Rendszertervezés házi feladat

CAN alapú autóipari hálózat megvalósítása

2010/2011. I. félév

Név	Neptun-kód
Erdős Csanád	FXX9LS
Farkas Gergő	L5OQ34
Fodor Gábor	YBGXAF
Fuli Balázs	IM0WP2

Tartalomjegyzék

I. Feladatkiírás

Specifikáció autóipari hálózat tervezéséhez

Cél egy CAN és/vagy LIN alapú egyszerű body hálózat első prototípusának megvalósítása mitmót alapon. A második iterációt nem kell megvalósítani, de a feladatot végzők specifikálják, hogy a jelenlegi mitmótos hálózat milyen tulajdonságai nem megfelelőek és milyen részeken kellene változtatni a következő iterációnál.

Az autós body hálózat a következő egységekkel rendelkezzen:

- A hálózat valósítsa meg az autóban található 4 darab ajtó, és az autó lámpáinak vezérlését.
- Az ajtók következő funkciókkal rendelkezzenek
    - Első ajtók
        - Központi zár
        - Ablakemelő (2 motort adunk hozzá)
        - Visszapillantó tükör vezérlés (egy tényleges visszapillantó tükröt adunk)
        - Állapot visszajelzés (nyitott, csukott)
    - Hátsó ajtók
        - Központi zár
        - Ablakemelő (2 motort adunk hozzá)
        - Állapot visszajelzés (nyitott, csukott)
- A lámpák funkciója
    - Első lámpa blokk (szokásos lámpák).
    - Hátsó lámpa blokk (szokásos lámpák, igény szerint tényleges lámpablokk kérhető, de LED-ek is tökéletesen elegendőek).

A funkcionalitás kialakítását a feladat végrehajtóra bízzuk, de indokolja, ha eltér a szokásos autóban használható kialakítástól. A fejlesztésnél a Bosch laborban található rendszer igénybe vehető, a kormány illetve a laborautó CAN üzeneteit fel lehet és kell is használni,

Elvégezendő feladatok:

1. A laza szöveges specifikációból készítsenek műszaki specifikációt. A specifikáció beépülését a tervbe kövessék végig a feladat során.
2. A feladat kidolgozásánál jelöljék ki a részfeladatokat.
3. Határozzák meg a részfeladatok felelőseit.
4. Készítsenek munkatervet, jelöljenek ki mérföldköveket.
5. Egyeztessék az egyes részfeladatok közötti kommunikációs, valamint hardware és software interfészt.
6. Alkalmazzanak verziókövetést.
7. Határozzanak meg egy közös code style guide-ot.
8. Tervezzék meg a modulok és az egész rendszer tesztelési lépcsőit és eljárásait.

II. Specifikáció

II.1. A body hálózat funkcionális egységei

Az alábbiakban áttekintjük az autó body rendszerének elemeit, és specifikáljuk a viselkedésüket.

II.1.a) Kormány

A rendelkezésünkre bocsátott kormányról tudjuk, hogy a kormány elfordulásának szögét, a pedálállásokat és a kormányon lévő gombok állapotát fix időközönként elküldi egy CAN üzenetben (60 ms időköz, 0x100-as azonosító). A kormány által szolgáltatott információk közül az általunk tervezett body hálózatnak bizonyos gombok állapotára és a fékpedál állására van szüksége.

A gombok funkciói (ld. II.1.b) és II.1.c) pontok):

- 1 gomb: helyzetjelző világítás ki/be kapcsolása
- 1 gomb: tompított fényszóró ki/be kapcsolása
- 1 gomb: ködfényszóró ki/be kapcsolása
- 1 gomb: ködzárfény ki/be kapcsolása
- 1 gomb: reflektor ki/be kapcsolása
- 1 gomb: bal index ki/be kapcsolása (A bekapcsolt index 1-2 Hz frekvenciával villog.)
- 1 gomb: jobb index ki/be kapcsolása (A bekapcsolt index 1-2 Hz frekvenciával villog.)

A különféle világítást kapcsoló gombok a fénykürt kivételével élvezérelt működésűek, azaz 1 gombnyomás hatására az adott lámpa állapotot vált (ki→be vagy be→ki) függetlenül a gombnyomás hosszától. Fénykürt esetén addig ég a reflektor, amíg a gomb lenyomott állapotban van (szintvezérelt működés).

Ha a fékpedál állása meghalad egy bizonyos küszöböt (például 5%-nál jobban le van nyomva), akkor kigyullad a féklámpa.

II.1.b) Első lámpablokkok

A rendszer két darab első lámpablokkot tartalmaz, egy bal és egy jobboldalit. Az indexet leszámítva a működés szimmetrikus, így a funkcionalitás egy pontban tárgyalható.

Az első lámpatest a következő lámpákból áll:

- Helyzetjelző izzó
- Tompított fényszóró izzója
- Reflektor (távolsági fényszóró) izzója
- Index izzó
- Ködlámpa

A felsorolt világítótesteket a kormányról kapcsolhatjuk a II.1.a) pontnak megfelelően.

II.1.c) Hátsó lámpablokkok

A rendszer két darab hátsó lámpablokkot tartalmaz, egy bal és egy jobboldalit. Az indexet leszámítva a működés szimmetrikus, így a funkcionalitás egy pontban tárgyalható.

A hátsó lámpatest a következő lámpákból áll:

- Helyzetjelző izzó
- Féklámpa izzó
- Index izzó
- Ködzárfény izzó

A felsorolt világítótesteket a kormányról, illetve a fékpedállal kapcsolhatjuk a II.1.a) pontnak megfelelően. Fontos, hogy azok a lámpák, melyek elöl és hátul is megtalálhatóak, szinkronban működjenek (egy időben nem lehetnek különböző állapotban), ez különösen a villogó indexnél kritikus.

II.1.d) Bal első ajtó

Az ajtók az előző egységekkel szemben egyszerre tartalmaznak kezelő és beavatkozó szerveket is. A vezető oldali, azaz bal első ajtó rendelkezik a legkomplexebb funkcionalitással.

Beavatkozó egységek:

- Ablakmozgató motor: 3 állapottal rendelkezik: fel/le/stop.
- Központi zár: 2 állapottal rendelkezik: zárt/nyitott.
- Visszapillantó tükör fűtés: 2 állapottal rendelkezik: fűt/nem fűt.
- Tükörmozgató egység: Két tengely (fel-le és jobbra-balra) irányában mozgatható a visszapillantó tükör. A tükörmozgató egység állapota a két tengelynek megfelelő motorállapotok együttes állapotával írható le (tükörmozgató állapot={fel/le állapot}×{jobbra/balra állapot}). Egy tengelyhez tartozó motornak 3 állapota lehet (fel, le, stop és jobbra, balra, stop), tehát összesen 9 állapot lehetséges.

Kezelőszervek:

- 1 gomb: központi zár nyitása/zárása (az összes ajtóra).
- 4×2 gomb: Egy gomb nyomva tartásával mozgatható az ablak felfelé, míg egy másikkal lefelé. Mind a 4 ajtó elektromos ablakemelővel rendelkezik, és ezek mindegyike mozgatható a vezető oldali ajtó tasztatúrájáról, így az ablakmozgatásokra itt összesen 8 gomb szolgál.
- 1 gomb: visszapillantó tükör fűtés ki/bekapcsolása (mind a két tükörre).
- 1 választókapcsoló: Visszapillantó tükör kiválasztása a beállításhoz (bal/jobb).
- 4 gomb: A választókapcsolóval kiválasztott visszapillantó tükröt 4 gomb segítségével állíthatjuk be, tengelyenként 1-1 gomb nyomva tartásával mozgathatjuk a szükséges irányba a tükröt (a 4 gomb szerepe: fel, le, jobbra, balra).

A központi zárat és a tükörfűtést kapcsoló gombok élvezérelt működésűek, azaz 1 gombnyomás hatására az adott beavatkozó szerv állapotot vált (ki→be vagy be→ki) függetlenül a gombnyomás hosszától. Motorvezérlés esetén (ablak és tükörmozgatás) addig

mozog a motor, amíg a megfelelő vezérlő gombot nyomva tartjuk (szintvezérelt működés). Itt feltételezzük, hogy a végállás elérése hardveresen kezelve van.

II.1.e) Jobb első ajtó

Beavatkozó szervek:

A jobb első ajtóban megtalálható beavatkozó szervek megegyeznek a bal első ajtóéval (II.1.d) pont).

Kezelőszervek:

A jobb első ajtón a következő kezelőszervek találhatóak:

- 1 gomb: központi zár nyitása/zárása (az összes ajtóra).
- 1×2 gomb: Egy gomb nyomva tartásával mozgatható az ablak felfelé, míg egy másikkal lefelé. Ennek az ajtónak a tasztatúrájáról csak az ezen az ajtón lévő ablak mozgatható.

II.1.f) Hátsó ajtók

A két hátsó ajtó funkcionalitás tekintetében szimmetrikusak, így együtt tárgyalhatóak.

Beavatkozó szervek:

- Ablakmozgató motor: 3 állapottal rendelkezik: fel/le/stop.
- Központi zár: 2 állapottal rendelkezik: zárt/nyitott.

Kezelőszervek:

- 1×2 gomb: Egy gomb nyomva tartásával mozgatható az ablak felfelé, míg egy másikkal lefelé. Az ajtó tasztatúrájáról csak az azonos ajtón lévő ablak mozgatható.

Az ajtók esetében is kritérium, hogy bizonyos beavatkozó egységek állapota csak szinkronban változhat (azaz az összetartozó beavatkozó szervek nem lehetnek eltérő állapotban). Az összetartozó beavatkozó szervek az alábbiak:

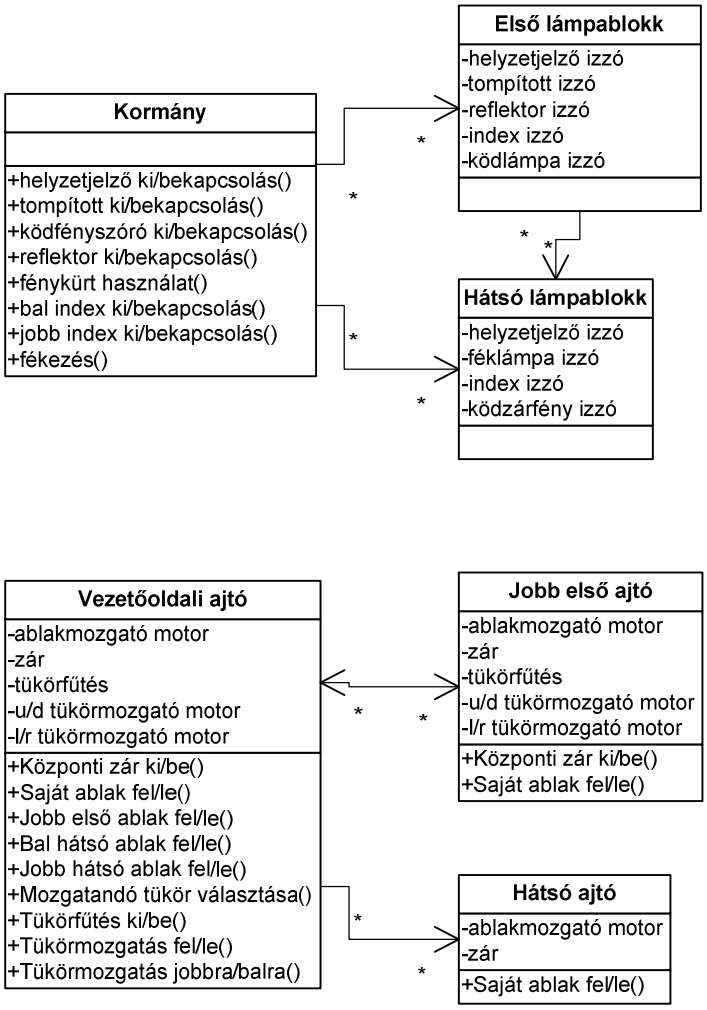
- A 4 ajtóban megtalálható (központi) zár.
- Az első ajtók esetén a tükörfűtés.

II.2. Funkcionális blokkdiagram

Az alábbi ábra a body rendszer funkcionális blokkdiagramját ábrázolja. Ehhez az UML osztálydiagram szokásos elemei lettek felhasználva, a következő értelmezés szerint: Az osztályok az egyes funkcionális egység típusoknak felelnek meg. Az osztályváltozók írják le az egység állapotát. Az osztályváltozókhoz közvetlenül nem férünk hozzá, csak más egységeken, kezelőszerveken keresztül, így azok private típusúak. A metódusok írják le, hogy a felhasználó mihez kezdhet az adott berendezéssel, azaz a metódusok közvetlenül a kezelőszervekhez rendelhetők.

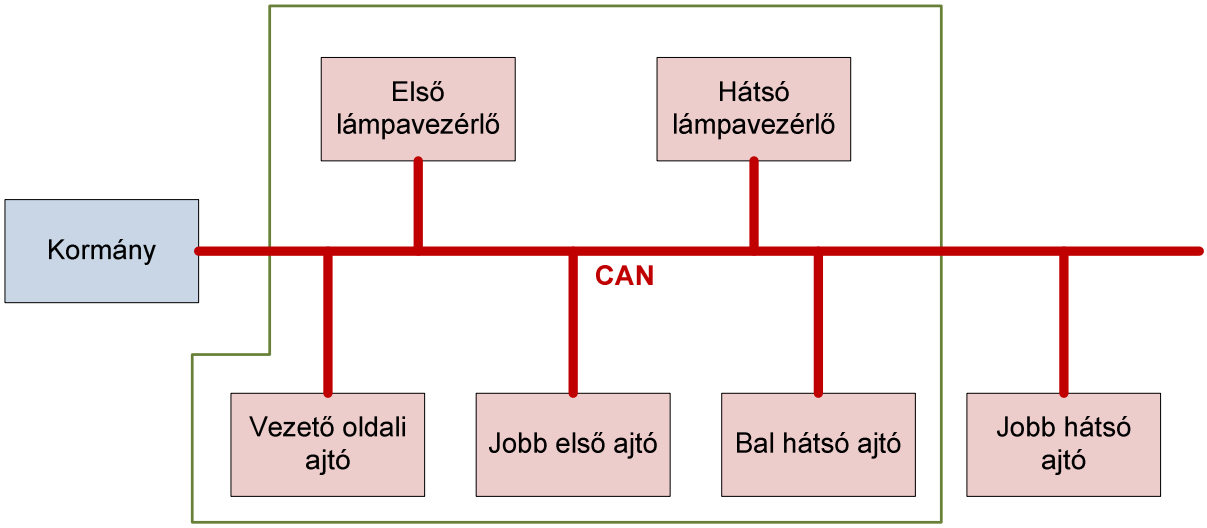
Az irányított asszociációk itt az eszközök közötti kommunikációs kapcsolatot jelentik. Az irányítottság az információáramlás fő irányát jelzik (konkrétabban a vezérlő típusú információkét, természetesen státuszinformációk, visszajelzések más irányokban is előfordulhatnak). Itt két kapcsolatnál van szükség magyarázatra:

Az első és a hátsó lámpablokk közötti, vezérlő jellegű kapcsolatra az index szinkronizált villogásának biztosítása végett van szükség: bekapcsolt index esetén az első lámpablokk generálja az 1-2 Hz-es villogást, és ez alapján vezérli a saját izzói mellett a hátsó lámpablokkot is.
A vezető oldali és a jobboldali ajtó között a kétirányú kapcsolat oka, hogy a jobb első ajtóról is kapcsolható a központi zár. A központi nyitásért és zárásért a bal első ajtó a felelős, tehát ilyenkor a jobb első ajtó jelzi a másiknak a nyitási/zárási szándékot, ami alapján az kiadja a parancsot az összes ajtónak.



II.3. Hardver architektúra tervezése a funkcionális egységek alapján

Az általunk megtervezett body rendszer blokkvázlata látható az alábbi ábrán.



A két első és a két hátsó lámpablokk kezeléséről 1-1 vezérlő egység (ECU) gondoskodik. A két első (illetve hátsó) lámpablokkban az indextől eltekintve páronként azonos működésű lámpák találhatóak, és ezek közel is találhatóak egymáshoz, így ésszerű ezek kezelését egyetlen vezérlővel megoldani. Ezen felül minden ajtóban 1-1 vezérlő egység található. Több ajtó funkcióját nem célszerű egy közös kontrollerbe összevonni, hiszen az ajtók között vezetékezés szempontjából nagy a távolság, és az egyes ajtók funkcionalitása is jobban szétválik, mint a lámpatestek esetében. Így összesen (a kormányt nem számolva) 6 vezérlő egységből áll az általunk megvalósítandó body rendszer. A két hátsó ajtóban található vezérlő működése a bal/jobb oldal megkülönböztetésétől eltekintve megegyezik, így a házi feladat során csak a bal hátsó ajtóvezérlőt implementáltuk. A házi feladat során megvalósítandó eszközöket zöld keret jelzi az ábrán.

II.4. A node-ok közötti kommunikáció specifikációja

II.4.a) A node-ok közötti kapcsolatok összerendelése

		Fogadó					
Küldő	Üzenet-azonosító	Kormány	Első lámpablokk	Hátsó lámpablokk	Bal első ajtó	Jobb első ajtó	Hátsó ajtó
Kormány	0x100		**X**	**X**			
Első lámpablokk	0x110			**X**			
Hátsó lámpablokk	0x111						
Bal első ajtó	0x120					**X**	**X**
Jobb első ajtó	0x121				**X**		
Hátsó ajtó	0x122						

II.4.b) A megvalósított body rendszer üzenetei

Az általunk létrehozott rendszerben a lentebb definiált üzenetek jelennek meg. Mindegyik forrása egy-egy ECU. Az üzenetek általánosan mindig 3 részre bonthatóak, ezek **[a | b | c]**, ahol:

a) Az adott ECU-hoz kapcsolódó összes kezelőszerv (gombok, kapcsolók, potméterek) pillanatnyi állapota. Más ECU vezérlésére és diagnosztikai célokra egyaránt felhasználható.

b) Az adott ECU által vezérelt összes beavatkozó szerv (motorok, relék, LED-ek, lámpák, stb.) aktuális állapota. Értelmet akkor nyer, ha az ECU-ban a beavatkozó szervről van valamilyen ellenőrző visszacsatolás (pl. áramfigyelés). Az ECU állapotjelzése is ide sorolható. Diagnosztikai célokra használható.

c) Parancsbitek. A c) csoportba olyan jelzések tartoznak, melyek nem sorolhatóak az előző két csoportba, tehát nem egy adott ECU-hoz kapcsolódó paraméter, hanem inkább az egész body hálózat valamilyen globálisabb jellemzője. Például a bal oldali összes index állapota, a központi zár (logikai, globális) állapota. Ezeket a jelzéseket parancsként lehet értelmezni, tehát olyan ECU-k adnak ki c) csoportba tartozó jelzéseket, melyek valamilyen, több ECU-n elosztva megvalósított funkcióért felelősek. (Pl. bal első ajtó – központi zár, első lámpablokk – indexek).

A fenti csoportok az üzenetben bájthatárra illeszkednek. Az egyes csoportokat a konkrét üzenetformátumokban színekkel különböztettem meg az egyszerűbb értelmezés érdekében.

Kormány – 0x100

Bájt	Bit	Funkció
0.	7-0	Kormányállás felső helyiérték
1.	7-0	Kormányállás alsó helyiérték
2.	7-0	Gázpedál állás
3.	7-0	Fékpedál állás
4.	7.	Bal felső szürke gomb
	6.	Jobb felső szürke gomb
	5.	Hátsó bal gomb
	4.	Hátsó jobb gomb
	3.	Jobb felső fekete (4.) gomb
	2.	Jobb felső fekete (3.) gomb
	1.	Jobb felső fekete (2.) gomb
	0.	Jobb felső fekete (1.) gomb
5.	7-4	Nincs szerepe
	3	Bal alsó szürke felső gomb
	2	Jobb alsó szürke felső gomb
	1	Jobb alsó szürke alsó gomb
	0	Bal alsó szürke alsó gomb

Első lámpablokk – 0x110

Bájt	Bit	Funkció
0.	7.	ECU állapota
	6.	Nem használt
	5.	Távolsági fényszóró izzó állapota
	4.	Tompított fényszóró izzó állapota
	3.	Ködlámpa izzó állapota
	2.	Jobb irányjelző izzó állapota
	1.	Bal irányjelző izzó állapota
	0.	Helyzetjelző izzó állapota
1.	7-3.	Nem használt
	2.	Helyzetjelző előírt állapota
	1.	Bal irányjelző előírt állapota
	0.	Jobb irányjelző előírt állapota

Hátsó lámpablokk – 0x111

Bájt	Bit	Funkció
0.	7.	ECU állapota
	6-5.	Nem használt
	4.	Féklámpa izzó állapota
	3.	Ködzárfény izzó állapota
	2.	Jobb irányjelző izzó állapota
	1.	Bal irányjelző izzó állapota
	0.	Helyzetjelző izzó állapota

Bal első ajtó – 0x120

Bájt	Bit	Funkció
0.	7.	„Jobb hátsó ablak fel” gomb állapota
	6.	„Jobb hátsó ablak le” gomb állapota
	5.	„Bal hátsó ablak fel” gomb állapota
	4.	„Bal hátsó ablak le” gomb állapota
	3.	„Jobb első ablak fel” gomb állapota
	2.	„Jobb első ablak le” gomb állapota
	1.	„Bal első ablak fel” gomb állapota
	0.	„Bal első ablak le” gomb állapota
1.	7.	„Visszapillantó tükör balra” gomb állapota
	6.	„Visszapillantó tükör jobbra” gomb állapota
	5.	„Visszapillantó tükör le” gomb állapota
	4.	„Visszapillantó tükör fel” gomb állapota
	3.	Visszapillantó tükör választókapcsoló állapota
	2.	„Visszapillantó tükör fűtés” gomb állapota
	1.	Nincs felhasználva
	0.	„Központi zár” gomb állapota
2.	7-6.	Ablakmozgató motor állapota (10: fel, 01: le)
	5.	Tükörfűtés állapota ezen a tükrön
	4-1.	Tükörmozgató motor állapota {balra,jobbra,le,fel}
	0.	Központi zár tényleges állapota ezen az ajtón
3.	7.	ECU állapota
	6-0.	Nem használt
4.	7.	Központi zár előírt állapota az összes ajtóra
	6.	Tükörfűtés előírt állapota mindkét tükörre
	5-0.	Nem használt

Jobb első ajtó – 0x121

Bájt	Bit	Funkció
0.	7-4.	Nem használt
	3.	„Jobb első ablak fel” gomb állapota
	2.	„Jobb első ablak le” gomb állapota
	1.	Nem használt
	0.	„Központi zár” gomb állapota
1.	7-6.	Ablakmozgató motor állapota (10: fel, 01: le)
	5.	Tükörfűtés állapota ezen a tükrön
	4-1.	Tükörmozgató motor állapota {balra,jobbra,le,fel}
	0.	Központi zár tényleges állapota ezen az ajtón
2.	7.	ECU állapota
	6-0.	Nem használt

Bal hátsó ajtó – 0x122

Bájt	Bit	Funkció
0.	7-6.	Nem használt
	5.	„Bal hátsó ablak fel” gomb állapota
	4.	„Bal hátsó ablak le” gomb állapota
	3-0.	Nem használt
1.	7-6.	Ablakmozgató motor állapota (10: fel, 01: le)
	5-1.	Nem használt
	0.	Központi zár tényleges állapota ezen az ajtón
2.	7.	ECU állapota
	6-0.	Nem használt

Jobb hátsó ajtó – 0x123

Bájt	Bit	Funkció
0.	7.	„Jobb hátsó ablak fel” gomb állapota
	6.	„Jobb hátsó ablak fel” gomb állapota
	5-0.	Nem használt
1.	7-6.	Ablakmozgató motor állapota (10: fel, 01: le)
	5-1.	Nem használt
	0.	Központi zár tényleges állapota ezen az ajtón
2.	7.	ECU állapota
	6-0.	Nem használt

III. Részfeladatok és mérföldkövek

A body rendszer megvalósítása során a viszonylag komplex feladatot részfeladatokra dekomponáltuk, ezekhez mérföldköveket, felelősöket és határidőket rendeltünk.

A részfeladatok az alábbiak lettek:

1. A részfeladatok meghatározása ☺
2. SVN szerver és tárhely felélesztése, beüzemelése
3. Code style guide meghatározása, specifikációja
4. Kommunikációs kapcsolatok megtervezése, mátrix készítése
5. CAN-es API feltérképezése, interruptos kiegészítése
6. Platform szoftver megtervezése
7. main.c függvény megírása
8. Node-ok implementálsa:
    a. Első és hátsó lámpablokk
    b. Bal első ajtó
    c. Jobb első ajtó
    d. Hátsó ajtók
9. Dokumentáció készítése

Az egyes feladatok felelősei és határidők:

Feladat megnevezése	Felelős	Határidő
1. A részfeladatok meghatározása	Közös	2010.10.15.
2. SVN szerver és tárhely felélesztése, beüzemelése	Gábor	2010.10.22.
3. Code style guide meghatározása, specifikációja	Csanád	2010.10.22.
4. Kommunikációs kapcsolatok megtervezése, mátrix készítése	Balázs	2010.10.22.
5. CAN-es API feltérképezése, interruptos kiegészítése	Gergő	2010.10.22.
6. Platform szoftver megtervezése	Csanád	2010.11.01.
7. main.c függvény megírása	Gábor	2010.11.01
8. Node-ok implementálása:		2010.11.20.
a. Első és hátsó lámpablokk	Gábor	2010.11.20.
b. Bal első ajtó	Csanád	2010.11.20.
c. Jobb első ajtó	Balázs	2010.11.20.
d. Hátsó ajtók	Gergő	2010.11.20.
9. Dokumentáció készítése	Közös	2010.12.10.

IV. Code style guide

IV.1. A .c fájlok mintája

A program egy fájlt azonosító summary-vel kezdődik az alábbi példa alapján:

```
/*****************************************************************************
* Title: Rendszertervezés HF                                                 *
* Hardware: CAN/LIN extension board for mitmót system                        *
* Processor: ATMEGA128                                                       *
* Author: Gergő Farkas                                                       *
* Date: 2010-10-17 23:24                                                     *
* Compiler: avr-gcc                                                          *
* ----------------------------                                               *
* Description: Simple example program for using the MCP2515 CAN interface    *
*****************************************************************************/


/*============================================================[ INCLUDES ]=*/
Ide kerül az összes include (CSAK .h fájlokra vonatkozhatnak!)

#include "platform.h"         Először a közös header fájl(ok)ra

#include "dpy_trm_s01.h"      Utána az összes headerre, amire a működése épül
#include "TIMER.h"

#include "main.h"             Végül a saját header fájljára

/*=================================================[ COMPILER SWITCH CHECK ]=*/

Ide kerülnek a feltételes fordításra vonatkozó ellenőrzések

/*=========================================================[ INTERNAL MACROS ]=*/

Ide kerülnek a belső (privát) makrók és define-ok, pl:
#define L_CAN_CS_DIR_OUTPUT()               GPIO_13_DIR_OUTPUT()

#define L_CAN_U8_VALTOZO                    ((uint8_t)(56))

L_mmm_tt_variable

Ahol:
- L prefix (local)
- mmm a 3 karakteres, csupa nagybetűs modulazonosító,
- tt a változó típusa (v, u8, s8, u8p, s8p, u16, ..., f32, stb.) csupa nagybetűvel
- variable a változó neve, csupa nagybetűvel

/*=========================================================[ INTERNAL TYPEDEFS ]=*/

Ide kerülnek a belső (privát) típusdeklarációk

/*=========================================================[ INTERNAL GLOBALS ]=*/

Ide kerülnek a belső (privát) változók definiálása (mind statikus)

static uint16_t* L_CAN_u16p_message;
static uint8_t   L_CAN_u8_counter = 0;

L_mmm_tt_variable

Ahol:
- L prefix (local)
- mmm a 3 karakteres, csupa nagybetűs modulazonosító,
- tt a változó típusa (v, u8, s8, u8p, s8p, u16, ..., f32, stb.) kisbetűvel
- variable a változó neve, tetszőlegesen kis- és nagybetűkkel

Nem struktúra változóknak kezdőértéket kell adni!

/*=========================================================[ EXTERNAL GLOBALS ]=*/

Ide kerülnek a külső (public) változók definiálása

uint16_t* CAN_u16p_message2;
```

```
uint8_t   CAN_u8_counter2 = 0;
```

mmm_tt_variable

Ahol:
- *mmm* a 3 karakteres, csupa nagybetűs modulazonosító,
- *tt* a változó típusa (v, u8, s8, u8p, s8p, u16, ..., f32, stb.) kisbetűvel
- *variable* a változó neve, tetszőlegesen kis- és nagybetűkkel

Nem struktúra változóknak kezdőértéket kell adni!

```
/*==================================================[ INTERNAL PROTOTYPES ]=*/
/* static functions only! */
```

Ide kerülnek a belső (privát) függvények prototípusa (mind statikus)

```
static u8_t L_CAN_u8_Read_Register_f(u8_t address);
static void L_CAN_v_Write_Register_f(u8_t address, u8_t dat);
```

L_*mmm_tt_function_f*

Ahol:
- L prefix (local)
- *mmm* a 3 karakteres, csupa nagybetűs modulazonosító,
- *tt* a visszatérés típusa (v, u8, s8, u8p, s8p, u16, ..., f32, stb.) kisbetűvel
- *function* a függvény neve, tetszőlegesen kis- és nagybetűkkel
- _f postfix (function)

```
/*=========================================[ INTERNAL FUNCTION DEFINITIONS ]=*/
```

Ide kerülnek a belső (privát) függvények definiálása (mind statikus)

```
static u8_t L_CAN_u8_Read_Register_f(u8_t address)
{
}

static void L_CAN_v_Write_Register_f(u8_t address, u8_t dat)
{
}

/*=========================================[ EXPORTED FUNCTION DEFINITIONS ]=*/
```

Ide kerülnek a külső (public) függvények definiálása

```
void ACC_v_Init_f(void)
{
	L_CAN_v_Write_Register_f(0x20,0b01000111);
	L_CAN_v_Write_Register_f(0x20,0b01000111);
}
```

(Formátumot lásd a .h fájl leírásánál!)

```
/*
 *============================================================================*
 * End of File
 *============================================================================*
*/
```

IV.2. A .h fájlok mintája

A program egy fájlt azonosító summary-vel kezdődik, ugyanúgy, ahogy a .c-nél:

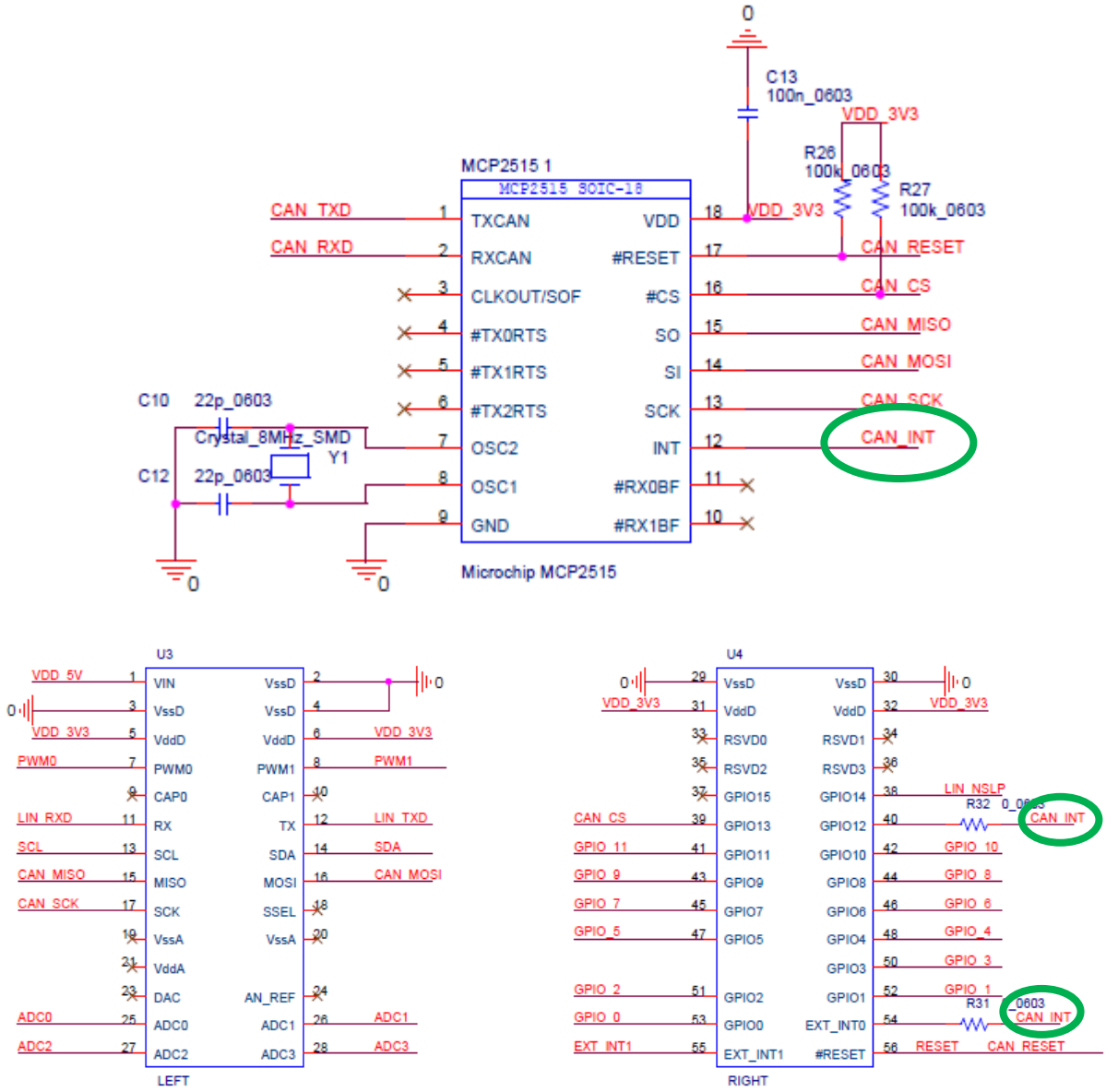
```
/*****************************************************************************
* Title: Rendszertervezés HF                                                 *
* Hardware: CAN/LIN extension board for mitmót system                        *
* Processor: ATMEGA128                                                       *
* Author: Gergő Farkas                                                       *
* Date: 2010-10-17 23:24                                                     *
* Compiler: avr-gcc                                                          *
* ----------------------------                                               *
* Description: Simple example program for using the MCP2515 CAN interface    *
*****************************************************************************/


/*=================================================[ COMPILER SWITCH CHECK ]=*/

Ide kerülnek a feltételes fordításra vonatkozó ellenőrzések

/*=============================================================[ HEADER INIT ]=*/

Ide kerül a többszörös include kivédésére szolgáló ifndef:

#ifndef CAN_H
#define CAN_H

mmm_H

Ahol:
- mmm a 3 karakteres, csupa nagybetűs modulazonosító,
- H postfix (header)


/*=========================================================[ EXTERNAL MACROS ]=*/
/* macro constants and read macros */

Ide kerülnek a külső (public) makrók és define-ok, pl:

#define CAN_CS_DIR_OUTPUT()          GPIO_13_DIR_OUTPUT()
#define CAN_U8_VALTOZO               ((uint8_t)(56))

mmm_tt_variable

Ahol:
- mmm a 3 karakteres, csupa nagybetűs modulazonosító,
- tt a változó típusa (v, u8, s8, u8p, s8p, u16, ..., f32, stb.) csupa nagybetűvel
- variable a változó neve, csupa nagybetűvel

/*=================================================[ EXTERNAL TYPEDEFS ]=*/

Ide kerülnek a külső (public) típusdeklarációk, pl:

typedef struct
{
    unsigned short int       id;
    unsigned char            rtr;
    unsigned char            length;
    unsigned char            data[8];
} CAN_st_message_t;

mmm_tt_type_t

Ahol:
- mmm a 3 karakteres, csupa nagybetűs modulazonosító,
- tt a változó típusa (v, u8, s8, u8p, s8p, u16, ..., f32, stb.) csupa kisbetűvel
- type a deklarált típus neve, tetszőlegesen kis- és nagybetűkkel
- t prefix (type)
```

```
/*=========================================================[ EXTERNAL GLOBALS ]=*/

Ide kerülnek a külső (public) változók extern definiálása

extern uint16_t* CAN_u16p_message2;
extern uint8_t   CAN_u8_counter2;

Kezdőértéket nem kell megadni!

/*=============================================[ EXTPORTED FUNCTION PROTOTYPES ]=*/

Ide kerülnek a külső (public) függvények prototípusa (mind extern)

extern void CAN_v_Init_f(void);
extern void CAN_v_Read_f(void);

mmm_tt_function_f

Ahol:
- mmm a 3 karakteres, csupa nagybetűs modulazonosító,
- tt a visszatérés típusa (v, u8, s8, u8p, s8p, u16, ..., f32, stb.) kisbetűvel
- function a függvény neve, tetszőlegesen kis- és nagybetűkkel
- _f postfix (function)


#endif /* CAN_H */
/*
 *=============================================================================*
 * End of File
 *=============================================================================*
*/
```

V. Szoftver implementáció

V.1. CAN réteg megvalósítása

Az eredeti api-ban a *can_receive_message()* függvény blokkolva várakozik, amíg nem érkezik üzenet. Az első feladat ezen függvény megírása úgy, hogy az üzenet érkezését egy megszakítási szubrutin jelezze.



A kártya kapcsolási rajzából kiolvasható, hogy az MCP2515-ös CAN vezérlő INT lába összeköttetésben van egy soros ellenálláson keresztül a mitmót hüvelysor GPIO12-es és az EXT_INT0 lábával.

MCP2515-ös CAN vezérlő adatlapja alapján a ha a CANINTE (interrupt enable) regisztérenek RX0IE (receive 0 interrupt enable) bitje 1 értékű, akkor egy megszakítást

generál a vezérlő, ha egy üzenet érkezett az RXB0 (receive buffer0) regiszterébe. Ugyanakkor ezzel párhuzamosan a CANINTF (interrupt flag) regiszter RX0IF (receive 0 interrupt flag) bitje 1 értékűvé válik, indikálva így a megszakítás forrását. A megszakítást törölni ezen RX0IF bit 0-ba állításával lehet. Ha egy megszakítás bekövetkezik, akkor a CAN vezérlő az INT lábát logikai 0 szintre állítja, és addig ott is tartja, amíg a mikrovezérlő a megszakítást nem törli az RX0IF bit törlésével.

A feladat megoldásához a következő kódrészleteket hoztam létre az **mcp2515.c** fájlban:
Az Atmega128 mikrovezérlő külső INT0 megszakításának engedélyezése/tiltása:

```
void INT0_IT_ENABLE(void)
{
    EICRA=0;          // The low level of INT0 generates an interrupt request
    EIMSK=(1<<INT0); // INT5: External Interrupt Request 0 Enable
}


void INT0_IT_DISABLE(void)
{
    EIMSK=0;          // External Interrupt Request disable
}
```

Az eredeti *mcp2515_init()* függvényben is szerepel az RX0IE bit 1-be állítása, így az én verzióban is megfelel ez a függvény az inicializálási feladatok ellátására.
A CAN üzenetek fogadását a következő két függvénnyel engedélyezhetjük/tilthatjuk:

```
/********  can_receive_message_ISR_ENABLE ******************/
void can_receive_message_ISR_ENABLE(void)
{
    INT0_IT_ENABLE();
}

/********  can_receive_message_ISR_ENABLE ******************/
void can_receive_message_ISR_DISABLE(void)
{
    INT0_IT_DISABLE();
}
```

Az üzenetek fogadására az eredeti api lehetőséget nyújt szűrők beállítására a
`CAN_v_mcp2515_Set_standard_mask_Rx0_f() ,illetve a`
`CAN_v_mcp2515_Set_standard_filter_RxF0_f() függvények használatával.`

A **platform.h** fájlban létrehoztam a bool típusú változót:

```
typedef unsigned char bool;
#define true    1
#define false   0
```

Az üzenetek fogadásához a főprogramban (jelen esetben **CAN_test.c** fájl) a következő globális változókat kell létrehozni:

```
CAN_message rx_message;
volatile bool New_message_flag=false;
```

Az `rx_message` a beérkezett üzenet tárolásához, a `New_message_flag` pedig azt jelzi, hogy új (eddig fel nem használt) üzenet van a az `rx_message` változóban.

Az üzenet fogadása szintén a főprogramban elhelyezett External Interrupt Request 0 megszakítási szubrutinban történik. A szubrutin az `rx_message` globális változóba menti a beérkezett üzenetet, majd beállítja a `New_message_flag`-et igaz értékűre. Továbbá a CAN kontrollerben törli a megszakítást a `CANINTF` regiszterének `RX0IF` bitjének 0-ba állításával. Mivel a megszakítási szubrutin nem megszakítható, így ezek a műveletek atomi műveletként hajtódnak végre.

```
/********  External Interrupt Request 0 ISR  ******************************/
ISR (SIG_INTERRUPT0)
{

    unsigned char data,i;

    MCP2515_CS_CLEAR();
        SPI_byte(SPI_READ_RX,0);     // Read Buffer Command

        // Standard ID kiolvasása
        SPI_byte(0,&data);
        rx_message.id =  ((unsigned int) data) << 3;
        SPI_byte(0,&data);
        rx_message.id |= ((unsigned int) data) >> 5;

        SPI_byte(0,&data);
        SPI_byte(0,&data);

        // Length
        SPI_byte(0,&data);
        rx_message.length = data&0xf;

        // Adatok kiolvasása
        for (i=0;i<rx_message.length;i++) {
            SPI_byte(0,&data);
            rx_message.data[i] = data;
        }

        // Setting the new message flag
        New_message_flag=true;

    // Clearing the receive interrupt by clearing the RXnIF bit.
    mcp2515_bit_modify(CANINTF, (1<<RX0IF), 0);

}
```

Végül egy példaprogram a CAN üzentek küldésére, illetve fogadására:

```
int main(void)
{
```

```
    CAN_message message;

    dpy_trm_s01__Init();                // A kijelző panel iniciálása
    mcp2515_init();                     // A CAN kommunikáció iniciálása
    can_receive_message_ISR_ENABLE();   // Üzenet fogadás engedélyezve
    sei();

    message.id = 0x0123;
    message.rtr = 0;
    message.length = 2;
    message.data[0] = 0;
    message.data[1] = 0;    // Egy CAN üzenet összeállítása

    while(1)
    {

        message.data[0]++;
        can_send_standard_message(&message);    // CAN üzenetküldés

        _delay_ms(50);

        // Ha van új üzenet
        if(New_message_flag){
            cli();
            New_message_flag=false;
            dpy_trm_s01__7seq_write_number(rx_message.data[2], 0);
            sei();
        }
    }

}
```

V.2. Platform szoftver és a main függvény működése

A fejlesztés során sok olyan függvény és forráskód keletkezik, melyre minden node-nak szüksége van. Ezeket nem érdemes, sőt nem is szabad külön-külön implementálni többször, ugyanis ebből kavarodás és felesleges munka adódik.

A platform szoftver (mint közös források) a következő fájlokból áll:

`dpy_trm_s01.c` és `.h`

`mcp2515.c` és `.h`

`mcu_avr_atmega128_api.c` és `.h`

`platform.h`

`TIMER.c` és `.h`

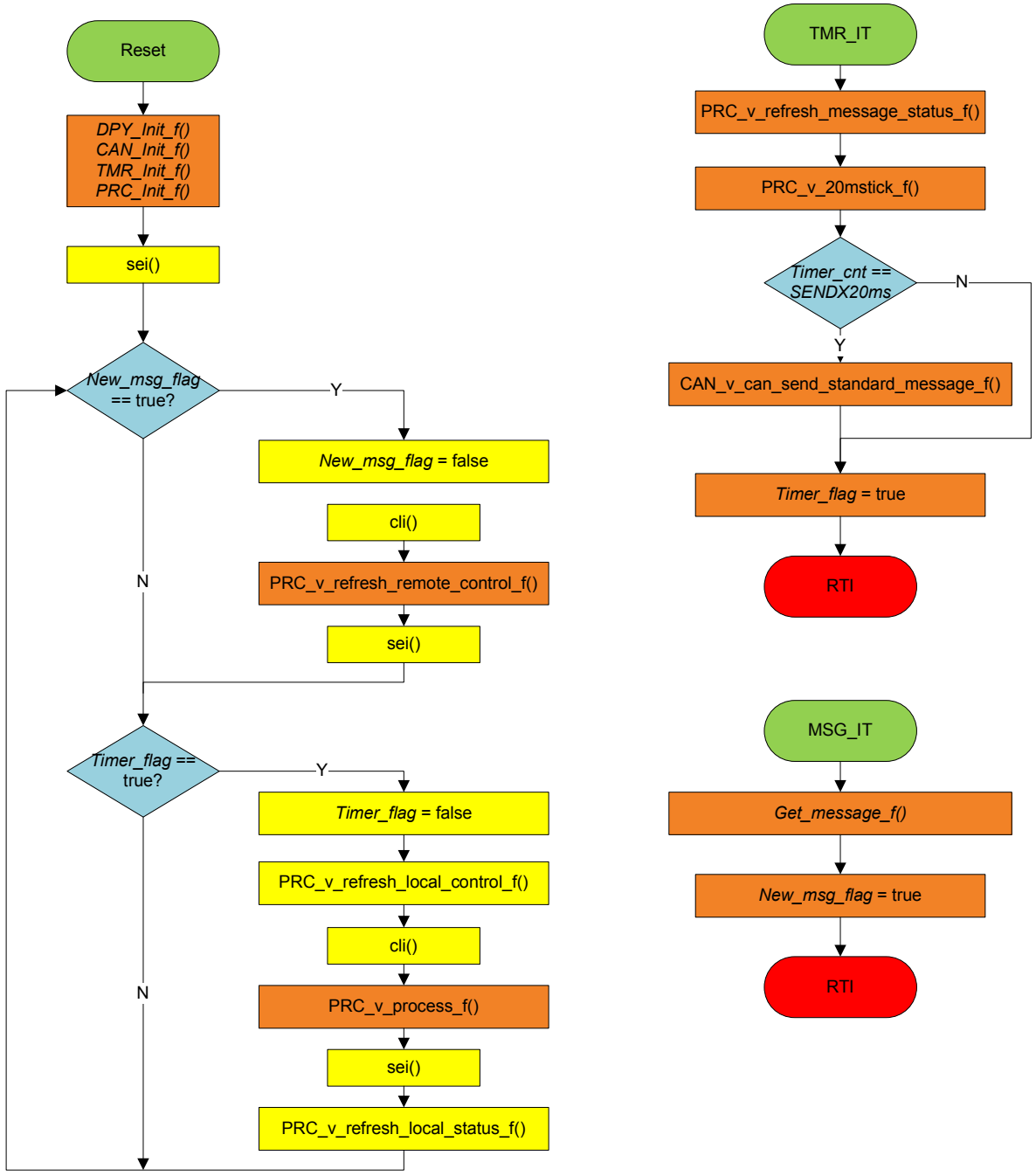
`main.c`

Ezeket egy közös (Common nevű) mappába helyeztük a verziókövető rendszerben, ezáltal ezekben csak úgy volt megengedhető változtatni, ha arról mindenki tud. Ez egyrészről nehézség (mert újabb kommunikációs overhead-et jelent), másrészről könnyítés, mert elegendő annak egyszer belenyúlnia ezekbe a kódokba, aki először találkozik egy problémával, a többieknek már ki lesz javítva.

Az előző pontban leírt CAN-es kártya apijának kiegészítésén kívül felhasználtuk a kijelző (`DPY_TRM_S01`) és a mikrokontroller kártya (`MCU_AVR_S01c`) apiját, valamint a `platform.h` fájlt. A `TIMER.c` és .h fájl már saját fejlesztés, ezáltal nyílik lehetőség az ütemezést megvalósító timer-ek felkonfigurálására, engedélyezésére.

A legfontosabb szempont azonban a `main.c` függvény egységessé tétele volt. Így lehetőség nyílt arra, hogy minden node-on ugyanaz az ütemezési lánc valósuljon meg, ezt egyszer kellett jól végiggondolni, és az egyes node-ok fejlesztőinek már csak a `main.c` által, jól definiált módon meghívott függvények törzsét kellett kitölteni az adott node működésének megfelelően (lényegében ez az applikáció szoftver). Az ilyen node-specifikus függvények a `PRC.c` és `.h` fájlba kerültek.

Az alábbi ábra mutatja a `main.c` működését:



Az ábrán dőlt betű jelzi azokat a változókat vagy függvényeket, amelyek nevénél érdemesebbnek találtunk egy lényeget kifejező, tömörebb név megjelenítését a folyamatábrában. Ezzel szemben az álló betűkkel jelzett függvények ténylegesen így szerepelnek a kódban is.

Különös figyelemmel kellett lennünk az SPI periféria kölcsönös kizárással történő kezelésére. Ugyanis több bájtos kommunikáció zajlik SPI periférián a CAN kontrollerrel, miközben a kijelző panel vezérlése is SPI kommunikációval történik. Nem engedhetjük meg, hogy egy megszakítás más perifériát válasszon ki, miközben a főprogramban éppen ki van választva az

egyik. Ezért alkalmaztunk kritikus szakaszokat, melyeket a folyamatábrán a sötét narancsságra dobozok jelölik.

V.3. Az applikáció szoftver függvényei (PRC.c és .h)

Ebben a részben röviden áttekintjük az applikáció szoftver függvényeit.

`PRC v refresh remote control f()`

A függvény egy üzenet érkezése után fut le. Az érkezett (a saját node működését befolyásoló) üzenet alapján beállítja azokat a belső változókat, ami alapján a feldolgozó függvény a node státuszát meghatározza majd.

`PRC v refresh local control f()`

Az előírt ütemezési gyakorisággal fut le. A node saját érzékelőinek beolvasása (tipikusan gombok), és ezek alapján a belső változók beállítása, ami alapján a feldolgozó függvény a node státuszát meghatározza majd. Eleinte ezt csak üzenet érkezésekor futtattuk, így azonban nem volt lehetséges a node tesztelése önállóan. Pedig elvárható, hogy pl. a bal első ajtó ablakát a saját gombjaival akkor is tudjuk vezérelni, ha a node nincs rajta a CAN hálózaton.

`PRC v process f()`

Ez a függvény végzi el a lokális és távolról érkező parancsok feldolgozását, előírja a beavatkozók állapotát. Az esetleges állapotgépek működtetése is itt zajlik (pl. felfutó él érzékelése).

`PRC v refresh local status f()`

Ebben a függvényben történik meg a beavatkozók tényleges állapotának visszaellenőrzése, azok betöltése a belső változókba. Egy későbbi iteráció során, ha már tényleges hardverek lennének a beavatkozó szervek LED-ek helyett, akkor az áramfelvétel mérése itt történne meg. Ezekhez belső függvényeket implementáltunk, így a továbbfejlesztés csak ezen függvények törzsének módosításával járna.

`PRC v refresh message status f()`

A belső állapotváltozók másolása a küldendő CAN üzenet megfelelő bitjeire. Üzenetküldés előtt hívódik meg. A CAN üzenet felépítésének specifikáció változásakor csak ehhez a függvényhez kell hozzányúlni.

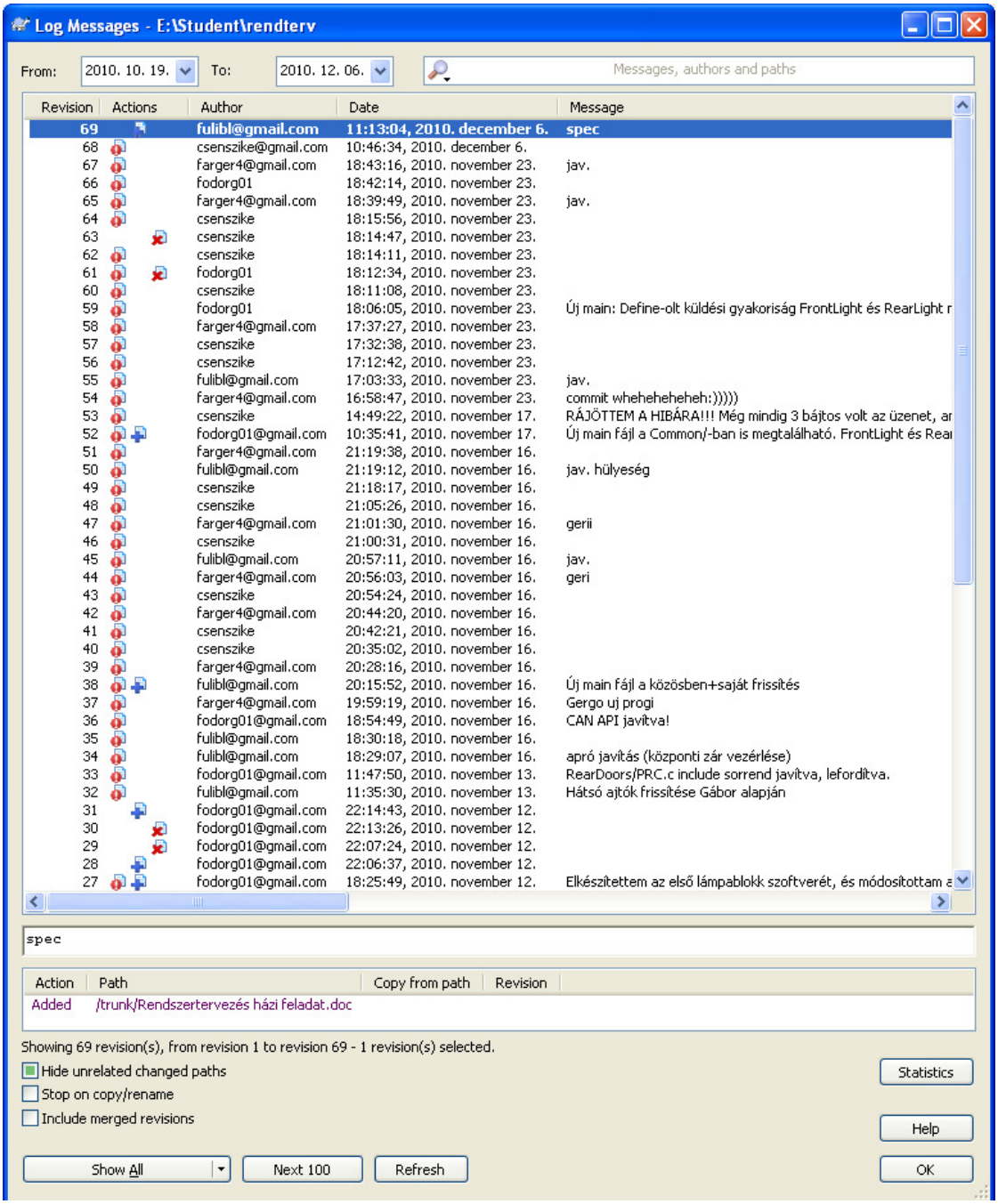
PRC_v_20mstick_f()

Egy általános célú, 20 ms-onként lefutó, interruptból meghívott (tehát megszakíthatatlan) függvény. A legtöbb node esetén üres függvény lett, de például az első lámpablokk ez alapján lépteti az index villogásához szükséges számlálót.

VI. Verziókövetés

A feladat megoldása során az SVN verziókövető-rendszert használtuk. Ehhez a Google Project rendszerében regisztráltunk tárhelyet (repository-t). A projekt címe http://canbs.googlecode.com lett. Ebben a rendszerben a checkouthoz nem szükséges jelszó, ilyen módon nyilvános a tárhely. A tárhely eléréséhez a Windows-os TortoiseSVN klienst használtuk.

A tárhely elérhető a https://canbs.googlecode.com/svn címen.



VII. Tesztelés

A node-ok tesztelése a következőképp zajlott. Kezdetben mindenki a saját node-ját tesztelte, ez a szakasz volt a modultesztelés. Mikor megfelelően működtek önálló módban, elkezdtük az integrációs tesztelést. Ennek a fázisait a következő táblázat tartalmazza.

Szakasz	1. Tesztelési csoport	2. Tesztelési csoport
1.	Első lámpablokk és kormány	Bal első ajtó és jobb első ajtó
2.	Hátsó lámpablokk és kormány	Bal első ajtó és hátsó ajtók
3.	Első lámpablokk, hátsó lámpablokk és kormány	Bal első ajtó, jobb első ajtó és hátsó ajtók
4.	Összevont rendszerteszt	

VIII. Összefoglalás, értékelés

A félév során egy autó CAN hálózat alapú body rendszerének szoftverfejlesztését végeztük el a megadott és általunk kibővített specifikáció alapján.

A fejlesztés során megismerkedtünk egy tipikus kis/közepes nehézségű és méretű feladat dekomponálásához szükséges eszközökkel (verziókövetés, code style guide, stb.).

A megismert eszközök nagyban hozzásegítettek a feladat sikeres, határidőnek megfelelő időre történő elkészítéséhez.

A feladat elkészült, az összevont rendszerteszten is megfelelt.