : https://getgocube.com/_products/gocube/?2 , accesat la data de: 09.06.2020.

[17] “Prezentarea zarului electronic GoDice” disponibil la : https://www.kickstarter.com/projects/1928372437/godice-your-favorite-dice-games-reimagined , accesat la data de: 10.06.2020.

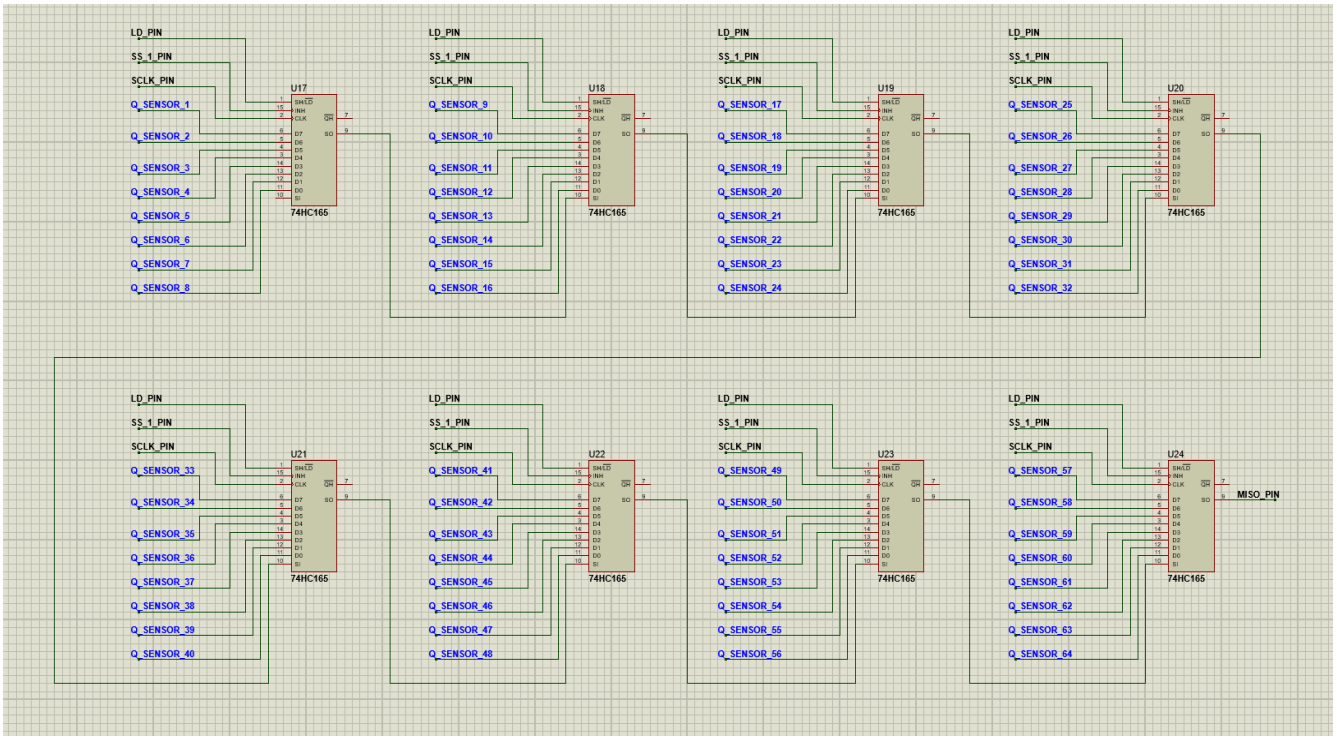
[18] “Biblioteca “light_ws2812” pentru controlul LED-urilor WS2812C” disponibil la: https://github.com/cpldcpu/light_ws2812 , accesat la data de : 12.03.2020.

Anexe

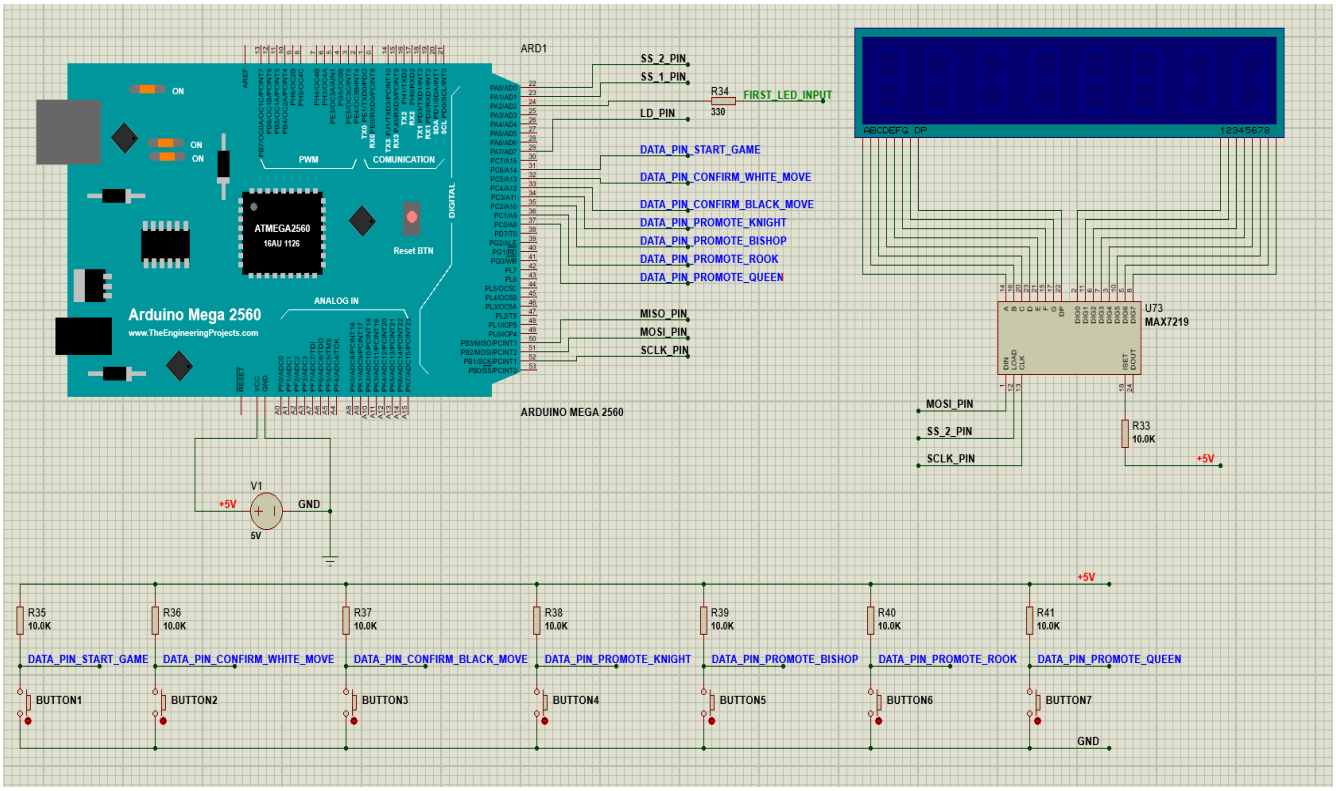
Anexa 1 – Schema hardware a tablei de şah inteligente



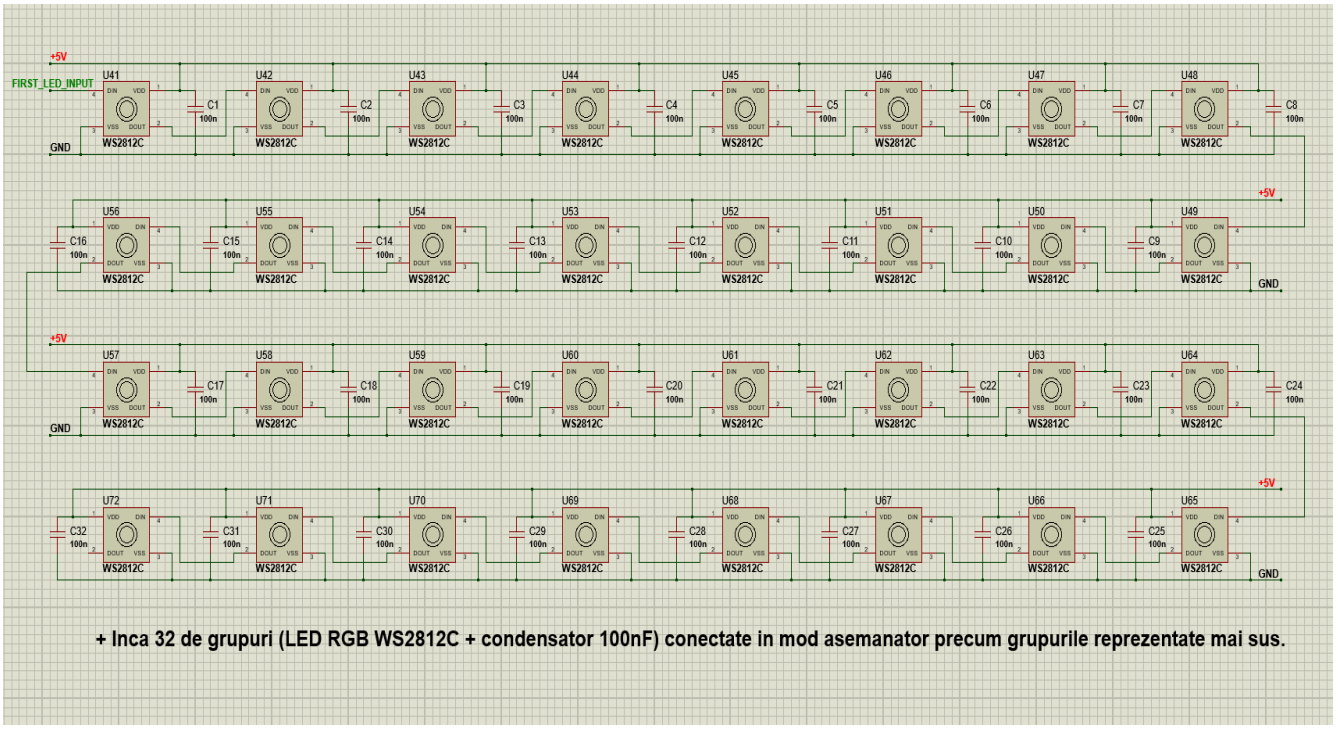
Conectarea celor 64 de senzori TLE4905



Conectarea celor 8 registre de deplasare SN74HC165N

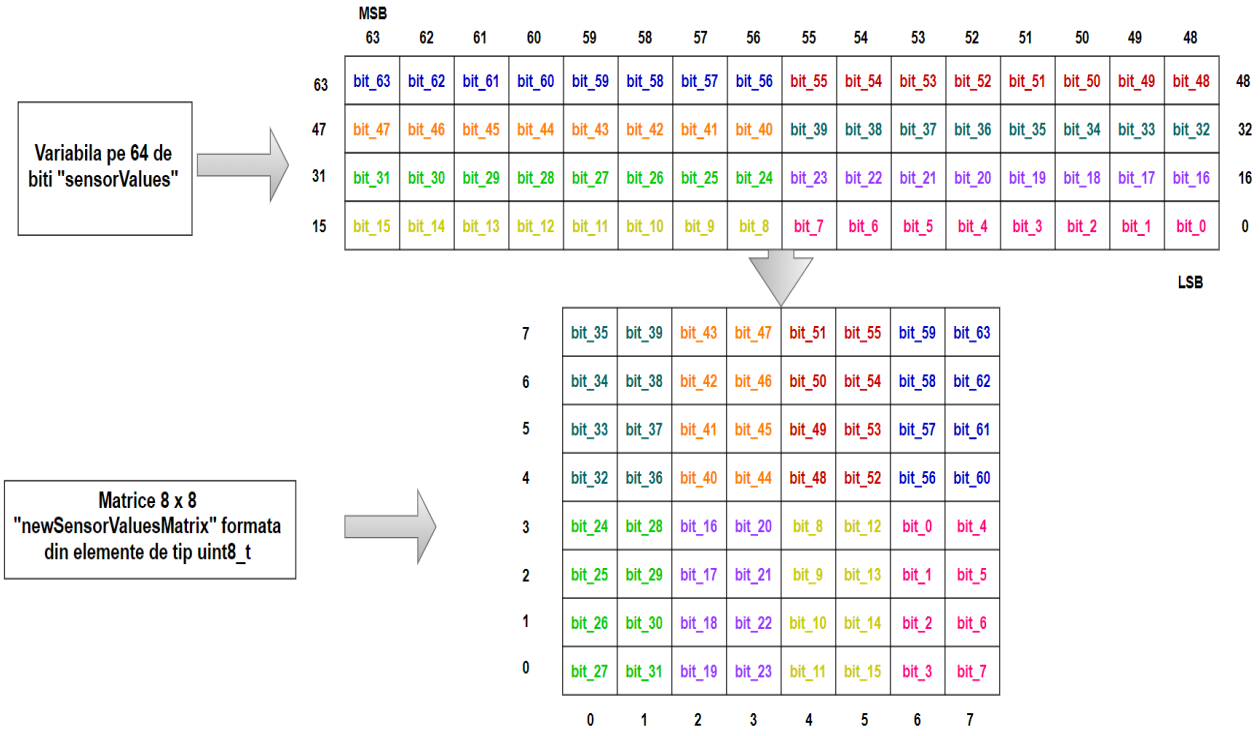


Conectarea plăcii de dezvoltare, a sursei de tensiune, a butoanelor şi a afişajului cu şapte segmente



Conectarea celor 64 de LED-uri RGB WS2812C

Anexa 2 – Diagrama de decodare a poziţiilor senzorilor magnetici de pe tabla de şah



Anexa 3 – Funcţii pentru implementarea protocolui SPI şi configurarea timer-ului

```
/**
   * File name: useful_functions.c
   * File type: source code file (.c)
*/

#include "useful_functions.h"

/*-------------------------------------------------------------------------------------------------
-----*/
/* 2) MAX7219 & 7-segment display - functions (implement SPI protocol): */
/* 2.2) */
void SPI_masterInit(void)
{
	DDR_SPI_extension |= ((1 << DDR_SS_transmit) | (1 << DDR_SS_receive_control));	/* Set
master SS_transmit and SS_receive_control as output */
	DDR_SPI |= ((1 << DDR_SS_receive) | (1 << DDR_SCK) | (1 << DDR_MOSI));		/* Set
SCK and MOSI as output, SS_receive and MISO as input */
	PORT_SPI_extension |= ((1 << PIN_SS_transmit) | (1 << PIN_SS_receive_control));	/*
Disable master SS_transmit and SS_receive_control pins (both active low) */
	SPCR |= (1 << SPE) | (1 << MSTR) | (1 << SPR0);				/*
Enable SPI, configure ATmega2560 as master device, set the clock rate at f_sys_ck/16 */
	SPCR |= (1 << CPOL);
	SPCR &= ~(1 << CPHA);
	/* Take data on the rising edge of clk(SPI data format), data order is MSB first */
}

/* 2.3) */
void MAX7219_SPI_masterTransmit(uint8_t address_SPI, uint8_t data_SPI)
{
	PORT_SPI_extension &= ~(1 << PIN_SS_transmit);
/* Start SPI transmission, master pull SS low */
	SPDR = address_SPI;							/* Transmit address
byte */
	while(!(SPSR & (1 << SPIF)));						/* Wait for
transmission complete */
	SPDR = data_SPI;							/* Transmit data
byte */
	while(!(SPSR & (1 << SPIF)));						/* Wait for
transmission complete */
	PORT_SPI_extension |= (1 << PIN_SS_transmit);
/* End SPI transmission, master pull SS high(other devices can be the master device) */
}
/*-------------------------------------------------------------------------------------------------
-----*/

/*-------------------------------------------------------------------------------------------------
-----*/
/* 3) SN74HC165N Parallel input - serial output (PISO) shift register - data types and functions:
*/
/* 3.4) */
char SN74HC165_SPI_masterReceive(void)
{
	char SPI_data_received;
	PORT_SPI_extension &= ~(1 << PIN_SS_receive_control);			/* Start SPI
reception, master pull SS low */
	/* (emulating this command because the slave device does not have a dedicated SPI
interface) */
	SPDR = 0x00;								/* Send dummy data
to take the received useful data */
	while(!(SPSR & (1 << SPIF)));						/* Wait for
transmission complete */
	SPI_data_received = SPDR;						/* Temporary
storage of the received data */
	PORT_SPI_extension |= (1 << PIN_SS_receive_control);			/* End SPI
reception, master pull SS high  */
	return SPI_data_received;						/* Return received
data */
}
```

```
/* 3.5) */
uint64_t SN74HC165_SPI_masterReceive_eightBytes(void)
{
	/* Initialize local variables */
	char receivedByte_SPI = 0x00;
	uint64_t receivedSensorData_SPI = 0x0000000000000000;

	/* Load the eight bytes read from the sensors into the 8 x SN74HC165 shift registers */
	SN74HC165_loadBytes();

	/* Read eight bytes from sensors through SPI and store them in the 64-bit variable
receivedSensorData_SPI. */
	receivedByte_SPI = SN74HC165_SPI_masterReceive(); /* Read the FIRST BYTE */
	receivedSensorData_SPI = (((uint64_t)receivedByte_SPI) << (64 - 8)) |
(receivedSensorData_SPI & 0x00FFFFFFFFFFFFFF); /* Apply 0x00FF_FFFF_FFFF_FFFF mask to CLEAR the
FIRST BYTE from receivedSensorData_SPI, replace that byte with receivedByte_SPI */
	_delay_ms(1);

	receivedByte_SPI = SN74HC165_SPI_masterReceive(); /* Read the SECOND BYTE */
	receivedSensorData_SPI = (((uint64_t)receivedByte_SPI) << (64 - 16)) |
(receivedSensorData_SPI & 0xFF00FFFFFFFFFFFF); /* Apply 0xFF00_FFFF_FFFF_FFFF mask to CLEAR the
SECOND BYTE from receivedSensorData_SPI, replace that byte with receivedByte_SPI */
	_delay_ms(1);

	receivedByte_SPI = SN74HC165_SPI_masterReceive(); /* Read the THIRD BYTE */
	receivedSensorData_SPI = (((uint64_t)receivedByte_SPI) << (64 - 24)) |
(receivedSensorData_SPI & 0xFFFF00FFFFFFFFFF); /* Apply 0xFFFF_00FF_FFFF_FFFF mask to CLEAR the
THIRD BYTE from receivedSensorData_SPI, replace that byte with receivedByte_SPI */
	_delay_ms(1);

	receivedByte_SPI = SN74HC165_SPI_masterReceive(); /* Read the FOURTH BYTE */
	receivedSensorData_SPI = (((uint64_t)receivedByte_SPI) << (64 - 32)) |
(receivedSensorData_SPI & 0xFFFFFF00FFFFFFFF); /* Apply 0xFFFF_FF00_FFFF_FFFF mask to CLEAR the
FOURTH BYTE from receivedSensorData_SPI, replace that byte with receivedByte_SPI */
	_delay_ms(1);

	receivedByte_SPI = SN74HC165_SPI_masterReceive(); /* Read the FIFTH BYTE */
	receivedSensorData_SPI = (((uint64_t)receivedByte_SPI) << (64 - 40)) |
(receivedSensorData_SPI & 0xFFFFFFFF00FFFFFF); /* Apply 0xFFFF_FFFF_00FF_FFFF mask to CLEAR the
FIFTH BYTE from receivedSensorData_SPI, replace that byte with receivedByte_SPI */
	_delay_ms(1);

	receivedByte_SPI = SN74HC165_SPI_masterReceive(); /* Read the SIXTH BYTE */
	receivedSensorData_SPI = (((uint64_t)receivedByte_SPI) << (64 - 48)) |
(receivedSensorData_SPI & 0xFFFFFFFFFF00FFFF); /* Apply 0xFFFF_FFFF_FF00_FFFF mask to CLEAR the
SIXTH BYTE from receivedSensorData_SPI, replace that byte with receivedByte_SPI */
	_delay_ms(1);

	receivedByte_SPI = SN74HC165_SPI_masterReceive(); /* Read the SEVENTH BYTE */
	receivedSensorData_SPI = (((uint64_t)receivedByte_SPI) << (64 - 56)) |
(receivedSensorData_SPI & 0xFFFFFFFFFFFF00FF); /* Apply 0xFFFF_FFFF_FFFF_00FF mask to CLEAR the
SEVENTH BYTE from receivedSensorData_SPI, replace that byte with receivedByte_SPI */
	_delay_ms(1);

	receivedByte_SPI = SN74HC165_SPI_masterReceive(); /* Read the EIGHTH BYTE */
	receivedSensorData_SPI = (((uint64_t)receivedByte_SPI) << (64 - 64)) |
(receivedSensorData_SPI & 0xFFFFFFFFFFFFFF00); /* Apply 0xFFFF_FFFF_FFFF_FF00 mask to CLEAR the
EIGHTH BYTE from receivedSensorData_SPI, replace that byte with receivedByte_SPI */
	_delay_ms(1);

	/* Returns the 64-bit variable that represents the read values from the 64 sensors. */
	return receivedSensorData_SPI;
}

/*-----------------------------------------------------------------------------------------------
-----*/
```

```
/*-------------------------------------------------------------------------------------------------
-----*/
/* 6) Timer1 and chess clock implementation - data types and functions: */
/* 6.2) */
void ATMEGA2560_TIMER1_init(void)
{
	TCCR1B |= (1 << WGM12); /* Configure timer1 in CTC mode */
	TCCR1B &= ~((1 << CS12) | (1 << CS11) | (1 << CS10)); /* Configure timer1 clock to no
clock source in order to disable timer1 */
	TCNT1 = 0; /* Initialize timer1 register. */
	OCR1A = 19999; /* Configure timer1 output compare match A register with the converted
value */
	TIMSK1 |= (1 << OCIE1A); /* Enable output compare match A interrupt */
}
/* 6.3) */
void ATMEGA2560_TIMER1_start(void)
{
	TCCR1B |= (1 << CS11); /* Configure timer1 with internal clock and prescaler = 8 in order
to enable timer1 */
}
/* 6.4) */
void ATMEGA2560_TIMER1_stop(void)
{
	TCCR1B &= ~((1 << CS12) | (1 << CS11) | (1 << CS10)); /* Configure timer1 clock to no
clock source in order to disable timer1 */
}
/* 6.5) */
void MAX7219_SPI_displayWhiteTime(uint8_t main_global_white_minutes, uint8_t
main_global_white_seconds)
{
	MAX7219_SPI_masterTransmit(0x05, (main_global_white_seconds % 10));
	MAX7219_SPI_masterTransmit(0x06, ((main_global_white_seconds / 10) % 10));
	MAX7219_SPI_masterTransmit(0x07, (0x80 | (main_global_white_minutes % 10)));
	MAX7219_SPI_masterTransmit(0x08, ((main_global_white_minutes / 10) % 10));
}
/* 6.6) */
void MAX7219_SPI_displayBlackTime(uint8_t main_global_black_minutes, uint8_t
main_global_black_seconds)
{
	MAX7219_SPI_masterTransmit(0x01, (main_global_black_seconds % 10));
	MAX7219_SPI_masterTransmit(0x02, ((main_global_black_seconds / 10) % 10));
	MAX7219_SPI_masterTransmit(0x03, (0x80 | (main_global_black_minutes % 10)));
	MAX7219_SPI_masterTransmit(0x04, ((main_global_black_minutes / 10) % 10));
}
/*-------------------------------------------------------------------------------------------------
-----*/
```

Anexa 4 – Funcţie pentru aprinderea LED-urilor la ridicarea pionului

```
/**
   * File name: chess_game_rules_functions.c
   * File type: source code file (.c)
*/

/* Functie de detectare (valori stocate in currentPiecePossibleMove si newLedMatrix) a campurilor
posibile pe care poate fi mutat un pion. */
void setLedsForPossibleMovesPawn(uint8_t local_currentPieceMove, uint8_t local_currentPieceRow,
uint8_t local_currentPieceColumn)
{
	if (local_currentPieceMove == WHITE_PAWN)
	{
		if ((local_currentPieceRow >= 1) && (local_currentPieceRow < 7))
		{
			/* Put the white pawn back on its current position.*/

	currentPiecePossibleMove.values[local_currentPieceRow][local_currentPieceColumn] =
103;//set newLedMatrix[local_currentPieceRow][local_currentPieceColumn];
			newLedMatrix.values[local_currentPieceRow][local_currentPieceColumn] =
103;

			/* Move the white pawn forward with one field.*/
			if ((local_currentPieceRow + 1) < NEWSENSORVALUESMATRIX_NR_ROWS)
			{
				/* Check that no other pieces occupies the front field. */
				if (piecesPositionMatrix.values[local_currentPieceRow +
1][local_currentPieceColumn] == NO_PIECE)
				{
					currentPiecePossibleMove.values[local_currentPieceRow +
1][local_currentPieceColumn] = 103;//set newLedMatrix[local_currentPieceRow +
1][local_currentPieceColumn];
					newLedMatrix.values[local_currentPieceRow +
1][local_currentPieceColumn] = 103;
				}
			}

			/* Move the white pawn forward with two fields.*/
			if ((local_currentPieceRow + 2) < NEWSENSORVALUESMATRIX_NR_ROWS)
			{
				/* Check that the pawn is in the starting position. */
				/* Check that no other pieces occupies the front field. */
				if ((local_currentPieceRow == 1) &&
(piecesPositionMatrix.values[local_currentPieceRow + 2][local_currentPieceColumn] == NO_PIECE))
				{
					currentPiecePossibleMove.values[local_currentPieceRow +
2][local_currentPieceColumn] = 103;//set newLedMatrix[local_currentPieceRow +
2][local_currentPieceColumn];
					newLedMatrix.values[local_currentPieceRow +
2][local_currentPieceColumn] = 103;
				}
			}

			/* Captures a black piece on the diagonal on the left side. */
			if (((local_currentPieceRow + 1) < NEWSENSORVALUESMATRIX_NR_ROWS) &&
((local_currentPieceColumn - 1) >= 0))
			{
				if (((piecesPositionMatrix.values[local_currentPieceRow +
1][local_currentPieceColumn - 1] >= BLACK_PAWN) &&
					(piecesPositionMatrix.values[local_currentPieceRow +
1][local_currentPieceColumn - 1] <= BLACK_KING)))
				{
					currentPiecePossibleMove.values[local_currentPieceRow +
1][local_currentPieceColumn - 1] = 1;//set newLedMatrix[local_currentPieceRow +
1][local_currentPieceColumn - 1]
					newLedMatrix.values[local_currentPieceRow +
1][local_currentPieceColumn - 1] = 1;
				}
				else
				{
					if (((piecesPositionMatrix.values[local_currentPieceRow +
1][local_currentPieceColumn - 1] >= WHITE_PAWN) &&
```

```
(piecesPositionMatrix.values[local_currentPieceRow + 1][local_currentPieceColumn - 1] <=
WHITE_QUEEN)))
                                        {
                                                /* Protects an allied piece. */

        currentPiecePossibleMove.values[local_currentPieceRow + 1][local_currentPieceColumn - 1]
= 1;//set newLedMatrix[local_currentPieceRow + 1][local_currentPieceColumn - 1]
                                        }
                                        else
                                        {
                                                if
(piecesPositionMatrix.values[local_currentPieceRow + 1][local_currentPieceColumn - 1] ==
WHITE_KING)
                                                {
                                                        /* Do nothing. */
                                                }
                                        }
                                }
                        }

                        /* Captures a black piece on the diagonal on the right side. */
                        if (((local_currentPieceRow + 1) < NEWSENSORVALUESMATRIX_NR_ROWS) &&
((local_currentPieceColumn + 1) < NEWSENSORVALUESMATRIX_NR_COLUMNS))
                        {

                                if (((piecesPositionMatrix.values[local_currentPieceRow +
1][local_currentPieceColumn + 1] >= BLACK_PAWN) &&
                                        (piecesPositionMatrix.values[local_currentPieceRow +
1][local_currentPieceColumn + 1] <= BLACK_KING)))
                                {
                                        currentPiecePossibleMove.values[local_currentPieceRow +
1][local_currentPieceColumn + 1] = 1;//set newLedMatrix[local_currentPieceRow +
1][local_currentPieceColumn + 1]
                                        newLedMatrix.values[local_currentPieceRow +
1][local_currentPieceColumn + 1] = 1;
                                }
                                else
                                {
                                        if (((piecesPositionMatrix.values[local_currentPieceRow +
1][local_currentPieceColumn + 1] >= WHITE_PAWN) &&
                                                (piecesPositionMatrix.values[local_currentPieceRow +
1][local_currentPieceColumn + 1] <= WHITE_QUEEN)))
                                        {
                                                /* Protects an allied piece. */

        currentPiecePossibleMove.values[local_currentPieceRow + 1][local_currentPieceColumn + 1]
= 1;//set newLedMatrix[local_currentPieceRow + 1][local_currentPieceColumn - 1]
                                        }
                                        else
                                        {
                                                if
(piecesPositionMatrix.values[local_currentPieceRow + 1][local_currentPieceColumn + 1] ==
WHITE_KING)
                                                {
                                                        /* Do nothing. */
                                                }
                                        }
                                }
                        }
                }
        }
        else
        {
                /* Case: local_currentPieceMove == BLACK_PAWN. */

                if ((local_currentPieceRow >= 1) && (local_currentPieceRow < 7))
                {
                        /* Put the black pawn back on its current position.*/

        currentPiecePossibleMove.values[local_currentPieceRow][local_currentPieceColumn] =
103;//set newLedMatrix[local_currentPieceRow][local_currentPieceColumn];
```

```
newLedMatrix.values[local_currentPieceRow][local_currentPieceColumn] = 103;

                        /* Move the black pawn forward with one field.*/
                        if ((local_currentPieceRow - 1) >= 0)
                        {
                                /* Check that no other pieces occupies the front field. */
                                if (piecesPositionMatrix.values[local_currentPieceRow -
1][local_currentPieceColumn] == NO_PIECE)
                                {
                                        currentPiecePossibleMove.values[local_currentPieceRow -
1][local_currentPieceColumn] = 103;//set newLedMatrix[local_currentPieceRow -
1][local_currentPieceColumn];
                                        newLedMatrix.values[local_currentPieceRow -
1][local_currentPieceColumn] = 103;
                                }
                        }

                        /* Move the black pawn forward with two fields.*/
                        if ((local_currentPieceRow - 2) >= 0)
                        {
                                /* Check that the pawn is in the starting position. */
                                /* Check that no other pieces occupies the front field. */
                                if ((local_currentPieceRow == 6) &&
(piecesPositionMatrix.values[local_currentPieceRow - 2][local_currentPieceColumn] == NO_PIECE))
                                {
                                        currentPiecePossibleMove.values[local_currentPieceRow -
2][local_currentPieceColumn] = 103;//set newLedMatrix[local_currentPieceRow -
2][local_currentPieceColumn];
                                        newLedMatrix.values[local_currentPieceRow -
2][local_currentPieceColumn] = 103;
                                }
                        }

                        /* Captures a white piece on the diagonal on the left side. */
                        if (((local_currentPieceRow - 1) >= 0) && ((local_currentPieceColumn - 1)
>= 0))
                        {
                                if (((piecesPositionMatrix.values[local_currentPieceRow -
1][local_currentPieceColumn - 1] >= WHITE_PAWN) &&
                                        (piecesPositionMatrix.values[local_currentPieceRow -
1][local_currentPieceColumn - 1] <= WHITE_KING)))
                                {
                                        currentPiecePossibleMove.values[local_currentPieceRow -
1][local_currentPieceColumn - 1] = 1;//set newLedMatrix[local_currentPieceRow -
1][local_currentPieceColumn - 1]
                                        newLedMatrix.values[local_currentPieceRow -
1][local_currentPieceColumn - 1] = 1;
                                }
                                else
                                {
                                        if (((piecesPositionMatrix.values[local_currentPieceRow -
1][local_currentPieceColumn - 1] >= BLACK_PAWN) &&
                                                (piecesPositionMatrix.values[local_currentPieceRow -
1][local_currentPieceColumn - 1] <= BLACK_QUEEN)))
                                        {
                                                /* Protects an allied piece. */

        currentPiecePossibleMove.values[local_currentPieceRow - 1][local_currentPieceColumn - 1]
= 1;//set newLedMatrix[local_currentPieceRow + 1][local_currentPieceColumn - 1]
                                        }
                                        else
                                        {
                                                if
(piecesPositionMatrix.values[local_currentPieceRow - 1][local_currentPieceColumn - 1] ==
BLACK_KING)
                                                {
                                                        /* Do nothing. */
                                                }
                                        }
                                }
```

```
}

                        /* Captures a white piece on the diagonal on the right side. */
                        if (((local_currentPieceRow - 1) >= 0) && ((local_currentPieceColumn + 1)
< NEWSENSORVALUESMATRIX_NR_COLUMNS))
                        {
                                if (((piecesPositionMatrix.values[local_currentPieceRow -
1][local_currentPieceColumn + 1] >= WHITE_PAWN) &&
                                        (piecesPositionMatrix.values[local_currentPieceRow -
1][local_currentPieceColumn + 1] <= WHITE_KING)))
                                {
                                        currentPiecePossibleMove.values[local_currentPieceRow -
1][local_currentPieceColumn + 1] = 1;//set newLedMatrix[local_currentPieceRow -
1][local_currentPieceColumn + 1]
                                        newLedMatrix.values[local_currentPieceRow -
1][local_currentPieceColumn + 1] = 1;
                                }
                                else
                                {
                                        if (((piecesPositionMatrix.values[local_currentPieceRow -
1][local_currentPieceColumn + 1] >= BLACK_PAWN) &&
                                                (piecesPositionMatrix.values[local_currentPieceRow -
1][local_currentPieceColumn + 1] <= BLACK_QUEEN)))
                                        {
                                                /* Protects an allied piece. */

        currentPiecePossibleMove.values[local_currentPieceRow - 1][local_currentPieceColumn + 1]
= 1;//set newLedMatrix[local_currentPieceRow + 1][local_currentPieceColumn - 1]
                                        }
                                        else
                                        {
                                                if
(piecesPositionMatrix.values[local_currentPieceRow - 1][local_currentPieceColumn + 1] ==
BLACK_KING)
                                                {
                                                        /* Do nothing. */
                                                }
                                        }
                                }
                        }
                }
        }
}
```

Anexa 5 – Funcţie pentru aprinderea LED-urilor la ridicarea nebunului

```
/* Functie de detectare (valori stocate in currentPiecePossibleMove si newLedMatrix) a campurilor
posibile pe care poate fi mutat un nebun. */
void setLedsForPossibleMovesBishop(uint8_t local_currentPieceMove, uint8_t local_currentPieceRow,
uint8_t local_currentPieceColumn)
{
	if ((local_currentPieceMove == WHITE_BISHOP) || (local_currentPieceMove == WHITE_QUEEN))
	{
		for (int8_t i = 0; ((i <= local_currentPieceRow) && (i <=
local_currentPieceColumn)); i++)
		{
			if (i == 0)
			{
				currentPiecePossibleMove.values[local_currentPieceRow -
i][local_currentPieceColumn - i] = 103;
				newLedMatrix.values[local_currentPieceRow -
i][local_currentPieceColumn - i] = 103;
			}
			else
			{
				if ((piecesPositionMatrix.values[local_currentPieceRow -
i][local_currentPieceColumn - i] >= BLACK_PAWN) &&
					(piecesPositionMatrix.values[local_currentPieceRow -
i][local_currentPieceColumn - i] <= BLACK_KING))
				{
					currentPiecePossibleMove.values[local_currentPieceRow -
i][local_currentPieceColumn - i] = 1;//set newLedMatrix[local_currentPieceRow -
i][local_currentPieceColumn - i]
					newLedMatrix.values[local_currentPieceRow -
i][local_currentPieceColumn - i] = 1;
					if (piecesPositionMatrix.values[local_currentPieceRow -
i][local_currentPieceColumn - i] != BLACK_KING)//Check
					{
						break;
					}
				}
				else
				{
					if ((piecesPositionMatrix.values[local_currentPieceRow -
i][local_currentPieceColumn - i] >= WHITE_PAWN) &&
						(piecesPositionMatrix.values[local_currentPieceRow -
i][local_currentPieceColumn - i] <= WHITE_QUEEN))
					{
						/* Protects an allied piece. */

	currentPiecePossibleMove.values[local_currentPieceRow - i][local_currentPieceColumn - i]
= 1;
						break;
					}
					else
					{
						if
(piecesPositionMatrix.values[local_currentPieceRow - i][local_currentPieceColumn - i] ==
WHITE_KING)
						{
							/* Do nothing. */
						}
						else
						{

	currentPiecePossibleMove.values[local_currentPieceRow - i][local_currentPieceColumn - i]
= 1;
							newLedMatrix.values[local_currentPieceRow -
i][local_currentPieceColumn - i] = 1;
						}
					}
				}
			}
		}

		for (int8_t i = 0; ((i <= local_currentPieceRow) && (i <
NEWSENSORVALUESMATRIX_NR_COLUMNS - local_currentPieceColumn)); i++)
```

```
{
                        if (i == 0)
                        {
                                currentPiecePossibleMove.values[local_currentPieceRow -
i][local_currentPieceColumn + i] = 103;
                                newLedMatrix.values[local_currentPieceRow -
i][local_currentPieceColumn + i] = 103;
                        }
                        else
                        {
                                if ((piecesPositionMatrix.values[local_currentPieceRow -
i][local_currentPieceColumn + i] >= BLACK_PAWN) &&
                                        (piecesPositionMatrix.values[local_currentPieceRow -
i][local_currentPieceColumn + i] <= BLACK_KING))
                                {
                                        currentPiecePossibleMove.values[local_currentPieceRow -
i][local_currentPieceColumn + i] = 1;//set newLedMatrix[local_currentPieceRow -
i][local_currentPieceColumn + i]
                                        newLedMatrix.values[local_currentPieceRow -
i][local_currentPieceColumn + i] = 1;
                                        if (piecesPositionMatrix.values[local_currentPieceRow -
i][local_currentPieceColumn + i] != BLACK_KING)//Check
                                        {
                                                break;
                                        }
                                }
                                else
                                {
                                        if ((piecesPositionMatrix.values[local_currentPieceRow -
i][local_currentPieceColumn + i] >= WHITE_PAWN) &&
                                                (piecesPositionMatrix.values[local_currentPieceRow -
i][local_currentPieceColumn + i] <= WHITE_QUEEN))
                                        {
                                                /* Protects an allied piece. */

        currentPiecePossibleMove.values[local_currentPieceRow - i][local_currentPieceColumn + i]
= 1;
                                                break;
                                        }
                                        else
                                        {
                                                if
(piecesPositionMatrix.values[local_currentPieceRow - i][local_currentPieceColumn + i] ==
WHITE_KING)
                                                {
                                                        /* Do nothing. */
                                                }
                                                else
                                                {

        currentPiecePossibleMove.values[local_currentPieceRow - i][local_currentPieceColumn + i]
= 1;
                                                        newLedMatrix.values[local_currentPieceRow -
i][local_currentPieceColumn + i] = 1;
                                                }
                                        }
                                }
                        }
                }

                for (int8_t i = 0; ((i < NEWSENSORVALUESMATRIX_NR_ROWS - local_currentPieceRow) &&
(i <= local_currentPieceColumn)); i++)
                {
                        if (i == 0)
                        {
                                currentPiecePossibleMove.values[local_currentPieceRow +
i][local_currentPieceColumn - i] = 103;
                                newLedMatrix.values[local_currentPieceRow +
i][local_currentPieceColumn - i] = 103;
                        }
                        else
```

```
{
                                if ((piecesPositionMatrix.values[local_currentPieceRow +
i][local_currentPieceColumn - i] >= BLACK_PAWN) &&
                                        (piecesPositionMatrix.values[local_currentPieceRow +
i][local_currentPieceColumn - i] <= BLACK_KING))
                                {
                                        currentPiecePossibleMove.values[local_currentPieceRow +
i][local_currentPieceColumn - i] = 1;//set newLedMatrix[local_currentPieceRow +
i][local_currentPieceColumn - i]
                                        newLedMatrix.values[local_currentPieceRow +
i][local_currentPieceColumn - i] = 1;
                                        if (piecesPositionMatrix.values[local_currentPieceRow +
i][local_currentPieceColumn - i] != BLACK_KING)//Check
                                        {
                                                break;
                                        }
                                }
                                else
                                {
                                        if ((piecesPositionMatrix.values[local_currentPieceRow +
i][local_currentPieceColumn - i] >= WHITE_PAWN) &&
                                                (piecesPositionMatrix.values[local_currentPieceRow +
i][local_currentPieceColumn - i] <= WHITE_QUEEN))
                                        {
                                                /* Protects an allied piece. */

        currentPiecePossibleMove.values[local_currentPieceRow + i][local_currentPieceColumn - i]
= 1;
                                                break;
                                        }
                                        else
                                        {
                                                if
(piecesPositionMatrix.values[local_currentPieceRow + i][local_currentPieceColumn - i] ==
WHITE_KING)
                                                {
                                                        /* Do nothing. */
                                                }
                                                else
                                                {

        currentPiecePossibleMove.values[local_currentPieceRow + i][local_currentPieceColumn - i]
= 1;
                                                        newLedMatrix.values[local_currentPieceRow +
i][local_currentPieceColumn - i] = 1;
                                                }
                                        }
                                }
                        }
                }

                for (int8_t i = 0; ((i < NEWSENSORVALUESMATRIX_NR_ROWS - local_currentPieceRow) &&
(i < NEWSENSORVALUESMATRIX_NR_COLUMNS - local_currentPieceColumn)); i++)
                {
                        if (i == 0)
                        {
                                currentPiecePossibleMove.values[local_currentPieceRow +
i][local_currentPieceColumn + i] = 103;
                                newLedMatrix.values[local_currentPieceRow +
i][local_currentPieceColumn + i] = 103;
                        }
                        else
                        {
                                if ((piecesPositionMatrix.values[local_currentPieceRow +
i][local_currentPieceColumn + i] >= BLACK_PAWN) &&
                                        (piecesPositionMatrix.values[local_currentPieceRow +
i][local_currentPieceColumn + i] <= BLACK_KING))
                                {
                                        currentPiecePossibleMove.values[local_currentPieceRow +
i][local_currentPieceColumn + i] = 1;//set newLedMatrix[local_currentPieceRow +
i][local_currentPieceColumn + i]
```

```
                                        newLedMatrix.values[local_currentPieceRow +
i][local_currentPieceColumn + i] = 1;
                                        if (piecesPositionMatrix.values[local_currentPieceRow +
i][local_currentPieceColumn + i] != BLACK_KING)//Check
                                        {
                                                break;
                                        }
                                }
                                else
                                {
                                        if ((piecesPositionMatrix.values[local_currentPieceRow +
i][local_currentPieceColumn + i] >= WHITE_PAWN) &&
                                                (piecesPositionMatrix.values[local_currentPieceRow +
i][local_currentPieceColumn + i] <= WHITE_QUEEN))
                                        {
                                                /* Protects an allied piece. */

        currentPiecePossibleMove.values[local_currentPieceRow + i][local_currentPieceColumn + i]
= 1;
                                                break;
                                        }
                                        else
                                        {
                                                if
(piecesPositionMatrix.values[local_currentPieceRow + i][local_currentPieceColumn + i] ==
WHITE_KING)
                                                {
                                                        /* Do nothing. */
                                                }
                                                else
                                                {

        currentPiecePossibleMove.values[local_currentPieceRow + i][local_currentPieceColumn + i]
= 1;
                                                        newLedMatrix.values[local_currentPieceRow +
i][local_currentPieceColumn + i] = 1;
                                                }
                                        }
                                }
                        }
                }
        }
        else
        {
                /* Case: local_currentPieceMove == BLACK_BISHOP or local_currentPieceMove ==
BLACK_QUEEN. */
                if ((local_currentPieceMove == BLACK_BISHOP) || (local_currentPieceMove ==
BLACK_QUEEN))
                {
                        for (int8_t i = 0; ((i <= local_currentPieceRow) && (i <=
local_currentPieceColumn)); i++)
                        {
                                if (i == 0)
                                {
                                        currentPiecePossibleMove.values[local_currentPieceRow -
i][local_currentPieceColumn - i] = 103;
                                        newLedMatrix.values[local_currentPieceRow -
i][local_currentPieceColumn - i] = 103;
                                }
                                else
                                {
                                        if ((piecesPositionMatrix.values[local_currentPieceRow -
i][local_currentPieceColumn - i] >= WHITE_PAWN) &&
                                                (piecesPositionMatrix.values[local_currentPieceRow -
i][local_currentPieceColumn - i] <= WHITE_KING))
                                        {

        currentPiecePossibleMove.values[local_currentPieceRow - i][local_currentPieceColumn - i]
= 1;//set newLedMatrix[local_currentPieceRow - i][local_currentPieceColumn - i]
                                                newLedMatrix.values[local_currentPieceRow -
i][local_currentPieceColumn - i] = 1;
```

```
                                                if
(piecesPositionMatrix.values[local_currentPieceRow - i][local_currentPieceColumn - i] !=
WHITE_KING)//Check
                                                {
                                                        break;
                                                }
                                        }
                                        else
                                        {
                                                if
((piecesPositionMatrix.values[local_currentPieceRow - i][local_currentPieceColumn - i] >=
BLACK_PAWN) &&

        (piecesPositionMatrix.values[local_currentPieceRow - i][local_currentPieceColumn - i] <=
BLACK_QUEEN))
                                                {
                                                        /* Protects an allied piece. */

        currentPiecePossibleMove.values[local_currentPieceRow - i][local_currentPieceColumn - i]
= 1;
                                                        break;
                                                }
                                                else
                                                {
                                                        if
(piecesPositionMatrix.values[local_currentPieceRow - i][local_currentPieceColumn - i] ==
BLACK_KING)
                                                        {
                                                                /* Do nothing. */
                                                        }
                                                        else
                                                        {

        currentPiecePossibleMove.values[local_currentPieceRow - i][local_currentPieceColumn - i]
= 1;

        newLedMatrix.values[local_currentPieceRow - i][local_currentPieceColumn - i] = 1;
                                                        }
                                                }
                                        }
                                }
                        }

                        for (int8_t i = 0; ((i <= local_currentPieceRow) && (i <
NEWSENSORVALUESMATRIX_NR_COLUMNS - local_currentPieceColumn)); i++)
                        {
                                if (i == 0)
                                {
                                        currentPiecePossibleMove.values[local_currentPieceRow -
i][local_currentPieceColumn + i] = 103;
                                        newLedMatrix.values[local_currentPieceRow -
i][local_currentPieceColumn + i] = 103;
                                }
                                else
                                {
                                        if ((piecesPositionMatrix.values[local_currentPieceRow -
i][local_currentPieceColumn + i] >= WHITE_PAWN) &&
                                                (piecesPositionMatrix.values[local_currentPieceRow -
i][local_currentPieceColumn + i] <= WHITE_KING))
                                        {

        currentPiecePossibleMove.values[local_currentPieceRow - i][local_currentPieceColumn + i]
= 1;//set newLedMatrix[local_currentPieceRow - i][local_currentPieceColumn + i]
                                                newLedMatrix.values[local_currentPieceRow -
i][local_currentPieceColumn + i] = 1;
                                                if
(piecesPositionMatrix.values[local_currentPieceRow - i][local_currentPieceColumn + i] !=
WHITE_KING)//Check
                                                {
                                                        break;
                                                }
```

```
}
                                        else
                                        {
                                                if
((piecesPositionMatrix.values[local_currentPieceRow - i][local_currentPieceColumn + i] >=
BLACK_PAWN) &&

        (piecesPositionMatrix.values[local_currentPieceRow - i][local_currentPieceColumn + i] <=
BLACK_QUEEN))
                                                {
                                                        /* Protects an allied piece. */

        currentPiecePossibleMove.values[local_currentPieceRow - i][local_currentPieceColumn + i]
= 1;
                                                        break;
                                                }
                                                else
                                                {
                                                        if
(piecesPositionMatrix.values[local_currentPieceRow - i][local_currentPieceColumn + i] ==
BLACK_KING)
                                                        {
                                                                /* Do nothing. */
                                                        }
                                                        else
                                                        {

        currentPiecePossibleMove.values[local_currentPieceRow - i][local_currentPieceColumn + i]
= 1;

        newLedMatrix.values[local_currentPieceRow - i][local_currentPieceColumn + i] = 1;
                                                        }
                                                }
                                        }
                                }
                        }

                        for (int8_t i = 0; ((i < NEWSENSORVALUESMATRIX_NR_ROWS -
local_currentPieceRow) && (i <= local_currentPieceColumn)); i++)
                        {
                                if (i == 0)
                                {
                                        currentPiecePossibleMove.values[local_currentPieceRow +
i][local_currentPieceColumn - i] = 103;
                                        newLedMatrix.values[local_currentPieceRow +
i][local_currentPieceColumn - i] = 103;
                                }
                                else
                                {
                                        if ((piecesPositionMatrix.values[local_currentPieceRow +
i][local_currentPieceColumn - i] >= WHITE_PAWN) &&
                                                (piecesPositionMatrix.values[local_currentPieceRow +
i][local_currentPieceColumn - i] <= WHITE_KING))
                                        {

        currentPiecePossibleMove.values[local_currentPieceRow + i][local_currentPieceColumn - i]
= 1;//set newLedMatrix[local_currentPieceRow + i][local_currentPieceColumn - i]
                                                newLedMatrix.values[local_currentPieceRow +
i][local_currentPieceColumn - i] = 1;
                                                if
(piecesPositionMatrix.values[local_currentPieceRow + i][local_currentPieceColumn - i] !=
WHITE_KING)//Check
                                                {
                                                        break;
                                                }
                                        }
                                        else
                                        {
                                                if
((piecesPositionMatrix.values[local_currentPieceRow + i][local_currentPieceColumn - i] >=
BLACK_PAWN) &&
```

```
(piecesPositionMatrix.values[local_currentPieceRow + i][local_currentPieceColumn - i] <=
BLACK_QUEEN))
                                                {
                                                        /* Protects an allied piece. */

        currentPiecePossibleMove.values[local_currentPieceRow + i][local_currentPieceColumn - i]
= 1;
                                                        break;
                                                }
                                                else
                                                {
                                                        if
(piecesPositionMatrix.values[local_currentPieceRow + i][local_currentPieceColumn - i] ==
BLACK_KING)
                                                        {
                                                                /* Do nothing. */
                                                        }
                                                        else
                                                        {

        currentPiecePossibleMove.values[local_currentPieceRow + i][local_currentPieceColumn - i]
= 1;

        newLedMatrix.values[local_currentPieceRow + i][local_currentPieceColumn - i] = 1;
                                                        }
                                                }
                                        }
                                }
                        }

                        for (int8_t i = 0; ((i < NEWSENSORVALUESMATRIX_NR_ROWS -
local_currentPieceRow) && (i < NEWSENSORVALUESMATRIX_NR_COLUMNS - local_currentPieceColumn));
i++)
                        {
                                if (i == 0)
                                {
                                        currentPiecePossibleMove.values[local_currentPieceRow +
i][local_currentPieceColumn + i] = 103;
                                        newLedMatrix.values[local_currentPieceRow +
i][local_currentPieceColumn + i] = 103;
                                }
                                else
                                {
                                        if ((piecesPositionMatrix.values[local_currentPieceRow +
i][local_currentPieceColumn + i] >= WHITE_PAWN) &&
                                                (piecesPositionMatrix.values[local_currentPieceRow +
i][local_currentPieceColumn + i] <= WHITE_KING))
                                                {

        currentPiecePossibleMove.values[local_currentPieceRow + i][local_currentPieceColumn + i]
= 1;//set newLedMatrix[local_currentPieceRow + i][local_currentPieceColumn + i]
                                                newLedMatrix.values[local_currentPieceRow +
i][local_currentPieceColumn + i] = 1;
                                                if
(piecesPositionMatrix.values[local_currentPieceRow + i][local_currentPieceColumn + i] !=
WHITE_KING)//Check
                                                {
                                                        break;
                                                }
                                        }
                                        else
                                        {
                                                if
((piecesPositionMatrix.values[local_currentPieceRow + i][local_currentPieceColumn + i] >=
BLACK_PAWN) &&

        (piecesPositionMatrix.values[local_currentPieceRow + i][local_currentPieceColumn + i] <=
BLACK_QUEEN))
                                                {
                                                        /* Protects an allied piece. */
```

```
        currentPiecePossibleMove.values[local_currentPieceRow + i][local_currentPieceColumn + i]
= 1;
                                                        break;
                                                }
                                                else
                                                {
                                                        if
(piecesPositionMatrix.values[local_currentPieceRow + i][local_currentPieceColumn + i] ==
BLACK_KING)
                                                        {
                                                                /* Do nothing. */
                                                        }
                                                        else
                                                        {

        currentPiecePossibleMove.values[local_currentPieceRow + i][local_currentPieceColumn + i]
= 1;

        newLedMatrix.values[local_currentPieceRow + i][local_currentPieceColumn + i] = 1;
                                                        }
                                                }
                                        }
                                }
                        }
                }
        }
}
```

Anexa 6 – Accesarea codului complet al tablei de şah inteligente

Codul prezentat în cadrul anexelor 3, 4 şi 5, precum şi cel din capitolul 4 reprezintă fragmente din codul complet al aplicaţiei firmware. Întreaga aplicaţie cuprinde mai bine de 3500 de linii de cod şi ar fi ocupat un număr de aproximativ 100 de pagini în cadrul acestui document. Mai jos se introduce link-ul pentru vizualizarea întregului cod încărcat pe pagina web “Github”:

https://github.com/VictorMocanu/Tabla-de-sah-inteligenta