

VILNIAUS UNIVERSITETAS  
MATEMATIKOS IR INFORMATIKOS FAKULTETAS  
PROGRAMŲ SISTEMŲ KATEDRA

**Praktikos uždarojoje akcinėje bendrovėje  
„Elektromotus“ ataskaita**

Atliko: 4 kurso 1 grupės studentas  
Rytis Karpuška .....  
(parašas)

Darbo vadovas:  
Irus Grinis, lekt.....  
(parašas)

Vilnius  
2014

# Ivadas

# 1. Apie UAB „Elektromotus“

UAB „Elektromotus“ 2010 metais įkurta trijų steigėjų: Gintauto Palucko, Mindaugo Milašausko ir Šarūno Šutavičiaus su tikslu realizuoti jų sukaupią patirtį elektromobilių valdymo sistemų kūrimo srityje. Nuo to laiko, įmonė išaugo ir samdyti darbuotojai pilnai perėmė įmonės darbo krūvius. Šiuo metu įmonėje yra per 15 darbuotojų, metinė apyvarta viršija milijoną litų, bei vykdomas ne vienas naujas projektas, su tikslu, patobulinti, išleisti, sukurti naujas technologijas bei produktus.

Buvo sukurti ir išleisti keli produktai, bei patentai.

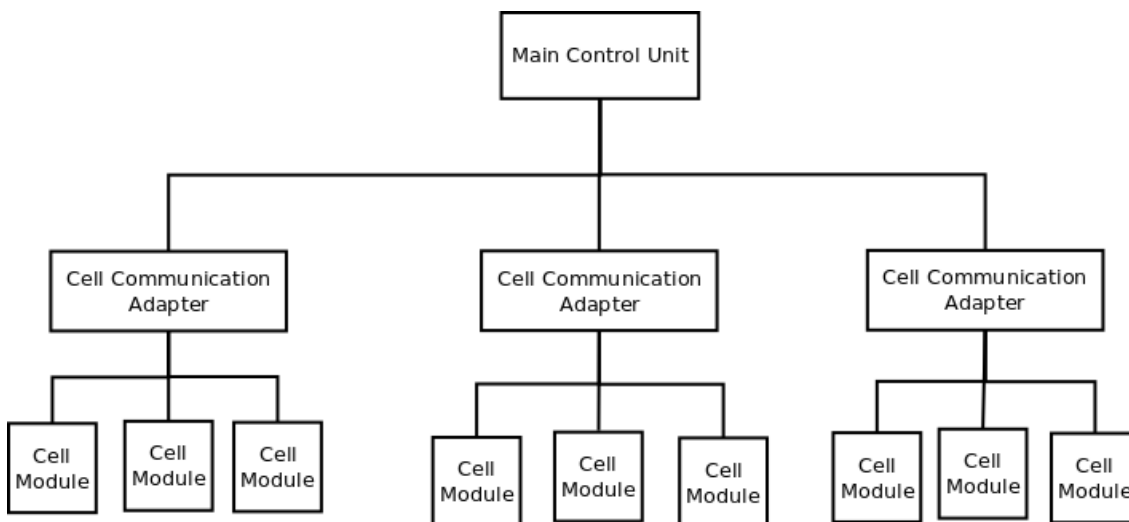
## 1.1. Produktai

### 1.1.1. „Emus BMS“ - Baterijų valdymo sistema

„Emus BMS“ yra pagrindinis ir seniausias įmonės produktas, įnešantis daugiausiai pajamų.

Tai yra valdymo sistema vidutinėms ir didelėms ličio jonų, ličio polimerų, ličio geležies fosfatų, nikelio-metalo hidratų ir kitoms baterijoms.

Sistemą sudaro trijų lygmenų hierarchinė struktūra:



1 pav. Emus BMS struktūra.

Sistemą sudaro šie komponentai:

- Main Control Unit - Pagrindinis valdymo blokas, komunikuoja su komunikacijos adapteriais CAN sąsaja.

- Cell Communication adapter - Adapteris persiunčiantis CAN sąsajos komandas į celių modulius privačia įmonės sukurta sąsaja pritaikyta dirbti EMI triukšmingose sąlygose.
- Cell module - Celių moduliai, galiniai įrenginiai, matuoja celės įtampą, temperatūrą, turi šiluminio balansavimo palaikymą.

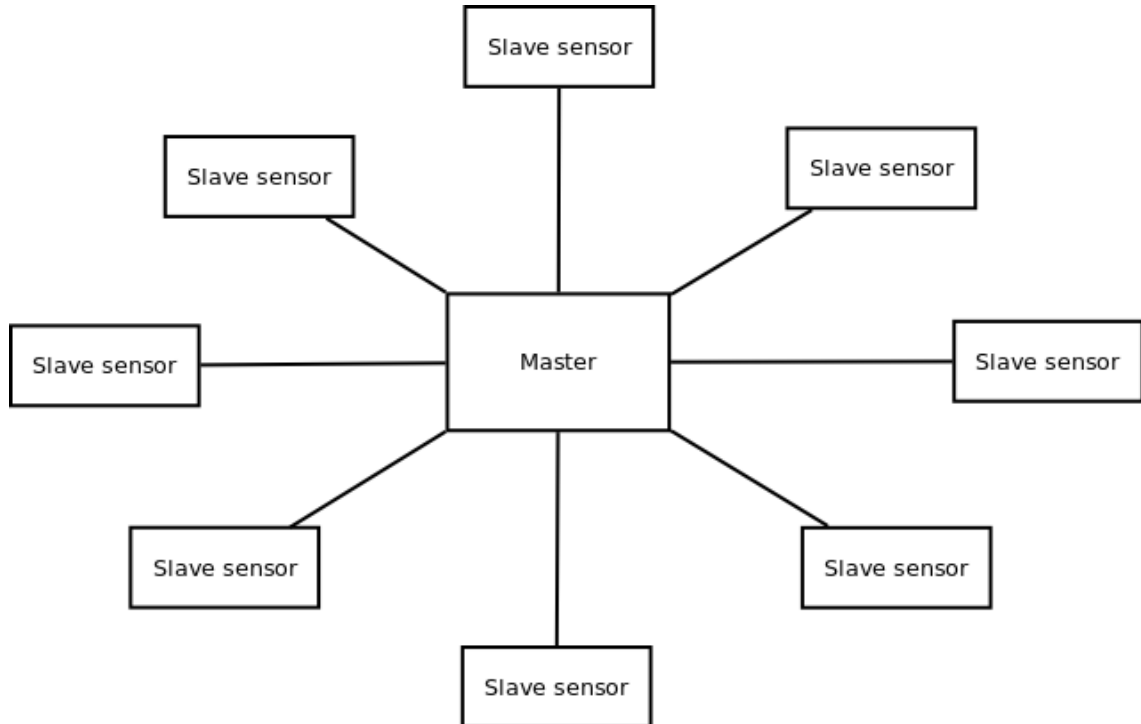
Panašių sistemų rinkoje labai mažai, tad šį susilaukė populiarumo. Ši sistema yra naudojama įvairiuose elektromobiliuose, laivuose, kranuose. ir t.t.

Ryškiausi klientų pasiekimai su „Emus BMS“:

- Komandos „ACCIONA“ bolidas dalyvavęs „Dakar“ ralyje
- Academic Motorsports Club Zurich (AMZ) automobilis, pasiekęs 100km/h per 1.785s, taip sumušdamas pasaulio rekordą
- „Intersolar 2013, Munich“ saulės jėgainės valdikliuose
- „EcoPower Hybrid“ kranas dirbantis texaso uoste
- „Lloyds paxter“ Pašto elektromobiliuose norvegijoje

## 2. Veikla praktikos metu

Praktikos metu įmonėje dirbau prie naujo įmonės projekto. Projekto metu buvo sukurta sistema skirta geležinkelių aukštos įtampos kabelio vibracijoms tirti. Sistema seka Šeimininko-Vergo (ang.: Master-Slave) topologiją, leidžiančią vienu metu palaikyti iki 16 „Slave“ tinklo mazgų.



2 pav. Vibracijų sensorių tinklo topologija.

- *Slave* - Moduliai matuojantys akseleracijas visomis trimis kryptimis ir komunuojantys su „Master“ moduliu
- *Master* - Modulis valdantis visą sistemą, priimančias matavimo duomenis, suteikiantis internetinę prieigą sistemos konfigūravimui, stebėjimui bei matavimų parsisiuntimui

Šio projekto metu nuo pagrindų buvo sukurtas žvaigždės topologijos radio tinklas, dėl labai aukštų elektros suvartojimo ir duomenų perdavimo greičio reikalavimų. Suprogramuotas specialus laiko dalinimosi algoritmas ramiom darbo sąlygom leidžiantis techninę radio ryšio įrangą įjungti mažiau nei 1% laiko, taip sutaupant elektros, kita vertus, atmetus visus protokolo nuostolius per vieną kanalą pasiekia-

mas pasiekiamas net 15KB/s srautas daugiau kaip 1 kilometro atstumu (jeigu tarp „Master“ ir „Slave“ įrenginių nėra kliūčių).

Šie algoritmai buvo pakankamai efektyvūs, kad „Slave“ sensoriai be pertraukų atliktų matavimus 7 dienas iki baterijos išsikrovimo.

Negana labai aukštų reikalavimų elektros suvartojimui, šie sensoriai turi stabiliai dirbti labai EMI triukšmingoje aplinkoje, nes jie montuojami teisiogiai ant aukštos įtampos kabelio. Tam buvo sukurta speciali dėžutė imituojanti faradėjaus narvo savybes.

„Master“ įrenginyje dirbo Linux operacinė sistema, kuriai buvo atskirai parašytos tvarkyklės privačiam šios sistemos protokolui palaikyti. Taip pat įrenginys buvo aprūpintas Wi-Fi ir GSM sąsajom, todėl parsisiųsti matavimus, stebėti kaip sistema dirba, o prireikus perkonfigūruotis „Slave“ bei „Master“ įrenginius galima nuotoliniu būdu.

## **2.1. Naujos Testavimo procedūros**

Prasidėjus profesinei praktikai projektas jau buvo pažengęs ir buvo atlikti bandymai realiuose geležinkeliuose su realiais traukiniais Trondheim ir Oslo traukinių stotyse, Norvegijoje. Deja, praktika parodė, kad sistemoje buvo nemažai defektų, tame tarpe ir kritinių, dėl kurių buvo įmanoma pilnai prarasti ryšį su „Slave“ sensoriais. Todėl buvo nuspręsta, kad reikia sustiprinti testavimo procedūras.

### **2.1.1. Regresiniai vienetų testai**

Vienetų testai įmonėje buvo naudojami ir anksčiau, bet, deja, darbas su jais buvo varginantis ir nepatogus. Vien jų paleidimas reikalavo specialios įrangos, kad pavyktų prijungti programuojamą įrenginį prie kompiuterio, testai turėdavo būti paleidinėjami rankomis. Reikėjo įvedinėti lengvai pamiršamas komandas. Testus pernešti iš vieno projekto (kodo bazės) į kitą būdavo labai sunku, tad kai kurie seniau parašytų modulių testai būdavo ištrynami norint greičiau pasiekti sėkmingą kompiliaciją. Nekartą pasitaikė situacija, kad testai būdavo išvis apleidžiami ir neatnešdavo realios naudos. Šių vienetų testų tikrai nebuvo galima pavadinti regresiniais, o tai yra viena iš stipresnių vienetų testų savybių.

Todėl po nesėkmingų sistemos bandymų Norvegijoje buvo nuspręsta sustiprinti vienetų testus.

**STM32 procesorių kodo struktūra (ang.: „Framework“).** Įmonėje tenka kompiliuoti kodą įvairioms STM32 procesorių platformoms ar įrenginiams

tačiau ankstesnioje kodo struktūroje pernešti kodą iš vieno įrenginio į kitą buvo sudėtinga užduotis. Tai turėjo įtakos ir vienetų testams. Todėl darbai prasidėjo kaip tik nuo kodo struktūros, ji buvo pilnai pertvarkyta.

**Nepriklausomumas nuo platformos.** Absoliučioje daugumoje įterptinių sistemų nuo įrenginio, platformos ar procesoriaus priklausantis kodas būna tik žemiausiame tvarkyklių lygmenyje, kitur priklausomybė lieka tik nuo modulių sąsajų. Todėl pirmas žingsnis kodo struktūros darbuose buvo sąsajų standartizavimas. Išskaidėme visas tvarkykles bei programinius modulius į šiuos tipus:

- *Output* - Išities įrenginys ar modulis.
- *Memory* - Atminties įrenginys ar modulis.
- *Measure* - Bendru atveju įities įrenginys ar modulis.
- *Stream* - Sąsajos įrenginys ar modulis.
- *Generic* - Kiti moduliai (dažniausiai - valdantieji).

Kiekvienam iš šių įrenginių ar modulių konkrečiai apibrėžėme jų sąsajas, bet jų apibūdinimas iškrenta iš šios ataskaitos apimties ribų, todėl čia nedetalizuosiu sąsajų.

Standartizuotos sąsajos leidžia pernaudoti kodą, kuris nėra tiesiogiai priklausomas nuo techninės įrangos, todėl norint pernešti kodą iš vienos platformos į kitą, tereikia užtikrinti, kad visos reikalingos tvarkyklės būtų kitoje platformoje. Taip pat standartizuotos sąsajos leidžia sukurti apsimestines (ang.: Mock) tvarkykles ir kompiliuoti kodą architektūroms, kur tos techninės įrangos galbūt nėra. Pavyzdžiui x86 architektūrai. Tai yra svarbiausia naujos kodo struktūros savybė vienetų ir integraciniams testams.

**Automatizuotas vienetų testų vykdymas.** Įmonėje yra naudojama git versijavimo sistema paleista privačiuose VPS serveriuose. Tai leidžia pilnai kontroliuoti ką, kada ir kaip git serveris atlieka. Pasinaudodami šiomis galimybėmis kartu su komanda sukūrėme git „kablius“ (ang.: hooks), kurie kiekvieną kartą kai kodo pakeitimai yra siunčiami į serverį („Push“ operacija), atlieka šiuos veiksmus:

- atnauja kodą naujausiais pakeitimais
- Ištrina bet kokius senesnių kompiliacijų likučius

- Švariai sukompiluoja naujausią kodą visoms palaikomoms platformoms, tame tarpe ir x86
- Automatiškai paleidžia vienetu testus

Įvykus kompiliacijos ar vienetų testų klaidai, serveris išsiunčia elektroninį laišką tam, kas siuntė kodą į serverį. Laiške pateikiama išsami informacija apie tai kodėl šis laiškas yra siunčiamas, jame pateikiami:

- Konkreti kodo versijos šaka ir versijos SHA1 identifikatorius
- Pilnas kompiliavimo žurnalas visoms platformoms (ang.: „log“)
- Vienetų testų rezultatai ir bet kokie klaidos pranešimai, jeigu jų yra.

Atlikę šiuos pakeitimus, galime drąsiai sakyti, kad vykdome regresinius vienetų testus.

### 2.1.2. „Checker“ modulis

Įterptinės sistemos daugeliu atveju pasižymi palyginus silpnais procesoriais, kilobaitų eilės „RAM“, „FLASH“ atmintimis, bei labai limituotom derinimo sąsajom. Dažna situacija kuomet vienintelis kelias derinimui yra „JTAG“ ar panašaus tipo derintuvai, ar paprastesniais atvejais „UART“ sąsaja su kompiuteriu, galinti išspausdinti tekstinę informaciją. Tai buvo mūsų situacija. Deja derinimas su tokiais įrankiais yra labai sudėtingas, ypač kai aplikacija yra realaus laiko ir labai jautri nenumatytiems uždelsimams. Taip yra dėl to, kad pats derinimo veiksmas reikalauja uždelsimų, kurie iškreipia normalų programos darbą, o 10 mikrosekundžių užgaištos spausdinant vieną raidę, gali lemti programos darbo sutrikimus.

**Sprendimo idėja.** Kol „UART“ sąsaja yra labai lėta, tai atminties operacijos šiuose procesoriuose yra ypatingai greitos dėl jų SRAM technologijų. Atminties priėjimas užtrunka vos keletą ciklų. Žinodamas tai pasiūliau komandai sukurti modulį, kurį pavadiname „Checker“.

**„Checker“ modulis.** „Checker“ modulis statiškai rezervuoja atminties bloką (prisiminkime, kad silpnesnieji iš mikrovaldiklių, net neturi dinaminio atminties valdymo, tad „malloc“ ir panašūs iškvietimai yra nepalaikomi), ir jame suformuoja žiedinį buferį. Ir suteikia tokią programinę sąsają kitiems moduliams:



```

#define chk(severity , code)  __check(severity , \
                                     __FILE__, \
                                     __LINE__, \
                                     __builtin_return_address(0) , \
                                     code)

#define chk_return(severity , code) \
    {chk(severity , code); return code;}

```

```

void __check(enum severity_e severity ,
             const char *filename ,
             int line ,
             void *return_addr ,
             int code);

```

```

int CHK_cnt();
const struct checker_record_s *CHK_getRecord(int idx);

```

Kiekviena iš šių funkcijų turi specialią reikšmę:

- *Macro `chk`* - macro nukreipiantis į `__check` funkciją
- *Macro `chk_return`* - macro nukreipiantis į `__check` funkciją ir gražinantis reikšmę „code“. Gali būti naudojamas tik funkcijose gražinančiose „int“ ar su juo suderinamus tipus
- *Funkcija `__check`* - funkcija užregistruojanti pranešimą į žiedinį buferį
- *Funkcija `CHK_cnt`* - funkcija gražinanti žiediniame buferyje esančių įrašų skaičių
- *Funkcija `CHK_getRecord`* - funkcija gražinantį rodyklę į žiediniame buferyje esantį įrašą su indeksu „idx“

Programuotojai pasinaudodami „chk“, „chk\_return“ ir kitom modulio sąsajos suteikiamom galimybėm deda įrašus į žiedinį buferį kiekvienoje klaidas tikrinančioje ar bet kaip kitaip „įtartinoje“ programos vietoje.

Prisipildžius žiediniam buferiui, seniausias įrašas yra pakeičiamas naujesniu, taip praprandant dalį informacijos, tačiau tai leidžia apsiriboti statistiškai išskirtu buferiu, bei tiksliai numatyti modulio atminties suvartojimą.

**Žiedinio buferio turinio gavimas.** Jeigu informacija būtų tik kaupiama žiediniame buferyje, tai nesuteiktų jokios naudos, todėl sąsajoje yra dar dvi funkcijos žiedinio buferio turiniui gauti: „CHK\_cnt“ ir „CHK\_getRecord“. Pasinaudojant šiomis funkcijomis buvo sukurta komanda išspausdinanti žiedinio buferio turinį per „USB“ ar „UART“ sąsajas. Komanda suformuodavo duomenis lentelės pavidalu:

Severity	Error Code	Return address	Line	File
0	-19 (No such device)	0x8013103 (DEVLIST_getActiveDevPBydevconf)	487	dev_list.c
0	-19 (No such device)	0x8010c71 (ACTRL_Init)	365	attitudeCtrl.c
0	-19 (No such device)	0x80130db (DEVLIST_getActiveDevPByDevidAndItf)	487	dev_list.c

1 lentelė. Žiedinio buferio turinio pavyzdys, kuomet nėra prijungtas sukonfigūruotas sensorius

Deja įmonė statistinių duomenų apie sugaištą laiką derinant sistemos darbą nekaupė, bet net ir be jų buvo labai aiškus laiko sutaupymas. Daugeliu atveju jeigu sistema dirbdavo netinkamai užtekdamo tik pasižiūrėti į žiedinio buferio turinį ir problema paaiškėdavo iškart. Tai sutaupė ilgas valandas derinimo, ieškant problemos.

Taip pat iškilus problemoms ir esant tuščiam žiedinio buferio turiniui (praktikos laikotarpiu pasitaikė 2 tokie atvejai), su dideliu patikimumu galima problemos ieškoti techninėje, o ne programinėje įrangoje. Abiejais atvejais reali problema ir buvo techninėje įrangoje.

**Modulis „Checker“ produkcinėje aplinkoje.** Funkcija „\_\_check“ (taigi atitinkamai ir macro „chk“ bei „chk\_return“) teatlieka atminties kopijavimo operaciją, kuri yra nepalyginamai greitesnė, negu spausdinimas „UART“ ar kitom sąsajom, todėl pats „Checker“ modulio darbas tampa nepastebimas normaliai programos eigai.

Dėl šių priežasčių modulis „Checker“ gali būti paliktas įjungtas net tik viso programavimo ir testavimo metu (ko negalėtume padaryti naudojant „JTAG“ ir panašius derintuvus), bet net ir produkcinėje aplinkoje. O paruošus tinkamus ir patogius įrankius, klientai gali atsiųsti žiedinio buferio turinį, kuris padeda diag-

nozuoti ne tik vidinius kodo defektus, bet ir pajungimo, instaliavimo bei konfigūravimo problemas.

### 3. Rezultatai

UAB „Elektromotus“ jau dirbu ketverius metus, nepaisant to, praktikos laikotarpiu buvo atlikti svarbūs testavimo procedūrų patobulinimai, kurie leido ne tik pagerinti įmonės programuotojų efektyvumą, bet leido man asmeniškai daug išmokti apie darbo bei testavimo procesus įterptinėse sistemose.

**Praktikos rezultatai.** Praktikos laikotarpiu buvo atnaujinta vienetų testavimo procedūra leidžianti efektyviau pritaikyti testais vedamo programavimo (Ang.: Test Driven Development), bei judriojo kūrimo metodus (Ang. „Agile development“). Naujas „Checker“ modulis suteikia galimybę atlikti gilią diagnostiką ne tik testinėje ar programavimo aplinkoje, bet ir produkcinėje, kartu stipriai sumažindama derinimui skirtą laiką. Pritaikius šiuos principus įmonės kuriamą sensorių tinklo sistema sėkmingai atliko matavimus Melhus geležinkelio stotyje Norvegijoje, naujų kritinių defektų nebuvo rasta.

Vilniaus Universitete Matematikos ir Informatikos fakultete Programų sistemų kurse dėmesys skiriamas įterptinėms sistemoms yra gana ribotas. Pateikiami metodai nevisada gali būti tiesiogiai pritaikomi įterptinių sistemų programavimo srityje, tačiau dėstytojų geranoriškumas ir aktyvumas padeda išspręsti šias problemas ir surasti sprendimus ne tik abstraktesniems principams, bet ir konkrečiom situacijoms. Norėčiau atskirai padėkoti lektoriui A. Adamoniui už aktyvią pagalbą ieškant sprendimų įterptinių sistemų testavimo problemoms spręsti.

**Pasiūlymai UAB „Elektromotus“ bei Universitetui.** Įmonėje programavimo bei testavimo procedūros tobulėja, produktų kokybė gerėja, tačiau vidiniuose procesuose dar daug neapibrėžtumų, nevisada aišku, kokių veiksmų reikia imtis tam tikrose situacijose, nevisada aiškos atsakomybių ribos. Neretai projektų įvykdymas vėluoja. Nėra tiesioginių motyvatorių darbotuojam gerai atlikti projektus, išleisti naujus produktus. Tad apibendrinant pateikiu pasiūlymus įmonei:

- Sustiprinti projektų valdymą, pasamdyti projektų vadovą.
- Sustiprinti projektų portfilio valdymą ir projektų išsirinkimą.
- Pateikti pasiūlymų, kurie pakeltų darbuotojų motyvaciją gerai vykdyti projektus.

Vilniaus Universitete Matematikos ir Informatikos fakultete dažnai vykdomos apklaustos išreikšti savo nuomuonei, todėl aktyvių studentų nuomonė nelieka

neiškakytą. Taip pat studentai, norintys daugiau sužinoti konkretesnėse srityse, turi pakankamai platų įvairių dalykų pasirinkimą. Konkrečiu atveju, man pasirodė, kad Programų Sistemų kurse buvo mažokai informacijos apie įterptines sistemas, jų programavimą, testavimą, taip pat projektų susijusių su įterptinėm sistemo valdymą.

Tad apibendrinant pateikiu pasiūlymus Universitetui

- Įterpti daugiau galimybių sužinoti apie įterptines sistemas ir jų programavimą, bei testavimą.
- Kituose kursuose pateikti daugiau pavyzdžių kaip pritaikyti dėstomus principus įterptinių sistemų kūrimo procese.
- Labiau paskatinti studentų aktyvumą domėtis nestandartinėmis temomis.

[SM04 ] - *Taming the Embedded Tiger – Agile Test Techniques for Embedded Software*, [http://www.leanagilepartners.com/library/XR5\\_Taming\\_Embedded\\_Tiger.pdf](http://www.leanagilepartners.com/library/XR5_Taming_Embedded_Tiger.pdf)