



SIMULATOR EMBEDDED PENTRU BORDUL UNUI AUTOTURISM FOLOSIND COMUNICAȚIA CAN

LUCRARE DE DIPLOMĂ

Autor: **Vlăduț-Alexandru DOBRĂ**

Conducător științific: **Prof. dr. ing. Ioan Nașcu**

2017



UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

Vizat,

DECAN
Prof.dr.ing. Liviu MICLEA

DIRECTOR DEPARTAMENT AUTOMATICĂ
Prof.dr.ing. Honoriu VĂLEAN

Autor: **Vlăduț-Alexandru DOBRĂ**

Simulator embedded pentru bordul unui autoturism folosind comunicația CAN

1. **Enunțul temei:** *Stabilirea unei conexiuni între două microcontrolere prin protocolul CAN și simularea unor indicatoare și afișaje dintr-un bord de mașină pe baza datelor din comunicația realizată.*
2. **Conținutul proiectului:** *Pagina de prezentare, declarația de autenticitate, sinteza, cuprinsul, introducerea, studiul teoretic, analiza proiectului, proiectare, implementare pe microcontrolere, concluzii, bibliografie.*
3. **Locul documentației:** *Universitatea Tehnică din Cluj-Napoca*
4. **Consultanți:** ing. Gabriel Harja
5. **Data emiterii temei:**
6. **Data predării:**

Semnătura autorului _____

Semnătura conducătorului științific _____



UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

**Declarație pe proprie răspundere privind
autenticitatea proiectului de diplomă**

Subsemnatul(a) **Vlăduț Alexandru Dobră**, legitimat cu CI/BI seria MM nr. 612405, CNP 1940607245027,

autorul lucrării: Simulator embedded pentru bordul unui autoturism folosind comunicația CAN elaborată în vederea susținerii examenului de finalizare a studiilor de licență

la **Facultatea de Automatică și Calculatoare**,
specializarea **Automatică și Informatică Aplicată**,
din cadrul Universității Tehnice din Cluj-Napoca,
sesiunea Iulie 2017 a anului universitar 2016-2017,

declar pe proprie răspundere, că această lucrare este rezultatul propriei activități intelectuale, pe baza cercetărilor mele și pe baza informațiilor obținute din surse care au fost citate, în textul lucrării, și în bibliografie.

Declar, că această lucrare nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române și a convențiilor internaționale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în fața unei alte comisii de examen de licență.

În cazul constatării ulterioare a unor declarații false, voi suporta sancțiunile administrative, respectiv, *anularea examenului de licență*.

Data

Prenume NUME

Vlăduț Dobră

(semnătura)



SINTEZA

lucrării de diplomă cu titlul:

Simulator embedded pentru bordul unui autoturism folosind comunicație CAN

Autor: **Vlăduț-Alexandru Dobră**

Conducător științific: **Prof. dr. ing. Ioan Nașcu**

1. Cerințele temei: implementarea unui afișaj analogic/digital ce va simula accelerația, turația și treapta de viteză a motorului unei mașini.
2. Soluții alese: utilizarea microcontrolerelor de la STM, s-a folosit comunicația CAN pentru transmiterea de informații, partea analogică are în componență două motoare pas cu pas, iar cea digitală un LCD cu două linii.
3. Rezultate obținute: afișarea, prin controlul motoarelor pas cu pas și al LCD-ului, a turației, vitezei și treptei printr-o comunicație CAN.
4. Testări și verificări: s-au efectuat teste atât pe partea de control al perifericelor cât și pe partea de transmisii de date.
5. Contribuții personale: implementarea modului de funcționare al dispozitivelor utilizate.
6. Surse de documentare: în mare parte din mediul online (site-uri de specialitate, documentația componentelor, forumuri și în special tutoriale video), dar și prin schimburi de idei cu colegi sau profesori.

Semnătura autorului

Semnătura conducătorului științific

Cuprins

1	INTRODUCERE.....	2
1.1	CONTEXT GENERAL	2
1.2	OBIECTIVE.....	3
1.3	SPECIFICAȚII	3
2	STUDIU BIBLIOGRAFIC.....	4
2.1	TRANSMITEREA DATELOR	4
2.1.1	<i>Interfața paralelă si serială.....</i>	5
2.1.2	<i>Interfață serială sincronă și asincronă.....</i>	6
2.1.3	<i>Controller Area Network.....</i>	8
2.2	DESCRIEREA PLĂCII.....	13
2.2.1	<i>STM32L476RG Nucleo-64</i>	13
2.3	DESCRIEREA COMPONENTELOR AUXILIARE	15
2.3.1	<i>Transceiver MCP2551</i>	15
2.3.2	<i>LCD.....</i>	16
2.3.3	<i>Driver motor pas cu pas.....</i>	17
2.3.4	<i>Motor pas cu pas</i>	18
3	ANALIZĂ.....	20
4	PROIECTARE.....	21
5	IMPLEMENTARE	23
5.1	DESCRIEREA PROGRAMELOR	23
5.1.1	<i>STM32CubeMX</i>	23
5.1.2	<i>IAR Embedded Workbench</i>	25
5.2	STADIILE PROIECTĂRII	26
5.2.1	<i>Microcontroler 1</i>	26
5.2.1.1	Achiziție date.....	26
5.2.1.2	Modelul mașinii.....	28
5.2.1.3	Configurația CAN.....	32
5.2.1.4	Trimiterea datelor pe magistrala CAN.....	35
5.2.1.5	Legătura fizică a comunicației CAN	36
5.2.2	<i>Microcontroler 2</i>	37
5.2.2.1	Continuarea legăturii hardware a comunicației CAN	37
5.2.2.2	Recepția datelor de pe magistrala CAN.....	39
5.2.2.3	Programarea și conexiunile afișajului.....	43
5.2.2.4	Inițializarea motoarelor.....	47
5.2.2.5	Comandarea motoarelor	49
5.2.2.6	Înglobarea într-un sistem compact	52
6	CONCLUZII.....	53
6.1	REZULTATE OBȚINUTE.....	53
6.2	DIRECȚII DE DEZVOLTARE.....	54
7	BIBLIOGRAFIE.....	54

1 Introducere

1.1 Context general

Comunicația dintre un emițător și un receptor, în ceea ce privește microcontrolerele, are nevoie de o legătură și de o înțelegere reciprocă a comunicației ce urmează a fi folosită. Această legătură poate să fie prin fir sau fără fir. Odată stabilită o conexiune între emițător și receptor se pot crea diverse aplicații.

O aplicație de acest gen este descrisă în această lucrare, ce dorește să folosească două microcontrolere STM32L476RG conectate între ele prin fire și componente auxiliare. Prin respectiva aplicație se poate vedea modul în care se realizează comunicația într-un sistem înglobat al unui board de autoturism. Comunicația se face printr-un protocol numit CAN(Controllor Area Network), ce reduce numărul de fire necesare pentru a transmite informații la componentele care compun sistemul.

Protocolul CAN permite microcontrolerelor și dispozitivelor acestora să comunice în aplicații fără a fi necesar un calculator gazdă pentru gestionarea rețelei de comunicație. Acest protocol este folosit în special în industria auto.

Transmiterea datelor se realizează cu o viteză suficient de mare pentru a nu exista probleme de întârziere. Siguranța transmiterii datelor este de asemenea prezentă, având un rol important în schimbul de informații.

Odată stabilită conexiunea microcontrolerelor are loc conectarea tuturor perifericelor.

Primul microcontroler (ce va ține locul de CAN master), va avea o rezistență variabilă (potentiometru) prin care se vor introduce date fizic în sistem și un dispozitiv de emisie-recepție folosit în comunicarea CAN.

Transmiterea informației prin magistrala CAN, numită și CAN Bus, va fi însoțită de două trancieivere(dispozitivele de emisie-recepție) MCP2551, două rezistente de 120 Ω și două de 1k Ω . Acestea din urmă (trancieivere și rezistente) vor servi în a face legătura între protocolul CAN de pe microcontroler și magistrala fizică pe care o completează.

Al doilea controler deține partea mobilă și grafică a proiectului. El va avea un dispozitiv de emisie-recepție identic cu cel al primului controler, va controla două dispozitive specializate în controlul motoarelor pas cu pas, motoare ce se regăsesc și ele în acest proiect, și va mai controla un display LCD.

Programele pentru controlere vor fi încărcate de pe un laptop, iar apoi controlerele se vor descurca singure. Alimentările vor fi oferite de către calculatorul gazdă pentru microcontrolere și de către o sursă externă de 9V pentru motoare.

Structura lucrării permite implementarea cu succes a aplicației amintite. Mai întâi se va discuta despre aspecte teoretice de care vom avea nevoie în punerea în funcțiune a dispozitivelor. În capitolul trei se va analiza aplicația, iar în capitolul patru se va indica fiecare dispozitiv unde va fi situat și cum. Capitolul cinci constituie pașii concreți

necesari în realizarea aplicației. Primul pas va fi legat de mediul de programare al microcontrolerelor, iar apoi vom implementa începând cu primul microcontroler și terminând apoi cu cel de-a doilea. În cele din urmă vom avea un capitol destinat concluziilor și al direcțiilor de dezvoltare ulterioară.

1.2 Obiective

Obiectul principal este de a controla motoarele și display-ul printr-o comunicație de tip CAN. Printre celelalte obiective se numără felul în care se produce acest control, mai precis simularea unei accelerații de la un autovehicul și afișarea rezultatelor acestei accelerații.

Prin urmare, la introducerea datelor în sistem (acceleram), un model matematic va interpreta aceste date și va genera o anumită viteză și o anumită turatie ce vor fi transmise celui alt microcontroler. Aici turatia și viteza vor fi afișate fiecare printr-un motor pas cu pas, aceasta afișare analogică fiind însoțită și de una digitală materializată printr-un afișor LCD pe care se va pune la vedere informația legată de treapta de viteză în care se află autoturismul.

Modelul matematic al sistemului determină poziția indicatoarelor noastre. De aceea, în funcție de ce regăsim în acel model, așa se vor manifesta și datele de ieșire.

Ieșirile acestui sistem vor fi motoarele și indicatorul digital. Vom poziționa pe capatul mobil al motoarelor, acele indicatoare pentru vizualizarea rezultatelor.

1.3 Specificații

În vederea realizării aplicației se vor folosi două programe:

- CubeMX, cu ajutorul căruia va avea loc setarea pinilor ce urmează a fi utilizați în aplicație, gestionarea configurației ceasului microcontrolerului în vederea obținerii rezultatelor dorite intern sau extern și configurarea perifericelor și protocoalelor alese.
- IAR Embedded Workbench pentru programarea în C a STM-urilor.

Programele pentru controlere vor fi încărcate de pe un PC, iar apoi controlerele se vor descurca singure, cu excepția alimentării care se va face de la dispozitivul gazdă.

Cea de-a doua alimentare va fi necesară pentru partea de putere a driverelor folosite la motoare. Pentru a satisface această cerință vom folosi o baterie de 9V ce va alimenta cu energie ambele motoare.

2 Studiu bibliografic

2.1 Transmiterea datelor

Transmisia de date între microcontrolere se poate face folosind fire (cabluri din cupru, fibră optică) sau folosind o comunicație fără fire (wireless) prin raze infraroșu sau unde radio.

Transmiterea datelor prin mediul wireless necesită dispozitive periferice atașate microcontrolerului ce ulterior vor fi folosite pentru a trimite date spre un alt dispozitiv ce captează aceste unde.

Folosirea acestui tip de comunicație are avantajul de a reduce drastic numărul de conexiuni fizice (cablaj) și posibilitatea de a modifica poziția lor în spațiu cu ușurință. Conexiunea wireless nu este mereu cea mai bună în orice situație deoarece pot apărea interferențe între dispozitive și suntem limitați de raza de acțiune a emițătorului și al receptorului.

Transmiterea datelor folosind un mediu fizic nu necesita dispozitive auxiliare. În marea majoritate a cazurilor sunt necesare conexiuni prin fire între controlere, rezistente pentru liniile de conexiuni sau cipuri de emisie-recepție (transceiver).

Acest mediu robust de conexiune da o garanție în plus a eficientizării și al corectitudinii transmiterii datelor, o viteză mai mare a comunicației și un cost redus.

Pentru realizarea unei conexiuni și a unei comunicatii totodata între mai multe microcontrolere este nevoie de un studiu în prealabil asupra modului de operare al aplicației, mediul de lucru, costuri, distanța de la care trebuie să comunice dispozitivele și tipul de utilizatori ce va folosi sistemul ce urmează să fie conceput. Toate aceste variabile trebuie puse în balanță și ales apoi modul de abordare.

Mediul fizic, prin cablu, în cazul microcontrolerelor, are mai multe tipuri de protocoale ce pot fi utilizate. În funcție de ce dorim să facem în proiect trebuie ales protocolul cel mai indicat pentru realizarea transmiterii și recepției de date. Putem aminti câteva protocoale de comunicație: SPI, I2C, UART, CAN, USB, Ethernet, PCIe.

Fiecare protocol este indicat în a fi folosit pentru anumite tipuri de operații, de exemplu: Ethernet este utilizat pentru microcontrolere ce au acces la internet, PCIe pentru transmiterea a mari cantități de date cum ar fi imagini, USB prin intermediul careia încarcăm programele pe microcontroler, I2C folosit la transmiterea datelor sincronizate necesitând conexiunea a doar doi pini de pe microcontroler (SDA și SCL).

Electronica înglobată (embedded) se bazează pe interconectarea dispozitivelor, senzorilor sau a altor circuite integrate pentru a crea un sistem. Pentru a putea comunica independent între ele, mai exact să facă schimb de date, aceste circuite trebuie să aibă un protocol de comunicație comun. S-au dezvoltat multe tipuri de protocoale de comunicație pentru a se realiza acest lucru. Putem să le separăm în două grupe: seriale și paralele.

2.1.1 Interfața paralelă si serială

Interfața paralelă transferă mai mulți biți în același timp. Se folosesc multe fire, datele fiind transmise in pachete mari. În figura 2.1 se poate vedea un astfel de transfer, pe fiecare săgeată se va transmite cate un bit ce va forma mai apoi un byte. La fiecare impuls de ceas, se vor transmite de la ieșirea 0->7 spre intrarea 0->7 câte un bit . În această schemă sunt necesare 9 fire pentru conectare.

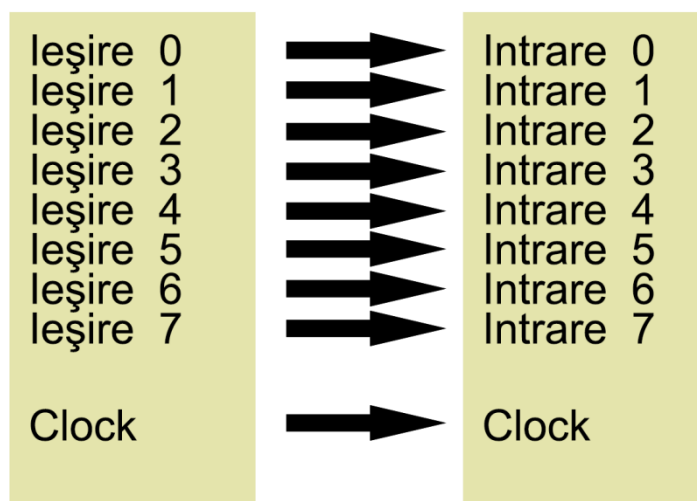


Figura 2.1

Comunicația paralela are avantajul de a fi rapida, directă și relativ simplu de folosit. Dezavantajul îl constituie numărul mare de pini de intrare/ieșire ce trebuie folosiți si multe legături fizice (cabluri suplimentare).

Interfața serială transmite datele diferit de cea paralelă. Transferul se face pe rând, asemănător unui tren, bit după bit. Această interfață poate opera pe un număr redus de fire, poate fi chiar și un singur fir, în cazul din figura de mai jos sunt doar două fire, unul pentru date și unul pentru ceas.



Figura 2.2

Un exemplu practic legat de cele două interfețe ar fi asemănarea interfeței paralele cu o autostradă cu opt benzi pe sens, iar cea serială cu un drum comunal cu o bandă pe sensul de mers. Prin urmare, autostrada cu opt benzi este mult mai eficientă, autovehiculele putând înainta cu viteze mari într-un număr mult superior celui cu o bandă pe sens, dar pentru zona unde este situat drumul mai îngust este suficient, făcând

față traficului de mașini din zonă. Costurile pentru autostradă sunt semnificativ mai mari față de drumul comunal. Concluzia ar fi ca trebuie adaptat modul de abordare în funcție de cerințe.

2.1.2 Interfață serială sincronă și asincronă

De-a lungul anilor au apărut o multitudine de protocoale seriale, cu posibilitatea de a fi folosite într-un sistem înglobat. Două dintre cele mai cunoscute și utilizate protocoale sunt Ethernet și USB (Universal Serial Bus). Alte interfete foarte răspândite sunt SPI (Serial Peripheral Interface), I2C (Inter-Integrated Circuit), UART (Universal asynchronous receiver/transmitter), CAN (Controller Area Network) etc.

Interfața serială sincronă își sincronizează liniile de date cu un semnal al ceasului microcontrolerului, pentru ca toate dispozitivele din aceasta magistrală să împartă un impuls de ceas comun. Asta face ca interfața serială sincronă să fie robustă, rapidă, dar necesită întotdeauna o legătură în plus între dispozitivele de comunicare. De exemplu: SPI și I2C (SCL – linia de ceas, SDA – linia de date).

Interfața serială asincronă se referă la transmiterea de date fără a avea influența unui impuls de ceas. Pentru un caz în care ne dorim minimizarea numărului total de conexiuni (fire) și pini de intrare/ieșire această metodă este de interes major. Mecanismul pentru pastrarea robusteții și al lipsei de erori în transferul de date, datorită lipsei unui ceas extern, are câteva reguli.

Aceste reguli sunt necesare pentru a fi siguri de buna funcționare a comunicației. Dispozitivele de pe o magistrală de acest gen trebuie configurate identic pentru a putea utiliza același protocol. Acest set de reguli se referă la:

- **Viteza de transmitere a datelor** pe linia serială (numită și Baud Rate), de obicei exprimată în biți pe secunda (bps). Această valoare determină cât timp emițătorul ține linia serială pe 1/0 logic sau ce perioadă este necesară pentru dispozitiv să eșantioneze linia de date.

Baud rate poate să fie aproape orice valoare. Singura cerință este ca fiecare dispozitiv să funcționeze la aceeași valoare. Cea mai întâlnită rată este de 9600 bps. Cu cât este mai mare cu atât datele sunt transmise sau recepționate mai repede. Printre cele mai mari viteze se numără cea de 115200 bps, ce este destul de rapid pentru multe controlere.

Dacă se introduce o viteză prea mare, ceasul intern și configurația microcontrolerului nu mai fac față și se vor trimite date eronate.

În fig. 2.3 avem un exemplu de date trimise cu o viteză prea mare (921600)

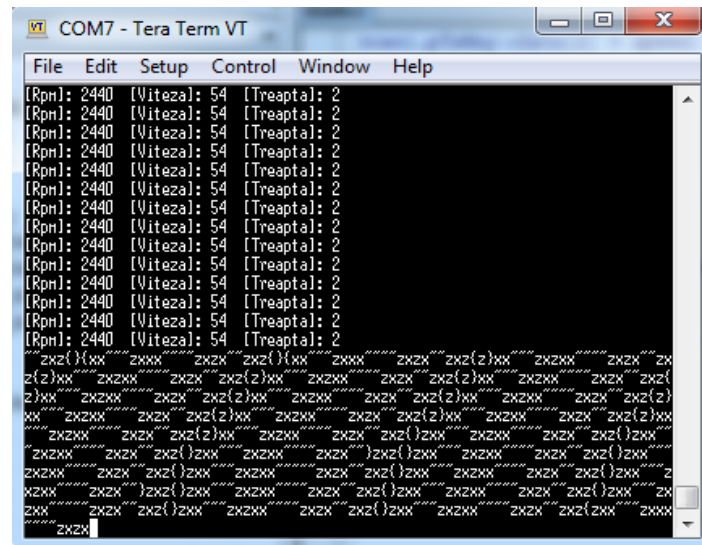


Figura 2.3

În fig. 2.4 avem un exemplu de date trimise cu o configurații diferite.

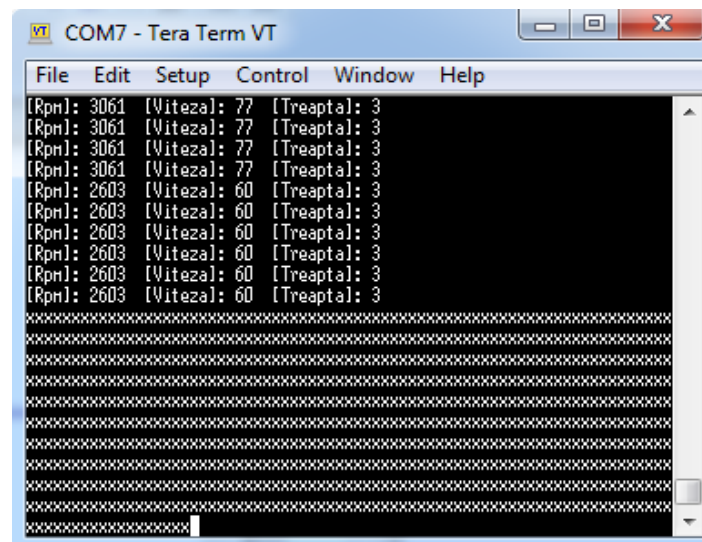


Figura 2.4

• **Divizarea datelor**(Data frame). Datele sunt transmise sub forma unui frame de biți. Aceste frame-uri sunt create adaugând biți de paritate și biți de sincronizare blocului de date. Un exemplu se poate vedea mai jos fig. 2.5:

Frame:	Start	Date	Paritate	Stop
Dimensiune (bits):	1	5-9	0-1	1-2

Figura 2.5

- **Biții de paritate** sunt o metodă foarte simplă de verificare a erorilor. Bitul de paritate poate să fie 0 sau 1 pentru a face numărul de biți de 1 să fie ori “even” ori “odd”. Când transmitem date electronice, se poate întâmpla ca biții să-și schimbe eronat starea din 1 în 0 sau invers.

De exemplu, avem biții 10010 cu bitul de paritate 1 rezultând secvența 100101. Această informație transmisă unui alt calculator, ajunge cu o eroare 100001. Calculatorul receptor verifică dacă datele sunt corecte adunând biții de 1, iar suma lor o împarte la 2, restul rezultat fiind bitul de paritate. Dacă restul împărțirii este egal cu bitul de paritate recepționat atunci informația va fi “even” dacă nu, va fi “odd”. În cazul în care este “odd” se va cere retransmiterea datelor până când acestea vor fi corecte.

- **Biți de start** și de stop, sau biți de sincronizare, sunt doi sau trei biți transferați cu fiecare pachet de date. După cum le spune și numele acești biți constituie delimitarea pachetului de date trimis. Întotdeauna există un singur bit de start însă biții de stop pot fi doi, depinde cum este configurat protocolul respectiv (de obicei este doar unul).

- **Datele** din frame sunt cele mai importante, pe ele dorim să le trimitem în mod special, celelalte campuri de biți ajutând doar la buna realizare a acestui transfer.

Considerând un frame cu dimensiunea datelor de 8 biți, adică un octet (byte, cel mai des întâlnit), trebuie să se specifice și capătul de la care citim datele, numit și endian. De obicei se consideră primul bit citit ca fiind cel mai nesemnificativ bit (LSB) din zona de date, dar se poate considera și ca cel mai semnificativ bit (MSB).

2.1.3 Controller Area Network

Dezvoltarea industriei automobilelor a fost cauza principală a necesității acestui protocol. Cererea foarte mare de autovehicule a determinat această dezvoltare, fiecare producător aducând ceva nou atât în ceea ce privește designul mașinii cât și al aerodinamicii, al motorului, al transmisiei sau al altor dispozitive (radio, ventilație, navigație etc).

Inițial toate legăturile electronice erau realizate prin cabluri. Adăugând tot mai multe dispozitive electronice autovehiculului au fost necesare tot mai multe legături, implicit tot mai multe cabluri. Asta n-a făcut decât să crească greutatea mașinii, să mărească costurile de producție datorită multitudinii de cabluri și al personalului de manufacturare. Pentru a elimina acest impediment producătorii au încercat să înlocuiască aceasta conexiune prin cabluri cu o conexiune de tip rețea de calculatoare, reducând costurile, materialele folosite, complexitatea și greutatea mașinii. În 1985 Bosch a propus Controller Area Network, protocol care a devenit mai târziu un standard. Putem aminti standardele ISO 15765 pentru automobile, J1939 pentru tractoare și mașini industriale DeviceNET sau CANopen.

Avantajul acestui protocol este dat de posibilitatea conectării la computerul de bază al mașinii, zis și ECU (Electronic Control Unit), prin magistrala CAN (CAN Bus).

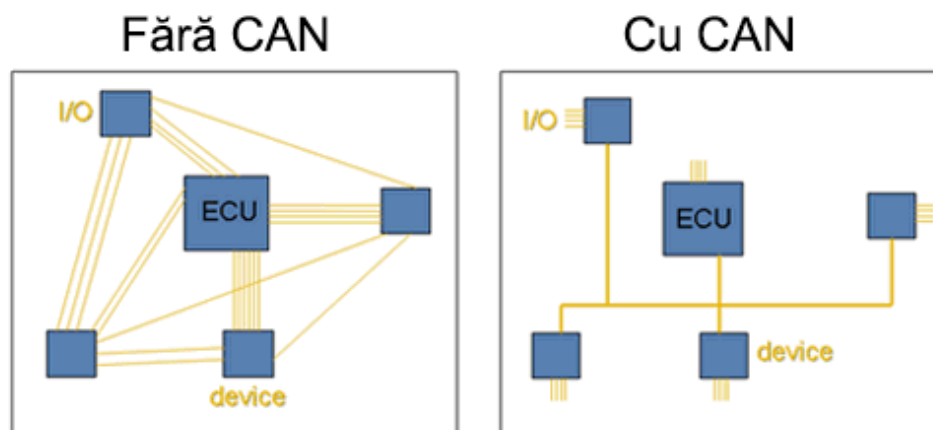


Figura 2.6

În figura 2.6 putem observa diferența de conexiuni dintre o rețea fără CAN (foarte multe fire) și o rețea cu CAN.

Controlerul CAN este sofisticat. Aproape toate acțiunile ce le face acest protocol sunt controlate de către controler cu o foarte mică intervenție din exterior (PC). Ideea de bază este să configurăm controlerul prin scrierea în regiștrii lui și să-i introducem date, de transmiterea mesajului pe magistrală se ocupa el. Controlerul citește toate frame-urile detectate pe magistrală și le păstrează într-un FIFO. Apoi va notifica calculatorul central de existența datelor din frame, date ce vor fi preluate de calculatorul central. Controlerul mai conține un mecanism de filtrare ce poate fi programat să ignore unele mesaje CAN ce nu se doresc a fi prelucrate.

Tranceiverele CAN ajută și mai mult rețeaua, contribuind printr-un mediu mai stabil și mai eficient fără a fi nevoie de cabluri scumpe și sofisticate. El are două sarcini de îndeplinit:

- Recepție- adaptează semnalele primite de pe magistrală pentru a putea fi interpretate de controlerul CAN, are și un rol de protecție;
- Transmitere – convertește biții primiți de la controler într-un semnal ce îl va trimite pe magistrală;

În figura de mai jos (fig 2.7) este prezentată o schemă generală a unei magistrale CAN cu n noduri. Fiecare nod este format dintr-un microcontroler cu protocol CAN(nod), un tranceiver CAN și legăturile MCU-Tranceiver-magistrală CAN.

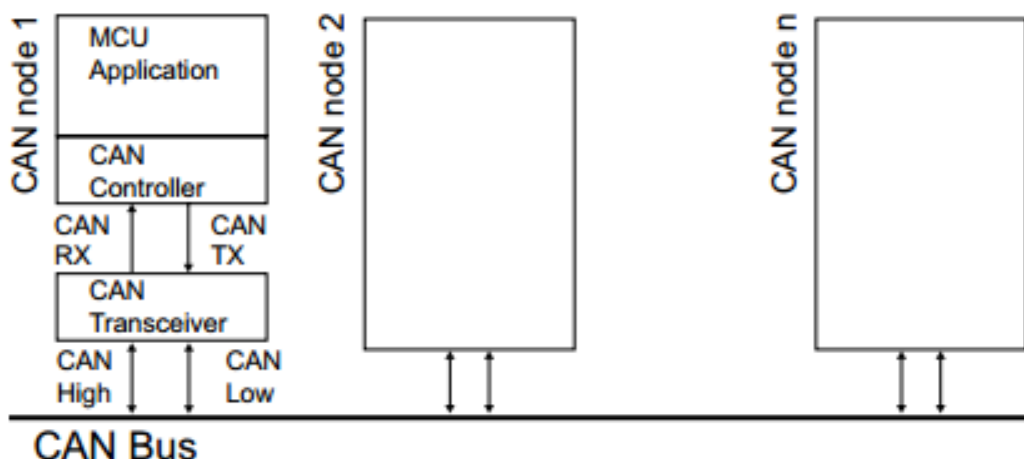


Figura 2.7

În mod normal magistrala CAN este formată din 2 fire reprezentate de CANH (CAN High) și de CANL (CAN Low) ce conectează toate dispozitivele din rețea. Cele două linii CAN au aceeași secvență de date, dar au amplitudini opuse. Deci dacă un impuls pe linia CANH va merge de la 2.5V la 3.75V atunci impulsul corespunzător de pe linia CANL va merge de la 2.5V la 1.25V. Transmițând datele în felul acesta, în impulsuri egale ca poziție și amplitudini opuse, se permite o imunitate mare la zgomot și șanse mai mari ca datele să nu fie eronate.

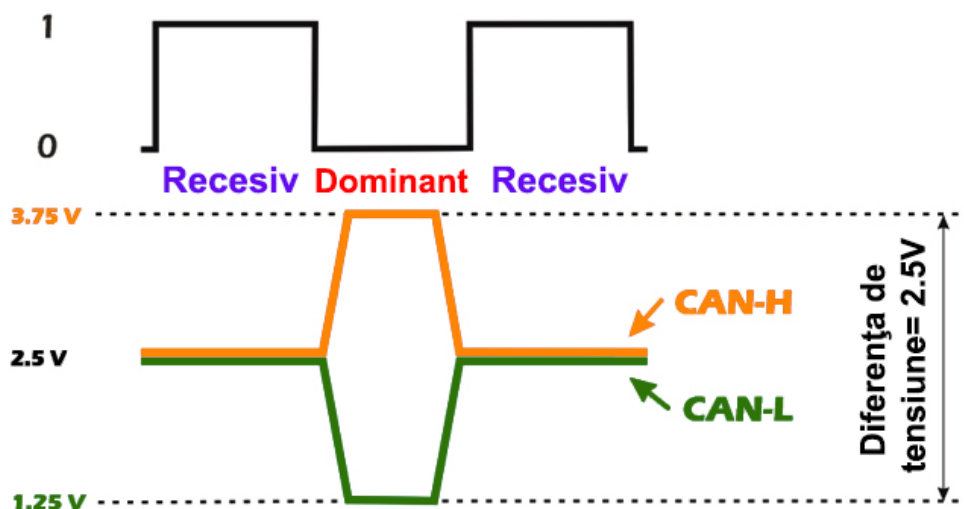


Figura 2.8

Dupa cum se poate observa și în figura de sus, un bit ce are o diferență de tensiune de 2.5V va fi un bit dominant (0 logic), iar un bit ce va avea o diferență de tensiune între CANH și CANL de 0V el va fi recesiv (1 logic).

Controlerul CAN primește datele de la integratul aflat în microcontroler ce face funcțiile de CAN (acest ansamblu controler+integrat CAN este cunoscut ca și nod CAN). Microcontrolerul procesează datele și le trimite traneiverului CAN. Microcontrolerul ce utilizează acest protocol CAN poate să și primească date de la traneiverul CAN, le procesează și le trimite zonei destinate lucrului cu interfața CAN. Traneiverul este un dispozitiv de emisie-recepție. El convertește datele, ce controlerul CAN le trimite ca

semnal electric, și le trimite pe magistrală. De asemenea primește date pentru microcontroler, date pe care le convertește cu scopul de a fi percepute de controler. La capătul magistralei se găsesc rezistențe, de obicei de 120Ω . Aceste previn ca datele trimise să se reflecte la capete și să fie trimise înapoi ca ecouri.

Transferul de date în cadrul protocolului CAN are următoarele etape: achiziția datelor, trimiterea datelor, recepționarea datelor, verificarea și acceptarea datelor.

- Achiziția datelor: nodul CAN aduce date controlerului CAN pentru transfer;
- Trimiterea datelor: dispozitivul de emisie-recepție primește date de la controlerul CAN, le convertește în semnal electric și îl trimite înapoi în rețea;
- Primirea datelor: toate nodurile CAN conectate la rețea (magistrală) devin zone de recepție;
- Verificarea datelor: nodul CAN verifică dacă a primit informația cerută;
- Acceptarea datelor: dacă datele primite sunt importante ele vor fi acceptate și procesate, dacă nu, ele vor fi ignorate.

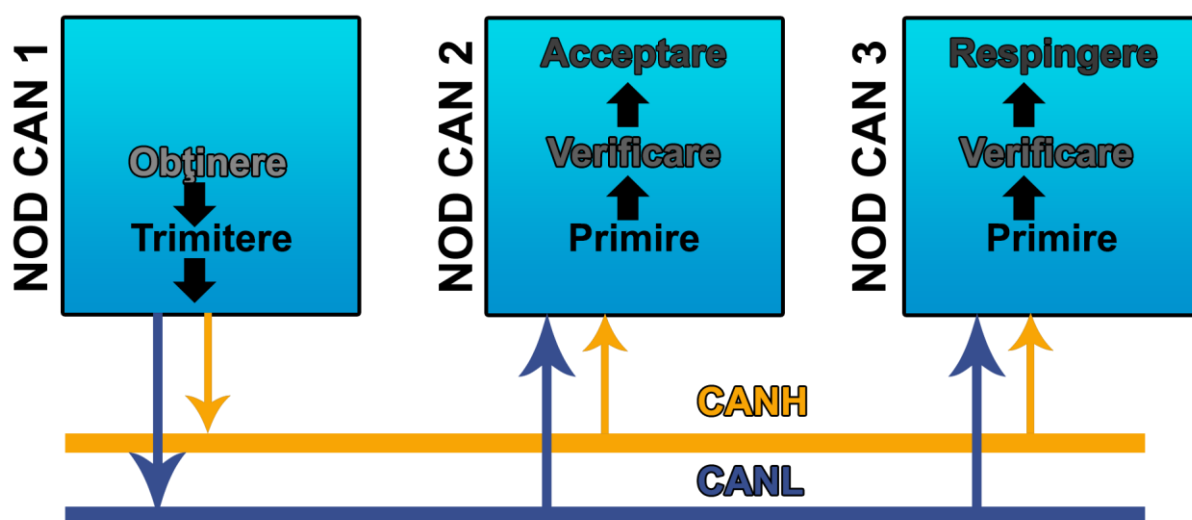


Figure 2.9

Studiind mai de aproape fenomenul, aflăm că sistemul CAN este împărțit în două în ceea ce privește mesajele. Așadar avem mesajul 2.0A și 2.0B, cele două standarde de mesaje diferite prin mărimea numărului de biți (a identificatorilor-ID):

Standard CAN (2.0A) folosește 11 biți de identificare în zona de mesaj.

Extended CAN (2.0B) folosește 29 biți de identificare, acesta este compus din cei 11 biți de identificare de la 2.0A, considerați biți de bază și 18 biți suplimentari.



Figure 2.10

SF(Start Field): indică începutul mesajului cu un bit dominant. De exemplu un bit cu 3.75V este transmis pe linia CANH și un bit de 1.25 V este transmis pe linia CANL, diferența va fi de 2.5V prin urmare vom avea un bit dominant.(1 bit)

Mesajul de identificare: definește nivelul de prioritate al protocolului de date. Dacă două noduri vor să transmită în același timp date, nodul CAN cu cea mai mare prioritate va avea întâietate. Cu cât valoarea este mai mică, prioritatea mesajului va fi mai mare. Totul depinde de standardul folosit (ID=11 sau ID=29).(11 sau 29 biți)

Control(cunoscut și ca zona de verificare): ne arată numărul de câmpuri de informație ce vor fi în zona de date. Acest câmp permite oricărui receptor să verifice dacă a primit toate informațiile ce i-au fost transferate.(6 biți)

Date(sau zona de date): din această zonă sunt preluate datele și transmise către alte noduri CAN. (până la 64 biți)

CRC(Cyclic Redundancy Check) cunoscută și ca zona de siguranță, conține 15 biți de cod și un bit delimitator recesiv. Acest câmp este folosit pentru a determina transferul eronat de date. (16 biți)

ACK(Acknowledge Field) numită și zona de confirmare, aici receptorii anunță emițătorii că au primit datele corect. Dacă se detectează o eroare, se anunță emițătorul imediat. Emițătorul transmițând datele din nou la aflarea acestei "vești". (2 biți)

EF(End Field): indică sfârșitul protocolului de date. Este ultima posibilitate de a semnaliza o eroare în transmisie. Dacă nu apare acest set de date atunci mesajul se va retransmite. (7 biți).

Într-un sistem CAN nu există o componentă de tip master ce controlează transmisia și recepția de date între noduri. Când un nod CAN este gata să transmită un mesaj, verifică statusul magistralei, iar dacă este liberă trimite un mesaj(frame) în rețea. Frame-urile CAN ce sunt transmise nu conțin adrese exacte ale nodurilor de unde trebuie să primească mesaje sau unde trebuie să transmită mesajele, în schimb ele au un ID de identificare unic în rețea prin care se etichetează.

Toate nodurile CAN din rețea primesc frame-ul CAN și depinzând de ID-ul de identificare transmis cu mesajul, fiecare nod decide dacă acceptă sau nu acest mesaj recepționat.

Dacă mai multe noduri vor să transmită în rețea în același timp se va ține cont de prioritate. Prioritatea cea mai mare (cu ID de arbitraj cel mai mic) primește automat acces la magistrală. Cele cu prioritate mai mică așteaptă până când magistrala devine disponibilă pentru a putea trimite informații.

2.2 Descrierea plăcii

2.2.1 STM32L476RG Nucleo-64

Seria placilor STM32 Nucleo este o serie low-cost, ușor de folosit și de programat. Este compatibilă cu o gamă mare de extensii, conectica este asemanatoare si compatibilă cu numeroase dispozitive (are în componență conexiuni Arduino Uno R3 si ST morpho). Integrează un sistem de depanare cunoscut ca ST-Link Debugger. Partea software vine cu library HAL (Hardware Abstraction Layer) și o multitudine de exemple ce pot fi testate și studiate folosind diverse medii compatibile de programare: IAR EWARM, Keil MDK-ARM sau ARM mbed online.

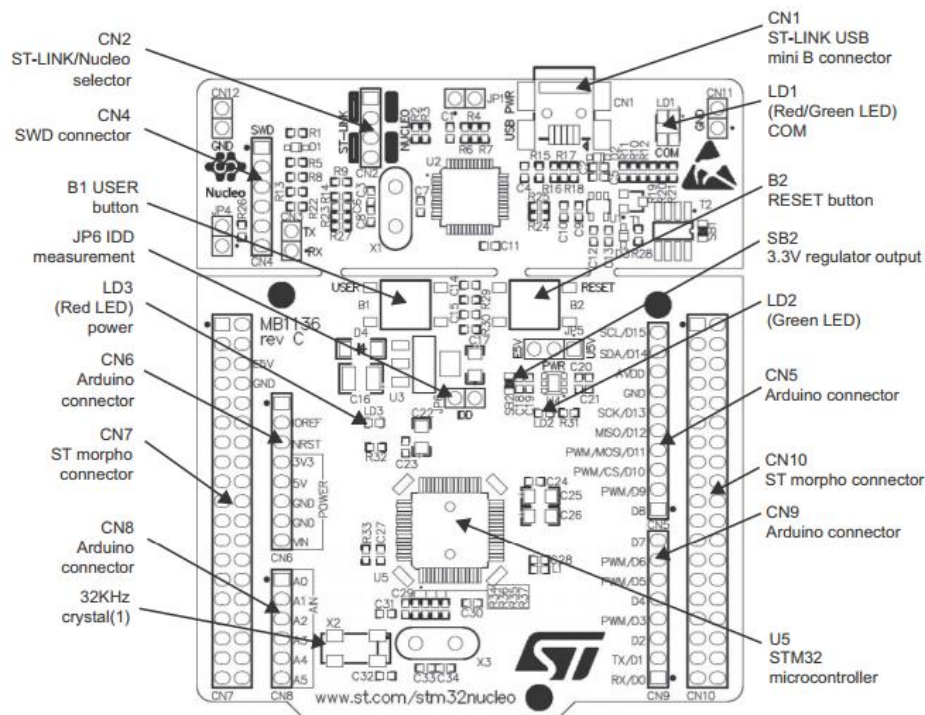


Figura 2.11

Placa Nucleo-L476RG are cea mai mare memorie flash (1 MB) din seria Nucleo-64 și o viteză 128-KB SRAM. Un processor ARM Cortex-M4 cu o frecvență de 80 MHz, două tipuri de conexiuni Arduino Uno si ST morpho si un debugger Embedded ST-LINK/V2-1. Placa mai are posibilitatea interfațării pinilor, ceasului si al protocoalelor cu un program numit STM32CubeMx.

Alimentarea se poate face atât de la un PC printr-un cablu USB cât si de la o sursă externă. Alimentarea prin cablu mini-USB se face prin portul CN1 (ST-LINK USB) când avem jumperul JP5 pus pe pinii 1 si 2 adică spre U5V (fig. 2.12). Dacă dorim sa alimentam de la o alta sursă va trebui sa mutăm jumperul pe pinii 2 si 3 adică spre E5V (fig. 2.13).

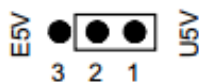


Figura 2.12

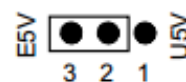


Figura 2.13

Pe placa STM32L476RG putem distinge urmatoarele particularitati:

- Un LED tricolor (verde, portocaliu si roșu) LD1; acest led oferă informații despre statusul comunicației cu ST-LINK. Culoare implicită a LED-ului este roșie. Se face verde atunci când indică o comunicare în curs între PC și ST-LINK/V2-1, cu urmatoarele particularități:

- LED roșu când conexiunea între PC si ST-LINK este completă;
- LED verde dupa o conexiune recentă realizată cu succes;
- Oscilație roșu/verde în timpul comunicării cu PC-ul;
- LED portocaliu înseamna o comunicare eșuată;
- Roșu intermitent atunci cand am conectat dispozitivul la tensiune dupa ce am inițializat o altă comunicare.

- Un LED tip utilizator, un LED verde pe care îl putem folosi (I/O PA5).

- Un LED roșu ce anunță că STM32 este alimentat și +5V sunt disponibili.

- Un buton pentru utilizator (I/O PC13).

- Un buton pentru reset, acest buton este folosit pentru a reseta microcontrolerul. În figura 2.11 se pot observa aceste particularități, iar în figura 2.14 sunt pinii de intrare/ieșire ai plăcii.

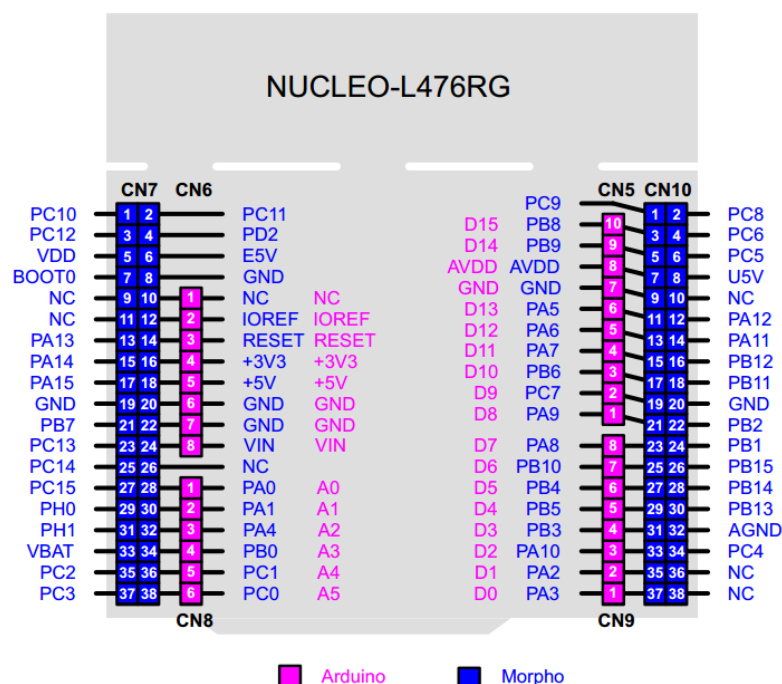


Figura 2.14

Această placă are numeroase componente și periferice înglobate ce pot fi utilizate. Amintim cele mai importante și cele mai utilizate dintre ele:

ADC=3, CAN=1, I2C=3, RCC=1, SPI=3, TIM=11, UART=5 etc.

2.3 Descrierea componentelor auxiliare

În vederea obținerii unui sistem controlat prin CAN vom implica mai multe dispozitive periferice. Pentru a afișa și a ajuta la afișarea datelor obținute din comunicația dintre microcontrolere prin magistrala CAN.

2.3.1 Transceiver MCP2551

MCP2551 este un dispozitiv pentru CAN, de mare viteză, cu toleranță la eroare, ce servește ca intermediar între un protocol CAN al unui microcontroler și o magistrală fizică. Funcționează la viteze de până la 1 Mb/s.

În mod normal fiecare nod CAN din sistem trebuie să aibă o componentă ce va converti semnalul digital generat de un microcontroler CAN în semnale ce vor putea fi transmise pe magistrala CAN. De asemenea poate oferi un tampon între controlerul CAN și zgomotul ce se poate genera pe magistrala din partea surselor externe (interferențe electromagnetice, designul circuitului sau variații ale tensiunii de alimentare). În figura 2.15 este prezentată schema emițător-receptor-ului:

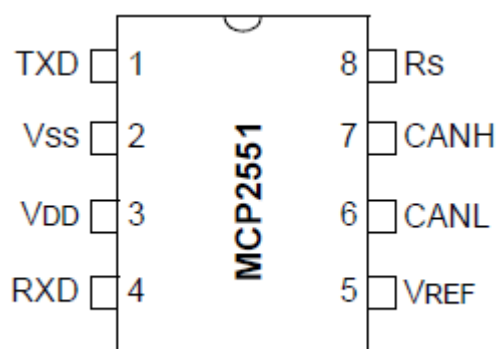


Figura 2.15

Descriere pini:

RXD(4) – recepționează date de pe magistrala CAN

CANL(6) – reprezintă voltajul minim din magistrală

CANH(7) – reprezintă voltajul maxim din magistrală

VSS(2) și VDD(3) – sunt GND, respectiv 5V

TXD(1) – conectat la Tx-ul controlerului

RS(8) – se alege modul de operare (H-Speed, Slope-Control sau Standby)

VREF(5) – tensiune de referință (definită ca $VDD/2$)

2.3.2 LCD

1602A QAPASS este un LCD de caractere industrial. Din denumirea lui putem extrage informații referitoare la numărul de linii, coloane și caractere. El fiind asociat și cu denumirea de 16 x 2 LCD, venind de la un LCD cu 16 coloane și două rânduri (facând un total de 32 de caractere).

LCD-ul are următorii pini:

- VSS – reprezentând ground-ul dispozitivului
- VDD – tensiunea de alimentare
- VO – intensitatea luminoasă a scrisului (rezistența variabilă)
- RS – selectarea registrului (H: Date IN, L: Instrucțiun IN)
- R\W – selecția tipului de comandă (H: Citire, L: Scriere)
- E – permite instrucțiunii să treacă
- D0,D1,D2,D3,D4,D5,D6,D7 – zona de date (se poate utiliza 4 sau 8 biți)
- A – anodul ecranului, conectat la 5V luminează în “high-power”
- K – catodul ecranului, de conectat la GND

Anodul și catodul dacă sunt cuplați la VCC respectiv GND, ecranul va funcționa în modul high-power, adică va fi luminat complet. Dacă nu se fac conexiunile între acești doi pini, ecranul va funcționa în modul low-power, ecranul va fi vizibil dar cu o intensitate mult redusă.



Figura 2.16

În figura de mai sus este afișat un mesaj pe un astfel de LCD.

Există un modul special pentru aceste display-uri care ajută la comunicarea cu microcontrolerul mult mai ușor. Se conectează dispozitivul respectiv (de exemplu PCF857AT) la LCD 16x2 și prin protocolul I2C se face legătura cu controlerul doar prin două fire (SCL și SDA). Această reducere de fire se face doar pentru biții de date, alimentările rămân aceleași.

2.3.3 Driver motor pas cu pas

A4988 este un driver pentru motoarele pas cu pas bipolare ce oferă mai multe moduri de funcționare: fullstep(pas întreg), halfstep (jumătate de pas 1/2), quarterstep(sfert de pas 1/4), eighthstep (optime de pas 1/8) si sixteenthstep (șaisprezecime de pas 1/16).

Driverul poate funcționa cu motoare ce necesită o tensiune de alimentare de până la 35 V și în jur de 2 A.

Implementarea este simplă. Datorită translatorului inter a driverului este de ajunsă introducerea unui puls pe intrarea STEP iar motorul va primi un semnal de a se mișca un pas. Nu are tabele cu secvențe de fază, controlul liniilor de frecvență înaltă sau interfețe complexe de programat. A4988 este ideal pentru aplicații unde un procesor complex nu este disponibil sau este supraincărcat.

Tabelul de adevăr pentru modurile de funcționare a rezoluției pașilor este prezentat mai jos, iar sub tabel este o imagine cu driverul respectiv (fig. 2.17):

MS1	MS2	MS3	Microstep Resolution
L	L	L	Full Step
H	L	L	Half Step
L	H	L	Quarter Step
H	H	L	Eighth Step
H	H	H	Sixteenth Step

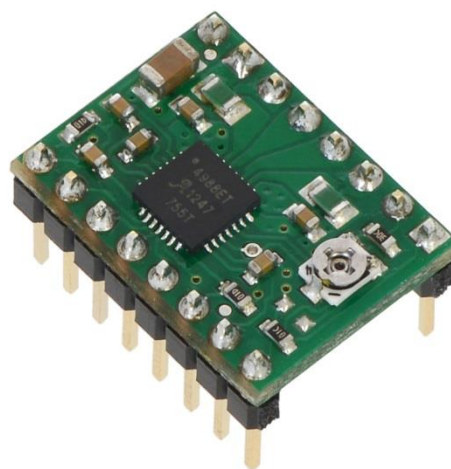


Figura 2.17

Driverul are în total 16 pini. Ei au următoarea distribuție: un pin de ground și unul de alimentare a cipului acțiune ce se va face de pe microcontrolerul folosit (între 3 și 5.5 V); patru pini de ieșire prin care se va comanda motorul și anume pentru prima fază 1A și 1B și a doua fază cu 2A și 2B (acestea patru vor fi conectate la motor); un pin de ground și unul de alimentare al motorului ce suportă o tensiune de alimentare între 8 și 35 V fiind indicat să se mai lege între cele două fire un condensator de 100μF pentru a rejecta eventualele oscilații venite de la motor; un pin de direcție numit DIRECTION, acest pin de intrare va determina direcția de deplasare a rotirii motorului; un pin pentru pași numit STEP, acest pin este tot de intrare și la fiecare puls va executa un pas; un pin de reset și unul de sleep, acești doi pini se leagă de cele mai multe ori între ei pentru a avea pe reset valoarea 0 logic ce va determina punerea motorului în funcțiune (pinul de sleep se folosește atunci când este dorită economisirea energiei electrice în cazul în care motorul nu are nici o comandă de executat); următorii pini: MS1, MS2, MS3 sunt pinii ce determină modul de executare a pașilor, acești trei pini sunt pini de intrare, dacă sunt lăsați în gol driverul va funcționa în modul de fullstep, dacă sunt activați toți cu 5V atunci driverul va funcționa în modul microstep (sixteenth steps); ultimul pin, cel de enable este folosit la activarea ieșirilor sau la dezactivarea acestora, dacă este lăsat în aer, fiind activ pe 0, acesta activează ieșirile și relația driver+motor va avea loc cu succes.

Acest tip de drivere au un mic potențiometrul pe ele. Cu ajutorul lui se poate regla tensiunea de referință cu care va funcționa driverul. Valoarea maximă a limitei de curent este determinată de rezistențele de pe driver RS. Intensitatea curentului maxim este dată de formulă:

$$I_{TripMAX} = V_{REF} / (8 \times R_s) \quad (2.1)$$

Unde, Vref este tensiunea măsurată la capetele potențiometrului și al ground-ului de pe driver conectat la microcontroler, Rs este rezistența de selecție care în cazul acestui driver este de 0.1Ω.

2.3.4 Motor pas cu pas

Acest tip de motor, cunoscut în literatură de specialitate ca "stepper motor", este un motor de curent continuu fără perii. Conceptul acestui motor este, după cum îi spune și numele, un motor cu pași discreți. Motorul are mai multe bobine grupate două câte două în faze. Aceste bobine se regăsesc în statorul motorului (partea statică). Polarizând fiecare fază în parte, în secvențe bine stabilite, rotorul (partea mobilă) a motorului se va roti câte un pas odată datorită magnetului aflat în rotorul motorului. În funcție de numărul de dinți pe care rotorul îl are, mișcarea va fi caracterizată de fluentă sau sacadări. Dacă are un număr mare de dinți rotația va fi mai fină, dacă numărul de dinți este mic rotația va fi caracterizată de o sacadare la fiecare polarizare.

Modul în care se face polarizarea bobinelor în statorul motorului determină comportamentul rotorului. Există mai multe moduri de operare a acestor polarizări. Amintim câteva moduri :

- **Wave drive:** fazele sunt polarizate pe rând, câte una, în ordine, rotorul aliniindu-se mereu cu zona polarizată;

- **Full drive:** fazele sunt polarizate in pereche, două câte două, rotorul ajungând în zona dintre doua faze alăturate;
- **Half drive:** fazele se polarizează combinând wave drive si full drive astfel: se polarizează o fază, apoi aceeaș fază cu următoarea, ca mai apoi să se polarizeze doar a doua fază. Procedul continuă asemanator și cu celelalte faze din stator. Rotorul se alinează atât în dreptul fazelor cât și între ele. Acest lucru crește rezoluția motorului.

Un mod de control al motorului mult mai fin este “microstepping”. Acesta poate să împartă pașii până la 256 de ori, făcând pașii mici și mai mici. Acest tip de polarizare folosește două sinusuri de 90° . În acest mod, motorul va fi foarte silențios și fără sacadări detectabile. Controlând direcția și amplitudinea curentului ce trece prin înfășurări, cresc caracteristicile motorului. Vibrațiile sunt mai mici și deplasările mai fluente. Aplicând polarizări de tip sinus, curentul va crește într-o înfășurare și va descrește în cealaltă rezultând o rotație a rotorului fluentă și progresivă.

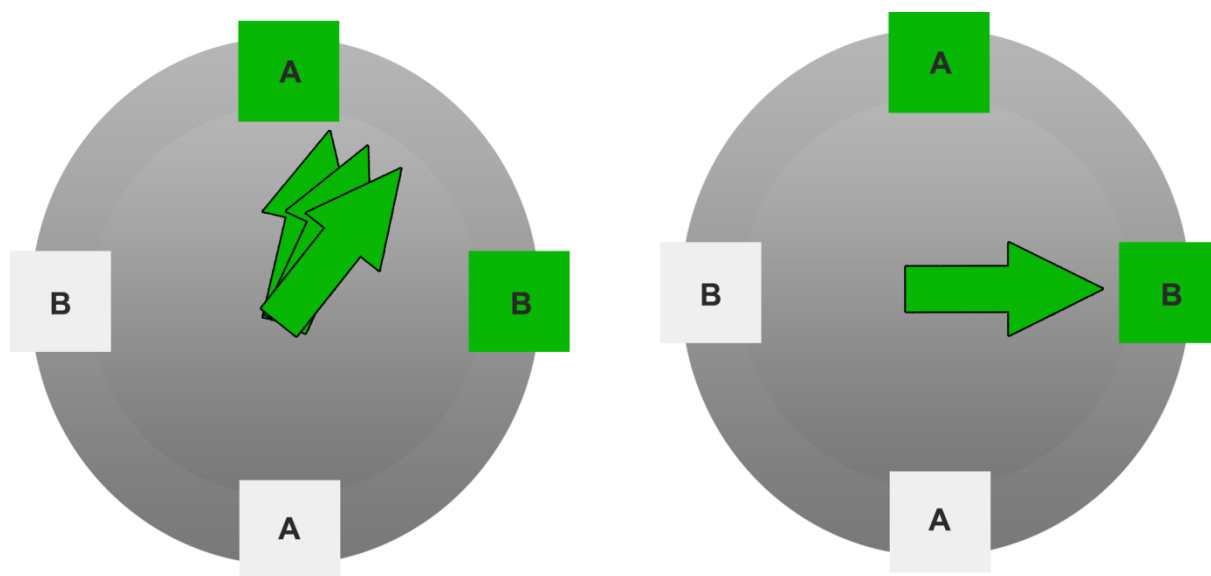


Figure 2.18

Există mai multe tipuri de motoare pas cu pas, și anume: motor pas cu pas cu magnet permanent, sincron hibrid și cu reluctanță variabilă(alinierea rotorului cu statorul se face cu un spațiu minim între dinți). Un exemplu de motor este mai jos:



Figura 2.19

3 Analiză

În cele ce urmează se va proiecta un sistem înglobat ce va simula un bord auto al unui autoturism. Această simulare va cuprinde un afișaj grafic și două indicatoare analogice ce vor reprezenta afișarea rotațiilor pe minut a mașinii, iar celălalt viteza în kilometrii pe ora.

Pentru realizarea acestui proiect sunt necesare cunoștințe legate de programarea microcontrolerelor, limbajul C, operații pe biți, transmisii de date, protocoale de comunicație, funcționarea motoarelor pas cu pas, electronica de bază precum și dexteritate cu obiecte mici și fragile.

Materialele folosite sunt următoarele:

- Două microcontrolere Nucleo-64 STM32L476RG
- Un afișaj LCD 1602A
- Două motoare pas cu pas
- Două potențiometre B1k si B50K
- Două transceivere MCP2551 pentru CAN
- Două drivere pentru motoarele pas cu pas A4988ET
- Două rezistențe de 120 Ω și două de 10k Ω
- O baterie 9V cu 1.5 A și un modul pentru cablare
- Doua breadboarduri pentru cablare
- Tipuri de fire (folosite): tată-tată(17), mamă-mamă(2), tată-mamă(28) și jumpere rigide(18)

Pentru protecție și aspect s-a folosit o cutie confecționată din carton tare în care s-au introdus toate componentele utilizate.

Proiectarea software a "cluster-ului" s-a realizat cu un laptop Lenovo G580 cu procesor i5, memorie de 4GB RAM ce rulează pe Windows 7 Ultimate Edition. Programele folosite au fost STM32CubeMX, IAR Embedded Workbench și emulatorul terminal Tera Term.

Utilitatea acestui proiect constă în deprinderea cunoștințelor de transmisii de date și controlul perifericelor prin cod C utilizând microcontrolere din gama STMicrocontrollers. După realizarea acestui proiect se va putea înțelege mai bine cum funcționează clusterul unui automobil, ce presupune proiectarea acestuia, iar apoi realizarea fizică și programarea astfel încât să simuleze un sistem real de control al indicatoarelor ce afișează utilizatorului mașinii informații despre turația motorului și viteza de deplasarea a mașinii. Toate acestea introducând o singură variabilă de intrare și anume accelerația (reprezentată prin potențiometrul B50K).

4 Proiectare

Dacă știm toate componentele necesare urmează să stabilim cum le vom aranja și conecta pentru a crea sistemul dorit. Asadar, după cum putem observa și în figura 4.1, ca date de intrare avem o rezistență variabilă reprezentată prin potențiometrul P1. Aceste valori sunt introduse în microcontrolerul MC1 de tip master care conține un model matematic, model ce determina trei variabile transmise mai apoi pe magistrala CAN microcontrolerului MC2 (slave). Acesta controlează, în funcție de informațiile primite de pe magistrală, două motoare pas cu pas și un afișaj LCD.

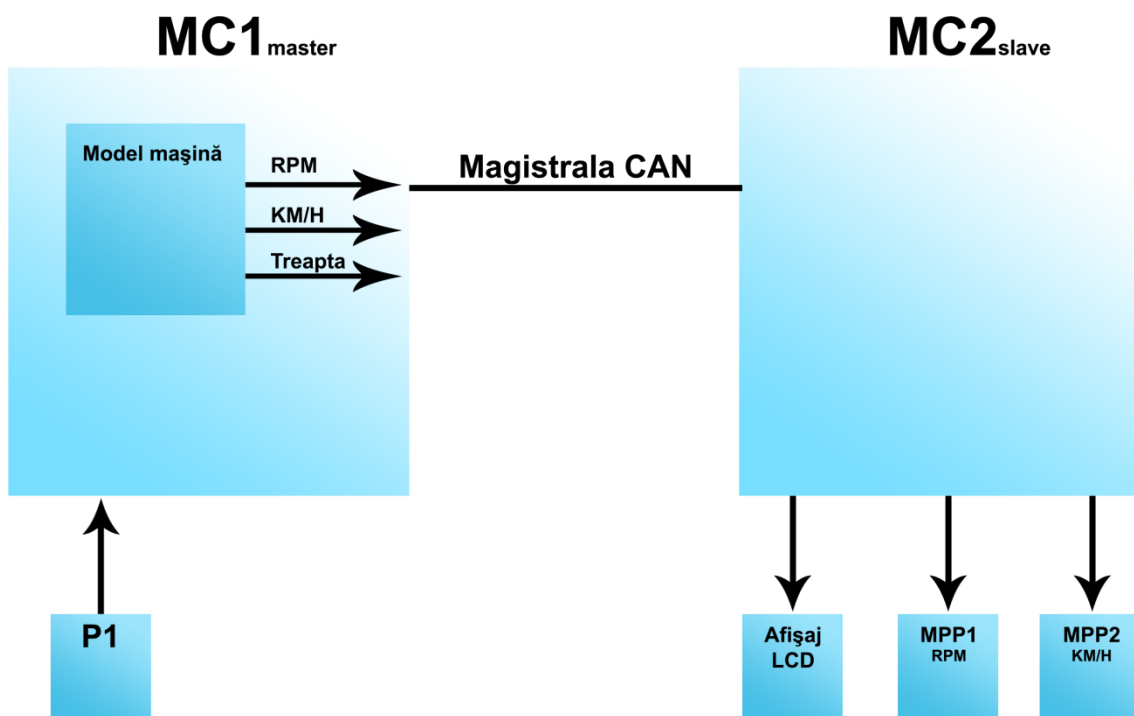


Figura 4.1

Transmiterea respectiv citirea datelor în cadrul celor două microcontrolere se face folosind timere. Timerele respective sunt periferice caracteristice controlerelor ce au diferite funcții ce pot fi utilizate. Funcția folosită în proiectul nostru se caracterizează prin numărarea unui anumit interval de timp și după trecerea timpului prestabilit se va executa instrucțiunea precizată în zona de cod din program.

În microcontrolerul MC2 controlul celor două motoare se realizează tot prin întreruperi folosind unități de timp.

În figura 4.2 este prezentată o schemă mai detaliată și dorește prezentarea componentelor utilizate în acest proiect. După cum s-a enumerat în partea de analiză, acestea sunt principalele componente utilizate.

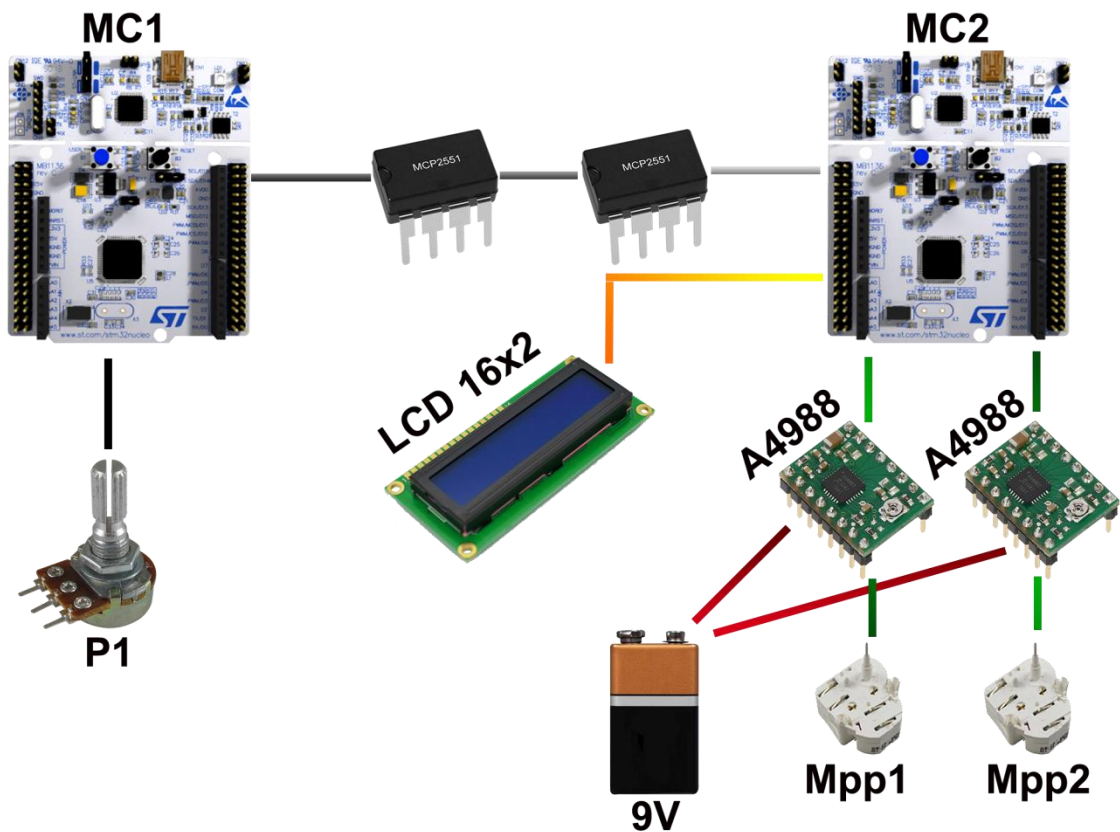


Figura 4.2

Introducerea traneiverelor și a driverelor ajută la buna funcționare a sistemului. Traneiverelor fac posibilă comunicația dintre cele două microcontrolere. Fără ele protocolul CAN nu ar putea să interpreteze magistrala la care este conectată și prin urmare, datele transmise pe aici nu ar putea fi recepționate sau transmise. Driverul A4988 face parte din mecanismul de punere în funcțiune a motorului stepper. Fără acesta ar fi fost necesare mult mai multe implementări software și elemente fizice auxiliare pentru a-l putea controla. Driverul ajută în principal la determinarea direcției și a pașilor executați de motor. Bateria de 9V este prezentă datorita consumului mai mare de energie apărut odata cu introducerea motoarelor. Alimentând motoarele de aici, consumul de energie se echilibrează și tot sistemul poate funcționa în condiții optime. În ceea ce privește display-ul, conectarea sa cu microcontrolerul necesită o multitudine de fire. Fapt datorat lipsei unei interfețe sau driver compatibil, element ce ar scădea drastic numărul de fire și ar fluidiza transmisia de date. Un astfel de element ar putea fi o interfață compatibilă cu protocolul I2C. Pinii utilizați din microcontroler s-ar reduce la doar doi: SCL și SDA, reprezentând transmiterea serială a ceasului și transmiterea serială a datelor. Pentru introducerea datelor în sistem am ales un potențiomtru deoarece este mai ușor de utilizat. Inițial s-a dorit utilizarea unui senzor cu infraroșu de distanță, dar din cauza lipsei de precizie al acestuia și al fluctuației permanente a valorilor, alegerea unei potențiomtru a fost mai benefică și mai simplă, pentru o gamă mai stabilizată de valori achiziționate.

5.1.1 STM32CubeMX

La crearea unui proiect nou în STM32CubeMX se selectează microcontrolerul folosit. După selectare o nouă fereastră va apărea ce va conține toți pinii microcontrolerului selectat anterior.

Problema este ca nu se pot folosi toti pinii oricand si oricum deoarece sunt periferice ce isi impart pini, asta ducand la o excluderi de utilizare. Exista totusi solutii pentru acest impediment, una ar fi posibilitatea mutarii unor periferice activate pe alti pini. Acest lucru se poate face dupa activare (culoarea verde a pinilor) se tine apasat butonul Ctrl, se tine apasat click stanga pe pin si apoi, cu albastru vor aparea pinii disponibili de a prelua aceasta sarcina.



5.1.2 IAR Embedded Workbench

IAR Systems este o companie de programare din Suedia ce ofera solutii de programare pentru sisteme inglobate. A fost fondat in 1983 si are sediul la Stockholm. Abrevierea vine de la Ingenjörfirman Anders Rundgren, prin traducere: Compania de Inginerie Anders Rundgren.

IAR Embedded Workbench este un IDE (Integrated Development Environment) unde toate uneltele necesare producerii aplicatiei sunt integrate (compilator C, librarii, editor, IAR C-SPY Debugger). Cu acest IDE se pot programa mai multe microcontrolere, trebuie doar descarcare pachetele pentru seriile de placute dorite.

Dupa instalare, pornirea aplicatiei se face accesand IarIdePm.exe ce se gaseste in directorul common\bin. Odata pornit, IDE-ul arata ca in figura 5.3 de mai jos:

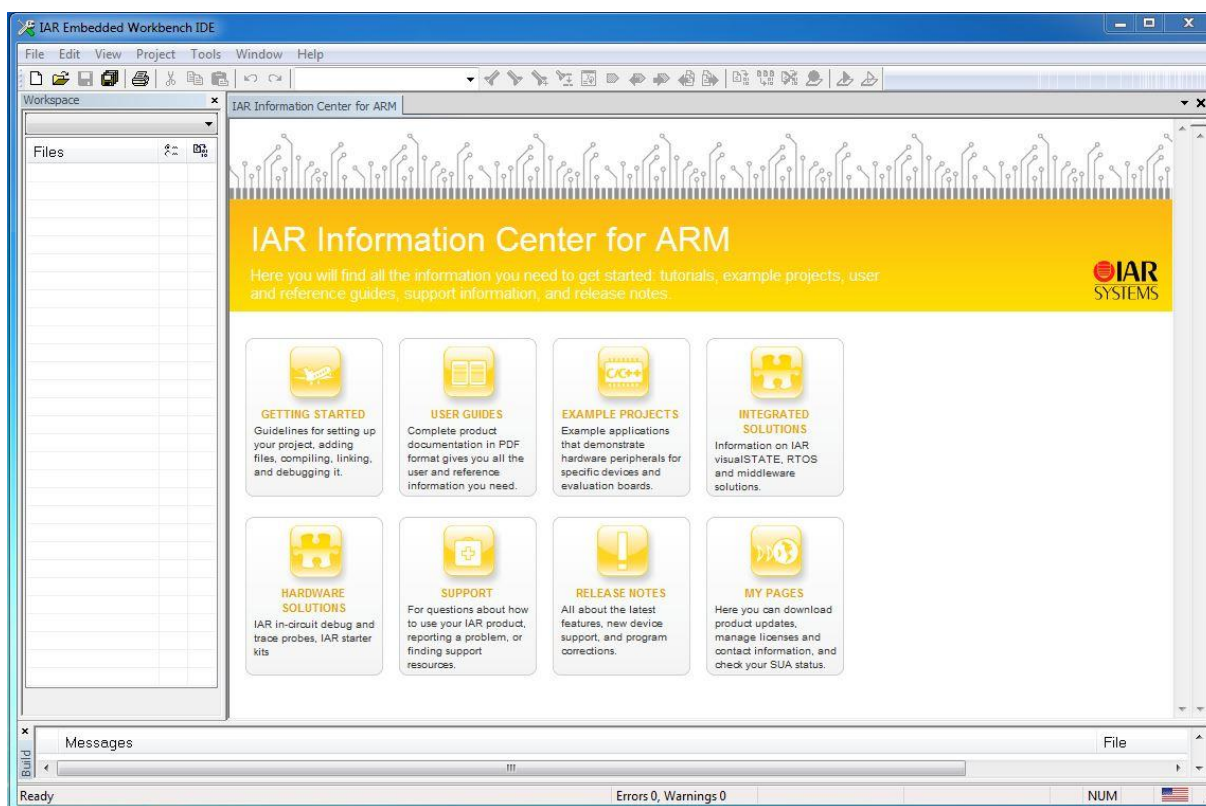






Figura 5.3

La prima deschidere a programului se va afisa centrul de informare unde se regasesc toate informatiile necesare pentru inceput (tutoriale, exemple de proiecte, suport informatic si un ghid de utilizare).

Pentru simplitate si eficienta in initializarea proiectului se recomanda folosirea programului descris anterior STM32CubeMX pentru initializarea pinilor, configurarea clock-ului si configura perifericelor folosite. Din CubeMX, dupa ce am terminat de setat si configurat, apasam butonul  ce ne va deschide o fereastră de unde ne vom seta cateva optiuni cum ar fi : numele proiectului, locatia, IDE-ul ce dorim sa-l folosim : EWARM pentru IAR (sau MDK-ARM V5 daca folosim Keil μVision 5). Setand acestea dam click pe OK si asteptam sa se deschida IAR Embedded Workbench. Odata deschis, programul ne

va afișa în stânga spațiul de lucru ce conține directoarele cu fișierele, în dreapta vom avea deschis fișierul `main.c` și jos zona de informații asupra compilării și debuggingului.

Pentru a compila programul trebuie să accesăm butonul  (cursorul trebuie să selecteze mai întâi zona de cod pentru ca butonul de compilare să fie activ). Dacă după compilare totul a decurs normal, fără erori sau warninguri, putem accesa butonul de rulare a programului  ce îl va încărca pe microcontroler.

Odată încărca programul pe controler, se vor deschide mai multe ferestre în IDE. Apăsând și butonul  se va intra în debugging mode și setând în Live Watch câteva variabile, se pot vedea cum acestea oscilează (cu roșu). În figura 5.4 se vede acest lucru:

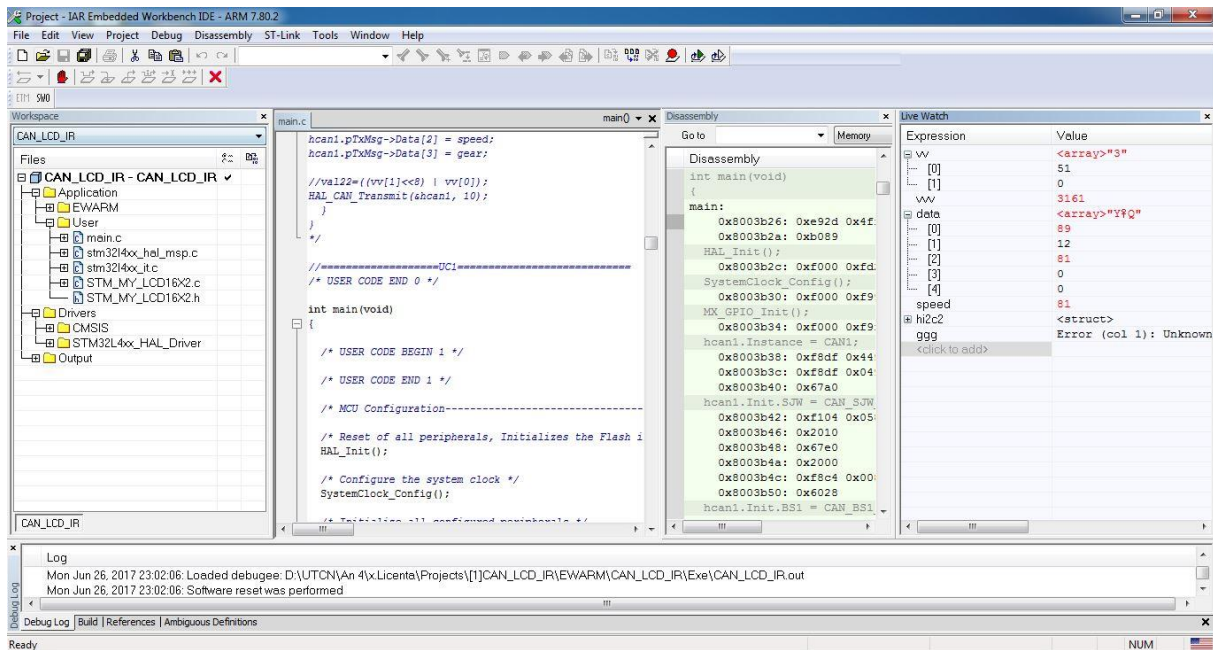


Figura 5.4

5.2 Stadiile proiectării

5.2.1 Microcontroler 1

Primul controler STM32L476RG îl vom considera microcontrolerul master din sistemul nostru deoarece el va trimite datele spre execuție către al doilea controler. Aici vor fi puține componente hardware atasate.

5.2.1.1 Achiziție date

În prima fază, vom conecta doar potentiometrul la microcontroler pentru a vedea cum primim date de la acesta. Introducem potentiometrul B50K în breadbord, legăm pinul 1 la VCC(5V) și pinul 3 GND(ground), la apoi legăm pinul 2 al potentiometrului la pinul A5 de pe placa STM. Pinul A5 se găsește în partea stângă jos a plăcii în zona CN8.

După cum se poate observa și în figura 5.5, potentiometrul este legat cu jumpere la zona de alimentare a breadboard-ului, alimentarea este oferită de controler prin cele

doua cabluri rosu si negru. Firul alb este cel de semnal, el leaga pinul 2 al potentiometrului cu pinul A5 (PC0) al microcontrolerului.

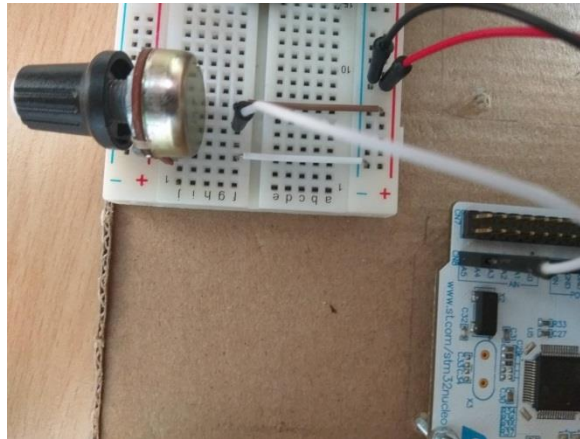


Figura 5.5

Dupa ce am terminat de conectat deschidem programul CubeMX. Aici va fi necesar sa ne instalam librariile necesare pentru familia de microcontrolere din care face parte placuta noastra. Astfel ca, intram in Help->New Libraries manager, de aici cautam STM32CubeL Releases si instalam Firmware-ul cu ultima versiune existenta. Dupa ce am terminat de instalat librariile, dam un restart la aplicatie si vom selecta placa noastra urmand succesiunea: New Project->Board Selector->Type of Board alegem Nucleo64 apoi de la MCU Series dam dublu click stanga pe NUCLEO-L476RG. În cel mai scurt timp se va deschide o fereastră ca in figura 5.1. Din partea dreapta alegem ADC1 din tabul Peripherals. Maximizam structura si in dreptul zonei IN1 apasam pe sageata si selectam IN1 Single-ended. Putem apoi observa in partea dreapta ca in zona pinului PC0 a aparut ADC1_IN1 si pinul s-a colorat verde, programul anuntandu-ne astfel ca pinul este activ si gata de a fi utilizat.

Configuratia pinului este facuta, acum urmeaza sa exportam aceste configuratii in IAR Embedded Workbench pentru a scrie codul ce il vom incarca pe microcontroler. Mergem la tabul Project si de acolo selectam Generate Code. Se va deschide o fereastră unde vom introduce un nume de proiect sub Project Name, locatia unde dorim sa salvam proiectul accesand Browse din dreptul chenarului Project Location, selectam Toolchain-ul pe care il folosim, in cazul nostru EWARM. In cele din urma trebuie sa selectam locatia librarii ce am instalat-o anterior. La ultimul chenar de jos, selectam Browse si dam locatia unde este salvata libraria pentru tipul de microcontrolere L4. Dupa ce am terminat de configurat apasam Ok si asteptam sa se deschida IAR Workbench.

Odata deschis IAR-ul deschidem Application, apoi User si in cele din urma intram in fisierul main.c unde vom incepe programarea de fiecare data. Cautam zona de cod unde avem o bucla while(1). Aceasta bucla este delimitata de niste comentarii: "USER CODE BEGIN/END WHILE" si "USER CODE BEGIN/END 3". Intre cele doua comentarii de inceput si sfarsit de cod din bucla while vom introduce doua linii de cod ce vor pornit pinul ADC setat in CubeMX si apoi va lua valoarea introdusa de potentiometru. Aceasta valoare o vom salva intr-o variabila de tip integer.

Urmatorul lucru ce trebuie să-l facem este să ne asigurăm că jumperul JP5 al microcontrolerului (figura 2.11) este conectat pe pinul 2 și 1 (vezi figura 2.12) deoarece alimentarea va fi prin cablul USB de la laptop. Conectam cablul la laptop folosind un cablu mini-USB->USB (intrarea USB 2.0 va fi în laptop, iar cea mini-USB în microcontrolerul STM32L476RG Nucleo)

Odata conectat microcontrolerul, se va aprinde LD3 (roșu) și LD1 (roșu). Din IAR vom compila programul să vedem dacă totul este bine și se poate încarca pe controler. Dacă nu primim erori sau avertizări după compilare apăsăm butonul de Run sau din tabul Project selectăm Download and Debug (CTRL+D). În program se va vedea cum fișierele se încarcă pe controler în partea de jos, iar pe controler vom vedea cum acesta intră în modul de comunicație cu laptopul și primește în memorie programul trimis de noi. După un semnal sonor din laptop și aprinderea becului verde la LD1 putem selecta din IAR butonul de ce ne introduce în zona de Debug. Odata selectat acest buton se va vedea pe microcontroler o alternanță de culori verde-roșu în ledul LD1. Acuma selectăm de sus tabul View și alegem Live Watch de la mijlocul culelei aparute. După selectare va apărea o fereastră numită cu același nume unde vom putea să vizualizăm date din variabilele utilizate de controler. În zona de <click to add> dăm click și scriem "val" apoi dăm enter. Presupunând că am salvat datele primite de la potentiometru în variabila denumită "var" de tip integer fără semn (uint32_t). Valorile variabilei "var" se pot observa în dreptul acesteia cu roșu. Ele se modifică în funcție de cum modificăm noi poziția potentiometrului.

Astfel am realizat introducerea datelor în microcontroler, date ce vor servi mai târziu ca materie primă pentru realizarea a trei mărimi: turatie, viteza de deplasare și treapta de viteza. În cele ce urmează vom vorbi despre aceste lucruri și cum le realizăm.

5.2.1.2 Modelul mașinii

În scopul interfațării cât mai exacte a unui bord auto trebuie să avem date cât mai apropiate de realitate. Acest lucru este destul de greu de obținut deoarece există foarte mulți factori și variabile ce compun o comportare de ansamblu a mașinii.

În cazul unei mașini reale accelerația, viteza mașinii, combustibilul consumat, emisiile de gaze, căldura generată toate acestea sunt particularități ce caracterizează mașina, cu cât acești parametri sunt mai buni cu atât mai mult putem spune că acel autoturism este mai bun.

Materialele folosite în construcția mașinii determină uzura ulterioară a autovehicolului, acesta fiind un alt factor ce influențează performanțele mașinii.

Zona de deplasare, înclinația șoselei, dimensiunile cauciucurilor și a jentilor, toate acestea au ca scop îngreunarea sau îmbunătățirea parametrilor pe care în această lucrare dorim să îi simulăm.

Vom minimiza acest sistem la o singură variabilă de intrare (potentiometrul) ce va simula accelerația. Determinarea, în funcție de variația potentiometrului, a parametrilor noștri, o vom face folosind câteva formule matematice și interpolări (scalări).

Idea de bază ar fi că, datele introduse aflate într-un interval de valori de 0-4095, să fie raportate la o gama de valori ce mai apoi sa fie folosita pentru controlul motoarelor stepper.

Mai trebuie sa avem in vedere că ulterior când vom folosi comunicația CAN nu vom putea trimite decat valori pe opt biți deoarece zona de date din CAN are opt spații disponibile de transfer de date de la zero la sapte si fiecare este pe un byte adica opt biți ceea ce inseamna valori de la 0 la 255.

Modelul mașinii creat pentru această aplicație constă în cateva analogii logice în prima fază. Așadar, pentru a transforma niste valori dintr-un anumit interval in alte valori raportate la un alt interval va trebui sa ne gandim astfel: daca datele ce le avem inițial sunt în intervalul închis $[0,a]$ și vrem să le convertim în $[0,b]$ trebuie să scalăm cu un anumit raport. Așa că, "0" corespunde lui "0", "a" corespunde lui "b" si un numar din intervalul datelor de intrare sa zicem "t" va fi egal cu:

$$t_{\text{nou}} = t_{\text{vechi}} * \left(\frac{b}{a}\right) \quad (5.1)$$

De exemplu:

Intrare: $[0,5]$ Iesire: $[0,8]$

Fie $t=3$ (din intrare). Scalat la intervalul de ieșire el va deveni:

$$t = 3 * \left(\frac{8}{5}\right) = 4.8 \quad (5.2)$$

Pentru o abordare mai concretă vom avea în vedere următoarele aspecte. Un interval de intrare dupa cum urmeaza : $[in_{\text{begin}}, in_{\text{end}}]$ are un total de numere egal cu:

$$in_{\text{end}} - in_{\text{begin}} + 1 \quad (5.3)$$

Poate fi considerat ca un interval de:

$$[0, in_{\text{end}} - in_{\text{begin}}] \quad (5.4)$$

Ieșirea poate avea aceeași analogie cu intervalul (5.4).

O intrare "in" este egala cu:

$$a(in_{\text{nou}}) = in - in_{\text{begin}} \quad (5.5)$$

Din analogia ecuației (5.1) reiese ca :

$$b = (in - in_{\text{begin}}) * \frac{out_{\text{end}} - out_{\text{begin}}}{in_{\text{end}} - in_{\text{begin}}} \quad (5.6)$$

Dacă ne uităm la return-ul funcției "map" folosita la programarea din Arduino o sa constatam următoarele:

Ca declarare:

`long map(long val, long in_min, long in_max, long out_min, long out_max)`

Unde val este valoarea ce urmează a fi scalată, in_min minimul posibil din intervalul lui val, in_max maximul din același interval, iar out_min și out_max sunt minimul și maximul din intervalul în care vrem să scalăm valoarea val.

Returnarea funcției map va fi următoarea:

$$\frac{(val - in_{min}) * (out_{max} - out_{min})}{in_{max} - in_{min}} + out_{min} \quad (5.7)$$

Față de analogiile făcute anterior și anume ajungerea la ecuația (5.6) observăm apariția ultimului termen de adunare. Prezența lui nu face decât ca atunci când se calculează noul termen să se înceapă de la prima valoare dorită din intervalul în care vom scala.

De exemplu:

Intrare: [4,16] Ieșire: [1,22]

Fie t=9 (din intrare). Scalat la intervalul de ieșire el va deveni, conform (5.7):

$$\frac{(9 - 4) * (22 - 1)}{16 - 4} + 1 = 9.75 \quad (5.8)$$

Noul "t" va fi pentru intervalul de ieșire 9.75 față de intervalul de intrare unde era 9.

Există diverse ecuații între viteză și turajie. Aceste ecuații le putem folosi pentru a ne determina o valoare în funcție de cealaltă. Noi având turatia, data de la potentiometru, avem nevoie doar de o formula ce ne va calcula viteza

$$viteza = \left(\frac{val * TireDim}{GearRatio * 208} \right) \quad (5.9)$$

Unde val este valoarea de la potentiometru, TireDim este dimensiunea anvelopei, GearRatio este un număr egal cu raportul de dinți dintre două roți dintate alăturate ale cutiei de viteze și numărul vine de la un raport de conversie a valorilor de măsură a variabilelor, acest fiind pentru km/h (pentru mile pe oră se folosește 336).

Deduția matematică a acestei formule se bazează pe ce dorim să aflăm în funcție de ce avem. Prin urmare, considerăm că avem nevoie de viteza măsurată în kilometri pe oră:

$$\frac{Km}{or\acute{a}} = \frac{RotM}{Min} * \frac{60Min}{h} * Dr * \frac{1km}{39370} * \pi * \frac{1}{Gr} \quad (5.10)$$

Unde, RotM=rotațiile motorului(rpm), Dr=diametrul roții(inci), Gr=gear ratio. Acel număr "39370" reprezintă corespondența în inci a unui kilometru.

$$\frac{Km}{or\acute{a}} = \frac{RotM * Dr * 1km * 188.4955}{GearRatio * 1h * 39370.08} \quad (5.11)$$

Simplificăm ecuația (5.11) cu 188.4955 și obținem:

$$\frac{\text{Km}}{\text{oră}} = \frac{\text{RotM} * \text{Dr} * 1\text{km} * \frac{188.4955}{188.4955}}{\text{GearRatio} * 1\text{h} * \frac{39370.08}{188.4955}} \quad (5.12)$$

Din (5.13) reiese ecuația vitezei folosite în modelul matematic:

$$\frac{\text{Km}}{\text{oră}} = \frac{\text{RotM} * \text{Dr} * 1\text{km} *}{\text{GearRatio} * 208.86} \quad (5.13)$$

Bazandu-ne pe aceste ecuații putem implementa un model matematic de recalculare a turatiei, vitezei si a treptelor de viteza pentru a avea in spatele afisarii bordului auto si o simulare a unei cutii de viteze automate.

Implementarea din IAR a modelului incepe prin a declara doua variabile de tip integer fara semn (uint32_t): rpm și speed. În prima variabilă “rpm” vom introduce turația în rotații pe minut și în “speed” viteza de deplasare a mașinii în kilometrii pe oră. Pentru citirea de la potențiometrul folosim un timer ce ne va citi cu o frecvență mare datele de la dipozitivul de intrare și le va salva în variabila “val”. Vom mai avea și un vector cu 5 elemente numit GearRatio ce va avea datele unui autoturism Volkswagen Passat din 1993(3.3, 1.944, 1.308, 1.034, 0.838), o variabila “gear” ce va indica poziția din vector a elementului dorit și TireDi, un macrou definit cu valoarea 16(reprezentând dimensiunea gentii masinii).

Aici vom avea o serie de condiții în funcție de rpm:

- Daca rpm ≤ 1600 atunci

Gear=0;

$$\text{speed} = (\text{int}) \left(\frac{\text{val} * \text{TireDim}}{\text{GearRatio}[\text{gear}] * 208} \right) \quad (5.14)$$

- Daca ((rpm > 1600) && (rpm ≤ 2300)) atunci

Gear=1;

$$\text{speed} = (\text{int}) \left(\frac{\text{val} * \text{TireDim}}{\text{GearRatio}[\text{gear}] * 208} \right) \quad (5.15)$$

- Daca ((rpm > 2300) && (rpm ≤ 3000)) atunci

Gear=2;

$$\text{speed} = (\text{int}) \left(\frac{\text{val} * \text{TireDim}}{\text{GearRatio}[\text{gear}] * 208} \right) \quad (5.16)$$

- Daca ((rpm > 3000) && (rpm ≤ 4000)) atunci

Gear=3;

$$\text{speed} = (\text{int}) \left(\frac{\text{val} * \text{TireDim}}{\text{GearRatio}[\text{gear}] * 208} \right) \quad (5.17)$$

- Daca $((rpm > 4000) \&\& (rpm \leq 5000))$ atunci

Gear=4;

$$\text{speed} = (\text{int}) \left(\frac{\text{val} * \text{TireDim}}{\text{GearRatio}[\text{gear}] * 208} \right) \quad (5.18)$$

Urmeaza implementarea acestui model pe microcontroler si apoi trimiterea datelor rezultate spre microcontrolerul numarul 2 pe magistrala CAN.

Deschidem din proiectul creat înainte, fișierul cu extensia .ioc reprezentand fisierul STM32CubeMX. Odata deschis vom activa din partea stanga timerul TIM2 astfel: maximizam structura lui TIM2 si de la Clock Source alegem Internal Clock, in tabul Clock Configuration verificam daca este bifat butonul HIS si in tabul Configuration mergem la sectiunea Control. Accesam TIM2, setam Prescaler-ul la 48000 si Counter Period la 1. Din tabul configuratorului TIM2 selectam NVIC Settings si bifam Enable din linia "TIM2 global interrupt". Aceste setari folosesc in continuare la implementarea in IAR a codului pentru transmisia de date.

5.2.1.2.1 Model real

Pentru rezolvarea problemei modelului matematic al masinii am încercat implementarea unei secțiuni de cod folosit pe un controler dintr-un cluster de serie. Acest cod simula în funcție de accelerația introdusă o cutie de viteze automate. Codul folosind librării externe am încercat reducerea lui până când am reușit doar cu funcția de interpolare(scalarea pe care am utilizat-o și în această lucrare), funcția modelului mașinii(calcularea turației, vitezei) si vectorii de mapping folosiți , să ruleze intr-un mediu de programare C (CodeBlocks). În CodeBlocks totul a decurs normal, dar la introducerea in IAR au aparut probleme. Deși se comporta bine in primele secunde de la rulare, intervenea o creștere exagerată a unor variabile ce au determinat creșterea vitezei de la 40-60 până la 600-1200 în doar cateva recalculari.

5.2.1.3 Configurația CAN

Dupa ce am terminat de configurat timerul ne intoarcem la tabul de Pinout si activam protocolul CAN. Activarea acestui protocol va seta automat pinii PA11 si PA12 ca fiind CAN1_RX respectiv CAN1_TX. Dorim sa mutam pinii respectivi in alta zona pentru a fi mult mai accesibili in zona hardware cand vom conecta traneiverul la controler. Pentru a muta pinii respectivi tinem apasat butonul CTRL si cu click stanga PA11 il vom putea muta in cealalta zona accesibila pentru CAN1_RX si anume in PB8, acelasi lucru facem si cu pinul PA12 ce il vom muta pe portul PB9 care va fi CAN1_TX.

Acum intram in tabul Configuration, intram la Connectivity pe CAN1 si astfel am intrat in fereastra de configurare a protocolului CAN de pe microcontroler. Urmatoarele configuratii au fost facute: Prescaler (for Time Quantum) a fost setat la 16, Time Quanta in Bit Segment 1 are valoarea 3 Times, iar Time Quanta in Bit Segment 2 are valoarea 5 Times, ReSynchronization Jump Width setam pe 1 Time. In optiunile din Basic Parameters toate vor fi Disale iar Operating Mode va fi Normal.

De asemenea nu vom uita sa pornim oscilatorul. Din Pinout maximizam RCC si la High Speed Clock (HSE) alegem Crystal/Ceramic Resonator.

În tabul Configuration-> System vom intra in NVIC de unde vom bifa Enable in dreptul CAN1 RX0 interrupt in cazul in care nu este bifat.

In ceea ce priveste configuratia ceasului setarile pot fi vazute in figura 5.6 :

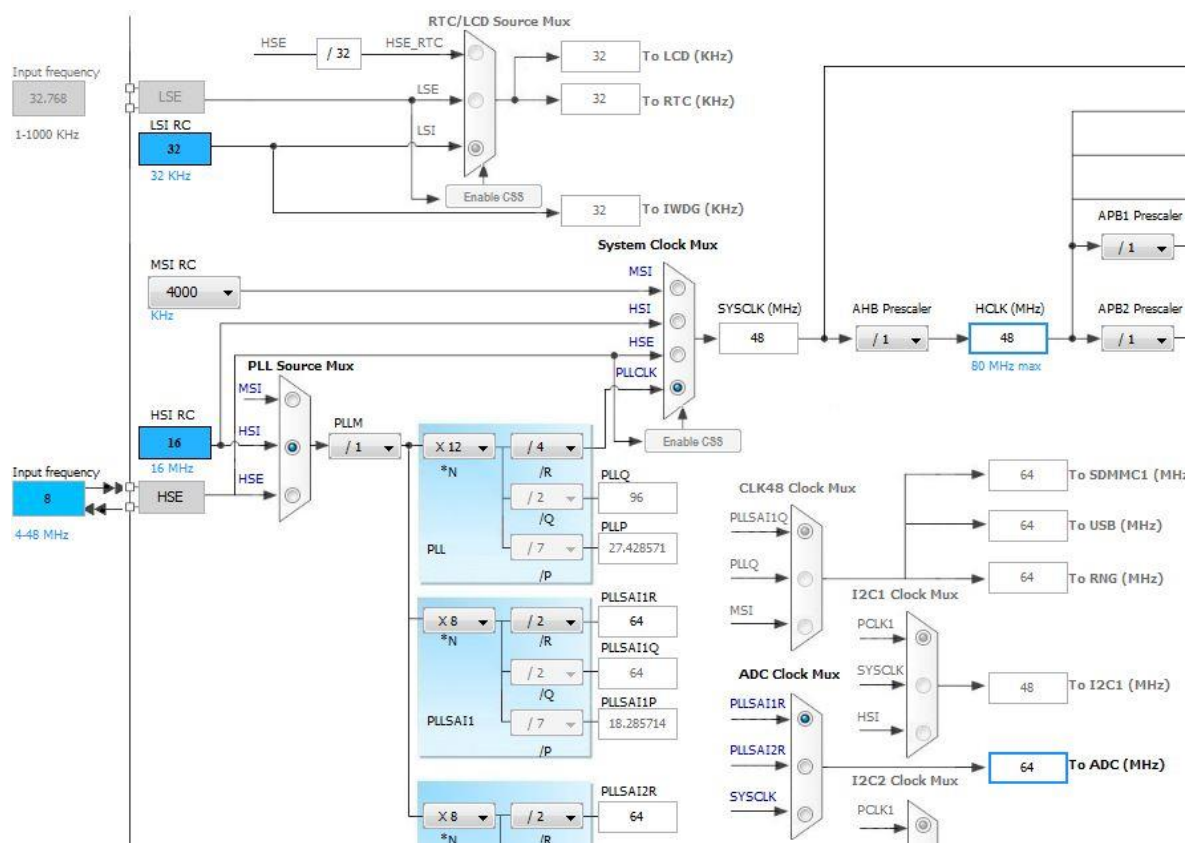


Figura 5.6

Dupa ce am facut toate acestea generam configuratia in codul C pentru IAR. Ajunsi in IAR intram in main.c de unde continuam configurarea inceputa in CubeMX pentru protocolul CAN. Prin urmare, ne setam trei structuri de date la inceputul programului in zona dedicata variabilelor private (*/* Private variables*/*). Prima data definim o structura de timp CAN_FilterConfTypeDef ce va contine filtrele folosite pentru protocol.

Urmatoarele doua sunt structuri pentru transmiterea si receptia datelor definite prin structuri de timp CanTxMsgTypeDef si CanRxMsgTypeDef. Denumirea acestor structuri este recomandat sa fie cat mai expresive cu scopul de a nu ne incurca pe viitor. Astfel ca pentru prima structura putem folosi FilterConf si pentru urmatoarele doua TxMesaj respectiv RxMesaj.

Restul configurarii protocolului CAN consta in setarea anumitor valori filtrelor din structurile amintite si declarate mai sus. Între comentariile USER CODE BEGIN/END 2 vom scrie tot ce este nevoie pentru finalizarea configuratiei protocolului.

În primul rand, dam adresele variabilelor TxMesaj si RxMesaj zonelor de pointer ale structuri hcan1 prin atribuirea hcan1.pTxMsg = &TxMesaj, acelasi lucru si pentru

receptia transmisiei Rx. Apoi vom seta valori mai multor campuri din structura de date definita mai sus FilterConf. Prin numele structurii declarate "." Si apoi denumirea filtrului vom face aceste modificari.

- FilterNumber ce va fi setat "0" (specifica filtrul ce va fi initializat, valori permise intre 0 si 27);
- FilterMode = CAN_FILTERMODE_IDLIST (specifica modul filtrului utilizat poate fi ori LIST ori MASK);
- FilterScale = CAN_FILTERSCALE_32BIT (poate fi unul de 32 de biți sau doua de 16);
- FilterIdHigh aici setam adresa protocolului CAN de pe acest microcontroler. Vom seta o valoare de adresa intre 0x0 si 0xFFFF. Alegem valoarea 0x244<<5 (in decimal va fi 18560 si in hexa 4880) ce va reprezenta adresa microcontrolerului master;
- FilterIdLow (numarul de identificare al filtrului) il vom seta 0;
- FilterMaskIdHigh = 0;
- FilterMaskIdLow = 0;
- FilterFIFOAssignment = 0 (specifica care din FIFO le vom folosi, 1 sau 0);
- FilterActivation il vom activa dandu-i valoare = ENABLE deoarece dorim sa folosim parte din filtrele puse la dispozitie de protocol;
- BankNumber seteaza de unde incepe zona de filtre (valori intre 0 si 28), noi vom seta 14;

Acestea au fost filtrele folosite, in continuare le vom introduce in structura protocolului pentru receptie. Functia HAL_CAN_ConfigFilter va seta in hcan1 filtrele initializate mai cu ajutorul structurii FilterConf definita in zona de variabile.

Pentru a receptiona mesaje trebuie activata o rutina de intreruperi responsabila de citirea acestor mesaj ce urmeaza sa fie primite. Ea se numeste HAL_CAN_Receive_IT avand ca parametrii hcan1 si CANFIFO0.

În ultimele doua functii, în afara de CANFIFO0, restul parametrilor sunt transmisi ca adrese, cu "&" in fata lor.

Datele transmise spre magistrala vor avea nevoie si de adresa celualt microcontroler unde vor ajunge datele. Asadar in structura hcan1, campul pTxMsg, variabila StdId va memora valoarea 0x245 ce va reprezenta "partenerul de discutie". Tot in cadrul acestui camp (pTxMsg) se vor initializa inca trei variabile:

- RTR = CAN_RTR_DATA (se va utiliza zona de date din frame si nu zona remote);
- IDE = CAN_ID_STD (aici se alege intre tipruiel de identificator ale CAN-ului Standard Id sau Extended Id, in cazul nostru folosim Standard);

- DLC = 4 (specifica cati bytes de date vor fi trimisi pe frame, noi am ales 4 pentru ca de atatea avem nevoie, se pot alege pana la 8 bytes de date)

5.2.1.4 Trimiterea datelor pe magistrala CAN

Ultimele doua subcapitole au descris implementarea modelului matematic si configurarea protocolului CAN. In acest subcapitol vom transmite datele acestea pe magistrala CAN controlerului MC2.

Avand toate configuratiile facute putem programa direct din IAR transmisia datelor. În partea de inceput a main.c între comentariile USER CODE BEGIN/END 0 vom apela o functie numita HAL_TIM_PeriodElapsedCallback. Aceasta functie se autoapeleaza in functie de conditiile pune in interiorul ei.

Creăm o funcție de tip void cu denumirea HAL_TIM_PeriodElapsedCallback si cu parametrul TIM_HandleTypeDef *htim. În interiorul functiei punem conditia ca odata avuta loc intreruperea timerului TIM2 sa se execute comenzile avute in interior. TIM2 l-am setat in capitolul 5.2.1.2 astfel incat se apeleze la fiecare 1 milisecunda. Condiția respectivă arată așa :

```
if (htim -> Instance == TIM2)
{
    //model matematic
    //date trimise spre MC2
}
```

În felul acesta evitam sa punem in bucla while codul respectiv, avem un control mai mare asupra lui si este mult mai eficient si ingrijit. Înainte de a transmite datele pe CAN trebuie sa le pregatim. Aceasta pregatire este necesara datorita limitarii de biți ce pot fi transmiși pe o linie de date a frame-ului (maxim 255). Astfel declarăm un vector cu doua elemente vv[2] de tip char (char pentru ca este de aceea dimensiune ca o zona de date).

Valoarea lui rpm este pe 16 biți deoarece 800 reprezentat in binar este pe 16 biți dupa cum urmeaza : 0000001100100000. Prin urmare in vectorul declarat mai sus "vv" vom pune in primul element vv[0] primii 8 biți din rpm astfel : vv[0]=rpm. Ceilalti 8 biți ii vom pune in vv[1] deplasand la dreapta biții din rpm. Atribuirea in vector va fi urmatoarea: vv[1]=rpm>>8;

În felul acesta am salvat o valoare pe 16 biți in doua locatii de cate 8 biți. Considerand secventa binara ca avand primul bit din dreapta cel mai nesemnificativ bit si primul bit din stanga cel mai semnificativ bit dupa cum este precizat si mai jos:

Cel mai semnificativ bit=0000001100100000=cel mai nesemnificativ bit.

Transmisia spre magistrala CAN se va produce atribuind aceste valori in zona de date a structurii hcan1 in campul pTxMsg la zona de date dorita. Pentru transmiterea rpm-ului selectam structura,campul si zona de date dupa cum urmeaza:

- hcan1.pTxMsg->Data[0]=vv[0] (aici am adaugat in frame-ul ce urmeaza a fi trimis primii 8 biți de date ai rpm-ului, valoarea din vv[0] a fost pusa in zona "0" de date);
- hcan1.pTxMsg->Data[1]=vv[1] (aici adaugam ceilalti 8 biți ai rpm-ului atribuind valoarea din vv[1] în zona de date numarul "1");
- hcan1.pTxMsg->Data[2]=speed (aceasta atribuire in zona de date "2" constituie viteza de deplasare a masinii, aflandu-se in marja 0-180 nu este nevoie sa o împartim deoarece este in limitele dimensiunii zone de date);

Ultima instructiune ce trebuie precizata este apelarea functiei de transmitere a mesajului ce reprezinta frame-ul CAN-ului. Utilizand HAL_CAN_Transmit si dând ca argumente adresa structurii hcan1 si o valoare de timeout, mesajul creat se trimite pe magistrala CAN ca mai apoi sa ajunga la celalalt microcontroler. Salvam fisierul si inchidem aplicatia.

5.2.1.5 Legatura fizica a comunicăției CAN

Înainte de a rula programul pe microcontroler trebuie sa stabilim si partea hardware. Conexiunea fizica cu transceiverele si cu celalalt microcontroler este necesara.

Prin urmare, ne asiguram ca avem microcontrolerul deconectat de la surse de alimentare. În breadboard introducem transceiverul MCP2551 astfel incat pinii sa fie pe o parte si pe cealalta a canalului longitudinal despartitor din mijlocul breadboardului. Apoi suntem atenti la schema cu pinii disponibili a transceiverului (Fig 5.7). Conectam pinul TXD al lui MCP2551 la Tx-ul microcontrolerului reprezentat de pinul D14 si pinul RXD de la transceiver la pinul D15 al STM32-ului. Apoi conectam pinul numarul 2 al transceiverului Vss la GND si pinul 3 al transceiverului Vdd la 5V. Pinul 8 si anume Rs il vom conecta la GND-ul Vss-ului in serie cu o rezistenta de 1kΩ.

Pinii 7(CANH) si 8(CANL) ai transceiverului îi vom conecta dupa cum urmeaza: CANH al primului transceiver se va conecta cu CANH de la al doilea transceiver si CANL de la primul la CANL de la al doilea. În aceste ultime doua legaturi se va include si doua rezistenta de 120Ω una legata la intrarile primului transceiver si cea de-a doua rezistenta la cel de-al doilea transceiver. (aceasta conectare se va face dupa ce vom pozitiona si transceiverul numarul 2 pe placa breadboard, detalii ce vor fi precizate în urmatorul capitol). În figura de mai jos se poate vedea modul de conectare prezentat mai sus.

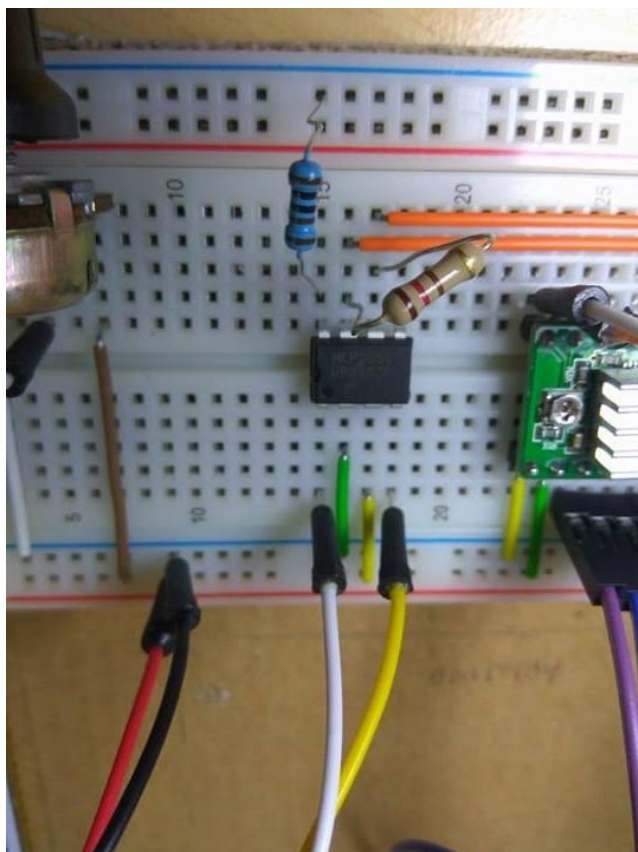


Figura 5.7

5.2.2 Microcontroler 2

5.2.2.1 Continuarea legaturii hardware a comunicației CAN

La sfarsitul ultimului subcapitol am prezentat modul de conectare al trancieverelor si al microcontrolerului master. În continuare vom conecta si trancieverul al doilea cu microcontrolerul slave.

Introducem trancieverul al doilea MCP2551 în cealalta parte a breadboardului. La fel cum l-am introdus si pe primul. Conectam conform figurii 5.8 Vss la GND, Vdd la 5 V, RS la GND cu o rezistenta de 1k Ω legata in serie. Conectam pinii TXD si RXD de la tranciever la pinii D14 respectiv D15 microcontrolerului slave.

În ceea ce priveste CANH si CANL se va proceda cum am explicat la sfarsitul subcapitolului 5.2.1.5 unde am precizat exact aceasta conexiune.

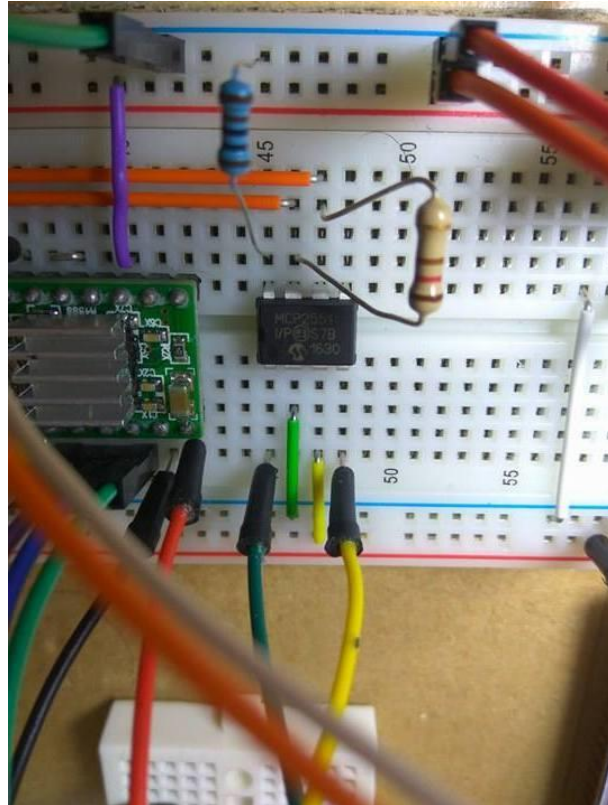


Figura 5.8

Pentru o înțelegere de ansamblu a implementării hardware a comunicației CAN se poate studia schema de mai jos

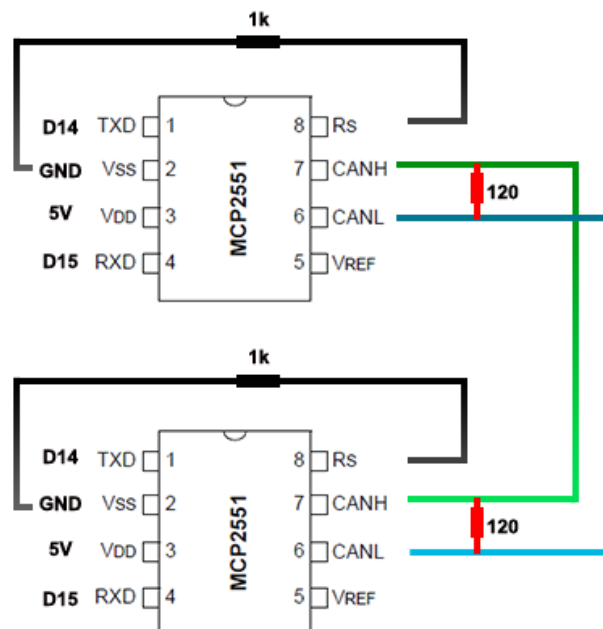


Figura 5.9

Alimentările celor două microcontrolere se va face cu grijă și respectând următorii pași pentru a nu deteriora dispozitivele:

- Se determină ce controler dorim să alimentăm deoarece dacă îl alimentăm direct de la laptop putem folosi debuggerul.
- Dacă alimentăm microcontrolerul master va trebui să ne asigurăm că jumperul JP5 se află pe pinul 1 și 2 (figura 2.12). Conectăm în zona de alimentare a breadboardului la "+" cu 5V și la "-" cu GND de pe placă. La microcontrolerul slave vom pune jumperul JP5 pe pinul 2 și 3 (figura 2.13). Conectăm în zona de alimentare a breadboardului la "+" vom conecta pinul VIN și la "-" un pin de GND.
- Dacă alimentăm microcontrolerul slave va trebui să ne asigurăm că jumperul JP5 se află pe pinul 1 și 2 (figura 2.12). Conectăm în zona de alimentare a breadboardului la "+" cu 5V și la "-" cu GND de pe placă. La microcontrolerul master vom pune jumperul JP5 pe pinul 2 și 3 (figura 2.13). Conectăm în zona de alimentare a breadboardului la "+" vom conecta pinul VIN și la "-" un pin de GND.

5.2.2.2 Receptia datelor de pe magistrala CAN

Implementarea în program va necesita în prima fază copierea fișierului creat pentru microcontrolerul master. Astfel că ne vom duce în zona unde avem salvat fișierul de unde am configurat controlerul 1 și îl vom copia în întregime dându-i alt nume. În acest nou director vom accesa CubeMX de unde vom debifa ADC1, pentru că nu mai avem nevoie de el pe acest microcontroler. Setările rămase sunt bune pentru a testa comunicația CAN. Mergem la Project și apăsăm Generate Code. După ce fereastra din IAR a fost deschisă mergem în Application->User->main.c unde vom continua scrierea codului.

Primul lucru pe care trebuie să-l facem este de a utiliza timerul TIM2 rămas activ de la microcontrolerul precedent. În zona de cod USER CODE BEGIN/END 0 folosim funcția HAL_TIM_PeriodElapsedCallback în care vom citi datele venite pe magistrală. Asemănător transmiterii datelor din microcontrolerul master, aici în microcontrolerul slave vom apela aceeași funcție de timp și aceeași condiție if (htim -> Instance == TIM2) doar că vom schimba de data asta câmpurile de transmitere cu cele de receptie.

Asadar vom apela funcția de întreruperi HAL_CAN_Receive_IT care va avea doi parametri, adresa mesajului ce va fi primit hcan1 și CAN_FIFO0, coada unde vor intra mesajele și vor ieși pe baza regulii de primul intrat primul iese. Apoi această funcție va fi urmată de patru linii de cod ce vor reprezenta citirea datelor din frame.

- `v[0]=hcan1.pRxMsg->Data[0]` (în acest element se va salva prima parte de 8 biți a rpm-ului)
- `v[1]=hcan1.pRxMsg->Data[1]` (aici se va salva a doua parte a rpm-ului)
- `v[2]=hcan1.pRxMsg->Data[2]` (viteza în km/h)

Pregătirea "rpm" este necesară și aici că atunci când am realizat transmiterea datelor. Declaram o variabilă pe 16 biți numită rpm în care salvăm imediat după ce am citit datele din frame sub următoarea formă valoarea: `rpm=((v[1]<<8) | v[0])`. Salvând astfel nu facem decât să mutăm la stânga 8 biți elementul 1 din vector și să-l adunăm cu

elementul 0 folosind operatorul sau "|". Mai departe atribuim variabilelor declarate pe 8 biți `speed=v[2]` și `gear=[3]`.

În ceea ce privește configurarea CAN, mai trebuie să modificăm unele detalii referitoare la adresa microcontrolerelor. În zona USER CODE BEGIN/END 2 căutăm `FilterIdHigh` și înlocuim `0x245` cu `0x244` și mai jos `StdId` înlocuim `0x244` cu `0x245`. Astfel va fi terminată configurarea între cele două controlere prin protocolul CAN.


Încă o linie de cod importantă trebuie precizată, aceasta fiind pornirea bazei de întreruperi pentru TIM2. Acest lucru îl facem chiar înainte de buclă `while(1)` apelând funcția cu un singur argument (adresa lui TIM2): `HAL_TIM_Base_Start_IT(&htim2)`.

Nu rămâne decât să încărcăm programele pe microcontrolere și să testăm dacă informațiile se transmit cum trebuie.

Deschidem din directorul de proiecte, proiectul microcontrolerului 1, intrăm în folderul EWARM și deschidem fișierul `Project.eww`, intrăm în fișierul `main.c` din User și pornim TIM2 și acolo folosind aceeași funcție ca la microcontrolerul 2: `HAL_TIM_Base_Start_IT(&htim2)`.

Legăm cablul între laptop și microcontrolerul master de la un port USB de la laptop și portul CN1 de la microcontroler. Odată aprinse becurile pe controler apăsăm butonul de încărcare a programului microcontrolerului master pe acesta. După terminare schimbăm și cuplăm celălalt controler la laptop de unde încărcăm programul microcontrolerului slave. Va trebui să avem grijă în permanență cum schimbăm jumperele și cum conectăm firele de alimentare (după cum am precizat la implementarea hardware de la 5.2.21-jos) și să avem grijă în program ca adresele controlerelor să fie bine precizate:

- `mC1` cu `FilterIdHigh=244` și `StdId=245`;
- `mC2` cu `FilterIdHigh=245` și `StdId=244`;

Verificarea datelor o vom face cu debugg mode-ul activ. Având un singur laptop la dispoziție vom face această verificare pe rand. Conectăm microcontrolerul master la laptop, rulăm programul lui din nou din IAR și după ce s-a încărcat apăsăm butonul . Deschidem Live Watch-ul și în fereastra apărută scriem `hcan1` și pe alta linie `val`. Maximizând frame-ul `hcan1`, apoi maximizând zona de transmisie de date vom putea vedea un câmp numit "Data" unde se pot vizualiza zonele activate de date ce transmit informația și cum se modifică valorile lor în funcție de potentiometru. În comparație se poate vedea și variabila `val`, prima variabilă ce salvează datele citite de la potentiometru, cu variabilele de după intrarea în modelul mașinii: `rpm`, `speed` și `gear`.

Dacă totul arată în regulă și datele se transmit fereastra de live watch va arăta ca în figura următoare:

val	2646	0x20000044
hcan1	<struct>	0x20000004
Instance	0x40006400	0x20000004
Init	<struct>	0x20000008
pTxMsg	TxMessage (0...	0x20000034
StdId	581	0x20000560
ExtId	0	0x20000564
IDE	0	0x20000568
RTR	0	0x2000056C
DLC	4	0x20000570
Data	<array>"011"	0x20000574
[0]	32	0x20000574
[1]	2	0x20000575
[2]	107	0x20000576
[3]	4	0x20000577
[4]	0	0x20000578
[5]	0	0x20000579
[6]	0	0x2000057A
[7]	0	0x2000057B
pRxMsg	RxMessage (0...	0x20000038
State	HAL_CAN_STAT...	0x2000003C
Lock	HAL_UNLOCKED	0x2000003D
ErrorCode	0	0x20000040

Figura 5.10

Daca vom conecta microcontrolerul slave (pe viitor vom mentie acest controler conectat la alimentare) la laptop o buna afisare a datelor va fi vazuta astfel:

hcan1	<struct>
Instance	0x40006400
Init	<struct>
pTxMsg	536872544
pRxMsg	536872572
StdId	581
ExtId	0
IDE	0
RTR	0
DLC	4
Data	<array>"011"
[0]	51
[1]	2
[2]	109
[3]	4
[4]	0
[5]	0
[6]	0
[7]	0
FMI	0
FIFONumber	0
State	HAL_CAN_STATE_READY
Lock	HAL_UNLOCKED
ErrorCode	0

Figura 5.9

O aplicatie de test, unde se poate vedea in mediul fizic si nu doar prin debugger cum are loc comunicarea pe magistrala CAN intre cele doua microcontrolere. O astfel de aplicatie usor si rapid de facut si implementat consta în construirea unui numarator realizat din leduri. Astfel ca la fiecare intrerupere de la butonul de USER al unei placi se va aprinde un bec legat la cealalta placa si invers. Becurile respective vor fi pozitionate pe linnie, in ordine, de la cel mai nesemnificativ bit pana la cel mai semnificativ bit.

În aceasta aplicatie se pot folosi patru leduri pentru fiecare microcontroler, numarand un total de opt leduri ce vor avea nevoie de inca pt rezistente de 1k Ω .

O abordare aproximativa are loc si in aceasta aplicatie cum am abordat comunicatia CAN in proiectul nostru. Ce difere sunt valoarea lui DLC=1 deoarece numaratorul nostru odata ajuns la 15 ceea ce va insemna toate cele patru becuri aprinse se va reseta trecand la 0000. El reprezentand in binar doar de la 0000 la 1111 in afisarea pe leduri. În IAR implementarea codului sa facut astfel: in main.c s-a configurat precum am configurat si in proiectul nostru si in stm32l4xx_it.c s-a facut restul programarii.

Aici, in functia CAN1_RX0_IRQHandler am scris urmatoarele: HAL_CAN_Receive_IT cu adresa frame-ului si FIFO0 si GPIOB->ODR ce va semnal de activare ledurilor citind mesajul din hcan1.pRxMsg si anume cu 12 poziti deplasate la stanga Data[0]<<12 pentru a prinde doar numerele de care am amintit mai sus.

În rutina de intreruperi a butonului de USER activat in CubeMX vom pune o conditie, daca butonul este sau nu apasat, se va intra intr-o zona de instructiuni constand in incrementarea unei variabile ce va fi pusa in zona de date a campului de transmitere, iar apoi frame-ul va fi trimis.

Mai jos avem schema acestui mini-proiect:

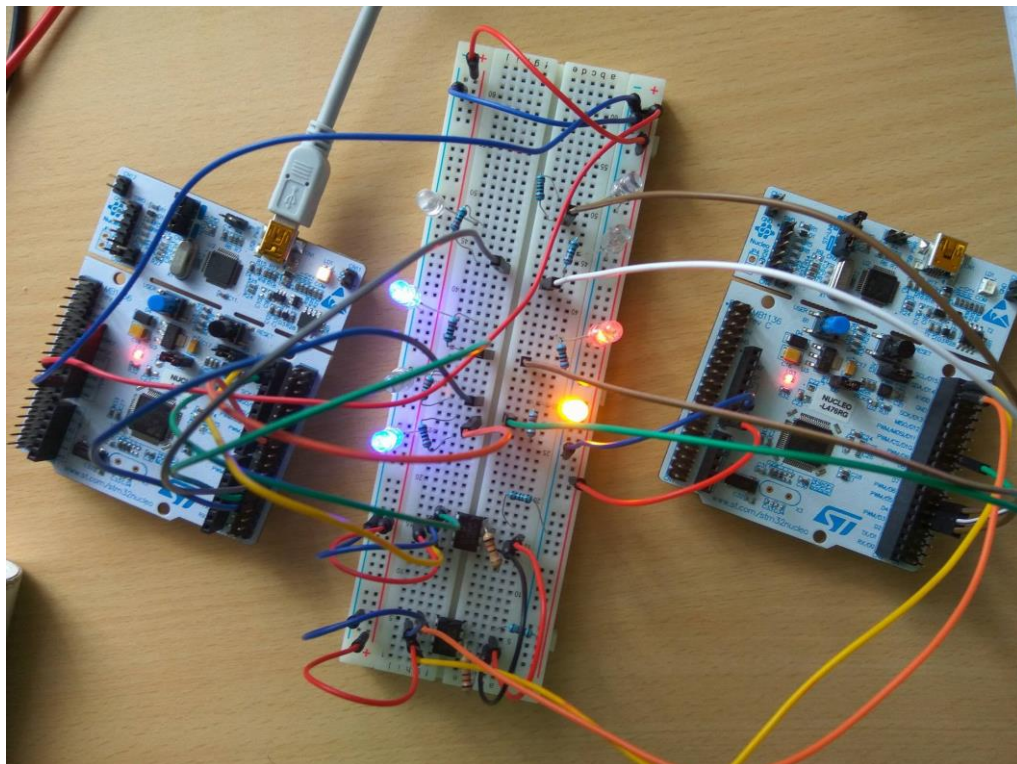


Figura 5.10

5.2.2.3 Programarea și conexiunile afișajului

Afișajul folosit este un LCD a1602, ce are 16 caractere pozitionate pe doua linii.

Pentru inceput trebuie sa activam pinii necesari configurarii si programarii LCD-ului. Acest lucru il facem intrand in documentul microcontrolerului slave si pornim de acolo aplicatia CubeMX. Odata deschisa fereastra vom activa urmatorii pini GPIO setati ca pini de iesire:

- Pinii PB6,PB5,PB4,PB3 ii vom alege ca pini de date, ca sa ne fie mai usor vom schimba denumirea implicita a pinilor. Acest lucru il putem face dand click dreapta pe un pin activat si alegem optiune "Enter User Label", o fereastra mica va aparea cu un chenar unde putem introduce noua denumire a pinului. Vom redenumi dupa cum urmeaza: PB6=D7, PB5=D6, PB4=D5 si PB3=D4.

- Pinii PC2 si PC3 ii vom folosi ca pini de reset si enable necesari display-ului. Denumirile lor vor fi de asemenea modificate: PC2=RS si PC3=E.

Dupa aceste setari ne mutam la tabul de Clock Configuration, aici ne asiguram ca HIS din PLL Source Mux este setat la fel si PLLCLK de la System Clock Mux. Mai departe AHB Prescaler de 4 si HCLK=16 MHz.

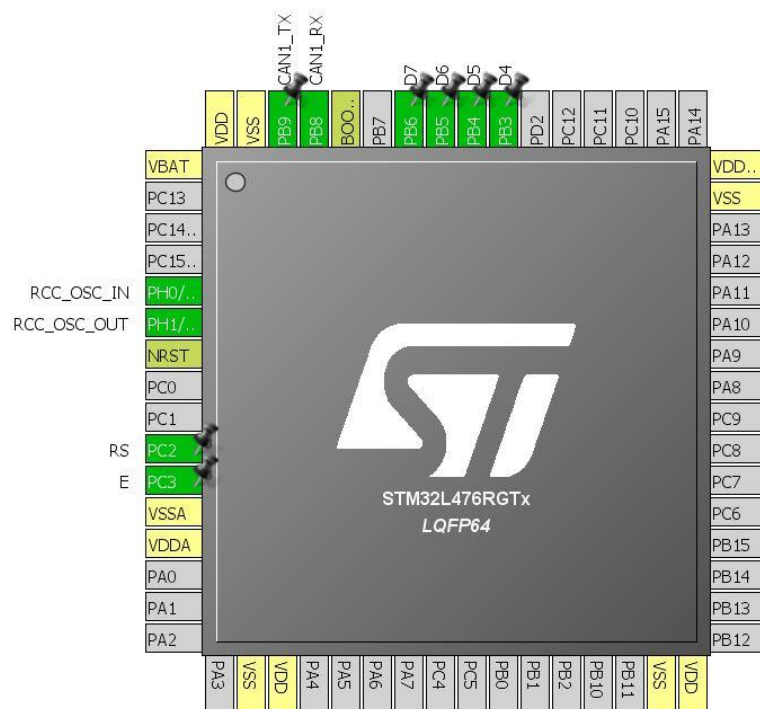


Figura 5.11

Mutandu-ne apoi in IAR (actiune realizata odata cu generarea codului din CubeMX), va fi nevoie sa importam librariile necesare LCD-ului. Aceste librarii se pot importa astfel:

- Copiem librariile respective in documentul de pe laptop unde se afla fisierele microcontrolerului slave. Apoi din IAR deschidem Application->User, dam click

dreapta pe User, selectam Add si apoi Add files. De aici cautam fisierul STM_MY_LCD16x2.c si STM_MY_LCD16x2.h pe care le selectam si le aducem in IAR.

- Dăm click dreapta pe fisierul mare din IAR(deasupra fisierului Application), ne ducem la Options, la Category selectam C/C++ Compiler si intram pe tabul Preprocessor si selectam butonul numit "...". Va apare o vereastra numita "Edit Include Directories", în aceasta fereastră mergem pana in josul ei si dam click pe "<Click to add>" de aici ne alegem calea catre directorul mare al proiectului nostru, dam click OK, apoi OK in Options.

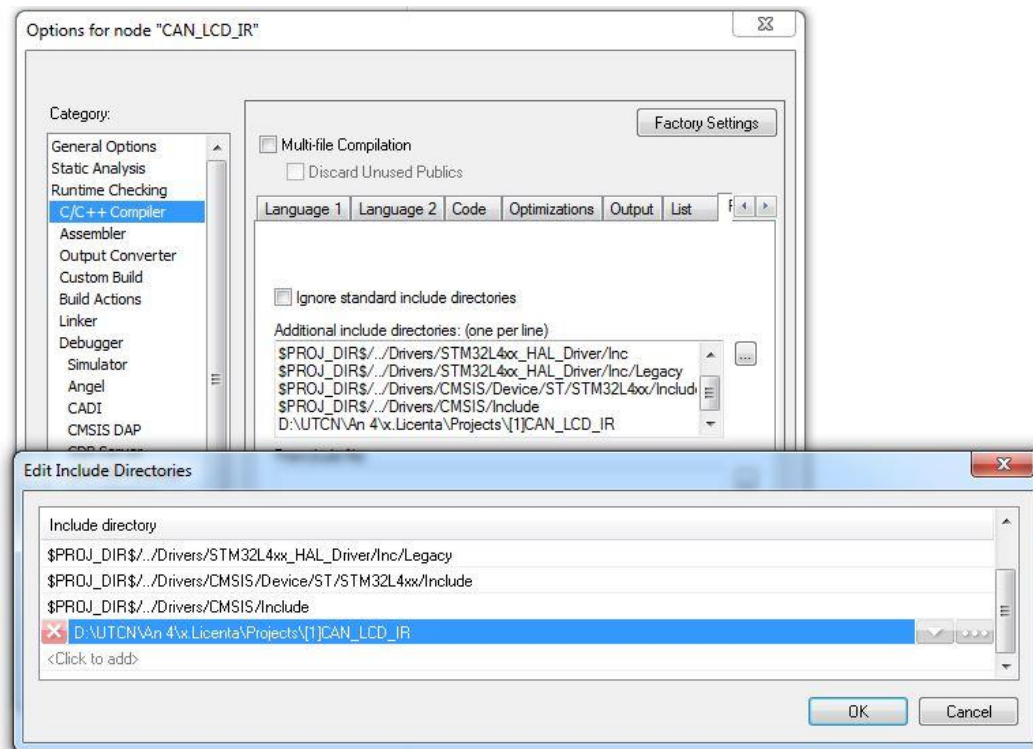


Figura 5.12

Dupa terminarea acestor operatiuni va trebui sa dam referinta in main.c a headerului librării LCD. Asta se face introducand linia de cod `#include "STM_MY_LCD16X2.h"` in zona USER CODE BEGIN/END Includes. Mai trebuie sa facem o modificare în libraria importata deoarece ea a fost creata pentru o alta familie de microcontrolere de la STM32. Asadar intram in IAR si din User accesam fisierul cu extensia .c a librării unde inlocuim include-ul referitor la placa folosita cu `#include "stm32l4xx_hal.h"`, la fel procedam si pentru fisierul header al librării.

Urmeaza definirea catorva macrouri in main.c si anume pini folositi. Folosind simbolul `"#"` urmat de `"define"` vom putea atribui valori unor variabile dupa cum urmeaza: RS=2, E=3, D4=3, D5=4, D6=5 si D7=6.

În zona de program USER CODE END 2 si Infinite loop vom folosi o functie din libraria nou introdusa numita LCD_begin4BIT(se vor folosi doar 4 biti de date). Aceasta librerie va indica porturile folosite pentru comunicarea cu LCD-ul. Parametrii ceruti de aceasta functie(în total opt) sunt dupa cum urmeaza: structura de date a grupului de pini

folositi în urmatoarele doua pozitii (GPIOC), pinul de restart si cel de , structura de date a grupului de pini folositi în urmatoarele patru pozitii, pinii D4,D5,D6 si D7.

Datorita necesitatii unui delay al afisajului, codul scris pentru LCD se va produce in bucla while(1) a fisierului main.c. Codul este scris intre USER CODE BEGIN/END.

Primul lucru va fi selectia cursorului prin functia LCD_setCursor avand ca parametrii linia si coloana de la care se incepe scrierea. Parametrii folositi in aceasta functie vor fi "1" (prima linie) si "2" (coloana a doua) deoarece vom afisa un text ce va trebui incadrat astfel.

Apeland functia LCD_print vom putea scrie un text tat printr-o variabila de tip string. Textul scris intre ghilimele "Treapta viteza" are 14 caractere si pentru o afisare ingrijita vom scrie de pe coloana a doua pana pe a 15-a, lasand prima si ultima coloana libere.

Trecem la urmatorul rand unde vom scrie un singur caracter si anume o cifra de la 0 la 5 reprezentand treapta de viteza in care ne aflam din functie de ceilalti doi parametrii (viteza si turatie). Setam cursorul pe linia 2 si coloana 9.

Mai departe avem nevoie de o functie ce face conversia intr-un parametru de tip string dintr-un parametru de timp integer. Asta se poate realiza cu functia din librarie numita LCD_itoa. Functia are nevoie de doi parametrii, variabila integer si un vector de tip char. Declaram in zona de declaratii o variabila de timp char pentru a ne folosi de ea in vederea conversiei executata de aceasta functie. Acesta conversie este necesara deoarece functia de scriere din librarie accepta doar parametru de tip string. Dupa conversie apelam functia de scriere cu parametrul dat de variabila char.

În continuare apelam o functie ce ne va sterge cursorul de pe display, functie numita LCD_noCursor. Ea nu are niciun parametru. Va fi necesara apelarea functiei HAL de întârziere deoarece textul afisajului se schimba, iar lipsa unei intarzieri nu ar facilita vizualizarea altor lucruri inafara de primul cod rulat in bucla(am setat o întârziere de 100 ms).

Ultima functie apelata este LCD_clear, aceasta functie va sterge ecranul urmand sa se reia codul pentru a se scrie altceva in cazul in care acest lucru este posibil. Functia clear nu are parametri. Dupa terminare salvam si iesim din program. În urma acestei programari display-ul va arata astfel. Mai urmeaza sa facem legaturile fizice, iar apoi putem rula noul program.



Figura 5.13

Conexiunile fizice necesare pentru a pune în funcțiune display-ul sunt cele mai numeroase din acest proiect. Nu mai puțin de 12(+2 de prelungire) fire sunt folosite pentru implementarea lui în proiect.

De la Vss se va duce o legatura la GND microcontrolerului si de la Vdd la 5V. Acesti doi pini vor alimenta afişajul în low power mode. Pentru o vizualizare mai puternică vom folosi pinii A(anod) si K(catod) legați la 5V respectiv GND. Pinul RS îl vom conecta la PC2, iar pinul de enable E la PC3. Vom folosi un potențimetru B1K cu pinul 1 la 5V I pinul 3 la GND ca pinul 2 să fie legat la V0 pe display pentru a putea jongla cu intensitatea luminoasă a textului. Pinul RW se va lega la GND pentru a fi activată opțiunea de scriere pe LCD. Pinii D4,D5,D6,D7 vor fi conectați la ieșiri active ale microcontrolerului după cum urmează: D4 la PB3, D5 la PB4, D6 la PB5 și D7 la PB6.

Această configurație poate fi observată și în imaginea de mai jos :

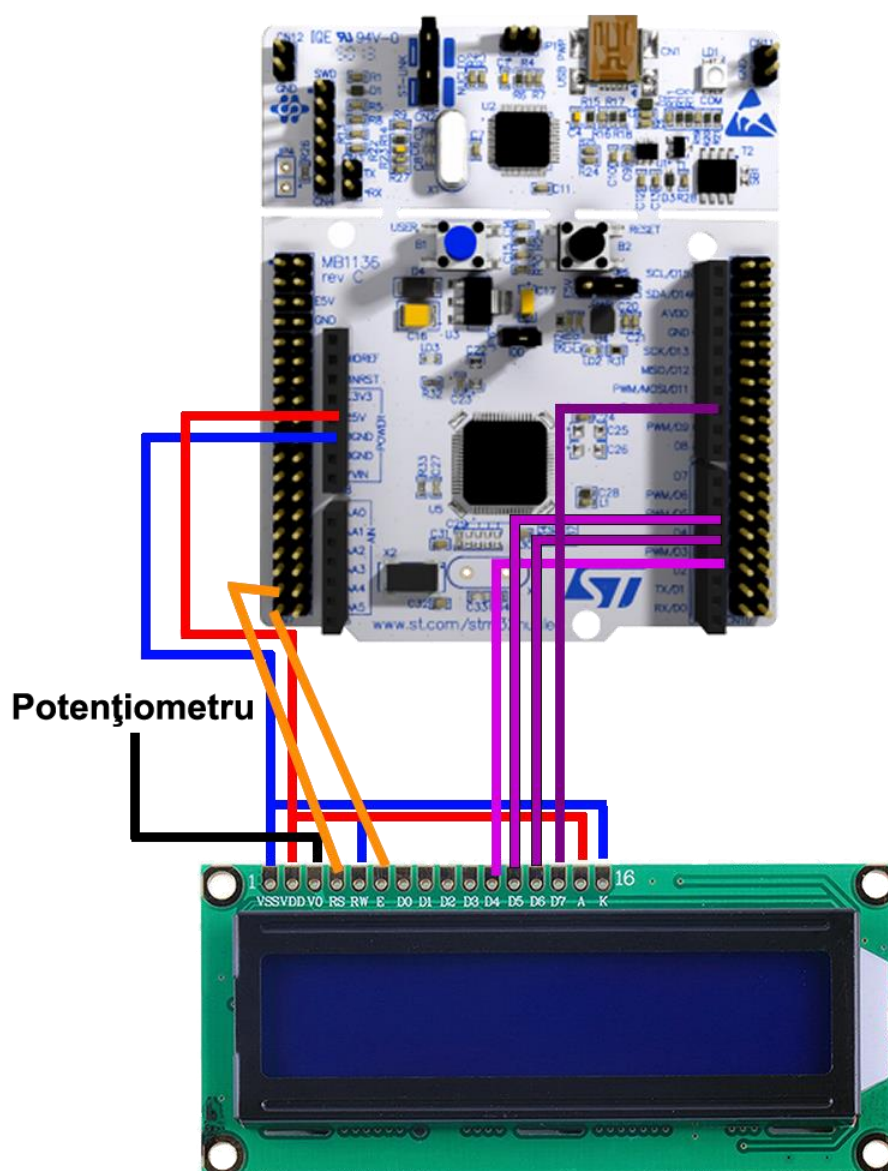


Figura 5.14

5.2.2.4 Inițializarea motoarelor

Proiectul necesita existenta a doua motoare stepper (motoare pas cu pas). Cele doua motoare vor afisa turatia motorului (in rotatii pe minut) respectiv viteza masinii (in kilometri pe ora). În varful acului rotativ vom pune un indicator de plastic. În figurile de mai jos sunt afisate cele doua componente utilizate:



Figura 5.15



Figura 5.16

Verificarea fazelor acestui motor se poate realiza utilizand aparatul de masura. Masurarea curentului ce trece prin cele doua bobine ne va indica cum sunt legate pentru a putea stii pe viitor cum sa e polarizam. Asadar cuplam sondele aparatului de masura, cea neagra la GND partea de jos a aparatului si cea rosie deasupra celei negre la desenul ce semnifica partea pozitiva. Apoi invartim de rotita aparatului pana ajungem la zona unde se masoara rezistenta simbolizata prin Ω . Aici selectam scala de 200.

Dupa ce am setat aparatul de masura acesta este gata pentru a verifica contactele motorului. Avem nevoie de o bucata de hartie si un pix pentru a face un tabel si a nota ce pini sunt conectati intre ei. Începem prin a pune sonda de neagra pe primul pin si apoi sonda rosie o mutam pe rand la fiecare din ceilalti 3 pini. Constatam ca doar la primii doi pini avem o valoare de 148.9Ω . Ne mutam cu sonda neagra la urmatorul pin si verificam ceilalti pini cu sonda rosie sa vedem unde mai obtinem asemenea valoare. Dupa toate incercarile posibile constatam ca pinii sunt grupati doi cate doi.

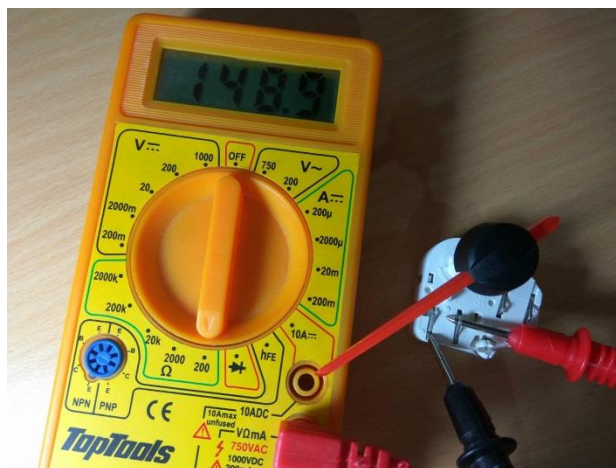


Figura 5.17

Acest tip de motor este folosit pentru a indica diverse marimi în mai multe domenii și aparaturi. Cel mai răspândit fiind cel auto unde se folosește în bordul din spatele volanului.

Interiorul motorului este compus din două bobine o melc prins la un capăt de un magnet aflat în zona dintre bobine de unde polarizarile survenite vor determina învârtirea acestui melc. Melcul aflat în conexiune cu o roată melcata o va învârti și pe aceasta. În mijlocul rotii melcate este înglobată o tijă lungă care iese destul de mult în încapsularea motorului. Aceasta tijă este capul pe care vom pune indicatorul de plastic. Melcul, odată pus în mișcare, se va tot roti învârtind cealaltă roată până când opritorul de pe ea se va lovi într-unul din capetele de cursă existente. Ajuns acolo mișcarea va tot trepida din cauza incapacității de a merge mai departe. Se poate observa interiorul motorului în imaginea de mai jos:

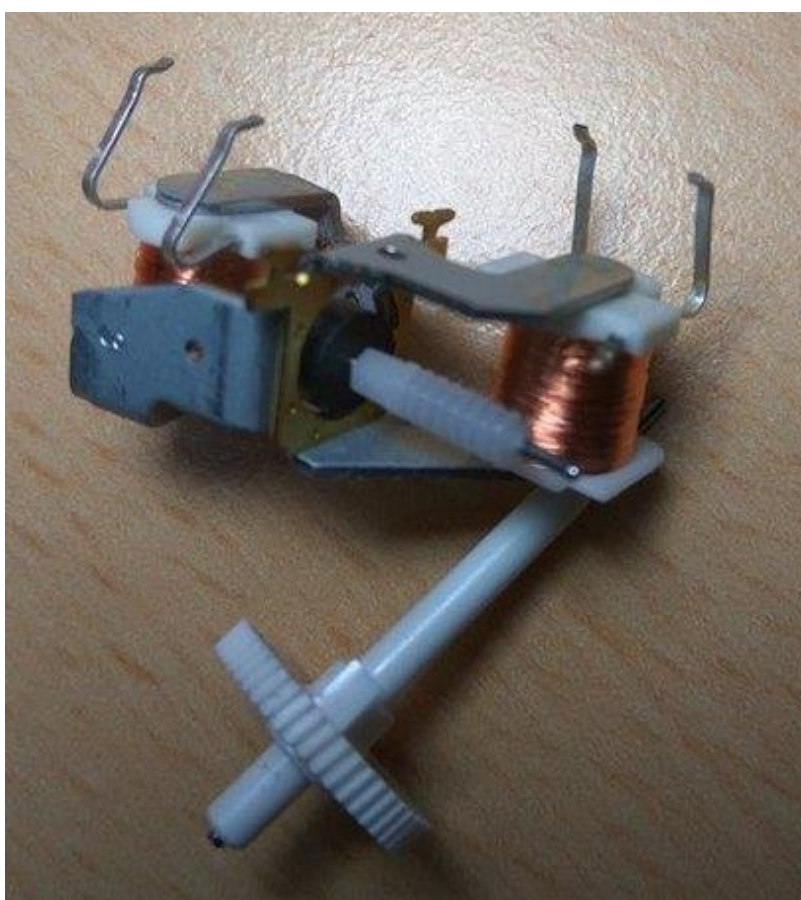


Figura 5.18

Stiute aceste aspect mai ramane doar determinarea numarului de pasi necesari pentru o rotatie completa. Rotorul motorului nu face o rotatie completa, el face un arc de cerc de la o pozitie de start pana la o pozitie de stop de 315° . Numarul de pasi efectuati pe acest arc de cerc l-am determinat experimental. Un total de 200 de pasi executati de la start inspre stop. Aceasta determinarea experimentală a fost făcută conectând motorul la microcontroler și dându-i semnale repetate într-o anumită ordine astfel încât rotorul să se învârtă. Numărând dinții de pe roata melcata aflăm că sunt un total de 50 de dinți.

5.2.2.5 Comandarea motoarelor

Pentru a putea comanda motoarele pas cu pas de care am vorbit în subcapitolul anterior va trebui să folosim drivere ce sunt capabile să le comande. De ce folosim drivere? Deoarece este mult mai greu să scriem un cod care să aibă în vedere toate aspectele unei astfel de operațiuni, numărul de legături fizice ar fi foarte mare, o multitudine de fire ar trebui incluse în proiect și mulți pini de pe controler ar fi ocupați pentru aceste chestiuni.

Prin urmare vom folosi două drivere pentru comanda motoarelor pas cu pas, drivere folosite adesea în imprimantele 3D, dar și în alte aplicații ce necesită un control fin al mișcărilor executate. Driverul de pe care îl vom folosi este A4988ET. Schema de principiu arată ca în imagine:

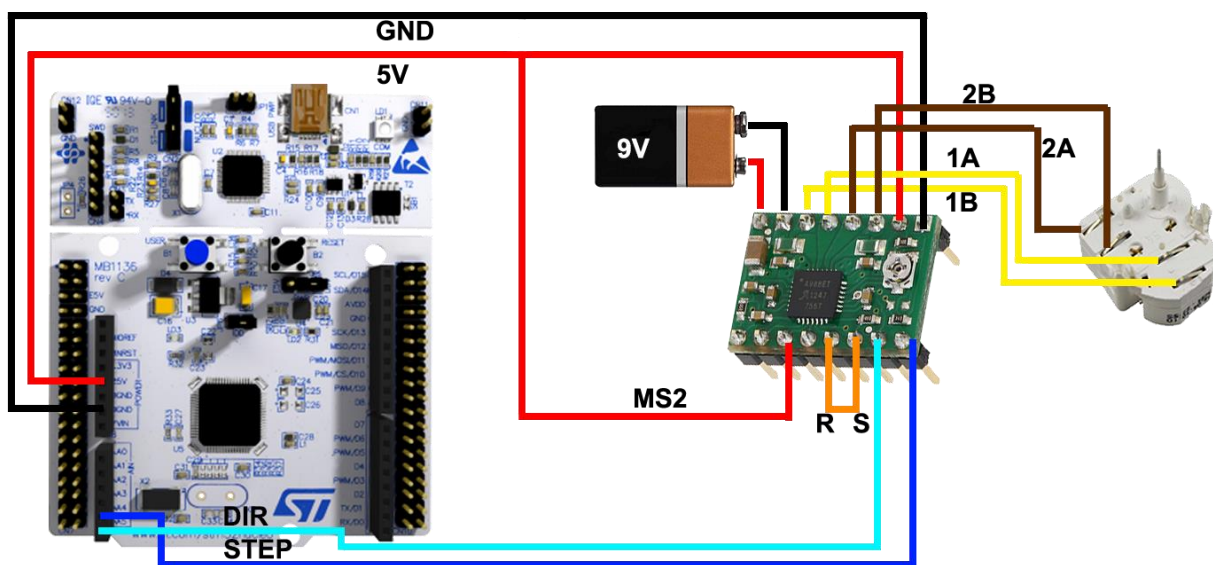


Figura 5.19

Să începem cu caracteristicile hardware ale implementării în proiect ale acestor drivere și ale motoarelor. Introducem driverele în breadboard în locul din mijloc unde a mai rămas spațiu, între cele două traneivere. Le vom poziționa cu potentiometrul spre dreapta și cu pinii de o parte și de cealaltă a canalului longitudinal de pe breadboard pentru a avea acces la toți pinii. După introducerea driverelor A4988 vom conecta pinii stepperelor la pinii driverului. Urmărind schema de mai sus vom proceda astfel: o pereche de pini de pe stepper o vom conecta la pinii 1A și 1B și cealaltă pereche la 2A și 2B. Procedăm astfel cu ambele stepper și ambele drivere.

După ce am terminat de legat stepperul de driver vom lega pinul DIRECTION de pe driver la pinul A2 al microcontrolerului și pinul STEP la pinul A3. Pentru driverul al doilea vom conecta pinul de direcție la pinul A4 și pinul de STEP la pinul A5. Vom conecta între ei pinii Reset și Sleep pentru a putea activa driverul (atât pentru driverul 1 cât și pentru driverul 2). Conectăm la ambele drivere pinul MS2 la 5V pentru a activa modul de microstepping de $\frac{1}{4}$.

În ceea ce privește tensiunea de alimentare a driverelor, acestea se vor lega în următorul mod: pinul Vss al driverului la GND și pinul Vdd la 5V de pe microcontroler. Alimentarea de putere se va face de la o sursă externă de tensiune de exemplu o baterie de 9V ce se va conecta cu “+” la Vmot și cu “-” la Vss-motor (partea de jos a driverului față de potentiometru). Vom decupla totuși sursa de tensiune externă de la driver pentru a putea face și alte modificări, iar acea tensiune să nu se influențeze acum.

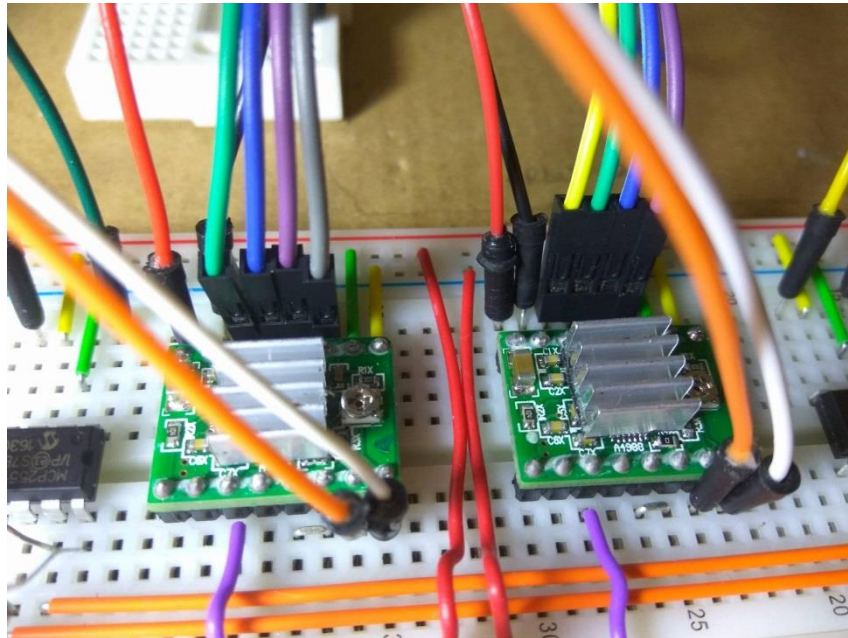


Figura 5.20

Potentiometrul respectiv ajută la determinarea tensiunii de funcționare respectiv la determinarea curentului. Am setat, măsurând cu aparatul de măsură de la pinul Vss alimentat de placă și potentiometru o tensiune egală cu 0.5 V pentru amândouă driverele. Aplicând formula (2.1) și știind că $R_S=0.1$ obținem o intensitate a curentului de 0.625 A.

Acuma trebuie să scriem codul pentru comanda driverelor și motoarelor. Întrăm în directorul proiectului, accesăm fișierul CubeMX de unde ne pornim timerele TIM8 și TIM3. Generăm codul C și intrăm în IAR Embedded Workbench. Aici în fișierul main.c adăugăm în funcția HAL_TIM_PeriodElapsedCallback încă două instanțe de întrerupere bazate pe timere.

Asemănător cu instanța de întrerupere folosită pentru citirea și trimiterea datelor de pe și pe magistrala CAN, tot așa vom proceda și cu comanda driverului.

Configurăm timerul pentru primul driver având prescaler de 1600 și perioadă de 49, iar pentru al doilea driver prescaler=1600 și perioadă=79.

Vom crea trei funcții pentru primul driver și încă trei pentru cel de-al doilea.

Prima funcție se va numi “step”, va face un singur pas și va da direcția de deplasare. Cele două argumente ale funcției prima de tip boolean și a doua de tip integer, compun funcția în felul următor: se pune condiția ca variabila integer să fie mai mare decât 2 (pentru a se face pași doar dacă este o diferență mai mare decât 2 între valoarea actuală și valoarea următoare rpm-ului). În această condiție de face “SET” și apoi “RESET” pe

pinul dedicat pasului ce va fi trimis spre driverul rpm-ului. Apoi vom avea o alta condiție ce va utiliza variabila booleana. Daca aceasta va fi "true" atunci r2 va creste cu o unitate, daca nu, r2 va descrește.

A doua funcție este numita "step_speed" cu un parametru integer. Ea va determina viteza de deplasare a acului indicator în functie de diferentele de valori aflate între o valoare a rpm-ului actuala și una precedentă. Astfel dacă valoarea va fi mai mică sau egală cu 200 se va apela funcția step, dacă va fi între 200 și 550 se va apela de două ori funcția, iar în ultimul caz se va apela de trei ori.

Ultima funcție, funcția "direction", va avea doi parametri întregi. Aceasta funcție face posibilă activarea pinului de direcție al driverului. Valorile venite ca parametri sunt valoarea actuală și valoarea precedentă. Dacă actual-precedent este mai mare decât 0 atunci se va scrie (HAL_GPIO_WritePin) SET pe pinul de direcție și acul se va învârti spre dreapta, dacă va fi mai mic decât 0 atunci se va scrie RESET și acul se va roti spre stanga.

Celălalt set de trei funcții este dedicat driverului și motorului numărul 2, reprezentând viteza. Singurele modificări în aceste funcții sunt legate de pinii pe care dorim să-i activăm. Dar mai intervine încă o funcție ce determină treapta de viteza ce va fi afișată pe ecran.

Această funcție numită "treapta" are un argument întreg. Ea conține 5 condiții. O condiție pentru fiecare treapta de viteza (1->5). Acest argument este dat la apelul funcției de variabila s2. Variabila s2 este variabila ce ține poziția actuală a acului indicator pentru motorul de viteza.

În ceea ce privește apelarea funcțiilor ele vor fi prin intreruperile timerelor amintite mai sus. Funcțiile le apelăm astfel:

Dacă (htim->Instance==TIM2) atunci

Scalăm r1 în intervalul 0->750

Aflăm diferența dintre r1 și r2 (int ref)

direction(r1,r2)

step_speed(ref)

Dacă (htim->Instance==TIM3) atunci

Scalăm s1 în intervalul 0->750

Aflăm diferența dintre s1 și s2 (int refe)

direction_2(s1,s2)

step_speed_2(refe)

treapta(s2)

Se salvează programul, apoi se conectează placa slave la laptop, după aprinderea becurilor se încarcă programul pe microcontroler.

Să nu uităm de alimentarea motoarelor, acestea trebuie să primească 8-9 V la pinul Vmot al driverelor. După ce am alimentat motoarele și al legat microcontrolerul slave la

laptop putem intra ori in debugger ori apăsăm butonul reset de pe placă. Actionând potentiometru putem vedea cum motorul stepper al turațiilor se mișcă mult mai repede decât cel al vitezei, iar pe display se afișează treapta de viteză în funcție de viteza de la motorul stepper ce afișează viteza de deplasare.

5.2.2.6 Înglobarea într-un sistem compact

Pentru a arata mult mai bine din punct de vedere estetic se vor introduce toate componentele într-o cutie de carton de duritate crescută. Aici se vor aranja în așa fel încât să încapă toate elementele ce compun acest proiect. Zona din față o vom decupa pentru a avea acces la componente. În capacul superior se vor decupa și găuri locuri specifice pentru a poziționa LCD-ul și motoarele. În partea de jos se va găuri și se vor introduce suruburi cu piulite prin microcontrolere pentru a le poziționa fix. Breadboardurile le vom lipi tot de partea inferioară a cutiei, în interior. Cablurile le vom așeza în ordine și le vom prinde cu bandă izolatoare atât între ele cât și de peretii cutiei.

În ultima instanță vom găuri spatele cutiei pentru a face loc ieșirilor pentru cablurile de alimentare. Vom înveli cutia într-un ambalaj neutru și vom marca zonele de indicator ale motoarelor.

În figura de mai jos se poate observa proiectul final:

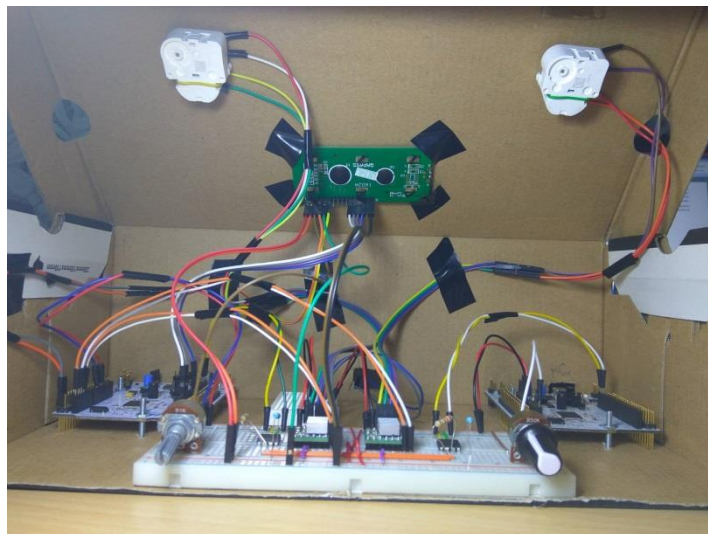


Figura 5.21

6 Concluzii

6.1 Rezultate obținute

S-a urmărit simularea unei părți din bordul unui autovehicul și anume a indicatoarelor de turație și de viteză a mașinii precum și afisarea treptei de viteză în care ne aflăm în funcție de ceilalți doi parametri.

Implementarea s-a realizat utilizând doua microcontrolere STM32L476RG. Aceste controlere jucând un rol esențial în coordonarea celorlalte componente ce alcătuiesc întregul proiect.

Datele preluate de la potențiometru au fost prelucrate și au oferit doua alte marimi de utilizat: turația și viteza. Prelucrarea a constat în rularea unui model matematic ce a simulat aceste doua informatii pentru un autovehicul aflat în miscare. În funcție de viteza am stabilit și treapta de viteză în care se afla simulatorul bordului auto.

Coordonarea motoarelor s-a realizat controland cu microcontrolerele doua drivere pentru motoare pas cu pas. Transmisia de date între controlere realizându-se, asemeni autovehiculelor, prin protocolul CAN.

Alegerea acestor doua controlere a fost datorita prețului scazut și al posibilității utilizării protocolului CAN. Nevând doua canale CAN pe un singur controler, a fost nevoie de doua microcontrolere. Acest lucru a facilitat înțelegerea și vizualizarea de ansamblu a proiectului. Dacă aveam un singur microcontroler în care puteam face aceasta comunicare între doua noduri, obțineam doar rezultate, doar valori, și nu puteam să vedem ce se întâmplă dacă una din conexiuni este eronat pusă. Astfel punerea în funcțiune a sistemului a necesitat și competențe hardware nu doar software.

Alt motiv al alegerii ar fi posibilitatea utilizării programului CubeMX și al programului IAR Embedded Workbench. Cu primul, configurarea este foarte accesibilă și facilă, putând fi modificate oricând și reimplementate după bunul plac. Ce-a dea doua aplicație a oferit un control foarte bun al codului pe tot parcursul proiectului. Oferta de librării bine structurată și la îndemână, opțiuni de căutare și patrundere adâncă în ierarhizarea și modularea funcțiilor și variabilelor, plus comentarii la obiect asupra instrucțiunilor ce puteau fi folosite.

Zona de debugging a fost din nou foarte avantajoasă. La început utilizam terminalul Tera Term și activam UART pentru a putea vizualiza informațiile de pe un pin sau altul. Dar cu debuggerul acestei plăci nu mai este nevoie de utilizarea acelui terminal în toate cazurile. Pentru a vedea dacă niste date sunt transmise sau nu este suficient să intrăm în debugger și din live watch să vizualizăm ce dorim.

6.2 Direcții de dezvoltare

Pe viitor se poate implementa un modul bluetooth prin care am putea controla simulatorul prin telefon. Acest lucru necesita crearea unei aplicatii Android sau IOS, depinde de telefonul mobil folosit, care sa poata transmite date unui modul bluetooth ce mai apoi va oferi informatiile microcontrolerului master.

Algoritmul de determinare al vitezei si rotației se poate imbunatatii, introducandu-se noi variabile ce va simula si mai bine experienta afișajului.

Controlul motoarelor stepper se poate face utilizand un mod mai fin de a face pașii(în proiect folosim $\frac{1}{4}$, dar am putea implementa $\frac{1}{16}$ pași).

Am putea introduce noi mecanisme pentru afișarea uleiului motorului, combustibilul existent in rezervor, temperatura motorului, temperatura din mediul exterior sau temperatura din mediul interior. Acest lucru se poate face fie cu indicatoare analogice(ace indicatoare rotite de un motor) fie cu indicatoare digitale (reprezentate printr-un display).

O idee de dezvoltare mult mai importanta ar fi eliminarea celor doua controlere si inlocuirea lor cu integrate specializate in transmisia CAN numite si controlere stand-alone CAN. Un exemplu de aceste cipuri ar putea fi MCP2515. Utilizând MCP2515 cu traneivererele MCP2551 vom avea nevoie doar de un microcontroler care nu va avea nevoie neaparată de un protocol CAN înglobat (doar daca se dorește a se utiliza si acesta). Acest microcontroler master va coordona munca celorlalte integrare externe. Astfel "munca" din proiect este divizată si devine mult mai eficientă.

În ceea ce privește alimentarea sistemului ar fi de avut în vedere înlocuirea bateriei cu o sursă ce nu își pierde energia in timp.

7 Bibliografie

- [1] *UM1724 User manual, STM32 Nucleo-64 board, 2016*
http://www.st.com/content/ccc/resource/technical/document/user_manual/98/2e/fa/4b/e0/82/43/b7/DM00105823.pdf/files/DM00105823.pdf/jcr:content/translations/en.DM00105823.pdf
- [2] *RM0351 Reference manual, STM32L4x5 and STM32L4x6 advanced ARM-based 32-bit MCUs, 2017*
http://www.st.com/content/ccc/resource/technical/document/reference_manual/02/35/09/0c/4f/f7/40/03/DM00083560.pdf/files/DM00083560.pdf/jcr:content/translations/en.DM00083560.pdf

- [3] *CAN Primer: Creating Your Own Network, CAN: Controller Area Network Lab using ST STM32 Cortex-M processors, V 2.0 Robert Boys*
http://www.keil.com/appnotes/files/apnt_236.pdf
- [4] *Automotive CAN Bus System, Kiril Mucevski, 8 decembrie 2015*
<https://www.linkedin.com/pulse/automotive-can-bus-system-explained-kiril-mucevski>
- [5] *High-Speed CAN Transceiver MCP2551 Microchip.*
<http://ww1.microchip.com/downloads/en/DeviceDoc/21667f.pdf>
- [6] *Where Does The "336" Come From In the Speed/RPM/Gear-Ratio/Tire-Size Formula, de Marlan Davis, în 6 februarie 2015*
<http://www.hotrod.com/articles/speed-rpm-gear-ratio-tire-size-formula/>
- [7] *What is a Stepper Motor?, de Bill Earl, 23 noiembrie 2015*
<https://learn.adafruit.com/all-about-stepper-motors/what-is-a-stepper-motor>
- [8] *A4988 Stepper Motor Driver Carrier*
<https://www.pololu.com/product/1182>
- [9] *Spcification for LCD Module 1602A-1*
<https://www.openhacks.com/uploadsproductos/eone-1602a1.pdf>
- [10] *Controlling STM32 Hardware Timers using HAL, 31 martie 2016*
<https://visualgdb.com/tutorials/arm/stm32/timers/hal/>