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, doc.....
(parašas)

Vilnius
2014

Įvadas

1. Apie UAB „Elektromotus“

UAB „Elektromotus“ 2010 metais įkurta trijų steigėjų: Gintauto Palucko, Mindaugo Milašausko ir Šarūno Šutavičiaus su tikslu realizuoti jų sukauptą 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 viršyja 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.

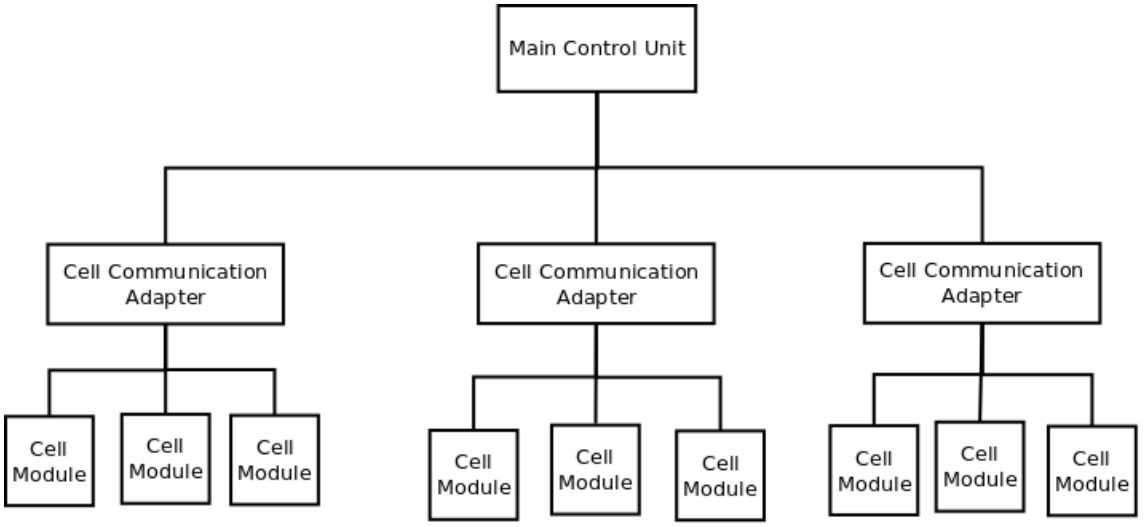
1.1. Produktai

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, temperatūra, turi šiluminio balansavimo palaikymą.

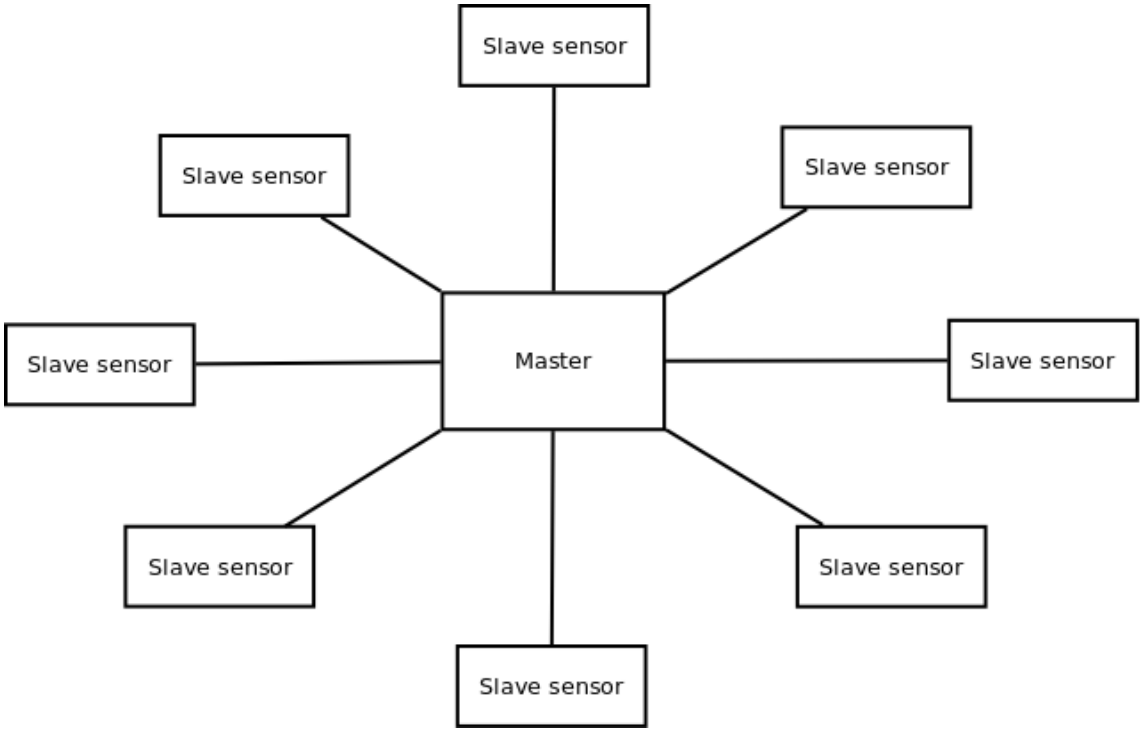
Panašių sistemų rinkoje labai mažai, tad šį susilaukė populiarumo. Ši sistema yra naudojama įvairiuose elektromobiliuose, laivuose, kranuose. ir t.t.

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 jegainės 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.

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 algorithmas 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 pasiekiamas 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 ankščiau, 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 pamirštamas 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 nuspresta 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* - Išeities įrenginys ar modulis.
- *Memory* - Atminties įrenginys ar modulis.
- *Measure* - Bendru atveju įeities 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 nedetalizuosių 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 standarizuotos 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:

- atnaujina kodą naujausiais pakeitimais
- Ištrina bet kokius senesnių kompiliacijų likučius
- Švariai sukompiliuoja 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 įdėja. 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 pavadinome „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 funkcija 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 statiškai 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. Daugelių atveju jeigu sistema dirbdavo netinkamai užtekdavo 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 priežaščių 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 atsiūsti žiedinio buferio turinį, kuris padeda diagnozuoti ne tik vidinius kodo defektus, bet ir pajungimo, instaliavimo bei konfigūravimo problemas.

3. Rezultatai

3.1. Vidutinis klaidų paieškos laikas

3.2. Sugauti defektai regresinio vienetų testavimo metu

4. Išvados

[SM04] - *Taming the Embedded Tiger – Agile Test Techniques for Embedded Software*, http://www.leanagilepartners.com/library/XR5_Taming_Embedded_Tiger.pdf