



Facultatea de Automatică și Calculatoare  
Programul de Licență: Calculatoare



---

# OPTIMIZAREA TIMPULUI DE REAȚIE ÎN SISTEME TCU MULTICORE

Proiect de diplomă

*Candidat:*  
**George-Lucian BUZURIU**

*Coordonator științific:*  
**Conf. Dr. Ing. Lucian-Adrian Prodan**

Timișoara  
2018

## Cuprins

1.Introducere .....	3
1.1 Importanța domeniului Automotive.....	3
1.2 Unități de control într-o mașină .....	4
1.3 TCU (Transmission Control Unit).....	5
1.4 Prezentarea sistemului de transmisie a puterii.....	6
1.5 Moduri de comunicație ale TCU-ului .....	10
1.6 Generația_1 a proiectului DL382 .....	12
1.7 Generația_2 a proiectului DL382 .....	14
1.8 Comunicarea microcontrolerului cu chipul EEPROM.....	16
1.9 Situație actuală.....	17
1.10 Temă propriu-zisă .....	17
2.Proiectare și implementare.....	18
3.Testare .....	24
3.1 Structura sistemului .....	24
3.2 Rezultate experimentale .....	24
3.3 Unelte folosite în testare.....	35
4.Concluzii.....	40
5.Bibliografie .....	42

# 1.Introducere

## 1.1 Importanța domeniului Automotive

Într-o lume aflată într-un proces de evoluție continuu, cu toții suntem conștienți că principalul mijloc de deplasare este automobilul. Fiind un element esențial al vieții noastre cotidiene, automobilul, constituie un motiv bine întemeiat pentru rapiditatea cu care avansează industria automotive.

Totodată, necesitatea dezvoltării unor noi tehnologii în acest vast domeniu este foarte ridicată. Atât dezvoltarea noilor soluții, cât și testarea lor, reprezintă elemente demne de o atenție sporită, deoarece chiar și o greșală minoră poate fi fatală.

Industria automotive este compusă într-o proporție covârșitoare din mașini. Mașina este un întreg sistem de sine stătător. Utilizarea atât de vastă și de diferită a automobilelor a condus la desprinderea și formarea mai multor tipuri de mașini. Stilul fiecărei mașini a fost influențat atât de marii producători, dar și de proprietari. Funcțiile unei mașini determină parametrii de proiectare și tipurile de constrângeri ale acesteia, privit atât ca întreg, cât și pe componente.

Deși există foarte multe tipuri de mașini, ce au fost create pentru funcționalități diferite, în construcția lor sunt folosite sisteme similare.

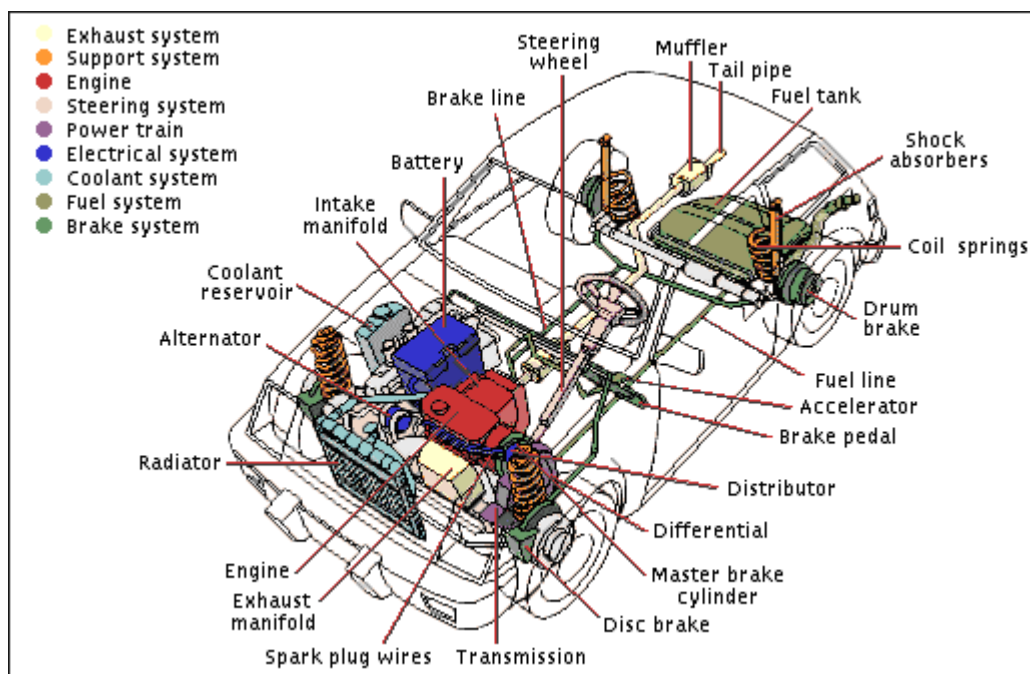
Cu toate că un automobil dispune de mai multe componente electrice, motorul reprezintă “inima” mașinii. Fără acest element esențial, automobilul nu ar fi putut să existe. Cu privire la automobile, motorul este cel care primește sarcina de a genera energie mecanică, transformând energia chimică din benzină.

Motoarele oferă diferite niveluri de putere și cuplu la viteze diferite. Transmisia permite vehicului să funcționeze într-un interval de viteză mai mare, pastrând în același timp turația motorului în intervalul de funcționare. În cele mai multe automobile, puterea motorului depinde numai de poziția accelerației. Condițiile diferite solicită diferită putere, de exemplu este nevoie de o anumită putere pentru a învinge rezistența la rularea statică. În cazul pornirii autovehiculului, din cauza nevoii de o cantitate mare de cuplu, puterea motorului este recondiționată prin transmiterea puterii roților.

Transmisia utilizează diferite combinații de viteze, sau rapoarte, pentru a multiplica turația motorului și cuplul pentru a se potrivi condițiilor de conducere. O transmisie manuală permite schimbarea vitezelor de către conducătorul auto, pe de altă parte o transmisie automată utilizează un sistem intern hidraulic/electric pentru transmiterea energiei de la motor la roțile automobilului fără intervenția conducătorului.

Din multitudinea de funcționalități ale unui automobil, îmi îndrept atenția spre unitățile de control existente în mașină, care fac posibile buna funcționare a mașinii și într-o siguranță cât se poate de mare.

Astfel îmi propun să ilustrez componentele electronice ale unei mașini:



**Figura 1.1.1**

## 1.2 Unități de control într-o mașină

Odată cu evoluția tehnologiei, industria automobilelor a fost nevoită să țină pasul. Prin urmare, au apărut o multitudine de sisteme interne, unități de control ale automobilului care realizează funcționalități cerute de diverși clienți.

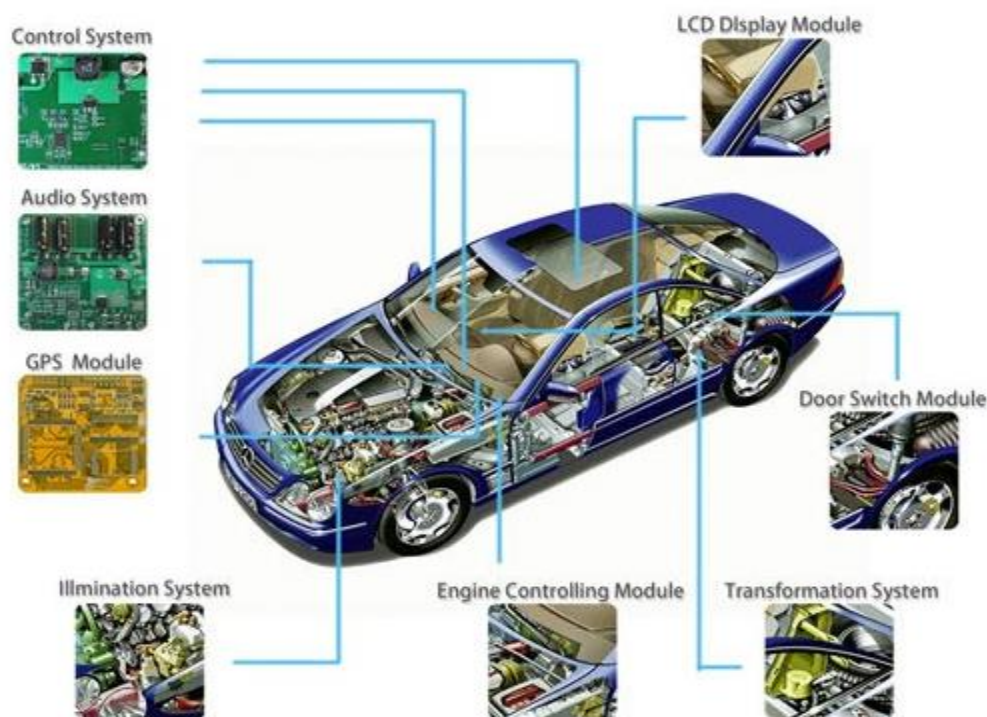
Un mic exemplu pe care l-aș da, îl reprezintă noua funcționalitate apărută în acest domeniu. Este vorba de pornirea automată a ștergătoarelor la apariția ploii. Sunt o mulțime de componente electrice și electronice care comunică între ele pentru a furniza acest serviciu.

Așadar, aici îmi propun să amintesc următoarele:

- **ECU**, unitate electronică de control
- **TCU**, unitate electronică de control a transmisiei
- **PDU**, unitate de control și distribuție a puterii
- **TCS**, sistem electronic de control al tracțiunii
- **DCU**, modulul de control electronic al sistemului de injecție
- **GPCU**, unitatea de control a bujiilor

Unitățile de control sunt asigurate de microcontrolere. Dacă înainte microcontrolerele erau singlecore, odată cu evoluția tehnologiei, după multe discuții de implementare și dezvoltare, au apărut și microcontrolere multicore. Tipul de microcontroler folosit în proiectul meu o să-l dezvălui în cele ce urmează.

Obiectul central privind unitățile de control este TCU-ul, unitate de control a transmisiei. În următoarele cuvinte o să trasez câteva dintre caracteristicile acestei unități de control, dar și comunicarea sa cu celelalte elemente dintr-un automobil.

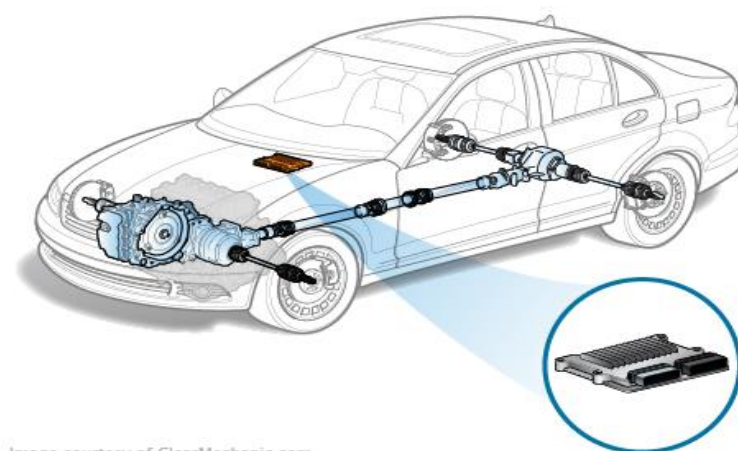


**Figura 1.2.1**

### 1.3 TCU (Transmission Control Unit)

TCU sau modulul de control al cutiilor cu transmisie automată este un dispozitiv care controlează transmisiile electronice moderne automate. Un TCU folosește, în general, senzorii de la vehicul, precum și datele furnizate de către unitatea de control al motorului pentru a calcula cum și când să schimbe vitezele în vehicul pentru performanțe optime, economie de combustibil și de calitate a schimbării treptelor.

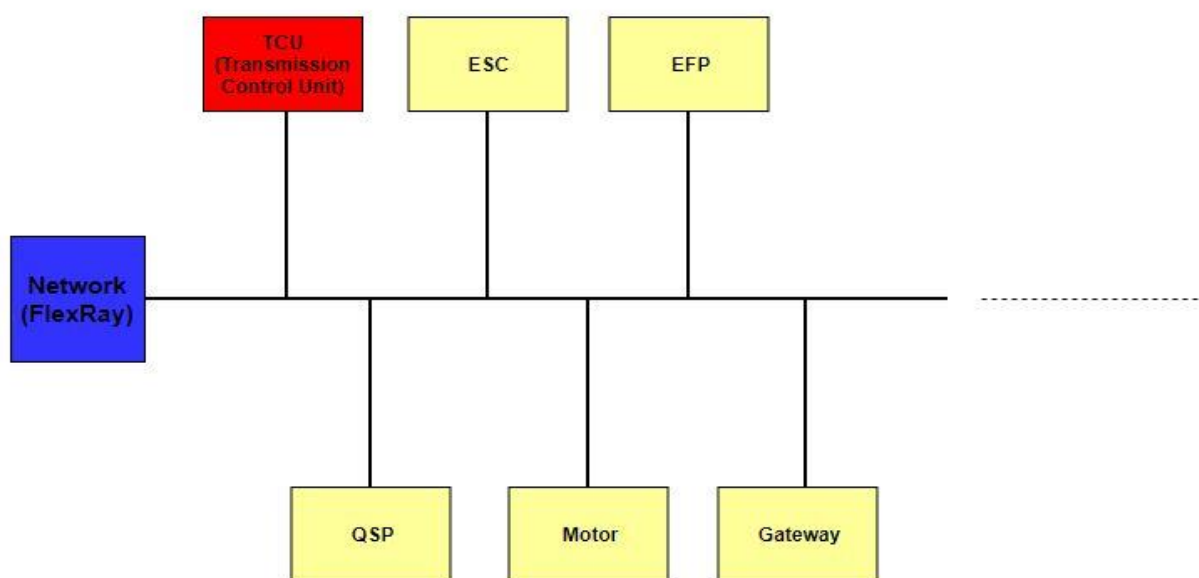
Transmisii automate electronice au fost trecerea de la schimbarea treptelor de viteze pur hidromecanic la cea electronică, de la sfârșitul anilor 1980. De atunci, a fost într-o continuă



**Figura 1.3.1**

dezvoltare, designul și modelele din ziua de azi există din mai multe etape de dezvoltare a unității de control al transmisiei automate. Solenoizi de transport sunt o componentă cheie pentru aceste unități de control.

Evoluția transmisiei automate moderne și integrarea controalelor electronice au permis progrese mari în ultimii ani. Transmisia automată modernă este acum capabilă de a realiza o mai bună economie de combustibil, reducerea emisiilor de motor, o mai mare fiabilitate a sistemului de schimbare a treptelor, viteza de deplasare crescută și creșterea manevrabilității autovehiculului. Gama imensă de programare oferite de un TCU permite transmisiei automate modernă să fie utilizată, cu caracteristici de transport adecvate, pentru fiecare situație. La unele aplicații, TCU și ECU sunt combinate într-o singură unitate, un modul de control Powertrain sau PCM. TCU se montează direct pe cutia de transmisii sau în habitacul motorului.



**Figura 1.3.2**

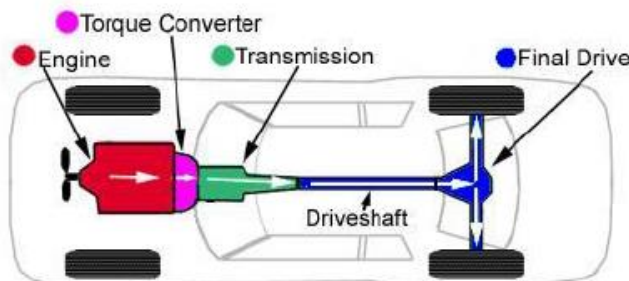
#### 1.4 Prezentarea sistemului de transmisie a puterii

- **Sistemul de tracțiune**

Sistemul de tracțiune reprezintă ansamblul componentelor ce fac posibilă aducerea și menținerea unui automobil în stare de mișcare.

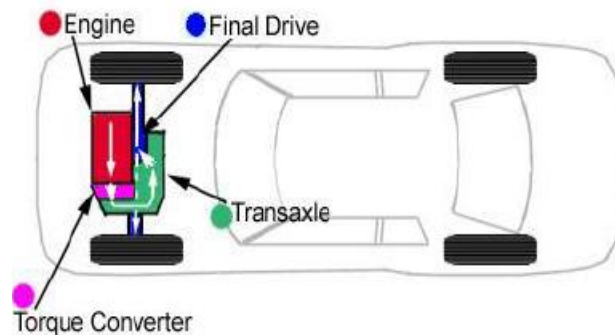
În funcție de tipul automobilului, desprindem mai multe categorii de tracțiune:

- Tracțiune spate, întâlnită cu precădere la mașinile sport (Lamborghini, Mercedes, BMW, Ferrari)



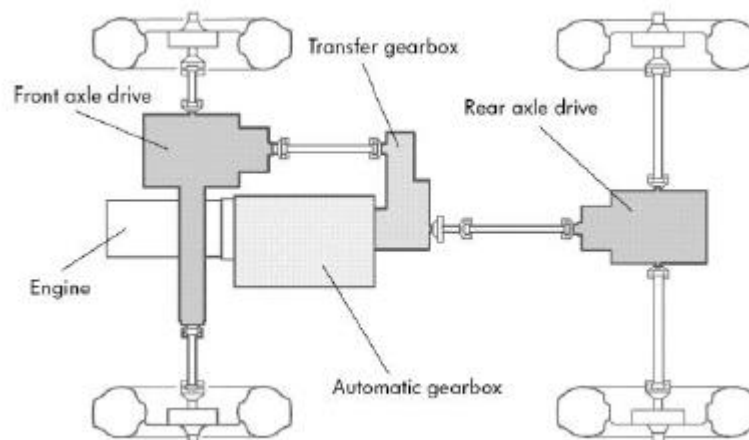
**Figura 1.4.1**

- Tracțiune față, întâlnită la majoritatea mașinilor mici (Dacia, Renault, Ford, Chevrolet, VW, Opel)



**Figura 1.4.2.**

- Tracțiune integrală, 4x4, întâlnită la SUV-uri (BMW X5, BMW X6, Audi Q7, VW Touareg)



**Figura 1.4.3.**

- **Elementele unui sistem de transmisie a puterii**

- **Ambreiajul**

Ambreiajul este un organ de mașină care prin cuplare transmite un moment de putere a unei mișcări de rotație, sau întrerupe această mișcare de rotație prin decuplare. Ambreiajele sunt folosite în mecanismele care au 2 axe, în general unul dintre ele este antrenat de un motor, iar celălalt antrenează cealaltă parte a mecanismului. Dezactivarea transmiterii cuplului de rotație, lasă cele două axe să se miște independent. Activarea ambreiajului se poate face atât în timp ce acesta se rotește sau în timp ce stă neînvârtit.





**Figura 1.4.4.**

▪ **Cutia de viteze**

Cutia de viteze este un ansamblu de roți dințate care servește la transformarea forței și transmiterea mișcării de rotație la roțile autovehiculului.

Raportul de transmisie este raportul dintre viteza de rotație a roților dințate ce compun cutia de viteze și viteza de rotație a roților automobilului.

Cutiile de viteze se împart în mai multe categorii, în funcție de:

1. Raportul de transmisie:
  - cu raport de transmisie fix
  - cu raport de transmisie variabil (CVT)
2. Gradul de automatizare:
  - manuală
  - CVT (Continuously Variable Transmission)
  - Automată

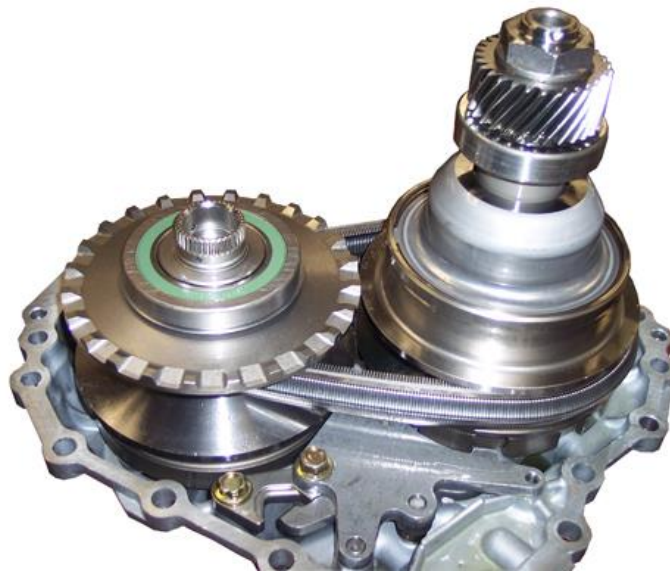
În cazul cutiei de viteze manuale, pentru fiecare viteză a cutiei este cuplată o roată dințată a acesteia pentru care există un raport de transmisie fix, ca în imaginea de mai jos:

<b>Viteză</b>	<b>Raport</b>
1	3.455
2	1.944
3	1.286
4	0.939
5	0.745
R	3.167

**Figura 1.4.5.**



**Cutiile de viteze CVT (Continuously Variable Transmission)**, spre deosebire de cutiile manuale, au capacitatea de a parcurge un număr infinit de rapoarte de transmisie. CVT oferă eficiență sporită în ceea ce privește consumul de carburant, deoarece îi permite motorului să funcționeze la cea mai eficientă turație.



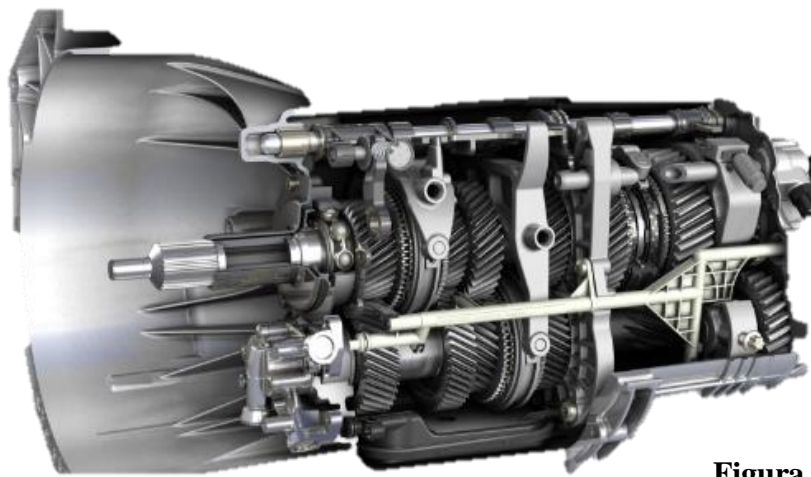
**Figura 1.4.6**

**Cutiile de viteze automate** sunt cutiile care realizează schimbarea treptelor de viteză fără intervenția conducătorului automobilului. Mai mult, decizia de schimbare a treptelor de viteză este luată de asemenea automat, pe baza informațiilor provenite de la diferiți senzori. Realizarea unei trepte de viteză se face prin intermediul mai multor mecanisme planetare.



**Figura 1.4.7.**

**Cutiile de viteze automate cu dublu ambreiaj**, din punct de vedere cinematic, sunt de fapt compuse din două cutii de viteze manuale, dispuse în paralel. Practic, în aceeași carcasă avem două cutii de viteze, fiecare cu propriul ambreiaj, o cutie conținând treptele impare (1, 3, etc.) iar a doua treptele pare (2, 4, etc.). Acest principiu permite ca vitezele să fie schimbate fără să se piardă din putere. În timp ce unul din ambreiaje transmite puterea, celălalt este pregătit pentru a cupla următoarea viteză, care este preselectată. Astfel, schimbul vitezei se realizează într-o fracțiune de secundă.

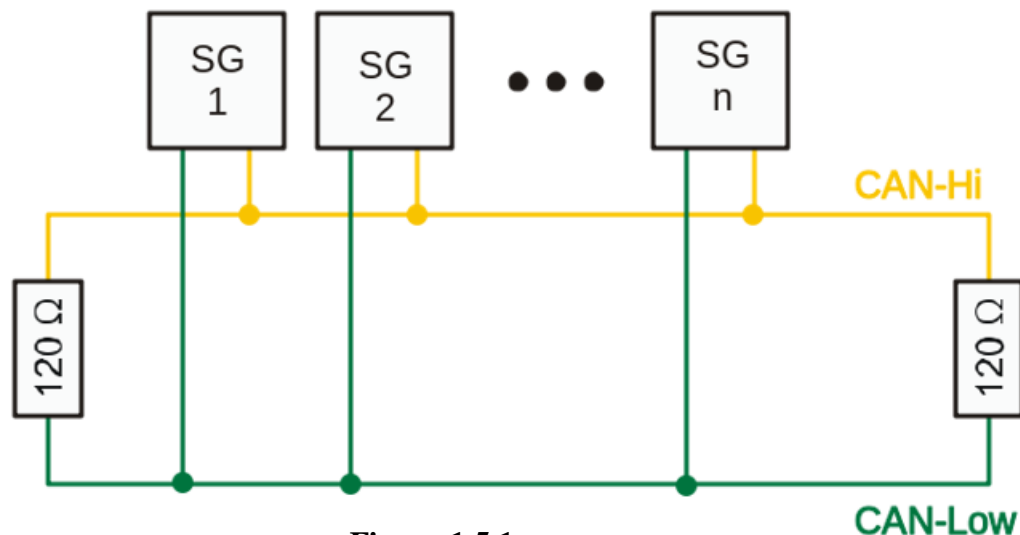


**Figura 1.4.8.**

### 1.5 Moduri de comunicație ale TCU-ului

Privind interacțiunea TCU-ului cu celelalte elemente, desprindem 2 moduri prin care ele comunică între ele:

#### 1. Controller Area Network (CAN)



**Figura 1.5.1**

- Modul autonom, poate organiza singur transmisia și recepția datelor conform specificațiilor
- Chipul modului de CAN trimite și recepționează în mod general și standard date cu 11 biți de identificare
- Are implementat și modul extins de date, cu 29 biți de identificare
- Se folosesc 2 pini pentru a interfața cu magistrala de transmisie
- Pinii respectivi trebuie să fie legați între ei printr-o rezistență de  $120\Omega$ , deoarece fiecare fir are o tensiune diferită, rezultând o cădere de tensiune între cele două fire
- Modul folosit pentru comunicarea facilă între calculator și microcontroller
- Programul scris în limbajul C este compilat și apoi se generează un fișier cu extensia .hex
- Acest fișier .hex este încărcat într-un program ce se folosește de transmisia prin CAN și flash-uieste microcontroller-ul cu programul dorit

## 2. FlexRay

- Este un protocol de comunicații în domeniul automotive, dezvoltat de Consorțiul Flexray
- Acesta este conceput pentru a fi mai rapid și mai fiabil decât CAN și TTP
- Diferența de calitate față de celelalte protocoale se resimte în preț, fiind mult mai costisitor
- Prima dată a fost folosit în producția seriei de vehicule BMW X5
- Utilizarea completă a fost introdusă în 2008 la noul BMW Seria 7
- Pe lângă vehiculele amintite anterior, a mai fost folosit la producerea:
  - Audi A4
  - Audi A6
  - Audi A7
  - Audi A8
  - Lamborghini Huracan
  - Mercedes-Benz S-Class

Interfața pentru rețeaua FlexRay, **VN7600**, este ideală pentru dezvoltarea, simularea sau testarea rețelelor FlexRay. Posibilitățile de conectare prin USB fac ca aceasta să fie aplicabilă în mai multe locuri diferite (aici putem aminti ca locuri aplicabile laboratorul sau chiar pe durata unui test-drive).

În plus, interfețele oferă acces la CAN.

În cadrul proiectului desfășurat de către compania Continental, au existat 2 generații:

### 1.6 Generația\_1 a proiectului DL382

1. **Generația 1**, după multe căutări succesive ale unui microcontroller satisfăcător, după înlocuiri multiple, s-a ajuns la ideea de a folosi un microcontroller de la Infineon și anume Infineon Tricore [IFX] TC1784

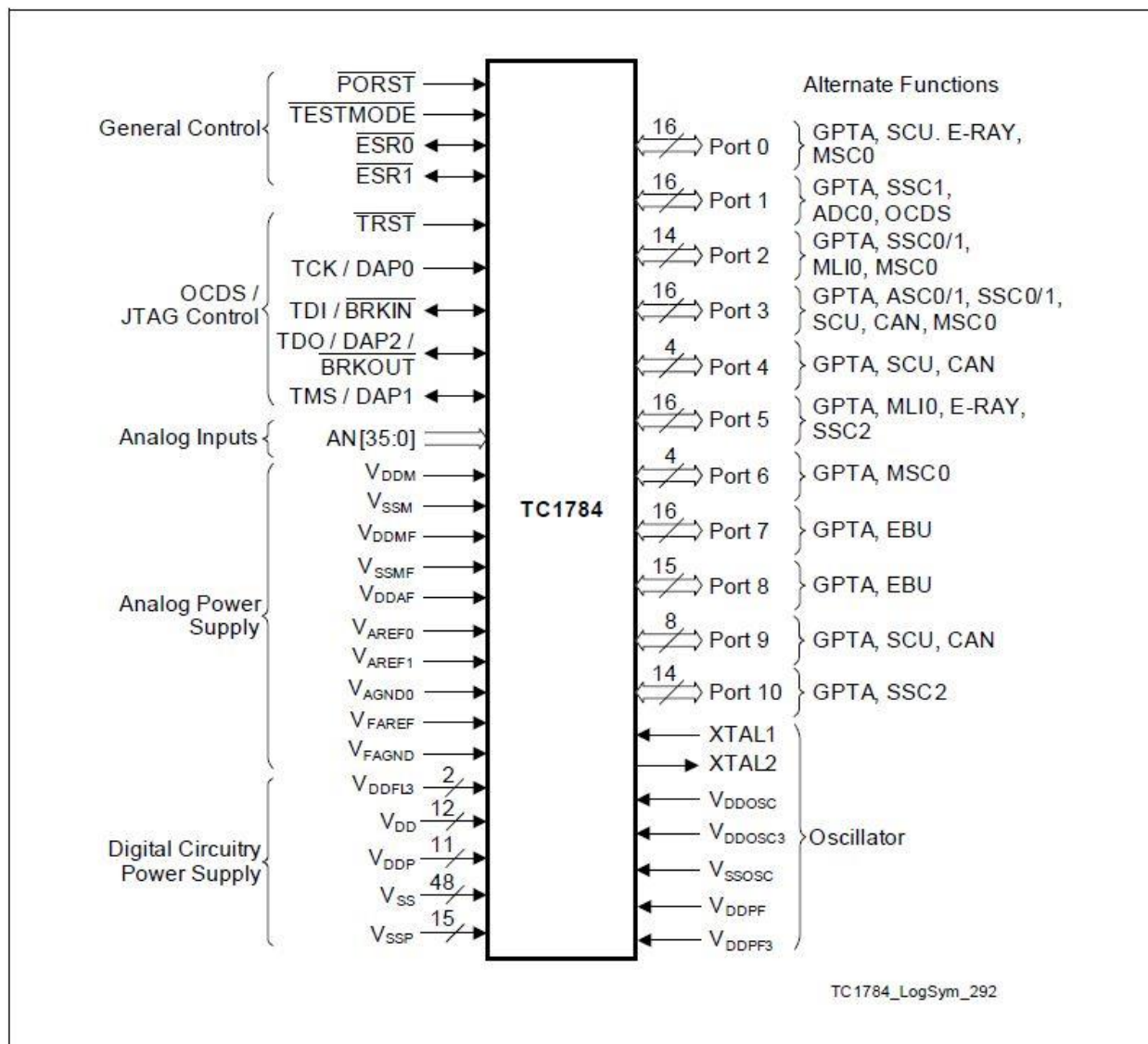


Figura 1.6.1

În continuare, o să ilustrez schema bloc a microcontroller-ului Infineon TC1784:

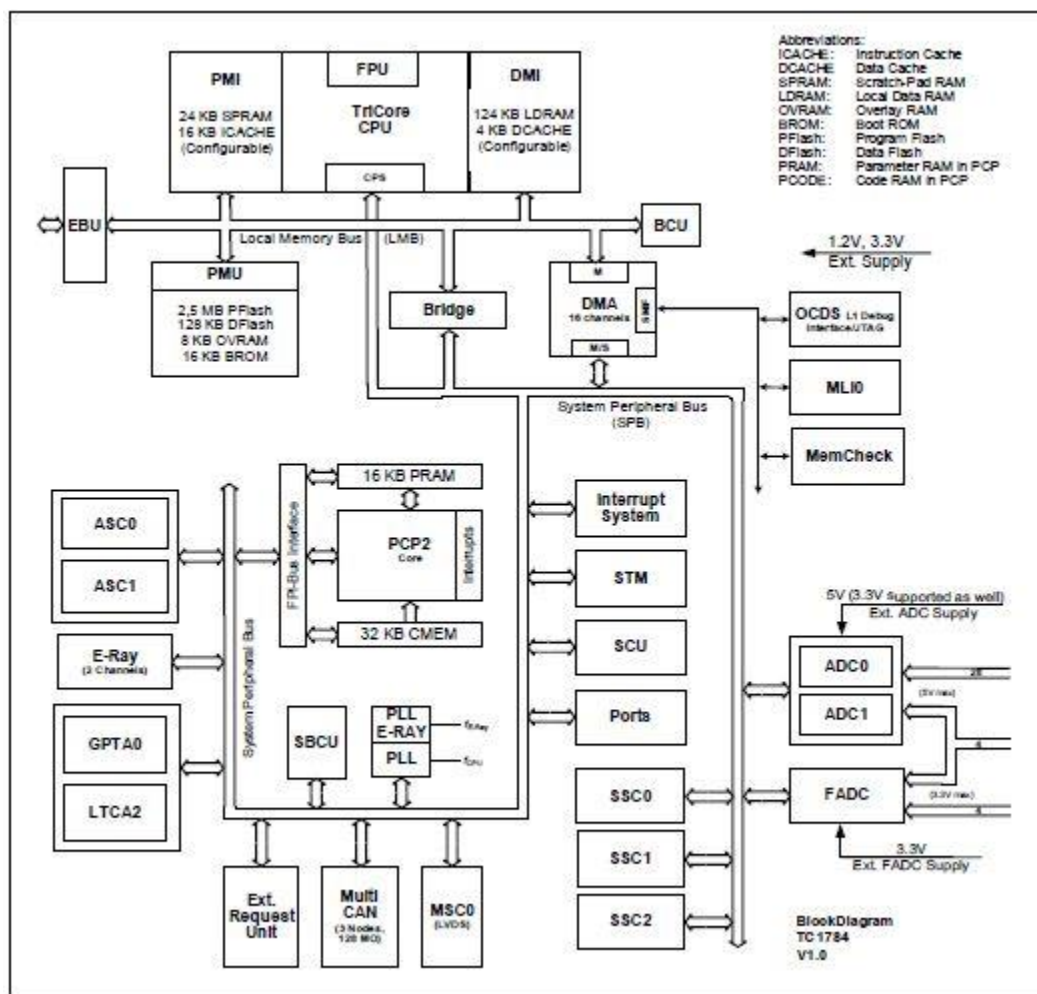


Figura 1.6.2.

- Din punct de vedere al construcției acestuia, desprindem următoarele caracteristici:
  - Frecvența maximă a CPU-ului: 180 MHz
  - Frecvența maximă a PCP-ului: 180 MHz
  - Frecvența maximă a sistemului: 90MHz
- Structural și privind CPU-ul, microcontroller-ul TC1784 include un CPU de mare performanță și un procesor de control periferic-PCP.
- În ceea ce privește PCP-ul, acesta este un procesor de control periferic foarte flexibil optimizat pentru controlul întreruperilor, “descărcând” astfel CPU-ul.
- Totodată, TC1784 include un controller DMA (Direct Memory Access) cu 16 canale DMA independente.
- TC1784 include un sistem programabil de întrerupere cu următoarele caracteristici:
  - Răspuns rapid de întrerupere



- Sisteme independente de întrerupere pentru CPU și PCP
- Noduri de cereri de servicii programabile (SRN)
- Fiecare SRN poate fi mapat la sistemul de întreruperi al CPU-ului sau PCP-ului
- Schemă flexibilă de prioritizare a întreruperilor cu 255 de nivele de prioritate întreruperi per întrerupere sistem

## 1.7 Generația\_2 a proiectului DL382

2. **Generația 2**, a folosit un alt microcontroller de la Infineon și anume Aurix TC277:

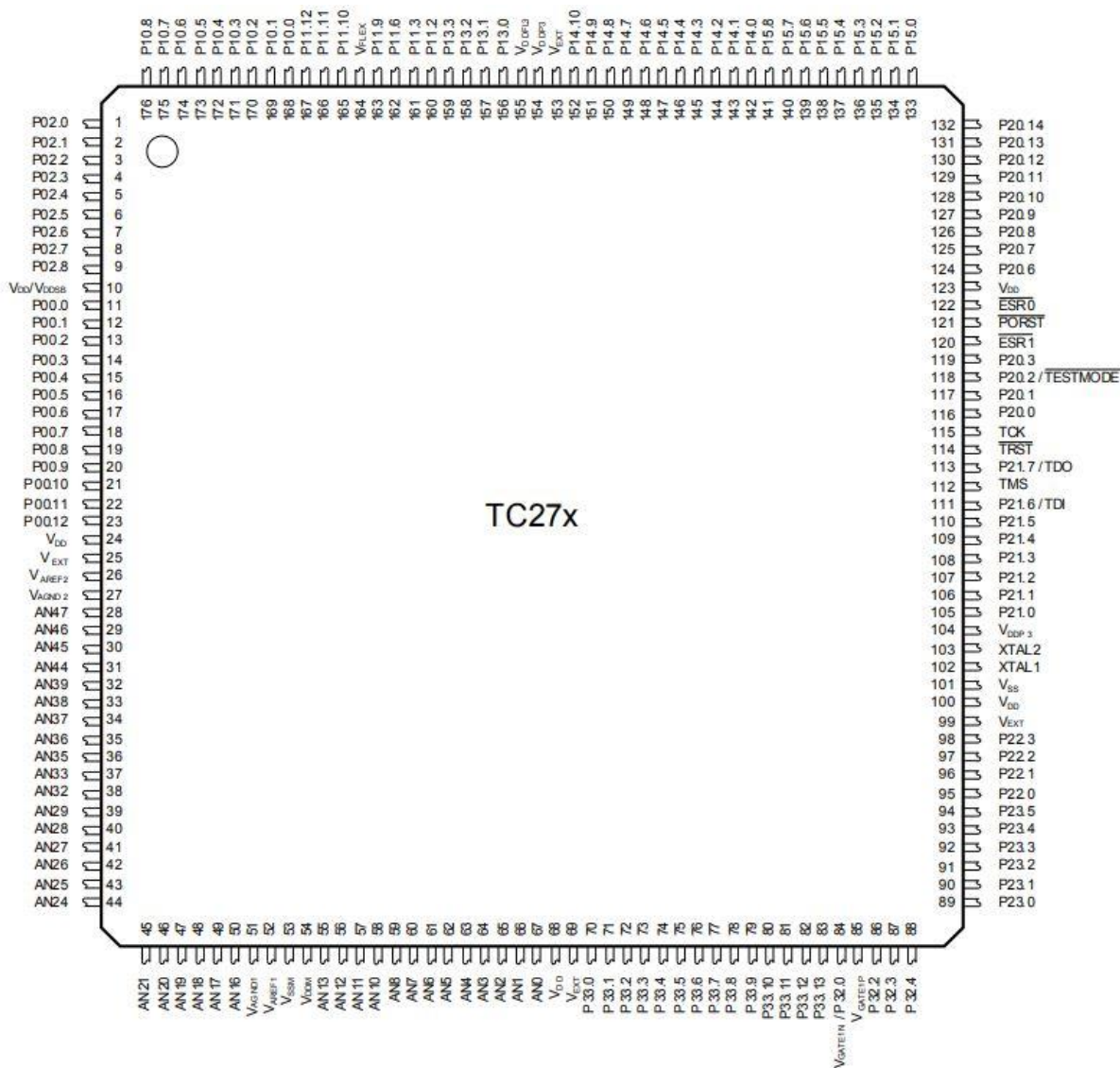
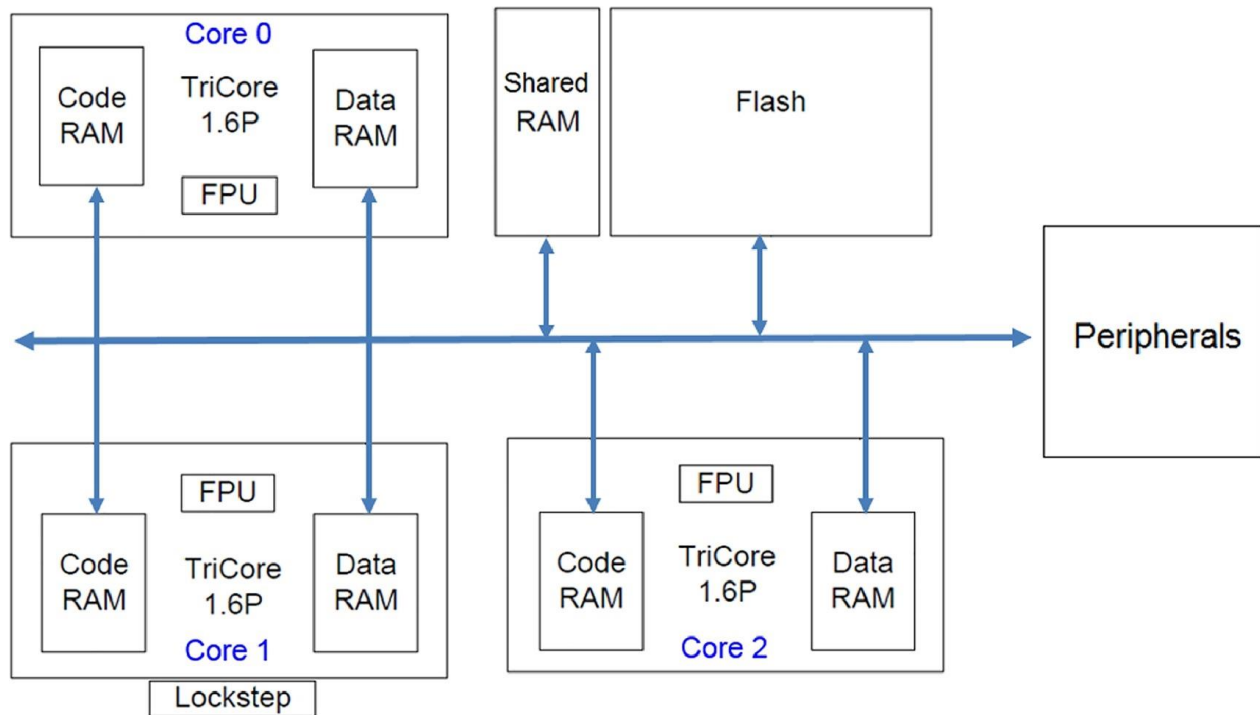


Figura 1.7.1

În continuare, o să prezint schema bloc simplificată a microcontroller-ului Aurix TC277:



**Figura 1.7.2.**

- Tehnologie TriCore, **core0 core1 core2**, cu o frecvență de până la 200 MHz per core
- Suportă operațiile cu virgulă flotantă și virgule fixă cu toate core-urile
- Temperatură: -40°C ... 145°C
- Alimentat la 5.5V sau 3V
- În ceea ce privește conectivitatea, amintesc:
  - FlexRay
  - SPI
  - CAN
  - LIN
  - UART
  - Ethernet

În fiecare dintre cele 2 generații, cu acronimele Gen1, respectiv Gen2, avem 2 metode pentru comanda cutiei de viteze automate: **SBC** (Shift by Cable) și **SBW** (Shift by Wire).

Despre **SBW**, putem spune că este sistemul prin care modurile de transmisie sunt antrenate/schimbate într-un automobil prin comenzi electronice, fără a avea vreo legătură mecanică între maneta schimbătorului de viteze și transmisie.



În mod tradițional, prin metoda **SBC**, schimbarea transmisiei a fost realizată prin legături mecanice pentru a pune vehiculul în poziții cum ar fi Park (P), Reverse (R), Neutru (N), Drive (D), folosind un mâner montat pe coloana de directive sau un schimbător de viteze în apropierea consolei centrale.

Prin metoda **SBW**, se elimină spațiul de rutare necesar pentru a acoperi legăturile mecanice dintre schimbătorul de viteze și transmisie. În plus, asigură deplasarea fără efort prin apăsarea unui buton sau butoane. Prin această optimizare se elimină orice efort din partea șoferului care-și selectează viteza.

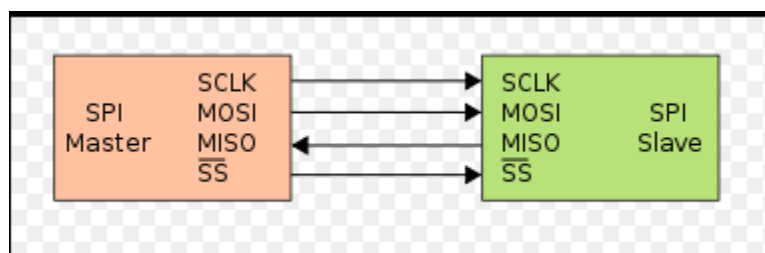
## 1.8 Comunicarea microcontrolerului cu chipul EEPROM

Microcontrolerul comunică cu chipul de EEPROM prin intermediul interfeței SPI.

Interfața serială SPI (Serial Peripheral Interface) este o interfață sincronă standard de mare viteză, ce operează în mod full duplex. Numele ei a fost dat de Motorola. Ea e folosită ca sistem de magistrală serială sincronă pentru transmiterea de date, unde circuitele digitale pot să fie interconectate pe principiul master-slave. Aici, modul master/slave înseamnă că dispozitivul (circuitul) digital master inițiază cuvântul de date. Mai multe dispozitive (circuite) digitale slave sunt permise cu slave select individual, adică cu selectare individuală.

SPI-ul are patru semnale logice specifice:

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



**Figura 1.8.1**

Pentru a începe comunicarea, master-ul mai întâi configurează ceasul, folosind o frecvență mai mică sau egală cu maximumul frecvenței suportată de slave. Aceste frecvențe sunt de obicei în intervalul 1-70 MHz. Atunci master-ul setează slave select-ul pe nivelul 'jos' (en. low) pentru chip-ul dorit. Dacă este necesară o perioadă de așteptare (ca la conversia analog-digitală) atunci master-ul așteaptă cel puțin acea perioadă de timp înainte de a începe ciclurile de ceas.

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

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

Nu toate transmisiile de date necesită toate aceste operații (de ex. transmisia unidirecțională) deși acestea se petrec.

## 1.9 Situație actuală

Cu toții suntem conștienți că ne aflăm în secolul vitezei. În secolul cu cele mai multe schimbări la nivelul tehnologiei. În secolul unde lumea nu mai are răbdare și toți își doresc ca lucrurile să se întâmple cât mai rapid cu putință.

Pe aceste considerente s-au bazat și cei de la Audi. Ei își doresc ca toate elementele să se inițializeze și să interacționeze între ele într-un timp cât se poate de scurt.

Din această dorință și totodată cerere din partea clientului, s-a născut și proiectul meu de diplomă. Astfel, am ajuns să fiu nominalizat ca fiind responsabil pentru îmbunătățirea timpului în care **TCU**-ul reacționează. Prin reacția **TCU**-ului înțeleg timpul său de inițializare, de unde derivă mai apoi comunicarea cu celelalte componente ale mașinii.

Durata inițializării unității de control a transmisiei, adică a **TCU**-ului, depinde într-o proporție covârșitoare de citirea totală a datelor din **EEPROM**.

În cadrul companiei Continental, divizia Powertrain Transmission, există un proiect cu distribuție multicore care folosește o unitate de control a transmisiei, **TCU**, pentru a manipula cutia de viteze a mașinilor Audi.

În trecut, folosind un proiect mai vechi, era utilizată o unitate de control a transmisiei cu un singur nucleu pentru a manevra cutia de viteze a mașinilor Audi.

Atât în proiectul părinte, cu singur core, cât și în proiectul recent (cel multicore), citirea datelor din **EEPROM** se face pe un singur core.

Dacă în proiectul vechi, soluția de citire pe un singur core era singura disponibilă, pe cel multicore se dorește distribuirea blocurilor de memorie **EEPROM** alături de celelalte elemente necesare inițializării **TCU**-ului într-un mod cât mai avantajos pe 2 core-uri.

## 1.10 Temă propriu-zisă

Având în vedere că în noul proiect unitatea de control a transmisiei, **TCU-ul**, dispune de un microcontroller cu 3 core-uri, îmi doresc să îmbunătățesc considerabil durata de citire a datelor din **EEPROM**. Astfel, din momentul în care s-a acționat cheia de pornire a mașinii (**KL15 => ON**), să dispun blocurile de memorie **EEPROM**, alături de celelalte elemente necesare inițializării **TCU**-ului, într-un mod avantajos și benefic pe 2 core-uri.

Ținând seama de faptul că microcontrollerul “primește” informații de la chipul memoriei **EEPROM** prin intermediul interfeței **SPI**, rezultă că funcționalitatea interfeței trebuie “transferată” și pe al-2lea core ce urmează a fi utilizat (pe core-ul 1, utilizat inițial, interfața a fost implementată).

## 2.Proiectare și implementare

După cum am descris în linii mari, în capitolul 1, îmi propun să reduc timpul de inițializare al TCU-ului (unitatea de control a transmisiei). Durata de timp între momentul acționării de către utilizator a cheii de la mașină (KL15 => ON) și momentul în care TCU-ul reacționează este influențată în proporții extrem de mari de citirea datelor din EEPROM.

În proiectul anterior, proiectul “părinte”, citirea datelor din EEPROM s-a făcut pe un singur core, fiind singura soluție la acel moment, microcontroler-ul din TCU având un singur core.

În cele din urmă, pentru ultima versiune a proiectului (și anume generația 2), s-a trecut la un microcontroler cu 3 core-uri.

Cu toate acestea, având disponibilitatea de a folosi într-un mod benefic și avantajos această nouă caracteristică, citirea datelor din EEPROM a rămas pe un singur core. Așadar, la un moment dat, s-a luat o decizie normală, previzibilă și logică, de a folosi microcontrolerul la adevărata performanță.

În cadrul proiectului, toate datele din EEPROM sunt citite de-a lungul a 2 funcții: `DlsReadByIndexInit()` și `EepReadImmediate()`.

Fiecare bloc de memorie din EEPROM, are mai multe informații prin care este unic:

- un index, care ne arată și adresa blocului
- dimensiunea, care arată câți octeți are fiecare bloc

În momentul în care se acționează cheia, microcontrolerul cere date de la chipul de EEPROM al TCU-ului. Pentru a comunica cu chipul de EEPROM, microcontrolerul se folosește de interfața SPI.

Așadar, pentru început, mi-am dorit o îmbunătățire severă a timpului de inițializare al TCU-ului, fără a “transfera” funcționalitățile interfeței SPI și pe un core ce urma a fi folosit. Deoarece SPI-ul trebuie să fie prezent doar pe core-ul care citește propriu-zis din EEPROM, am dezvoltat o primă soluție prin care să citesc blocurile de memorie din EEPROM pe un singur core și să transfer celelalte operații necesare citirii din chipul de EEPROM pe un alt core.

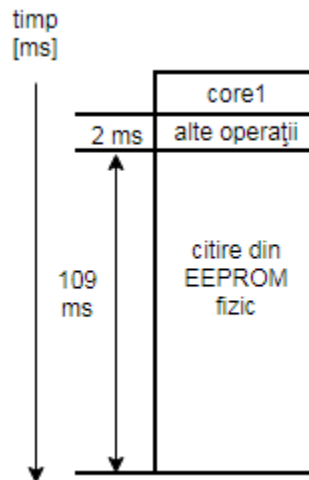
În aceste condiții, descrise anterior, se evita prezența simultană a funcționalităților interfeței SPI pe mai multe core-uri.

În practică, lucrurile nu s-au întâmplat așa cum au fost gândite, proiectate cu “hârtia și creionul”. Îmbunătățirea, existentă totuși, nu a fost nici pe departe cea cerută de client. Dacă în mod clasic, inițializarea TCU-ului avea o durată de aproximativ 200 milisecunde, aici însumând citirea “brută” a datelor, dar și celelalte operații, cu prima soluție am ajuns undeva spre 183 milisecunde.

Deoarece rezultatul nu a fost cel așteptat, nici pe departe, am început investigația și astfel am măsurat cât durează doar citirea din EEPROM-ul fizic, pentru a putea dezvolta în continuare soluții viabile.

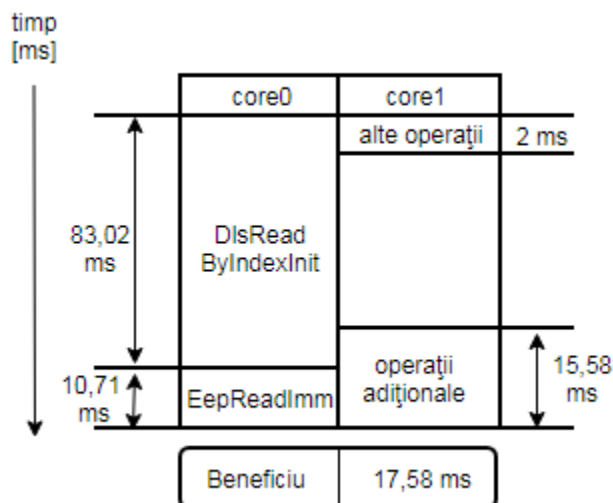
Astfel, am urmărit exact unde sunt apelate cele 2 funcții care fac citirea completă a EEPROM-ului fizic, și folosind interfața timer a microcontrolerului în cauză, înainte și după apelul funcțiilor, am ajuns la următoarele rezultate:

- citirea “brută” a datelor din EEPROM-ul fizic pe core-ul inițial folosit



**Figura 2.1**

- citirea datelor din EEPROM-ul fizic mutate pe core-ul 0 și lăsarea operațiilor adiționale pe core-ul 1, folosit inițial



**Figura 2.2.**

Prin urmare, comparând toate aceste rezultate obținute era evident că nu am făcut altceva decât să mut lucruri aproape nesemnificative pe celălalt core.

Pentru ultimele rezultate, am mutat funcția de inițializare a EEPROM-ului fizic în task-ul de init al core-ului 0.

Datorită timpului de inițializare total al TCU-ului, următorul pas logic era să determin exact toate elementele care alcătuiesc acest timp de inițializare. Astfel am ajuns la următoarele:

- citirea datelor din **chipul de EEPROM**
- operații necesare citirii din EEPROM-ul fizic
- citirea din **EEPROM-ul emulat**
- **switch off path check** sau **switch off path test**

Un element important în toată implementarea îl reprezintă oglinda RAM. Oglinda RAM este o stare de tranziție a informațiilor necesare inițializării TCU-ului.

Astfel, imediat după acționarea cheii de către utilizator, microcontrolerul primește informații din EEPROM-ul fizic și din EEPROM-ul emulat și le salvează în oglinda RAM. Acolo, datele pot suferi modificări chiar în timpul funcționării. Mai departe, toate datele din oglinda RAM sunt utilizate local.

După oprirea mașinii, KL15 => OFF, datele din RAM se pierd. Rămân salvate doar în EEPROM. De acolo vor fi cerute din nou pentru inițializarea TCU-ului, la următoarea rulare a mașinii.

O să ilustrez o mică schemă care arată procesul în linii mari

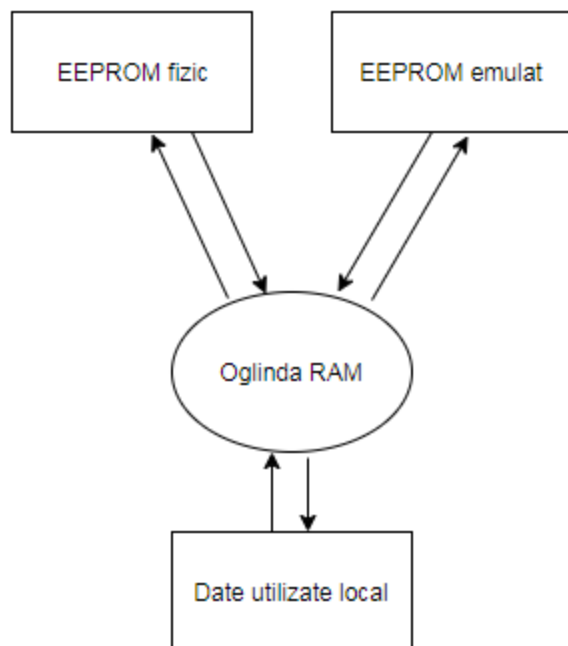


Figura 2.3

**Switch off path check-ul (SOPC)** este un element esențial în inițializarea TCU-ului, este cel care determină starea valvelor și a tuturor conexiunilor pentru a se asigura că totul rulează la parametri optimi înainte de pornire. Este cel responsabil cu oprirea sistemului în cazul detecției unei erori de orice natură.

Prin **SOPC**, trebuie să fim capabili să dovedim că mecanismele de safety funcționează. În această categorie intră toate funcțiile care pot duce la răniri. Acestea din urmă, trebuie să fie tratate special conform ISO 26262.

În ceea ce privește departamentul Transmission, în proiectele noastre, în caz de eroare, trebuie să dezactivăm imediat controlul electric, să nu mai primească motorul sau valvele energie.

Prin această funcție de safety înțelegem 2 canale redundante. Prin aceste 2 canale, se dorește existența unei verificări speciale, suplimentare. De reținut este faptul că cele 2 canale sunt independente. Este arhisuficient ca unul din canale să răspundă cu o eroare, iar sistemul intră imediat în safe state, starea de siguranță.

Înainte de **Switch Off Patch Check (SOPC)**, microcontrolerul își efectuează check-urile proprii prin safety\_lib. Totodată, Cy-ul își face un self test pentru a fi sigur că poate răspunde la cererile venite de la microcontroler.

Dacă ne îndreptăm atenția spre starea de siguranță, safe state, desprindem mai multe surse care ne pot duce în această situație:

- se verifică Question<->Answer între microcontroler și unitatea de monitorizare. Dacă vin prea multe răspunsuri greșite, Cy va transmite starea de siguranță. Dacă situația de greșală se restaurează înainte ca microcontrolerul să-și dea reset, atunci poate să-și revină întreg procesul
- microcontrolerul poate să mai ceară starea de siguranță pe o linie digitală supervizată de către unitatea de monitorizare
- după ce s-au efectuat verificările că se poate intra în safe state, se resetează și se face self test dacă BLDC răspunde la schimbări

În plus, toate chestiile în derularea SOPC-ului, trebuie să respecte un time\_check. Altfel, în cazul depășirii acestui timp tolerant, se va genera un timeout. În aceste condiții, pe comunicația TCU-ului cu celelalte elemente ale mașinii se va transmite un mesaj de eroare.

După măsurările aferente, **SOPC-ul** durează undeva la 63 milisecunde.

Celălalt timp rămas se datorează EEPROM-ului emulat.

În mod tradițional, **SOPC-ul**, se executa după citirea din EEPROM-ul fizic și din EEPROM-ul emulat.

Sesizând faptul că citirea datelor din chipul de EEPROM durează undeva la 111 milisecunde, mi-am dat seama că dacă aș reuși să pun în “paralel”, pe de-o parte citirea din EEPROM-ul fizic, iar de cealaltă parte citirea din EEPROM-ul emulat alături de SOPC, toată acțiunea ar tinde spre un oarecare echilibru în ceea ce privește cele două procese.

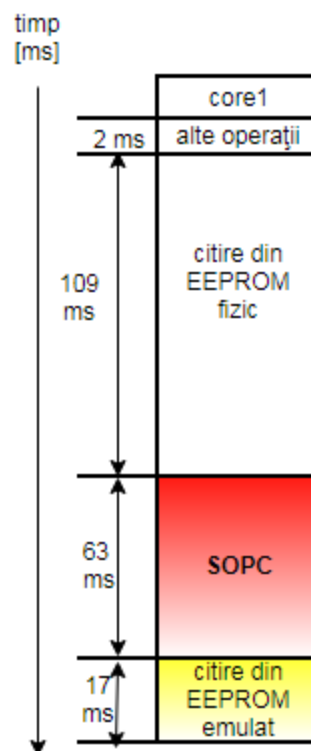
În timp ce pe core-ul 0 se începe citirea clasică a blocurilor de memorie din EEPROM-ul fizic, în același timp să înceapă și pe core-ul 1 realizarea SOPC-ului urmată de citirea din EEPROM-ul emulat.

Privind cele spuse anterior, prin încercare mea de a "echilibra" durata de inițializare a TCU-ului, țin să precizez că în toate aplicațiile și proiectele din domeniul IT, se urmărește dezvoltarea cât mai "curată" și eficientă a paralelismului. Aici, avem câteva exemple: Thread-uri, Pipe-uri etc. Știm foarte bine că, deși se spune despre soluțiile amintite anterior ca se execută în paralel, nu este vorba despre un paralelism "curat".

După mai multe "creionări" și documentări ale ideii ce urma să o implementez, am încercat cele spuse anterior.

Deoarece și SOPC-ul are nevoie de comunicație cu interfața SPI, atunci a trebuit evident ca driverul de SPI să fie prezent pe cele 2 core-uri.

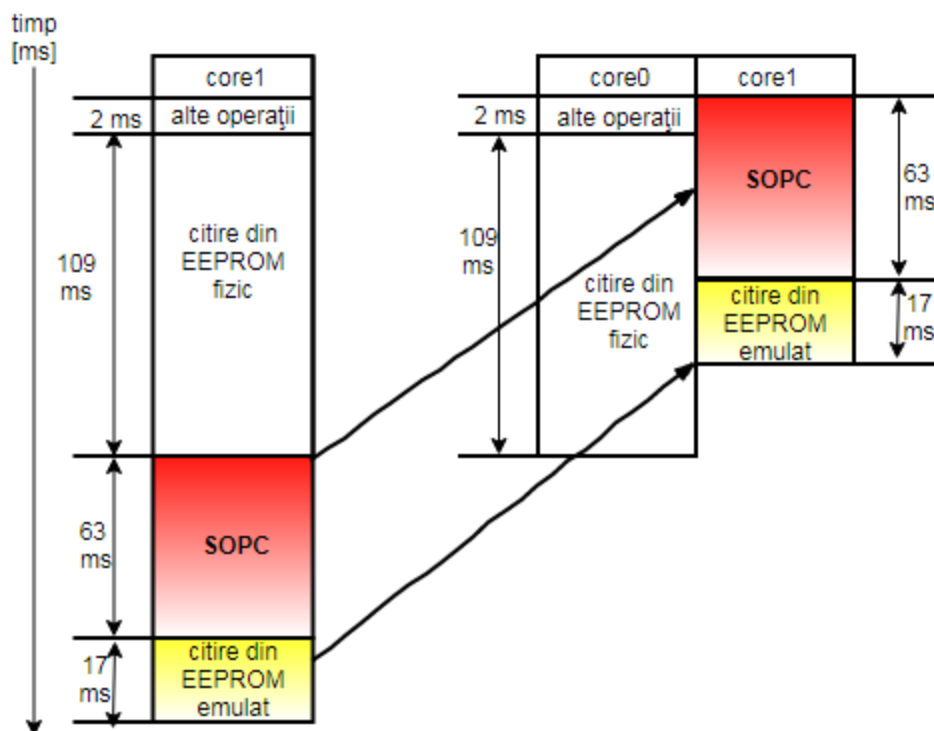
Situația actuală de inițializare o să o ilustrez succint:



**Figura 2.4**



Apoi, implementând în totalitate ideea enunțată, am ajuns la următoarele rezultate:



**Figura 2.5**

Așadar, timpul de inițializare al TCU-ului cu aceste modificări ar ajunge aproximativ la durata citirii tuturor datelor din EEPROM-ul fizic.

Un element de punctat ar fi că EEPROM-ul emulat nu necesită comunicație cu interfața SPI.

Privind problema cu driver-ul de SPI, care este obligatoriu funcțional pe cele 2 core-uri, am realizat 2 job-uri diferite ale interfeței care să vină cu întreruperi pe cele două core-uri.

- Cy328, pentru core1 respectiv SOPC și EEPROM emulat
- SCC, pentru core0 respectiv citire din EEPROM fizic

Dacă îmi îndrept atenția spre citirile din EEPROM, trebuie să precizez că la pornire se cer date de la ultima funcționare a mașinii. În EEPROM rămân date salvate care se colectează:

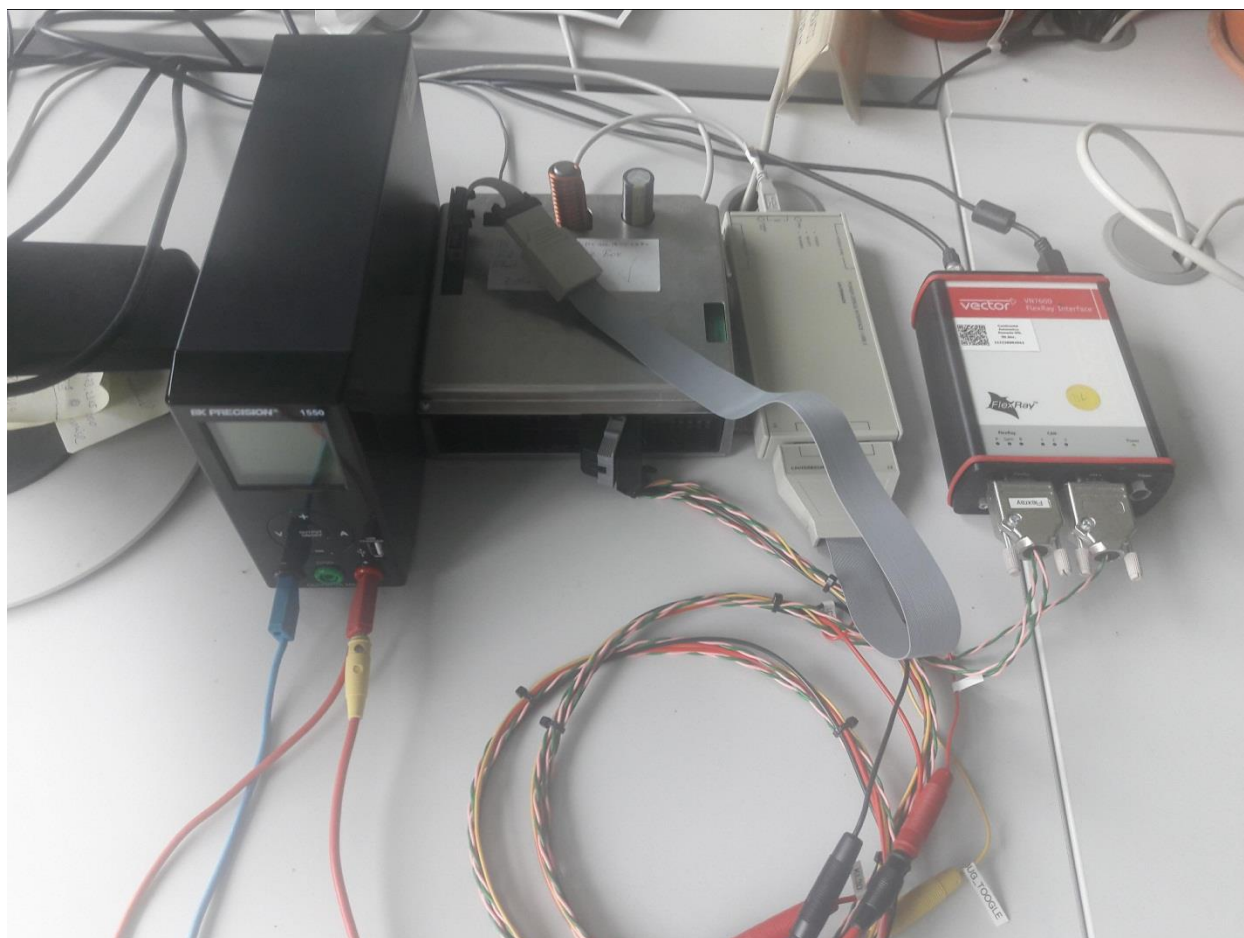
- În momentul în care se face ignition OFF/ON (KL15 => OFF, KL15 => ON)
- În timpul funcționării automobilului
- Doar la cerere, în momente de timp bine stabilite

## 3.Testare

### 3.1 Structura sistemului

Pentru acest proiect am dispus de un sistem format din următoarele componente:

1. Sursa de tensiune
2. VN 7600
3. **TCU**, unitatea de control a transmisiei
4. Trace32, debugger
5. Cablu



**Figura 3.1.1**

### 3.2 Rezultate experimentale

După ce am realizat codul cu modificările dorite, următorul pas pe care trebuie să-l facem e să încărcăm (să flashuim) acel cod pe TCU, unitatea de control a transmisiei. Pentru a realiza acest lucru, ne folosim de o unealtă des utilizată în cadrul departamentului nostru și anume Diagra.

În procesul propriu-zis de flashuire, primul pas e să alegem unul din cele două moduri de comunicație ale TCU-ului:

- Controller Area Network (CAN)
- FlexRay

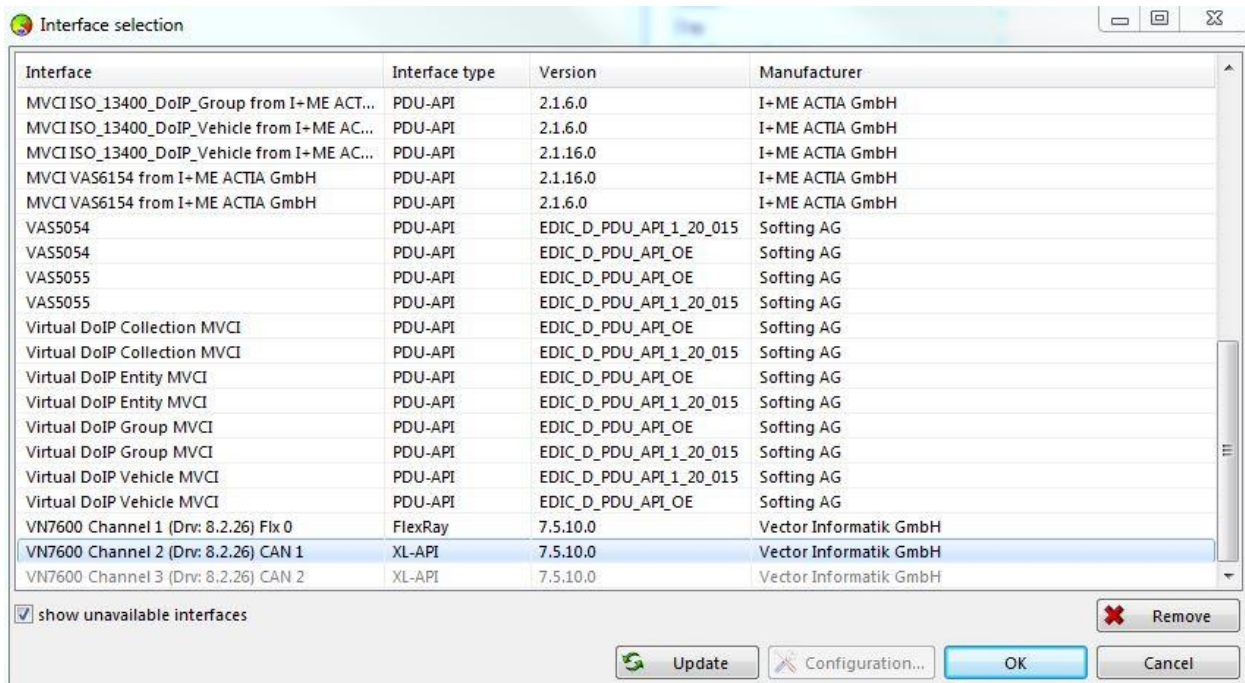


Figura 3.2.1

În continuare, fiecare funcție a TCU-ului are un anumit identificier. Prin meniul de selecție al funcțiilor, putem să alegem spre vizualizare oricare proprietate pe care ne-o dorim.

După căutarea și găsirea după identificier a lucrurilor ce se doresc vizualizate, din partea stângă a meniului, se apasă săgeata dreapta și se trec în partea dreaptă. Acolo, sunt toate elementele ce vor fi mai târziu observate.

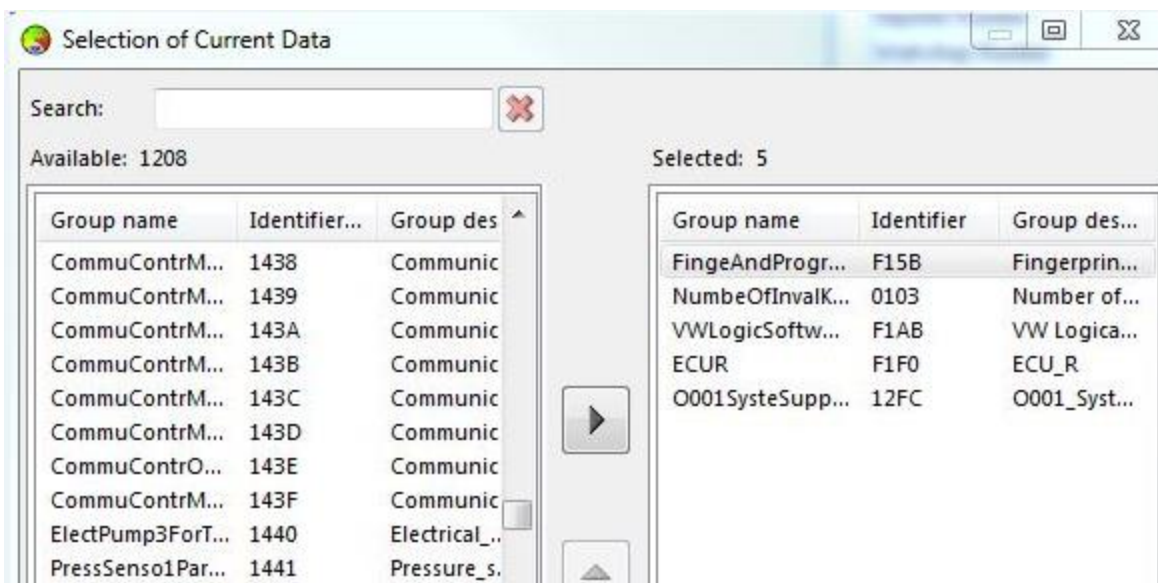


Figura 3.2.2

După ce am selectat tot ce ne-am dorit, apăsăm butonul OK și revenim la meniul principal. În cele ce urmează o să observăm informațiile grupate pe mai multe coloane, astfel:

- Descrierea valorilor
- Valoarea
- Identifier
- Descrierea grupului din care fac parte

Measured values				
Value description	Value	Identifier	Group description	
Year		00 \$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
Month		01 \$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
Day		01 \$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
VW Device Number		0 \$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
Importer Number		0 \$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
Workshop Number		0 \$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
Programming State	Correct Result	\$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
Year	18	\$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
Month	06	\$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
Day	07	\$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
VW Device Number	200	\$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
Importer Number	780	\$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
Workshop Number	99	\$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
Programming State	Correct Result	\$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
Year	18	\$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
Month	06	\$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
Day	07	\$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
VW Device Number	200	\$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
Importer Number	780	\$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
Workshop Number	99	\$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
Programming State	Correct Result	\$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
Year	18	\$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
Month	06	\$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
Day	07	\$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
VW Device Number	200	\$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
Importer Number	780	\$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
Workshop Number	99	\$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
Programming State	Correct Result	\$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
Year	00	\$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
Month	01	\$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
Day	01	\$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
VW Device Number	0	\$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
Importer Number	0	\$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
Workshop Number	0	\$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
Programming State	Correct Result	\$F15B	Fingerprint And Programming Date Of Logical Software Blocks	
Counter Value	0	\$0103	Number of Invalid Keys	
data block 0	X704	\$F1AB	VW Logical Software Block Version	
data block 1	XB0X	\$F1AB	VW Logical Software Block Version	
data block 2	XB0X	\$F1AB	VW Logical Software Block Version	
data block 3	XB0X	\$F1AB	VW Logical Software Block Version	
data block 4	---	\$F1AB	VW Logical Software Block Version	
Data	40 0C 8F 07 16 00 F6 01 0A 1C 08 21 60 01 A2 25 72 8F FC 3C	\$F1F0	ECU_R	
System Supplier HW/ Version	0600	\$12FC	0001_System Supplier ECU Identification	
System Supplier HW/ Version 2	Application	\$12FC	0001_System Supplier ECU Identification	
System Supplier Calibration Data Identification	B9 2.0IT 185kW	\$12FC	0001_System Supplier ECU Identification	
System Supplier Software Identification	B04X 180605171429GUS 8231VAC00 2	\$12FC	0001_System Supplier ECU Identification	

Selection
Read
Cyclical
Once

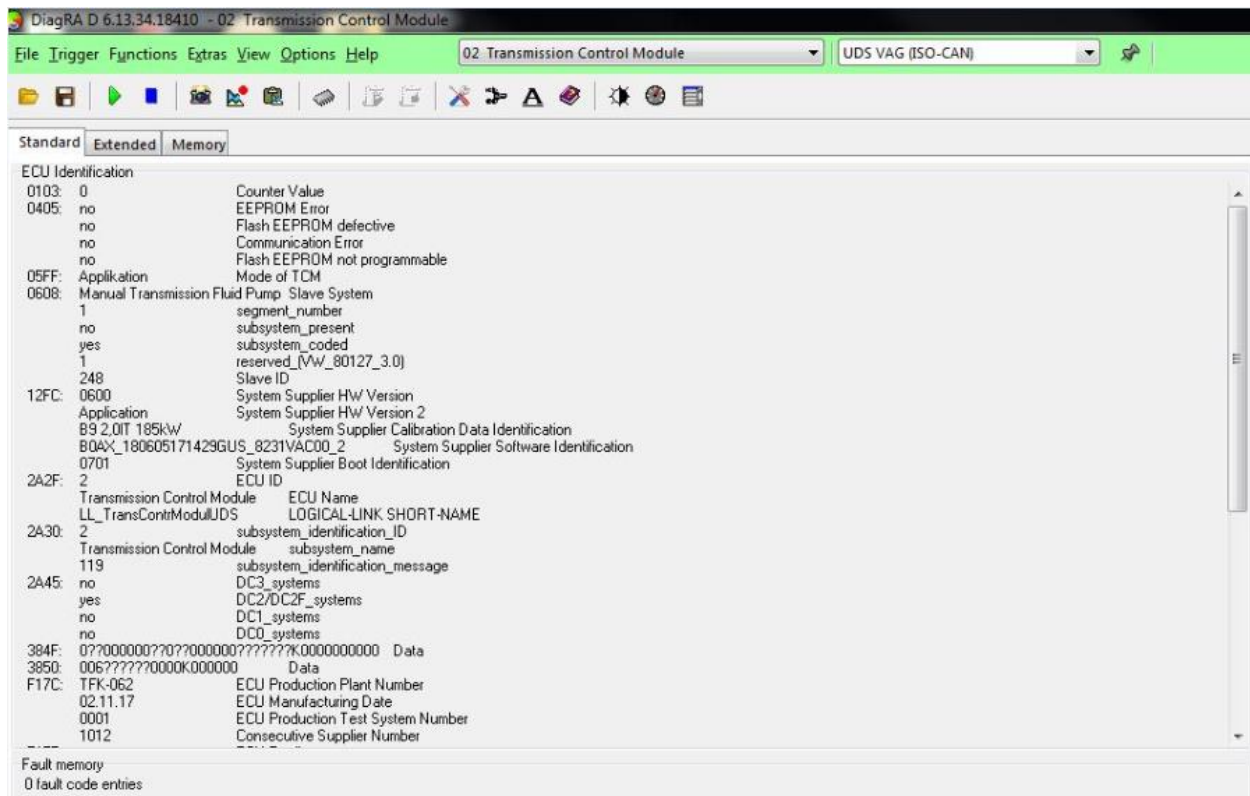
LOG EV TCMDL382021 002031
VN7600 Channel 2 (Drv: 8.2.26) CAN 1 / XL-API
File info...

Figura 3.2.3



Încărcarea valorilor se face prin apăsarea butonului READ, iar citirea valorilor se poate face o singura dată sau chiar ciclic, la fiecare interval de timp.

În partea de jos, putem să observăm canalul setat de comunicație.



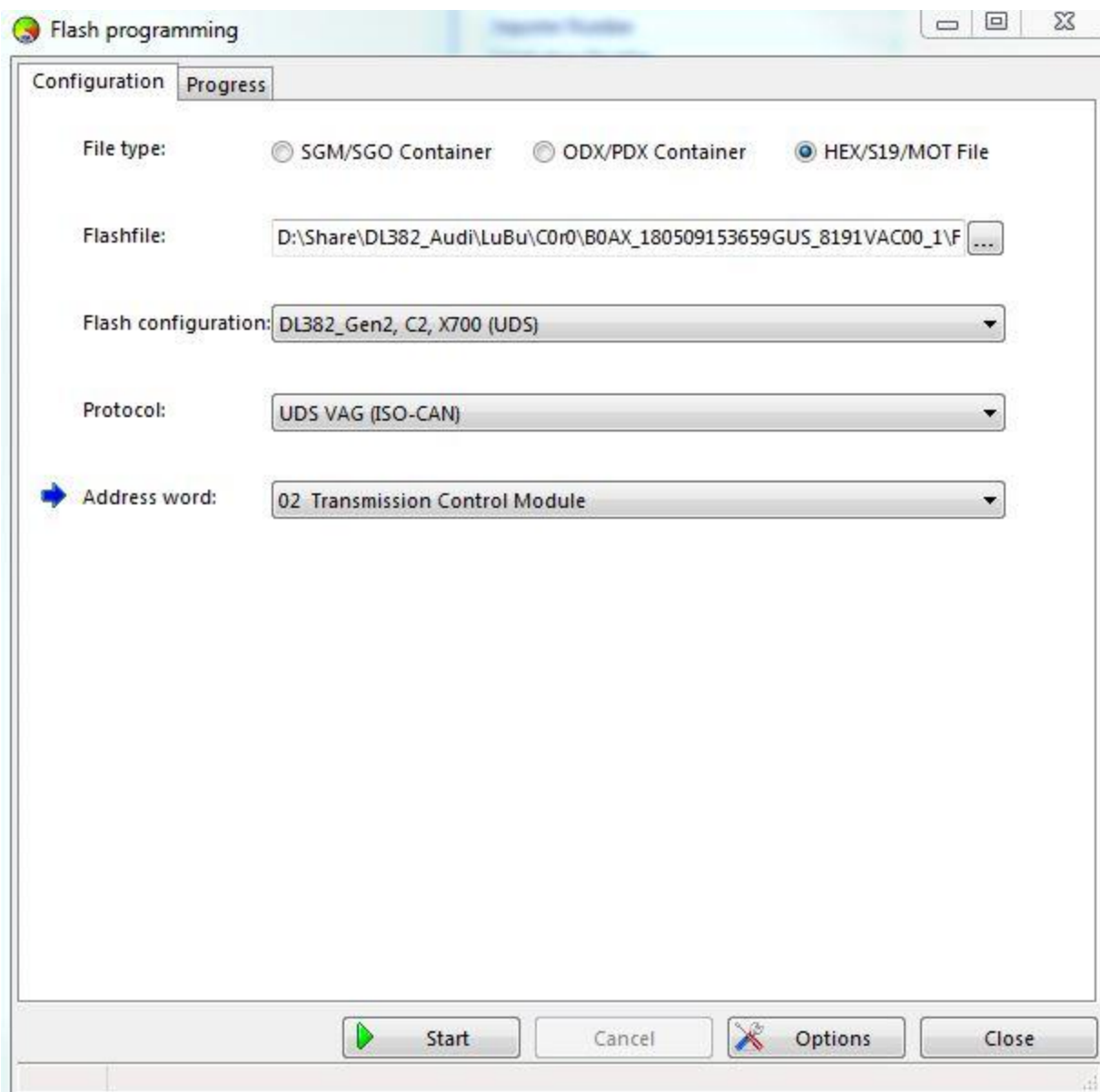
**Figura 3.2.4**

Încărcarea valorilor se face prin apăsarea butonului READ, iar citirea valorilor se poate face o singura dată sau chiar ciclic, la fiecare interval de timp.

În partea din stânga putem citi sau șterge DTC-uri.

Aceste au fost date generale despre interfața tool-ului cu utilizatorul.

La opțiunile de flashuire se încarcă fișierul cu extensia .hex, ce a fost generat după compilarea codului scris în limbajul de programare C.



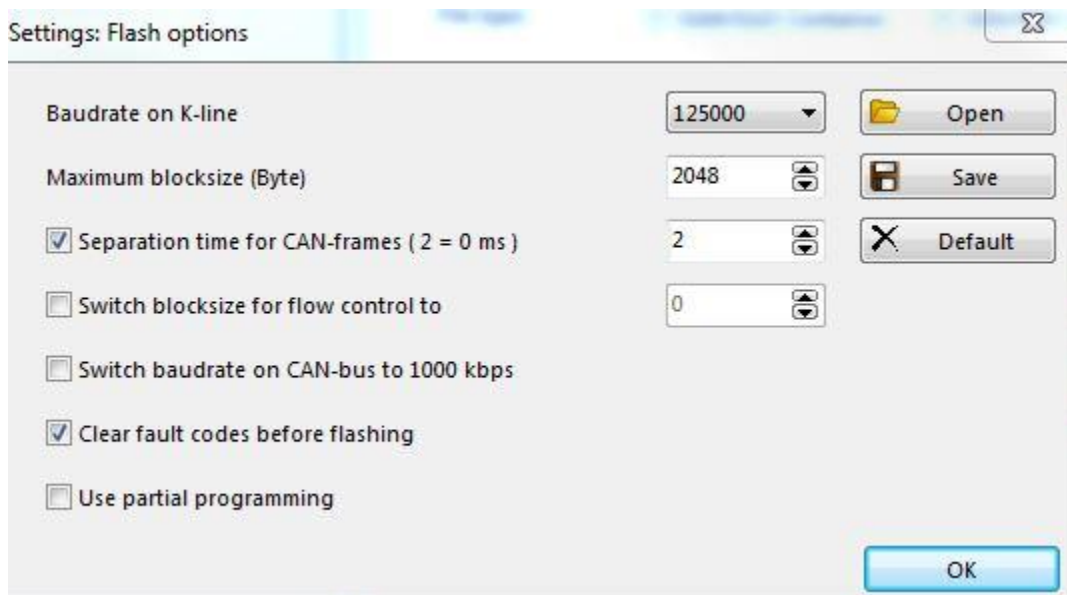
**Figura 3.2.5**

După ce am ales fișierul cu extensia .hex dorit, ne rămâne să alegem configurația flashuirii, protocolul și address word-ul.

După ce caracteristicile anterioare au fost stabilite și setate, prin butonul OPTIONS, ni se deschide un panou, unde avem acces și putem modifica următoarele:

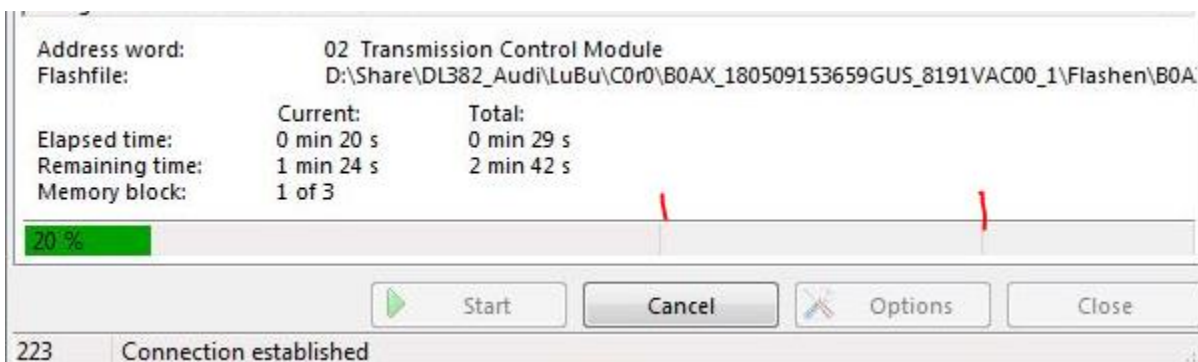
- Baudrate-ul
- Dimensiunea maximă a blocurilor de memorie, în octeți

Totodată, mai avem și niște check box-uri care selectează sau deselectează anumite funcționalități ce pot influența procesul de flashuire.



**Figura 3.2.6**

Pentru proiectul în care-mi desfășor activitatea, există 3 blocuri de memorie care trebuie scrise.



**Figura 3.2.7**

Cele 3 blocuri de memorie sunt următoarele:

- Bloc1, SW aplicativ
- Bloc2, Calibrări
- Bloc3, SW basic

După ce indicatorul ajunge la 100% pentru fiecare dintre cele 3 blocuri, procesul se sfârșește cu succes.

Astfel, TCU-ul, unitatea de control a transmisiei, rulează din acest moment cu modificările gândite, dezvoltate și implementate.



Din acest moment, putem începe vizualizare informațiilor de care avem nevoie. Ne vom folosi de tool-ul Trace32, în care a fost încărcat un fișier cu extensia .elf generat la compilarea codului pentru a putea urmări toate variabilele și structurile utilizate.

Astfel, vom genera un tabel cu informații având următoarea structură:

- Identifier-ul fiecărui bloc de memorie EEPROM
- Durata de execuție preluată din Trace32, măsurată în nanosecunde
- Durata de execuție, transformată în milisecunde
- Timestamp-ul fiecărui bloc de memorie
- Delta reprezintă diferența între 2 timestamp-uri și transformarea în milisecunde a diferenței
- Adresa blocului
- Dimensiunea blocului

ID bloc EEPROM	Durata de execuție (ns)	Durata de execuție (ms)	Timestamp (STM0)	Delta	Adresă	Dimensiune
----------------	-------------------------	-------------------------	------------------	-------	--------	------------

**Figura 3.2.8**

Pentru a popula tabelul cu structura proaspăt stabilită, este necesar să urmărim în Trace32 citirea blocurilor de memorie EEPROM. Aici, este vorba de EEPROM-ul fizic.

În cele ce urmează, trebuie să ținem cont de faptul că EEPROM-ul fizic este citit folosind 2 funcții. Prin urmare, ajungem în situația de adăuga în Trace32 2 variabile.

```
testcode = (
  (FirstParameter = 104, Timestamp = 4138932017, SystemTimer0_Duration = 2629950, ByteSize = 255),
  (FirstParameter = 171, Timestamp = 4139689247, SystemTimer0_Duration = 193650, ByteSize = 9),
  (FirstParameter = 14, Timestamp = 4139708742, SystemTimer0_Duration = 1230610, ByteSize = 124),
  (FirstParameter = 4, Timestamp = 4139832450, SystemTimer0_Duration = 219110, ByteSize = 13),
  (FirstParameter = 161, Timestamp = 4140277360, SystemTimer0_Duration = 305020, ByteSize = 21),
  (FirstParameter = 162, Timestamp = 4140309612, SystemTimer0_Duration = 254830, ByteSize = 17),
  (FirstParameter = 142, Timestamp = 4140339327, SystemTimer0_Duration = 525280, ByteSize = 48),
  (FirstParameter = 24, Timestamp = 4141360895, SystemTimer0_Duration = 194920, ByteSize = 10),
  (FirstParameter = 115, Timestamp = 4141384790, SystemTimer0_Duration = 624730, ByteSize = 61),
  (FirstParameter = 112, Timestamp = 4141454150, SystemTimer0_Duration = 144930, ByteSize = 4),
  (FirstParameter = 121, Timestamp = 4141468764, SystemTimer0_Duration = 385880, ByteSize = 68),
  (FirstParameter = 122, Timestamp = 4141508900, SystemTimer0_Duration = 90180, ByteSize = 8),
  (FirstParameter = 123, Timestamp = 4141518046, SystemTimer0_Duration = 114950, ByteSize = 14),
  (FirstParameter = 165, Timestamp = 4141529729, SystemTimer0_Duration = 98130, ByteSize = 10),
  (FirstParameter = 158, Timestamp = 4141541001, SystemTimer0_Duration = 81340, ByteSize = 6),
  (FirstParameter = 172, Timestamp = 4141551344, SystemTimer0_Duration = 160860, ByteSize = 5),
  (FirstParameter = 173, Timestamp = 4141677739, SystemTimer0_Duration = 291010, ByteSize = 20),
  (FirstParameter = 174, Timestamp = 4141710394, SystemTimer0_Duration = 413570, ByteSize = 36),
  (FirstParameter = 175, Timestamp = 4141751871, SystemTimer0_Duration = 386660, ByteSize = 33),
  (FirstParameter = 116, Timestamp = 4141825355, SystemTimer0_Duration = 561920, ByteSize = 53),
  (FirstParameter = 25, Timestamp = 4142118861, SystemTimer0_Duration = 1454060, ByteSize = 92),
  (FirstParameter = 109, Timestamp = 4142317238, SystemTimer0_Duration = 63030, ByteSize = 1),
  (FirstParameter = 5, Timestamp = 4142341125, SystemTimer0_Duration = 153360, ByteSize = 5),
  (FirstParameter = 156, Timestamp = 4142411671, SystemTimer0_Duration = 989030, ByteSize = 192),
```

**Figura 3.2.9**

În figura recent atașată, avem informațiile necesare și suficiente despre fiecare bloc în parte. De remarcat este că avem și ordinea exactă în care blocurile de memorie sunt citite de către microcontroller.

Punctăm faptul că prin FirstParameter ne este dat identifier-ul blocului de memorie EEPROM. Timestamp-ul reprezintă momentul în care s-a ajuns la blocul respectiv. SystemTimer0\_Duration reprezintă perioada de timp a blocului cu pricina, după cum am spus și mai sus, măsurată în

nanosecunde. Pentru a avea durată de execuție, folosim modulul Timer0 al microcontrolerului folosit.

Toate informațiile recent ilustrate și dezbătute sunt furnizate de funcția DlsReadByIndexInit().

Țin să semnaliez faptul că acestea nu sunt toate blocurile de memorie EEPROM citite prin această funcție. Sunt arătate doar primele blocuri. În total, avem 163 de blocuri.

Cu privire la cealaltă funcție, EepReadImmediate(), avem aceleași informații. De această dată, sunt alte blocuri.

```

testcodeIMM = (
+ (FirstParameter = 15776, Timestamp = 4140396718, SystemTimer0_Duration = 665300, ByteSize = 40),
+ (FirstParameter = 14648, Timestamp = 4140463405, SystemTimer0_Duration = 626140, ByteSize = 62),
+ (FirstParameter = 14712, Timestamp = 4140526192, SystemTimer0_Duration = 882670, ByteSize = 84),
+ (FirstParameter = 14800, Timestamp = 4140614594, SystemTimer0_Duration = 890240, ByteSize = 84),
+ (FirstParameter = 14888, Timestamp = 4140703753, SystemTimer0_Duration = 889810, ByteSize = 84),
+ (FirstParameter = 14976, Timestamp = 4140792866, SystemTimer0_Duration = 886380, ByteSize = 84),
+ (FirstParameter = 15064, Timestamp = 4140881646, SystemTimer0_Duration = 888750, ByteSize = 84),
+ (FirstParameter = 15152, Timestamp = 4140970650, SystemTimer0_Duration = 890230, ByteSize = 84),
+ (FirstParameter = 15240, Timestamp = 4141059807, SystemTimer0_Duration = 882580, ByteSize = 84),
+ (FirstParameter = 15328, Timestamp = 4141148204, SystemTimer0_Duration = 890510, ByteSize = 84),
+ (FirstParameter = 15728, Timestamp = 4141249507, SystemTimer0_Duration = 137160, ByteSize = 20),
+ (FirstParameter = 15752, Timestamp = 4141263331, SystemTimer0_Duration = 135790, ByteSize = 20),
+ (FirstParameter = 10704, Timestamp = 4141328883, SystemTimer0_Duration = 132450, ByteSize = 2),
+ (FirstParameter = 10720, Timestamp = 4141342258, SystemTimer0_Duration = 127470, ByteSize = 2),
+ (FirstParameter = 10744, Timestamp = 4141964494, SystemTimer0_Duration = 131760, ByteSize = 2),
+ (FirstParameter = 10736, Timestamp = 4142632778, SystemTimer0_Duration = 179100, ByteSize = 1),
+ (FirstParameter = 15912, Timestamp = 4145020050, SystemTimer0_Duration = 182760, ByteSize = 8),
+ (FirstParameter = 15896, Timestamp = 4149515039, SystemTimer0_Duration = 144600, ByteSize = 4),
+ (FirstParameter = 11936, Timestamp = 4149529737, SystemTimer0_Duration = 317540, ByteSize = 26),
+ (FirstParameter = 15952, Timestamp = 4149642695, SystemTimer0_Duration = 152120, ByteSize = 4),
+ (FirstParameter = 0, Timestamp = 0, SystemTimer0_Duration = 0, ByteSize = 0),
+ (FirstParameter = 0, Timestamp = 0, SystemTimer0_Duration = 0, ByteSize = 0),
+ (FirstParameter = 0, Timestamp = 0, SystemTimer0_Duration = 0, ByteSize = 0),

```

**Figura 3.2.10**

După cum se ilustrează și în figură, aici avem un număr de 20 de blocuri. Cu toate aceste informații, o să reușim popularea completă a tabelului dorit.

Prin urmare, tabelul ajunge în următoarea formă, pentru blocurile citite prin intermediul funcției DlsReadByIndexInit():

ID bloc EEPROM	Durata de execuție (ns)	Durata de execuție (ms)	Timestamp (STM0)	Delta		Adresă	Dimensiune
104	2629950	2.62995	4138932017				255
171	193650	0.19365	4139689247	7.5723		0x2AF0	9
14	1230610	1.23061	4139708742	0.19495	EE_ADR_IDX_PAV_MEMOR	0x28C8	124
4	219110	0.21911	4139832450	1.23708			13
161	305020	0.30502	4140277360	4.4491			21
162	254830	0.25483	4140309612	0.32252			17
142	525280	0.52528	4140339327	0.29715			48
24	194920	0.19492	4141360895	10.21568			10
115	624730	0.62473	4141384790	0.23895			61
112	144930	0.14493	4141454150	0.6936			4
121	385880	0.38588	4141468764	0.14614			68
122	90180	0.09018	4141508900	0.40136			8
123	114950	0.11495	4141518046	0.09146			14
.	.	.	.	.			.
.	.	.	.	.			.
.	.	.	.	.			.
.	.	.	.	.			.
.	.	.	.	.			.
.	.	.	.	.			.
Total duration	83021270	83.02127		109.328			

**Figura 3.2.11**

Pentru bloucrile citite prin intermediul funcției EepReadImmediate(), tabelul ajunge în următoarea formă:

ID bloc EEPROM	Durata de execuție (ns)	Durata de execuție (ms)	Timestamp (STM0)
15776	665300	0.6653	4140396718
14648	626140	0.62614	4140463405
14712	882670	0.88267	4140526192
14800	890240	0.89024	4140614594
14888	889810	0.88981	4140703753
14976	886380	0.88638	4140792866
15064	888750	0.88875	4140881646
15152	890230	0.89023	4140970650
15240	882580	0.88258	4141059807
15328	890510	0.89051	4141148204
15728	137160	0.13716	4141249507
15752	135790	0.13579	4141263331
10704	132450	0.13245	4141328883
10720	127470	0.12747	4141342258
10744	131760	0.13176	4141964494
10736	179100	0.1791	4142632778
15912	182760	0.18276	4145020050
15896	144600	0.1446	4149515039
11936	317540	0.31754	4149529737
15952	152120	0.15212	4149642695
<b>Total duration</b>	<b>10033360</b>	<b>10.03336</b>	

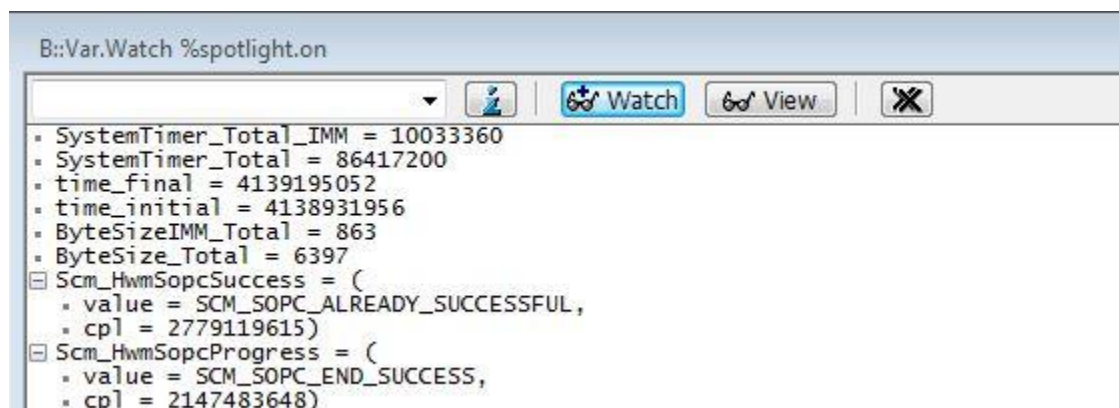
**Figura 3.2.12**

Acestea fiind spuse, avem rezultatele indicate și despre care am mai discutat și anterior.

De toate acestea, ne-am folosit în implementarea finală a soluției și anume:

- Pe core-ul 0, citirea datelor din EEPROM-ul fizic
- Pe core-ul 1, realizarea SOPC-ului și citirea datelor din EEPROM-ul emulat

Totodată, în vederea rezultatelor obținute, sunt deosebit de importante și următoarele informații, pe care le citim cu ajutorul debugger-ului:



**Figura 3.2.13**

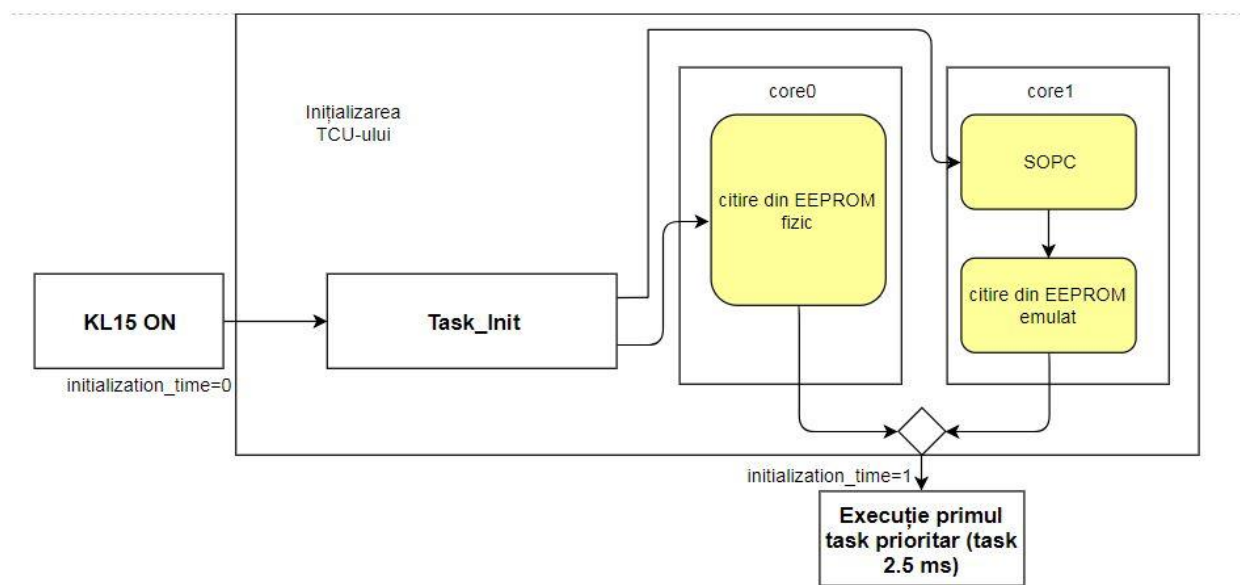


Pentru a obține rezultatele practice și ilustrate, am făcut măsurători folosind tool-ul CANape. Am luat o variabilă și anume `initialization_time` pe care am inițializat-o cu 0.

Cât timp se inițializează TCU-ul, această variabilă nu-și schimbă valoarea. Abia după ce task-ul de init se termină (aici se întâmplă citirea datelor din EEPROM-ul extern și emulat și salvarea lor în oglinda RAM aferentă + realizarea SOPC-ului), poate începe primul task prioritar și anume task-ul de 2.5 milisecunde. În acest moment, variabila `initialization_time` trece pe valoarea 1.

Prin acest procedeu, putem observa timpul în care s-a inițializat TCU-ul.

Schema care arată acest procedeu este:



**Figura 3.2.14**

Astfel, folosind și punând în practică ce am spus mai devreme, o să obțin 2 rezultate conforme cu timpii de inițializare ai TCU-ului.

Prima figură se referă la timpul de inițializare al TCU-ului, timpul în care începe să reacționeze cu celelalte ECU-uri și componente ale mașinii, cu versiunea de soft inițială.

A doua figură reprezintă timpul de reacție al TCU-ului, cu modificările stabilite și gândite. Prin aceste figuri, se compară cele două variante, dar se și deduce timpul îmbunătățit, optimizat.

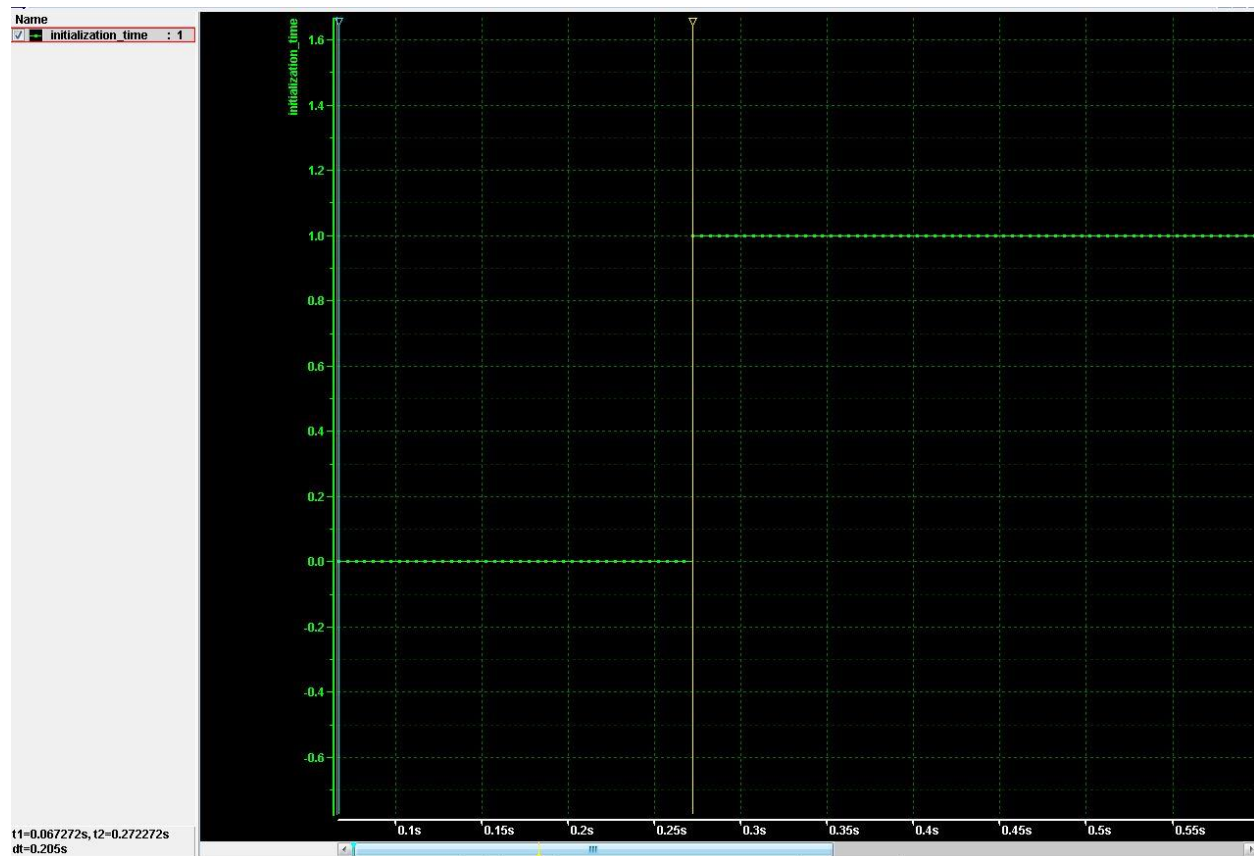


Figura 3.2.15

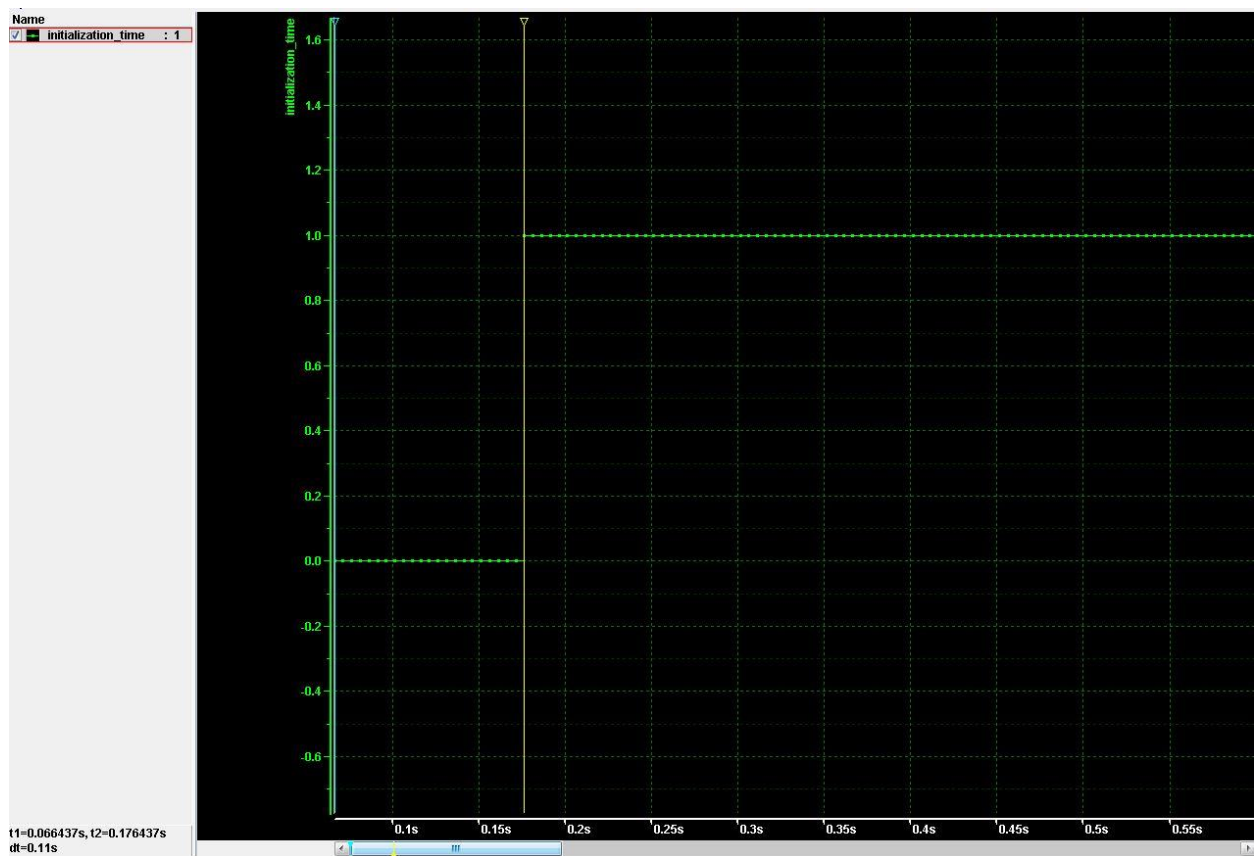


Figura 3.2.16

### 3.3 Unelte folosite în testare

#### CANape

Domeniul principal de aplicare al tool-ului CANape este optimizarea parametrizării (calibrării) unităților de control electronice. În timpul unui proces de măsurare, putem calibra și înregistra simultan semnale. Comunicarea între CANape și ECU se realizează prin intermediul unor protocoale precum XCP sau prin intermediul interfețelor specifice microcontrolerului cu hardware-ul VX1000.

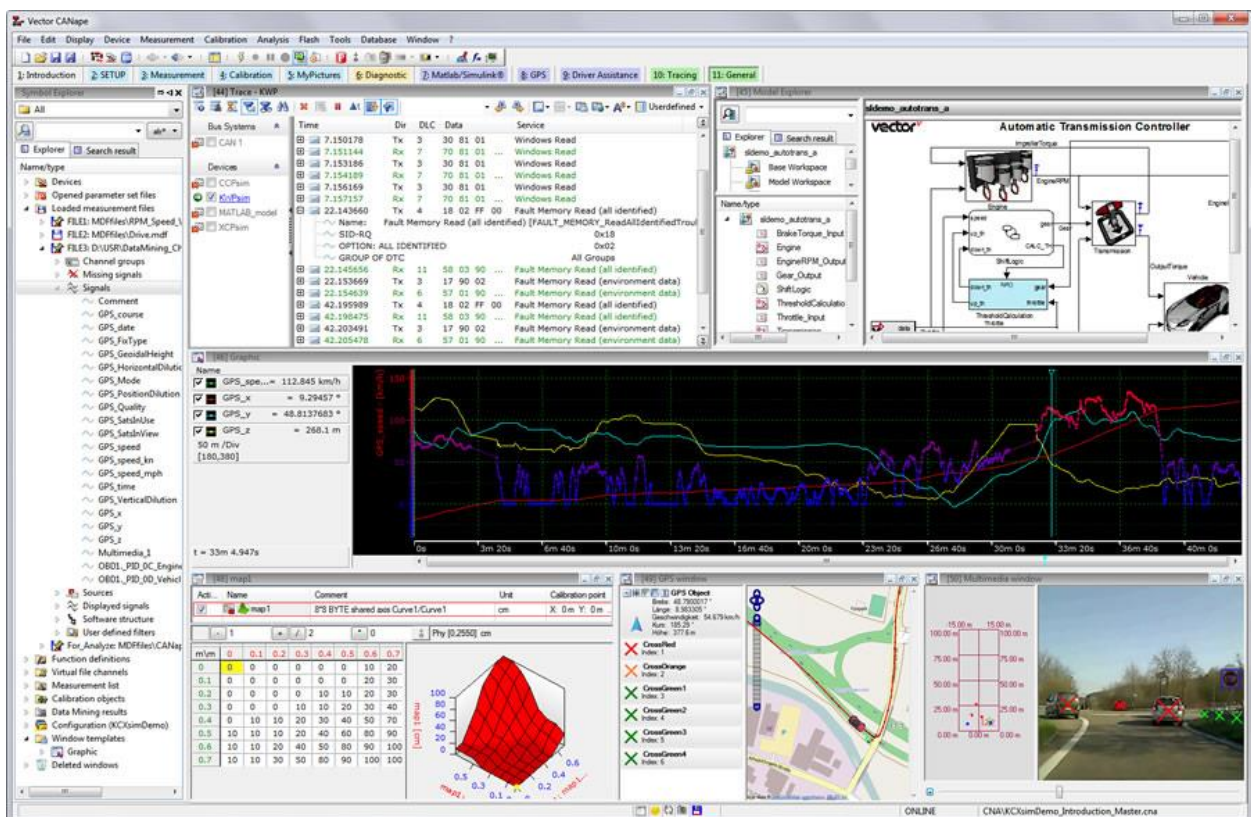
CANape oferă acces de diagnostic, analiză bus și integrarea tehnologiei de măsurare analogică. Gestionarea datelor de calibrare și evaluarea convenabilă a datelor de măsurare, inclusiv raportarea, fac CANape un instrument complet pentru calibrarea ECU. Acesta este un tool modern dezvoltat pentru sistemul de operare Windows care a fost deja folosit cu succes de mulți clienți pentru testele funcționale și diagnoză în timpul programării controller-ului parameter set file (ECU). CANape are un editor integrat, ASAP2, care creează și procesează baza de date a controller-ului, în care toate informațiile pot fi introduse și schimbate în dialoguri. În fișierul ASAP2 este, de asemenea, posibilă actualizarea adreselor și tipurilor de date.

În următoarea fază de dezvoltare CANape-ul ajută dezvoltatorul aplicației să optimizeze algoritmi de control. Pentru a facilita munca inginerilor aplicației, au fost implementate numeroase funcții importante pentru accesul la date, analiză, vizualizarea și îndrumarea operatorului.

1. CANape oferă elemente de control manuale, ușor de accesat , pentru calibrarea parametrilor. Valorile acestor parametrii pot fi introduse sau afișate în format alfanumeric (textual), iar cele cu o structură mai complexă pot fi, de asemenea, selectiv, afișate în diferite tipuri de reprezentări grafice (exp. Reprezentarea unei hărți 3D)
2. Sunt oferite câteva posibilități de vizualizare diferite pentru afișarea datelor măsurate curent. Spre exemplu, valorile măsurate pot fi afișate alfanumeric în fereastră, numeric sau grafic pe ecranul osciloscopului.
3. Datele obținute sunt stocate, dacă este nevoie, pe hard disk, pentru ca mai târziu să poată fi analizate în laboratorul de dezvoltare sau să fie exportate în diferite formate pentru evaluări mai amănunțite cu alte tool-uri. Volumul de date poate fi limitat la rezultatele relevante cu ajutorul condițiilor de declanșare.
4. CANape oferă funcții complete pentru a salva fișierele cu parametrii (economisind

5. Modul în care este concepută interfața grafică a CANape-ului permite utilizatorilor să distribuie elementele de control pe diferite ecrane virtuale. Astfel, utilizatorul poate crea configurații complexe ale ecranului care să corespundă cerințelor problemei.
6. În plus față de interfața controler-ului, CANape permite accesul la controler prin standardul ISO “Keyword Protocol 2000” (KWP2000).
7. Pot fi integrate tehnici de măsurare analogice externe specific industriei automotive și de aceea, în plus față de datele interne de măsurare ale controlerului, pot fi memorate și cantități de date analogice. Compania Vector Informatik GmbH pune la dispoziția noastră propriile module de măsurare analogică pentru tensiune sau temperatură (VS 6xx Module).

Parametrii grafici sau numerici pot fi schimbați direct cu mouse-ul sau tastatura deoarece CANape-ul va transmite automat valorile actualizate la controler.



### Figura 3.3.1



## Trace32

Programarea și depanarea unui sistem integrat pune programatorul în fața unei sarcini dificile, deoarece de cele mai multe ori acesta nu oferă componente atât de utile pentru depanare, cum ar fi o tastatură, monitor, hard disk-uri sau interfețe similare.

Ideea de bază a unui On Circuit Debugger (OCD) sau a unui Back Ground Debug Module este de a realiza un fel de fereastră pentru un sistem embedded. ODC-urile sunt tool-uri care realizează conexiunea dintre un terminal sau un PC cu un sistem embedded, și care facilitează în mare măsură procesul de depanare.

Dezvoltatorul se folosește de acestea pentru a încărca programul în memoria dispozitivului dorit (obiectul în care trebuie încărcat: terminal, PC), executa și apoi pentru a merge pas cu pas prin cod.

În timpul acesta, datele utilizate de program pot fi observate și analizate, astfel putând fii excluse posibile erorii ale software-ului.



**Figura 3.3.2**

Programarea și depanarea obiectivului se face cu In-Circuit-Debugger LA-7708 a companiei Lauterbach.

Tricore este echipat cu un sistem de depanare On-Chip-Debug-System (OCDS), care realizează depanarea unui proces hardware conform descrierii de mai sus. Pe dispozitivul gazdă rulează între timp programul, în cazul nostru Trace32, care într-o direcție comunică cu depanatorul, iar în cealaltă direcție cu OCDS-ul. ICD-ul acționează ca un fel de convertor de protocol pentru a permite comunicarea între componente.

În următoarea figură putem vedea interfața cu utilizatorul a programului Trace32:

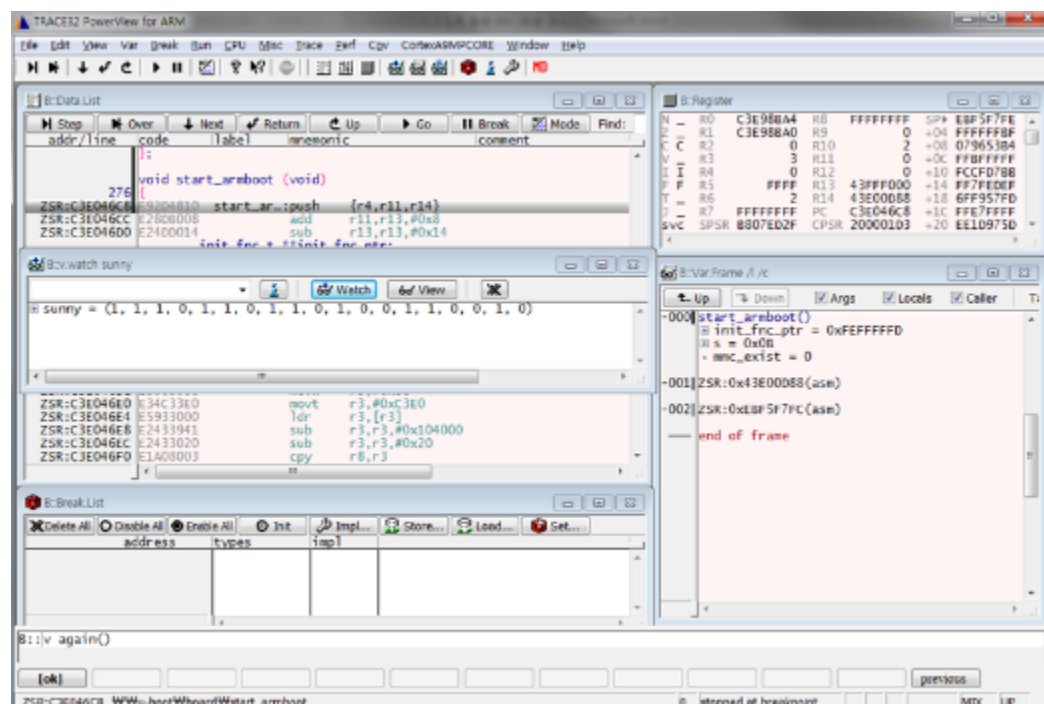


Figura 3.3.3

## CANalyzer

CANalyzer este instrumentul software cuprinzător cu o operare intuitivă pentru analiza și stimularea comunicării prin magistrale. Se utilizează CANalyzer pentru a verifica dacă și ce tip de comunicare are loc pe magistrala.

Acesta poate fi, de asemenea, utilizat pentru a trimite sau înregistra date. Pentru fiecare aplicație oferă funcții de bază puternice pentru începători, precum și funcții mai ample pentru utilizatorii experimentați.

Pe lângă domeniul său principal de aplicare, care este rețeaua electronică în vehicul în industria automobilelor, CANalyzer este de asemenea utilizat în multe alte industrii, cum ar fi transportul feroviar, vehiculele grele, vehiculele speciale, avionica, tehnologia medicală și multe altele. Noile tehnologii bazate pe arhitecturi IP din industria automobilelor sunt susținute de CANalyzer.

În cadrul proiectului nostru, CANalyzer simulează celelalte componente din mașină, pentru ca TCU-ul să comunice pe magistrală.

De punctat este faptul că se poate simula starea unui semnal, prin scriere de cod. Astfel, vom putea folosi un semnal anume cu o valoare care să ne ajute.

## Diagra

Totodată, în cadrul acestui tool, se pot obține informații clare și concrete despre unitate de control respectivă. Aici, exemplificăm:

- Ultima versiune de software flashuită
- Starea de programare -> success sau insucces
- Data la care au fost scrise informațiile
- Versiunea hardware a supplier-ului

## 4. Concluzii

De primă dată când am vorbit despre această posibilă și dorită îmbunătățire, mintea mea a fost foarte deschisă și combativă. Mi-am dorit ca, prin munca depusă de mine, să îmi pun cu adevărat amprenta asupra funcționării mașinilor Audi. Prin ambiție și prin gândul că aş putea să fac ceva cu adevărat de folos, am început treaba. Astfel, pentru început, am încercat să îmi dau seama exact de ceea ce își dorește clientul.

Foarte entuziasmat de toate aceste gânduri, am trecut la treabă.

După primele măsurători și după o implementare naturală, rezultate nu au fost cele scontate. Însă, am continuat documentarea, investigarea, măsurarea și implementarea de soluții posibile.

În cadrul dezvoltării și implementării proiectului meu de diplomă, am întâmpinat destul de multe probleme. Aici aş vrea să nominalizez următoarele:

- Familiarizarea cu microcontrolerul existent, prin citirea documentației
- Descoperirea întregului proces de inițializare al **TCU**-ului
- Documentarea despre **Switch off path check**
- Dobândirea cunoștințelor în materie de EEPROM emulat
- Implementarea interfeței SPI, astfel încât să pot folosi simultan cele 2 core-uri

Având în vedere toate aceste lucruri exprimate anterior, procesul a fost unul destul de lung. Dar, știu cu toții că misiunea unui inginer este de a gândi limpede și corect spre rezolvarea problemelor.

Pășind pe un teren nou, cu multe necunoscute, orice lucru pe care l-am descoperit de la zi la zi a fost doar în beneficiul și în avantajul meu.

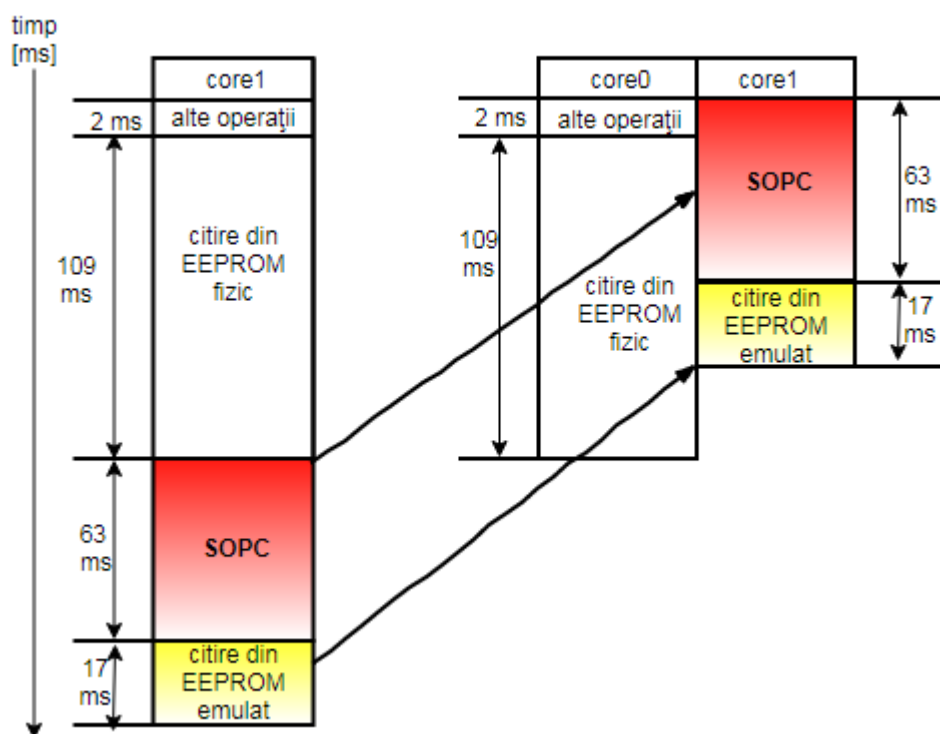
Dacă la început, tentația mea (și probabil a multor altor persoane), a fost să încep împărțirea blocurilor de memorie citite din EEPROM-ul fizic pe cele 2 core-uri stabilite spre folosire, pe parcurs am realizat că nu este o idee prea bună și benefică.

Când am trecut la împărțirea la propriu a blocurilor de memorie, m-am lovit de o problemă foarte mare. Practic, SPI-ul pentru un același job și anume citirea blocurilor de memorie din EEPROM-ul fizic, trebuia să permită citirea blocurilor când pe un core, când pe altul. Prin urmare, intervenea procesul de arbitraj pe magistrala SPI.

Având în vedere că dispunem și de EEPROM fizic, constituit de chipul din cadrul **TCU**-ului, dar și de EEPROM emulat, se desprinde întrebarea de ce? De ce ambele variante, de ce nu doar una?

Din punctul meu de vedere, luând parte la mai multe ședințe și discuții, în viitorul apropiat cred că se urmărește trecerea completă spre EEPROM-ul emulat.

Având în vedere soluția adoptată și rezultatele obținute, sunt de părere că o ușoară împărțire din calupul de date provenite din EEPROM-ul fizic ar genera un timp și mai mic. Însă, cu aceste rezultate, eu mi-am îndeplinit cerințele clientului și am optimizat timpul de inițializare al TCU-ului.



**Figura 4.1**

Direcția de dezvoltare propusă de mine, pentru a îmbunătăți și mai mult timpul necesar inițializării se referă la împărțirea “deficitului” de 30 de milisecunde pe cele două core-uri. Spun acestea deoarece pe core-ul 0, după ce soluția a fost implementată avem aproximativ 111 milisecunde, iar în partea cealaltă undeva spre 80 milisecunde. Printr-un calcul simplu, teoretic aș putea să ajung la un timp de 95.5 milisecunde pe fiecare dintre cele 2 core-uri.

Însă, după cum bine se știe, în practică nu stau lucrurile identic cu teoria. Sunt de părere că ar fi aproape imposibil un echilibru absolut perfect. Însă, cu toate acestea, rămâne o idee bună spre o dezvoltare ulterioară.

Țin să menționez faptul că nu am știut exact la început ce voi face cu adevărat și cât timp o să salvez. Astfel, privind cerințele clientului, care-și dorea o reducere cu cel puțin 50 de milisecunde, am început să gândesc soluții și mai apoi să le implementez.

În final, pot să spun că mi-am îndeplinit ce mi-am propus să realizez în cadrul inițial.

## 5.Bibliografie

### 1.Introducere

- [1]- Ramadasu, T., "Trends in Automotive Remote Diagnosis", 2005
- [2]-\*\*\*, <http://oica.net/category/production-statistics/>
- [3]-\*\*\*, [https://www.infineon.com/dgdl/Infineon-AURIX-TC275T-TC277T-PB-v01\\_00-EN.pdf?fileId=5546d4625696ed7601569796b6891fd1](https://www.infineon.com/dgdl/Infineon-AURIX-TC275T-TC277T-PB-v01_00-EN.pdf?fileId=5546d4625696ed7601569796b6891fd1)
- [4]- lenz14\_02\_01\_Audi\_Schoeffmann\_dt\_1\_19.pdf, documentație internă Continental Automotive SRL
- [5]- TC1784\_DS\_v071.pdf, documentație internă Continental Automotive SRL
- [6]- TC27x\_bare\_ts\_v1.0D2.pdf, documentație internă Continental Automotive SRL
- [7]- [www.silver-atenas.de/en/products/simulators/flexray-gateway](http://www.silver-atenas.de/en/products/simulators/flexray-gateway)
- [8]- [https://vector.com/vi\\_canape\\_en.html](https://vector.com/vi_canape_en.html)

### 2.Proiectare și implementare

- [9]-documentație internă Continental Automotive (cerințele despre cum trebuie analizat și implementat task-ul)
- [10]- <https://www.iso.org/obp/ui/#iso:std:iso:26262:-5:ed-1:v1:en> –Versiunea completă oferită de Continental Automotive S.R.L

### 3.Testare

- [11]- [https://vector.com/portal/medien/cmc/info/CANape\\_ProductInformation\\_EN.pdf](https://vector.com/portal/medien/cmc/info/CANape_ProductInformation_EN.pdf)
- [12]- [https://vector.com/vi\\_canalyzer\\_en.html](https://vector.com/vi_canalyzer_en.html)
- [13]- <https://www.rac.de/en/automotive-products/software/diagnostics/diagra-d/>



## **Lista figurilor**

- 1.1.1-Componentele electronice ale mașinii
- 1.2.1-Sistemele unui automobil
- 1.3.1-Localizarea TCU-ului într-o mașină
- 1.3.2-Comunicarea TCU-ului cu celelalte componente
- 1.4.1-Tracțiune spate
- 1.4.2-Tracțiune față
- 1.4.3-Tracțiune integrală
- 1.4.4-Ambreiaj
- 1.4.5-Raport de transmisie
- 1.4.6-Cutie cu variație continuă
- 1.4.7-Cutie automată
- 1.4.8-DCT (Double Clutch Transmission)
- 1.5.1-Magistrala CAN
- 1.6.1-Schema logică a microcontrolerului TC1784
- 1.6.2-Schema bloc a microcontrolerului TC1784
- 1.7.1-Schema logică a microcontrolerului Aurix TC277
- 1.7.2-Schema bloc a microcontrolerului Aurix TC277
- 1.8.1-Interfața SPI
- 2.1-Citire EEPROM fizic
- 2.2-Soluție pentru citirea din EEPROM fizic
- 2.3-Salvarea datelor din EEPROM în oglinda RAM
- 2.4-Inițializarea actuală a TCU-ului
- 2.5-Soluție de îmbunătățire a timpului
- 3.1.1-Schemă testbench
- 3.2.1-Flashuirea pe CAN/FlexRay
- 3.2.2-Selecția informațiilor despre TCU
- 3.2.3-Vizualizarea informațiilor despre TCU

- 3.2.4-Comunicația cu ECU-urile
- 3.2.5-Caracteristici de flashuire TCU
- 3.2.6-Opțiuni de flashuire
- 3.2.7-Procesul de flashuire
- 3.2.8-Structură tabel
- 3.2.9-Informații provenite din DlsReadByIndexInit
- 3.2.10-Informații provenite din EepReadImediate
- 3.2.11-Tabel de informații DlsReadByIndexInit
- 3.2.12-Tabel de informații EepReadImediate
- 3.2.13-Informații debugger
- 3.2.14-Schema procesului de inițializare al TCU-ului
- 3.2.15-Rezultate cu soluția inițială
- 3.2.16-Rezultate cu soluția implementată
- 3.3.1-Interfață CANape
- 3.3.2-Debugger Trace32
- 3.3.3-Interfață Trace32
- 3.3.4-Interfață CANalyzer
- 4.1-Soluție de îmbunătățire a timpului

## **Lista prescurtărilor**

TCU-Transmission Control Unit

EEPROM-Electrically Erasable Programmable Read-Only Memory

RAM-Random Access Memory

SOPC-Switch Off Path Check

SPI-Serial Peripheral Interface