**Introducere**

**Context**

Într-o lume aflată într-un proces de evoluţie continuu, cu toţii suntem conştienţi că principalul mijloc de deplasare este automobilul. Fiind un element esenţial al vieţii 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 soluţii, cât şi testarea lor, reprezintă elemente demne de o atenţie sporită, deoarece chiar şi o greşeală minoră poate fi fatală.

**Situaţie actuală**

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 distribuţie multicore care foloseşte o unitate de control a transmisiei, **TCU**, pentru a manipula cutia de viteze a maşinilor 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 maşinilor Audi.

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

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

**Temă propriu-zisă**

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

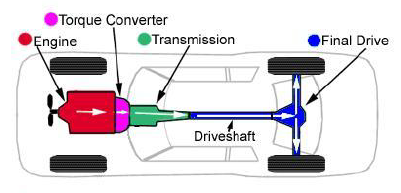
Ţinând seama de faptul că microcontrolerul “primeşte” informaţii de la chipul memoriei EEPROM prin intermediul interfeţei SPI, rezultă că funcţionalitatea interfeţei trebuie “transferată“ şi pe al-2lea core ce urmează a fi utilizat (pe core-ul 1, utilizat iniţial, interfaţa a fost implementată).

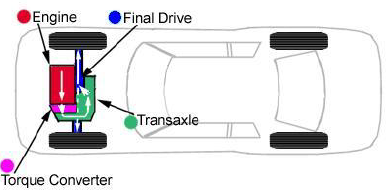
**Prezentarea sistemului de transmisie a puterii**

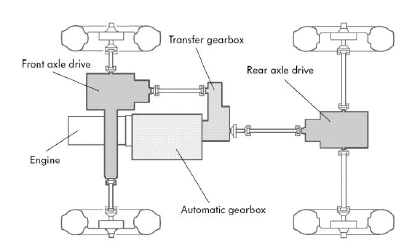
* **Sistemul de tracțiune**

Sistemul de tracţiune reprezintă ansamblul componentelor ce fac posibilă aducerea şi menţinerea unui automobil în stare de mişcare.

În funcţie de tipul automobilului, desprindem mai multe categorii de tracţiune:

* Tracţiune spate, întâlnită cu precădere la maşinile sport (Lamborghini, Mercedes , BMW, Ferrari)
* Tracţiune faţă, întâlnită la majoritatea maşinilor mici (Dacia, Renault, Ford, Chevrolet, VW, Opel)



* Tracţiune integrală, 4x4, întâlnită la SUV-uri (BMW X5, BMW X6, Audi Q7, VW Touareg)
* **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. Activarea ambreiajului se poate face atât în timp ce acesta se rotește sau în timp ce stă neînvârtit.



* **Cutia de viteze**

Cutia de viteze este un ansamblu de roți dințate care servește la transformarea forței și transmiterea mișcării de rotație la rotile 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:

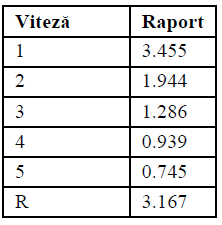
1. Raportul de transmisie:

* cu raport de transmisie fix
* cu raport de transmisie variabil (CVT)

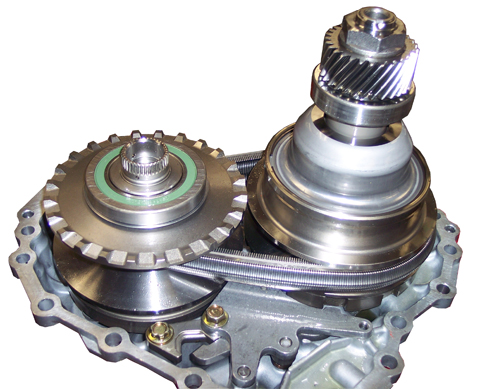
1. Gradul de automatizare:

* manuală
* CVT (Continuously Variable Transmission)
* Automată

Ȋn 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:

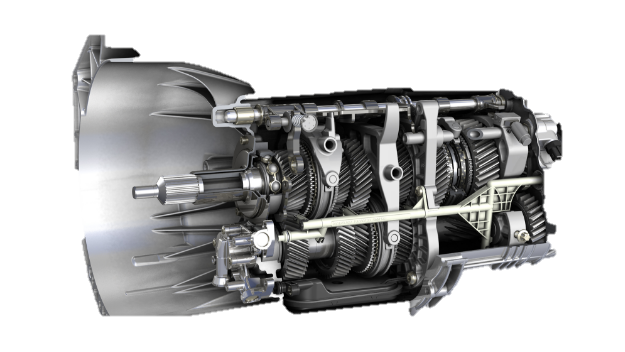


**Cutiile de viteze CVT (Continuously Variable Transmission)**, spre deosebire de cutiile manuale, au capacitatea de a parcurge un număr infinit de rapoarte de transmisie. CVT oferă eficiență sporită în ceea ce privește consumul de carburant, deoarece îi permite motorului să funcționeze la cea mai eficientă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



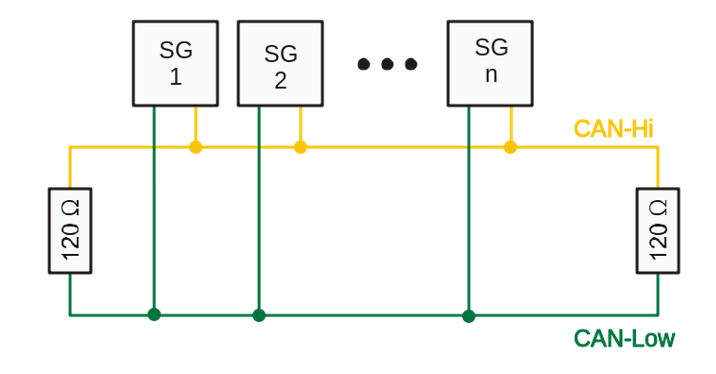
**Cutiile de viteze automate** sunt cutiile care realizează schimbarea treptelor de viteză fără intervenția conducătorului automobilului. Mai mult, decizia de schimbare a treptelor de viteză este luată de asemenea automat, pe baza informațiilor provenite de la diferiți senzori. Realizarea unei trepte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



**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. Ȋn 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.

**Moduri de comunicaţie ale TCU-ului**

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

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

* Modul autonom, poate organiza singur transmisia și recepția datelor conform specificațiilor
* Chipul modului de CAN trimite si 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 120Ω, 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-uiește microcontroller-ul cu programul dorit

1. **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 in producția seriei de vehicule BMW X5
* Utilizarea completă a fost introdusă în 2008 la noul BMW Seria 7
* Pe langă vehiculele amintite anterior, a mai fost folosit la producerea:
  + - Audi A4
    - Audi A6
    - Audi A7
    - Audi A8
    - Lamborghini Huracan
    - Mercedes-Benz S-Class

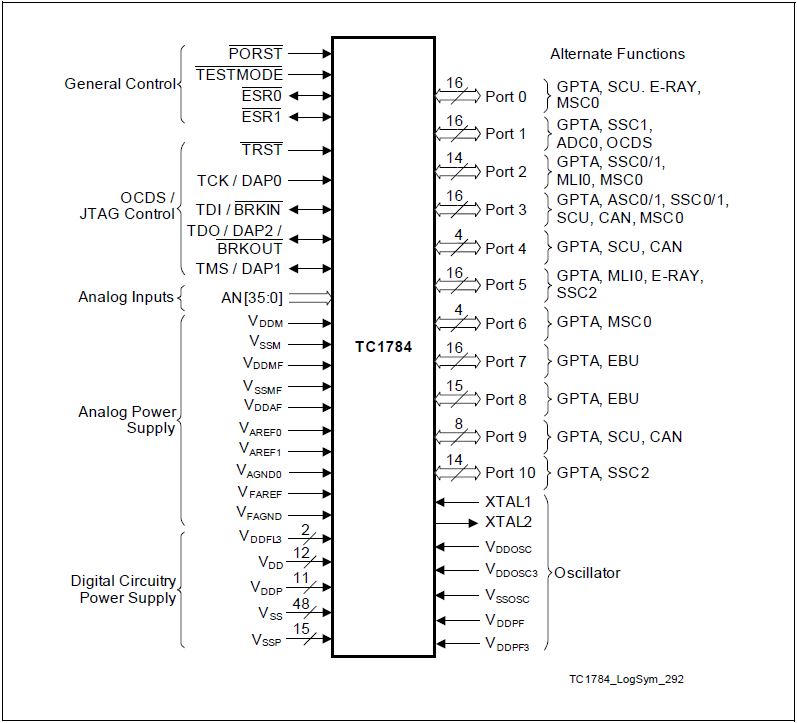
Interfața pentru rețeaua FlexRay, **VN7600**, este ideală pentru dezvoltarea, simularea sau testarea rețelelor FlexRay. Posibilitățile de conectare prin USB fac ca aceasta să fie aplicabilă în mai multe locuri diferite (aici putem aminti ca locuri aplicabile laboratorul sau chiar pe durata unui test-drive).

În plus, interfețele oferă acces la CAN.

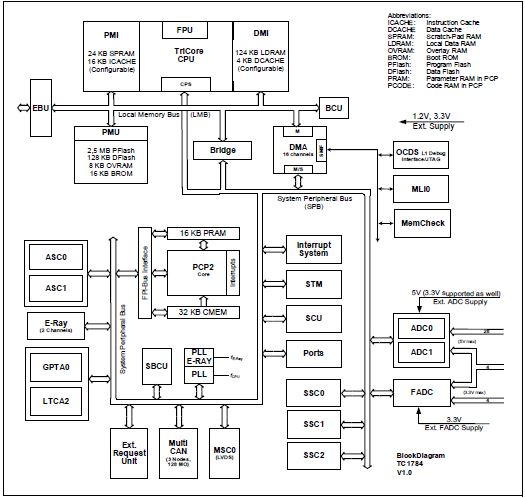


În cadrul proiectului desfășurat de către compania Continental, au existat 2 generații:

1. **Generația 1**, după multe căutari 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

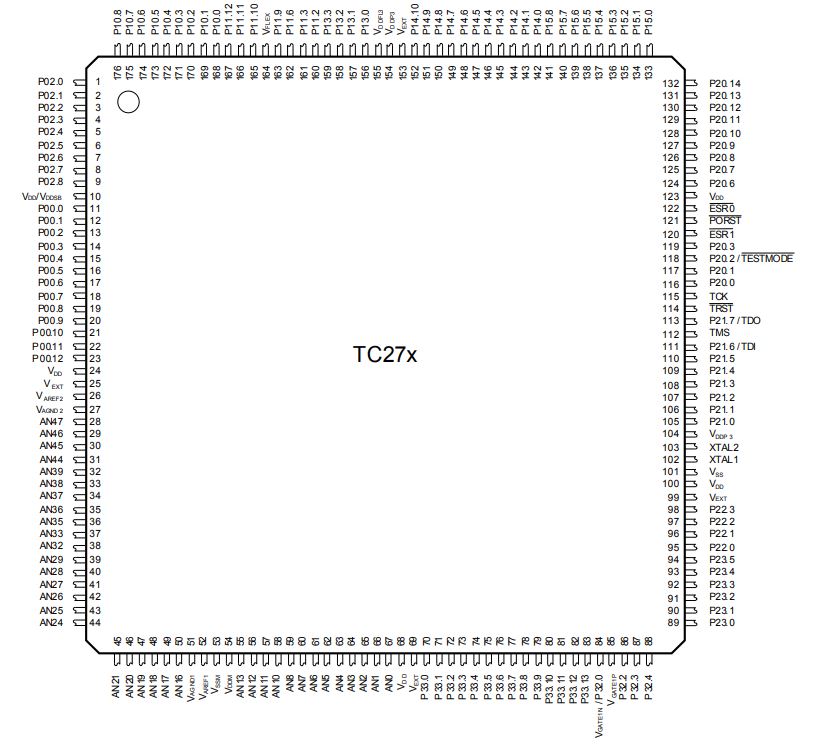


În continuare, o să ilustrez schema bloc a microcontroller-ului Infineon TC1784:

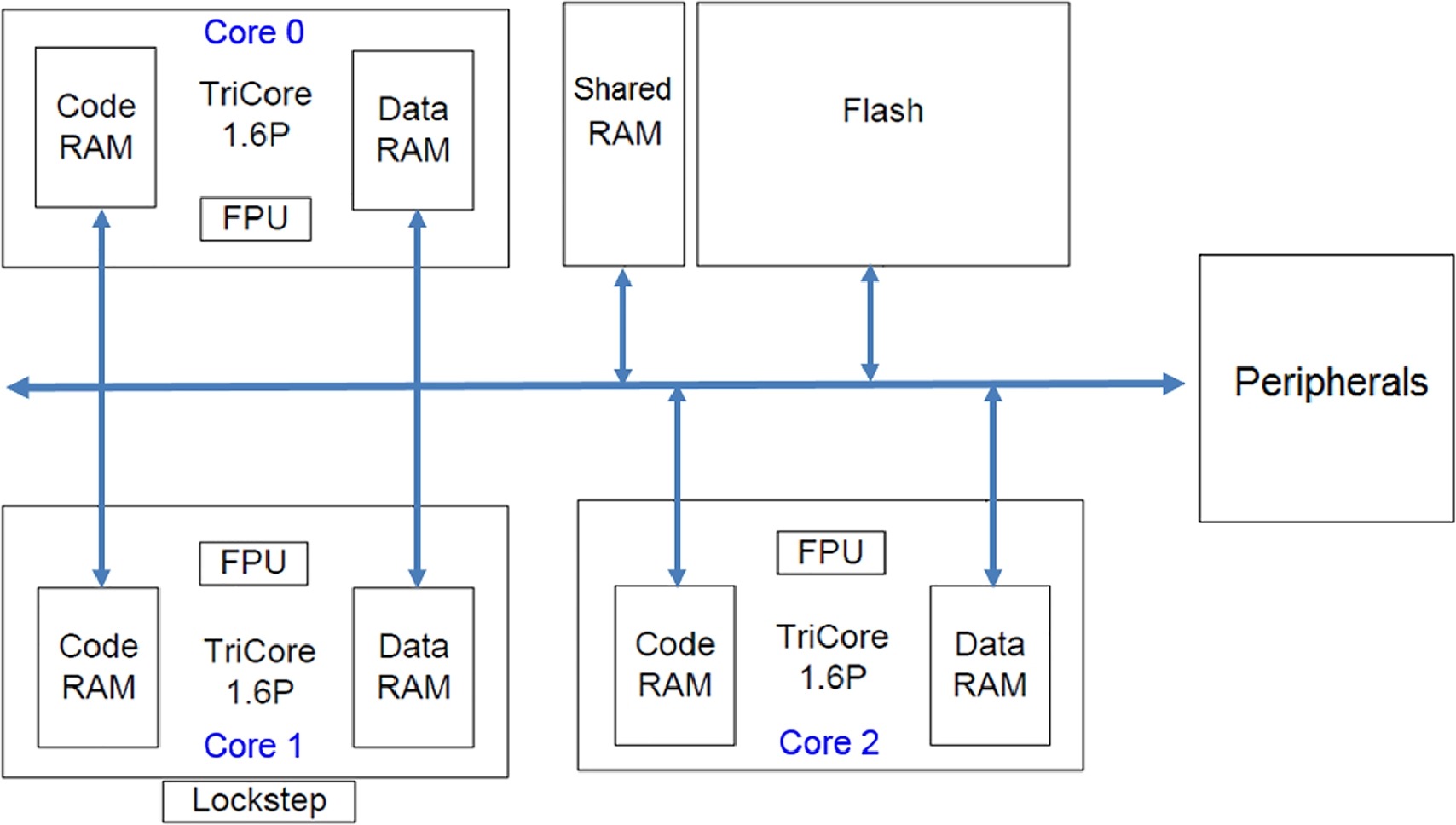


* Din punct de vedere al construcției acestuia, desprindem următoarele caracteristici:
  + - Frecvența maximă a CPU-ului: 180 MHz
    - Frecvența maximă a PCP-ului: 180 MHz
    - Frecvența maximă a sistemului: 90MHz
* Structural și privind CPU-ul, microcontroller-ul TC1784 include un CPU de mare performanță 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. **Generația 2**, a folosit un alt microcontroller de la Infineon și anume Aurix TC277:



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



* 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 legatură mecanică între maneta schimbătorului de viteze și transmsie.

În mod tradițional, prin metoda **SBC,** schimbarea transmisiei a fost realizată prin legaturi mecanice pentru a pune vehiculul in 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.

**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.

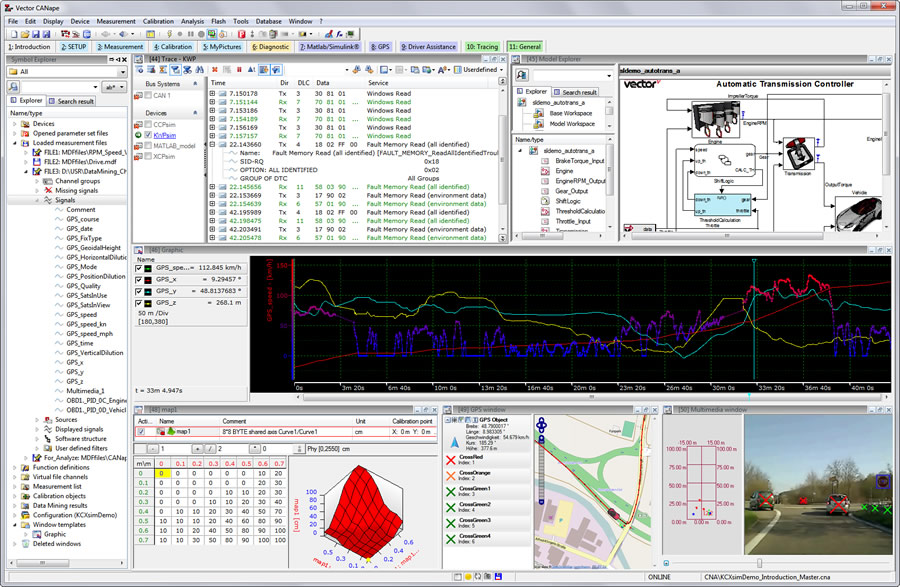
CANape oferă acces de diagnostic, analiză bus și integrarea tehnologiei de măsurare analogică. Gestionarea datelor de calibrare și evaluarea convenabilă a datelor de măsurare, inclusiv raportarea, fac CANape un instrument complet pentru calibrarea ECU. Acesta este un tool modern dezvoltat pentru sistemul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

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

1. CANape oferă elemente de control manuale, uşor de accesat , pentru calibrarea parametrilor. Valorile acestor parametrii pot fi introduse sau afişate în format alfanumeric (textual), iar cele cu o structură mai complexă pot fi, de asemenea, selectiv, afişate în diferite tipuri de reprezentări grafice (exp. Reprezentarea unei hărţi 3D)
2. Sunt oferite câteva posibilităţi de vizualizare diferite pentru afişarea datelor măsurate curent. Spre exemplu, valorile măsurate pot fi afişate alfanumeric în fereastră, numeric sau grafic pe ecranul osciloscopului.
3. Datele obţinute sunt stocate, daca 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 amănunţite cu alte tool-uri. Volumul de date poate fi limitat la rezultatele relevante cu ajutorul condiţiilor de declanşare.
4. CANape oferă funcţii complete pentru a salva fişierele cu parametrii (economisind hard disk-ul), să reactiveze mulţimile de parametrii salvate în controler sau să compare fişirele cu parametrii între ele;
5. Modul în care este concepută interfaţa grafică a CANape-ului permite utilizatorilor să distribuie elementele de control pe diferite ecrane virtuale. Astfel, utilizatorul poate creea configuraţii complexe ale ecranului care să corespundă cerinţelor problemei.
6. În plus faţă de interfaţa controler-ului, CANape permite accesul la controler prin standardul ISO “Keyward Protocol 2000” (KWP2000).
7. Pot fi integrate tehnici de măsurare analogice externe specific industriei automotive şi de aceea, in plus faţă de datele interne de măsurare ale controlerului, pot fi memorate şi cantităţi de date analogice. Compania Vector Informatik GmbH pune la dispoziţia noastră propriile module de măsurare analogică pentru tensiune sau temperatură (VS 6xx Module).

Comunicarea dintre CANape şi controler poate fi vizualizată în fereastra de Trace la nivelul mesajului pe CAN. Limbajul de programare existent în CANape include câteva funcţii şi macrouri specifice diagnozei astfel încât utilizatorul poate programa proceduri complexe de testare a diagnozei.

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



**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 masură procesul de depanare.

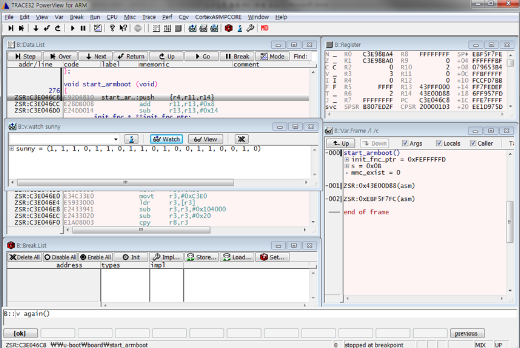
Dezvoltatorul se folosește de acestea pentru a încărca programul în memoria dispozitivului dorit (obiectul în care trebuie încărcat: terminal, PC), executa și apoi pentru a merge pas cu pas prin cod.

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

Programarea și depanarea obiectivului se face cu In-Circuit-Debuger 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 interfaţa cu utilizatorul a programului Trace32:



**Proiectarea și implementarea aplicației**

**Structura sistemului**

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

1. Sursa de tensiune TOE8952
2. VN 7600
3. **TCU**, unitatea de control a transmisiei
4. Trace32, debugger
5. Cablu

**Aplicația propriu-zisă**

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 reacționeză 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 benific 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
* un timp inițial, care arată momentul în care s-a început citirea unui bloc anume
* un timp final, care ilustrează momentul în care citirea blocului respectiv s-a încheiat
* 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.

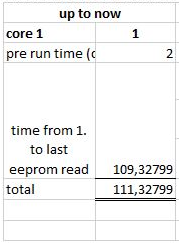
Așadar, pentru început, mi-am dorit o îmbunătățire severă a timpului de inițializare al TCU-ului, fără a “transfera” funcționalitățile interfeței SPI și pe un core ce urma a fi folosit. Deoarece SPI-ul trebuie să fie prezent doar pe core-ul care citește propriu-zis din EEPROM, am dezvoltat o primă soluție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

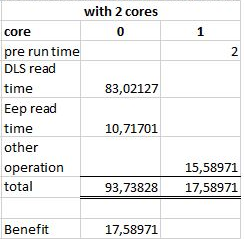
În aceste condiții, descrise anterior, se evita prezența simultană a functionalităților interfeței SPI pe mai multe core-uri.

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 aşteptat, nici pe departe, am început investigaţia şi astfel am măsurat cât durează doar citirea din EEPROM-ul fizic, pentru a putea dezvolta în continuare soluţii viabile.

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

* citirea “brută“ a datelor din EEPROM-ul fizic pe core-ul iniţial folosit
* citirea datelor din EEPROM-ul fizic mutate pe core-ul 0 şi lăsarea operaţiilor adiţionale pe core-ul 1, folosit iniţial



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

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

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

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

**Switch off path check-ul (SOPC)** este un element esenţial în iniţializarea TCU-ului, este cel care determină starea valvelor şi a tuturor conexiunilor pentru a se asigura că totul rulează la parametri optimi înainte de pornire. Este cel responsabil cu oprirea sistemului în cazul detecţiei 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 1262.

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 Cy. Dacă vin prea multe răspunsuri greșite, Cy va transmite starea de siguranță. Dacă situația de greșeală 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 Cy
* după ce s-au efectuat verificările că se poate intra în safe state, se resetează FS0B și se face self test dacă BLDC răspunde la schimbarea FS0B

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 măsurarile aferente, **SOPC-ul** durează undeva la 60 milisecunde.

Celălalt timp rămas se datorează EEPROM-ului emulat.

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

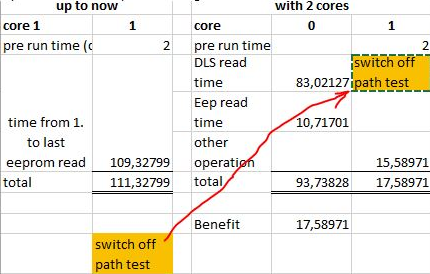
Sesizând faptul că citirea datelor din chipul de EEPROM dureză undeva la 111 milisecunde, mi-am dat seama că dacă aş reuşi să pun în “paralel”, pe de-o parte citirea din EEPROM-ul fizic, iar de cealaltă parte citirea din EEPROM-ul emulat alături de SOPC, toata acţiunea ar tinde spre un oarecare echilibru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

În timp ce pe core-ul 0 se începe citirea clasică a blocurilor de memorie din EEPROM-ul fizic, în acelaşi timp să înceapă şi 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 iniţializare a TCU-ului, ţin să precizez că în toate aplicaţiile şi proiectele din domeniul IT, se urmăreşte dezvoltarea cât mai ”curată” şi eficientă a paralelismului. Aici, avem câteva exemple: Thread-uri, Pipe-uri etc. Ştim foarte bine că, deşi se spune despre solutiile amintite anterior ca se executa in paralel, nu este vorba despre un paralelism “curat”.

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

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

Apoi, implementând în totalite ideea enunţată, am ajuns la următoarele rezultate:

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

Chestiile de punctat ar fi că EEPROM-ul emulat nu necesită comunicaţie cu interfaţa SPI.

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

* Cy328, pentru core1 respectiv SOPC şi EEPROM emulat
* SCC, pentru core0 respectiv citire din EEPROM fizic

De completat.....

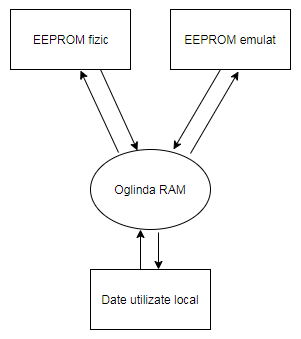
Dacă îmi îndrept atenţia spre citirile din EEPROM, trebuie să precizez că la pornire se cer date de la ultima funcţionare a maşinii. În EEPROM rămân date salvate care se colectează:

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

Un element important în toată implementarea îl reprezintă oglinda RAM. Oglinda RAM este o stare de tranziţie a informaţiilor necesare iniţializării TCU-ului.

Astfel, imediat după acţionarea cheii de către utilizator, microcontrolerul primeşte informaţii din EEPROM-ul fizic şi din EEPROM-ul emulat şi le salvează în oglinda RAM. Acolo, datele pot suferi modificări chiar în timpul funcţionării. Mai departe, toate datele din oglinda RAM sunt utilizate local.

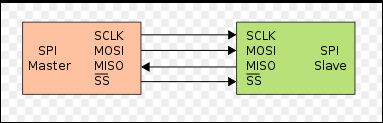
După oprirea maşinii, KL15 => OFF, datele din RAM se pierd. Rămân salvate doar în EEPROM. De acolo vor fi cerute din nou pentru iniţializarea TCU-ului, la următoarea rulare a maşinii.

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

Microcontrolerul comunică cu chipul de EEPROM prin intermediul interfeței SPI.

Interfața seriala SPI (Serial Peripheral Interface) este o interfața 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)

Pentru a începe comunicarea, master-ul mai întâi configurează ceasul, folosind o frecvență mai mică sau egală cu maximul frecvenței suportata 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 perioada 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.

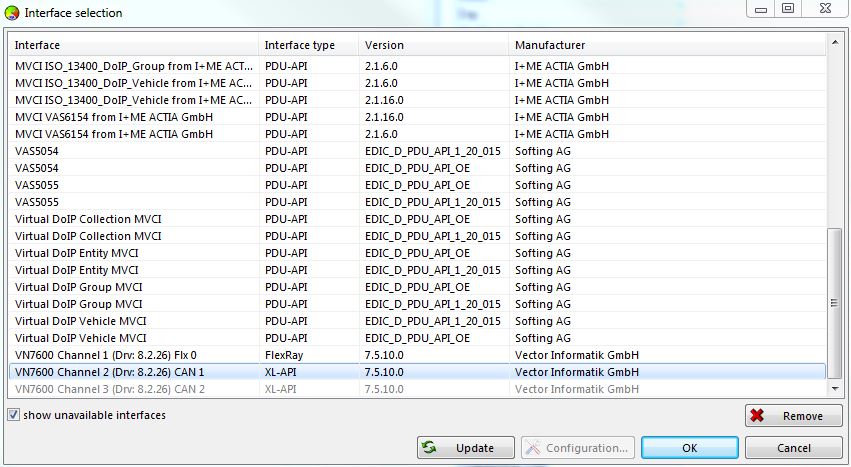
**Testarea aplicației**

**Unelte folosite în testare**

**Diagra**

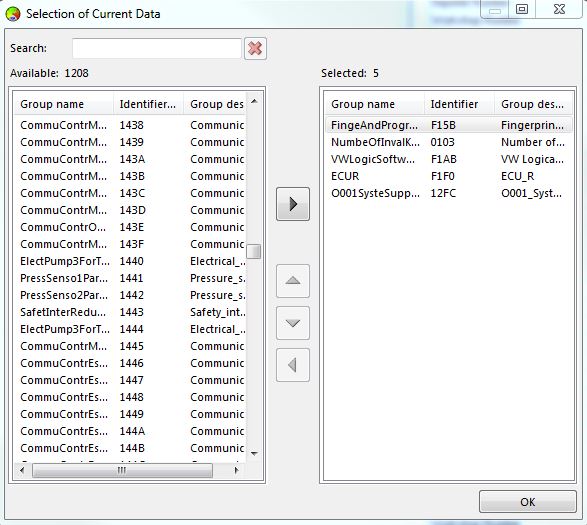
După ce am realizat codul cu modificările dorite, următorul pas pe care trebuie să-l facem e să încărcăm (să flashuim) acel cod pe TCU, unitatea de control a transmisiei. Pentru a realiza acest lucru, ne folosim de o unealtă des utilizată în cadrul departamentului nostru și anume Diagra.

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

* Controller Area Network (CAN)
* FlexRay

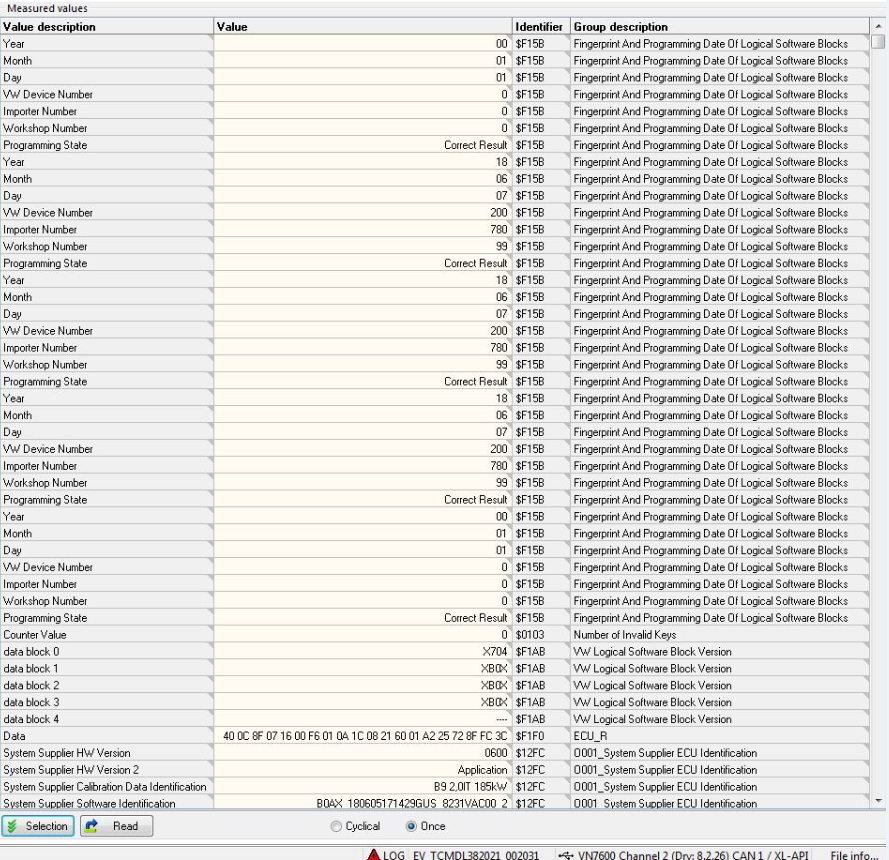
În continuare, fiecare funcție a TCU-ului are un anumit identifier. Prin meniul de selecție al funcțiilor, putem să alegem spre vizualizare oricare proprietate pe care ne-o dorim.

După căutarea si găsirea după identifier 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.



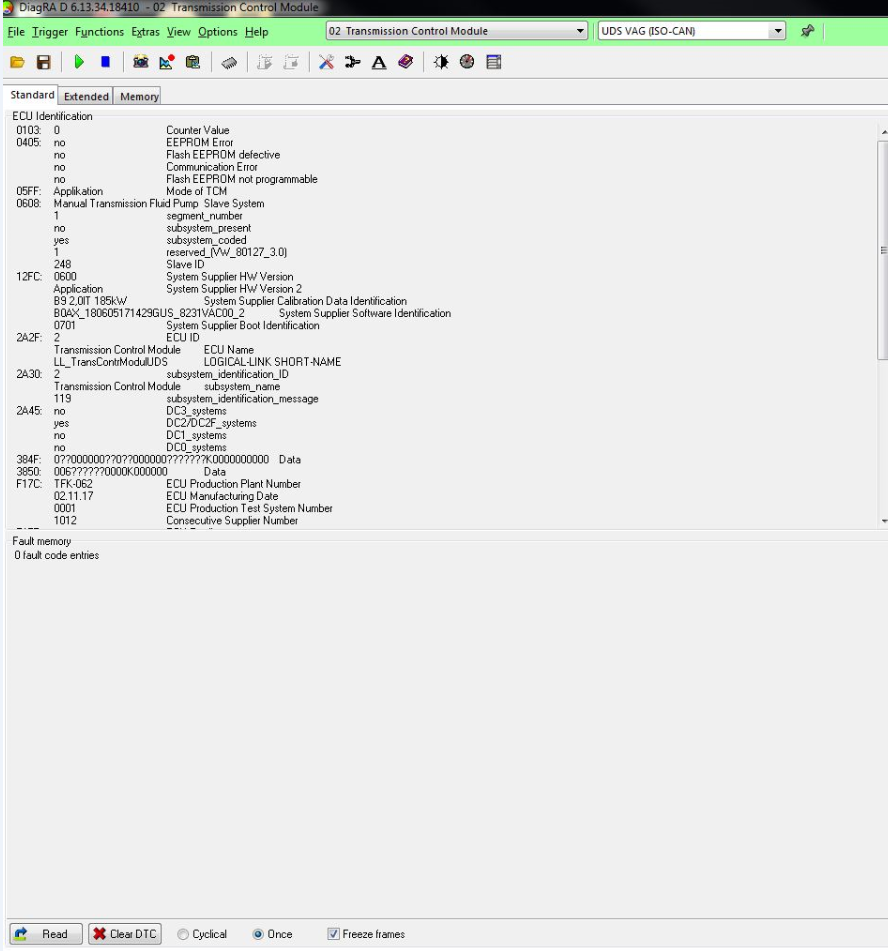
După ce am selectat tot ce ne-am dorit, apăsăm butonul OK și revenim la meniul principal. În partea dreaptă o să observăm informațiile grupate pe mai multe coloane, astfel:

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



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.

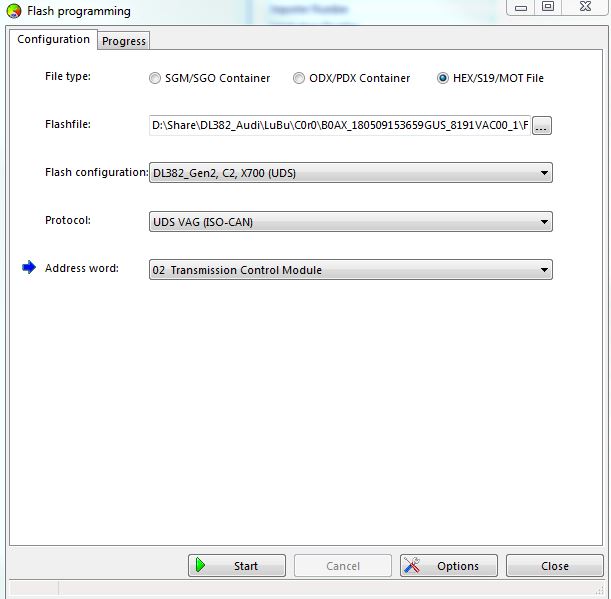


Încărcarea valorilor se face prin apăsarea butonului READ, iar citirea valorilor se poate face o singura dată sau chiar ciclic, la fiecare interval de timp.

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

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

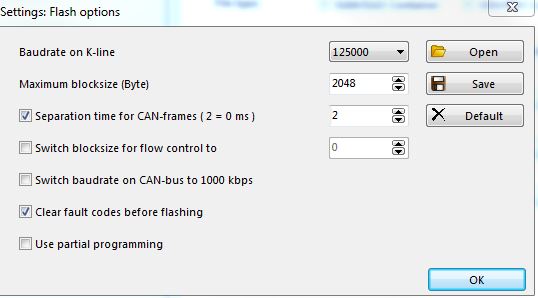
La opțiunile de flashuire se încarcă fișierul cu extensia .hex, ce a fost generat după compilarea codului scris în limbajul de programare C.

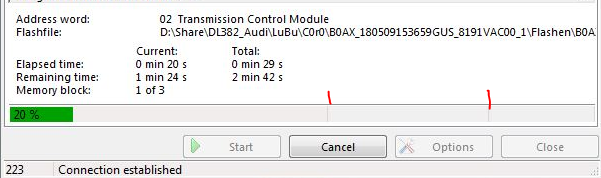


După ce am ales fișierul cu extensia .hex dorit, ne rămâne să alegem configurația flashuirii, protocolul și address word-ul.

După ce caracteristicile anterioare au fost stabilitie 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.

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

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.

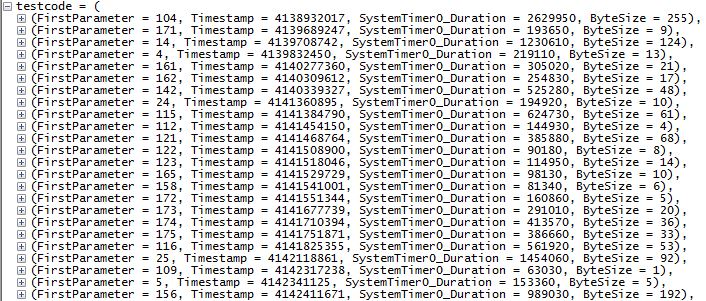
Din acest moment, putem începe vizualizare informațiilor de care avem nevoie. Ne vom folosi de tool-ul Trace32, în care a fost încărcat un fișier cu extensia .elf generat la compilarea codului pentru a putea urmări toate variabilele și structurile utilizate.

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 timpestamp-uri și transformarea în milisecunde a diferenței
* Adresa blocului
* Dimensiunea blocului

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.



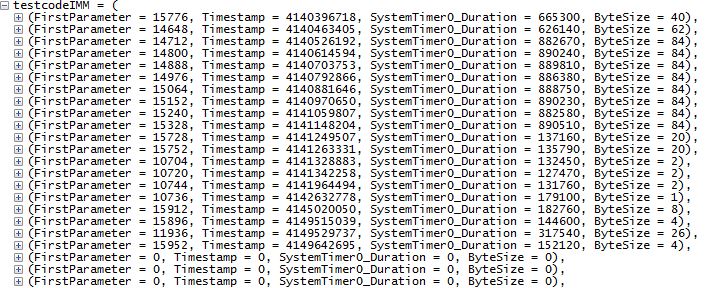
În figura recent atașată, avem informațiile necesare și suficiente despre fiecare bloc în parte. De remarcat este că avem și ordinea exactă în care blocurile de memorie sunt citite de către microcontroler.

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, măsurata în nanosecunde. Pentru a avea durata de execuție, folosim modulul Timer0 al microcontrolerului folosit.

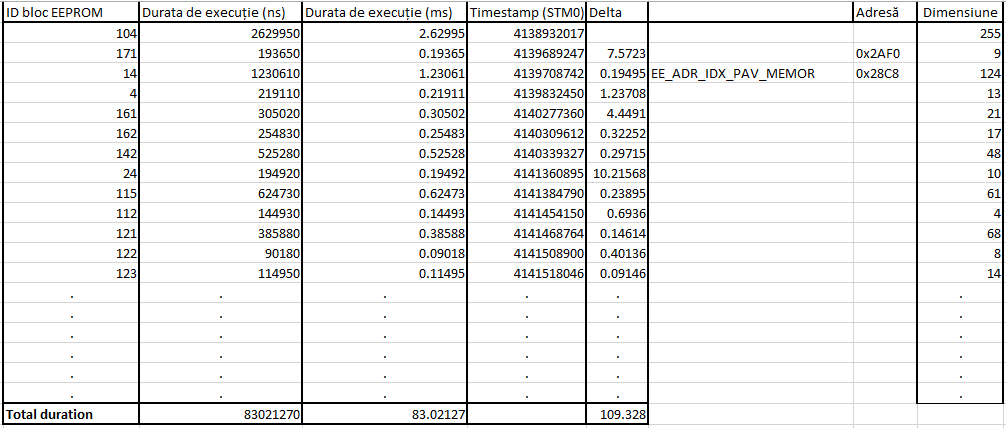
Toate informațiile recent ilustrate și dezbătute sunt furnizate de funcția DlsReadByIndexInit().

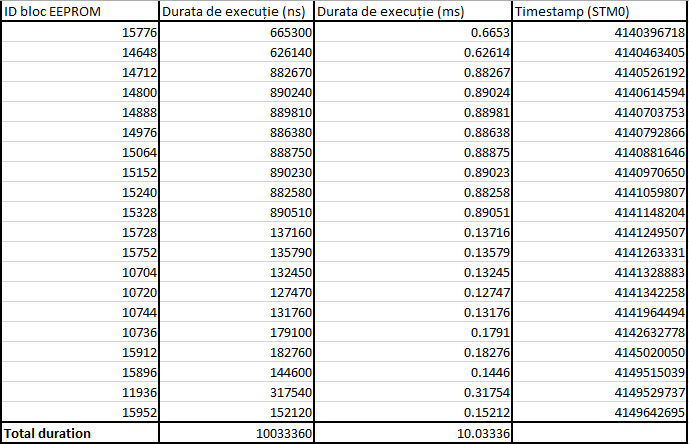
Țin să semnalez faptul că acestea nu sunt toate blocurile de memorie EEPROM citite prin această funcție. Sunt arătate doar primele blocuri. În total, avem 163 de blocuri.

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



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():

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